



6G Vertical trials for Sustainability

Deliverable D4.1

Clusters' Enablers-Initial



Co-funded by
the European Union



6G-VERSUS project has received funding from the [Smart Networks and Services Joint Undertaking \(SNS JU\)](#) under the European Union's [Horizon Europe research and innovation programme](#) under Grant Agreement No 101192633.

Date of delivery:	08/05/2026	Version:	1.0
Project reference:	101192633 — 6G-VERSUS	Call:	HORIZON-JU-SNS-2024- STREAM-D-01-01
Start date of project:	01/01/2025	Duration:	36 months

Document properties:

Document Number:	D4.1
Document Title:	Clusters' Enablers-Initial
Editor(s):	NCSRD - Main editor(s) of the document
Authors:	Antti Pauanne (UOULU); Sanna Tuomela (UOULU); Mahdi Salmani (UOULU); Mina Tavangar (UOULU); Spyros Athanasiadis (ICCS); Thanasis Douklias (ICCS); Dimitrios Fragkos (ICCS); Aimilios Leftheriotis (ICCS); Georgios Lentaris (ICCS); Xenofon Vasilakos (ICCS); Spyridon Georgoulas (NCSRD); George Makropoulos (NCSRD); Panagiotis Pavlidis (NCSRD); Harilaos Koumaras (NCSRD); Francisco Luque Schempp (UMA); Delia Rico (UMA); Pablo Herrera (UMA); Doan Khuc (TSGF); Véronique Capdevielle (TSGF); Sérgio Figueiredo (IPN); Rafael Gonçalves (IPN); João Gonçalves (IPN); Rúben Leal (IPN); Marco Silva (IPN); Evgenia Petkova (A1); Nuria Oyaga (NOKIA SPAIN); Marta Orduna (NOKIA SPAIN); Davide Montagno Bozzone (HPE); Nicola di Pietro (HPE); Irina Ciornei (UCY); Kyriakos Karlettis (UCY); Panagiotis Chrysanthou (UCY); Dimitris Salmatanis (OTE); Foteini Setaki (OTE); Sofiane Messaoudi (EUR); Bernardo Duarte (ALB); Gonçalo Machado (ALB); Jorge Proença (ONE); Luis Cordeiro (ONE); Duarte Feveireiro (ONE); José Guimarães (ONE); João Tomás (ONE); José Xavier (ONE); Iraklis Spyrou (INF); Vaios Koumaras (INF); Evangelos Anagnostopoulos (INF); Georgios Koumaras (INF); Eugenia Vergi (INF); Panagiotis Koumaras (INF); Valentin Velez (SCBG); Francois Hamon (GREEN); Daniel Corujo (ITAV); David Santos (ITAV); Rui Silva (ITAV); Tiago Barros (ITAV); Jorge Carapinha (ITAV); Afonso Castanheta (ITAV); Tomás Hilário (ITAV); Bernardo Lázaro (ITAV); Diogo Duarte (ITAV); Paulo Cunha (ITAV); Rafael Ferreira (ITAV); Francisco Gonçalves (ITAV); Raul Barranquero Simon (TID); Pablo Iglesias Sanuy (TID); Ángel Canete (CET)
Contractual Date of Delivery:	30/04/2026
Dissemination level:	PU ¹
Status:	Submitted
Version:	1.0
Nature of Deliverable:	DEM

¹ SEN = Sensitive, only members of the consortium (including the Commission Services). Limited under the conditions of the Grant Agreement

PU = Public

Revision History

Revision	Date	Issued by	Description
0.1	22/01/26	NCSRD	Initial Table of Contents (ToC) and deliverable structure issued by NCSRD as deliverable leaders.
0.2	03/02/26	6G-VERSUS Consortium	Start of contribution collection process from all partners.
0.3	16/03/26	NCSRD, OTE, ICCS, EUR, INF, NOKIA, UMA	Updated ToC and redesigned document structure following the first internal review and partner comments, in order to better align with the technical and reporting needs of the consortium.
0.4	02/04/26	6G-VERSUS Consortium	Consolidated version including technical inputs and contributions from all partners.
0.5	28/04/26	NCSRD, SCBG, OTE, ICCS, NOKIA, UOULU	Second internal review completed, including comments, requested additions, corrections, and restructuring of specific sections.
0.6	06/05/26	NCSRD	Final editing, formatting, consistency checks, and preparation of the submission-ready version.
1.0	08/05/26	6G-VERSUS Consortium	Final version submitted.

Abstract

This deliverable presents the initial development status of the application enablers within the 6G-VERSUS project, focusing on the implementation of Network Applications (N-Apps), Vertical Applications (V-Apps), and AI Applications (AI-Apps) across the participating clusters. It builds upon the architectural definitions and use case requirements introduced in earlier project deliverables and reports on how these specifications are translated into concrete software components. The document details the design, development, and deployment of the application triplet concept, where V-Apps, N-Apps, and AI-Apps interact to deliver end-to-end 6G services. For each cluster, it outlines the implemented applications, their functionalities, interfaces, and integration with 5G/6G network infrastructures, including the use of standardized frameworks such as CAPIF and exposure of network capabilities through APIs. Furthermore, the deliverable describes the common development approach, integration mechanisms, and early validation results, highlighting the progress achieved during the initial phase of the project. It also provides an overview of the ongoing developments planned for the next phase, setting the foundation for large-scale trials and final system validation to be reported in subsequent deliverables.

Keywords

6G; N-Apps; V-Apps; AI-Apps; Application Triplets; CAPIF; 5G Core; AI/ML; Integration; Clusters

Disclaimer

Funded by the European Union. The views and opinions expressed are however those of the author(s) only and do not necessarily reflect the views of 6G-VERSUS Consortium nor those of

the European Union or Horizon Europe SNS JU. Neither the European Union nor the granting authority can be held responsible for them.

Executive Summary

This deliverable reports the intermediate development status of the N-/V-/AI-App ecosystem, targeting project partners, developers, integrators, and technical stakeholders involved in the specification, implementation, integration, and validation of the application components. Also, the completion of the current deliverable marks the fulfilment of Milestone 9, corresponding to the availability of the initial software components and their integration into the common development environment.

Building upon the initial architectural foundations of D2.1 and the experimentation framework introduced with D3.1, the present document focuses on the realisation and integration of application components into operational 6G Application triplets, namely the combination of N-App, V-App, and AI-App. It provides a consolidated and structured view of the progress achieved across all clusters, including software development, deployment status, and initial validation results, together with references to available implementations in the project GitLab repository, where permitted. Therefore, within the above context, the current deliverable follows and considers the requirements already defined in D2.1, which remain the reference requirements for the developed software artefacts and components described in this document.

Across the six clusters, a diverse and evolving portfolio of N-Apps, V-Apps, and AI-Apps has been designed and developed, addressing key domains such as network QoS optimisation, energy efficiency, location awareness, environmental monitoring, remote operation, and intelligent control. These components are progressively integrated into end-to-end triplets, enabling early-stage validation of the proposed architecture and demonstrating its capability to support complex, distributed, and data-driven 6G services across heterogeneous environments. The reported results focus mainly on applications used during Phase 1 trial events (i.e., for trials until May 2026), with references to future application component evolvments (preliminary design and/or implementation results) for Phase 2 trials (i.e., trials until May 2027). These results confirm the feasibility and relevance of the triplet-based approach, while clearly evidencing the added value of tightly coupling network programmability, AI-driven intelligence, and vertical service logic. At the same time, they underline the need for further integration, optimisation, and large-scale validation, which will be pursued in the next phase of the project through cluster-level trials and systematic validation and evaluation activities, as documented in D5.2.

Last, the final consolidation and full-scale validation of the developed components, together with their comprehensive impact assessment, will be presented in D4.2 at the end of the project.

Table of Contents

Executive Summary	4
1. Introduction.....	16
1.1 Project overview	16
1.2 Scope and Intended Audience.....	16
1.3 Relation to milestones and other deliverables	16
1.4 Structure of the Document.....	17
2 Scope and Objectives of N/V/AI-Apps.....	17
2.1 Transformation of Use-Cases to 6G Applications.....	17
2.2 Definition of N/V/AI-Apps in 6G-VERSUS	18
2.3 Interfaces between N/V/AI-Apps	19
3 Development of N-Apps.....	21
3.1 Common Development Process and N-App Template.....	21
3.2 Bulgarian Cluster	24
3.2.1 N-App for Network QoS Monitoring and Improvement	24
3.2.2 N-Apps under Development for Phase 2	26
3.3 Finnish Cluster.....	27
3.3.1 N-App – Control and Data Transfer Function	27
3.3.2 N-Apps under Development for Phase 2	29
3.4 Greek Cluster	29
3.4.1 N-App for Location Awareness	30
3.4.2 N-Apps under Development for Phase 2	33
3.5 French Cluster.....	35
3.5.1 N-App NEF+CAPIF	35
3.5.2 N-App RAN/AI-App interface.....	38
3.5.3 N-Apps under Development for Phase 2	39
3.6 Portuguese Cluster	39
3.6.1 QoS N-App	40
3.6.2 Network Operations N-App	43
3.6.3 N-Apps under Development for Phase 2	46
3.7 Spanish Cluster	47
3.7.1 Decision Support System for water quality N-App	48
3.7.2 Multi-connectivity protocol N-App	49

3.7.3	Immersive driving of an AGV with autonomous missions N-App.....	52
3.7.4	Automatic Surveillance & Detection in Farming Operation N-App	55
3.7.5	DSS for reliable water treatment N-App.....	57
3.7.6	N-Apps under Development for Phase 2	59
4	Development of V-Apps.....	59
4.1	Bulgarian Cluster	59
4.1.1	V-App Central -WAM	60
4.1.2	V-App Regional – monitoring DERs	62
4.1.3	V-App Central – WAC.....	63
4.1.4	V-App Regional Controller – RC	65
4.1.5	V-Apps under Development for Phase 2	67
4.2	Finnish Cluster.....	67
4.2.1	Common functionality of V-App Central	67
4.2.2	V-Apps under Development for Phase 2	74
4.3	Greek Cluster	74
4.3.1	INUS Asset V-App	75
4.3.2	EEaaS Dashboard V-App.....	77
4.3.3	Wildfire Detection Control Room V-App.....	78
4.3.4	V-Apps under Development for Phase 2	81
4.4	French Cluster.....	82
4.4.1	V-App Environmental Indicators Dashboard UC2	83
4.4.2	V-App Abnormal watering detection.....	84
4.4.3	V-App B5G connector	86
4.4.4	V-App soil and level sensor	87
4.4.5	V-Apps under Development for Phase 2	89
4.5	Portuguese Cluster	89
4.5.1	Port Environmental Monitoring V-App	90
4.5.2	Mobitrust V-App.....	93
4.5.3	Port Safety V-App.....	94
4.5.4	Infrastructure Energy Tracking V-App.....	96
4.5.5	V-Apps under Development for Phase 2	99
4.6	Spanish Cluster	99
4.6.1	DSS for water quality V-App.....	100

4.6.2	Unmanned Vehicle control V-App	101
4.6.3	Immersive driving of an AGV with autonomous missions V-App.....	103
4.6.4	Automatic Surveillance & Detection in Farming Operation V-App	106
4.6.5	DSS for reliable water treatment V-App.....	107
4.6.6	V-Apps under Development for Phase 2	108
5	Development of AI-Apps	109
5.1	Bulgarian Cluster	109
5.1.1	AI-App Grid Frequency Trajectory Prediction for the Bulgarian Pilot	110
5.1.2	AI-Apps under Development for Phase 2	113
5.2	Finnish Cluster.....	113
5.2.1	Arctic Energy Controller AI-App	114
5.2.2	Arctic Traffic Controller AI-App.....	117
5.2.3	AI-Apps under Development for Phase 2	118
5.3	Greek Cluster	119
5.3.1	AI-App Detection & Tracking	120
5.3.2	AI-App for Early Wildfire Monitoring	123
5.3.3	AI-App for Location Awareness.....	129
5.3.4	9091AI-Apps under Development for Phase 2	133
5.4	French Cluster.....	134
5.4.1	AI-App RAN optimization for battery lifetime extension	134
5.4.2	AI-Apps under Development for Phase 2	137
5.5	Portuguese Cluster	138
5.5.1	Mobitrust AI-App	138
5.5.2	Simple Object Detection System AI-App	139
5.5.3	HazMat Vehicles Detector AI-App.....	142
5.5.4	AI-Apps under Development for Phase 2	146
5.6	Spanish Cluster	147
5.6.1	DSS for water quality AI-App	147
5.6.2	Immersive driving of an AGV with autonomous missions AI-App	149
5.6.3	DSS for reliable water treatment AI-App	151
5.6.4	AI-Apps under Development for Phase 2	154
6	Triplet Integration	155
6.1	Bulgarian Cluster	155

6.2	Finnish Cluster	159
6.3	Greek Cluster	164
6.4	French Cluster	169
6.5	Portuguese Cluster	176
6.5.1	Scenario 1 Triplet	176
6.5.2	Scenario 2 Triplet	179
6.5.3	Scenario 3 Triplet	180
6.6	Spanish Cluster	184
6.6.1	Activity 1: DSS for water quality	184
6.6.2	Activity 2: Unmanned Vehicle control V-App and Multi-connectivity protocol N-App	186
6.6.3	Activity 3: Immersive driving of an AGV with autonomous missions	187
6.6.4	Activity 4: Automatic Surveillance & Detection in Farming Operation ..	190
6.6.5	Activity 5: DSS for reliable water treatment	191
7	Conclusion	193
8	Bibliography	193

List of Tables

Table 1 CAPIF Invoker and Provider Deployment Scenarios	20
Table 2 AsSessionwith QoS API Methods	25
Table 3 Monitoring Event API Methods.....	25
Table 4 Implemented and under development N-App for the Finnish cluster.....	27
Table 5 Implemented and under development N-Apps for the Greek cluster.....	30
Table 6 MonitoringEvent API methods	31
Table 7 Implemented and under development N-Apps of the French cluster	35
Table 8 API Methods of the N-App	36
Table 9 Implemented and under development N-Apps in the Portuguese cluster	39
Table 10 QoS N-App API Methods	41
Table 11 N-App network operations API methods.....	45
Table 12 Set of API endpoints and methods used by the Network Operations N-App	45
Table 13 Implemented and under development N-Apps in the Spanish cluster	47
Table 14 Set of API endpoints and methods used by DSS for water quality N-App.....	48
Table 15 Set of API endpoints and methods used by Multi-connectivity protocol N-App	50
Table 16 Custom 5G API methods	53
Table 17 Set of API endpoints and methods used by Automatic Surveillance & Detection in Farming Operation N-App.....	55
Table 18 Set of API endpoints and methods used by DSS for reliable water treatment N-App.....	57
Table 19 Implemented and under development V-Apps in the Bulgarian cluster	59
Table 20 current V-App structure, ownership, and deployment/testing locations.....	60
Table 21 Implemented and under development V-Apps in the Finnish cluster.....	67
Table 22 Implemented and under development V-Apps in the Greek cluster	74
Table 23 Implemented and under development V-Apps in the French cluster.....	82
Table 24 Implemented and under development V-Apps in the Portuguese cluster	89
Table 25 Implemented and under development V-App in the Spanish cluster.....	99
Table 26 Implemented and under development AI-Apps for the Bulgarian cluster	109
Table 27 Implemented and under development AI-Apps in the Finnish cluster	113
Table 28 Implemented and under development AI-Apps in the Greek cluster.....	119
Table 29 Implemented and under development AI-Apps in the French cluster	134
Table 30 Implemented and under development AI-Apps in the Portuguese cluster.....	138
Table 31 Model Performance Metrics.....	143
Table 32 Implemented and under development AI-Apps in the Spanish cluster.....	147
Table 33 End-points exposed by the triplet applications	162
Table 34 End points exposed by the N-App for location.....	166
Table 35 End points exposed by AI-App	166

Table 36 AI-App MQTT topics	166
Table 37 End points of triplet integration in French cluster	171
Table 38 End points for triplet integration in the Portuguese cluster.....	177
Table 39 APIs utilized for Triplet	181
Table 40 APIs utilized for Triplet	187
Table 41 N-App - 5G Core (SMF API).....	188
Table 42 V-App - N-App (WebSocket API).....	189

List of Figures

Figure 1 6G application as a V-App / N-App / AI-App triplet depicted.....	18
Figure 2 Interaction Architecture of V/AI/N-Apps Triplets and Testbed Components.....	20
Figure 3 N-App template sequence diagram.....	23
Figure 4 N-App for QoS Monitoring and improvement	24
Figure 5 The architecture of N-App.	28
Figure 6 N-App software architecture.....	28
Figure 7: Results of the N-App API tests.....	29
Figure 8 NEF Monitoring Event API container logs	32
Figure 9 Invoker's application Successful container logs.....	32
Figure 10 Open5gs Core alongside N-App for location awareness	33
Figure 11 CAPIF deployment instance	33
Figure 12 N-App Coverage Extension.....	34
Figure 13 N-App NEF+CAPIF Architecture	36
Figure 14 N-App NEF+CAPIF test of the French testbed on real RedCap UE and 5G SA.....	38
Figure 15 N-App 3 (RAN/AI-App Interface) Architecture for the French Cluster	38
Figure 16 N-App "qos_flow_get" response	39
Figure 17 N-App "ue_get" response	39
Figure 18 QoS N-App architecture	41
Figure 19 QoS N-App basic workflow	41
Figure 20. Logging information of the interaction between the QoS N-App and the NEF.....	43
Figure 21 Network Operations N-App architectural view	44
Figure 22 Network Operations N-App basic workflow	44
Figure 23 Overview of N-App involvement in the Portuguese Cluster	47
Figure 24 Multi-connectivity protocol N-App architecture	50
Figure 25 Multi-connectivity protocol N-App showcased in Trial 1.....	52
Figure 26 N-App and NaC logs during IP request.....	55
Figure 27 Configurable delay is used to ensure that a signal remains beyond the alarm point	61
Figure 28 Snapshot of the web UI of the V-App Central – WAM	62
Figure 29 Snapshot of the web UI of the V-App Regional.....	63

Figure 30 WAC Web Dashboard	65
Figure 31 V-App Regional Controller – RC dashboard.....	66
Figure 32 WEMS Dashboard.....	69
Figure 33 Snapshot from the User Login screen.....	69
Figure 34 Snapshot from the User Registration screen.....	70
Figure 35 User Profile Editing and user activity.....	71
Figure 36 Snapshot of User deletion screen.....	71
Figure 37 AI Result view on the main dashboard.....	72
Figure 38 FMI Dashboard view	73
Figure 39 Energy Management Dashboard view.....	74
Figure 40 INUS Asset V-App architecture	76
Figure 41 INUS V-App main dashboard with interactive map and live video streams validated during 1st trial	77
Figure 42 EEaaS V-App Energy Monitoring Dashboard.....	78
Figure 43 The architecture of the Wildfire Detection Control Room V-App.....	79
Figure 44 Dashboard view (Camera status, Detections statistics)	81
Figure 45 Map View (Info about the site, fire zones from NASA)	81
Figure 46 Pop up alert showing a detection came along with the detection video	81
Figure 47 :Architecture of V-App Environmental Indicators	83
Figure 48 : V-App Environmental Indicators Dashboard.....	84
Figure 49 : V-App Abnormal Watering Detection.....	85
Figure 50 V-App Environmental Indicators Dashboard	86
Figure 51 Architecture of B5G Connector V-App.....	87
Figure 52 Architecture of V-App Soil and Level Sensor.....	88
Figure 53 : Soil moisture sensor measurement reported in V-App.....	89
Figure 54 V-App Client Sequence Diagram	92
Figure 55 V-App Server Sequence Diagram.....	92
Figure 56 V-App web interface.....	93
Figure 57 Mobitrust V-App dashboard in the Portuguese UC trial	94
Figure 58 Port Safety V-App/AI-App integration.....	95
Figure 59 Port Safety V-App dashboard	96
Figure 60 AI-App/V-App messages.....	96
Figure 61 Snapshot of the Infrastructure Energy Tracking V-App dashboard	98
Figure 62 Snapshot of the Infrastructure Energy Tracking V-App dashboard for Solar panels.....	99
Figure 63 Sample alert email for plant process monitoring. The message includes the type of alert detected by the model and an image to help the plant operator make an informed decision on how to resolve the issue	101
Figure 64. Unmanned Vehicle control V-App architecture	102
Figure 65. Robotic arm integration with the drone	103

Figure 66. Robotic arm remote control via drone remote controller	103
Figure 67. Robotic arm remote control via	103
Figure 68 Immersive remote driving V-App Architecture.....	104
Figure 69 V-App logs during IP request	105
Figure 70 RTT measurements in V-App.....	106
Figure 71 Location details in V-App.....	106
Figure 72 Mock-up of the DSS for reliable water treatment V-App	108
Figure 73 Frequency response of the simulated system under different generation-loss scenarios.....	113
Figure 74 Example of the end-to-end AI-App development workflow, from synthetic dataset generation and preprocessing to model training, ML-Ops tracking, and deployment-oriented inference validation	113
Figure 75 Architecture of the Arctic Energy Controller AI-App.....	115
Figure 76 Actual vs Predicted Solar Energy	115
Figure 77 Actual vs Predicted Wind Energy.....	116
Figure 78 AI-App Detection and Tracking architecture	121
Figure 79 AI-Apps detection logs	123
Figure 80 Real-time human detection and tracking from UAV and Robotic dog video streams	123
Figure 81 YOLOv8 smoke detection model training convergence curves, showing training/validation losses and precision, recall, and mAP evolution over 80 epochs	128
Figure 82 F1-confidence curve used for operational threshold selection of the smoke detection model during deployment.....	128
Figure 83 Containerized deployment validation of the AI-App, showing healthy Docker runtime state, mounted NAS volumes, assigned GPU device, and NVIDIA A10 utilization metrics	129
Figure 84 AI-App runtime validation logs showing CUDA initialization, MQTT broker connectivity, RTSP stream acquisition, and continuous real-time frame inference.....	129
Figure 85 MQTT message example that AI-app sends.....	129
Figure 86 AI-App for Location Awareness-AI-Model Architecture	130
Figure 87 Confusion Matrix of the location awareness AI-App.....	132
Figure 88 Training and Validation Loss Curves of the location awareness AI-App	132
Figure 89 Training and Validation Loss Curves of the location awareness AI-App	133
5.3.4 9091AI-Apps under Development for Phase 2.....	133
Figure 92 AI-App RAN optimization for battery lifetime extension.....	135
Figure 93 Evolution of the cumulated reward as a function of the steps	136
Figure 94 Cumulative Distribution Function on the reward/objective function	137
Figure 95 - Mobitrust AI-App in the Portuguese UC trial	139
Figure 96 Technological architecture of Simple Object detection and Hazmat truck detection AI-Apps.....	140
Figure 97 Initial truck detection	142
98.....	143
Figure 99 Normalized confusion matrix	144

Figure 100 Training curves: loss functions (box, classification, DFL) and performance metrics (precision, recall, mAP) over 107 epochs	144
Figure 101 High-resolution identification of license plate.....	146
Figure 102 Detection of ADR orange plates.....	146
Figure 103 Architecture of the DSS for water quality	148
Figure 104 Person detection with YOLO.....	151
Figure 105 Architecture of the DSS for reliable water treatment	152
Figure 106 A. Components of the equipment installed in the laboratory. B. Equipment installed in the water treatment plant.....	154
Figure 107 Correlation between E. coli concentration in the IDEXX Colilert method and enzymatic activity recorded on the ColiMinder instrument.	154
Figure 108 Bulgarian-cluster triplet integration architecture in the UCY pre-piloting environment, showing the OPAL-RT power-system testbed, UCY 5G-OAI transport and exposure layer, V-App Central WAC, V-App Regional RC, AI-App, ML-Ops services, and the N-App / netw	156
Figure 109. Inter-communication of main applications	160
Figure 110. Handling of measurement data	161
Figure 111. Sequence of AI decision handling	162
Figure 112 Sequence diagram for the interaction of the components in the Greek use case.....	165
Figure 113 Inter-communication of main components in the Athens B5G testbed.....	166
Figure 114 N-App location (invoker app) logs	167
Figure 115 Inference Results.....	168
Figure 116 AI-App Detection and Tracking runtime logs	169
Figure 117 French 5G Extreme Edge-to-Cloud Testbed Integrating Sensors, V/AI/N-Apps, MLOps, and CAPIF.....	171
Figure 118 Four RedCap UEs integrated into the French testbed using the OAI 5G Standalone (SA) network	173
Figure 119 The V-App Soil Sensor receives payloads and forwards them to the cloud.....	174
Figure 120 2-step RACH OAI UE successfully attached over the air (OTA) using two USRPs (one for the UE and one for the gNB), with an established PDU session and assigned IP address, in the OAI 5G Standalone (SA) network	175
Figure 121 V-App Environmental Indicators Dashboard receiving payloads sent by soil sensors integrated with RedCap and 2-step RACH UEs	175
Figure 122 N-App NEF + CAPIF integration and control-plane event subscription logs in the French testbed	176
Figure 123 Simplified Scenario 1 sequence diagram.....	177
Figure 124. Manual change of a device profile.....	179
Figure 125. Logging information of the Port Environmental V-App (top), and the QoS N-App (bottom)...	179
Figure 126 Scenario 2 typical sequence diagram.....	180
Figure 127 Scenario 3 triplet integration sequence diagram.....	181
Figure 128 - Snapshot of the Infrastructure Energy Tracking V-App dashboard for EV Chargers 1.....	183
Figure 129 – Snapshot of the Infrastructure Energy Tracking V-app dashboard for EV Chargers 2.....	183

Figure 130 - Snapshot of the Infrastructure Energy Tracking V-App dashboard for Solar panels 1.....	184
Figure 131 - Snapshot of the Infrastructure Energy Tracking V-App dashboard for Solar panels 2.....	184
Figure 132 DSS for water quality sequence diagram.....	185
Figure 133. Unmanned Vehicle control V-App and the Multi-connectivity protocol N-App sequence diagram	186
Figure 134 The Triplet integration in the Spanish use case	188
Figure 135 DSS for reliable water treatment sequence diagram.....	192

Acronyms and abbreviations

Term	Description
3GPP	3rd Generation Partnership Project
5G	Fifth Generation
5GC	5G Core
5QI	5G QoS Identifier
6G	Sixth Generation
AEF	API Exposing Function
AF	Application Function
AI	Artificial Intelligence
AI-App	AI Application
API	Application Programming Interface
CAPIF	Common API Framework
CN	Core Network
CNFs	Containerized Network Functions
DER	Distributed Energy Resources
DRX	Discontinuous Reception
DSS	Decision Support System
E2E	End-to-End
EaaS	Energy as a Service
EEaaS	Energy Efficiency as a Service
EFN	Energy Footprint Notification
ETSI	European Telecommunications Standards Institute
FMI	Finnish Meteorological Institute
GNSS	Global Navigation Satellite System

GUI	Graphical User Interface
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IMSI	International Mobile Subscriber Identity
IoT	Internet of Things
JSON	JavaScript Object Notation
JWT	JSON Web Token
MaaS	Model-as-a-Service
ML	Machine Learning
MLOps	Machine Learning Operations
MQTT	Message Queuing Telemetry Transport
MSISDN	Mobile Station International Subscriber Directory Number
N-App	Network Application
NBI	Northbound Interface
NEF	3rd Generation Partnership Project; Network Exposure Function
NFs	Network Functions
NWDAF	Network Data Analytics Function
OBU	On-Board Unit
OAI	OpenAirInterface
PDU	Protocol Data Unit
PPDR	Public Protection and Disaster Relief
QoS	Quality of Service
RAN	Radio Access Network
REST	Representational State Transfer
RTT	Round-Trip Time
SA	Standalone
SBA	Service-Based Architecture
SBI	Southbound Interface
SDK	Software Development Kit
SNS JU	Smart Networks and Services Joint Undertaking
TCP	Transmission Control Protocol
TS	Technical Specification
UART	Universal Asynchronous Receiver-Transmitter
UC	Use Case
UE	User Equipment
UI	User Interface

UAV	Unmanned Aerial Vehicle
V-App	Vertical Application
VAE	Vertical Application Enabler
VM	Virtual Machine
XR	Extended Reality
xApp	Application in RAN Intelligent Controller architecture
YAML	YAML Ain't Markup Language
YOLO	You Only Look Once (AI Model)

1. Introduction

1.1 Project overview

6G-VERSUS is a collaborative research project under the HORIZON-JU-SNS-2025 Work Programme. It focuses on advancing sustainable and inclusive 6G networks by integrating results from previous research into a shared experimental infrastructure. The project currently supports both the validation of emerging technologies and the evaluation of relevant use cases across six 6G testbeds in Europe. The work explores key technical directions such as AI-native orchestration, modular and open architectures, and energy-aware network functions. Emphasis is placed on the evolution of vertical applications and the role of machine learning in enabling intelligent and adaptive network behaviour. Pilots across the project's testbeds target real-world challenges in sectors like renewable energy, emergency response, and water and waste management. These trials also introduce a key innovation: the 6G Application triplet, which combines a Vertical App (V-App), a Network App (N-App), and an AI-assisted App (AI-App). Together, they deliver distributed, AI-enhanced services and represent a shift in how next-generation network services are structured and deployed. The project follows a three-step methodology: transforming use cases into 6G Applications, testing and validating them on the enhanced platforms, and assessing their impact from sustainability (environmental, societal, and economic) perspectives. Alongside the technical work, 6G-VERSUS engages with stakeholders from research, industry, and policy to ensure the outcomes are aligned with broader sustainability and innovation goals.

1.2 Scope and Intended Audience

This deliverable reports on the intermediate development status of Network Applications (N-Apps), Vertical Applications (V-Apps), and AI Applications (AI-Apps), targeting project partners across the different use cases and pilots, together with developers, integrators, and other technical stakeholders involved in their specification, development, integration, and validation.

1.3 Relation to milestones and other deliverables

This deliverable builds upon the initial specifications and architectural definitions provided in Deliverable D2.1 (Platform Architecture and Use Case Requirements – Initial) and the experimentation environment described in Deliverable D3.1 (6G-VERSUS Experimentation Environment – Initial). Within this deliverable, the focus is on reporting the development of the Network Applications (N-Apps), Vertical Applications (V-Apps), and AI Applications (AI-Apps), including their integration into application triplets. Furthermore, the deliverable provides references to the corresponding software implementations in the project GitLab repository, where available and provided they are not subject to company confidentiality or access restrictions, together with deployment, demonstration, and validation results showcasing the developed components and their current maturity. In this context, it is clarified that this deliverable does not introduce new or updated requirements. The requirements considered for the design, development, and implementation of the WP4 components are those already defined and reported in WP2 Deliverable D2.1, which are used as the reference requirements throughout this document. The completion of this deliverable also marks the fulfilment of

Milestone 9, corresponding to the availability of the initial version of the software components and the upload of the initial prototypes to the project's common software development repository ([GitLab](#)). The developed components reported here will be integrated and assessed in the project's cluster-level trials, with the corresponding validation and evaluation activities documented in Deliverable D5.2 (6G Trials Validation and Evaluation – Initial). The final version and complete reporting of these developments will be presented in Deliverable D4.2 at the end of the project.

1.4 Structure of the Document

Following this introductory section, Section 2 presents the overall scope, objectives, and architectural context of the N/V/AI-App ecosystem within 6G-VERSUS, including the transformation of use cases into 6G application triplets and the definition of the interfaces and requirements governing their interaction. Sections 3, 4, and 5 report the development status of the Network Applications (N-Apps), Vertical Applications (V-Apps), and AI Applications (AI-Apps), respectively, structured per application category and cluster, including implementation progress, deployment status, and current results. Section 6 provides an overview of the triplet-level integration activities, highlighting the integration of the developed application components into end-to-end application triplets. Finally, Section 7 concludes the document and outlines the next steps towards the final implementation and validation phases of the project.

2 Scope and Objectives of N/V/AI-Apps

This section introduces the conceptual and architectural foundations of the 6G-VERSUS application ecosystem, focusing on the transformation of vertical use cases into distributed 6G application triplets composed of Network Applications (N-Apps), Vertical Applications (V-Apps) and AI Applications (AI-Apps). It explains the role of each application component, their interaction with the underlying B5G infrastructure, and the interfaces that enable interoperability, programmability, and AI-assisted operation across the project's testbeds. Furthermore, this section establishes the common application model adopted across all clusters, which serves as the basis for the implementation, deployment, and integration activities presented in the following sections of this deliverable.

The 6G-VERSUS project is structured around six pilot clusters across Europe, namely the Bulgarian, Greek, Spanish, Portuguese, French, and Finnish clusters, each addressing distinct sustainability-oriented vertical use cases through the deployment and validation of the developed application triplets across domains such as energy, robotics, immersive communications, smart infrastructures, critical infrastructure management, and sustainable telecommunications.

In the context of the project validation activities, 6G-VERSUS is structured into two main trial phases (Phase 1 and Phase 2). Each phase includes two trial events (Trials 1–2 and Trials 3–4 respectively) across the participating clusters, progressively increasing the maturity, integration level, and validation scope of the developed application triplets. This distinction is referenced throughout the document in order to clearly differentiate the developments, demonstrations, and results. The “Under Development for Phase 2” sections provide an overview of the application components currently under design or active implementation, including their targeted functionalities, planned integration approach, and expected role within the upcoming Phase 2 trials and validation activities.

2.1 Transformation of Use-Cases to 6G Applications

Fifth and beyond Networking Generations such as B5G and 6G enable a far more systematic interaction between third-party entities and the network control plane as opposed to previous networking generations. This is largely achieved through exposure and analytics functions, including NEF [1] northbound APIs and NWDAF analytics services [2]. In parallel, 3GPP introduced an application enablement layer for verticals [3], with Vertical Application Enablers (VAEs) emerging in Release 16 [4] and evolving further thereafter. This direction supports reusable capabilities such as message distribution, service continuity, application resource management, and group management, while enabling more automation [5] through analytics-driven decisions and predictions. 3rd Generation Partnership Project (3GPP) Common API Framework (CAPIF) [6] complements the above picture by providing a common framework and security baseline for northbound API exposure and onboarding.

In 6G-VERSUS, the core idea is to transform each use-case application into a 6G application that is natively built for capability exposure, analytics, and AI-assisted control loops. The project is built on (i) the VAE mindset for vertical enablement [4], (ii) the Network Applications (NetApps) concept introduced in 5GPP Phase 3 [7], and (iii) a tighter AI-network coupling to support zero-touch operation [8]. Accordingly, we structure each 6G application as a V-App / N-App / AI-App triplet portrayed in Figure 1.

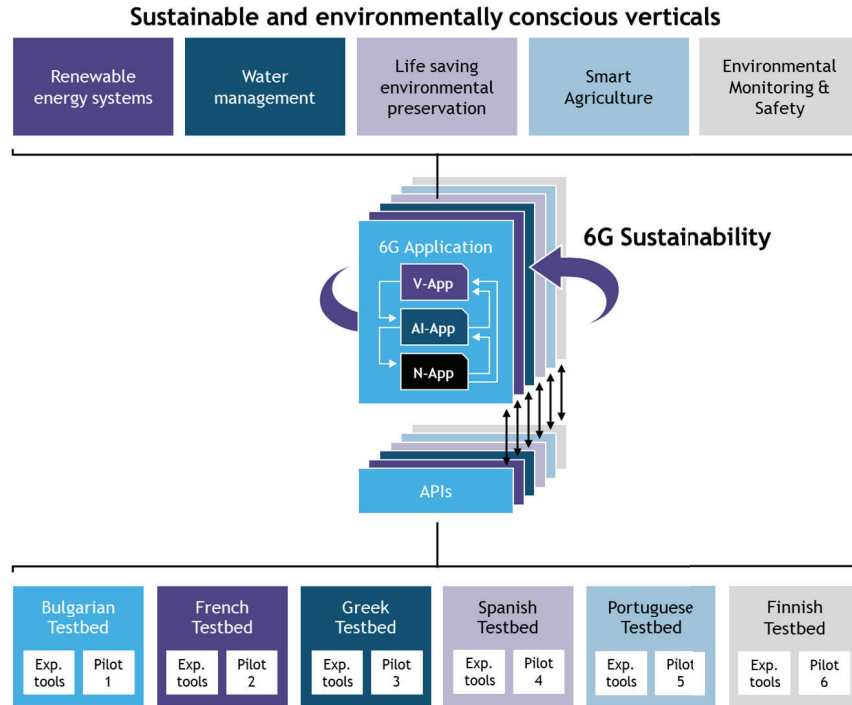


Figure 1 6G application as a V-App / N-App / AI-App triplet depicted

2.2 Definition of N/V/AI-Apps in 6G-VERSUS

This section provides a concise synopsis of the N/V/AI-App definitions introduced in WP2, primarily based on the 6G-VERSUS reference architecture and application triplet concept presented in 6G-VERSUS [Deliverable D2.1](#), adapted here to provide the necessary contextual background for the development-oriented reporting of D4.1.

In 6G-VERSUS, we structure each 6G application as a N-App / V-App / AI-App triplet depicted in Figure 1:

- The V-App captures the **vertical logic**. It poses a primary user interface tailored to domain-specific requirements, providing context awareness including real-time data insights for vertical applications.
- The **AI-App** encapsulates **decision and prediction logic**, fed by network-driven and user-driven data [2], to improve contextual awareness and optimise actions [5] across control and data planes. It leverages machine learning and data analytics to transform raw data into intelligence-based actionable insights, optimizing service quality and energy efficiency.
- The N-App is both the **integration anchor** and **mediator between (i) the vertical service and AI-assisted operations and the (ii) the Network**. As such, it lies at the borderline between the network architecture layer and the 6G-Application, consuming **native APIs** such as NEF [1], and NWDAF [2], and exposing business APIs “upwards” to the V-App and “sideways” to the AI-App, hence mediating between the V-App and the AI-App for optimised decision-making. In order to consume their native APIs, the N-App uses CAPIF to access and consume core Network Functions (NFs) in a standardized, unified and secure API exposure fashion, enabling seamless integration, authentication, and access control. AFs, on the other, leverage CAPIF to expose their northbound APIs for data access and Management & Orchestration (M&O), hence allowing verticals to make decisions or impact M&O decisions via the V-App and/or AI-App communication through N-App. The N-App, on the other, uses

CAPIF APIs to facilitate interaction with testbeds, enabling programmability and adaptability of connectivity services.

The 6G Application triplet is designed to be executed and support experimentation [8] across six pilots and seven use cases, aligning with the broader European view on 6G architecture evolution [9], including the gradual convergence of exposure, analytics, automation, and AI-native functions.

2.3 Interfaces between N/V/AI-Apps

The project architecture relies on standardized reference points to ensure interoperability between vertical logic and network infrastructure (see Figure 2). These interfaces are classified into Northbound/Southbound and East-West communication flows.

Northbound and Southbound Interfaces (NBI/SBI)

The N-App serves as the primary mediation layer between the vertical application and the underlying 5G/6G infrastructure.

- **N-App Interfaces:** The N-App must expose Northbound Interfaces (NBIs) to provide abstracted network services to the V-App layer. Simultaneously, it maintains Southbound Interfaces (SBIs) to communicate with the 5G Core (e.g., NEF, NWDaf) or the RAN via the CAPIF framework.
- **V-App Interfaces:** V-Apps consume network capabilities through their SBIs, which link directly to the N-App's exposed APIs.

East-West Interfaces for Intelligence and Lifecycle Management

To facilitate advanced analytics and the "Model-as-a-Service" (MaaS) paradigm, specialized East-West interfaces are utilized:

- **AI-App Interaction:** AI-Apps must expose at least one East-West interface to either a V-App or an N-App. This allows the AI-App to ingest vertical data (from the V-App) or network-level telemetry (from the N-App) to generate inference results.
- **AI/ML Ops Integration:** To support continuous integration and deployment of intelligence, V-Apps, N-Apps, and AI-Apps may expose East-West interfaces to the AI/ML Ops framework. This enables automated model training, performance monitoring, and lifecycle management of ML models.

Inter-Component Structural Relationships

The architecture shown in Figure 2 supports flexible deployment topologies to accommodate the diverse requirements of the six clusters. Relationships between V-Apps, N-Apps, and AI-Apps are defined as 1–1, N–1, 1–N, or N–N interfaces. This cardinality allows for example a single high-performance N-App to serve multiple vertical clients or a centralized AI-App to aggregate data from multiple network sources.

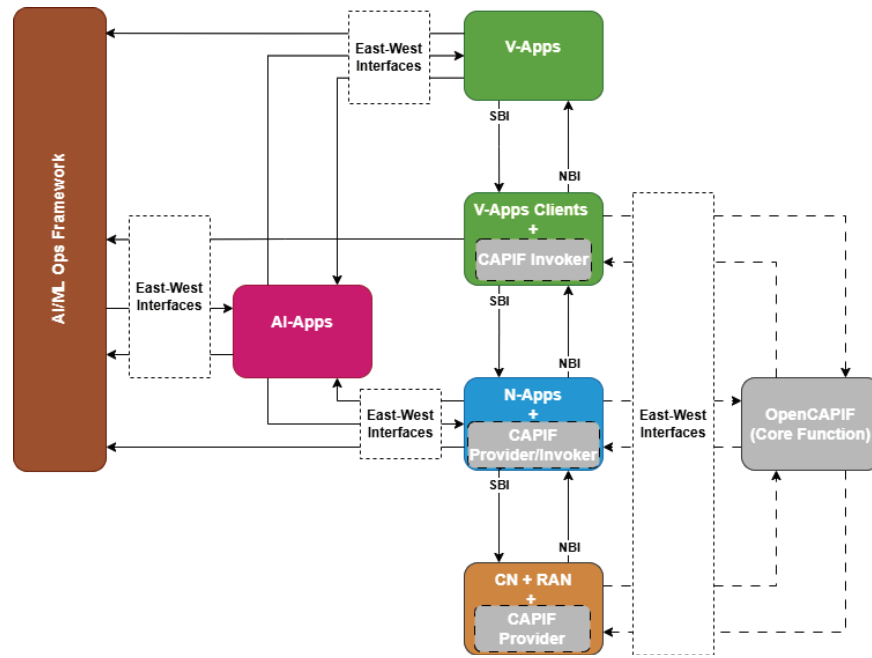


Figure 2 Interaction Architecture of V/AI/N-Apps Triplets and Testbed Components

In the architecture illustrated in Figure 2, all V-Apps and AI-Apps communications with the network are mediated through N-Apps, ensuring abstraction, control, and security. CAPIF (Common API Framework) provides a standardized way to expose and consume network services, with two main components: the Invoker (service consumer) and the Provider (service exposers). In our implementation, we rely on OpenCAPIF to enable flexible deployment and interoperability. CAPIF components (i.e., Invoker and Provider) can be flexibly deployed across different layers to support various interaction patterns between applications and network functions. In some cases, both Invoker and Provider can reside within the same component (e.g., an N-App), enabling efficient service chaining and reducing latency.

- **CAPIF Invoker deployment:**
 - Can be located in the V-App Clients layer, enabling vertical applications onboard and discover exposed network services through CAPIF and consume them through the N-Apps.
 - Can also be embedded within N-Apps, allowing them to act both as service consumers and intermediaries between network functions and higher-layer applications.
- **CAPIF Provider deployment:**
 - Can be implemented within N-Apps, exposing processed or abstracted network capabilities to V-Apps and AI-Apps.
 - Can also reside directly in the Core Network (CN) + RAN layer, exposing native 5G Core and RAN services (e.g., NEF, NWDAF, or RAN telemetry) through CAPIF and N-Apps.

This flexible placement enables multiple service exposure models depending on the use case, ranging from direct network exposure to more abstracted and application-driven interfaces (See Table 1).

Table 1 CAPIF Invoker and Provider Deployment Scenarios

Scenario	CAPIF Invoker Location	CAPIF Provider Location	Description
1	V-App Client	CN + RAN	Direct exposure of network APIs; CAPIF handles discovery and authentication
2	V-App Client	N-App	N-App exposes abstracted APIs; CAPIF manages service registry and access
3	N-App	CN + RAN	N-App handles API consumption; CAPIF used for provider discovery and tokens

Scenario	CAPIF Invoker Location	CAPIF Provider Location	Description
4	N-App	N-App	Chained services; CAPIF manages multi-provider discovery and security

3 Development of N-Apps

The present section reports the development status of the Network Applications (N-Apps) across the six 6G-VERSUS clusters. As part of the 6G application triplet concept, N-Apps act as the interface between the vertical applications and the underlying 5G/6G infrastructure, enabling network programmability, capability exposure, telemetry collection, and interaction with core network functions through standardized APIs and frameworks such as CAPIF. The section presents the implemented N-Apps, their architecture, interfaces, deployment status, and current integration progress within each cluster.

3.1 Common Development Process and N-App Template

To streamline the development process and ensure consistency across all clusters, an N-App template has been introduced within the project. This template consists of python scripts and Make target rules so to provide a clear, practical example of how to integrate the OpenCAPIF framework using its SDK, enabling secure authorization for the APIs. The template demonstrates the end-to-end onboarding and authentication process for both Provider and Invoker applications through OpenCAPIF. Additionally, it leverages the [FastAPI](#) framework for both exposing and consuming services, offering an efficient approach to building and integrating telecom API workflows.

The template's architecture consists of:

- **Provider Onboarding App**
 - Publishes a service API via OpenCAPIF.
 - Validates incoming requests using CAPIF-issued certificates and JWT tokens.
- **Invoker App**
 - Discovers and consumes provider's services through OpenCAPIF.
 - Retrieves ([JWT](#)) tokens from OpenCAPIF [6] for authorized requests.
- **User Application**
 - Acts as an external application that consumes the data retrieved from the Invoker Application endpoint.

Furthermore, two flavours of the template are provided. One utilizes an external sftp server for exchanging the OpenCAPIF's certificate after the provider onboarding through the template towards provider applications for authorizing their incoming requests. While this is enough, for a more flexible integration solution, a native integration of OpenCAPIF's functionality, with regards to provider onboarding to OpenCAPIF, inside the provider's application is showcased for OpenCAPIF's certificate for authorizing incoming requests. A workflow sequence diagram is provided in Figure 3 to showcase the integration with OpenCAPIF for the NApp template. It mainly consists of the following steps:

1. **User Registration**
 - a. A user is created in CAPIF using available python scripts and json configurations.
 - b. The same user can be both a provider and an invoker depending on configuration and the use case needs.
2. **Provider Onboarding**
 - a. The provider is onboarded to CAPIF via available python scripts and json configurations.
 - b. The service is published using the provider's app OpenAPI YAML schema.
 - c. CAPIF returns `capif_cert_server.pem`, stored under a configurable folder path, used later for verifying JWT tokens.
3. **Invoker Onboarding & Discovery**
 - a. The invoker is onboarded to CAPIF via existing python scripts and json configurations.
 - b. CAPIF provides service discovery results and returns a **JWT token**.
 - c. This token allows the invoker to send **authorized requests** to the provider's app.

4. Authorized Communication

- a. The invoker consumes the provider's API upon request of **V/AI-App**.
- b. The provider verifies the JWT token against the CAPIF certificate (capif_cert_server.pem) to authorize requests.

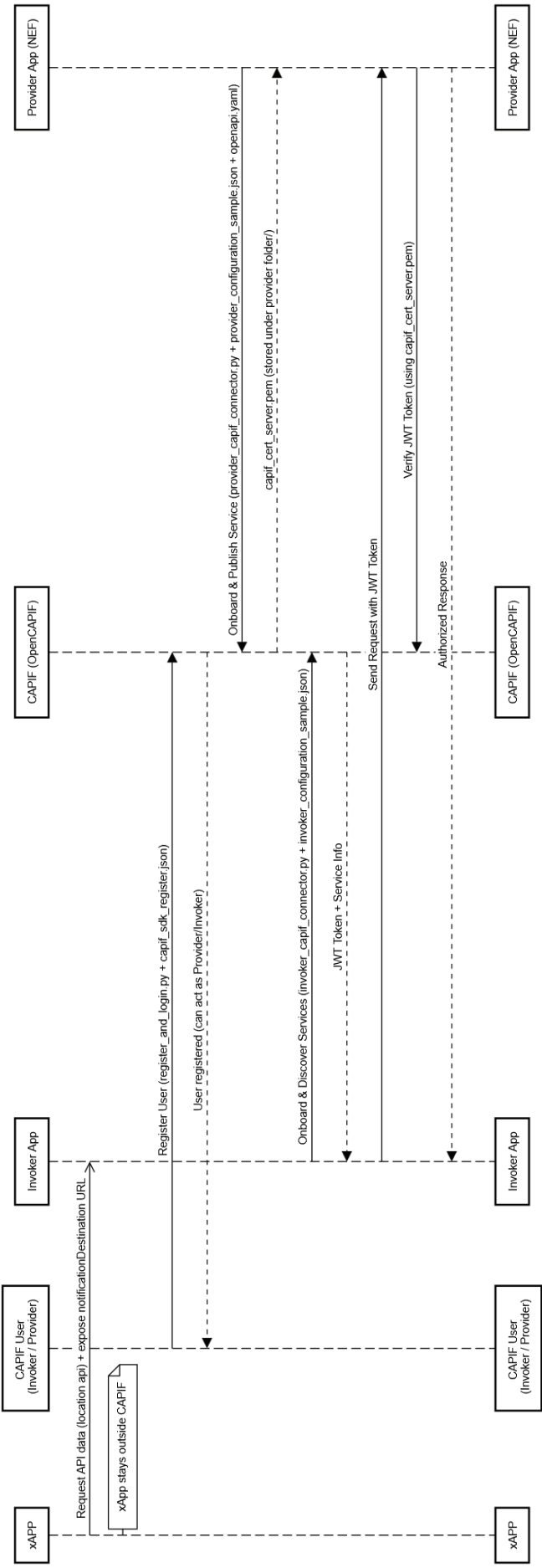


Figure 3 N-App template sequence diagram

A [GitLab Repository](#) that illustrates the N-App template and related scripts and configurations is available at the project's GitLab server alongside with flow charts and sequence diagrams that describes the operational workflow of the N-App template.

3.2 Bulgarian Cluster

The Bulgarian cluster contains a single Network Application (N-App) under development whose primary role is to mediate between the 5G Core Network and the V-App Central. It acts as the integration layer that both consumes control/data from the core and presents northbound capabilities to the V-App Central. Network data and telemetry are exposed to V-App Central and authorized consumers through OpenCAPIF

Cluster Name	App Name	Phase (1or 2)
Bulgarian	N-App for Network QoS Monitoring and Improvement	Phase 1 & 2
Bulgarian	N-App for Event Exposure Network API	Phase 1 & 2

3.2.1 N-App for Network QoS Monitoring and Improvement

3.2.1.1 Overview

In the Bulgarian cluster, the architecture incorporates one Network Application (N-App) designed to interface with underlying network functions of the Core Network. The N-App will act as an essential bridge between the 5G Core Network (CN) and the upper V-App Central. Network APIs are exposed using OpenCAPIF.

3.2.1.2 Architecture

As seen in Figure 4, the N-App receives network performance data from A1 performance and monitoring platform. This N-app provides network data to V-App Central and applies requests to the core network.

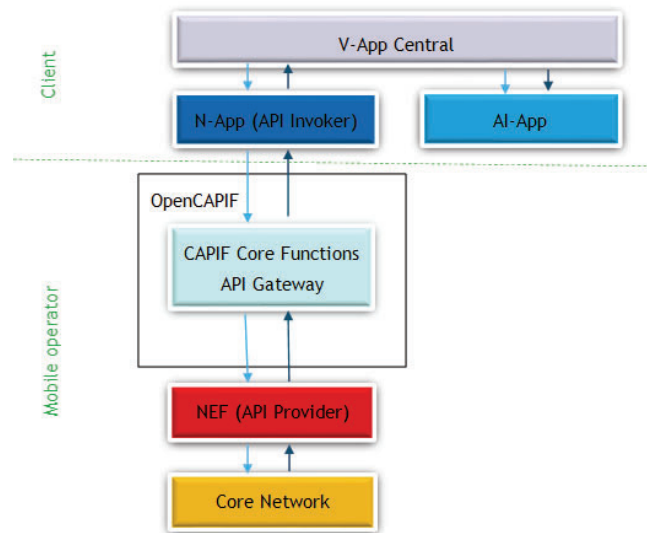


Figure 4 N-App for QoS Monitoring and improvement

3.2.1.3 Analysis and development of 5G APIs

The Bulgarian pilot in 6G-VERSUS project aims to implement Use Case scenarios which consider the Network Exposure capabilities of the B5G A1 network and to configure and provide the following 5GC NEF Network APIs:

- AF Session with Required QoS Network API (see Table 2)

- Event Exposure Network API (see Table 3)

AF Session with Required QoS Network API Overview

The Application Function (AF) session with required QoS service is to allow the AF to influence the QoS of one or multiple service data flows of one ongoing PDU session of a User Equipment (UE). Also, this service notifies the AF on the service data flow related events if subscribed by AF.

The QoS Network Service enables the following:

- AF creates a new subscription for AF session with required QoS service.
- AF partially updates an existing subscription for AF session with required QoS service.
- AF gets an existing subscription data for AF session with required QoS service.
- AF deletes an existing subscription for AF session with required QoS service.
- Advanced Service: AF using UE public IP or MSISDN creates a new subscription for AF session with required QoS.

Table 2 AsSessionwith QoS API Methods

REST Method	Resource URI	Body Parameter	Description
POST	3gpp-as-session-with-qos/v1/{afId}/subscriptions (IPv4addr, msisdn, flowInfo (assigned))	afId; flowInfo;qosReference; ueIpv4Addr; qosMonInfo;	Creates a new subscription resource
GET	/3gpp-as-session-with-qos/v1/{afId}/subscriptions/<subscription Id>	afId; subscriptionId	Read an active subscriptions for the AF and the subscription Id
PATCH	/3gpp-as-session-with-qos/v1/{afId}/subscriptions/{subscription Id}	afId; subscriptionId; flowInfo; qosReference; ueIpv4Addr; qosMonInfo;	Updates/replaces an existing subscription resource
DELETE	{apiRoot}/3gpp-as-session-with-qos/v1/{afId}/subscriptions/{subscriptionId}	afId; subscriptionId	Deletes an already existing subscription

Event Exposure for 5GC acting as a network function of NEF, is intended for monitoring specific events in the 5GC and making the information of these monitoring events available for AF via the NEF. Event Exposure is that authorized AFs can get the events notification or immediate event status of the UE, for example the UE reachability, Loss of Connectivity and PDU Session Status in 5GC term, which are valuable for AF to provide more services. The Monitoring Event (ME) resource allows API Invokers or Application Function (AFs) to read all of the active monitoring event subscriptions or create a new monitoring event subscription at NEF through the 3GPP-monitoring event API. The Monitoring Event API allows the AFs to subscribe to notifications about specific events in 3GPP networks. It also allows the NEF to report the event by sending notifications to the authorized users when the corresponding event is detected. The 3GPP-monitoring-event facilitates the following operations:

Table 3 Monitoring Event API Methods

REST Method	Resource URI	Body Parameter	Description
-------------	--------------	----------------	-------------

POST	/3gpp-monitoring-event/v1/{afId}/subscriptions	{"msisdn":"123456789101112", "notificationDestination":"http://test_server-af:8001/af/notifications", "monitoringType": "UE_REACHABILITY/LOSS_OF_CONNECTIVITY/PDN_CONNECTIVITY_STATUS", "reachabilityType": "DATA", "idleStatusIndication": "true",	Create a MonitoringEvent subscription or fetch status of UE_Reachability_Status; UE_Reachability ; Loss_Of_Connectivity
GET	/3gpp-monitoring-event/v1/{afId}/subscriptions/{subscriptionId}	afId; subscriptionId	Read an active subscription for the AF and the subscription Id
DELETE	/3gpp-monitoring-event/v1/{afId}/subscriptions/{subscriptionId}	afId; subscriptionId	Deletes an already existing subscription

NEF supports the creation of discovery groups that are assigned to API invokers to enable controlled and administered access to service APIs. Within the CAPIF framework, a discovery group defines a collection of API Exposing Functions (AEFs) and their related APIs, allowing specific NEF APIs to be made available to selected API invokers. CAPIF **serves as a secure unified framework for API exposure and security context management.**

3.2.1.4 Integration of APIs with OpenCAPIF

The N-App is developed as Python application using OpenCAPIF SDK. The scripts for interaction with OpenCAPIF are included – user creation, provided onboarding, invoker registration. Using of N-App requires installation and administration of OpenCAPIF instance by the Mobile Operator.

3.2.1.5 Development, Deployment and Results

The N-App is developed as Python application with fastapi and uvicorn libraries.

The N-App development utilises the functionality of GitLab. The N-App (invoker) and the NEF service run in Docker containers and are deployed via Helm charts in a Kubernetes cluster. The functionality of the N-App, which uses Monitoring Event API, is tested and integrated with V-App Central.

- The link to the sources of the N-App (Invoker):
https://gitlab.6g-versus.eu/wp4/napps/bulgarian-cluster/scbg/-/tree/main/invoker_impl
- The link to the sources of the NEF service in GitLab:
https://gitlab.6g-versus.eu/wp4/napps/bulgarian-cluster/scbg/-/tree/main/nef_monitoring_ran
- The link to the source of testing application for the NEF service:
<https://gitlab.6g-versus.eu/wp4/napps/bulgarian-cluster/scbg/-/tree/main/TestNEF>
- The link to the source of testing application for the N-App (Invoker):
<https://gitlab.6g-versus.eu/wp4/napps/bulgarian-cluster/scbg/-/tree/main/TestInvoker>

Python scripts are used for administrative tasks of the mobile operator connected with onboarding of the provider in OpenCAPIF.

- Script for the login as administrator and creation of user in OpenCAPIF:
https://gitlab.6g-versus.eu/wp4/napps/bulgarian-cluster/scbg/-/tree/main/user_creation
- Script for the registration of the provider in OpenCAPIF:
https://gitlab.6g-versus.eu/wp4/napps/bulgarian-cluster/scbg/-/tree/main/provider_impl

3.2.2 N-Apps under Development for Phase 2

For Phase 1 the cluster will showcase N-App, which uses Monitoring Event API. For Phase 2 the N-App will include functionality of QoS Network API and be sufficiently tested.

3.3 Finnish Cluster

The Finnish cluster has one N-App under development (see Table 4). N-App in Finnish cluster acts as a control and data transfer function responsible for storing various data in a shared database by communicating with triplet and other applications in the solution. The Finnish cluster participates only in Phase 1 (Trial 2), which involves running a mature version of N-App as a part of a triage, where our self-sustainable base station solution is run for a full 1-year period. During the year Trial 2 is run, diagnostics data is gathered and will be used to improve the implementation of N-App for future work. For Trial 2, the cluster will showcase a mature N-App, which is sufficiently developed and integrated and should include all functionality and be fully tested and bugs fixed.

Table 4 Implemented and under development N-App for the Finnish cluster

Cluster Name	App Name	Phase (1or 2)
Finnish	Control and Data Transfer Function (CDTF) N-App	Phase 1

3.3.1 N-App – Control and Data Transfer Function

3.3.1.1 Overview

In the architecture of the Finnish cluster, there is one Network Application (N-App). The responsibility of N-App is to receive data from other application components and store them in a shared database. Other responsibilities of N-App consist of fetching meteorological weather forecast data via an interface provided by the Finnish Meteorological Institute (FMI) upon invocation from V-App. The fetching of weather forecast data was introduced in N-App because in our architecture N-App is a data plane application directly accessing the database so it was more natural to have the fetching feature in N-App to reduce unnecessary and resource-consuming signaling between applications when N-App is already connected to the database. Further, in our architecture, V-App is a graphical user interface responsible for handling user interaction so also from this perspective N-App was a natural choice to handle fetching and storing forecast data.

3.3.1.2 Architecture

In Figure 5, the architecture of N-App is described in detail. In Finnish cluster, N-App acts as a simple collection point of data from various components (i.e. remote intelligent controller, which is sending power production and consumption data; network intelligent controller, which is sending application specific power consumption data; X-App, which is sending UE-specific network statistics data; AI-App1, which is sending ready AI decision for storing). N-App is connected to a database, which maintains all the received data from other components.

N-App also communicates with V-App on two cases: 1) on 3-hour basis requests N-App to fetch weather forecast data from FMI and store it in the database, 2) N-App calls V-App REST API to inform V-App that new AI decision is available in the database for accessing.

All communications take place over REST APIs.

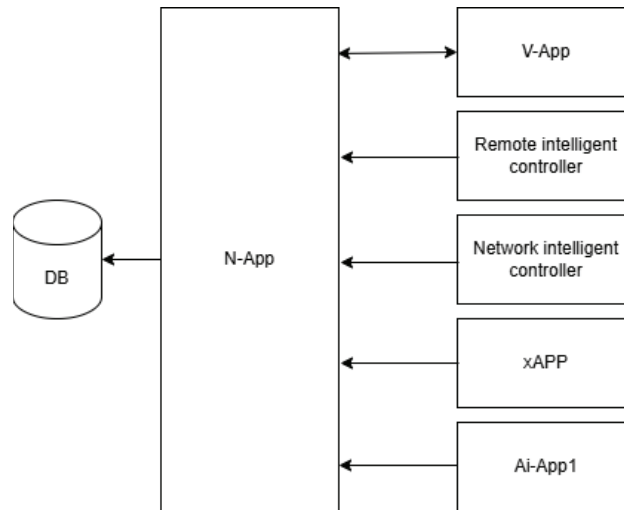


Figure 5 The architecture of N-App.

3.3.1.3 Analysis and development of 5G APIs

The Finnish Cluster SW architecture it really is so that N-App does not need to utilize any 5G APIs. As that is the case there are no APIs to be listed. The Finnish Cluster architecture does not require the utilization of standardized 5G APIs, since the implemented N-App fulfills its role through monitoring and interaction with lower-layer system components in accordance with the N-App definition provided in D2.1.

3.3.1.4 Integration of APIs with OpenCAPIF

The integration work of APIs with OpenCAPIF is still ongoing. We are currently determining the best way to integrate our solution with OpenCAPIF. Most likely we will utilize CAPIF Gateway to provide authentication and authorization for critical V-App-to-N-app communications.

3.3.1.5 Development, Deployment and Results

N-App is implemented as a simple Flask application using Python programming language. N-App consists of one class (Flask), which has six decoration functions, each acting as a separate REST API end-point. The software architecture of N- App is depicted in Figure 6.

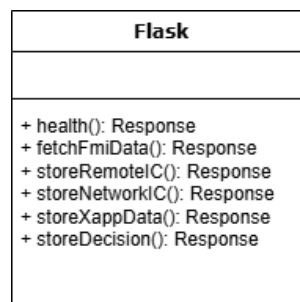


Figure 6 N-App software architecture.

The first end-point health() is exposed to V-App and acts as a simple ping end-point. The second end-point fetchFmiData() is for V-App to nudge N-App to fetch weather forecast data from FMI and store it in the database. Then there are storeRemoteIC() and storeNetworkIC() end-points, which are used to receive energy measurement data. The storeXappData() is exposed to X-App for receiving per-UE E2 interface data. Finally, storeDecision() is used by AI-App1 to send and store AI decisions to the database. All above-mentioned decoration functions (end-points) return a Flask Response with a status code indicating whether the invocation was successful or not. N-App is uploaded in the project's Gitlab and can be accessed via <https://gitlab.6g-versus.eu/wp4/napps/finnish-cluster/N-app>.

The N-App is deployed as a containerized application using Docker within the UOULU testbed infrastructure. The Docker containerized approach ensures portability, scalability, and ease of deployment across different environments. The deployment of N-App is in align with microservice architecture, where all the triplet applications and other components (network intelligent controller and remote intelligent controller) are connected and communicate via same Docker network on a shared server.

Development work of N-App has almost been concluded and the N-App application has been tested on a local development machine as part of the development work. The results of REST API test runs are shown in Figure 7, which shows that every call to APIs provided by the N-App is successfully processes (i.e. the status code is 200 – OK). By Trial 2, N-App integration tests, which involve all triplet applications and other components, will be conducted once the Finnish cluster's solution has been fully integrated and deployed in the UOULU testbed.

```
+ curl -w '%{http_code}\n' -sS http://localhost:5000/health/
200
+ curl -w '%{http_code}\n' -sS -X POST http://localhost:5000/start/
200
+ curl -w '%{http_code}\n' -sS -X POST http://localhost:5000/xapp-data/ -H 'Content-Type: application/json' -d '{"ue_count": 2, "ue": [{"ue_id": 2, "timestamp": 1754303881569, "ul_mcs_mean_last_hour": 7, "dl_mcs_mean_last_hour": 8, "ul_bler_mean_last_hour": 0.008863, "dl_bler_mean_last_hour": 0.0, "sinr_mean_last_hour": 22.5, "rsrp_mean_last_hour": -73, "rsrq_mean_last_hour": -10.5, "ul_throughput_bytes_mean_last_hour": 233742, "dl_throughput_bytes_mean_last_hour": 233742, "ul_throughput_bytes_max_last_hour": 234002, "dl_throughput_bytes_max_last_hour": 234002}]}'
200
+ curl -w '%{http_code}\n' -sS -X POST http://localhost:5000/remote-ic-data/ -H 'Content-Type: application/json' -d '{"storage": {"soc": {"value": 50.5, "timestamp": 1774953603680}, "soh": {"value": 32, "timestamp": 1774953603680}, "hydrogen_hours_left": {"value": 50, "timestamp": 1774953603680}, "hydrogen_tank_amount": {"value": 100, "timestamp": 1774953603680}}, "sensing": {"windspeed": {"value": 3.4, "timestamp": 1774953603680}, "wind_direction": {"value": 65.7, "timestamp": 1774953603680}, "solar_irradiance": {"value": 34.3, "timestamp": 1774953603680}, "temperature_outside": {"value": -19.4, "timestamp": 1774953603680}, "temperature_battery": {"value": 15.4, "timestamp": 1774953603680}, "temperature_hydrogen": {"value": 20.4, "timestamp": 1774953603680}, "temperature_devicebox": {"value": 17.4, "timestamp": 1774953603680}}, "production": {"wind": {"value": 30.4, "timestamp": 1774953603680}, "solar": {"value": 50.4, "timestamp": 1774953603680}, "hydrogen": {"value": 399.4, "timestamp": 1774953603680}}, "consumption": {"radio_unit": {"value": 60.1, "timestamp": 1774953603680}, "microwave_link": {"value": 60.1, "timestamp": 1774953603680}, "hybrid_inverter": {"value": 60.1, "timestamp": 1774953603680}, "hvac": {"value": 60.1, "timestamp": 1774953603680}, "switch": {"value": 60.1, "timestamp": 1774953603680}, "hydrogen_cell": {"value": 60.1, "timestamp": 1774953603680}, "raspberrypi": {"value": 60.1, "timestamp": 1774953603680}, "sensors": {"value": 60.1, "timestamp": 1774953603680}, "netio": {"value": 60.1, "timestamp": 1774953603680}}}'
200
+ curl -w '%{http_code}\n' -sS -X POST http://localhost:5000/network-ic-data/ -H 'Content-Type: application/json' -d '{"measurements": [{"time": 1768820038004, "device": "server", "energy": 21}, {"time": 1768820038016, "device": "radio", "energy": 46}, {"time": 1768820038030, "device": "radio", "energy": 32}]}'
200
+ curl -w '%{http_code}\n' -sS -X POST http://localhost:5000/ai-decision/ -H 'Content-Type: application/json' -d '{"timestamp": 1772709600000, "hydrogen_on": true, "battery_soc_limit": 30.5, "pv_power_kw": 2.4, "wind_power_kw": 5.1, "site_load_kw": 6.0, "energy_surplus_kw": 1.5, "predicted_traffic_load": 0.82, "oai_box_configuration": {"mimo": "4x4", "bandwidth_mhz": 100, "tdd_ratio": "DDDDDDDFUU"}, "oai_box_energy_state": "power saving"}'
200
+ curl -w '%{http_code}\n' -sS -X POST http://localhost:5000/fmi/
200
+ curl -w '%{http_code}\n' -sS -X POST http://localhost:5000/stop/
200
```

Figure 7: Results of the N-App API tests

3.3.2 N-Apps under Development for Phase 2

The N-App is currently under development and will be introduced in Phase 1 (Trial 2). Once implemented, this N-App implementation will be used in Phase 2.

3.4 Greek Cluster

Within the Greek cluster and based on the use case as well as the different phases of the scenario, four N-Apps are targeted, as presented in the Table 5 below. During the first round of trials, the N-App for location awareness will be demonstrated, as it constitutes the first reference implementation based on the N-App template. It will also be leveraged in Phase 2, enriched with additional functionalities enabled by the Monitoring Event API. The majority of the N-Apps in the Greek cluster are planned for Phase 2, namely the N-App for Coverage Extension, the EEaaS N-App, and the QoS N-App. Phase 2 of the Greek Pilot, scheduled

for Trials 3 and 4 in 2027, focuses on the development and validation of these three N-Apps, complementing the capabilities introduced in Phase 1 (Trials 1 and 2). Specifically, the Coverage Extension N-App aims to support connectivity extension and service continuity through UAV-assisted relay (rUAV) mechanisms, particularly in cases where 5G coverage or QoS for the vertical application (i.e., the INUS V-App) becomes degraded or unavailable. In parallel, the EEaaS N-App is designed to enable the monitoring and exposure of energy consumption across distributed infrastructure components, supporting enhanced energy observability and reporting at both system and device levels. Finally, the QoS N-App will allow the selection of different QoS profiles from the network, which will be further exploited by the V-App for wildfire monitoring. All previously mentioned N-Apps are currently under development and are planned for validation in Phase 2, with their final stable versions to be reported in D4.2.

Table 5 Implemented and under development N-Apps for the Greek cluster

App Name	Phase (1or 2)
N-App for location awareness	1 and 2
N-App Coverage Extension	2
N-App EEaaS	2
N-App for QoS	2

3.4.1 N-App for Location Awareness

3.4.1.1 Overview

The N-App for location awareness provides the required means for an external application to communicate and request cell-level location information area through the open5GS core by utilizing a 3GPP NEF MonitoringEvent API for location info feature. In the context of the Greek use case the AI-App for location and in general the upper external applications, can receive real time location information for a requested UE that is registered in a cell. The communication between the 5G Native NEF MonitoringEvent API and the invoker service that pass the data to the AI-App is based on N-APP template that was developed from NCSR for that project, thus the data is securely exposed by the means of CAPIF's security context.

3.4.1.2 Architecture

The implementation follows entirely the N-App template that has been provided. Particularly, it consists of an invoker application, provider application and its interactions with CAPIF instance as well as the interaction between the provider application with 5G Core Network. In a much more sense, the provider application includes the implementation of NEF 3GPP MonitoringEvent API which provides mechanisms to fetch location data information from 5G Core Network towards the invoker application. This is done, via invoker's application interaction with any external application that asks for network location data in a way that an invoker securely requests that data on behalf of the external application from NEF API by leveraging the CAPIF Framework for securely access the NEF's API endpoints that has been exposed as well through CAPIF. After the invoker application receive any response from the provider application, in our case NEF MonitoringEvent API, passes it in the external application that initially asked for location information data.

3.4.1.3 Analysis and development of 5G APIs

Within the Greek Cluster, the implementation of a 3GPP 5G NEF MonitoringEvent API has been provided. It exposes [3GPP TS 29.122](#) Monitoring Event API (see Table 6) operations for subscription creation, retrieval, deletion, and immediate last-known-location queries. Particularly, the MonitoringEvent API offers location monitoring capabilities to vertical applications in the context of cell level location information coverage area. It provides both real time monitoring capabilities based on asynchronous and event driven cloud native 5G architecture as well as last known location information area according to the latest known location information

that is stored in its internal database alongside with the active subscriptions that may be created for current-location monitoring. The implementation utilizes separation of concerns principle for a cloud native architecture on which business logic and security authorization are separated, while the latter is achieved by utilizing [CAPIF framework](#). The CAPIF framework is used to provide a secure unified entry point for exposing and managing the API's security context. The API is compatible with an Open5GS core network.

Table 6 MonitoringEvent API methods

REST Method	Resource URI	Body Parameter	Description
POST	/3gpp-monitoring-event/v1/{scsAsId}/subscriptions	{ "msisdn": "001010143245445", "notificationDestination": "http://test_server:8001", "monitoringType": "LOCATION_REPORTING", "locationType": "LAST_KNOWN_LOCATION" }	Create a MonitoringEvent subscription or fetch last known location
GET	/3gpp-monitoring-event/v1/{scsAsId}/subscriptions/	-	Read all of the active subscriptions for the AF
GET	/3gpp-monitoring-event/v1/{scsAsId}/subscriptions/{subscriptionId}	-	Read an active subscription for the AF and the subscription Id
DELETE	/3gpp-monitoring-event/v1/{scsAsId}/subscriptions/{subscriptionId}	-	Deletes an already existing subscription

3.4.1.4 Integration of APIs with OpenCAPIF

The integration follows the N-App template where CAPIF is deployed as an instance in the cloud native environment where both provider's application and invoker's application can communicate with it. To that extent, the N-App template provides some python scripts that automates the process of onboarding and publishing the NEF API as well as the process of onboarding the invoker application, discovering existing published APIs and retrieve a token that will be used for requesting securely location data from the NEF API. The python scripts utilize the python CAPIF SDK that is provided from the CAPIF community.

3.4.1.5 Development, Deployment and Results

The implementation is stored in the Gitlab's project [repository](#). The application has been developed using python, utilizing FastAPI framework for building the required REST APIs while using pymongo client to retrieve and store data in its own mongodb database that is used internally. Lastly, the use of make and docker tools has been utilized for an ease of deployment of the application in a cloud native environment.

References links:

- [Python](#)
- [FastAPI](#)
- [Pymongo](#)

- MongoDb
- Make
- Docker

In terms of development, an automation workflow with the usage of make tool has been implemented to automate the infrastructure deployment of the implementation. All the components that conclude the N-App for location awareness are implemented and deployed as a cloud native architecture, so the deployment consists only of containerized applications communicating with each other. Figure 8 presents the creation of subscription through the NEF Monitoring Event API reflecting the utilisation of the N-App and the MonitoringEvent API.

[illegible]

Figure 8 NEF Monitoring Event API container logs

The log file in Figure 9Figure 9 presents the complete interaction and the notification phase with the Monitoring Event API for location reporting of the previous subscription. A POST request is sent to the /3gpp-monitoring-event/v1/subscriptions endpoint to create a monitoring subscription, targeting a UE identified by their IMSI, which uniquely represents the device in the mobile network. The subscription is configured with the monitoring type Location reporting and requests the last known location of the device. As part of this, the network provides cell-level information, including the Cell ID and Tracking Area ID, which indicate the current position of the device. Once the subscription is successfully created (confirmed by a 200 OK response), the system sends a notification callback containing the requested location data. This callback includes key details such as the IMSI, location identifiers, and timing information.

[illegible]

Figure 9 Invoker's application Successful container logs

Figure 10 presents the deployment of all Open5GS core network functions, including the NEF, as individual containers. It also presents the N-app (Invoker application), which consumes the API of location reporting event. On the other hand, Figure 11 illustrates the CAPIF instance deployed in the Athens B5G platform that is utilised for the exposure of the Monitoring event API.

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS
62d73fbd586	grafana/grafana-oss:10.4.0	"/run.sh..."	5 weeks ago	Up 5 weeks	0.0.0.0:3000->3000/tcp
11fbaab19b0	prom/prometheus:v2.51.0	"/bin/prometheus --c..."	5 weeks ago	Up 5 weeks	0.0.0.0:9090->9090/tcp
5ed077ac061b	webui:v2.7.5	"npm run dev"	5 weeks ago	Up 5 weeks	0.0.0.0:9099->9099/tcp
0da97462362	smf:v2.7.5	"open5gs-smfd -c /et..."	5 weeks ago	Up 5 weeks	0.0.0.0:8088->808/tcp, 0.0.0.0:9095->9090/tcp
0e028005f4c	udr:v2.7.5	"open5gs-udrd -c /et..."	5 weeks ago	Up 5 weeks	0.0.0.0:8183->808/tcp
f104aee0b4c	pcf:v2.7.5	"open5gs-pcfd -c /et..."	5 weeks ago	Up 5 weeks	0.0.0.0:8086->808/tcp, 0.0.0.0:9092->9090/tcp
5a5553bf46f1	ausf:v2.7.5	"open5gs-ausfd -c /e..."	5 weeks ago	Up 5 weeks	0.0.0.0:8089->808/tcp
c5657555d5b	udm:v2.7.5	"open5gs-udmd -c /et..."	5 weeks ago	Up 5 weeks	0.0.0.0:8182->808/tcp
29496ee3466	amf:v2.7.5	"entrypoint.sh -c /e..."	5 weeks ago	Up 5 weeks	0.0.0.0:38412->38412/sctp, 0.0.0.0:8087->808/tcp, 0.0.0.0:9093->9090/tcp
0a502dc0e297	nssf:v2.7.5	"open5gs-nssfd -c /e..."	5 weeks ago	Up 5 weeks	0.0.0.0:8184->808/tcp
fc39b3bce0b	bsf:v2.7.5	"open5gs-bsfd -c /et..."	5 weeks ago	Up 5 weeks	0.0.0.0:8085->808/tcp
5e4c8d478e6	nrf:v2.7.5	"open5gs-nrfd -c /et..."	5 weeks ago	Up 5 weeks	0.0.0.0:8180->808/tcp
f1ab30df40ac	mongo:8.0	"docker-entrypoint.s..."	5 weeks ago	Up 5 weeks	0.0.0.0:27017->27017/tcp
085ae0720d40	upf:v2.7.5	"entrypoint.sh -c /e..."	5 weeks ago	Up 5 weeks	0.0.0.0:2152->2152/udp, 0.0.0.0:9004->9090/tcp
7b93e2332d48	mongo	"docker-entrypoint.s..."	5 weeks ago	Up 5 weeks (healthy)	0.0.0.0:27018->27017/tcp, [::]:27018->27017/tcp
62f38d0847f4	mongo-template-invoker_app	"sh -c 'uvicorn app..."	2 months ago	Up 2 months	0.0.0.0:8088->8080/tcp, [::]:8088->8080/tcp
023f996d203e	nef-core_crowler	"python -m core_crowl..."	3 months ago	Up 5 weeks	8080/tcp
2dc141933fab	nef-nef	"sh -c 'uvicorn app..."	3 months ago	Up 3 months	0.0.0.0:8080->8080/tcp, [::]:8080->8080/tcp

Figure 10 Open5gs Core alongside N-App for location awareness

014f2003f636	labs.etsi.org:5850/ocf/capif/prod/mock-server:v2.x.x-release	"python mock_server..."	4 months ago	Up 4 months	0.0.0.0:9100->9100/tcp, [::]:9100->9100/tcp
85d0d0e00210	labs.etsi.org:5850/ocf/capif/mongo-express:1.0.0-alpha.4	"tini -- /docker-ent..."	4 months ago	Up 5 weeks	0.0.0.0:8083->8081/tcp, [::]:8083->8081/tcp
505fdd5c30b2	labs.etsi.org:5850/ocf/capif/prod/register:v2.x.x-release	"sh register_prepare..."	4 months ago	Up 5 weeks	0.0.0.0:8084->8080/tcp, [::]:8084->8080/tcp
461d39e31477	labs.etsi.org:5850/ocf/capif/mongo:6.0.2	"docker-entrypoint.s..."	4 months ago	Up 5 weeks	27017/tcp
a950f4f848e1	labs.etsi.org:5850/ocf/capif/prod/ocf-access-control-policy-api:v2.x.x-release	"sh prepare_capif.ac..."	4 months ago	Up 4 months	8880/tcp
8030b0f931d8	labs.etsi.org:5850/ocf/capif/prod/helper:v2.x.x-release	"sh prepare_helper.sh"	4 months ago	Up 4 months	8080/tcp
520e0f4c78a2	labs.etsi.org:5850/ocf/capif/prod/ocf-api-provider-management-api:v2.x.x-release	"sh prepare_provider..."	4 months ago	Up 4 months	8080/tcp
04fbb0e0e4e	labs.etsi.org:5850/ocf/capif/prod/ocf-api-invoker-management-api:v2.x.x-release	"sh prepare_invoker..."	4 months ago	Up 4 months	8080/tcp
018030408af	labs.etsi.org:5850/ocf/capif/prod/ocf-security-api:v2.x.x-release	"sh prepare_security..."	4 months ago	Up 4 months	8080/tcp
034f22ec09f	labs.etsi.org:5850/ocf/capif/nginx-ocf-patched:1.27.1	"sh nginx_prepare.sh"	4 months ago	Up 4 months	0.0.0.0:443->443/tcp, [::]:443->443/tcp, 0.0.0.0:8080->8080/tcp, [::]:8080->
08009/tcp	services-mysql-1				
3f30c0e627c7	labs.etsi.org:5850/ocf/capif/prod/ocf-publish-service-api:v2.x.x-release	"sh prepare_publish..."	4 months ago	Up 4 months	8880/tcp
02836a049fc	labs.etsi.org:5850/ocf/capif/prod/ocf-events-api:v2.x.x-release	"sh prepare_events.sh"	4 months ago	Up 4 months	8880/tcp
06c2300a7f7	services-capif-events-1				
06c2300a7f7	services-celery-beat	"/celery/start_celen..."	4 months ago	Up 4 months	
ac15f8e71196	labs.etsi.org:5850/ocf/capif/prod/ocf-auditing-api:v2.x.x-release	"sh prepare_audit.sh"	4 months ago	Up 4 months	8880/tcp
061929497893	labs.etsi.org:5850/ocf/capif/prod/ocf-mongo-express:1.0.0-alpha.4	"tini -- /docker-ent..."	4 months ago	Up 5 weeks	0.0.0.0:8082->8081/tcp, [::]:8082->8081/tcp
0609af1c3569	labs.etsi.org:5850/ocf/capif/prod/ocf-discover-service-api:v2.x.x-release	"sh prepare_discover..."	4 months ago	Up 4 months	8880/tcp
2b191677373b	labs.etsi.org:5850/ocf/capif/prod/ocf-logging-api-invocation-api:v2.x.x-release	"sh prepare_logging..."	4 months ago	Up 4 months	8080/tcp
dae0f954e10e	labs.etsi.org:5850/ocf/capif/prod/ocf-routing-info-api:v2.x.x-release	"sh prepare_routing..."	4 months ago	Up 4 months	8080/tcp
072e1cc7e0d7	labs.etsi.org:5850/ocf/capif/mongo:6.0.2	"docker-entrypoint.s..."	4 months ago	Up 5 weeks	27017/tcp
f1722c0e2684	labs.etsi.org:5850/ocf/capif/redis:7.4.2-alpine	"docker-entrypoint.s..."	4 months ago	Up 4 months	0.0.0.0:6379->6379/tcp, [::]:6379->6379/tcp
0b41d8add7ea	labs.etsi.org:5850/ocf/capif/prod/vault:v2.x.x-release	"docker-entrypoint.s..."	4 months ago	Up 4 months	0.0.0.0:8200->8200/tcp, [::]:8200->8200/tcp
	services-vault-1				

Figure 11 CAPIF deployment instance

3.4.2 N-Apps under Development for Phase 2

3.4.2.1 N-App Coverage Extension

The proposed N-App for Coverage extension (N-App Coverage Ext.) is designed to monitor and enhance connectivity in Public Protection and Disaster Relief (PPDR) scenarios by leveraging both conventional cellular connectivity (Uu) and device to device connectivity based on NR PC5 sidelink technology. The goal is to facilitate a UAV-based relay support in a semi-automated way. In principle, the N-App continuously collects signal quality metrics from the UEs and location-related information (e.g., GNSS metrics) to enable spatial awareness of the network conditions. Two operational scenarios should be supported through a combination of proactive (server-side) and reactive (client-side) mechanisms.

Proactive Network-Assisted Operation (N-App Coverage Ext. server)

Based on the collected measurements, the N-App performs policy-driven decision making to maintain service continuity before connectivity degradation leads to service disruption. A typical proactive policy is defined as follows:

- If the Uu signal quality is proactively identified as weakening (e.g., will eventually be unable to support the connection) the N-App:
 1. Identifies the affected geographic area using GNSS data
 2. Triggers notification messages, guiding the movement (i.e., via a UAV human operator) of the UAV relay to that location
 3. Enables PC5 sidelink connectivity between nearby devices

This proactive behavior ensures that coverage gaps are anticipated since the sidelink communication is pre-established. It should be noted that the ultimate goal is full autonomy, where UAV positioning is performed

automatically by the system. However, for trial and validation purposes, the N-App will initially generate notification messages indicating the target UAV location. These notifications will be sent to a UAV controller, and the UAV will be manually deployed accordingly.

Reactive UE-Assisted Operation (N-App Coverage Ext. Client)

In addition to the network-side procedure, a lightweight N-App client is deployed at the UE (as shown in Figure 12) to provide reactive resilience. If a UE loses Uu connectivity unexpectedly, and no prior sidelink configuration has been triggered by the N-App:

- The N-App client locally detects the loss of connectivity
- It autonomously initiates the sidelink (PC5) discovery and resource allocation procedure
- The UE attempts to connect to nearby devices (e.g., UAV relay node)

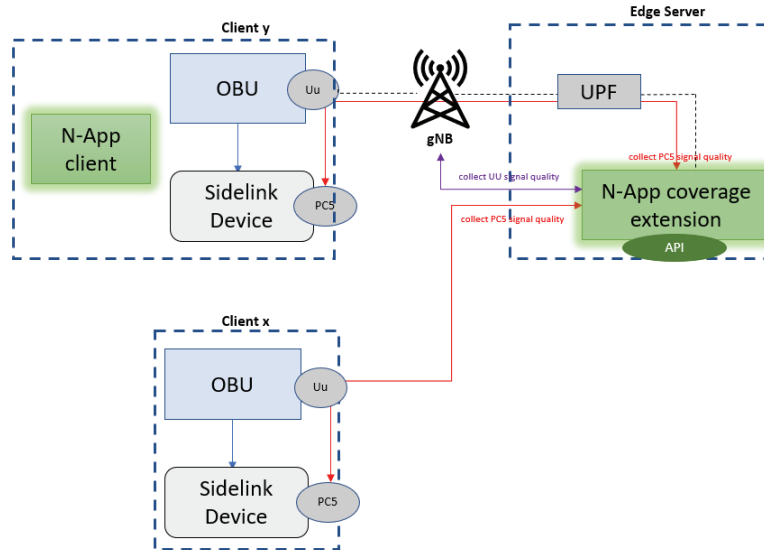


Figure 12 N-App Coverage Extension

The N-App Coverage Ext., is undergoing a development stage with an initial prototype having been developed following the N-App-template workflow, and is currently hosted in the project's [GitLab](#) repository. It will be used and tested during the second phase of trials of the Greek Pilot planned for 2027, and its final stable version will be documented in D4.2.

3.4.2.2 N-App for Energy Efficiency as a Service

The N-App for Energy Efficiency as a Service (N-App EEaaS) provides a framework for monitoring, aggregating, and exposing energy consumption metrics of distributed infrastructure components (e.g., UEs, edge applications) in a 5G environment. It enables both real-time observability and API-based energy footprint reporting.

N-App EEaaS consists of the following main components:

1. Per-Target Metrics Collector (**N-App EEaaS Client**)
 - a. Deployed on each target device (e.g., OBU, UAV)
 - b. Continuously scrapes energy consumption metrics from the underlying hardware
 - c. Metrics may include:
 - i. Power consumption (Joules or kWh)
 - ii. Device-specific telemetry (CPU, GPU, radio usage etc.)
2. Per-Target Metrics Aggregator (Local Database)
 - a. Each N-App EEaaS client stores collected metrics in a local database
3. Centralized Metrics Collector – Prometheus (N-App server)
 - a. N-App EEaaS clients periodically push metrics to a centralized Prometheus instance which in turn provides time-series storage and aggregation across multiple devices and applications

The N-App exposes an Energy Efficiency as a Service API through the EaaS API, aligned with the CAMARA Energy Footprint Notification (EFN) API. The CAMARA EFN API is a computing-services API that allows an API consumer (e.g., V-App or AI-App) to obtain the end-to-end energy consumption of a service delivered across multiple application instances running on a telco infrastructure. Beyond the CAMARA specification, the proposed N-App extends this functionality by enabling the collection and exposure of hardware-level energy metrics. In particular, it provides fine-grained energy consumption measurements at the device level (e.g., OBU, UAV). This allows capturing the full energy footprint of the system, including contributions from communication interfaces (e.g., 5G modem, PC5 sidelink), embedded processing units, and platform-specific hardware components, thus offering a more comprehensive and realistic view of the energy usage. The N-App EEaaS, currently undergoes the development stage, with an initial prototype having been developed following the napp-template workflow and is currently hosted in the project's GitLab repository. It is planned to be used and tested during second phase of trials of the Greek Pilot, planned for 2027, thus the final and stable version of N-App EEaaS will be documented in D4.2.

3.4.2.3 N-App for QoS

The N-App for QoS currently undergoes the final development phase and utilises the AsSessionWithQoS API in order to establish and manage application sessions with specific Quality of Service (QoS) according to 3GPP standards. The V-App WF in the Greek use case will specify the required QoS level by indicating a 5QI (5G QoS Identifier), and additional policies such as bandwidth limits ([3GPP TS 29.122, subclause 5.14.1](#)). Through RESTful API the AS sessions with required QoS will be managed. The session setup and policy enforcement are implemented via the PCF's Npcf_PolicyAuthorization standard call, with the NEF mediating the request to the PCF of Open5GS.

3.5 French Cluster

For Phase 1 (Trials 1 and 2), the cluster will showcase a limited set of mature N-Apps that are sufficiently developed and integrated. The N-App NEF+CAPIF will be demonstrated to enable exposure of network capabilities to applications, and the N-App RAN/AI-App Interface will be used to support interaction between the RAN and AI-based applications. Both N-Apps have reached a high level of readiness and will be actively used during the Phase 1 trials. For Phase 2 (Trials 3), additional N-App functionalities are planned. The N-App NWDAF+CAPIF will be introduced to provide analytics-driven network insights and support intelligent decision-making. In addition, further enhancements of the N-App NEF+CAPIF and N-App RAN/AI-App Interface are expected, extending their capabilities and integration within the overall system. These developments are currently ongoing and will be progressively validated in Phase 2 trials. A list of the French N-Apps are visible in Table 7.

Table 7 Implemented and under development N-Apps of the French cluster

Cluster Name	App Name	Phase (1or 2)
French	N-App NEF+CAPIF	1
French	N-App NWDAF+CAPIF	2
French	N-App RAN/AI-App interface	1

3.5.1 N-App NEF+CAPIF

3.5.1.1 Overview

The N-App NEF+CAPIF acts as the primary exposure gateway between the 5G Core Network and external applications within the French cluster. It enables event-driven monitoring by exposing network capabilities such as UE reachability, connectivity status, and location information. Through its integration with CAPIF, it provides a secure and standardized interface for V-App B5G Connector to subscribe to network events and receive real-time notifications. This N-App plays a central role in enabling cross-layer interaction between the 5G Core and application layers, ensuring that applications remain aware of network conditions and can react accordingly.

3.5.1.2 Architecture

The architecture of the N-App follows the 3GPP Service-Based Architecture (SBA). It is implemented as an Application Function (AF) interfacing with the NEF and exposing its services through the CAPIF framework. The CAPIF invoker and provider components ensure secure service exposure and consumption. A shared CAPIF invoker is used by V-Apps, enabling efficient and unified access to network services. The architecture supports an event-driven model where applications subscribe to specific network events and receive asynchronous notifications. Figure 13 shows the followed architecture.

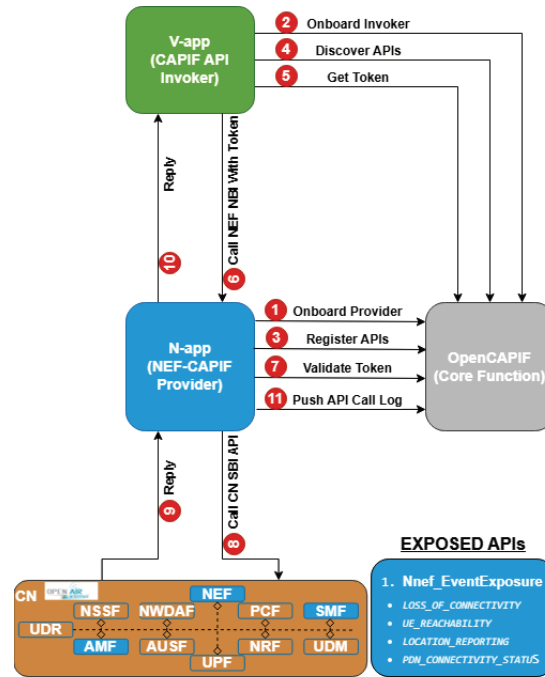


Figure 13 N-App NEF+CAPIF Architecture

3.5.1.3 Analysis and development of 5G APIs

The N-App implements NEF-based APIs to expose monitoring services. These APIs are aligned with 3GPP specifications and are used to provide event notifications to external applications (see Table 8).

Table 8 API Methods of the N-App

REST Method	Resource URI	Body Parameters	Description
POST	/subscriptions	UE ID, event type, callback URL	Creates a monitoring subscription for a UE via the NEF. Supports events such as LOCATION_REPORTING, UE_REACHABILITY, and PDN_CONNECTIVITY_STATUS.
GET	/events	UE ID	Retrieves monitoring event notifications received from the NEF for a specific UE.
DELETE	/subscriptions/{id}	N/A	Deletes an existing monitoring subscription using its identifier.

The APIs are developed to support event-driven communication and ensure real-time delivery of network information.

3.5.1.4 Integration of APIs with OpenCAPIF

The N-App is fully integrated with OpenCAPIF, which acts as the central API framework. The integration includes the registration of the N-App as a CAPIF provider, enabling secure service exposure. The CAPIF invoker is used by external applications to discover and access the NEF services. Authentication and authorization are handled using OAuth2, ensuring that only authorized applications can access sensitive network data. This integration enables seamless interoperability between the 5G Core and application layers.

3.5.1.5 Development, Deployment and Results

The N-App is implemented in Go, chosen for its efficiency and concurrency support. It integrates with the OpenAirInterface (OAI) 5G Core Network and supports CAPIF onboarding, service discovery, authentication, and authorization mechanisms. The application subscribes to NEF monitoring events and processes real-time notifications related to UE status. It has been fully validated in the EURECOM testbed and is currently undergoing end-to-end integration with V-Apps.

The development of this N-App follows a modular and scalable design, aligned with 3GPP standards and the OAI implementation. It supports concurrent device monitoring, real-time event processing, and secure API exposure. The N-App is designed to be extensible, allowing the addition of new monitoring capabilities and integration with other network functions in future iterations. The N-App is deployed as a containerized application using Docker within the EURECOM testbed infrastructure. It runs alongside CAPIF components (Core, Provider, and Invoker) and interacts with the 5G Standalone Core Network. The containerized approach ensures portability, scalability, and ease of deployment across different environments. The N-App has been successfully validated in the laboratory environment. It demonstrates the ability to monitor UE events such as reachability and connectivity in real time. End-to-end validation with V-Apps is ongoing, with successful data exchange already observed between the 5G Core and application layers. The system provides reliable and low-latency event delivery, confirming its readiness for Phase 1 trials. **Error! Reference source not found.** shows the logs of the test procedures within the French testbed.

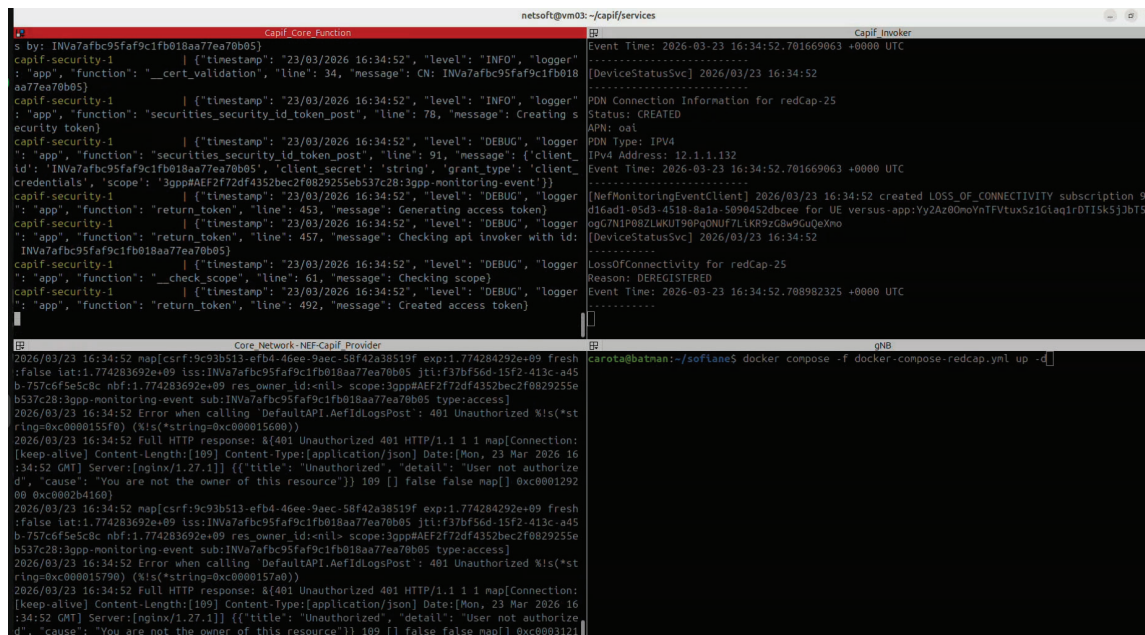


Figure 14 N-App NEF+CAPIF test of the French testbed on real RedCap UE and 5G SA.

3.5.2 N-App RAN/AI-App interface

3.5.2.1 Overview

The third N-App is specifically designed to interface the Radio Access Network (RAN) with the AI-App layer. This application facilitates a critical closed-loop control mechanism between the radio environment and intelligence-driven optimization tools

3.5.2.2 Architecture

As shown in Figure 15, it enables a bidirectional data pipeline: RAN performance measurements from the RAN to the AI-App for processing to select the optimal RAN configuration (DRX configuration) and control parameters flow from the AI-App back to the RAN to enhance energy efficiency while maintaining connectivity quality. The interface between the network and the AI-App uses a WebSocket pipeline and is processed in real-time.

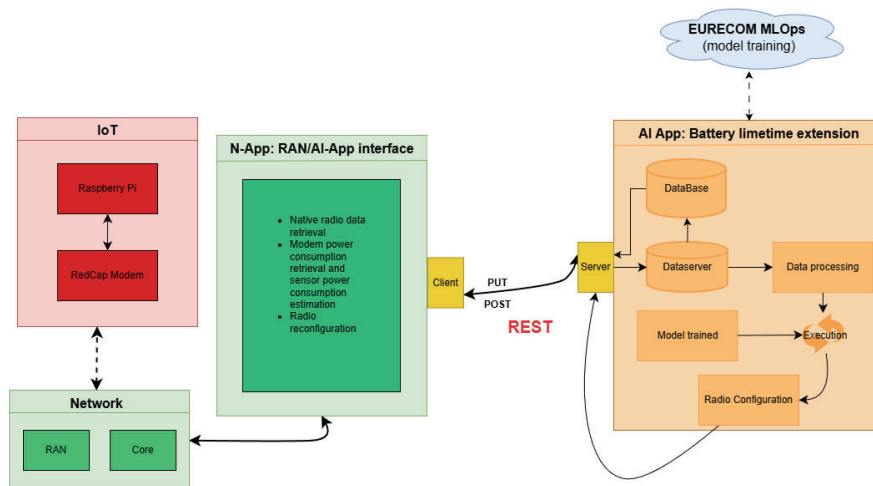


Figure 15 N-App 3 (RAN/AI-App Interface) Architecture for the French Cluster

3.5.2.3 Analysis and development of 5G APIs

The N-App is designed to be connected to the Network (especially the RAN) and the AI-App.

The N-App is fully developed and consists of an API that interconnects the RAN to the AI-App. The protocol used is a WebSocket as defined in RFC 6455 ([RFC 6455 - The WebSocket Protocol](#)). The messages between the client and the radio server are in strict JSON format.

Between the N-App and the RAN, the N-App requests some information about the network, thanks to a WebSocket that is sent to the RAN. The messages to send to the RAN are “qos_flow_get” (per flow measurements) and “ue_get” (per UE radio measures such as Channel Quality Indicator, Modulation and Coding Scheme, SNR) and are specific to the RAN that is used. This App can also send to the RAN a new configuration by modifying some RAN parameters in order to update the RAN configuration. For instance, the message “edrx” can reconfigure the DRX parameters of the RAN.

For the HTTP requests, the N-App acts as a client and the AI-App acts as a server. A ‘POST/PUT’ request is made to send the data, including radio measures, to the AI-App and store it into the dataserver. Moreover, a new RAN configuration is pushed to the network.

3.5.2.4 Integration of APIs with OpenCAPIF

As the only purpose of this N-App is to send the data collected from the RAN to the AI-App, and inversely to send to the RAN a new configuration, the OpenCAPIF framework has not been used. This N-App is autonomous and is based on a remote API developed by the RAN stack that is used.

3.5.2.5 Development, Deployment and Results

This N-App is fully operational and integrated into the French cluster. It is deployed as a containerized application, using Docker. The N-app has been tested in laboratory conditions. As expected, the collection of radio measurements is done in real-time. A request can be sent through this N-App in less than 100 milliseconds to monitor the UE signal quality depending on the current RAN configuration. On the other side, the request of pushing a new configuration to the RAN can be made in less than 1 second, which is largely sufficient.

```
{ "message": "qos_flow_get", "qos_flow_list": [{"pdu_session_id": 1, "dl_total_bytes": 1120638, "ul_total_bytes": 16248584, "sst": 1, "ran_ue_id": 5, "qfi_list": [{"qfi": 1, "sqi": 6}]}], "timestamp": 15733809, "time": 15733.809, "utc": 1777907680.316}
```

Figure 16 N-App “qos_flow_get” response

```
{ "message": "ue_get", "ue_list": [{"ran_ue_id": 5, "amf_ue_id": 102, "qos_flow_list": [{"pdu_session_id": 1, "dl_total_bytes": 1118350, "ul_total_bytes": 16111993, "sst": 1}], "rnti": 17925, "cells": [{"cell_id": 1, "dl_bitrate": 13440, "ul_bitrate": 273858, "dl_tx": 5749, "ul_tx": 5159, "dl_err": 0, "ul_err": 0, "dl_retx": 521, "ul_retx": 457, "pusch_snr": 14.4, "epre": -99.2, "ul_phr": -2, "ul_path_loss": 87.9, "p_ue": 19, "ul_rank": 1, "ue_ul_bwp_id": 0, "ul_bwp_id": 0, "dl_mcs": 25.8, "ul_mcs": 14.3, "ul_n_layer": 1, "turbo_decoder_min": 1, "turbo_decoder_avg": 3.822, "turbo_decoder_max": 5, "cqi": 15, "ri": 1, "ue_dl_bwp_id": 0, "dl_bwp_id": 0, "initial_ta": 3}]}], "time": 15733.681, "utc": 1777907680.197}
```

Figure 17 N-App “ue_get” response

Figure 16 and figure 17 are native radio measures collected from the RAN. They are essential to train the AI model and also essential for the inference. The answers of those requests take less than 10 milliseconds to arrive to the N-App.

3.5.3 N-Apps under Development for Phase 2

The N-App NWDAF+CAPIF is currently under active development and will be introduced in Phase 2 (Trials 3). This N-App will provide advanced analytics capabilities based on network data, enabling intelligent decision-making across the system. It will integrate tightly with the ML-Ops framework deployed in the French testbed, enabling the full lifecycle management of machine learning models, including training, validation, deployment, and monitoring. Through CAPIF, the N-App will expose analytics services to V-Apps, enabling them to consume network intelligence in a secure and standardized manner. This N-App will support use cases such as anomaly detection, and traffic prediction. It will process both real-time and historical network data, allowing the system to move from reactive monitoring to proactive and predictive network management. Ongoing development efforts focus on improving model accuracy, scalability, and integration with the existing 5G Core and application ecosystem.

3.6 Portuguese Cluster

For Phase 1 (Trials 1, 2 and 3), the cluster will showcase the QoS N-App (Scenario 1) and the Network Operations N-App (Scenario 2). The QoS N-App handles requests to the Network Core for changing multiple QoS parameters and measurement configurations of connected devices. The Network Operations N-App handles the dynamic reconfiguration of the network (e.g. allocation and deallocation of Core and RAN network resources, and energy-related enablement), as a result of specific events. For Phase 2 (Trials 4, 5, 6, and 7) the Cluster will add two N-Apps:

- the Location N-App (Scenario 3), for to retrieving, processing, and delivering accurate location information for authorized services and users.
- The Traffic Influence N-App (Scenario 1, 2), for influencing UE’s traffic routing.

Table 9 Implemented and under development N-Apps in the Portuguese cluster

Cluster Name	App Name	Phase (1 or 2)
--------------	----------	----------------

Portuguese	QoS N-App	Phase 1
Portuguese	Network Operations N-App	Phase 1
Portuguese	Location N-App	Phase 2
Portuguese	Traffic Influence N-App	Phase 2

3.6.1 QoS N-App

3.6.1.1 Overview

The QoS N-App functions as an abstraction layer for managing requests to the Core network, simplifying interactions with the AI-App and V-App. It supports ONEmNEF capabilities (e.g., QoS management, event monitoring, traffic influence, and analytics exposure) depending on the underlying Core implementation and incorporates CAMARA APIs were aligned with the planned feature set. Additionally, it provides predefined energy profiles for IoT devices, enabling dynamic adjustment of QoS parameters through the 3GPP AsSessionWithQoS API via the NEF.

3.6.1.2 Architecture

The QoS N-App iterates over the N-App template by automating the CAPIF invoking and onboarding and QoS request handling.

presents the QoS N-App architecture inside the scenario 1, while the flow-char for requesting a QoS profile can be seen in Figure 19.

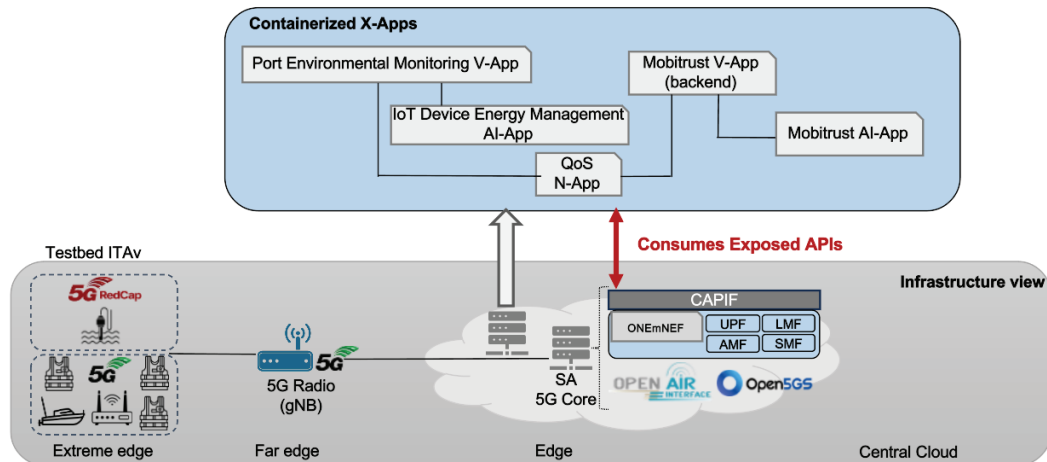
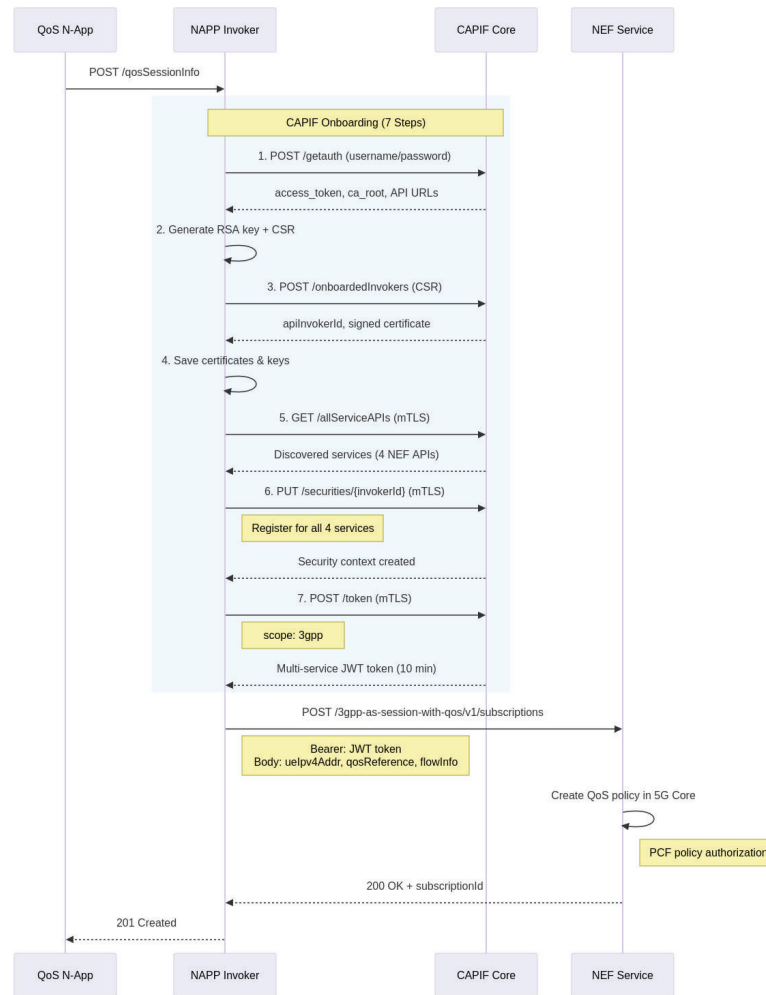


Figure 18 QoS N-App architecture**Figure 19 QoS N-App basic workflow**

3.6.1.3 Analysis and development of 5G APIs

Table 10 QoS N-App API Methods

REST Method	Resource URI	Body Parameter	Description
GET	/3gpp-as-session-with-qos/v1/{scs_as_id}/subscriptions		Query QoS session information from NEF. (open5gs and OAI)
POST	/3gpp-as-session-with-qos/v1/{scs_as_id}/subscriptions	Follows 3GPP TS 29.122 payload	Create QoS session with direct 3GPP AsSessionWithQoS request. (open5gs and OAI)
GET	3gpp-monitoring-event/v1/{scs_as_id}/subscriptions		Query existing monitoring subscriptions from

			NEF. (open5gs and OAI)
POST	3gpp-monitoring-event/v1/{scs_as_id}/subscriptions	Follows 3GPP TS 29.122 payload	Create a new monitoring subscription on NEF. (open5gs and OAI)

3.6.1.4 Integration of APIs with OpenCAPIF

The QoS N-App is fully integrated with the OpenCAPIF framework, enabling seamless interaction with CAPIF-compliant services. The processes of onboarding, service discovery, access token acquisition and reuse, as well as token expiration handling, are fully automated. This ensures efficient, secure, and uninterrupted service consumption while minimizing operational overhead and eliminating the need for manual intervention. The QoS N-App was deployed on a virtual machine at the edge of the Aveiro 6G-VERSUS testbed infrastructure, leveraging containerization technologies such as Docker and Kubernetes. Integration was successfully validated with both Open5GS and OAI core networks, as well as with the CAPIF service available within the testbed.

3.6.1.5 Development, Deployment and Results

The QoS N-App is built upon the Python-based 6G-VERSUS template and is fully containerized. Its invoker component is designed for deployment in Kubernetes environments, adhering to cloud-native design principles that improves scalability, resilience, and operational flexibility. The code is available through the 6G-VERSUS GitLab repository: https://gitlab.6g-versus.eu/wp4/napps/napp-template/-/tree/ipn?ref_type=heads

Figure 20 shows the results of conducting an AsSessionWithQoS request from the QoS N-App (top terminal) to the NEF (bottom terminal).


```

INFO: Started server process [461794]
INFO: Waiting for application startup.
INFO: Application startup complete.
INFO: 127.0.0.1:58428 - "GET /profiles/1 HTTP/1.1" 200 OK
2026-05-06:11:17:33,287 INFO [router.py:303] Device characteristics:
{
  "ue_ipv4_address": "10.45.0.31",
  "mac_address": "00:18:44:11:3A:B8",
  "profile_id": 1
}
2026-05-06:11:17:33,287 INFO [router.py:310] Profile characteristics:
{
  "description": "This profile is meant to get a large number of data points quickly. It increases the amount of data collected
, and decreases the time it takes to receive data. However, it uses more energy in the device.",
  "mbr_uplink": 150000,
  "mbr_downlink": 227000,
  "gbr_uplink": 150000,
  "gbr_downlink": 227000,
  "period_uplink": 10,
  "period_downlink": 10,
  "read_period": 1,
  "profile_id": 1
}
2026-05-06:11:17:34,287 INFO [router.py:318] Successfully associated energy profile 1 with device 1.
INFO: 127.0.0.1:58440 - "POST /devices HTTP/1.1" 201 Created

ubComps': {'0': {'fNum': 1, 'fDescs': ['permit out ip from 10.0.0.1 to 10.45.0.31', 'permit in ip from 10.45.0.31 to 10.0.0.1'], 'marBwD
l': '227000 Kbps', 'marBwUL': '150000 Kbps'}}}}}
2026-05-06 10:17:33,367 - SGW - INFO - HTTP2: True - PROTOCOL: http - POST REQUEST URL: http://10.2.0.89:7777/npcf-policyauthorization/v
1/app-sessions
2026-05-06 10:17:33,373 - SGW - INFO -

npcf-policyauthorization/v1/app-sessions

2026-05-06 10:17:33,373 - SGW - INFO - {'ascReqData': {'afAppId': 'napp-test', 'notifUri': 'http://sgw.onemnef.com:8500/npcf-policyautho
rization/v1/notify', 'suppFeat': '1', 'dnn': 'internet', 'ueIpv4': '10.45.0.31', 'sliceInfo': {'sst': 1, 'sd': '000001'}, 'medComponents
': {'1': {'afAppId': 'napp-test', 'medCompN': 1, 'medType': 'VIDEO', 'marBwDL': '227000 Kbps', 'marBwUL': '150000 Kbps', 'mirBwDL': '227
000 Kbps', 'mirBwUL': '150000 Kbps', 'medSubComps': {'0': {'fNum': 1, 'fDescs': ['permit out ip from 10.0.0.1 to 10.45.0.31', 'permit in
ip from 10.45.0.31 to 10.0.0.1'], 'marBwDL': '227000 Kbps', 'marBwUL': '150000 Kbps'}}}}} - DATA
2026-05-06 10:17:33,373 - SGW - INFO - {'Content-Type': 'application/json'} - HEADERS
2026-05-06 10:17:33,373 - SGW - INFO - Request with data and headers
2026-05-06 10:17:33,373 - SGW - INFO - ENDPOINT npcf-policyauthorization/v1/app-sessions

DATA {'ascReqData': {'afAppId': 'napp-test', 'notifUri': 'http://sgw.onemnef.com:8500/npcf-policyauthorization/v1/notify', 'suppFeat':
'1', 'dnn': 'internet', 'ueIpv4': '10.45.0.31', 'sliceInfo': {'sst': 1, 'sd': '000001'}, 'medComponents': {'1': {'afAppId': 'napp-test',
'medCompN': 1, 'medType': 'VIDEO', 'marBwDL': '227000 Kbps', 'marBwUL': '150000 Kbps', 'mirBwDL': '227000 Kbps', 'mirBwUL': '150000 Kbp
s', 'medSubComps': {'0': {'fNum': 1, 'fDescs': ['permit out ip from 10.0.0.1 to 10.45.0.31', 'permit in ip from 10.45.0.31 to 10.0.0.1']
, 'marBwDL': '227000 Kbps', 'marBwUL': '150000 Kbps'}}}}}
2026-05-06 10:17:33,373 - SGW - INFO -

http://10.2.0.89:7777/npcf-policyauthorization/v1/app-sessions

2026-05-06 10:17:33,376 - SGW - DEBUG - RESPONSE <Response [201 Created]>
2026-05-06 10:17:33,377 - SGW - INFO - Publish on topic 'api/sgw/policy-auth'
2026-05-06 10:17:33,377 - SGW - DEBUG - [SGW SEND] message_id=eec222d6-faf3-4ebc-8d5e-8ebc0f07aa0a, type=create_policy-auth, instance_id
=73cd203c-ca9a-4df9-ae2f-8c39fff78aa5, status_code=201, has_header_info=True, has_payload=True
2026-05-06 10:17:33,377 - SGW - INFO - MESSAGE PUBLISHED

```

Figure 20. Logging information of the interaction between the QoS N-App and the NEF.

3.6.2 Network Operations N-App

3.6.2.1 Overview

The Network Operations N-App handles the dynamic reconfiguration of the network (e.g. allocation and deallocation of Core and RAN network resources), as a result of specific events, in order to guarantee the required level of performance while optimising energy efficiency. A typical example of such an event, from scenario 2 of the Portuguese cluster, is the detection of a truck approaching the port premises, which is expected to trigger the allocation of additional network resources required to properly identify the vehicle license plate and the HazMat sign, if any.

3.6.2.2 Architecture

The Network Operations N-App as depicted in Figure 21 provides a mediation layer between the 5G network core and the AI/V-Apps, hiding the complexity of the network infrastructure and handling the interaction with 5G core northbound service APIs, as well as with the CAPIF Core Function module.

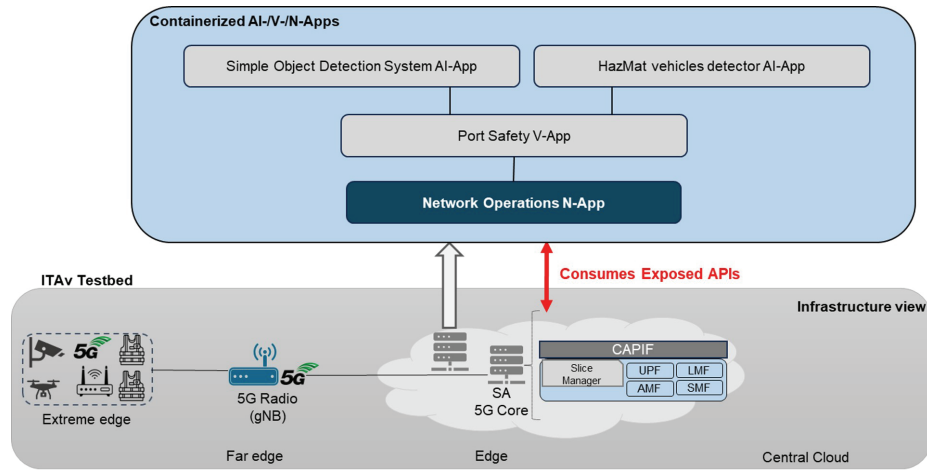


Figure 21 Network Operations N-App architectural view

The sequence diagram in Figure 22 illustrates the basic role of the Network Operations N-App, fundamentally enabling the integration of the Port Safety V-App requests (on the left in Figure 20) with the network capabilities exposed by the Slice Manager APIs through CAPIF (on the right in Figure 20).

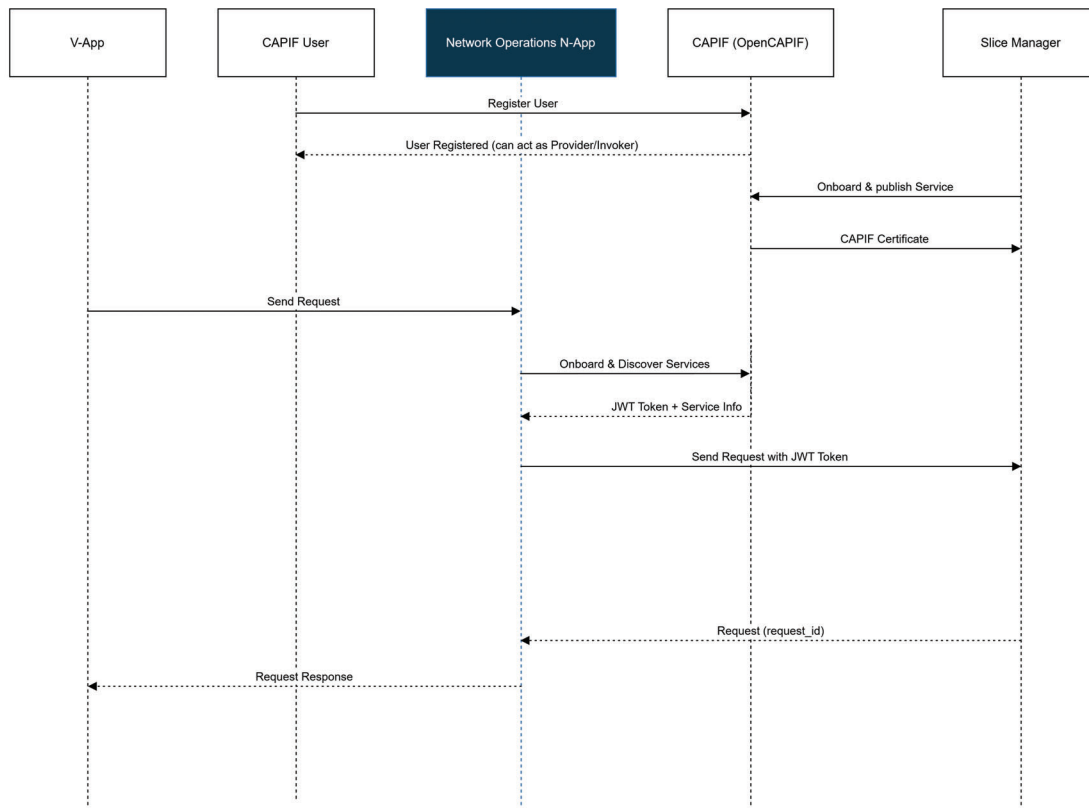


Figure 22 Network Operations N-App basic workflow

The V-App/N-App interworking is supported by a REST API exposed by the N-App, which can be used to execute scaling operations of network resources (Scale Up/Down), as shown in the table below.

Table 11 N-App network operations API methods

REST Method	Resource URI	Body Parameter	Description
POST	/napp/v1/scaling-operation/slice	{ "action": "SCALE_DOWN", "uplinkKbps": 3000, "downlinkKbps": 1000, "notificationDestination": " https://one.one.one.one/ ", "requestId": "b15f1ab7-28c8-48a7-8f20-5c81c6d3bd44", "sliceId": "Nexus" }	Scale Down
POST	/napp/v1/scaling-operation/slice	{ "action": "SCALE_UP", "uplinkKbps": 25000, "downlinkKbps": 1000, "notificationDestination": " https://one.one.one.one/ ", "requestId": "fc63a285-c0c9-4e16-95f3-a7d309f05ad7", "sliceId": "Nexus" }	Scale Up

It should be noted that for phase 1 trials the Network Operations N-App functions include the Handover N-App functions, as described in D2.1. This consolidation makes the east-west interface described in D2.1 redundant.

3.6.2.3 Analysis and development of 5G APIs

The 5G API implemented in the Portuguese cluster is exposed by the Slice Manager network function and consumed by the Network Operations N-App, responsible for handling the dynamic configuration of Core and RAN resources. For 6G-VERSUS phase 1, the set of API endpoints used by the Network Operations N-App is shown in the table below:

Table 12 Set of API endpoints and methods used by the Network Operations N-App

REST Method	Resource URI	Body Parameter	Description
POST	/ran/antenna/activate	{ "antenna_id": "string", "ran": "string" }	Activate an antenna in the RAN
POST	/ran/antenna/deactivate	{ "antenna_id": "string", "ran": "string" }	Deactivate an antenna in the RAN
PUT	/ran/antenna/modify	{ "antenna_id": "string", "modifications": { "MIMO": true, "max_transmit_power": 0 }, "ran": "string" }	Modify parameters of antenna in the RAN (e.g. increase or decrease power)
POST	/core/slice/change	{ "imsi": "string", }	Change the UE to a different core network slice

		<pre>"snssai": "string", "dnn": "string", "dnnqostplid": 0 }</pre>	
--	--	--	--

In scenario 2, the network APIs play a key role by dynamically adapting the consumption of network resources to ongoing events, enabling the allocation of resources, when needed to deliver a specific performance threshold, while guaranteeing the optimization of energy efficiency by disconnecting resources that are no longer required, especially during extended periods of low to moderate activity.

3.6.2.4 Integration of APIs with OpenCAPIF

The Slice Manager is a CAPIF-enabled network function, which means that the interworking with the Network Operations N-App is fully compliant with the CAPIF procedures, including publishing and discovery of the APIs. In addition, the Slice Manager enforces CAPIF-compliant security mechanisms by validating every incoming request through token-based authentication. Specifically, API Invocations are authenticated using access tokens (JWTs) issued by the CAPIF Core Function (CCF), which acts as the authorization server. From a CAPIF perspective, the Network Operations N-App plays the role of CAPIF API invoker. It requests an access token to the CCF using its onboarded identity and credentials, which is subsequently included in each API request. This interaction enables the Port Safety V-App to take advantage, in a secure manner, of network capabilities exposed by the Slice Manager through a northbound service API (described above), following the basic workflow displayed in Figure 19.

3.6.2.5 Development, Deployment and Results

As a key building block of scenario 2 ("Port safety awareness and enhancement via AI and B5G/6G"), the Network Operation N-App was deployed in a VM, at the ITAV data centre, and successfully integrated with the Aveiro 6G-Versus testbed infrastructure, both at ITAV premises and at the Port of Aveiro. The N-App were initially integrated and tested in an isolated environment at ITAV and then integrated with the ITAV indoor trial site testbed infrastructure for preliminary testing and evaluation. A key component of this integration was the interworking with CAPIF. The Network Operations N-App, in charge of handling the interworking between the application triplet and the CAPIF Core Function module, as well as with service APIs exposed by the 5G network, was successfully integrated with OpenCAPIF and Slice Manager, respectively. The integration with the Port of Aveiro 5G network infrastructure was the final deployment step.

The Network Operations N-App enables the dynamic reconfiguration of Core and RAN network resources, required to accomplish the envisioned levels of performance and energy efficiency, which have been successfully demonstrated through trial 2 of the Portuguese Cluster, on 25/3/2026.

The Network Operations N-App is based on the Python 6G-VERSUS template. The current version of the code is available through the 6G-VERSUS GitLab repository: <https://gitlab.6g-versus.eu/wp4/napps/portuguese-cluster>. At the time of writing, the N-App overall objectives for phase 1 can be considered fully accomplished. Plans for the project second phase are currently under study.

3.6.3 N-Apps under Development for Phase 2

First off, the developments will focus on extending the existing N-Apps' capabilities beyond the basic connectivity validation performed in Phase 1, in addition to increase its robustness while operating the different scenario steps. The objective is to evolve N-Apps into more functional components capable of leveraging network exposure features to support advanced application use cases within the port environment. Additionally, a new the ability to obtain UE location, to be developed as a Location N-App, will be developed detailed below. Importantly, an ambition for Phase 2 is to integrate the various cluster-specific N-Apps and progressively converge toward a unified N-App framework. This consolidation aims to streamline N-App functionality and ensure consistent interaction across the cluster's triplet.

3.6.3.1 Location N-App

A N-App able to receive and process location information of user equipment (UEs) from the network will be developed, enabling location-aware services within the infrastructure. The application interacts with core network components and positioning systems to retrieve, process, and deliver accurate location information for authorized services and users. This capability will be particularly relevant for use cases such as smart lighting, where lighting conditions can be dynamically adapted based on the presence and movement of users or devices. Overall, these developments aim to enhance the contextual awareness of the system and support more efficient and responsive operation of port infrastructure services. Figure 23 represents the overall structure of the Portuguese triplet architecture, reflecting the role of the envisioned unified N-App and exposed capabilities.

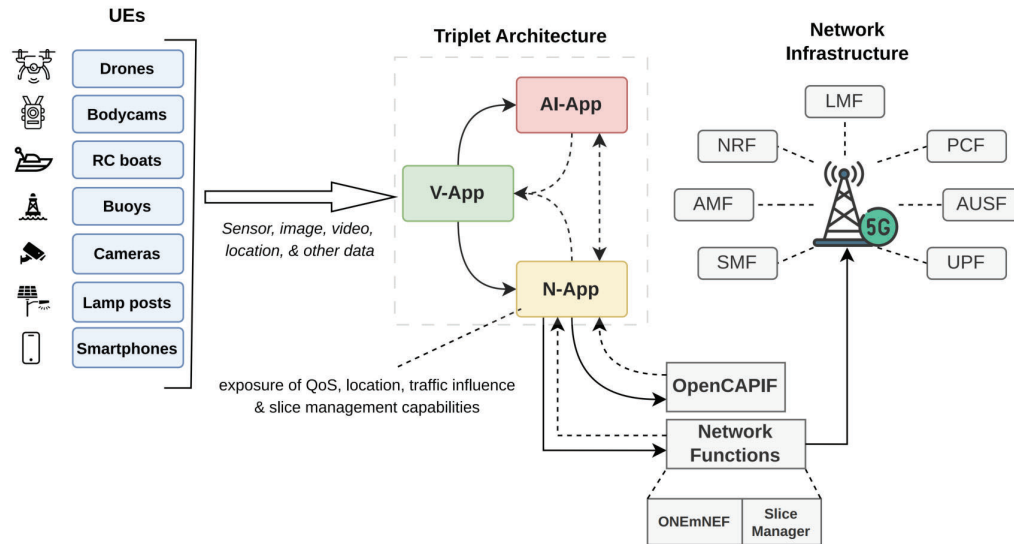


Figure 23 Overview of N-App involvement in the Portuguese Cluster

3.7 Spanish Cluster

The Spanish Cluster includes a set of N-Apps developed and demonstrated across both trial phases. In Phase 1, some N-Apps are introduced as mock-ups or integrated into specific trial scenarios, while Phase 2 expands their role through full implementation and deployment in additional operational contexts. As shown in Table 13, the N-Apps developed within the Spanish Cluster address water quality decision support, multi-connectivity protocols, immersive remote driving, automatic surveillance, and reliable water treatment, highlighting the cluster's progressive contribution to the project's technical validation activities.

Table 13 Implemented and under development N-Apps in the Spanish cluster

Cluster Name	App Name	Phase
Spanish cluster	DSS for water quality N-App	Phase 1 & 2
Spanish cluster	Multi-connectivity protocol N-App	Phase 1 & 2
Spanish cluster	Immersive Remote Driving N-App	Phase 2
Spanish cluster	Automatic surveillance N-App	Phase 2
Spanish cluster	DSS for reliable water treatment N-App	Phase 1 & 2

3.7.1 Decision Support System for water quality N-App

3.7.1.1 Overview

DSS for water quality N-App is a Decision Support System (DSS) network application for water quality monitoring. Its main role is to provide connectivity support for water-quality sensors deployed in the field, ensuring reliable network attachment and stable data transmission across the system. This application will be deployed in La Mayora.

3.7.1.2 Architecture

The architecture of the N-App is centered on its role as the connectivity layer between field-deployed water-quality sensors and the AI-App in the triplet. The N-App directly interfaces with the sensors, manages their network attachment, and ensures stable transmission of measurements toward the AI-App. Conceptually, the N-App acts as an intermediary service between the sensing layer and the AI-based analytics layer, providing communication reliability and basic QoS support.

3.7.1.3 Analysis and development of 5G APIs

DSS for water quality is being developed as a network support component for water-quality monitoring deployments. Its implementation focuses on interfacing with field sensors, supporting their connectivity to the communication network, and ensuring reliable transmission of sensor measurements to the AI-App (see Table 14). From a software perspective, the application is intended to implement functions for sensor communication handling, connectivity supervision, QoS observation, and data forwarding. The application is designed to be deployable as a containerized service, facilitating portability and integration with the rest of the triplet components.

Table 14 Set of API endpoints and methods used by DSS for water quality N-App

REST Method	Resource URI	Body Parameter	Description
POST	/POST_AsSessionWithQoSSubscription	suppFeat, notifUri, ipAddress, medCompN, 5qi, medSubComps (fNum, marBwDl, marBwUl, mirBwDl, mirBwUl)	Allows an AF to create or update an AF session with specific QoS requirements. The AF can request QoS parameters such as 5QI and bandwidth limits (maximum and minimum, uplink and downlink) for one or more media flows associated with a UE IP address, enabling fine-grained QoS control for the service.
POST	/POST_IP2IMSI	ipAddress	Allow the Application Function to map and identify the IMSI using an IPv4 address, enabling correct user identification within the network.
POST	/POST_IMSI2IP	supi	Enable the Application Function to map the IPv4 address based on the provided SUPI, supporting

			IP-level user identification.
--	--	--	-------------------------------

3.7.1.4 Integration of APIs with OpenCAPIF

The integration of the application with OpenCAPIF is considered to expose selected application interfaces in a secure and standardized manner. In particular, OpenCAPIF is going to be used for authentication and sensors discovery.

3.7.1.5 Development, Deployment and Results

Beyond the current conceptual definition, the DSS for water quality N-App has been implemented in Python as the connectivity support component of the water-quality monitoring triplet. The N-App provides reliable network attachment for field-deployed water-quality sensors, supports stable transmission of measurements, and helps preserve end-to-end data continuity toward the AI-App. The next phase of development will therefore focus on:

- connecting the water quality sensors in La Mayora, once installed, to the Internet through this application.
- packaging the solution as a containerized application suitable for deployment in the final testbed architecture.

The containerized deployment approach supports portability, modular integration, and simplified lifecycle management across different environments. Any future 5G API exposure and OpenCAPIF integration will also be deployed as part of this containerized environment. At the current stage, the results available for the application are limited to the conceptual and architectural definition of the application, as the sensors for water quality have not been deployed in the field yet.

3.7.2 Multi-connectivity protocol N-App

3.7.2.1 Overview

The multi-connectivity protocol N-App is responsible for managing different communication paths efficiently. The N-App is based on a multi-connectivity protocol called Multi-connection Tactile Internet Protocol (MTIP), tailored for low-latency and high-reliability communications, enabling the coordinated use of multiple available paths to enhance robustness and end-to-end performance. In addition, the N-App is designed to leverage the HPE 5GC API in order to interact with the 5G Core, for example: to retrieve the IP address of connected devices using the SUPI; to limit the bandwidth or to request a specific 5QI for the remote control of the Unmanned Vehicle (UV) and its payload. The N-App directly supports the associated V-App by exposing multiple paths to it, while the N-App remains in charge of path management and networking decisions.

3.7.2.2 Architecture

The architecture of the Multi-connectivity protocol N-App is depicted in Figure 24. It is focused on a path-management component that controls multiple available communication paths in order to improve service performance for the associated teleoperation application. The N-App can be deployed in an emulated environment, where a Linux host runs Mininet to provide multiple paths, or in a real unmanned vehicle platform where multiple modems create distinct paths. In the current implementation, the N-App focuses on the multi-connectivity protocol itself. In the next development iteration planned for Trial 2, the N-App will be extended with the project N-App template to support interaction with the HPE 5GC API through OpenCAPIF, so that the N-App can request information and differentiated treatment from the HPE 5G Core.

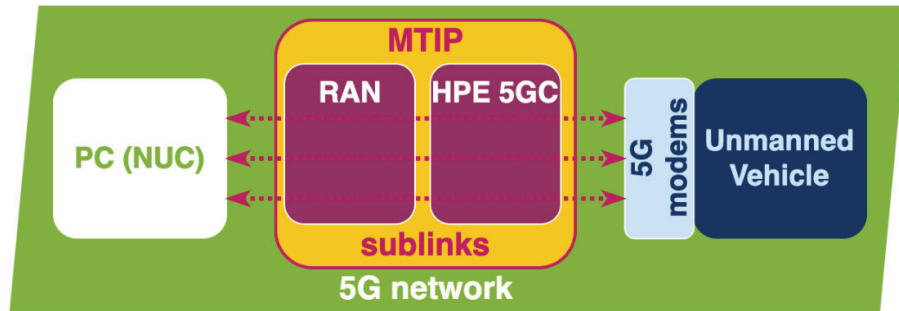


Figure 24 Multi-connectivity protocol N-App architecture

3.7.2.3 Analysis and development of 5G APIs

The Multi-connectivity protocol N-App is intended to leverage 5G Core APIs provided by HPE in order to complement the protocol-level path management with network-aware control functions. In particular, the planned API usage includes: (i) retrieving the IP address of connected devices from their IMSI; (ii) limitation of the bandwidth; and (iii) requesting a suitable 5QI for unmanned vehicle remote-control traffic. These capabilities are relevant to the N-App because they enable the application to identify the target network endpoints and request network treatment better suited to teleoperation traffic.

At this stage, the HPE API is already implemented and available, and the exact methods, resource URIs, and body parameters have now been selected for use in the N-App, which are presented in Table 15. Their selection and integration have been completed together with the adoption of the project N-App template.

Table 15 Set of API endpoints and methods used by Multi-connectivity protocol N-App

REST Method	Resource URI	Body Parameter	Description
POST	/POST_AsSessionWithQoSSubscription	suppFeat, notifUri, ipAddress, medCompN, 5qi, medSubComps (fNum, marBwDl, marBwUl, mirBwDl, mirBwUl)	Allows an AF to create or update an AF session with specific QoS requirements. The AF can request QoS parameters such as 5QI and bandwidth limits (maximum and minimum, uplink and downlink) for one or more media flows associated with a UE IP address, enabling fine-grained QoS control for the service.
POST	/POST_AF_Traffic_Influence	afServiceIdentifier, anyUEIndication, gpsi, dnn, sNssai (sst, sd)	Allows an Application Function (AF) to influence user traffic by selecting or changing the target DNN and/or network slice (sNssai) for one or more UEs identified by GPSI, exposed via the NEF to provide controlled and secure access toward the 5G Core. The request can be applied to specific UEs or to all UEs, and is used to steer traffic toward a specific data network or slice according to service needs.
POST	/POST_IP2IMSI	ipAddress	Allow the Application Function to map and identify the IMSI using an IPv4 address, enabling correct user identification within the network.

REST Method	Resource URI	Body Parameter	Description
POST	/POST_IMSI2IP	supi	Enable the Application Function to map the IPv4 address based on the provided SUPI, supporting IP-level user identification.

3.7.2.4 Integration of APIs with OpenCAPIF

The integration with OpenCAPIF is planned as the mechanism through which the N-App will consume the relevant HPE 5GC APIs. In the project testbed, OpenCAPIF is already deployed and integrated, and the HPE API is used through OpenCAPIF, with the corresponding invoker and provider already onboarded. This provides the required basis for Trial 2, where the N-App will incorporate the project N-App template and act as the API invoker for the selected HPE 5GC functions. The OpenCAPIF onboarding and discovery workflow is aligned with the CAPIF model defined for API invokers and providers. Beyond the current protocol implementation, the development roadmap of the Multi-connectivity protocol N-App aims to evolve the existing C++ implementation into a fully integrated N-App capable of combining multi-connectivity management, 5G API consumption, and OpenCAPIF-based API interaction.

The next phase will therefore focus on:

1. integrating the project N-App template
2. defining the northbound interface between the V-App and the N-App
3. packaging the solution as a containerized application suitable for the final testbed architecture.

The Multi-connectivity protocol N-App is currently deployed manually in the testbed environment. The target deployment foresees that the N-App will be distributed across a remote Linux PC, associated with the remote-control side of the application, and an onboard Linux machine associated with the unmanned vehicle platform. This deployment model is consistent with the distributed nature of the use case and with the need to manage multiple communication paths close to both endpoints.

3.7.2.5 Development, Deployment and Results

At present, the N-App has been deployed and tested in an emulated scenario, where multiple paths are reproduced in a Linux-based setup. The current deployment is not yet containerized. However, containerization using Docker is planned for the next implementation stage in Trial 2. The same future deployment stage is expected to include the integrated N-App template, the selected HPE 5GC API functions, and the OpenCAPIF-based API invocation workflow.

The current results demonstrate that the Multi-connectivity protocol N-App has been successfully deployed and tested in an emulated scenario, validating the operation of the core protocol in a multi-path environment. In addition, the solution was demonstrated during Trial 1, which took place on 2 February 2026 at the Ada Byron building (scenario 1). At the current stage, the demonstrated functionality corresponds to the core multi-connectivity protocol implementation. The integration of the project N-App template, the integration with OpenCAPIF for API discovery and authorization, and the invocation of selected HPE 5GC APIs, together with the final containerized deployment, are planned for Trial 2. Figure 24 shows a screenshot of the Multi-connectivity N-App showcased in Trial 1. In the screenshot, an emulated drone is being controlled and the traffic is being captured to show the MTIP protocol in use.

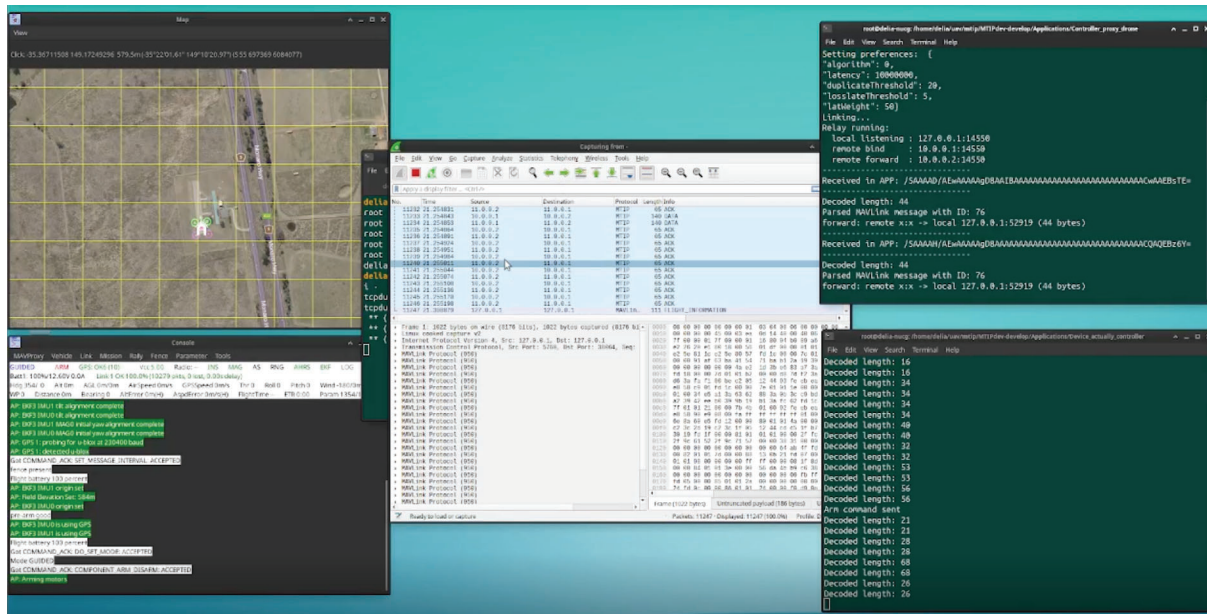


Figure 25 Multi-connectivity protocol N-App showcased in Trial 1.

The Multi-connectivity protocol N-App that controls different MTIP links has been implemented in C++ and is already available in the project GitLab repository:

<https://gitlab.6g-versus.eu/wp4/napps/spanish-cluster/multi-connectivity-protocol-n-app>

The current implementation focuses on the core multi-connectivity protocol, whose purpose is to manage different available communication paths efficiently in order to improve the performance of low-latency and high-reliability traffic. The software has been developed for Linux-based environments and supports execution both in an emulated scenario, where multiple paths are reproduced using Mininet, and in future real deployments on remote and onboard Linux machines.

At the current stage, the available repository contains the core protocol interaction implementation only. In the next development step, planned for Trial 2, the N-App will be extended with the project N-App template. The current deployment is performed manually. Containerization is planned for the next iteration, where the N-App is expected to be packaged as a Docker partially, in alignment with the final testbed architecture.

3.7.3 Immersive driving of an AGV with autonomous missions N-App

3.7.3.1 Overview

This N-App provides network-level support for the immersive remote driving use case by enabling dynamic discovery of network endpoints associated with connected devices. Specifically, it allows client applications to retrieve the IP address of a device based on its IMSI identifier. The N-App exposes a real-time communication interface that ensures continuous availability and low-latency responses. By leveraging 5G Core capabilities through CAPIF and Network as Code (NaC) functions, the N-App enables seamless integration between network services and application-layer components.

3.7.3.2 Architecture

The N-App is designed as a network-facing application that provides dynamic endpoint discovery services through a WebSocket-based interface. Client applications connect to the N-App via a persistent WebSocket channel, through which IMSI-based queries are submitted. The N-App processes these requests asynchronously and interacts with the 5G Core network to retrieve the corresponding IP address and session information. The interaction with the 5G Core is achieved through CAPIF, invoking Network as Code (NaC) functions. These functions rely on an ad-hoc API exposed within the Session Management Function (SMF) of

the Open5GS core. This architecture enables secure, real-time access to network information and ensures seamless integration between application and network layers. The design follows the N-App paradigm and is aligned with CAPIF-based service exposure mechanisms.

3.7.3.3 Analysis and development of 5G APIs

The N-App utilizes a custom 5G API (Table 16) implemented within the Open5GS Session Management Function (SMF) to retrieve PDU session information associated with a given IMSI. This API is exposed through a lightweight HTTP server integrated into the SMF.

The implementation includes a microhttpd daemon and an HTTP handler that processes GET requests to the `"/pdu-sessions"` endpoint. The handler accesses internal SMF data structures (`"smf_self()->smf_ue_list"`, `"ue->sess_list"`) to retrieve session information and supports filtering by IMSI through query parameters. The response is formatted as JSON using the JSON library and includes fields such as IMSI, PDU session ID, IP address (IPv4/IPv6), SST, and SD.

The API depends on the Open5GS core and requires a minor modification of the SMF source code. Configuration parameters such as binding address and port are controlled via environment variables (`"OGS_PDUAPI_ADDR"`, `"OGS_PDUAPI_PORT"`).

Table 16 Custom 5G API methods

REST Method	Resource URI	Body Parameter	Description
GET	/pdu-sessions	IMSI (query)	Retrieves PDU session information and IP address associated with a given IMSI

For the remote immersive driving use case, a dedicated API has been developed within the N-App to enable dynamic discovery of network endpoints associated with connected devices. To achieve this, the N-App communicates with the 5G Core network by invoking Network as Code (NaC) functions via CAPIF. These NaC functions rely on an ad-hoc API exposed in the Session Management Function (SMF) process of the Open5GS core, requiring a simple modification to its source code. This modification consists of a microhttpd daemon and an HTTP handler that handles GET requests for `"/pdu-sessions"`. It collects PDU sessions from internal structures (`"smf_self()->smf_ue_list"`, `"ue->sess_list"`), supports IMSI filtering via query string, extracts fields (IMSI, pdu_session_id, dnn, IPv4/IPv6 addresses, sst, and sd), and constructs the JSON response using cJSON. The code uses the environment variables `"OGS_PDUAPI_ADDR"` and `"OGS_PDUAPI_PORT"` for binding and port configuration.

3.7.3.4 Integration of APIs with OpenCAPIF

The integration with OpenCAPIF enables the N-App to securely access and expose network capabilities following standardized 5G service exposure principles. The N-App acts as a CAPIF API consumer, invoking Network as Code (NaC) functions to query the 5G Core for session and addressing information. The custom SMF API is registered and accessed through CAPIF, ensuring proper authentication, authorization, and service discovery. This integration allows the N-App to operate within a standardized API exposure framework, promoting interoperability and secure access to network data. By leveraging OpenCAPIF, the solution aligns with 3GPP specifications for API exposure and enables seamless interaction between network functions and application-layer services.

3.7.3.5 Development, Deployment and Results

The N-App has been implemented as a backend service exposing a persistent WebSocket interface to support real-time bidirectional communication with client applications. The solution has been developed following a modular approach and is deployed within the 5G Core infrastructure at La Mayora. The application is containerized to facilitate deployment, scalability, and integration within the testbed environment. The N-App has been developed as a fully integrated network application combining WebSocket-based communication, CAPIF integration, and custom 5G Core extensions. The system enables real-time interaction between

application clients and network services, supporting dynamic endpoint discovery. The use of asynchronous communication mechanisms, lightweight APIs, and containerized deployment ensures high performance, scalability, and ease of integration. The solution demonstrates the practical application of Network as Code principles within a 5G testbed environment. The N-App has been deployed within the 5G Core infrastructure at La Mayora. The application runs as a containerized service and is accessible to authorized devices and services connected to the local 5G network. The custom SMF API is deployed as part of the Open5GS core, requiring minor modifications to the SMF component. OpenCAPIF is also deployed within the infrastructure to enable standardized API exposure and access control. This deployment ensures seamless interoperability between the N-App, the 5G Core, and external applications such as V-Apps, supporting real-time operation and secure access to network capabilities. The N-App has been successfully deployed and validated within the 5G Core infrastructure at La Mayora. The implemented API demonstrates reliable and efficient retrieval of network endpoint information (IP address) associated with AGVs based on their IMSI identifiers. The WebSocket-based interface enables continuous and low-latency communication with client applications, allowing real-time query processing and response delivery. The integration with the 5G Core through CAPIF and Network as Code (NaC) functions has been fully validated, ensuring secure and standardized access to network information. The custom API implemented within the Open5GS SMF successfully exposes PDU session data, including IMSI-to-IP mapping, which is correctly consumed by the N-App. The system has demonstrated stable operation under test conditions, supporting multiple queries and continuous service availability.

Example API responses confirm the correct retrieval of session information:

```
xruser@UMA-MA-NOK-Core-SA5:~$ curl 127.0.0.4:7771/pdu-sessions
[{"imsi":"001020000000250","pdu_session_id":1,"ipv4":"10.45.21.50","sst":1,"sd":"000007"}]
xruser@UMA-MA-NOK-Core-SA5:~$ curl 127.0.0.4:7772/pdu-sessions
[{"imsi":"001020000000251","pdu_session_id":1,"ipv4":"10.45.22.51","sst":1,"sd":"000002"}]
xruser@UMA-MA-NOK-Core-SA5:~$ curl 127.0.0.4:7773/pdu-sessions
```

The N-App effectively enables seamless integration with external applications such as the V-App, supporting dynamic configuration and real-time operation of the immersive remote driving system. The logs of a succeeded IP retrieve are shown in Figure 26.

```
Waiting for input...
IMSI: 001020000000251
instantiating NaC API...
instantiated.
received UE session.
IP: 10.45.22.51
}
imsi: '001020000000251',
pdu_session_id: 2,
ipv4: '10.45.22.51',
sst: 1,
sd: '000002'
}
}
qos: {
  arp: {
    priority_level: 8,
    pre_emption_capability: 1,
    pre_emption_vulnerability: 1
  },
  index: 9
},
ambr: { downlink: { value: 1, unit: 3 }, uplink: { value: 1, unit: 3 } },
_id: new ObjectId("690df0e6bfc56e03bb4ceae8"),
name: 'internet2',
type: 9,
ue: { ipv4: '10.45.22.51' },
pcc_rule: []
}
}
state: 'ACTIVE',
imsi: '001020000000251',
dnn: 'internet2',
snssai: { sst: 1, sd: '000002' },
pduSessionType: 'IPv4v6',
sessionAmbr: { uplink: '1 Gbps', downlink: '1 Gbps' },
QosFlow: { fiveqi: 9, arp: 8 },
pduAddress: { ipv4: '10.45.22.51' }
}
GET /locations/null/sites/1002_torre_n40/5G/SA/00102/imsis/001020000000251/ue_sessions?status=active 200 1435.612 ms --
```

Figure 26 N-App and NaC logs during IP request

3.7.4 Automatic Surveillance & Detection in Farming Operation N-App

3.7.4.1 Overview

The Automatic Surveillance & Detection in Farming Operation N-App is responsible for interacting with the 5G Core in support of the associated V-App. Its main functionality is to leverage the HPE 5GC API in order to retrieve network-related information for connected devices, such as the IP address associated with a given IMSI, and to request differentiated network treatment, for example a specific 5QI or higher priority for the UEs that report SRS information. Through this interaction, the N-App provides the V-App with the network information required to identify specific UEs and to support suitable QoS handling for surveillance-related traffic. The current implementation is lightweight and has already been deployed and tested in an emulated scenario.

3.7.4.2 Architecture

The architecture of the Automatic Surveillance & Detection in Farming Operation N-App follows a lightweight network-assistance approach aligned with the project N-App concept. The N-App includes a northbound interface towards the associated V-App, through which it receives requests related to UE identification and differentiated network treatment, and a southbound interface towards the HPE 5GC API, through which it retrieves UE-related information and issues QoS-related requests. Internally, the N-App contains a simple control logic component that maps the V-App needs to the appropriate 5GC API calls, for example resolving the IP address associated with a specific IMSI and requesting a suitable 5QI or higher priority for the UE carrying SRS-related traffic. Overall, the architecture is aligned with the project N-App template, although the current implementation is intentionally simplified due to the limited functional scope of this N-App.

3.7.4.3 Analysis and Development of 5G APIs

The Automatic Surveillance & Detection in Farming Operation N-App leverages the HPE 5GC API (Table 17) in order to perform two basic network-assistance functions for the associated V-App: (i) retrieving network-related information for connected UEs, in particular the IP address associated with a given IMSI, and (ii) requesting differentiated QoS treatment, such as a specific 5QI or higher priority, for the UEs that report SRS information. These functions are relevant because they allow the V-App to identify the correct UEs and to obtain network support for the radio sensing workflow. At the current stage, the N-App has been validated against the HPE 5GC API in an emulated scenario. Since the exact method naming and resource structure may depend on the specific HPE 5GC API exposure available in the testbed, the API usage is reported below at functional level.

Table 17 Set of API endpoints and methods used by Automatic Surveillance & Detection in Farming Operation N-App

REST Method	Resource URI	Body Parameter	Description
POST	/POST_AsSessionWithQoSSubscription	suppFeat, notifUri, ipAddress, medCompN, 5qi, medSubComps (fNum, marBwDL, marBwUL, mirBwDL, mirBwUL)	Allows an AF to create or update an AF session with specific QoS requirements. The AF can request QoS parameters such as 5QI and bandwidth limits (maximum and minimum, uplink and downlink) for one or more media flows associated with a UE IP address, enabling fine-

			grained QoS control for the service.
POST	/POST_IP2IMSI	ipAddress	Allow the Application Function to map and identify the IMSI using an IPv4 address, enabling correct user identification within the network.
POST	/POST_IMSI2IP	supi	Enable the Application Function to map the IPv4 address based on the provided SUPI, supporting IP-level user identification.

3.7.4.4 Integration of APIs with OpenCAPIF

The integration with OpenCAPIF is considered as the project-compliant mechanism through which the N-App can consume the relevant HPE 5GC APIs in the final testbed architecture. At the current stage, the main focus has been on validating the functional interaction with the HPE 5GC API in an emulated scenario. Nevertheless, the N-App is aligned with the project OpenCAPIF approach, where OpenCAPIF provides the API exposure, onboarding, and discovery framework, while the N-App acts as the API invoker for the selected 5GC functions required by the surveillance use case. Overall, the development of the Automatic Surveillance & Detection in Farming Operation N-App has been intentionally kept simple, since its role is focused on a narrow but important set of network-assistance functions for the associated V-App. The current work has demonstrated the feasibility of retrieving UE information from the 5G Core and requesting differentiated treatment for the UEs involved in SRS reporting. The next development steps may include a more formalized northbound interface with the V-App, fuller documentation of the selected HPE 5GC API calls, and packaging of the N-App according to the final testbed deployment model. The Automatic Surveillance & Detection in Farming Operation N-App has been deployed and tested in an emulated scenario. In the current setup, the deployment focuses on validating the interaction between the N-App, the HPE 5GC API, and the associated V-App, particularly for IMSI-based UE identification and QoS-related requests for the UEs that report SRS information. This validation confirms that the N-App can provide the necessary network information and differentiated treatment support required by the farming surveillance workflow. In the target testbed architecture, the N-App is expected to be integrated as a containerized application together with the corresponding 5G API exposure framework and OpenCAPIF-based API consumption model.

The current results demonstrate that the Automatic Surveillance & Detection in Farming Operation N-App has been successfully deployed and tested in an emulated scenario. The validation confirms that the N-App can interact with the 5G Core through the HPE 5GC API in order to retrieve UE-related information and support differentiated network treatment for the UEs associated with SRS reporting. These results confirm the feasibility of the N-App as a network-support component for the farming surveillance workflow. In addition, the solution was shown during Trial 1 in Scenario 3 (Ada Byron building) as part of the overall use case demonstration. At the current stage, the reported results correspond to functional validation of the lightweight N-App concept and its interaction with the associated V-App and the 5G Core in an emulated setup.

3.7.4.5 Development, Deployment and Results

The development of the Automatic Surveillance & Detection in Farming Operation N-App has focused on implementing the minimum network-control functionality required to support the associated farming surveillance V-App. In particular, the current implementation concentrates on the interaction with the HPE 5GC API in order to retrieve UE-related network information and request differentiated treatment for the UEs responsible for reporting SRS information. In this way, the N-App acts as a lightweight intermediary between the V-App and the 5G Core, enabling the surveillance workflow to become network-aware. At the current stage, the implementation is functionally validated but still relatively simple, and therefore detailed repository-

level information is not yet reported in this version of the document. The present effort has been primarily devoted to validating the interaction logic and confirming that the N-App can support the V-App by providing UE identification information and requesting appropriate priority treatment in an emulated environment.

3.7.5 DSS for reliable water treatment N-App

3.7.5.1 Overview

DSS for reliable water treatment N-App provides adaptive connectivity support for sensing devices deployed in the water treatment monitoring environment. Its main function is to ensure stable and efficient data delivery by dynamically provisioning and tuning network resources according to monitoring needs and available communication capabilities. Through integration with Telefónica Open Gateway, the application can adjust bandwidth and related service parameters in order to preserve reliability and increase monitoring capacity when required, for example by supporting higher image resolution, increasing capture frequency, or prioritising critical telemetry.

3.7.5.2 Architecture

The architecture of DSS for reliable water treatment N-App is centred on adaptive connectivity management. The application evaluates current communication capabilities and applies QoS-related adaptations to support reliable and efficient transmission from heterogeneous sensing devices. Its internal design includes connectivity management, dynamic parameter tuning, performance validation, and interaction with Telefónica Open Gateway for the execution of Quality on Demand (QoD) policies. The architecture is aligned with the N-App role as a network-oriented component, focused on real-time adjustment of communication resources in response to changing monitoring and service conditions.

3.7.5.3 Analysis and development of 5G APIs

The N-App makes use of Telefónica Open Gateway Quality on Demand (QoD) API to adjust service quality parameters according to monitoring requirements (see Table 18). Through this integration, the application can request changes to network quality conditions directly by code, enabling adaptive provisioning of communication resources for sensing devices. At the current stage, the N-App has already been integrated with this API and is able to modify QoS settings programmatically.

Within the Spanish cluster, the pilot Movistar commercial 5G network. This environment enables the testbed to leverage standardized network exposure capabilities via the GSMA Open Gateway framework, specifically CAMARA APIs. By integrating these programmable APIs, the Network Application (N-App) can seamlessly interact with the commercial 5G core to request and manage network resources tailored to the specific needs of the vertical use cases. For the current phase of the trials, the N-App implementation primarily focuses on the **Quality on Demand (QoD) API**. This API empowers the N-App to dynamically request elevated Quality of Service (QoS) profiles—such as prioritized bandwidth in congested network scenarios—ensuring stable, performant, and reliable data delivery for the critical devices and sensors deployed in the field.

Table 18 Set of API endpoints and methods used by DSS for reliable water treatment N-App

REST Method	Resource URI	Body Parameter	Description
POST	Create QoD session	Device, application server, device ports, application server ports, QoS profile, session duration	Create QoS Session to manage latency/throughput priorities
GET	Check QoD session	Session id	Gets information about an active QoS session

DELETE	Cancel QoD session	Session id	Release resources related to QoS session
POST	Extend QoD session	Session id, requested additional duration	Extend the overall duration of an active QoS session.

3.7.5.4 Integration of APIs with OpenCAPIF

The integration of the application with OpenCAPIF is considered to expose selected application interfaces in a secure and standardized manner. In particular, OpenCAPIF is going to be used for authentication and sensors discovery.

The overall development of DSS for reliable water treatment N-App is focused on delivering a network-oriented application capable of adaptive connectivity management for sensing devices operating under variable field conditions. The integration already achieved with Telefónica Open Gateway and its QoD API provides the basis for dynamic service adjustment by code. The remaining work will focus on finalizing the implementation, refining the adaptation logic, and preparing the application for deployment in the target testbed environment. DSS for reliable water treatment N-App will be deployed as a containerized application in the target testbed infrastructure. This deployment approach supports portability, modular integration, and maintainability across environments. The deployed application will include the configured integration with Telefónica Open Gateway and the associated SIM-based connectivity required by the sensing devices.

At the current stage, the main results achieved for DSS for reliable water treatment N-App are the registration of the application in Telefónica Open Gateway, the registration of the SIM cards that will provide service to the sensors, and the integration of the N-App with Telefónica's QoD API. These developments already allow programmatic modification of quality-of-service conditions from the application. Additional results related to deployment, validation under operational conditions, and performance assessment will be reported in a later phase.

3.7.5.5 Development, Deployment and Results

The development of DSS for reliable water treatment N-App focuses on implementing the adaptive connectivity functions required to support sensing devices under variable operating conditions. The application has already been registered in Telefónica Open Gateway, together with the SIM cards that will provide service to the sensors. In addition, the N-App has been integrated with Telefónica's QoD API, making it possible to modify quality-of-service conditions programmatically from the application. The next development steps will focus on consolidating the adaptation logic, completing the software implementation, and packaging the solution as a containerized application suitable for deployment in the final testbed infrastructure.

GitLab link: As the application handles sensitive information, the software will remain in a restricted development environment and will not be uploaded to GitLab.

Technical details:

- Programming language: Python
- Containerization: Docker
- External integration: Telefónica Open Gateway
- Network API: Quality on Demand (QoD) API
- Connectivity support: SIM-based service provisioning for sensing devices

The current results show that the N-App is capable of creating, verifying, deleting, and extending QoD sessions, and its benefits have been verified in the laboratory. Further testing will be conducted in the coming phases.

3.7.6 N-Apps under Development for Phase 2

During Phase 2, the Immersive Remote Driving N-App will concentrate on building the full system responsible for utilizing the information provided by the AI-App to dynamically manage network slicing in the 5G environment. This includes processing real-time data and making intelligent decisions to trigger slice changes, ensuring optimal network performance based on current conditions. For the DSS for water quality N-App, the next development steps will focus on implementing the application in Python, finalising the interfaces with the sensing layer and the AI-App, and incorporating the mechanisms required to monitor and preserve data continuity through basic QoS indicators. Additional work will address containerization, integration in the target testbed infrastructure, and end-to-end validation under representative operating conditions. Regarding the DSS for reliable water treatment N-App, the next steps will focus on consolidating the current implementation, refining the adaptive connectivity logic already integrated with Telefónica Open Gateway, and completing the remaining software components required for operational deployment. Further work will also include validation of the QoS adaptation mechanisms under realistic conditions, containerization, and full integration within the testbed infrastructure.

4 Development of V-Apps

The development of the Vertical Application (V-App) focuses on creating the user-facing layer of the 6G ecosystem. Designed to act as the primary interface for the system, the V-App is engineered to translate complex network capabilities into accessible, domain-specific functionalities. Within the scope of the 6G testbeds, the V-App is not a generic solution; rather, it is tailored to meet the specific requirements of each pilot domain. Its development prioritizes ensuring that the unique challenges of each use case are met with precision, providing intuitive interfaces and real-time data insights that facilitate enhanced decision-making for end-users. To provide a comprehensive overview of this creation lifecycle, this section outlines the foundational requirements of the V-Apps and the strategic development plan and milestones guiding their delivery. It further details the technical development process and workflow employed, before concluding with an examination of the specific V-Apps implemented per cluster. This section provides a comprehensive overview of the Vertical Applications (V-Apps) landscape across all clusters, captured in the final version of the initial prototype that has been uploaded in the common software development platform. The following subsections detail the **technical integration**, verifying the interoperability of components, and the overall **readiness status** of each cluster. These assessments ensure that all necessary dependencies are met so that the initial version of the V-Apps-related software is ready for the initial phase (M3-M15), prioritizing the identification of application functionalities as part of this first prototype.

4.1 Bulgarian Cluster

During Phase 1 (Trials 1), the Bulgarian cluster showcased the V-App components and V-App Regional – monitoring DERs. Two more V-Apps were also developed in this period, the V-App Central WAC and the V-App Regional Controller – RC and they will be demonstrated as part of Trial 2 of Phase 1 to be hosted by M18 (see Table 19). Together, these applications cover PMU-based wide-area monitoring, regional DER monitoring, central control computation, and regional execution of control requests. For Phase 2 (Trials 2 and 3), the same V-App family will be further matured through tighter integration with the AI-App and ML-Ops framework, wider end-to-end validation in the Bulgarian pilot infrastructure, and additional hardening of the distributed deployment and control workflow. In this phase, the focus is not on introducing new V-Apps, but on extending the maturity, interoperability, and operational readiness of the existing V-Apps.

Table 19 Implemented and under development V-Apps in the Bulgarian cluster

Cluster Name	App Name	Phase (1or 2)
Bulgarian Cluster	V-App Central – WAM	1
Bulgarian Cluster	V-App Central – WAC	1
Bulgarian Cluster	V-App Regional – Monitoring DERs	1
Bulgarian Cluster	V-App Regional Controller - RC	1

In the Bulgarian cluster, two main V-Apps were originally defined in [D2.1](#) (Platform architecture and use case requirements – Initial, Section 3.1.5) : V-App Central (Wide Area Monitoring and Control, WAM-C) and V-App Regional (Controller). As the development and maturity of these applications evolved between the submission of D2.1 and the present deliverable, it was decided to separate them into four software sub-components according to their functionality, logical interaction, ownership, and development responsibility.

More specifically, the originally defined V-App Central (WAM-C) was separated into:

- App#1: V-App Central – WAM
- App#2: V-App Central – WAC

Similarly, the originally defined V-App Regional (Controller) was separated into:

- App#3: V-App Regional – monitoring DERs
- App#4: V-App Regional Controller – RC

The V-App Central – WAM and V-App Regional – monitoring DERs are developed and owned by SCBG, while the V-App Central – WAC and V-App Regional Controller – RC are developed and owned by UCY – KIOS CoE.

This separation was necessary for two main reasons. First, it facilitates the CI/CD development, validation, and testing process, as the V-App Central – WAC and V-App Regional Controller – RC are implemented by UCY, whereas the V-App Central – WAM and V-App Regional – monitoring DERs are implemented by SCBG. Second, it reflects the actual way the Bulgarian-cluster functions are validated across the project environments, including the UCY digital-twin and pre-piloting setup, where some fault and disturbance scenarios cannot be safely or realistically reproduced in the production-like Bulgarian pilot environment. The following table summarizes the current V-App structure, ownership, and deployment/testing locations.

Table 20 current V-App structure, ownership, and deployment/testing locations

V-App Name	Owner / Developer	Location of deployment and testing in the Bulgarian testbed
App#1: V-App Central – WAM	SCBG	Central cloud of the A1 testbed – Sofia,
App#2: V-App Central – WAC	UCY – KIOS CoE	and virtual central cloud of the UCY testbed
App#3: V-App Regional – Monitoring DERs	SCBG	Virtual central cloud of the UCY testbed
App#4: V-App Regional Controller - RC	UCY – KIOS CoE	Edge cloud of A1 – Varna
		Virtual edge cloud of the UCY testbed

All four V-Apps of the Bulgarian cluster run in Docker containers and are deployed through Helm charts in Kubernetes environments. In addition, all four V-Apps expose a web-based user interface or service interface appropriate to their functional role.

4.1.1 V-App Central -WAM

4.1.1.1 Overview

The V-App Central – WAM is the central wide-area monitoring application of the Bulgarian cluster. It receives real-time measurements from PMU devices and from the regional V-Apps, stores the data in a time-series-optimized database, and provides real-time visualization of the wider power-grid state. It is intended for use by grid operators in the control room and serves as the central monitoring entry point of the pilot. In addition to real-time supervision, it supports data exchange with N-Apps and AI-Apps and performs basic validation of incoming PMU measurements, including latency-related checks.

4.1.1.2 Architecture

The V-App Central – WAM follows the central-cloud side of the Bulgarian-cluster application architecture. It ingests PMU and regional monitoring data, validates and stores them, and exposes them to operators through a web-based interface. The application is aligned with the overall V-App design template of the project, with a clear separation between data ingestion, database persistence, alarming, visualization, and northbound application interaction. It is positioned at the central operator side of the pilot and acts as the main monitoring and data-access point for the wider grid view.

4.1.1.3 Development, Deployment and Results

The V-App Central – WAM is developed by SCBG as a monitoring-oriented software application with a time-series data backend and a browser-accessible UI. The application supports PMU data received through different synchrophasor transfer protocols, including IEEE Std C37.118.2-2011, IEEE Std C37.118-2005, IEEE Std 1344-1995, and IEC 61850-90-5. In addition to basic data validation, the implementation evaluates the latency of frames arriving from the PMUs.

The application uses an SQL-based configuration script. This script is editable by the administrator and defines the input data streams connected to the V-App by inserting the relevant records into the configuration database. User accounts are created through the SQL script included in the Helm chart of the V-App. The login web page is aligned with the corresponding regional application, and credentials are provided to users by the administrator. The application includes a dedicated web page presenting the list of connected devices. Device properties are accessible through device-specific web forms and provide the operator with the information required for real-time supervision. The application implements condition-based alarms. Alarm conditions can be defined using comparison operators such as equal to, not equal to, greater than, less than, greater than or equal to, and less than or equal to. The most commonly used operations are greater-than and less-than thresholds. To avoid chattering behaviour, a configurable delay is used to ensure that a signal remains beyond the alarm point before the alarm is raised. Alarm definitions are configured through a dedicated SQL script included in the Helm chart. Raised alarms are displayed in the UI and all alarm actions are recorded in an alarm log.

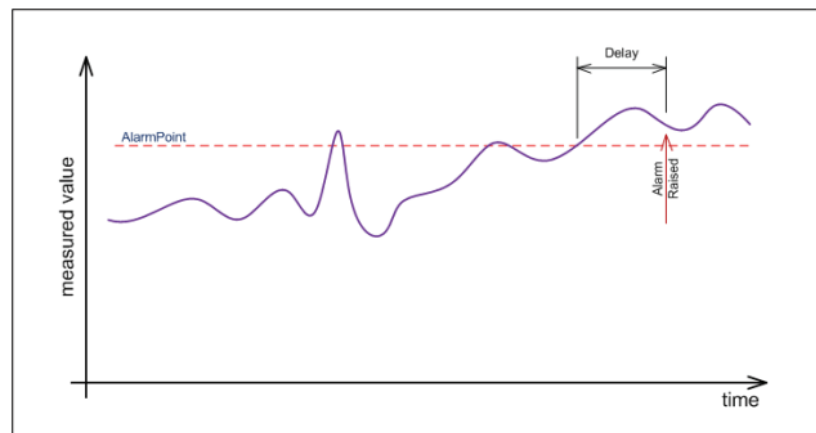


Figure 27 Configurable delay is used to ensure that a signal remains beyond the alarm point

Access to historical data is implemented through dedicated UI tabs. Operators can select signals of interest, visualize stored data from the database, and export the selected signal history for further analysis. The V-App visualizes selected PMU signals in real time on the Measurements page. For example, the frequency reported by the PMU of Entra Energy can be visualized through the corresponding live interface.

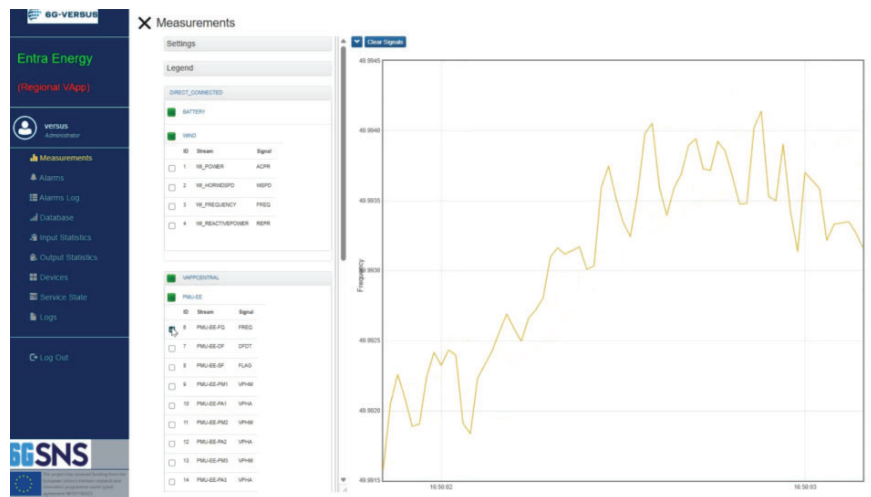


Figure 28 Snapshot of the web UI of the V-App Central – WAM

The application reports statistical parameters for each PMU, supporting engineering assessment of the quality and behaviour of the monitored data streams. Moreover, it provides a state page to monitor technical parameters of the V-App. Log access is also supported through the UI. A status log is always created, while error and connection-error logs are generated when needed. The V-App Central – WAM is deployed as a **containerized application**. In the Bulgarian pilot, it is deployed in the **central cloud of the A1 testbed in Sofia**, and a corresponding virtual central-cloud deployment is also maintained in the **UCY testbed** for integration and validation purposes. Deployment is performed using **Helm charts** in a **Kubernetes cluster**. Access to the web interface requires access to the internal pilot network, typically through VPN-based connectivity.

4.1.2 V-App Regional – monitoring DERs

4.1.2.1 Overview

The **V-App Regional – monitoring DERs** is the regional monitoring application of the Bulgarian cluster. It is intended for power producers such as Entra Energy that operate portfolios of distributed energy resources, including wind turbines, small hydro units, and a virtual BESS. The application receives real-time measurements from the energy resources of the producer, stores them in a time-series-optimized database, and provides real-time monitoring of the regional assets and their operational state.

4.1.2.2 Architecture

The V-App Regional – monitoring DERs is aligned with the edge-side portion of the Bulgarian-cluster architecture. It is deployed close to the producer-side environment and focuses on the acquisition, storage, and visualization of regional DER telemetry. Architecturally, it follows the same application principles as the central monitoring service, with clear separation between configuration, data ingestion, persistence, alarms, and the user-facing web interface, but it is specialized for regional DER monitoring rather than PMU-based wide-area monitoring.

4.1.2.3 Development, Deployment and Results

The V-App Regional – monitoring DERs is developed by **SCBG** as a regional monitoring application for DER owners and operators. Its software organization is similar in style to the central monitoring application but specialized for regional resource telemetry. The application uses an **SQL-based configuration script** to define the connected streams and monitored devices. This script is editable by the administrator and forms the basis of the application configuration. User accounts are created through the SQL script included in the Helm chart.

A login page is provided, and credentials are delivered to users by the administrator. The application includes a web page listing connected energy resources. Device properties are visible in dedicated device forms and allow operators to inspect the state and characteristics of the monitored assets. The regional monitoring application supports condition-based alarms in the same overall manner as the central monitoring application. Alarms can be configured for selected signals and operational thresholds, are displayed in the UI, and are recorded in the corresponding alarm log. The application supports access to saved historical data, including the ability to select signals, visualize recorded information, and export the relevant data for offline analysis. Finally, the V-App visualizes selected signals provided by distributed energy resources in real time on the **Measurements** page in Figure 29. For example, the power output reported by the wind turbine of Entra Energy can be displayed through the corresponding live view.

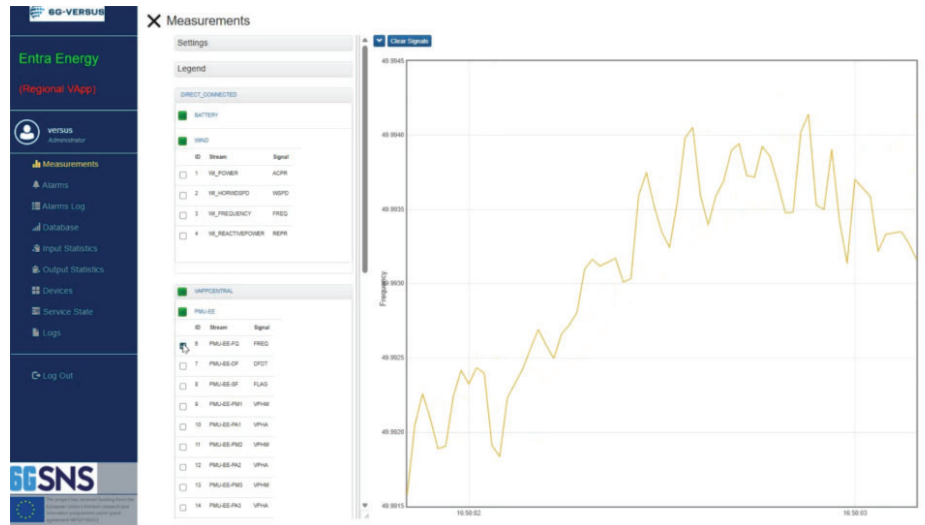


Figure 29 Snapshot of the web UI of the V-App Regional

The application reports statistical parameters for the data received from the connected energy resources, supporting engineering and operational analysis of the regional monitoring inputs. The V-App Regional – monitoring DERs is deployed as a **containerized application** in the **edge cloud of A1 in Varna**. It runs in a Kubernetes-based environment and is installed through Helm charts. This application represents the regional/producer-side monitoring entry point of the Bulgarian pilot.

4.1.3 V-App Central – WAC

4.1.3.1 Overview

The **V-App Central – WAC** is the software central control application developed by **UCY – KIOS CoE** for the Bulgarian pilot. It is implemented as a **Python service** and acts as the supervisory control layer between the PMU measurement layer and the regional control layer. In the current implementation, the WAC is not a generic monitoring dashboard. It is a real-time control application that ingests synchronized PMU streams, aligns the measurements, computes system-level electrical indicators, detects disturbances, calculates the required active-power response, and dispatches control requests to the V-App Regional Controllers. The design explicitly supports a **distributed deployment model** in which the WAC and RC are not expected to run on the same machine. Instead, they are designed to be deployed in different data centres located approximately **400 km apart**, communicating through explicit network APIs. This is an actual implementation target of the Bulgarian-cluster architecture.

4.1.3.2 Architecture

The V-App Central – WAC is architected as a modular control service. From a software-engineering perspective, it comprises several cooperating Python modules: a **WAC System** launcher responsible for

runtime orchestration, a **PDC** layer for PMU ingestion and timestamp alignment, a **WAC Processor** implementing event detection and central control logic, a **Control Actions** layer that communicates with the RC side and optional field/model interfaces, and a **Web Dashboard** providing live operator supervision through Flask and Socket.IO. The architecture is aligned with the project application model but specialized for real-time control rather than simple data presentation.

4.1.3.3 Development, Deployment and Results

The WAC is developed by **UCY – KIOS CoE** and maintained in the Bulgarian-cluster UCY GitLab repository: <https://gitlab.6g-versus.eu/wp4/v-apps/bulgarian-cluster/ucy>. The implementation is organized so that the same codebase can be used in:

- replay and engineering testing,
- OPAL-RT closed-loop validation,
- distributed pilot deployment.

The main software modules in the current implementation are:

- **WAC System:** runtime launcher for live PMU profiles,
- **PDC (phasor data concentrator):** PMU ingestion, frame parsing, timestamp alignment, batching, and handoff,
- **WAC Processor:** control logic, event detection, baseline capture, power-imbalance estimation, and dispatch preparation,
- **Control Actions:** outbound command delivery to the RC layer through network APIs and optional Modbus/TCP control paths,
- **Web Dashboard:** Flask / Flask-SocketIO UI for live supervision and validation, as seen in Figure 30.

The WAC supports runtime configuration through **environment variables**, **deployment profiles**, and optional **JSON-based local configuration**. This allows the same application to adapt to different PMU endpoints, event thresholds, control gains, baseline behaviour, RC communication settings, dashboard settings, optional Modbus mappings, and optional forwarding of aligned PMU slices to AI or ML-Ops endpoints. The current WAC is an engineering and pilot-stage service. Access is mainly provided through the live web dashboard. The implementation does not currently include a full RBAC subsystem, so deployment-level access control is expected to be enforced through the network environment, VPN, reverse proxy, or infrastructure policy.

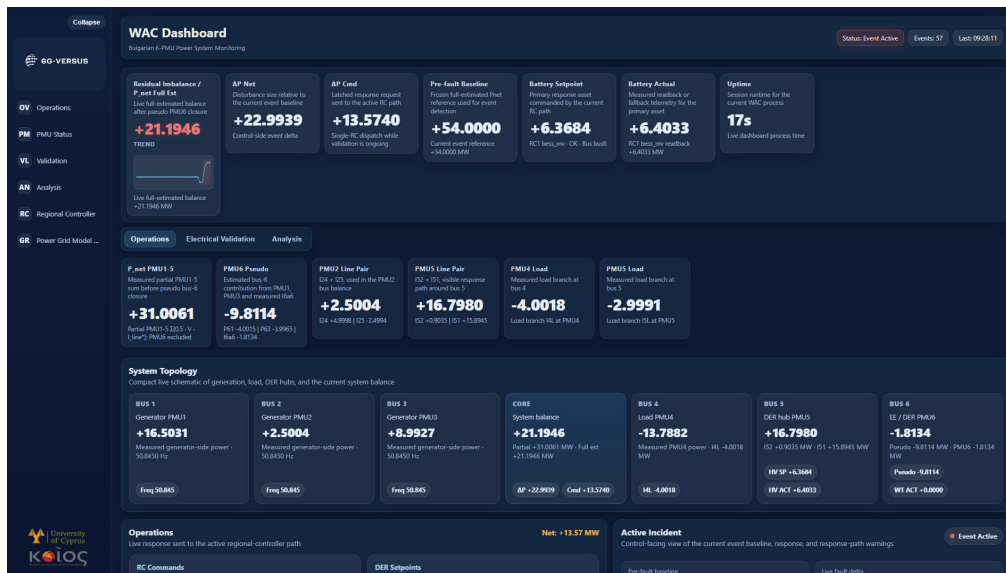


Figure 30 WAC Web Dashboard

The **V-App Central – WAC** includes a Flask and Flask-SocketIO dashboard that provides real-time visibility of the control process, including incoming PMU state, calculated power imbalance and ΔP values, event state, RC dispatch commands, BESS setpoint versus actual response, and validation-oriented workbench views. At application level, the WAC manages six PMU streams from the modified IEEE 9-bus / OPAL-RT digital-twin setup, the regional control entities, regional DER representations, aggregated RC capabilities, and optional Modbus-mapped assets used during OPAL-RT validation. For these entities, the WAC processes control-facing properties such as frequency, RoCoF, phasor-derived active power, system imbalance estimates, event baseline, live ΔP values, dispatched commands, RC capability, DER headroom, setpoints, actual response, and BESS SOC telemetry where applicable.

A key implemented function of the WAC is event detection and control preparation. The WAC continuously evaluates aligned PMU measurements and detects abnormal operating conditions requiring corrective action, using frequency deviation, RoCoF, and power-imbalance / ΔP logic derived from system Pnet. In the current OPAL-RT validation profile, the power-based detection mode is used because it is more reliable than the available frequency feed in the present setup. The implemented logic includes pre-fault baseline capture, event threshold crossing, event latching, short update windows for ΔP evolution, stabilization / freeze behaviour, event clearing, and return to steady-state operation. The WAC also supports engineering-oriented validation through dashboard event history, live trends, optional PMU persistence, replay tools, and optional forwarding of aligned PMU slices to AI-App or ML-Ops endpoints.

PMU ingestion is implemented through the PDC module, which connects to six IEEE C37.118 PMU streams, parses incoming frames, extracts timestamps, performs fixed-window cross-PMU alignment, validates complete measurement slices where required, and forwards aligned batches to the WAC processor and dashboard. Although monitoring is not the main purpose of the WAC, this live visibility is important during integration and trial validation because it allows the engineering team to verify the intermediate control state and not only the final dispatch command. The WAC also supports input and output assessment, including PMU alignment, continuity, freshness, electrical consistency, commanded ΔP , RC capability, DER setpoint, and actual response.

The WAC interfaces with the PMU measurement layer for synchronized data acquisition, the V-App RC layer for capability/status retrieval and dispatch submission, the operator dashboard for supervision, optional Modbus/TCP devices or OPAL-RT register maps for closed-loop validation, and optional AI-App / ML-Ops endpoints for aligned-slice export. The WAC–RC interaction is implemented through HTTP APIs using JSON payloads, which is important because the WAC and RC are designed to be deployable in different data centres located approximately 400 km apart. In the current implementation, the RC exposes APIs for retrieving regional capability, retrieving regional status and asset-level telemetry, and submitting dispatch commands, with representative resources including `/rc/<rc_id>/capability`, `/rc/<rc_id>/status`, `/rc/<rc_id>/dispatch`, and `/rc`. The WAC is implemented mainly in Python, with Flask, Flask-SocketIO, Requests, pymodbus, HTML, CSS, and JavaScript supporting the control logic, PMU ingestion, dashboard, RC communication, and optional Modbus integration. In its current validation form, it runs in the virtual central cloud of the UCY testbed and is integrated with the OPAL-RT and 5G-OAI pre-piloting environment, while its design supports future distributed operation where the WAC and RC are not co-located.

4.1.4 V-App Regional Controller – RC

4.1.4.1 Overview

The V-App Regional Controller – RC is the regional control application of the Bulgarian pilot. Its main role is to receive control requests from the V-App Central – WAC, assess the real-time flexibility available from the DERs of its region, and translate the requested regional response into device-level control actions. It acts as the execution layer between the central control logic and the regional DER fleet and supports the practical realization of coordinated DER support in the Bulgarian cluster.

4.1.4.2 Architecture

The RC is architected as a regional execution service positioned between the central control layer and the regional DER or DER-model layer. It combines telemetry acquisition, capability assessment, allocation logic, and control-output generation in a single continuous regional control loop. Upward, it exchanges status and dispatch data with the WAC through application APIs. Downward, it interfaces with model or field representations of DER assets, including Modbus-connected resource representations where applicable. This architecture is aligned with the project application model but specialized for distributed regional control execution.

4.1.4.3 Development, Deployment and Results

The **V-App Regional Controller – RC**, as seen in Figure 31, is developed by **UCY – KIOS CoE** and maintained in the Bulgarian-cluster **UCY** **GitLab** repository: <https://gitlab.6g-versus.eu/wp4/v-apps/bulgarian-cluster/ucy>

The RC is implemented as a **Python-based regional control service**. Its role is to receive active and reactive power requests from the V-App Central – WAC, assess the current state and available flexibility of the DER assets in its region, and translate the requested regional response into device-level control references. The RC is therefore not only a communication proxy, but the regional execution layer of the control chain.

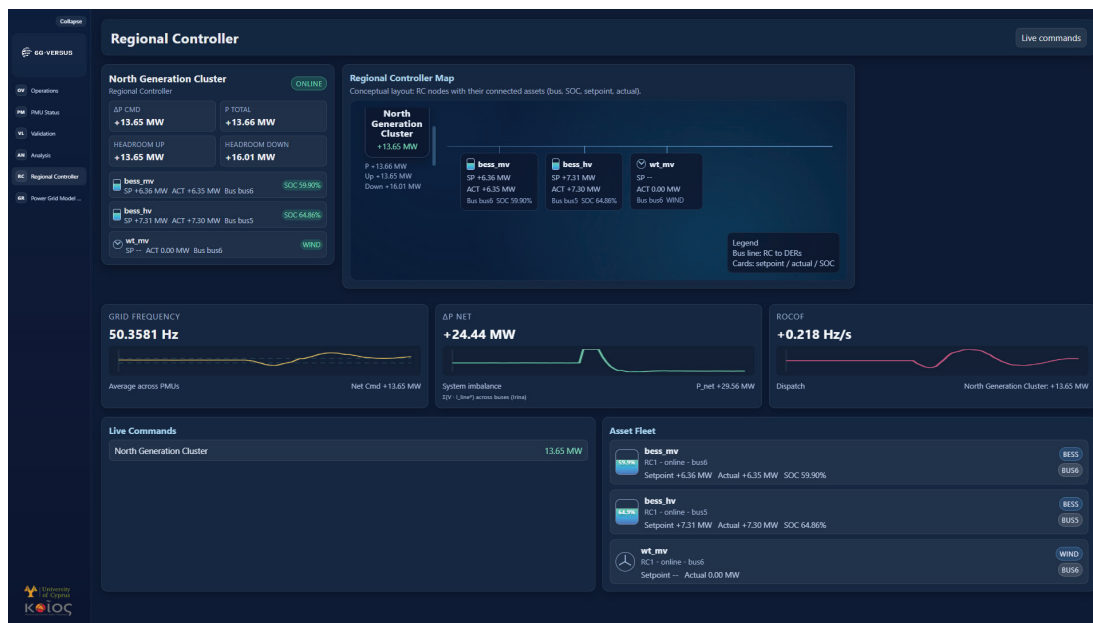


Figure 31 V-App Regional Controller – RC dashboard

The RC supervises regional DER assets such as storage systems, inverter-based resources, renewable generation units, and other controllable resources depending on the pilot configuration. For these assets, it evaluates operational limits, current operating point, upward and downward capability, state-related constraints such as BESS state of charge, and device availability. Based on this information, it allocates the requested response across the available DERs while respecting their technical constraints and priority class.

At the core of the implementation is the **regional allocation logic**, which continuously combines local telemetry, DER capability assessment, and dispatch requests received from the WAC. In the current implementation, active-power allocation follows a priority cascade, with storage resources considered first, followed by dispatchable resources and then renewable resources. This enables the RC to provide a feasible regional response while improving consistency between the requested dispatch and the actual DER behaviour.

The RC communicates upward with the WAC through **HTTP/REST APIs** for status reporting, capability exchange, and dispatch execution. Downward, it interacts with DER field or model interfaces, including **Modbus/TCP** where applicable. This API-based design is important for the Bulgarian pilot because the WAC and RC are intended to operate as distributed components deployed in physically separated locations, rather than as services running on the same machine.

The RC is deployed as a **containerized regional software service**. In the current Bulgarian-cluster validation setup, it runs in the virtual edge-cloud environment of the UCY testbed, supporting the regional execution side of the OPAL-RT and 5G-OAI pre-piloting environment. In Figure 31 a screenshot can be seen by the RC dashboard.

4.1.5 V-Apps under Development for Phase 2

For Phase 2, the Bulgarian cluster will continue to mature the same V-App family rather than introduce entirely new V-Apps. The main direction is to increase the robustness, interoperability, and deployment readiness of the four existing V-Apps while strengthening their interaction with the AI-App and ML-Ops framework. This includes further hardening of the distributed deployment model, refinement of the control workflow between WAC and RC, and broader validation across both the UCY pre-piloting environment and the Bulgarian pilot infrastructure. On the monitoring side, the focus is on improving integration with the rest of the application ecosystem, including AI- and N-App-related data exchange, as well as continued support for historical analysis and operational visibility. On the control side, the focus is on improving the stability and maturity of the regional execution logic, the WAC–RC communication path, and the distributed multi-site operation where central and regional functions are deployed in different locations.

A further Phase 2 objective is to complete tighter end-to-end validation of the triplet components with the AI-App and the ML-Ops layer, so that the V-App family evolves from a currently validated alpha chain into a more mature and interoperable pilot-ready software stack for the Bulgarian use case.

4.2 Finnish Cluster

The V-App has been successfully developed for both phases of the project. Phase 1 provides core functionality, including trial lifecycle management (start, run, and stop), real-time component health monitoring, and basic dashboard visualization. Phase 2 focuses on full integration with all system components, complete data pipeline support covering hourly collection cycles and three-hour FMI weather cycles, AI decision result visualization using ApexCharts, and a fully containerized deployment aligned with all other system components. The integration work for Phase 2 is nearly complete and the system is ready for deployment.

Table 21 Implemented and under development V-Apps in the Finnish cluster

Cluster Name	App Name	Phase (1or 2)
Finnish	V-App Web interface for the Energy Management System (WEMS)	Phase 1&2

4.2.1 Common functionality of V-App Central

4.2.1.1 Overview

Web interface for the Energy Management System (WEMS) vApp is a web-based energy and weather monitoring/control platform built with a Django REST backend, PostgreSQL for storage, and a React dashboard frontend. It stores user information, notifications/alarms, trial lifecycle data, and incoming measurement datasets (PV modules, inverter, wind generator, hydrogen fuel cell, back-up battery) in the database, then fetches and visualizes FMI data, AI results, logs, and measurement data through a unified, user-friendly dashboard.

4.2.1.2 Architecture

The V-App is architected as a multi-tier monitoring and control service. From a software-engineering perspective, it comprises several cooperating layers: a Django-based backend service responsible for business logic, authentication, and API exposure; a Daphne ASGI server enabling real-time bidirectional communication via Django Channels and Redis; a React/TypeScript frontend providing the operator dashboard with live gauges, time-series charts, and presence awareness; a Celery worker with a Beat scheduler executing periodic ingestion and health-check tasks; an Nginx reverse proxy unifying HTTP, WebSocket, and static asset delivery; and a PostgreSQL database whose schema is centrally managed by Alembic migrations and shared with the project's other components. The V-App communicates with geographically separated services — including the N-App ingestion layer, xApp, RemoteIC, NetworkIC, and AI components — over a shared Docker network using HTTP APIs. The architecture is aligned with the project application model and specialized for centralized supervision, data visualization, and trial lifecycle orchestration of the Finnish cluster pilot site.

4.2.1.3 Development, Deployment and Results

The dashboard, as visible in Figure 32, includes core panels (Status, Notification, Graph, Sidebar) to enable real-time visualization, system monitoring/control, log monitoring, and configuration features (e.g., power-saving mode). The web interface supports standard data exchange formats such as JSON/CSV to simplify integration with external tools and platforms. The platform also targets GDPR compliance, including Terms of Use and Privacy Policy pages, user profile editing, and user self-deletion with full removal of associated data. The user management module and the dashboard module have both been completed and will be presented in the following sections, with final refinements currently underway. Operationally, vApp orchestrates multiple external components via REST-based GET/POST interactions and internal scheduling. It provides a mechanism to GET all components to verify reachability and aggregate their health into a single system state, while handling partial failures and emitting clear user-facing notifications/alarms. It provides a mechanism to POST standby/startup commands to components, then transitions the system through Down, Starting, and Running only when acknowledgments are successful. During Running state, vApp triggers hourly data-push cycles for xApp, Remote IC, and Network IC, notifies AI-app1/AI-app2 (via POST notify) when database data is ready, and then retrieves and displays AI outputs in dashboard panels once analysis completes. In parallel, it runs a 3-hour FMI scheduler to request FMI data through nApp, fetches FMI data from the database once available, and visualizes it in the dashboard. When Stop Trial is issued, vApp sends stop commands to all components, terminates timers/processes, resets the system state back to Down, closes the session, and publishes a final notification summarizing the shutdown outcome.

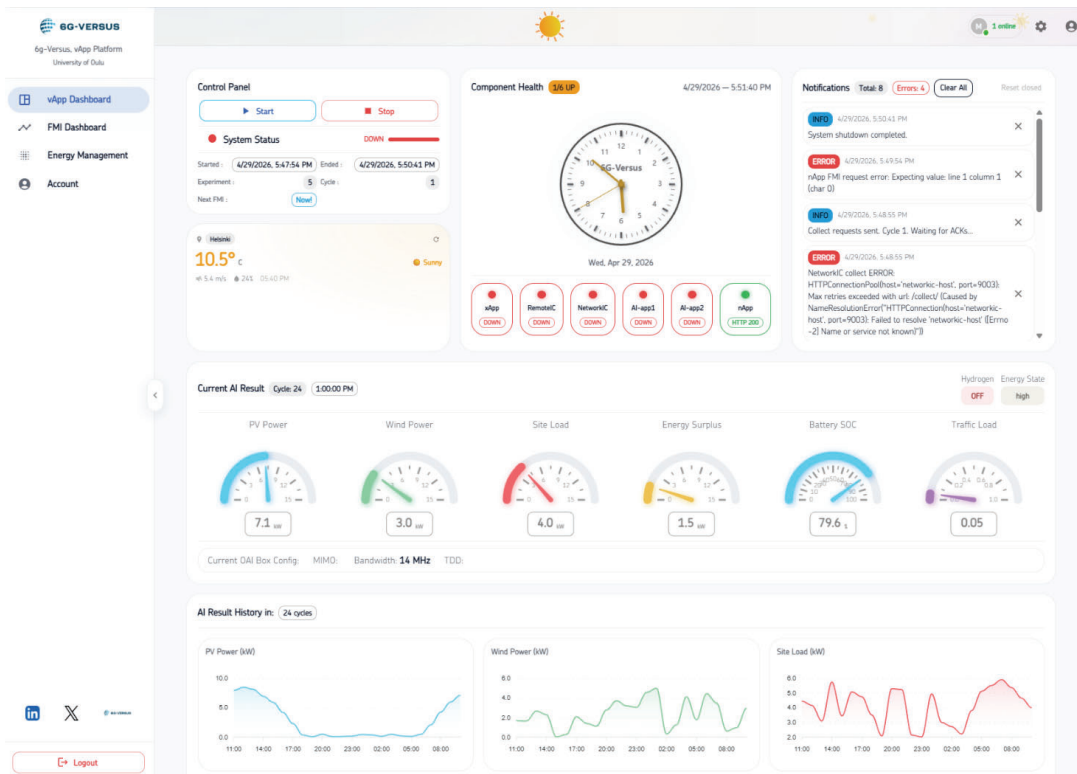


Figure 32 WEMS Dashboard

This section is related to user-specific operations and dashboard features, including account management, authentication, profile editing, activity tracking, account deletion, and the main dashboard interface. The user logs in on the login page using a username and password, which are stored in the database. The user can also navigate to the registration page, as seen in Figure 33.

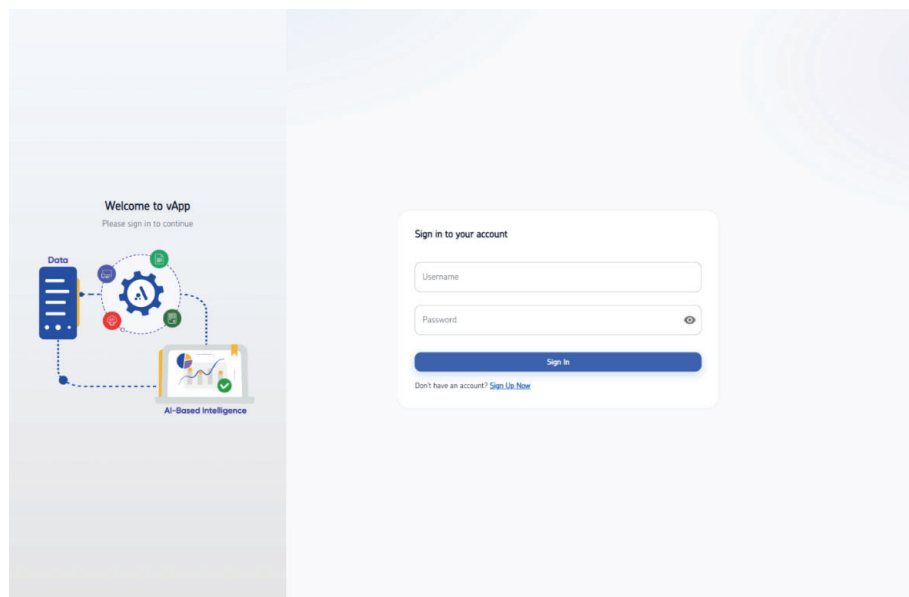
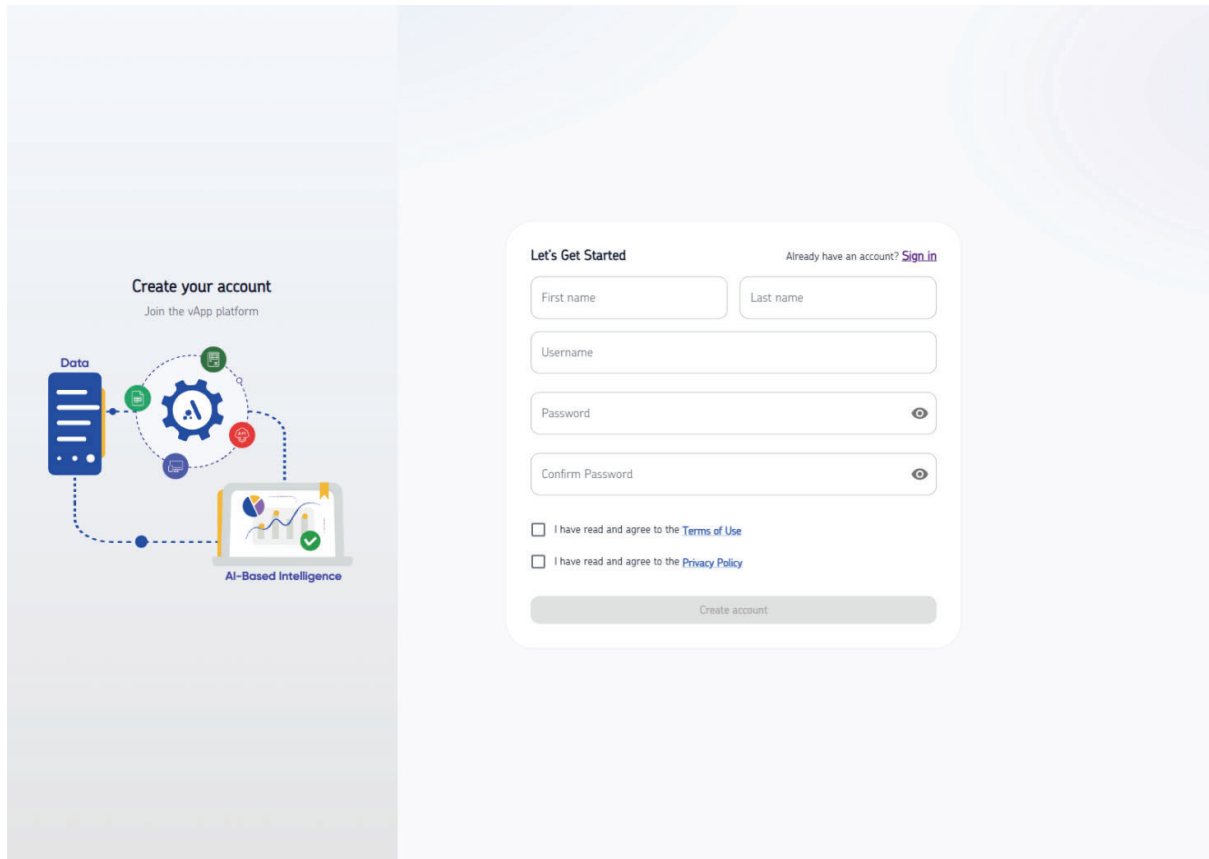


Figure 33 Snapshot from the User Login screen

The user can create a new account on the registration page by entering their first name, last name, username, and password as seen in Figure 34. The password is confirmed by re-entering it and is stored in the database.

As part of the registration flow, the user must accept the Terms of Use and Privacy Policy to comply with GDPR requirements. After successful registration, the user is redirected to the login page to sign in.



The image shows a user registration interface. On the left, there is a graphic titled "Create your account" with the subtitle "Join the vApp platform". The graphic includes icons for "Data", "AI-Based Intelligence", and a gear with a person icon, connected by dashed lines. On the right, there is a registration form titled "Let's Get Started". The form includes a link "Already have an account? [Sign in](#)". The form fields are: "First name", "Last name", "Username", "Password" (with an eye icon for visibility), and "Confirm Password" (with an eye icon for visibility). Below the fields are two checkboxes: "I have read and agree to the [Terms of Use](#)" and "I have read and agree to the [Privacy Policy](#)". At the bottom of the form is a button labeled "Create account".

Figure 34 Snapshot from the User Registration screen

The user can view and edit their personal information, change their password, and review their activity history as seen in Figure 35. The profile page also allows the user to permanently delete their account along with all associated personal data, in compliance with GDPR's right-to-erasure requirement.

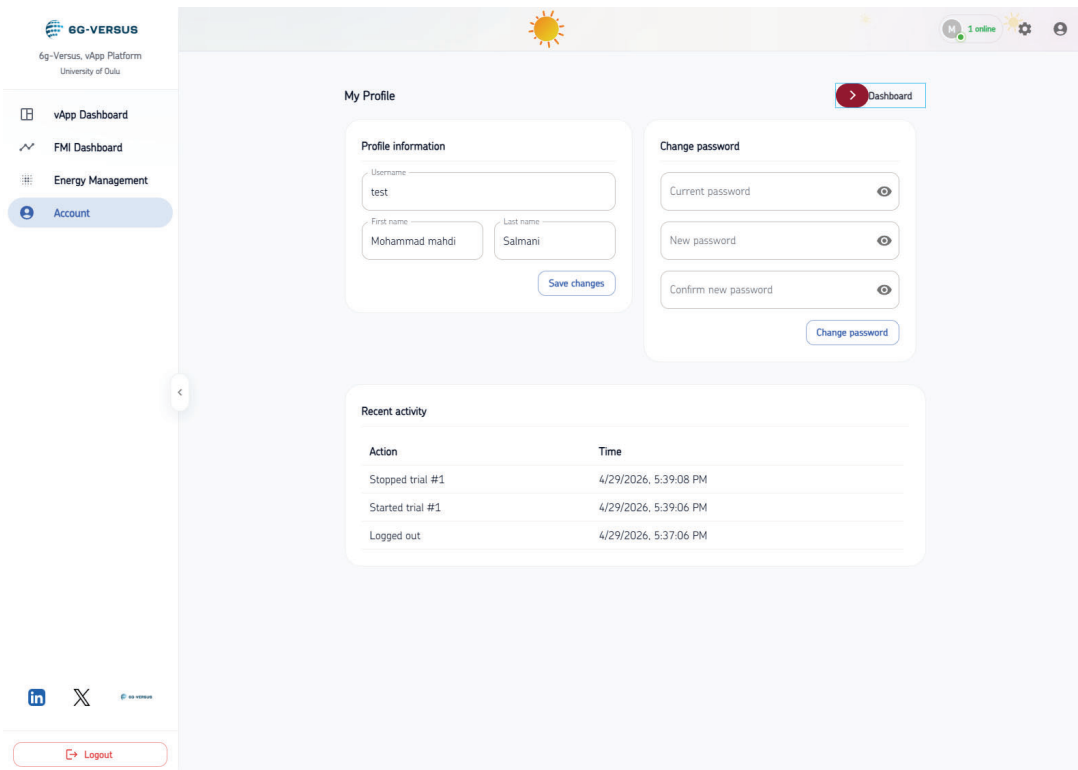


Figure 35 User Profile Editing and user activity

By selecting this option, the user can delete their account, as seen in Figure 36. All user-related data, including personal information and any trials created by the user, will be permanently removed from the system.

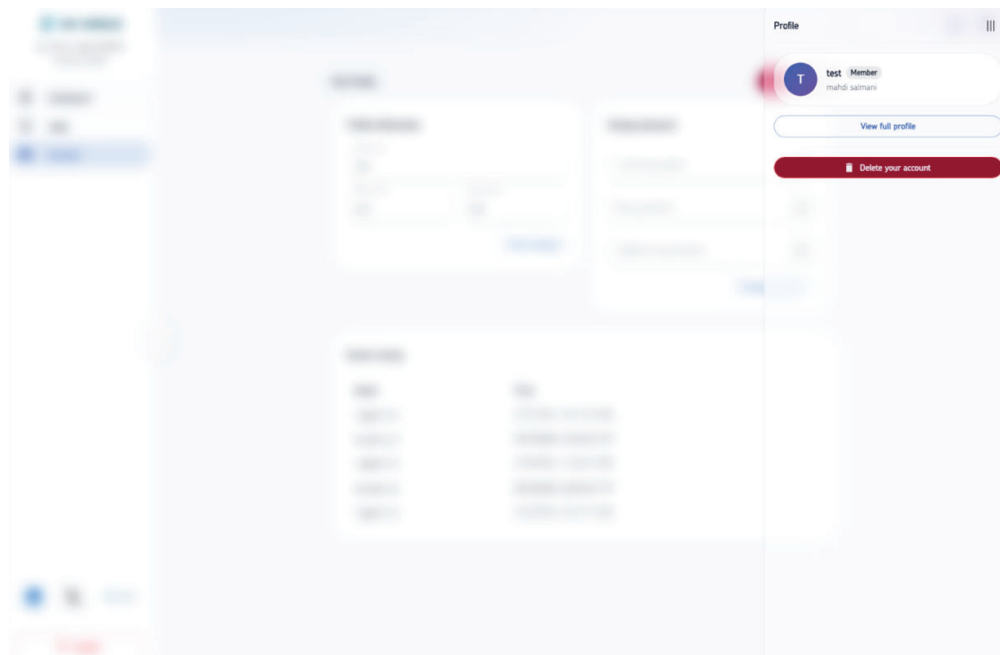


Figure 36 Snapshot of User deletion screen

The AI Result section displays the latest decisions produced by the AI optimization engine for the pilot site, retrieved from the central database after being persisted by the AI components through the N-App ingestion layer, as seen in Figure 37. Each decision reflects the recommended operational state of the site based on current environmental conditions, generation, storage, and predicted load. The view presents key indicators

including the hydrogen system activation flag, battery state-of-charge limit, photovoltaic and wind power outputs, total site load, energy surplus, predicted traffic load, OAI-Box configuration parameters, and the resulting OAI-Box energy state. A historical timeline of past decisions is also available, allowing operators to review how the AI guidance has evolved over time and correlate it with corresponding changes in environmental and energy conditions.

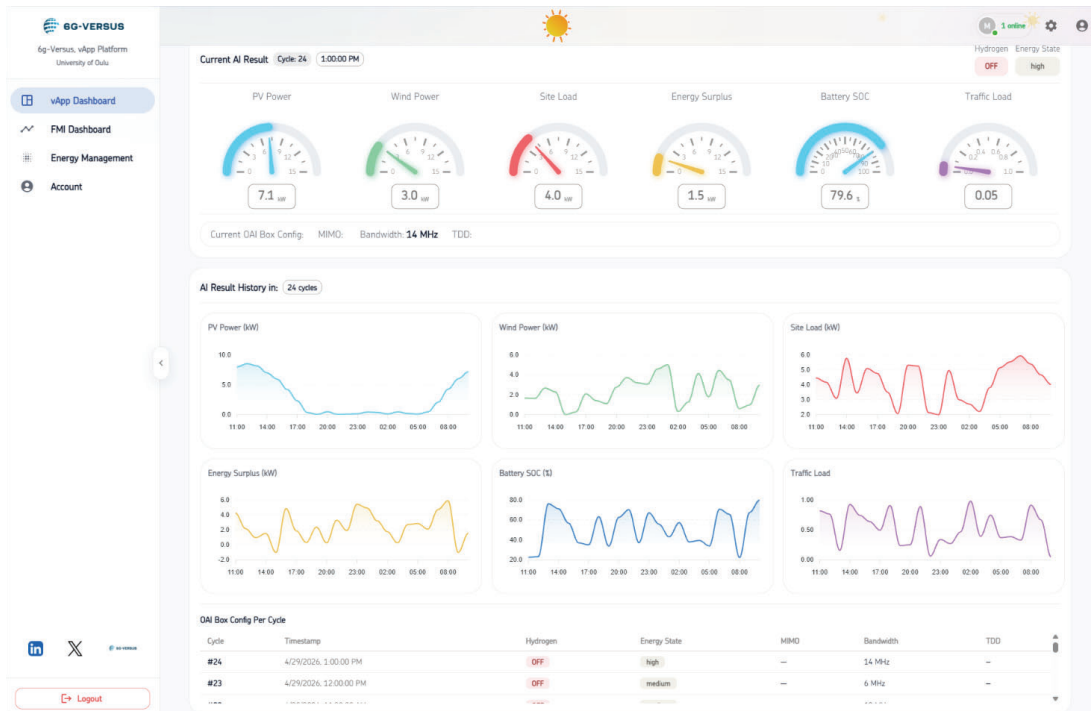


Figure 37 AI Result view on the main dashboard

The FMI Dashboard provides operators with a dedicated view of weather and solar irradiance forecasts retrieved from the Finnish Meteorological Institute (FMI) and previously persisted in the central database by the N-App ingestion layer, as seen in Figure 38. The dashboard visualizes key meteorological parameters — including ambient temperature, wind speed, cloud cover, global, direct, and diffuse horizontal irradiance (GHI, DNI, DHI), and plane of array radiation — as both real-time gauge widgets reflecting the latest snapshot and time-series area charts illustrating historical trends. This view is designed to give operators situational awareness of the environmental conditions driving renewable generation at the pilot site, supporting both day-to-day supervision and post-trial analysis.

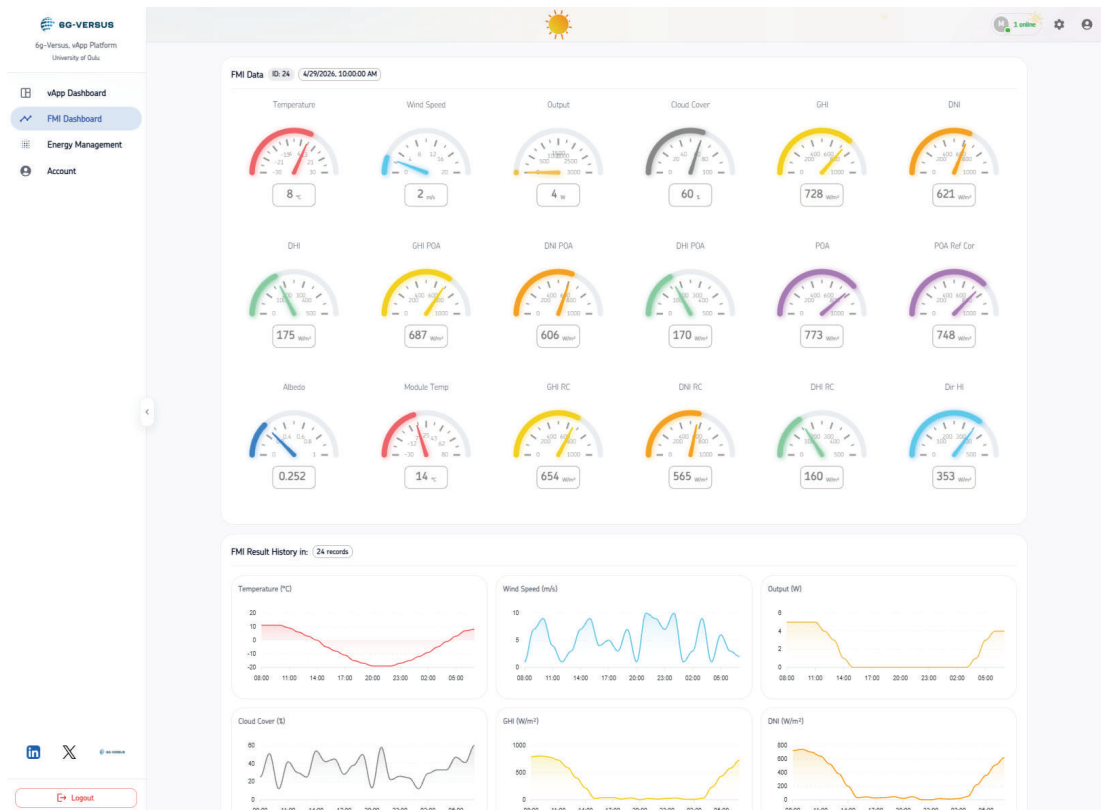


Figure 38 FMI Dashboard view

The Energy Management Dashboard provides operators with a unified view of the renewable energy ecosystem at the pilot site, retrieved from the central database after being ingested through the N-App layer from the Remote-IC component, as seen in Figure 39. The dashboard organizes data into four logical sections — Storage, Sensing, Production, and Consumption — each presenting both real-time gauge widgets reflecting the latest measurement and time-series area charts illustrating historical trends. The Storage section displays battery state-of-charge (SOC), state-of-health (SOH), and hydrogen reserve indicators. The Sensing section reports environmental measurements such as wind speed and direction, solar irradiance, and temperature readings from the outside environment, battery, hydrogen tank, and device enclosure. The Production section visualizes generation from wind, solar, and hydrogen sources. The Consumption section breaks down power usage across radio units, microwave links, hybrid inverters, HVAC, network switches, hydrogen cells, Raspberry Pi controllers, sensors, and NetIO devices. This unified view supports operators in correlating environmental conditions, generation, storage state, and consumption patterns during trials.

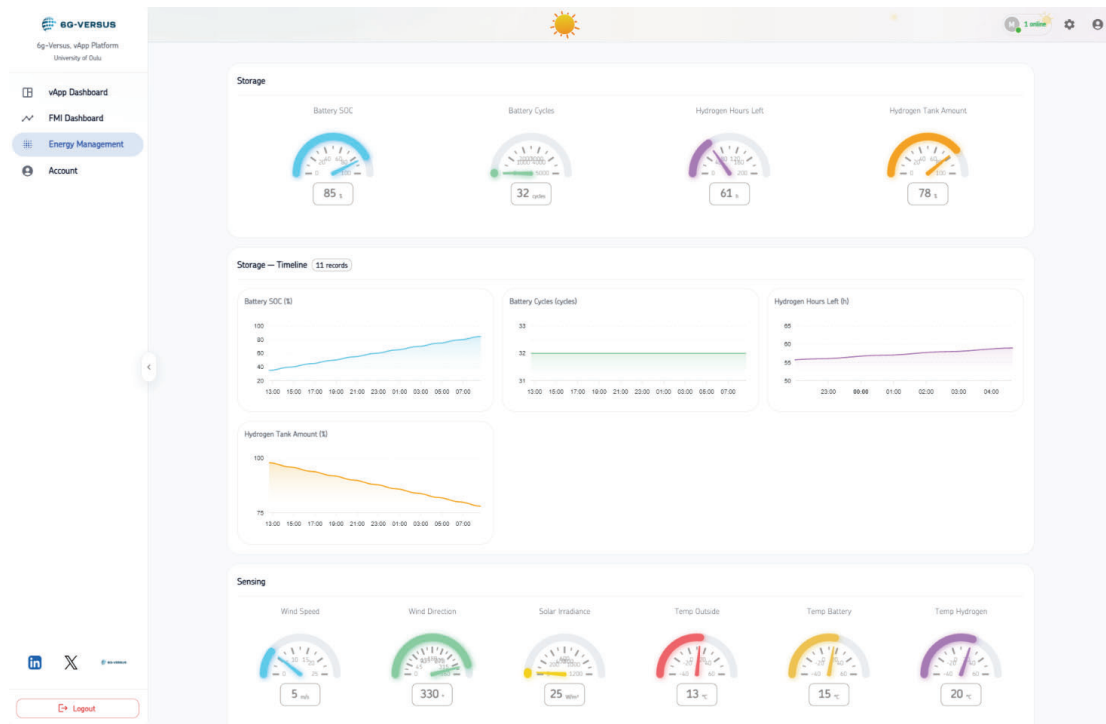


Figure 39 Energy Management Dashboard view

4.2.2 V-Apps under Development for Phase 2

Phase 2 is nearly complete. The data pipeline has been extended to support full end-to-end integration, AI decision results are retrieved from the shared database and visualized in real-time using ApexCharts, and FMI weather data visualization has been finalized. The V-App has been fully containerized using Docker with deployment configuration aligned across all system components. The remaining work is limited to final integration with external components and minor frontend refinements.

4.3 Greek Cluster

Within the Greek cluster, three Vertical Applications (Table 22) are currently planned across the two implementation phases of the project. During Phase 1 (Trials 1 and 2), the Greek pilot will showcase the **INUS Asset V-App**, the **EEaaS Dashboard V-App**, and the **Wildfire Detection Control Room V-App**, which constitute the primary vertical-facing interfaces for the initial wildfire monitoring and emergency response scenarios. In particular, a prototype INUS Asset V-App was planned and developed as a Phase 1 V-App in the Greek cluster and was showcased in Trial 1 as the main interface for monitoring and managing field assets and events. The aforementioned V-Apps provide the operational front-end for asset monitoring, energy observability, and real-time situational awareness/control of the wildfire detection workflow. For Phase 2 (Trials 3 and 4), the relevant V-Apps, most notably the INUS Asset V-App and the Wildfire Detection Control Room V-App, are planned to be further integrated and enhanced towards the 2027 trial activities, building on the outcomes and observations from the first implementation phase. All V-Apps are under continuous development, with their final stable versions to be reported in D4.2.

Table 22 Implemented and under development V-Apps in the Greek cluster

Cluster Name	App Name	Phase (1or 2)
Greek	INUS Asset V-App	1 and 2
Greek	Wildfire Detection Control Room V-App	1 and 2

Cluster Name	App Name	Phase (1or 2)
Greek	EEaaS Dashboard V-App	1 and 2

4.3.1 INUS Asset V-App

4.3.1.1 Overview

The INUS Asset V-App is the primary visualisation and operational interface of the Intelligent UxV Surveillance (INUS) system within the Greek cluster. It provides a unified platform for the real-time monitoring, visualisation, and management of surveillance assets, including UAVs, ground robotic platforms, and fixed camera infrastructures. Its main functionalities include the monitoring of live video feeds, the visualisation of assets and detected events on an interactive map, and the provision of a unified dashboard through which operators can observe the overall system status. In this way, the V-App enhances situational awareness in critical scenarios such as search and rescue operations and wildfire monitoring, while interacting with the INUS AI-App for Detection and Tracking.

4.3.1.2 Architecture

The INUS Asset V-App adopts a modular microservice-based edge–cloud architecture in which the main functions are decomposed into interoperable components responsible for video streaming, event handling, authentication, monitoring, and data aggregation. At the edge, heterogeneous robotic assets, including UAVs and the robotic dog, interface with the system through platform-specific bridge components, such as ROS, MAVLink, and dedicated controller-side integration modules, while local video-processing services enable near-source handling of perception data before transmission to the cloud. This approach supports the integration of diverse autonomous platforms under a common operational framework, while reducing latency and bandwidth demand for time-critical data flows. The V-App also interfaces with the AI-App for Detection and Tracking by ingesting AI-generated detections, events, and alerts, and relies on the N-App for Coverage Extension to sustain connectivity and support the operation of assets deployed in harsh or communication-constrained environments, such as the robotic dog. At the backend, system coordination is centred around a lightweight messaging and service layer that combines RESTful APIs with a broker-based internal communication backbone, enabling the exchange of telemetry, processed data, events, and commands across distributed services. For real-time video delivery, the architecture leverages WebRTC to provide low-latency streaming from the UAV and the robotic dog to the frontend. Operators interact with the system through a web-based dashboard that offers unified access to live video feeds, asset and event visualisation on an interactive map, and system-level operational information. Supporting services include [Keycloak](#) for authentication and access control, data-model translation modules for harmonising information from heterogeneous sources, and a monitoring stack based on Prometheus and Grafana for system observability. Overall, the architecture reflects a scalable and service-oriented design that enables interoperable multi-robot supervision, secure data exchange, real-time situational awareness, and efficient integration of AI and networking enablers within the INUS framework.

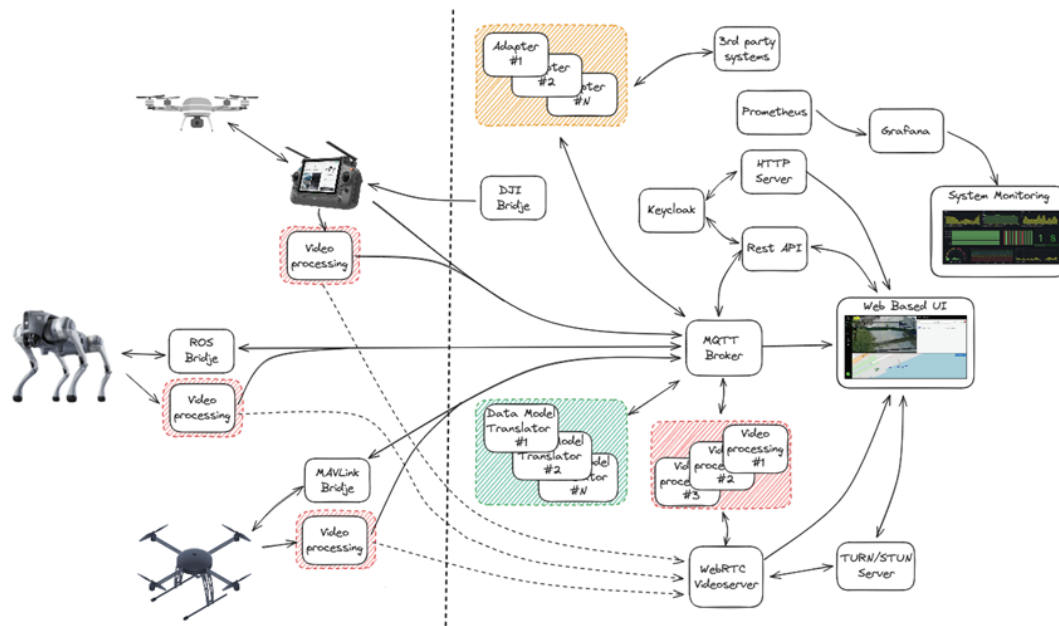


Figure 40 INUS Asset V-App architecture

4.3.1.3 Development, Deployment and Results

The INUS Asset V-App frontend has been developed as a web-based user environment composed of two complementary graphical interfaces, each addressing a distinct operational need within the system. The first interface is based on a Grafana–Prometheus combination and is dedicated to system monitoring, performance supervision, and KPI recording. Through this environment, operators and technical users can observe the health and status of the deployed components, monitor resource utilisation and operational metrics, and systematically collect the indicators required for performance assessment and evaluation activities. This monitoring layer therefore provides both real-time awareness of the technical operation of the platform and a structured basis for assessing system behaviour during trials and demonstrations. The second interface constitutes the main operational user interface of INUS and has been developed in React.js as a modern, responsive web application. It is hosted through an Nginx server and integrated with Keycloak for secure user authentication and access control. Through this interface, authorised users can interact with the surveillance system, issue tasking commands to the available assets, and query their operational status via a REST API implemented in FastAPI/Python. In parallel, the interface is connected to an MQTT server for the reception of telemetry and intelligence-related information, enabling the presentation of near real-time updates from the deployed assets and subsystems. For live visual situational awareness, the interface also integrates with a WebRTC server to support low-latency real-time video streaming from the surveillance platforms.

The INUS Asset V-App is deployed as a containerised application following a cloud-native approach. Depending on the operational scenario and the required system configuration, deployment can be performed using either single-host orchestration mechanisms, such as Docker Compose, or multi-host orchestration platforms, such as Kubernetes, thereby supporting both centralized and distributed deployment models. In particular, the Kubernetes-based deployment aligns with cloud-native principles and enables modular service orchestration, resilience, and scalability. The microservices composing the V-App are packaged and deployed independently, facilitating maintenance, lifecycle management, and future extensibility. This deployment approach also supports horizontal scaling, allowing the system to accommodate an increasing number of connected assets and simultaneous video streams as operational demands evolve.

The development and deployment of the INUS Asset V-App resulted in an operational platform capable of supporting the real-time monitoring and management of heterogeneous surveillance assets within the Greek cluster. As seen in Figure 41, the V-App provides a main dashboard that integrates live video streams, an interactive map for the visualisation of assets and events, and system-level monitoring information in a unified

operator-facing environment. It presents detected events and alerts while relying on the underlying connectivity support required for stable communications and streaming. The use of WebRTC enables near real-time visual feedback from a UAV and a robotic dog, both equipped with cameras, which is particularly important for time-sensitive operations such as geo-fencing monitoring and search and rescue. In parallel, the integration of Grafana supports the observation of key operational metrics, including active streams, service health, and resource utilisation. The main functionalities of the V-App were validated during the 1st Greek trial, where the platform was used as the main interface for monitoring live asset status, video feeds, and operational events.

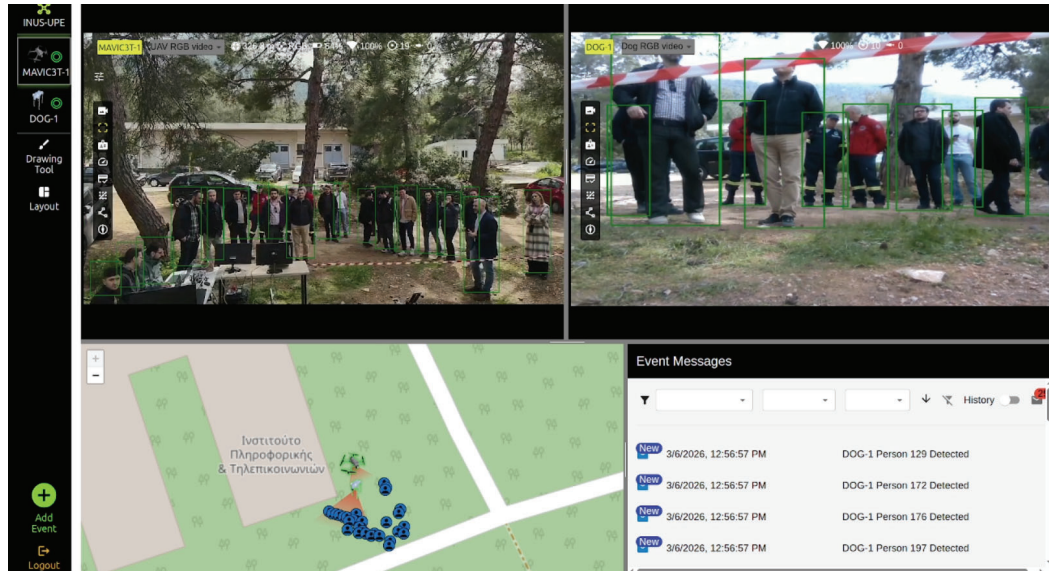


Figure 41 INUS V-App main dashboard with interactive map and live video streams validated during 1st trial

4.3.2 EEaaS Dashboard V-App

4.3.2.1 Overview

The EEaaS Dashboard V-App provides the visual front-end for the Energy Efficiency as a Service functionality of the Greek Pilot. It retrieves energy-related monitoring data from the N-App EEaaS server and presents them through dashboard-based visualisations for the end user. Its main role is to support real-time and historical observability of energy consumption across distributed devices and software components, including per-device and system-level views. In the current design, the V-App is realised as a dashboarding layer on top of the central metrics backend and is intended to support operators, experimenters, and developers during the monitoring and assessment of the pilot's energy behaviour.

4.3.2.2 Architecture

This V-App is positioned above the N-App EEaaS server and acts as the visualisation layer of the overall EEaaS solution. Energy metrics are first collected at device level by the N-App EEaaS clients and then aggregated centrally by the N-App server. The V-App retrieves the relevant time-series information from this central backend and presents them through dashboards to the end user. In the current implementation approach, the dashboard layer is realised through Grafana-based visual components, while the central metrics backend is provided by the N-App server. The V-App does not directly interact with the target devices. Instead, it consumes the data already exposed by the N-App backend.

4.3.2.3 Development, Deployment and Results

The EEaaS Dashboard V-App has been developed as the visualisation component of the broader N-App EEaaS framework. Its implementation relies on dashboard-oriented technologies, with Grafana used as the main tool

for building visual panels and presenting monitored metrics in an accessible way. Through this approach, the V-App can display energy-related information already collected and aggregated by the N-App backend. The development has focused on defining the appropriate dashboard structure, selecting the relevant metrics to be displayed, and ensuring that the visual layer can support both per-device and system-level inspection of energy consumption.

The EEaaS Dashboard V-App is a containerised application and can be instantiated in a controlled and portable manner within the testbed environment. The V-App is deployed at the infrastructure side rather than on the field devices, since its role is to provide centralised visualisation of the data already aggregated by the N-App EEaaS server.

At the current stage, the **EEaaS Dashboard V-App** has reached an initial prototype level used for trial 1, to be also used for trial 2. The current result is the availability of a first dashboard-based visualisation approach for presenting energy monitoring data collected through the EEaaS framework as depicted in Figure 42.



Figure 42 EEaaS V-App Energy Monitoring Dashboard

4.3.3 Wildfire Detection Control Room V-App

4.3.3.1 Overview

The Wildfire Detection Control Room V-App is a web-based operational interface designed to support real-time early wildfire monitoring and operator decision-making. The application receives smoke detection alerts generated by the AI-App for Early Wildfire Monitoring and visualizes them through a unified control room dashboard. Key functionalities include live smoke alert reception, interactive map-based visualization of deployed cameras and detected incidents, historical event galleries, camera status monitoring, and contextual operational layers such as weather information, NASA FIRMS satellite fire zones, and detection statistics. The V-App leverages the images and videos already generated by the AI-App and stored on NAS-backed volumes, enabling direct access to incident evidence while avoiding redundant processing.

4.3.3.2 Architecture

The architecture of the Wildfire Detection Control Room V-App, presented in Figure 43, is designed as a modular containerized service, integrating real-time alert reception, visualization, and historical event access into a unified operational control room platform. The backend is implemented as a Python Flask multi-process web application, where dedicated internal services handle MQTT event subscription, camera registry synchronization, NASA FIRMS fire data updates, snapshot processing, and web GUI serving. Real-time updates are delivered to browser clients through Server-Sent Events (SSE) and Socket.IO, enabling immediate propagation of newly received wildfire alerts to the control room interface. The frontend is implemented using

Jinja2-rendered HTML templates and vanilla JavaScript, with Leaflet-based interactive maps, Chart.js dashboards, and live alert components. The V-App subscribes to the MQTT broker in order to receive smoke detection events from the AI-App in real time. In parallel, the same MQTT messages are persisted by the cluster's InfluxDB service and visualized through Grafana dashboards for operational monitoring and statistics. The images and videos produced by the AI-App are stored on NAS-backed volumes, which are directly accessed by the V-App for event visualization and historical evidence browsing.

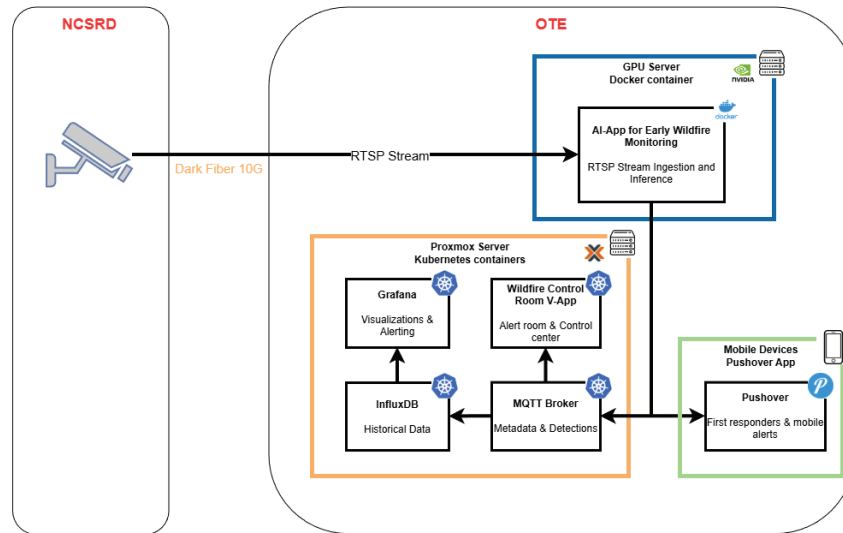


Figure 43 The architecture of the Wildfire Detection Control Room V-App

4.3.3.3 Development-Deployment and Results

The Wildfire Detection Control Room V-App was developed as a Python 3.9-based web platform, using the Flask framework (<https://flask.palletsprojects.com/en/stable/>) as the core backend technology and packaged as a Dockerized multi-service application. The implementation follows a modular service-oriented approach inside a single container, where multiple coordinated Python processes are launched through a centralized endpoint script. This design was selected to separate responsibilities while maintaining simplified deployment and operational management.

At implementation level, the backend was developed around five main functional service groups:

1. Real-time event ingestion services, responsible for subscribing to the MQTT broker and continuously receiving smoke detection alerts generated by the AI-App.
2. Camera and site registry services, which dynamically maintain camera metadata, coordinates, and configuration records used by the map-based GUI.
3. Background data enrichment services, which periodically retrieve and preprocess contextual wildfire information such as NASA FIRMS (https://firms.modaps.eosdis.nasa.gov/active_fire/) satellite fire datasets, weather data, and derived risk layers.
4. Snapshot and media access services, which resolve the NAS-backed (https://en.wikipedia.org/wiki/Network-attached_storage) storage paths of event images and videos produced by the AI-App and make them directly accessible to the web interface.
5. Web visualization services, which expose the main operator GUI, REST endpoints, and real-time update channels through Server-Sent Events (SSE) and Socket.IO, enabling immediate propagation of new alerts to connected browsers.

The backend stack uses Flask (<https://flask.palletsprojects.com/en/stable/>), Flask-MQTT (<https://flask-mqtt.readthedocs.io/en/latest/>), Flask-SocketIO (<https://flask-socketio.readthedocs.io/en/latest/>), Flask-CORS (<https://pypi.org/project/flask-cors/>), paho-mqtt (<https://pypi.org/project/paho-mqtt/>), pandas (<https://pypi.org/project/pandas/>), NumPy (<https://numpy.org/>), OpenCV (<https://pypi.org/project/opencv-python/>), and Pillow (<https://pypi.org/project/pillow/>), supporting event ingestion, analytics preprocessing,

media manipulation, and live UI synchronization. The frontend was implemented using Jinja2 (<https://jinja.palletsprojects.com/en/stable/templates/>) templates, vanilla JavaScript, the Leaflet (<https://leafletjs.com/reference.html>) interactive mapping library for GIS visualization, and Chart.js for operational dashboards and statistics.

For containerization and reproducible execution, the V-App was packaged with Docker (<https://docs.docker.com/reference/>), exposing the web application and dynamic service ports. The final deployment artifact is built from a production-ready Docker image and later deployed on the Kubernetes-based 6G-VERSUS infrastructure. The implementation process followed an incremental feature-driven development workflow, where the V-App was initially built around the core MQTT alert ingestion and map visualization pipeline, and subsequently extended with historical event browsing, NAS-backed multimedia access, contextual wildfire layers, and operational dashboards. Particular emphasis was placed on the end-to-end data flow between the AI-App, MQTT broker, NAS storage, and browser clients, ensuring that each new smoke detection event could be received, enriched with contextual metadata, resolved to the corresponding multimedia evidence, and rendered to operators with minimal latency. The frontend and backend components were developed iteratively in parallel, allowing rapid validation of new visualization features directly against live wildfire event data collected from the pilot infrastructure. The source code is maintained internally; however, due to IPR protection and planned exploitation activities, the source code is not uploaded to the shared GitLab repository. The implementation is currently maintained in an internal controlled-access development repository and container registry, ensuring version traceability while preserving exploitation-sensitive intellectual property.

The Wildfire Detection Control Room V-App was deployed as a containerized application on the 6G-VERSUS Kubernetes testbed infrastructure. The application runs as the deployment `v-app-wildfire-vertical-app` with one active replica in the default namespace. The container image used is `172.16.0.20:5005/dev-projects/wildfire-v-app/wildfire-detection-webgui:latest`. The V-App exposes two service ports: `5005/TCP → NodePort 30005`, `5006/TCP → NodePort 30006`. Persistent data and multimedia evidence are mounted through HostPath-backed NAS-style Kubernetes volumes, providing direct access to detected images, detected videos, retraining datasets, camera registry, historical SQLite/JSON data.

The V-App has been fully implemented and successfully tested during the NCSR Demokritos wildfire pilot trials, as evidenced in . The system demonstrated real-time reception of AI-App wildfire alerts, immediate visualization on the map-based control room interface, and seamless retrieval of associated image and video evidence. Historical event exploration, camera health monitoring, and contextual overlays were also successfully validated. The deployment on Kubernetes confirmed stable operation, multi-device accessibility, and high service availability, while integration with the co-located analytics stack enabled operational statistics and event monitoring.

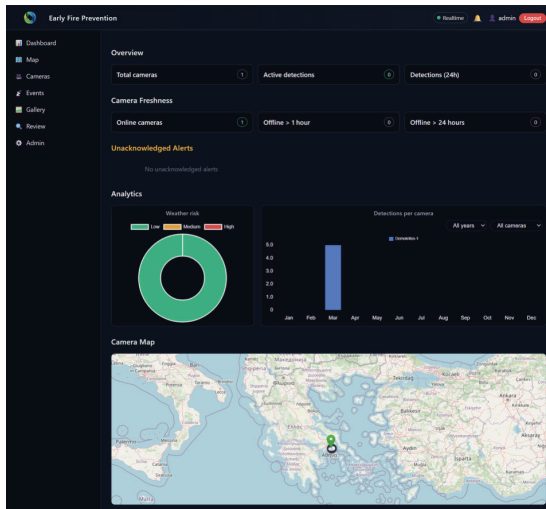


Figure 44 Dashboard view (Camera status, Detections statistics)

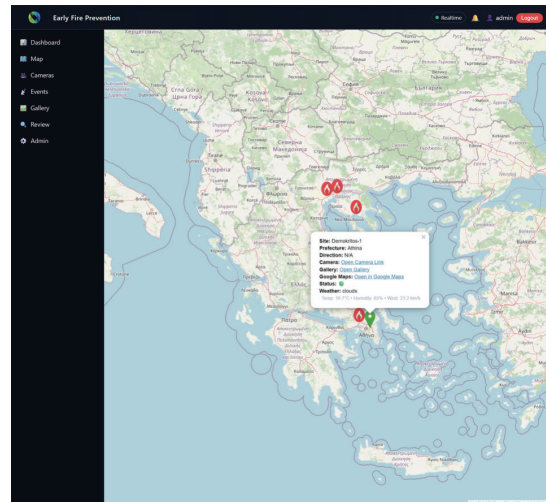


Figure 45 Map View (Info about the site, fire zones from NASA)

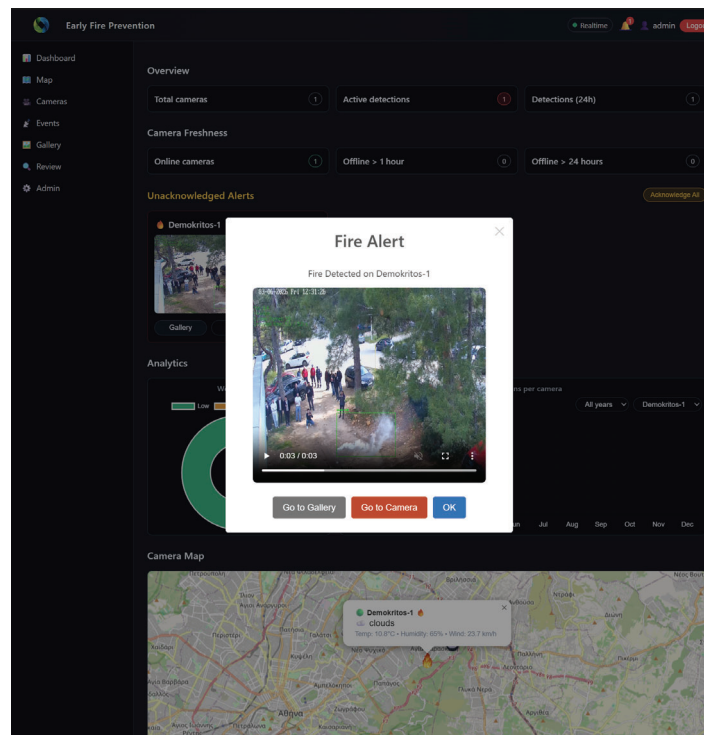


Figure 46 Pop up alert showing a detection came along with the detection video

4.3.4 V-Apps under Development for Phase 2

4.3.4.1 Wildfire Detection Control Room V-App for Phase 2

For Phase 2, the Wildfire Detection Control Room V-App is planned to be extended with direct interoperability with the N-App for QoS management, enabling the operator to select and apply network service profiles for the different components of the wildfire monitoring system. Through this enhancement, the V-App will act not only as a situational awareness dashboard but also as a control interface for service-level behavior, allowing operators to dynamically choose QoS profiles according to the operational context. In addition, the V-App will be enhanced with a detection review workflow, enabling operators to inspect generated wildfire alerts and

manually classify each event as true smoke, false positive with specific category e.g. Fog, clouds etc. The purpose of this review process is to create a continuously enriched validation dataset based on real operational detections, supporting the future improvement and optimization of the overall wildfire monitoring application through operator feedback and curated event categorization.

4.3.4.2 INUS Asset V-App for Phase 2

For Phase 2, the INUS Asset V-App will be further enhanced towards a more mature and fully integrated operational platform for the supervision and coordination of heterogeneous surveillance assets in the Greek pilot. Building on the initial prototype demonstrated in Phase 1, the next development stage will focus on strengthening the interoperability of the V-App with the AI-App for Detection and Tracking and with the N-App for Coverage Extension, so as to provide a more robust and adaptive operational environment for field deployments under demanding conditions. In particular, emphasis will be placed on improving the end-to-end handling of telemetry, detected events, alerts, and video streams, while ensuring smoother support for multi-asset operations and more reliable service continuity in challenging environments. In addition, Phase 2 development will focus on extending the operational capabilities of the web-based user environment, including improved asset tasking, richer event visualisation, and enhanced situational awareness features for operators. Further work will also target the hardening of the deployment model, the support of more distributed deployment configurations, and the scalability of the platform with respect to the number of connected assets and simultaneous video feeds. These developments aim to evolve the INUS Asset V-App from an initial operational prototype into a more robust, scalable, and interoperable V-App, ready to support the advanced trial activities planned for the second phase of the Greek pilot.

4.4 French Cluster

In the French cluster, we support two Use Cases (UC): UC1: Sewer Network Monitoring and UC2: Smart Watering in Green Spaces. This implies that multiple V-Apps (Table 23) are developed and currently under development to support both use cases. For Phase 1 (Trials 1 and 2), the cluster will showcase a set of mature V-Apps focused on environmental monitoring and anomaly detection. The V-App Environmental Indicators Dashboard will provide a high-level view of environmental benefits such as water savings and CO₂ reduction, supporting decision-making through clear visual panels, with a focus on UC2. The V-App Abnormal Watering Detection and the V-App Leakage Detection will generate alerts related to irregular irrigation behavior, including leaks and system malfunctions, helping operators quickly identify and address issues. In addition, the V-App B5G Connector (data plane) will ensure the collection of sensor data and device information for this phase. The V-App Soil Sensor will also be demonstrated, acting as a bridge between physical soil humidity sensors and the B5G modem interface to enable real-time data acquisition at the extreme edge. These applications are sufficiently developed and integrated to support the Phase 1 trials. For Phase 2 (Trials 3), more advanced visualization and analysis capabilities will be introduced through further development and integration. The V-App Alert Dashboard: Drilling, Mapping and Filtering will enhance alert management by allowing operators to analyse, filter, and prioritize alerts through multi-dimensional views, thereby reducing alert fatigue and improving operational efficiency. Additionally, further enhancements of the V-App Environmental Indicators Dashboard are planned to extend its functionality and coverage to UC1. The V-App B5G Connector will also be further integrated with the testbed and N-Apps to support not only the data plane but also connectivity status from the network layer, enabling seamless data flow across the system. These V-Apps are currently under development and will be progressively integrated and validated during Phase 2.

Table 23 Implemented and under development V-Apps in the French cluster

Cluster Name	App Name	Phase (1or 2)
French	V-App Environmental Indicators Dashboard	1 for UC2 and 2 FOR UC1
French	V-App Abnormal watering detection	1
French	V-App Alert Dashboard: Drilling, mapping and filtering	2

Cluster Name	App Name	Phase (1or 2)
French	V-App B5G Connector	1 for data plane and 2 for N-Apps integration
French	V-App Soil Sensor	1
French	V-App Level Sensor	2

4.4.1 V-App Environmental Indicators Dashboard UC2

4.4.1.1 Overview

V-App produces and displays environmental benefit indicators to quantify the positive impact of the system. These indicators can include water savings, CO2 emissions avoided, and other sustainability-related metrics. Presenting this data in an accessible way enables stakeholders to monitor performance, demonstrate environmental value, and support more informed operational and strategic decisions. The V-App is deployed in the cloud to ensure high availability and unique deployment site for all customers.

4.4.1.2 Architecture

The V-App is designed in alignment with the N-App architecture as seen in Figure 47, leveraging the 5G Data Plane as the backbone for data acquisition and processing. Field devices such as Soil Sensors transmit data through the 5G Data Plane, which interfaces with the V-App B5G Connector to ensure standardized data ingestion into the central Data Server. This server acts as the core component, hosting data storage, and the Environmental Calculation Engine responsible for computing indicators such as water savings and CO2 savings. These indicators are then exposed to multiple application modules, including the Environmental Indicators Dashboard.

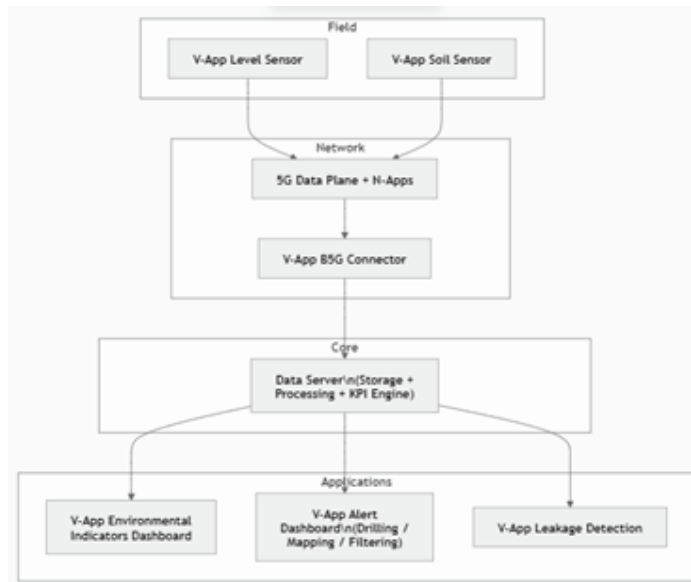


Figure 47 :Architecture of V-App Environmental Indicators

4.4.1.3 Development, Deployment and Results

The V-App was developed as a two-tier web solution composed of a Vue.js web application and a Node.js server-side calculation engine. The implementation approach was driven by the need to separate clearly the presentation layer from the business and computation logic. In practice, Vue.js was used to build a modular user interface with reusable components for dashboards, charts, filtering, alert visualization, and user navigation, while Node.js was used on the server to centralize data ingestion, processing, KPI calculation, and

exposure of results through APIs. This design makes the application easier to maintain, test, and evolve, because changes to environmental formulas or data-processing rules can be implemented on the server without impacting the front-end structure, and new views can be added in the web application without modifying the core calculation engine. The source code of the V-App has not been uploaded to GitLab due to confidentiality, security, and intellectual property constraints associated with the project. The V-App Environmental Indicators Dashboard is deployed on public cloud, the user interface of the dashboard is illustrated on Figure 48.



Figure 48 : V-App Environmental Indicators Dashboard

4.4.2 V-App Abnormal watering detection

4.4.2.1 Overview

The V-App “Abnormal Watering Detection” delivers alerts dashboard regarding abnormal watering. This may include leak detection, detection of configuration errors that could lead to reduced water savings, and detection of abnormal hydraulic behavior such as a valve failing to open or close. Within the triplet, the V-App collect measurement data from N-App and contextual or operational data from other V-App. The V-App is deployed in the cloud to ensure high availability and unique deployment site for all customers.

4.4.2.2 Architecture

The V-App Abnormal Watering Detection is implemented as a cloud-based application built around a Vue.js front-end and a Node.js server-side detection engine, and its architecture is aligned with the N-App template because it reuses the N-App data flow and keeps a clear separation between acquisition, processing, and visualization layers. In this design, the N-App / 5G Data Plane provides measurement data, the V-App B5G Connector handles secure ingestion, and the Node.js Data Server hosts the internal components required to operate the service: a Data Ingestion module to collect streams from N-App and other V-Apps, a Normalization and Context Fusion module to merge measurement, contextual, and operational data, an Abnormal Watering Detection Engine to apply rules and detection logic for leaks, configuration errors, and hydraulic anomalies such as valves failing to open or close, an Alert Manager to structure and prioritize incidents, and an API layer exposing results to the Vue.js Alert Dashboard, which provides mapping, filtering, drill-down, and operator views. This approach therefore follows the N-App template in principle, while making explicit the internal detection and alert-management components inside the Node.js server so that anomaly logic remains centralized, maintainable, and reusable across customers in a single high-availability cloud deployment. Global architecture is depicted in Figure 49.

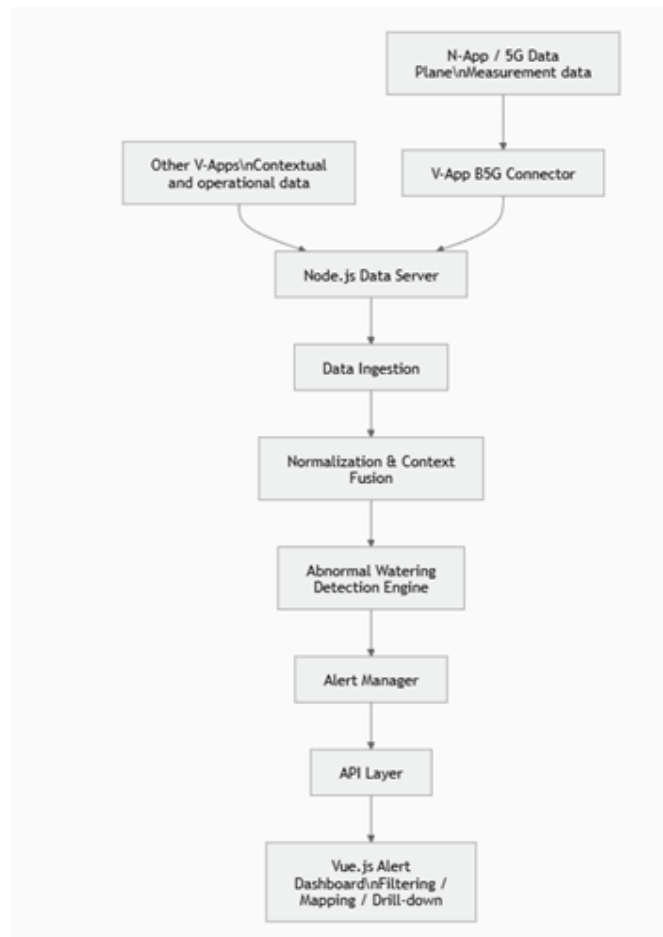


Figure 49 : V-App Abnormal Watering Detection

4.4.2.3 Development, Deployment and Results

The source code of the V-App has not been uploaded to GitLab due to confidentiality, security, and intellectual property constraints associated with the project. The V-App Abnormal Watering Detection was developed following a modular, service-oriented architecture based on a Vue.js front-end and a Node.js back-end. The implementation started with the definition of data flows aligned with the N-App ecosystem, identifying how measurement data from N-Apps and contextual data from other V-Apps would be collected, processed, and exposed.

- The architecture was designed to separate clearly:
- Data acquisition (via B5G Connector and APIs)
- Data processing and anomaly detection (Node.js services)
- Visualization and user interaction (Vue.js application)
- This separation allowed parallel development of front-end and back-end components and ensured scalability and maintainability from the beginning.

The V-App Environmental Indicators Dashboard, presented in Figure 50, is deployed on public cloud.

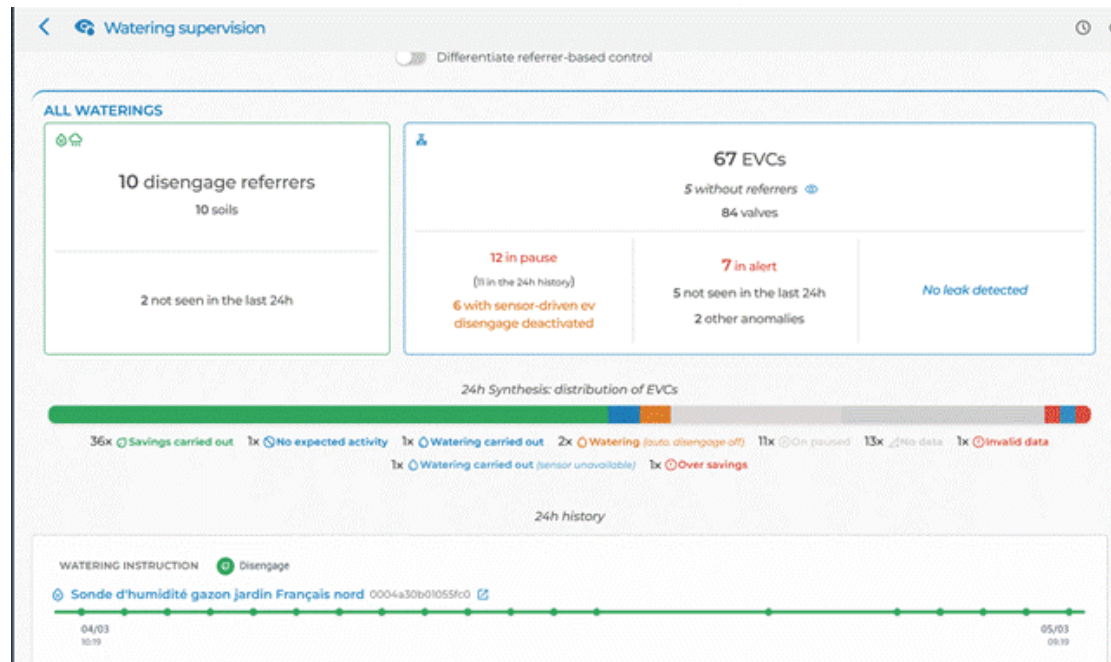


Figure 50 V-App Environmental Indicators Dashboard

4.4.3 V-App B5G connector

4.4.3.1 Overview

The V-App B5G Connector is implemented as a dedicated module within the Node.js back-end to interface with the N-App through the 5G Data Plane. Its primary role is to collect sensor measurements from the N-App, such as flow, level, or soil humidity data, and forward them to the V-App ingestion and processing pipeline. During the Phase 1 trial, the B5G Connector will be deployed in a public cloud environment, interfacing only with the Data Plane service, which will be the sole available component at this stage. As a result, the implementation will focus on reliable data acquisition, validation, and normalization of measurement data, enabling the abnormal watering detection engine and dashboards to operate on real-time inputs. At this stage, device connectivity status (B5G connection status) will not yet be available or displayed in the application. In Phase 2, the architecture will be extended to include additional capabilities. The V-App will become fully containerized (e.g., via Docker) to improve portability, scalability, and deployment automation. The B5G Connector will also be enhanced to collect and process device connection status information (e.g., online/offline state, last communication timestamp), which will then be exposed through the API layer and displayed in the devices list menu of the Vue.js application. This phased approach allows an incremental implementation: first validating the end-to-end data flow and anomaly detection logic using the available Data Plane service and then enriching the system with infrastructure-level monitoring and more advanced deployment capabilities in Phase 2.

4.4.3.2 Architecture

The B5G Connector App is implemented as a lightweight intermediary service, developed in Node.js, whose role is to interface the V-App ecosystem with the N-App Data Plane service and transform incoming field data into a format usable by the V-App back-end. Internally, the B5G Connector App will include a Data Plane Interface module to connect to the N-App service and receive sensor payloads, a Payload Parsing and Validation module to check message integrity and required fields, a Normalization module to harmonize formats, timestamps, and units, a Forwarding/API Client module to securely transmit processed data to the V-App Node.js server, and a Monitoring and Logging module to trace exchanges, detect failures, and support troubleshooting. In Phase 1, the connector will be deployed in the public cloud and will rely only on the available Data Plane service; in Phase 2, the design can be extended with device connection status handling

and containerized deployment to improve observability, portability, and operational scalability. Therefore, the architecture is not a deviation from the V-App template, but rather an implementation-focused adaptation of it, where the connector is isolated as a reusable gateway component dedicated to secure ingestion and protocol adaptation between the N-App environment and the V-App processing layer. Global architecture is depicted in Figure 51.

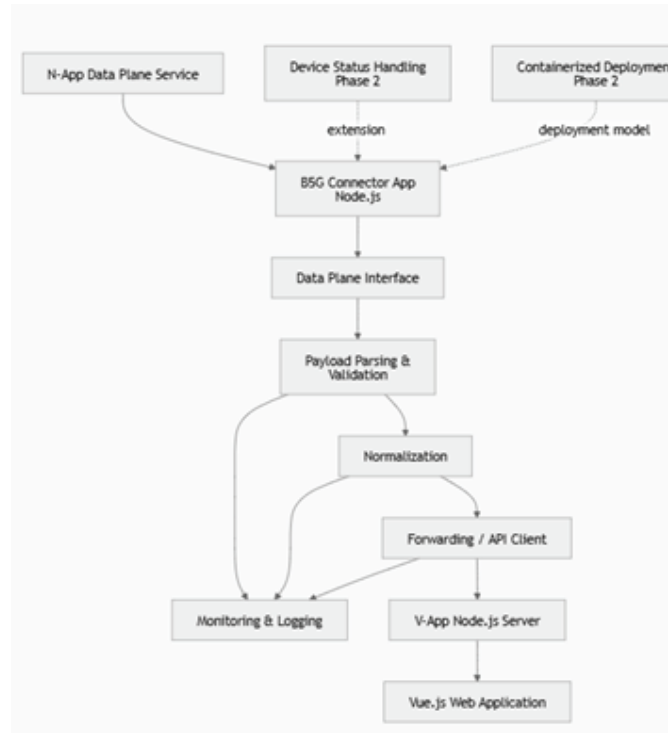


Figure 51 Architecture of B5G Connector V-App

4.4.3.3 Development, Deployment and Results

The source code of the V-App has not been uploaded to GitLab due to confidentiality, security, and intellectual property constraints associated with the project. The V-App was developed following a modular, service-oriented architecture based on a Vue.js front-end and a Node.js back-end. In phase 1, the V-App B5G connector is deployed on public cloud. While in phase 2 the V-App will be containerized and deployed on EURECOM test bed.

4.4.4 V-App soil and level sensor

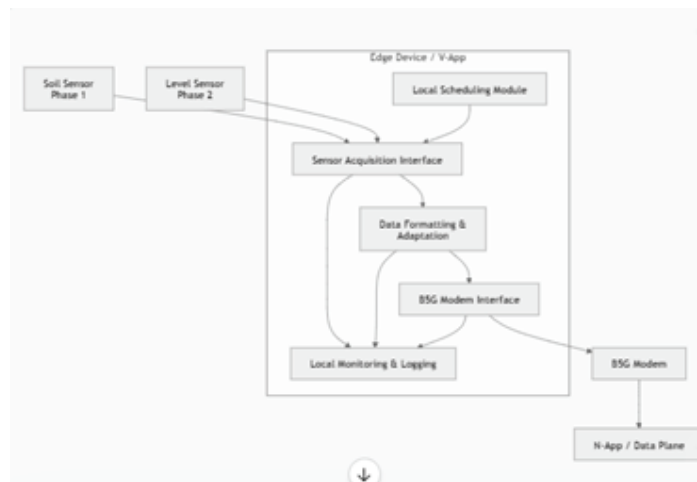
4.4.4.1 Overview

The V-App “Soil and Level Sensor” is designed as an embedded application deployed at the extreme edge, directly on the sensor devices themselves, in order to act as a bridge between the native sensor data acquisition system and the B5G modem interface of the EURECOM testbed. Its implementation allows the reuse of existing hardware and firmware capabilities, including sensor data acquisition, local scheduling, and measurement management, while extending the device to be compatible with the B5G communication environment. In Phase 1, the implementation focuses on the soil sensor, which is already available and integrated: the V-App collects soil measurements, formats them according to the expected data model, and transmits them through the B5G modem interface to the Data Plane. In Phase 2, the same approach is extended to the level sensor, which will be integrated once available, ensuring consistency in data handling and communication mechanisms across both device types. Internally, the V-App includes a Sensor Interface module to access raw measurements, a Data Formatting module to structure payloads, a Communication Interface module to interact with the B5G modem, and a Lightweight Scheduling module to manage

transmission timing. This edge deployment minimizes latency, reduces dependency on external processing, and ensures efficient, standardized data transmission from heterogeneous sensors into the N-App ecosystem.

4.4.4.2 Architecture

The V-App “Soil and Level Sensor” is implemented as an embedded edge application running directly on the sensor device itself, and its architecture is aligned with the N-App template in principle because it preserves the same separation between data acquisition, adaptation, and transmission, while being specialized for deployment at the extreme edge rather than in the cloud. Its role is to provide a bridge between the existing sensor functions and the B5G modem interface of the EURECOM testbed, so that current devices can be reused without redesigning their native acquisition logic. Internally, the application will include a Sensor Acquisition Interface to access measurements from the local soil or level sensing functions, a Local Scheduling Module to reuse the device’s existing acquisition and transmission timing, a Data Formatting and Adaptation Module to structure payloads in the format expected by the B5G environment, a B5G Modem Interface Module to manage communication with the modem, and a Local Monitoring and Logging Module to trace execution and detect communication issues. In Phase 1, the implementation covers the soil sensor, which is already available and can therefore be connected to the B5G chain first; in Phase 2, the same architecture is extended to the level sensor once it becomes available. This means the design does not deviate from the N-App template but rather adapts it to an embedded-device context where the application is deployed directly on the sensor node and acts as a lightweight gateway between native sensing functions and the B5G-enabled data path. Global architecture is depicted on Figure 52.



Figureⁱ 52 Architecture of V-App Soil and Level Sensor

4.4.4.3 Development, Deployment and Results

Figure 53 shows the V-App dashboard. The source code of the V-App has not been uploaded to GitLab due to confidentiality, security, and intellectual property constraints associated with the project. The V- was developed in C code over an STM32 platform. The V-App is deployed on the embedded STM32 platform. Figure 53 illustrates soil moisture measurement reported on dashboard.

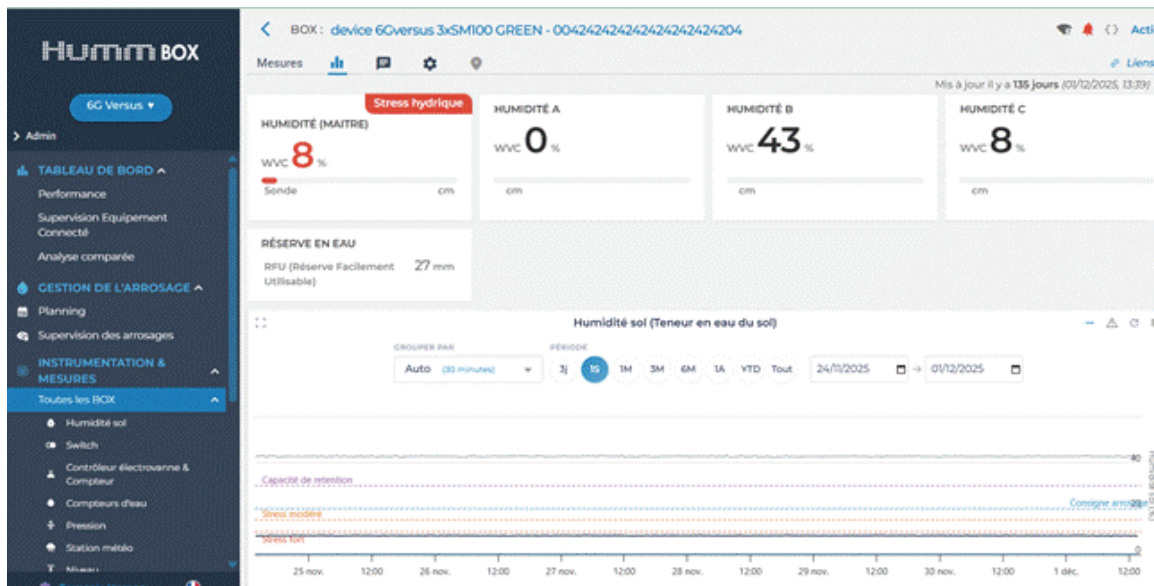


Figure 53 : Soil moisture sensor measurement reported in V-App

4.4.5 V-Apps under Development for Phase 2

4.4.5.1 Alert Dashboard: Drilling, mapping and filtering

The V-App “Alert Dashboard: Drilling, mapping and filtering” delivers tools to avoid alert fatigue by limiting the use of hard alert. Multi axis analysis tool and reducing the ratio of false alert. Multi-axis analysis tools allow assets to be positioned on decision maps rather than triggering hard decisions. Operators can then filter and prioritize alerts themselves. The V-App is deployed in the cloud to ensure high availability and unique deployment site for all customers. Within the triplet, the V-App collect measurement data from N-App and contextual or operational data from other V-App.

4.4.5.2 B5G Connector

The V-App B5G Connector collects sensor data and B5G device connection status from the N-App. The B5G connection status is displayed in the devices list menu. The V-App is containerized and deployed on the EURECOM testbed.

4.5 Portuguese Cluster

For Phase 1 (Trials 1, 2 and 3), the cluster will showcase the Port Environmental Monitoring V-App (Scenario 1), Mobitrust V-App (Scenario 1 and 2), Port Safety V-App (Scenario 2) and the Infrastructure Energy Tracking V-App (Scenario 3) (see Table 24). The purpose of the Port Environmental Monitoring V-App is to receive and display environmental quality data from diverse IoT Dataloggers, including water and air quality parameters. The Mobitrust V-App is a situational awareness application designed to collect, process, and transmit real-time multimedia and sensor data (video, audio, GPS, environmental and contextual parameters) from mobile assets such as drones, RC boats, cameras, and field devices. The Port Safety V-App represents a pre-gate control in place at a maritime port. Its function is to optimize the admission process of trucks and goods. The Infrastructure Energy Management V-App collects data from solar panels and EV chargers and also actuates on the EV chargers by increasing or decreasing the power to charge electric vehicles. It will also receive the location of the people devices from the N-App, in order to manage the turning on/off of the streetlights. For Phase 2 (Trials 4, 5, 6, and 7) the main goal is the consolidation of the V-Apps (e.g. accounting for additional triplet integrations).

Table 24 Implemented and under development V-Apps in the Portuguese cluster

Cluster Name	App Name	Phase (1 or 2)
--------------	----------	----------------

Portuguese	Port Environmental Monitoring V-App	Phase 1
Portuguese	Mobitrust V-App	Phase 1
Portuguese	Port Safety V-App	Phase 1
Portuguese	Infrastructure Energy Tracking V-App	Phase 1

4.5.1 Port Environmental Monitoring V-App

4.5.1.1 Overview

The Port Environmental Monitoring V-App is a software platform designed for the collection, processing, and visualisation of environmental data from IoT Dataloggers, supporting water and air quality parameters. The V-App enables real-time monitoring, automated device management, and intelligent energy- and network-aware operations, providing operators with a comprehensive view of environmental conditions and device performance. Its operation is complemented by supporting applications: the IoT Device Energy Management AI-App, which delivers intelligent configuration recommendations and data-driven insights, and the QoS N-App, which provides network state awareness to enable adaptive communication, optimise transmission scheduling, and improve energy efficiency.

4.5.1.2 Architecture

The architecture of the Port Environmental Monitoring V-App is designed as a modular containerised service, integrating real-time data ingestion, automated device management, and interactive visualisation into a unified operational platform. The system is structured around two primary components: the V-App Server, an edge-hosted backend that orchestrates all data processing and device control logic, and the V-App Client, a web-based frontend that provides operators with dashboards for monitoring and interaction. The V-App Server is responsible for ingesting and validating data transmitted by IoT Dataloggers, persisting measurements in a time-series database, and exposing RESTful APIs consumed by the V-App Client and external integrations. Automatic device management is executed at the server level, leveraging configuration recommendations from the AI-App and network state inputs from the QoS N-App to enable adaptive, energy-aware operation. The V-App Client retrieves data from the server APIs and renders live and historical dashboards, Datalogger status panels, and network KPI views directly in the browser.

4.5.1.3 Development-Deployment and Results

The Port Environmental Monitoring V-App was developed as a Python-based platform, using FastAPI as the core backend technology, and packaged as a Dockerised application. The implementation follows a modular service-oriented approach, where dedicated internal components handle data ingestion, validation, device management, database persistence, REST API serving, and real-time client updates. This design separates responsibilities while maintaining simplified deployment and operational management. At implementation level, the backend was developed around the following main functional service groups:

1. Data ingestion and validation services, responsible for receiving environmental measurements from IoT Dataloggers, verifying data integrity, and flagging anomalies or missing transmissions.
2. Device management services, which execute automatic configuration and control of IoT Dataloggers, incorporating recommendations from the AI-App and network state inputs from the QoS N-App.
3. Database persistence services, which store validated measurements in a time-series database, supporting efficient queries by timestamp, device identifier, and measurement type.
4. REST API services, which expose standardised endpoints to the V-App Client and external system components, enabling data access, configuration management, and event notifications.
5. Web visualisation services, which serve the V-App Client dashboard interface and deliver real-time updates to connected browsers.

For containerisation and reproducible execution, the V-App was packaged with Docker, exposing the web application and API service ports. The final deployment artifact is built from a production-ready Docker image and deployed on the 6G-VERSUS testbed infrastructure. The implementation followed an incremental feature-driven development workflow, where the V-App was initially built around the core data ingestion and time-series persistence pipeline, and subsequently extended with device management automation, AI-App and N-App integration, RESTful API exposure, and the web-based visualisation layer. Particular emphasis was placed on the end-to-end data flow between IoT Dataloggers, the server backend, and the browser client, ensuring that each new measurement could be received, validated, persisted, and rendered to operators with minimal latency. Figures 54 and 55 illustrate the system interaction.

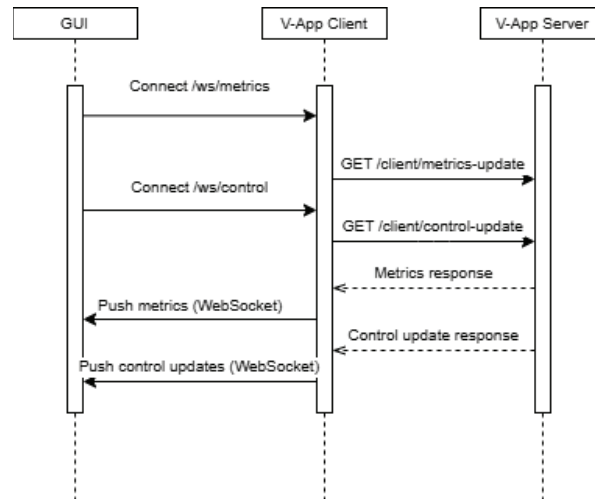


Figure 54 V-App Client Sequence Diagram

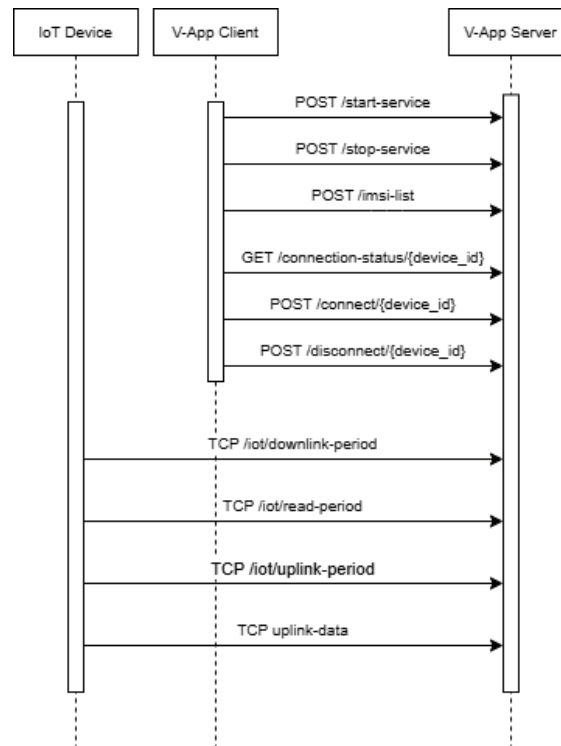


Figure 55 V-App Server Sequence Diagram

The Port Environmental Monitoring V-App code is available at <https://gitlab.6g-versus.eu/wp4/v-apps/portuguese-cluster>.

The Port Environmental Monitoring V-App is deployed as a containerized application on a dedicated VM within the IT Aveiro testbed infrastructure. The deployment is managed using Docker Compose, with the V-App Server as the primary service, built from a Dockerfile in the `src/` directory. It runs with an `unless-stopped` restart policy to ensure continuous availability and exposes port 6000/TCP for the REST API and backend services.

The server connects to its dependencies via environment variables, including InfluxDB v3 (for time-series data persistence) and the QoS N-App (for network state queries). Persistent data is mounted via a host-path volume,

binding the VM's /data directory to /app/data inside the container, enabling access to IoT Datalogger records, measurement archives, and historical data.

The V-App Server is connected to the v-app-net Docker network, shared with other 6G-VERSUS services on the same VM. A health check polls the /service-status endpoint every 30 seconds to monitor service availability. The broader deployment includes AI-App, API, InfluxDB v3, Nginx, QoS N-App, V-App Client, and V-App Server, each in its own directory.

The V-App has been fully implemented and successfully tested during the trials. The system demonstrated real-time ingestion and persistence of environmental measurements from IoT Dataloggers, correct execution of automatic device management operations informed by QoS N-App network state inputs, and accurate visualisation of live and historical data through the web-based dashboard. Datalogger status monitoring, alert detection, were successfully validated. The Figure 56 illustrates the web-based dashboard.

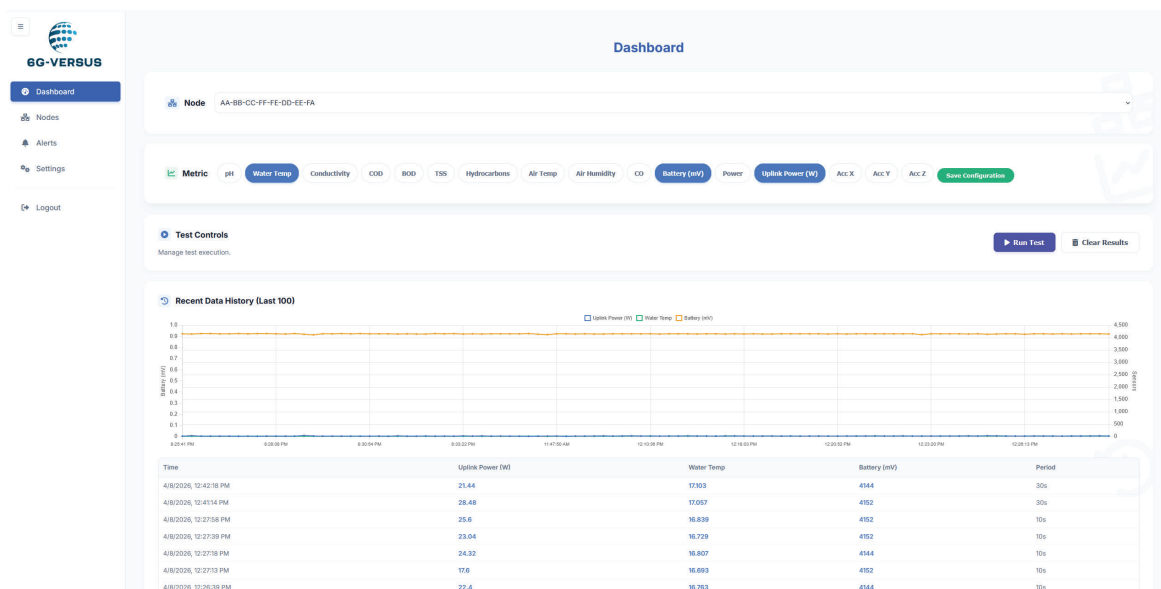


Figure 56 V-App web interface

4.5.2 Mobitrust V-App

4.5.2.1 Overview

The Mobitrust V-App is based on OneSource Mobitrust product. This platform is being integrated in 6G-VERSUS by adapting it, and integrating it, as a V-App. The scope is focused on the integration of the features provided by the triplet architecture.

4.5.2.2 Architecture

The Mobitrust platform will integrate the AI-App and N-App interfaces to enabled it's integration into the 6G-VERSUS architecture.

4.5.2.3 Development-Deployment and Results

The Mobitrust V-App scope has been integrated as part of the triplet architecture as a cloud-native application as described in the N-App section 3.6. It is depicted in Figure 57

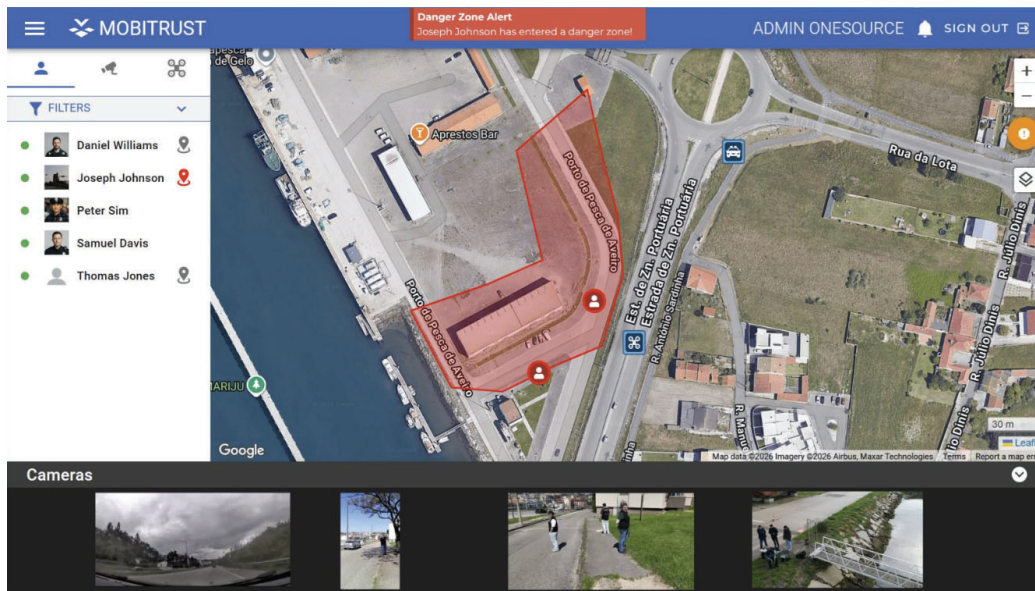


Figure 57 Mobitrust V-App dashboard in the Portuguese UC trial

4.5.3 Port Safety V-App

4.5.3.1 Overview

The V-App plays a pivotal role in the port safety scenario by controlling the interworking of the triplet components and overseeing the process evolution.

4.5.3.2 Architecture

The Port Safety V-App provides the business logic and the decision engine enabling the exploitation of the computer vision capabilities provided by the AI-Apps, and at the same time interworks with the Network Operations N-App for the dynamic control of the network resources. The V-App is notified by the AI-Apps through a REST API about vehicles approaching the port premises, including information that might be relevant to permit, or deny, the circulation through a specific route or apply special conditions associated to the vehicle cargo (e.g. hazardous materials). Based on the input received from the AI-Apps, upon the detection of a new vehicle, the Port Safety V-App is expected to trigger (through the Scaling Operations API exposed by the N-App) the reconfiguration of network resources needed to guarantee the proper upscaling of the image used to identify the vehicle approaching the port area (through the license plate) and respective cargo (through the HazMat symbols, if any). Upon the completion of this process, the Port Safety V-App is responsible for notifying the Network Operations N-App that the network resources can be reconfigured to return to the original “low-energy” state. The communication between the Port Safety V-App and the Network Operation N-App is supported by the REST API exposed by the N-App, as described in the previous sections. The integration of the Port Safety V-App with the Simple Object Detection System and HazMat Vehicles Detector AI-Apps (described also in the AI-APP section) is represented in Figure 58.

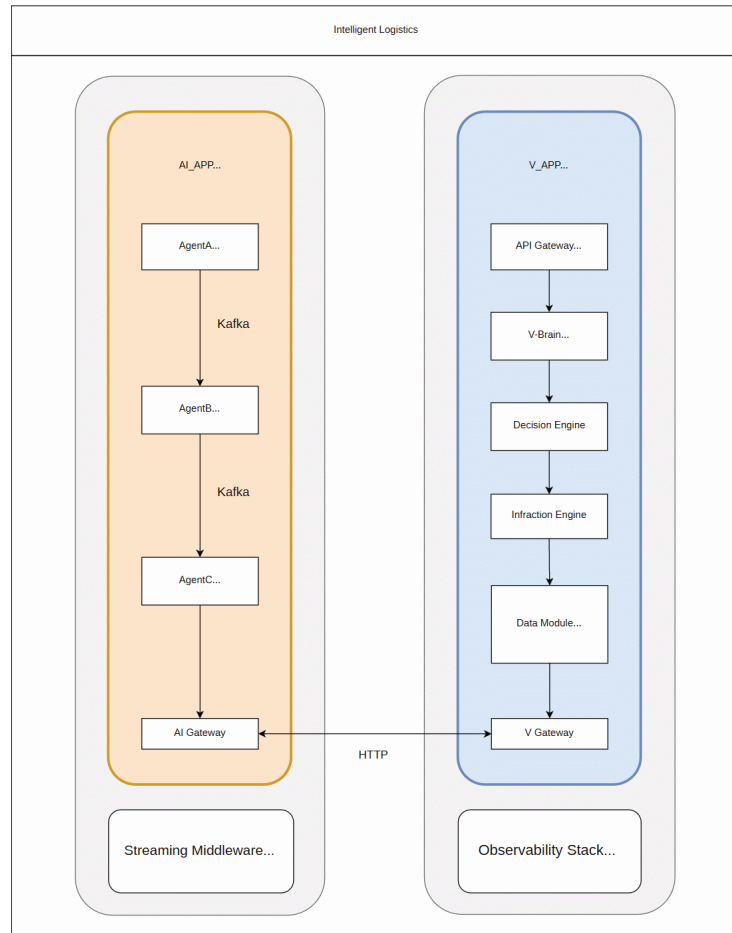


Figure 58 Port Safety V-App/AI-App integration

4.5.3.3 Development-Deployment and results

The latest version of the Port Safety V-App code is continuously made available at <https://gitlab.6g-versus.eu/wp4/v-apps/portuguese-cluster>, as it is being progressed. A detailed technical description of the code is included there. The Port Safety V-App is being deployed in a VM hosted at the ITAV data centre and successfully integrated with the Aveiro 6G-Versus testbed infrastructure through the Network Operations N-App. End-to-end testing was carried out both at ITAV premises and at the Port of Aveiro. The first public demonstration was included in the Portuguese Cluster Trial 2, which took place on 25/3/2026. The V-App dashboard (Figure 59) provides real-time information on the scenario status, including the video feed received from the cameras installed at checkpoints and safety-related information to support decision-making (e.g. detection of a truck carrying potentially dangerous cargo in forbidden areas adjacent to the port), as well as real-time RAN energy consumption.

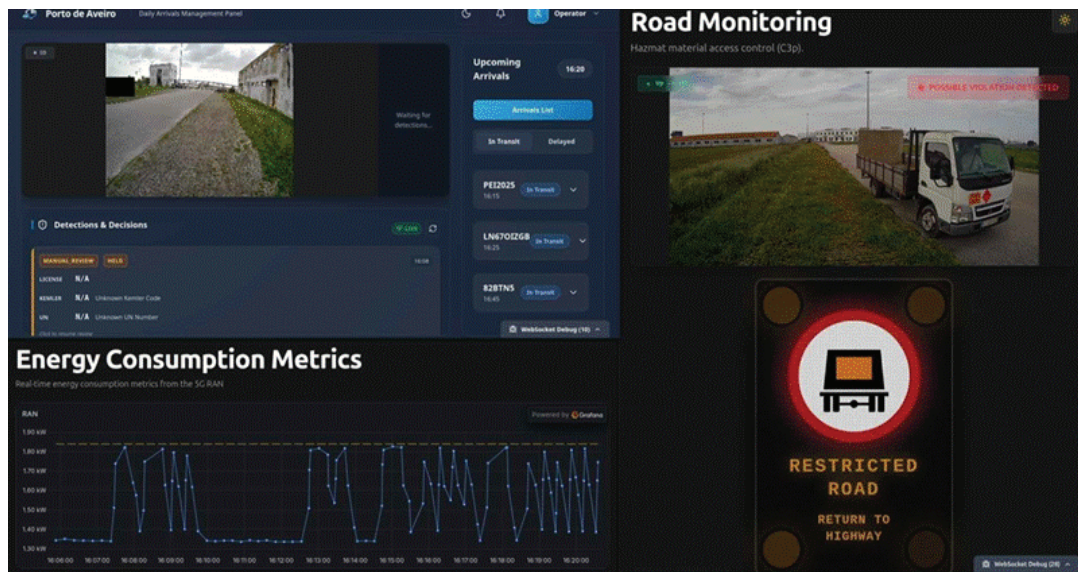


Figure 59 Port Safety V-App dashboard

Figure 60 illustrates the AI-App/V-App communication related to three relevant events in the scenario – truck detected, license plate results, hazard_plate results.

```
{
  "message_type": "truck_detected",
  "timestamp": 1776259527559,
  "confidence": "0.8535102367401123",
  "num_detections": 1
}

{
  "message_type": "license_plate_results",
  "timestamp": 1776259527568,
  "license_plate": "87AX60",
  "crop_url":
    "http://10.255.32.70:9000/agentb-1/crops/TRK72611659_1776259527.jpg?X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Credential=pei_user%2F20260415%2Fus-east-1%2Fs3%2Faws4_request&X-Amz-Date=20260415T132527Z&X-Amz-Expires=604800&X-Amz-SignedHeaders=host&X-Amz-Signature=ada9742c9f511da5ce6549aba51aa831f8b1f64dbe023e7d1518c8425221f47a",
  "confidence": "0.8216912378298847"
}

{
  "message_type": "hazard_plate_results",
  "timestamp": 1776259527568,
  "un": "1831",
  "kemler": "X886",
  "crop_url":
    "http://10.255.32.70:9000/agentc-2/crops/TRK28cda259_1776259496.jpg?X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Credential=pei_user%2F20260415%2Fus-east-1%2Fs3%2Faws4_request&X-Amz-Date=20260415T132456Z&X-Amz-Expires=604800&X-Amz-SignedHeaders=host&X-Amz-Signature=323f0e1a27cfb0f897baa03b1a98733ca258874f70e2a1f41a9e512c67347657",
  "confidence": "0.9187264635105326"
}
```

Figure 60 AI-App/V-App messages

4.5.4 Infrastructure Energy Tracking V-App

4.5.4.1 Overview

The Infrastructure Energy Tracking V-App is responsible for collecting, aggregating, and visualizing energy-related data from port infrastructure assets, namely solar panels and EV chargers. Its main functionality is to ingest real-time and historical data from these sources and present it through intuitive dashboards, providing visibility over energy production and consumption. This enables operators to monitor the performance of the

energy infrastructure and supports initial analysis of energy usage patterns. In addition, the V-App is designed to actuate on EV chargers by dynamically adjusting their charging power based on the energy production from solar panels, with the goal of maximizing the use of renewable energy. This allows for a more efficient and sustainable management of energy resources. The V-App thus serves as the application layer interface for both monitoring and control, with future extensions planned to further enhance optimization capabilities.

4.5.4.2 Architecture

The Infrastructure Energy Tracking V-App follows a modular architecture aligned with the overall application framework adopted in the project, ensuring interoperability with the N-App for network connectivity and data access. The V-App is composed of three main internal components: a data ingestion module, responsible for collecting real-time and historical data from solar panels and EV chargers; a processing and aggregation module, which normalizes and structures incoming data for analysis; and a visualization and control module, which exposes interactive dashboards and enables control actions on EV chargers. Communication with external systems is performed through API interfaces exposed by the underlying infrastructure, ensuring secure and reliable data exchange. The architecture is aligned with the general triplet-based design (N-App / V-App / AI-App), although in this implementation, the V-App operates independently from AI/ML components, focusing instead on energy monitoring and rule-based control logic.

4.5.4.3 Development, Deployment and Results

The Infrastructure Energy Tracking V-App was developed as a lightweight, modular application focused on energy data ingestion, processing, and visualization for port infrastructure assets. The implementation was carried out primarily in Go and Python, leveraging its ecosystem for data handling and API integration. The application was containerized using Docker to ensure portability and consistent deployment across environments, facilitating integration with the broader trial infrastructure. The backend logic was implemented in Go (<https://go.dev/>) to expose REST APIs for data querying and control operations, enabling interaction with external energy data sources such as solar panel monitoring systems and EV charging infrastructure. Data processing and aggregation were handled using Pandas (<https://pandas.pydata.org/>), allowing efficient transformation of time-series energy data. For visualization, a lightweight dashboard layer was implemented using Plotly Dash (<https://dash.plotly.com/>), providing real-time and historical views of energy consumption and production. The application was packaged using Docker (<https://www.docker.com/>), ensuring reproducible deployment and simplified integration with the ALB testbed environment. The V-App runs as a standalone service and communicates with external systems via REST APIs exposed by the infrastructure layer and the N-App when required. At the time of the first trial, a preliminary version of the dashboards was available, focusing on basic visualization of energy flows and initial validation of data ingestion pipelines.

The source code is available in the project GitLab repository: <https://gitlab.6g-versus.eu/gmachado/ai-ml-framework>

No additional AI/ML components were integrated in this V-App, as its scope is currently limited to energy tracking, visualization, and rule-based control of EV charging power. Future iterations will consider deeper integration with optimization and forecasting frameworks. The Infrastructure Energy Tracking V-App was deployed as a containerized application using Docker, ensuring portability and ease of deployment across different environments. The application was executed as an independent service within the ALB environment, prepared specifically to support the trial activities. The V-App was configured to ingest data from external sources, namely solar panels and EV chargers, enabling real-time data collection. It also provided preliminary dashboards to visualize energy production and consumption, supporting the demonstration activities. This deployment approach ensured flexibility and simplicity, allowing the validation of core functionalities without full integration into the testbed infrastructure, while maintaining readiness for future integration and scaling in subsequent project phases.

The deployment of the Infrastructure Energy Tracking V-App successfully validated its core functionalities, particularly in terms of data ingestion, processing, and visualization. The application was able to collect data from solar panels and EV chargers, confirming the correct integration with external data sources and the

reliability of the ingestion mechanisms. The V-App provided preliminary dashboards displaying energy production and consumption data, enabling basic monitoring of the infrastructure. These visualizations confirmed the correct aggregation and representation of the collected data, supporting initial analysis of energy usage patterns. Although still at an early stage, the dashboards demonstrated the feasibility of providing real-time visibility over energy-related metrics. Application logs confirmed the correct reception and handling of incoming data streams, including timestamps and energy values from the connected systems. These logs validated the stability of the application and the consistency of the data processing pipeline during execution. Representative examples of logs and dashboard screenshots are included to illustrate the system's behaviour and outputs. It should be noted that the current implementation focuses on monitoring capabilities, with control functionalities (e.g., dynamic adjustment of EV charging power) being not implemented. Overall, the results demonstrate a functional baseline for the V-App (see Figure 61 and Figure 62), which can be further extended with advanced control and optimization features in future development phases.

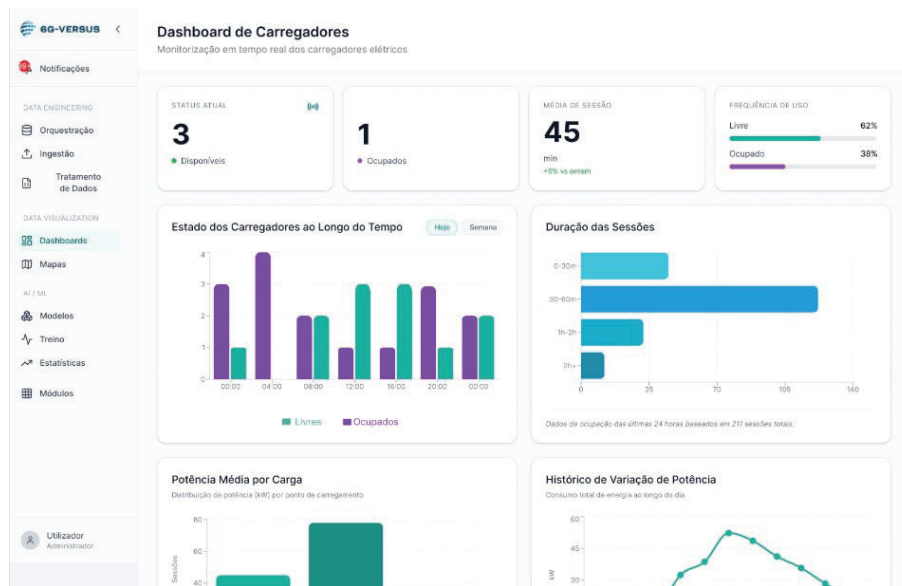


Figure 61 Snapshot of the Infrastructure Energy Tracking V-App dashboard

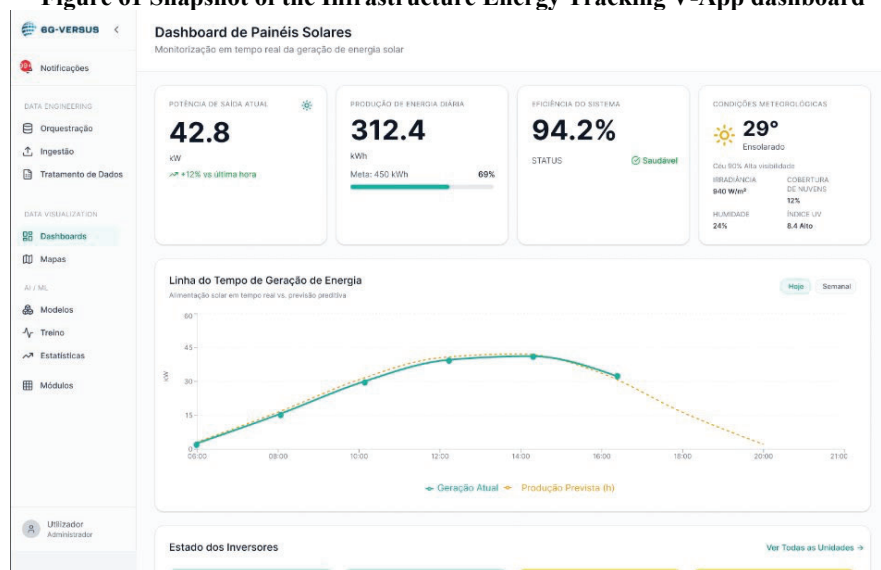


Figure 62 Snapshot of the Infrastructure Energy Tracking V-App dashboard for Solar panels

4.5.5 V-Apps under Development for Phase 2

4.5.5.1 Port Environmental Monitoring V-App

In the second phase, the primary focus will be on achieving full integration between the Port Environmental Monitoring V-App and the IoT Device Energy Management AI-App, establishing a tightly coupled feedback loop in which the V-App Server forwards environmental measurements to the AI-App for analysis and subsequently acts on the returned configuration recommendations to manage and control the IoT Dataloggers. In parallel, the data pipeline will be extended to retrieve a greater volume of measurements from the sensors already attached to the IoT Dataloggers, increasing the density and continuity of data available to the MLOps pipeline for continuous training and refinement of the AI-App's models.

4.5.5.2 Port Safety V-App

In phase 2 the Port Safety V-App will be enhanced with an optimization of robustness and performance. In addition, new application scenarios will be defined and eventually implemented (currently under study together with the Port of Aveiro).

4.5.5.3 Infrastructure Energy Tracking V-App

In the second phase, the main focus for the Infrastructure Energy Tracking V-App will be its integration with the AI-App and the AI/ML framework. This evolution aims to extend the current monitoring capabilities towards a more intelligent and automated system. The V-App will be enhanced to exchange data with the AI/ML framework, sending data from the solar panels and the EV Chargers such as production and consumption data, and receiving optimization outputs and recommended actions. These outputs, forwarded by the AI-App, will be used by the V-App to trigger control mechanisms, particularly in adjusting EV charging power and smart lighting behaviour. This integration will enable a closed-loop system, where data is continuously collected and processed. This data will be used to generate predictions and translated into actions, supporting real-time and energy-efficient operations.

4.6 Spanish Cluster

The Spanish Cluster includes several V-Apps developed and implemented across both trial phases. In Phase 1, some V-Apps are introduced as mock-ups or early trial components, while others are already demonstrated in specific trial scenarios. Their role is further expanded in Phase 2, where selected applications reach full implementation and are integrated into additional operational contexts. As presented in Table 25, the V-Apps of the Spanish Cluster cover water quality decision support, unmanned vehicle control, immersive AGV driving with autonomous missions, automatic surveillance, and reliable water treatment, reflecting the cluster's progressive contribution to the project's validation activities.

Table 25 Implemented and under development V-App in the Spanish cluster

Cluster Name	App Name	Phase
Spanish cluster	DSS for water quality V-App	Phase 1 & 2
Spanish cluster	Unmanned vehicle control V-App	Phase 1 & 2
Spanish cluster	Immersive driving of an AGV with autonomous missions V-App	Phase 1 & 2

Spanish cluster	Automatic surveillance V-App	Phase 2
Spanish cluster	DSS for reliable water treatment V-App	Phase 1 & 2

4.6.1 DSS for water quality V-App

4.6.1.1 Overview

DSS for water quality V-App is the visualisation and notification component of the water quality Decision Support System. Its main functions are to deliver early-warning alarms to farmers when water is detected as unfit for consumption, for example via email, and to provide a cloud-based user interface for displaying water-quality data, trends, and basic status indicators. The V-App supports timely decision-making by presenting current measurements together with historical context. Within the triplet, the AI-App triggers the V-App whenever predefined water-quality thresholds or anomaly conditions are met, enabling the V-App to notify end users. The application is deployed in the cloud in order to ensure high availability and easy access across different devices.

4.6.1.2 Architecture

The architecture of DSS for water quality V-App is centred on two main functions: user notification when the water quality decreases and it is not suitable for irrigation, and periodic water-quality information. The V-App receives alert triggers and related context from the AI-App whenever water-quality thresholds or anomaly conditions are met. Based on these triggers, the V-App activates the notification module to send early-warnings to farmers through supported communication channels such as email. Additionally, the V-App can be configured to send periodic information about the water quality (e.g., daily, weekly, etc.) so the farmers can have up-to-date information about the water they use for irrigation. The application architecture is aligned with the V-App role in the triplet, acting as the user-facing component that transforms AI-based assessments into understandable information and actionable warnings.

4.6.1.3 Development, Deployment and Results

The development of the DSS for water quality V-App focuses on implementing the user-facing components of the water quality DSS, including both early-warning alerts triggered when water quality deteriorates and periodic notifications providing updated information on water quality conditions (e.g., on a daily basis). A version of the early warning alarm module capable of sending dummy alarms has already been implemented, providing an initial basis for the notification workflow. The next steps in development will focus on integrating the V-App with the AI-App trigger mechanism, implementing the cloud interface for visualisation of current and historical water-quality data, and extending the notification functionality toward operational alert delivery. The V-App is intended to be developed as a cloud-deployed and containerized application to support modular integration within the triplet and accessibility across devices. GitLab link: As the application handles sensitive information, the software will remain in a restricted development environment and will not be uploaded to GitLab.

Technical details:

- Programming language: Python
- Notification channels: email
- Containerization: Docker
- Deployment model: Cloud

DSS for water quality V-App will be deployed as a containerized application in the cloud-based testbed infrastructure. This deployment approach supports portability, maintainability, and simplified integration with the other triplet components. At the current stage, the main development result for DSS for water quality V-

App is the implementation of an initial early-warning module capable of sending dummy alarms (see Figure 63). This prototype demonstrates the feasibility of the notification workflow and provides the foundation for future integration with the AI-App when available.

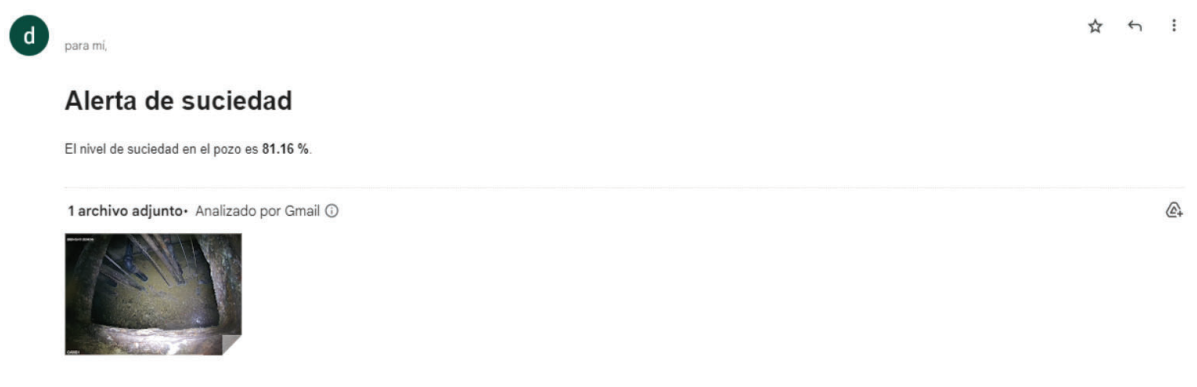


Figure 63 Sample alert email for plant process monitoring. The message includes the type of alert detected by the model and an image to help the plant operator make an informed decision on how to resolve the issue

4.6.2 Unmanned Vehicle control V-App

4.6.2.1 Overview

Note that this V-App was originally reported in D2.1 as “Drone remote control V-App”. However, the name has been updated to “Unmanned Vehicle control V-App” to reflect the broader applicability of the solution to different unmanned vehicle platforms. The Unmanned Vehicle control V-App is responsible for the remote control of the unmanned vehicle (UV) and its payloads. Its main functionality is to transmit commands that control the movement of the vehicle and the operation of attached devices, such as a robotic arm, from a remote location. In the current implementation, the V-App supports remote operation over a 5G connection and also includes a backup control path based on the vehicle remote controller. The application has been designed to support not only drones but also other unmanned vehicles, such as robotic dogs or similar platforms, thus motivating the change from the original D2.1 naming to the broader current name.

4.6.2.2 Architecture

The architecture of the Unmanned Vehicle control V-App is shown in Figure 64. It is based on a distributed teleoperation approach with components running on both the remote PC and the onboard computing platform of the unmanned vehicle. In the current implementation used for Trial 1, a teleoperation script is executed on the remote PC to send control commands through the 5G connection, while a complementary script runs on the Raspberry Pi onboard the drone to receive and process these commands and forward them to the appropriate vehicle or payload interfaces. The robotic arm is connected to the onboard platform through USB serial, while control signaling is handled using MAVLink channels in the main 5G-based operation mode. A backup connection is available through the Skydroid H16 remote controller, which was also used for robotic arm commands in the Trial 1 setup. The current implementation does not yet interact with the N-App. However, the architecture has been designed so that in the next version, planned for Trial 2, the V-App will cooperate with the N-App. For example, to assign specific 5QIs or assign certain traffic priorities for remote control of the unmanned vehicle and the robotic arm. Overall, the architecture follows the project V-App approach while adapting it to the specific requirements of teleoperation and payload control in unmanned vehicle scenarios.

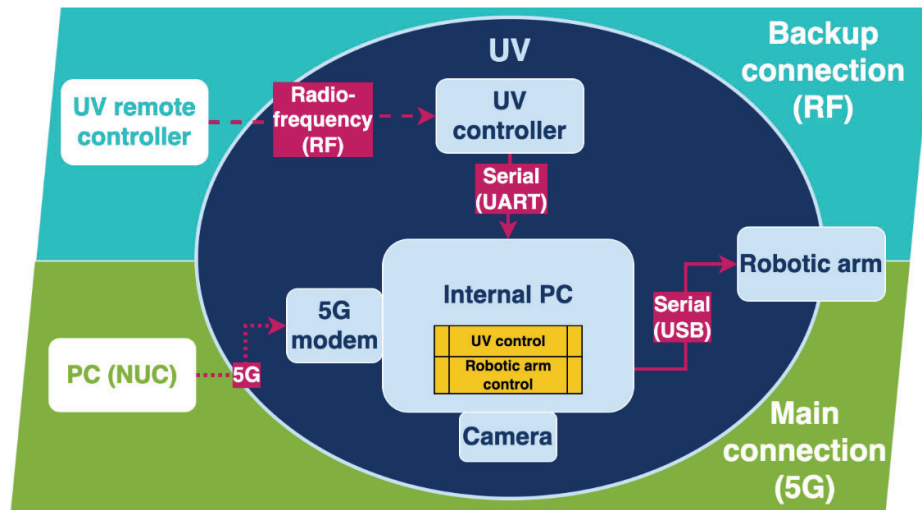


Figure 64. Unmanned Vehicle control V-App architecture

4.6.2.3 Development, Deployment and Results

The Unmanned Vehicle Control V-App has been implemented mainly in Python, with a smaller part developed in shell scripts. The version currently available in the repository corresponds to the implementation used in Trial 1, while further development is planned for Trial 2, which remains part of Phase 1 of the activity. The implementation follows a lightweight teleoperation approach: a script running on the remote PC handles the generation and transmission of commands for 5G-based remote operation, while a second script running on the Raspberry Pi onboard the drone manages the commands received either through the 5G channel or through the H16 remote controller backup mechanism. The communication with the platform and payload is based on the MAVLink protocol, particularly in the case where robotic arm commands are sent through MAVLink channels during 5G remote operation. In addition, the robotic arm is connected to the onboard Raspberry Pi through a USB serial interface, which is used to access and control the device from the onboard script. The current implementation was tailored to the drone platform used in Trial 1; however, the objective of the next development iteration is to generalize the solution so that the same V-App concept can be applied to a wider range of unmanned vehicles, including robotic dogs and similar platforms. The application code is maintained in the project GitLab repository and is available at the following link: <https://gitlab.6g-versus.eu/wp4/v-apps/spanish-cluster/unmanned-vehicle-control-v-app>. Regarding software packaging, the current implementation uses a Docker container on the Raspberry Pi onboard the drone. This containerized approach simplifies software installation, execution, portability, and integration in the project testbed. For the next version, the Docker-based packaging will be adapted to the final architecture envisaged for the broader unmanned vehicle scope and for the integration planned in Trial 2. At this stage, deployment is performed through manual container launch, without an orchestration framework.

The Unmanned Vehicle control V-App has been deployed as a containerized application in the testbed infrastructure. In the current setup, the deployment spans two execution points: (i) a remote PC, where the teleoperation script is executed for 5G-based remote operation, and (ii) the Raspberry Pi onboard the drone, where the main control logic is executed inside a Docker container. The onboard component is responsible for receiving commands from the remote side, managing the main 5G control path and the backup H16-based control path, and interfacing with the robotic arm through USB serial. At this stage, the deployment is performed through manual container launch. From the validation perspective, the system was deployed and tested in the environment used for Trial 1. In particular, the robotic arm functionality was tested in a lab environment corresponding to Scenario 3 (Ada Byron building) and demonstrated during Trial 1, which took place on 2nd February 2026. The current deployment corresponds to the Trial 1 implementation, and it will be further evolved for Trial 2, including adaptation to the final unmanned vehicle architecture and the planned interaction with the N-App for differentiated network treatment of teleoperation traffic.



Figure 65. Robotic arm integration with the drone



Figure 66. Robotic arm remote control via drone remote controller

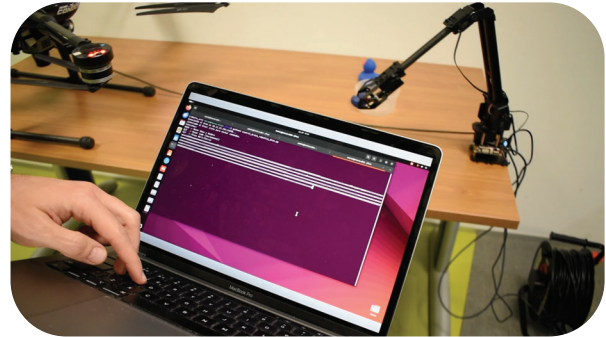


Figure 67. Robotic arm remote control via laptop

4.6.3 Immersive driving of an AGV with autonomous missions V-App

4.6.3.1 Overview

This V-App enables immersive remote driving of an Automated Guided Vehicle (AGV) over a 5G network. It provides real-time bidirectional communication between a remote driver and the AGV, allowing the operator to control the vehicle using a steering wheel and pedals while receiving live visual feedback through a virtual reality headset. The system integrates network-aware capabilities by dynamically retrieving the AGV's IP address via the N-App, ensuring seamless connectivity and adaptive configuration during operation. Additionally, the system supports monitoring by collecting operational metrics at the driver side and forwarding them to a Grafana dashboard.

4.6.3.2 Architecture

The V-App is designed as a two-part distributed system composed of a network integration module and an immersive remote driving system, as seen in Figure 68. The first module interacts with the N-App to retrieve the AGV's IP address based on its IMSI and dynamically updates the configuration file used during operation. The immersive remote driving system consists of two subsystems: a driver-side system running on a PC and an AGV-side system deployed on a NVIDIA Jetson Xavier. The AGV-side subsystem processes control commands received from the driver and captures visual data through a three-camera system. Both video streams and operational metrics are transmitted from the AGV to the driver-side system. The driver-side subsystem processes user inputs through a Node.js-based control application and renders immersive visual feedback using a Unity-based VR application. Additionally, it aggregates the operational metrics received from the AGV and forwards them to a Grafana dashboard for monitoring and visualization.

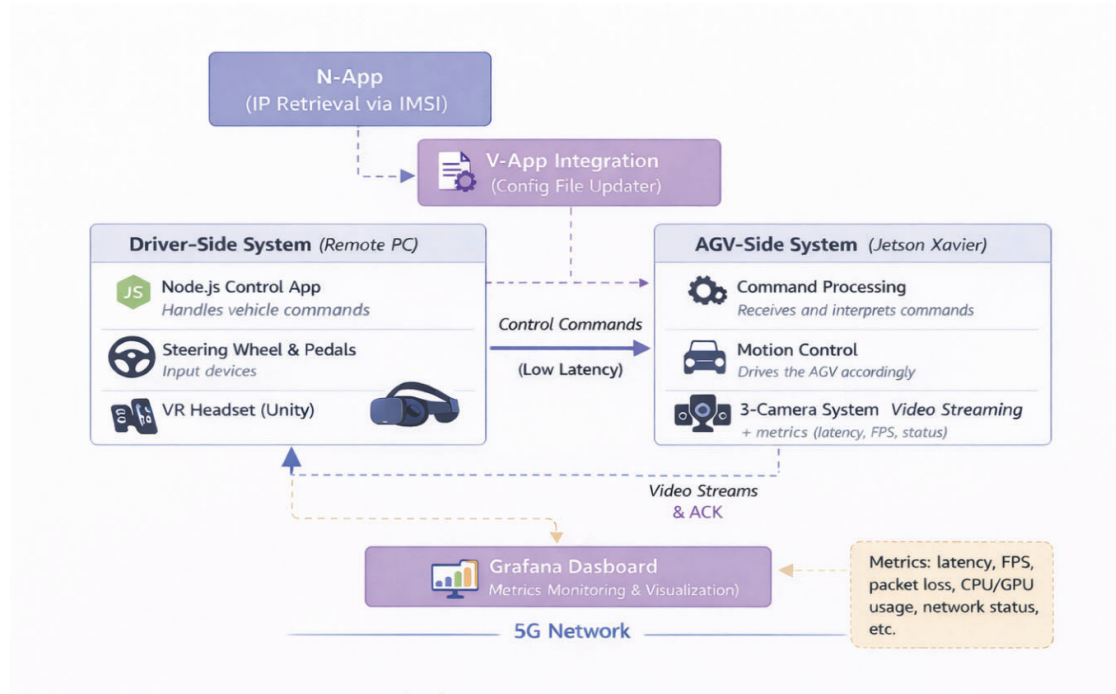


Figure 68 Immersive remote driving V-App Architecture

4.6.3.3 Development, Deployment and Results

The V-App has been developed as a distributed system combining multiple technologies. The control logic application on the driver side is implemented using Node.js, enabling real-time acquisition and processing of input data from the steering wheel and pedals. The immersive visualization is developed using Unity and deployed on a virtual reality headset to provide real-time feedback from the AGV cameras. On the AGV side, the system runs on an NVIDIA Jetson Xavier, where control commands are processed and translated into motion instructions for the vehicle. Additionally, a three-camera system captures video streams and system metrics, which are transmitted to the driver side. A dedicated software module has been implemented to interact with the N-App, retrieve the AGV IP address using the IMSI, and automatically update the configuration file, as shown in Figure 69 V-App logs during IP request. The system is containerized and designed to operate over a 5G core network. The code has not been uploaded to Git because a code non-sharing agreement was signed in accordance with company policy. The V-App has been deployed in the 5G testbed infrastructure at La Mayora. The system is executed as a set of applications distributed between the driver-side workstation and the AGV platform. The deployment leverages the 5G core network to enable low-latency communication between both subsystems. The integration with the N-App is fully operational, allowing dynamic retrieval of network configuration parameters during runtime. The monitoring component is deployed at the driver side, where metrics are collected and forwarded to the Grafana dashboard. The immersive remote driving system is fully operational at La Mayora. The integration between the V-App and the N-App has been successfully completed and tested, enabling dynamic IP configuration and seamless connectivity.

```

Receive petition IMSI: 001020000000251
Receive answer IP: 10.45.22.51

"VEHICLE_RD_IP": "0.0.0.0",
"VEHICLE_RD_PORT": "5002",
"USE_GRAFANA": "true",
"TELEGRAF_IP": "127.0.0.1",
"TELEGRAF_PORT": "8094",
"LOG_FILE": "logs/cockpit_log.json",
"LOG_LEVEL": "2",
"USE_BLOCKCHAIN": "true",
"RUN_GSTREAMER": "false",
"COMMAND_PERIOD_MS": "100",
"CAR_TYPE": "ROS"
}
joaquin@C-5CG4194PMN:~/temporal$ source venv_versus/bin/activate
(venv_versus) joaquin@C-5CG4194PMN:~/temporal$ python update_ip_client.py config_xrlab.json 001020000000251
IP address for IMSI 001020000000251: 10.45.22.51
Old IP: 0.0.0.0
New IP set to: 10.45.22.51
(venv_versus) joaquin@C-5CG4194PMN:~/temporal$ cat config_xrlab.json
{
  "COCKPIT_RD_IP": "192.168.250.101",
  "COCKPIT_RD_PORT": "5000",
  "UNITY_APP_IP": "127.0.0.1",
  "UNITY_APP_PORT": "8051",
  "VEHICLE_RD_IP": "10.45.22.51",
  "VEHICLE_RD_PORT": "5002",
  "USE_GRAFANA": "true",
  "TELEGRAF_IP": "127.0.0.1",
  "TELEGRAF_PORT": "8094",
  "LOG_FILE": "logs/cockpit_log.json",
  "LOG_LEVEL": "2",
  "USE_BLOCKCHAIN": "true",
  "RUN_GSTREAMER": "false",
  "COMMAND_PERIOD_MS": "100",
  "CAR_TYPE": "ROS"
}

```

Figure 69 V-App logs during IP request

The system demonstrates stable real-time control of the AGV and reliable transmission of multiple video streams and system metrics to the driver-side system. The driver-side application successfully forwards monitoring data to the Grafana dashboard, enabling real-time visualization of system performance.

Low-latency communication over the 5G network enables an immersive and responsive driving experience.

Video demonstration: <https://youtu.be/BINYJJE2iXA>

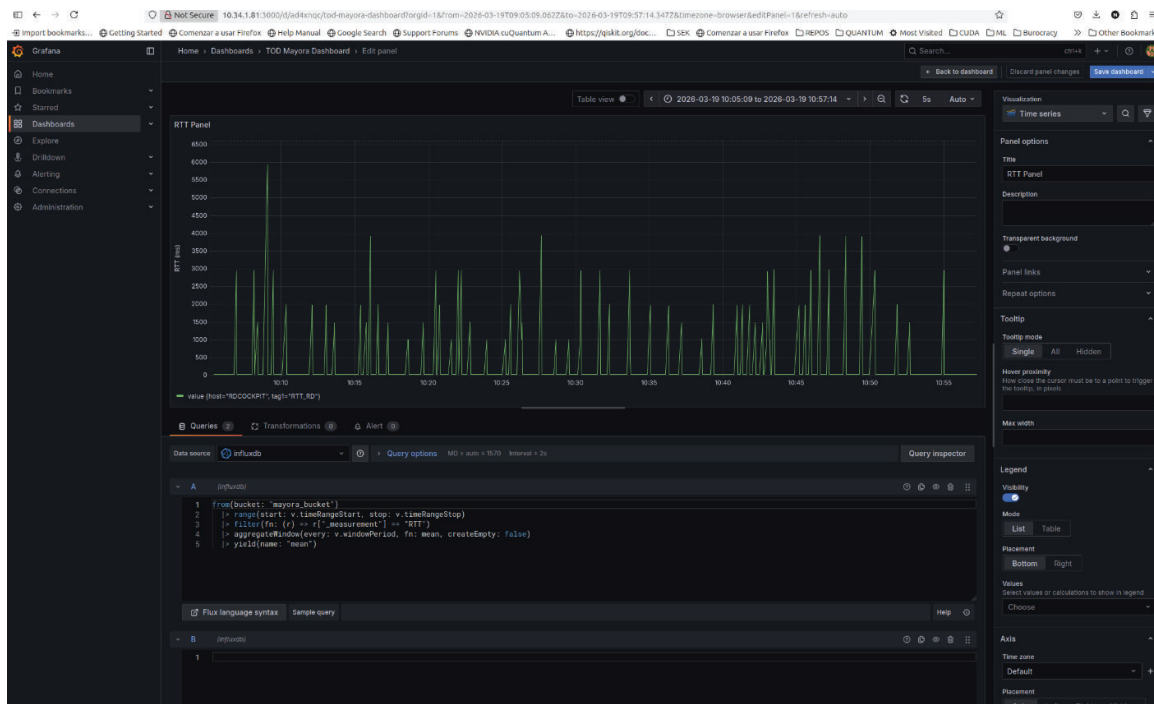


Figure 70 RTT measurements in V-App

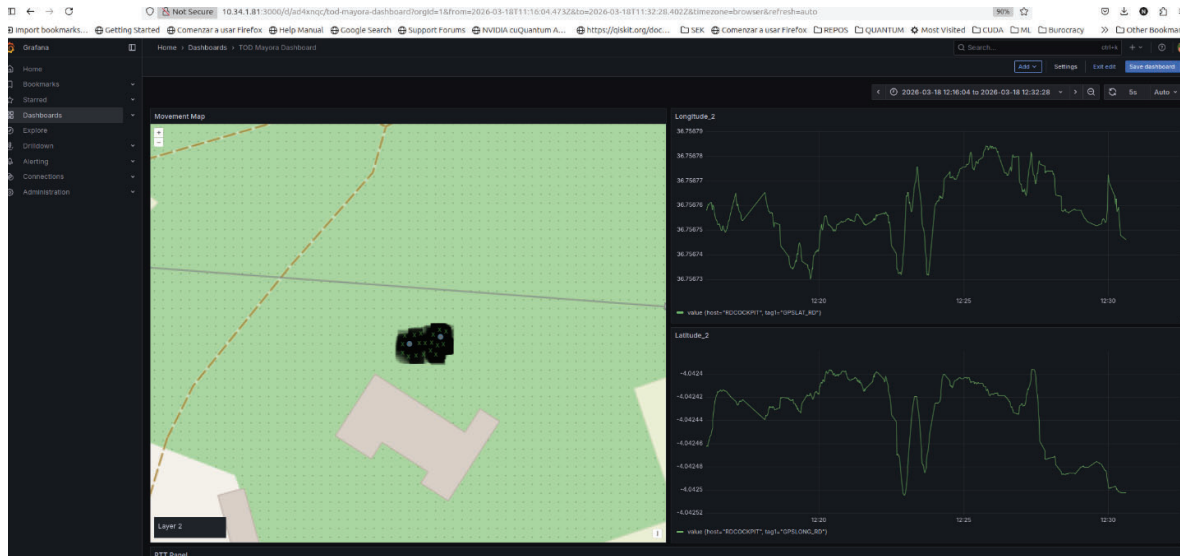


Figure 71 Location details in V-App

4.6.4 Automatic Surveillance & Detection in Farming Operation V-App

4.6.4.1 Overview

The Automatic Surveillance & Detection in Farming Operation V-App is responsible for automatically collecting and processing data from both the 5G network and the robotic platform in order to support surveillance and detection functionalities in farming environments. The V-App retrieves radio information from the 5G system, in particular Sounding Reference Signal (SRS) parameters, and combines it with LiDAR data collected from the robot. This information is then delivered to the AI-App, which performs the analysis required to identify anomalies, objects, or other events of interest for farming surveillance. The detection insights generated by the AI-App are received back by the V-App, which in turn triggers automated detection events. At the current stage, SRS and LiDAR data extraction are already operational, and the optimization of full-bandwidth radio parameter processing has been completed.

4.6.4.2 Architecture

The architecture of the Automatic Surveillance & Detection in Farming Operation V-App follows a modular data-integration approach aligned with the project V-App concept. A radio acquisition component retrieves SRS parameters from the 5G system, while a sensor acquisition component collects LiDAR data from the robot. Both data streams are then handled by the V-App processing layer, which prepares and forwards the information to the AI-App for surveillance and detection analysis. Once the AI-App generates the corresponding detection outputs, these are received back by the V-App through a dedicated interface, where they are translated into automated detection events for the farming operation workflow. Overall, the architecture is aligned with the project V-App template, as the V-App acts as the application-layer component that integrates heterogeneous data sources and interfaces with the AI-App, while the intelligence and event interpretation are distributed between both components according to their roles.

4.6.4.3 Development-Deployment and Results

The current development of the Automatic Surveillance & Detection in Farming Operation V-App focuses on the implementation of the data acquisition and integration pipeline required for surveillance and detection in farming scenarios. At this stage, the V-App already supports the extraction of SRS parameters from the 5G system and LiDAR parameters from the robot, which constitutes the core input required for the subsequent AI-based analysis. In parallel, the optimization of full-bandwidth radio parameter processing has been

completed, improving the readiness of the solution for broader and more demanding monitoring conditions. From the functional perspective, the V-App has been designed as the intermediary layer between the infrastructure and sensing domains on the one hand, and the AI-App on the other. In practice, this means that the V-App is responsible for retrieving the relevant radio and sensor data, organizing the information flow towards the AI-App, and receiving the corresponding detection insights back in order to trigger automated events. This modular split allows the sensing and communication interfaces to remain decoupled from the AI processing logic, which is beneficial for future extensions and integration with additional data sources. At the current stage, the V-App has reached functional readiness for SRS and LiDAR data extraction, while the final end-to-end deployment in the target testbed infrastructure is still ongoing. At this stage, the main result is that both SRS extraction from the 5G system and LiDAR extraction from the robot are operational. Final integration results will be shown on phase 2.

4.6.5 DSS for reliable water treatment V-App

4.6.5.1 Overview

DSS for reliable water treatment V-App is the user-facing notification and visualisation component of the Decision Support System for reliable water treatment. Its main functions are to send early-warning alarms to water treatment plant staff, for example via email, when a treatment process is detected as malfunctioning or operating outside expected conditions, and to provide a simple view of relevant process status indicators. Within the triplet, the AI-App triggers the V-App whenever anomalies, faults, or performance degradations are identified, enabling the V-App to notify plant operators and support faster intervention, more efficient plant management, and process optimisation. The application is deployed in the cloud, as it does not have strict latency requirements and benefits from scalable notification services and remote accessibility.

4.6.5.2 Architecture

The architecture of DSS for Reliable Water Treatment V-App is designed around two complementary functions: alarm delivery and simplified operational visualisation. The V-App receives triggers and contextual information from the AI-App whenever anomalies, malfunctions, or performance degradations are detected in the treatment process. Based on these triggers, the notification module sends early-warning alarms to plant operators through supported communication channels such as email. In parallel, the V-App can present a simple view of relevant process status indicators and computer vision outputs to improve operational awareness and facilitate faster interpretation of plant conditions. The application architecture is aligned with the V-App role in the triplet, as it acts as the user-facing layer that transforms AI-generated findings into actionable notifications and accessible process information.

4.6.5.3 Development, Deployment and Results

The development of DSS for reliable water treatment V-App focuses on implementing, in Python, the user-facing components of the reliable water treatment DSS, including early-warning notifications and simple process visualisation capabilities. A functional version of the early-warning module, capable of generating and sending dummy email notifications, has already been implemented. Each email notification includes an image associated with the detected event, enabling the plant operator to independently assess the situation and thereby supporting a human-in-the-loop verification process prior to any intervention. In addition, a mock dashboard for visualizing computer vision outputs has been developed. These initial implementations provide the basis for the future integration of the V-App with the AI-App, enabling the reception of alert triggers and the presentation of relevant process status information to operators. The next development steps will focus on connecting the notification workflow to the AI-App outputs, refining the visualisation layer, and preparing the application for deployment as a cloud-based and containerized service. GitLab link: As the application handles sensitive information, the software will remain in a restricted development environment and will not be uploaded to GitLab.

Technical details:

- Programming language: Python

- Notification channels: email
- Visualization component: dashboard for computer vision outputs and alarm system
- Containerization: Docker
- Deployment model: Cloud

DSS for reliable water treatment V-App will be deployed as a containerized application in the cloud-based testbed infrastructure. This deployment model supports portability, scalability, maintainability, and easy integration with the other triplet components. In the deployed setup, the V-App will receive alert triggers from the AI-App and provide notification services together with a simple visualisation interface for relevant process indicators and computer vision outputs (see Figure 72). Cloud deployment is appropriate for this use case because the application does not impose strict latency constraints and benefits from scalable notification services and remote accessibility for plant personnel. At the current stage, the main results achieved for DSS for reliable water treatment V-App are the implementation of a functional early-warning module capable of generating and sending dummy notifications, and the development of a mock dashboard for visualizing computer vision outputs. These results demonstrate the feasibility of both the notification workflow and the visualisation approach.

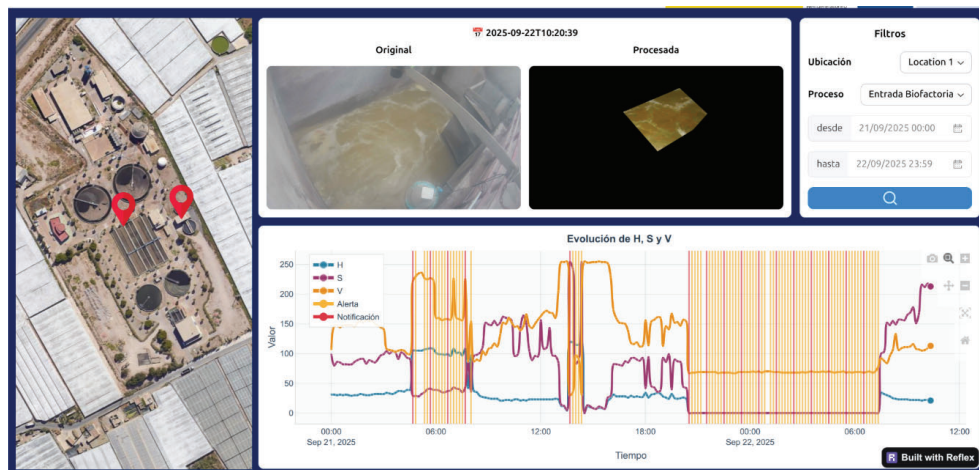


Figure 72 Mock-up of the DSS for reliable water treatment V-App

4.6.6 V-Apps under Development for Phase 2

Regarding DSS for water quality V-App, in Phase 2 the next development steps will focus on consolidating the cloud-based user interface and completing its integration with the AI-App outputs once developed. This includes finalising the mechanisms for early-warning alerts and periodic water-quality notifications, and preparing the application for containerized deployment and validation in the target testbed environment. In Phase 2, the Immersive Remote Driving V-App will focus on developing an autonomous mission system that enables the AGV to navigate independently through previously defined locations. This functionality will allow the vehicle to follow predefined routes without manual intervention, improving operational efficiency. In addition, a PTZ camera mounted on the AGV will be integrated, allowing remote control of its orientation and zoom while the vehicle is being remotely controlled.

In Phase 2, the Automatic Surveillance & Detection in Farming Operation V-App will focus on the integration of the already available SRS and LiDAR extraction functionalities into a complete end-to-end surveillance workflow. This will enable the V-App to provide consolidated radio and sensing data to the AI-App and to present detection results based on the outputs generated by the AI-App. In addition, the V-App will support the automatic triggering of notifications when a relevant event is detected, thus demonstrating the full detection chain from data acquisition to event reporting in farming operation scenarios.

In Phase 2, DSS for reliable water treatment V-App will focus on extending the current prototype alarm module into an operational notification service and refining the visualisation components already developed. This

includes completing the integration with the AI-App triggers, improving the dashboard for process status and computer vision outputs, and preparing the application for containerized deployment and validation under representative operating conditions.

5 Development of AI-Apps

The development of the AI-Apps focuses on creating the intelligence layer of the 6G-VERSUS ecosystem. Designed to transform network and application data into real-time actionable insights, the AI-Apps provide advanced analytics, prediction, and decision-support capabilities tightly integrated with the corresponding V-Apps and N-Apps. Within the scope of the 6G testbeds, the AI-Apps are not generic solutions; instead, they are tailored to address the specific operational requirements and challenges of each pilot domain, leveraging AI/ML techniques adapted to the needs of each cluster. Unlike the N-Apps, which follow a more homogeneous architecture and common implementation template, the AI-Apps significantly differ in terms of objectives, datasets, AI models, inference workflows, and deployment requirements. Therefore, a fully unified template is not applicable, as each AI-App is designed and optimized according to the specific needs of the corresponding use case and cluster environment. The development of the AI-Apps follows a structured and iterative roadmap aligned with the WP4 implementation plan and the staged execution of WP5 trials. The initial development phase focused on ensuring architectural readiness, interoperability, and functional validation within the triplet architecture. This included the definition of interfaces with the V-Apps and N-Apps, implementation of containerized deployments, training of initial AI models, and validation of end-to-end communication flows, resulting in the delivery of the initial prototype under MS09 and D4.1. The second phase focuses on refinement and operational maturity, including model retraining using trial outputs, API stabilization, integration with additional system components, and progressive integration with MLOps workflows (as reported in deliverable D3.1). To provide a comprehensive overview of the AI-App development lifecycle, this section presents the development methodology, workflow followed across all clusters alongside with quantitative results from the training phase. It further details the technical integration and readiness status of the AI-Apps implemented per cluster, highlighting their interoperability within the overall 6G-VERSUS architecture. All AI-App developments are maintained in the common WP4 GitLab repository, organized per cluster and aligned with the project software development and deployment guidelines.

5.1 Bulgarian Cluster

During **Phase 1 (Trials 1 and 2)**, the Bulgarian cluster will showcase the core triplet components required for the pilot use case (see Table 26), namely the **V-App Central (WAM)**, **V-App Central (WAC)**, **V-App Regional (RC)**, and the **AI-App**. The V-App components support real-time monitoring, central control, and regional actuation. The AI-App is introduced progressively: a dummy version is already part of the target deployment architecture, while the functional AI-App is being integrated in the UCY pre-piloting environment and is planned to support the second trial through predictive frequency-stability analytics. For **Phase 2**, the same AI-App will be further matured with broader datasets, improved models, and tighter operational coupling with the WAC and ML-Ops chain.

Table 26 Implemented and under development AI-Apps for the Bulgarian cluster

Cluster Name	App Name	Phase (1 or 2)
Bulgarian Cluster	AI-App – Grid Frequency Trajectory Prediction	1 and 2

For **Phase 2**, the Bulgarian cluster does not currently define a separate new AI-App, but rather a more mature version of the same AI-App, with extended functionality, broader scenario coverage, and tighter integration with the rest of the triplet components.

5.1.1 AI-App Grid Frequency Trajectory Prediction for the Bulgarian Pilot

5.1.1.1 Overview

The AI-App of the Bulgarian pilot is being developed to process streaming synchrophasor measurements and predict the short-term evolution of grid frequency under disturbance conditions. Its main purpose is to provide an early indication of the expected frequency trajectory, including the anticipated nadir, so that the control chain can react earlier and more effectively in low-inertia operating conditions. In the pilot architecture, the AI-App complements the V-App Central WAC by adding predictive intelligence on top of real-time PMU monitoring, thereby supporting preventive control actions aimed at avoiding power outages, preserving stability, and maintaining synchronization of the power system.

5.1.1.2 Architecture

The AI-App architecture follows the overall logic of the 6G-VERSUS triplet design, but is specialized for AI/ML operation by separating the training path from the online inference path. In the training path, synthetic datasets are generated from a MATLAB/Simulink dynamic grid model, processed through the ML-Ops pipeline, and used to train predictive models. In the inference path, the deployed AI-App consumes streaming PMU-derived measurements from the pilot environment and produces short-term frequency predictions and early-warning indicators for the central control layer. Functionally, the AI-App is positioned next to the central-side application environment, where it supports the V-App Central WAC with predictive outputs, while the V-App RC remains deployed separately at the regional side. This architecture is aligned with the project application model, while extending it with AI-specific lifecycle management, model training, experiment tracking, and deployment through ML-Ops.

5.1.1.3 Data Preprocessing

To support the AI-App, a synthetic dataset generation workflow was developed using a dynamic power-system model that represents a modified transmission grid with DERs, synchronous generators, inverter-based resources, dynamic loads, and detailed network components. The simulation model is implemented in MATLAB/Simulink and executed under a parametric simulation campaign designed to produce a broad range of operating conditions relevant to frequency stability.

The generated scenarios include:

- generation loss events,
- variations in renewable penetration,
- different inertia conditions,
- variations in synchronous generator control parameters,
- changes in load dynamics,
- different grid-fault and disturbance conditions.

Each simulation produces time-series outputs corresponding to PMU-like measurements, including voltage and current phasors, frequency, and RoCoF. These outputs are exported as structured datasets and processed through a preparation pipeline where they are filtered, aligned, converted to machine-learning-ready format, and labelled according to the simulated operating condition or disturbance scenario. Feature engineering is then applied to derive control-relevant indicators such as frequency statistics, RoCoF behaviour, voltage variations, and power imbalance proxies.

5.1.1.4 Training, Evaluation and Inference of AI Models

The labelled datasets are used for the training of predictive AI models targeting short-term frequency behaviour under disturbances. The current development effort includes both baseline machine-learning models and preliminary transformer-based approaches. The initial objective is to learn the relationship between the evolving PMU measurements and the expected future frequency response, with particular interest in the frequency nadir and the overall trajectory immediately following a disturbance. At the present stage, the

evaluation effort focuses on whether the trained models can reliably distinguish different disturbance conditions and provide useful short-term predictive information about the frequency response. Final benchmark figures are still under development, as the dataset is currently being expanded and the model family is still being refined. Nevertheless, the preliminary experiments have already shown promising capability in learning patterns associated with different disturbance scenarios and frequency behaviours. In inference mode, the trained model is intended to operate on streaming PMU-derived data from the pilot or pre-pilot environment and to produce early-warning outputs that can support the central WAC function. In this way, the AI-App moves beyond simple post-event analytics and becomes part of the operational decision-support chain.

5.1.1.5 Connection to MLOps

The AI-App is tightly connected to the project ML-Ops framework. The ML-Ops layer is used to orchestrate dataset preparation, model training, experiment tracking, and model lifecycle management. In the current implementation, **Airflow** is used to coordinate the processing workflow, while **MLflow** is used for experiment tracking, parameter management, and model versioning. This integration is important because it provides a structured path from dataset generation to repeatable model training and controlled deployment. It also creates the basis for future retraining, comparison of model versions, and more automated promotion of models from development to inference environments. As a result, the AI-App is not being developed as an isolated ML prototype, but as part of a maintainable software and ML lifecycle aligned with the broader Bulgarian-cluster architecture.

5.1.1.6 Development, Deployment and Results

The AI-App is being developed as a Python-based software component with a containerized deployment model. Its development combines three main implementation layers:

- (i) synthetic data generation in MATLAB/Simulink,
- (ii) training and lifecycle management through the project ML-Ops framework using Airflow and MLflow, and
- (iii) online inference through the deployed AI-App service.

The application is validated first in the UCY pre-piloting environment, where OPAL-RT emulates the power-system dynamics and the UCY 5G-OAI facility transports PMU telemetry to the application layer. This allows the AI-App to be developed and tested under realistic disturbance conditions before broader pilot-level deployment. Source code and technical documentation are maintained in the 6G-VERSUS GitLab repository for the UCY Bulgarian-cluster AI-App.

GitLab repository: <https://gitlab.6g-versus.eu/wp4/ai-apps/bulgarian-cluster/ucy>

The current implementation focus is on building the end-to-end path from simulated PMU-like data to a trained predictive model and then to deployable inference logic that can be connected to the WAC.

The AI-App is being developed with a clear distinction between:

- **offline model development**, where synthetic datasets are generated and models are trained and evaluated,
- **online inference**, where the deployed AI-App consumes live or streamed measurements and returns predictive outputs,
- **system integration**, where the AI-App is connected with the V-App Central WAC, the UCY pre-piloting setup, and later the broader pilot deployment chain.

This separation is important for engineering reasons, since it allows the AI model to evolve independently while preserving a stable application interface for the rest of the triplet components.

In terms of deployment, the AI-App is designed to be deployed as a **containerized application** within the Bulgarian-cluster infrastructure. In the A1 deployment architecture, the central location in Sofia hosts the central-side applications, including the AI-App placeholder/dummy deployment, while the regional side in Varna hosts the V-App RC. All applications in the A1 infrastructure run in Docker containers and are deployed through Helm charts in Kubernetes clusters. In parallel, UCY operates the pre-piloting integration environment

used to develop and validate the functional AI-App. In this setup, the AI-App is integrated with the OPAL-RT power-system testbed, the UCY 5G-OAI transport layer, and the central application environment where the V-App Central WAC and the ML-Ops services are hosted. This two-stage deployment approach allows the Bulgarian cluster to keep the target deployment architecture visible in the A1 infrastructure while maturing the functional AI-App in the UCY environment before full pilot-level end-to-end validation.

The current results demonstrate that the Bulgarian cluster has already established the core development chain for the AI-App. First, a realistic synthetic-data generation workflow has been implemented in MATLAB/Simulink, covering multiple disturbance scenarios and operating conditions relevant to low-inertia power systems. Second, these datasets are already connected to the ML-Ops pipeline for structured preparation, experiment tracking, and model training. Third, preliminary model-development activities have started, including baseline models and early transformer-based experiments. At integration level, the UCY pre-piloting environment already provides the required closed-loop validation context: OPAL-RT emulates the power-system dynamics, the 5G-OAI chain transports the measurement streams, and the V-App Central and V-App RC are already integrated with the simulated DER actuation path. Within this context, the AI-App is the remaining functional component under active integration, with the target role of providing predictive support to the central control application. Therefore, the main result at this stage is that the Bulgarian cluster has de-risked the AI-App path substantially: the data-generation workflow exists, the ML-Ops connection exists, the pre-pilot integration environment exists, and the deployment architecture is already defined. The next milestone is the completion of functional AI-App inference integration for Trial 2. The results presented in Figure 73 illustrate variations in frequency nadir and settling dynamics used to generate the training dataset for the AI-App.

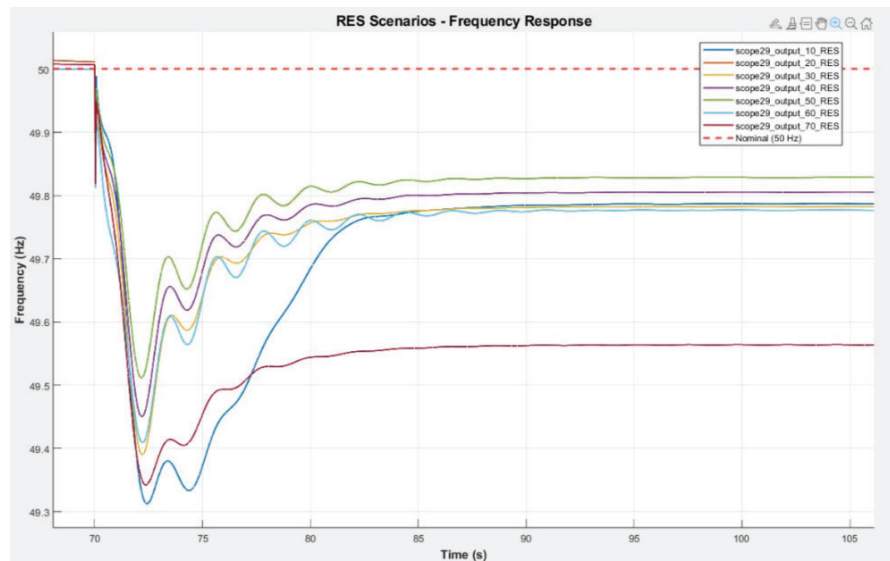


Figure 73 Frequency response of the simulated system under different generation-loss scenarios

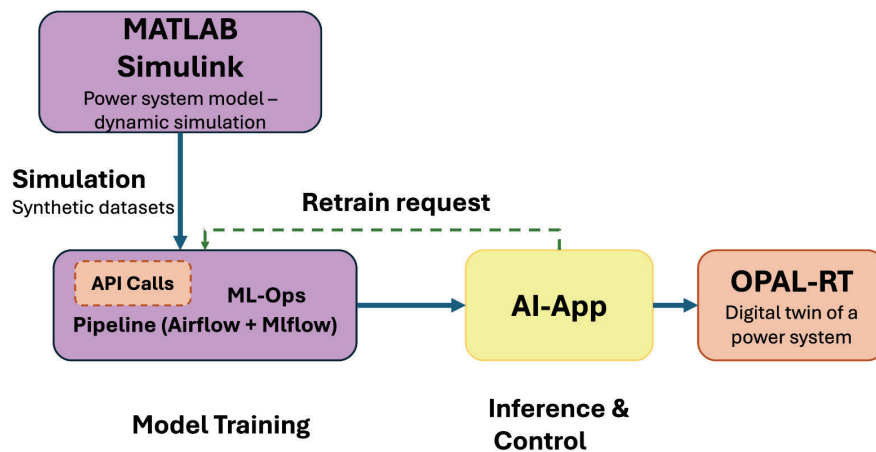


Figure 74 Example of the end-to-end AI-App development workflow, from synthetic dataset generation and preprocessing to model training, ML-Ops tracking, and deployment-oriented inference validation

5.1.2 AI-Apps under Development for Phase 2

For Phase 2, the Bulgarian cluster plans to continue the maturation of the same AI-App rather than introduce a different AI-App. The main direction is to improve the predictive quality, robustness, and operational usefulness of the AI-App by expanding the simulation campaign, enriching the synthetic training datasets, and refining the model architectures used for short-term frequency prediction. In particular, the ongoing development effort targets broader coverage of operating conditions, including additional disturbance categories, different combinations of renewable penetration and inertia, and more extensive DER behaviour. This will allow the AI-App to generalize better across realistic power-system conditions and to provide more reliable early-warning indicators to the WAC. In parallel, the Phase 2 effort will strengthen the operational integration of the AI-App with the rest of the triplet components. This includes tighter connection to the WAC decision-support loop, more mature deployment through the ML-Ops pipeline, and broader end-to-end validation in both the UCY pre-piloting environment and the Bulgarian pilot infrastructure. The objective is to move from a promising predictive prototype to a robust AI-enabled application that can contribute meaningfully to preventive control in low-inertia grids.

5.2 Finnish Cluster

The Finnish Cluster will follow a two-phase development and deployment approach aligned with the project trials (see Table 27). In Phase 1 (Trials 1 and 2), the focus is on validating the core intelligence of the system through the deployment of the two main AI applications: the Arctic Traffic Controller and the Arctic Energy Controller. These applications will be demonstrated together to show end-to-end energy-aware network optimization, including traffic forecasting, energy prediction, and adaptive energy management in Arctic conditions. In Phase 2 (Trials 2 and 3), the cluster will expand its functionality by introducing enhanced AI capabilities and additional supporting applications to improve system robustness, predictive accuracy, and operational resilience. These will build on the results and insights from Phase 1 and focus on optimization, scalability, and advanced decision-making support. Only the core applications will be showcased in Phase 1, while Phase 2 will focus on extensions and improvements.

Table 27 Implemented and under development AI-Apps in the Finnish cluster

Cluster Name	App Name	Phase (1or 2)
Finnish Cluster	ArcticTraffic Controller	Phase 1
Fnnish Cluster	ArcticEnergy Controller	Phase 1

5.2.1 Arctic Energy Controller AI-App

5.2.1.1 Overview

The Arctic Energy Controller AI-App is responsible for intelligent energy management in remote Arctic network sites. It dynamically selects the optimal operational mode of the site (full-power, power-saving, or emergency mode) based on predicted energy availability and network demand. By combining weather forecasts, on-site renewable energy production data, battery and hydrogen storage states, and predicted network traffic, the application ensures continuous and efficient operation of the communication infrastructure. It optimizes energy usage by balancing renewable energy utilization, storage discharge strategies, and backup hydrogen fuel cell activation.

5.2.1.2 Architecture

The Arctic Energy Controller follows a modular AI driven architecture. It consists of four main internal components: data ingestion module (from DB and APIs), prediction module (using the trained model), decision making module which is the logic of the app for energy optimization based on data and predictions and communication module (REST API interfaces). The application fetches the input data from DB (historical and FMI) plus the predicted traffic from ATC outputs. It processes these inputs to generate energy decisions, which are then will be sent to Remote IC, Network IC and napp. The architecture, seen in Figure 75 is fully aligned with the triplet system.

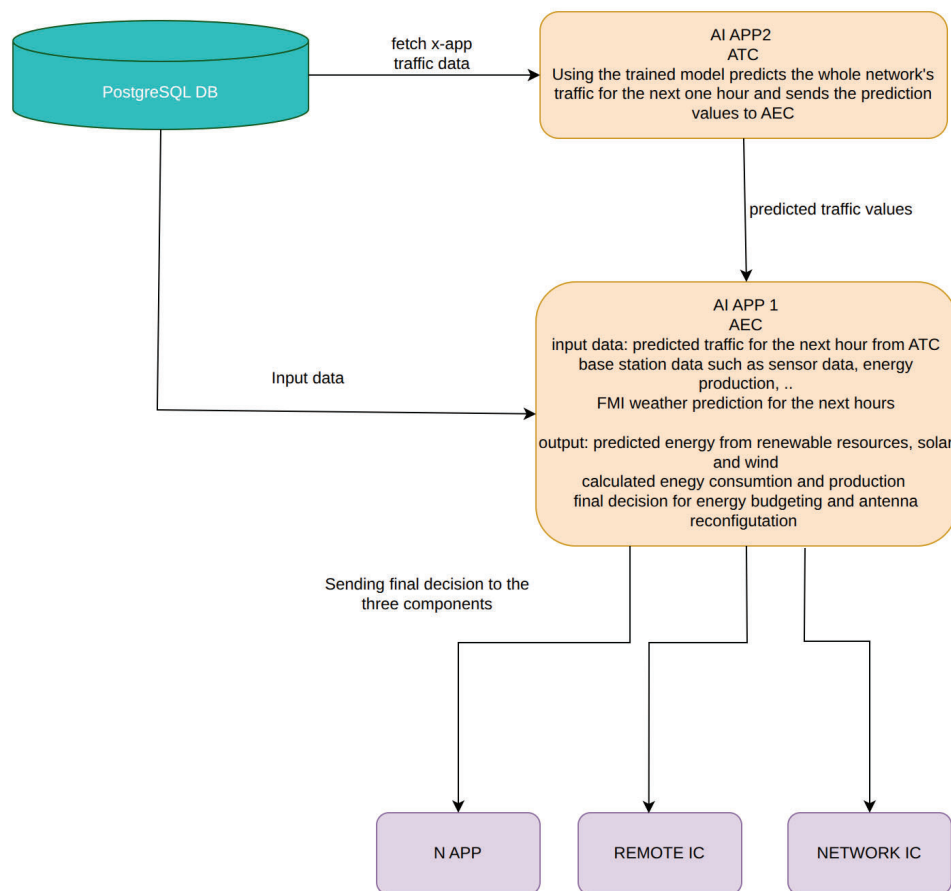


Figure 75 Architecture of the Arctic Energy Controller AI-App

5.2.1.3 Data Preprocessing

Input data is collected from FMI weather forecasts, real-time sensor data (solar/wind production), battery and hydrogen storage states, and network traffic predictions. Data is cleaned, time aligned and normalized before being fed into the model. Missing values are handled using interpolation techniques, and temporal features are engineered to improve prediction accuracy.

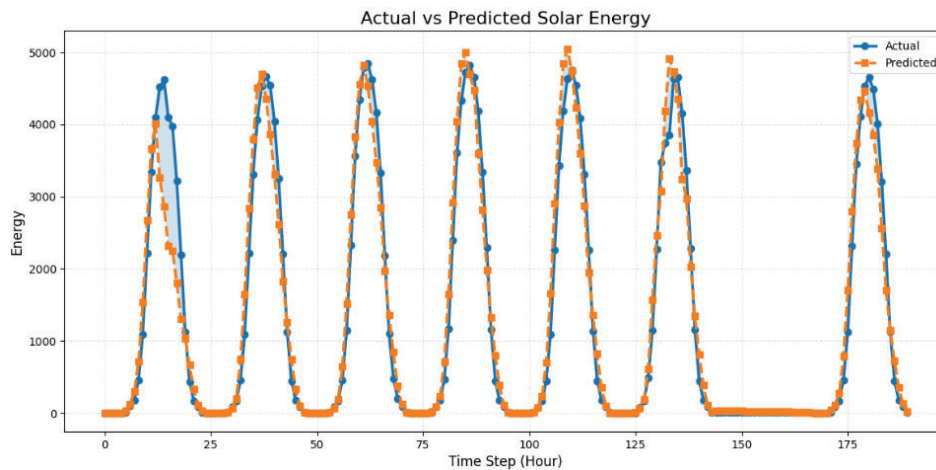
5.2.1.4 Training, Evaluation and Inference of AI Models

The model is trained using historical time-series data combining weather, energy production, and consumption patterns. A deep learning model is used to predict short-term energy availability and battery state evolution. Evaluation is performed using metrics such as MAE (Mean Absolute Error) and RMSE (Root Mean Square Error). Inference is performed in real time during operation, where the model continuously updates predictions based on incoming data streams.

To extend this framework, two separate models have been developed: one for predicting energy production from wind turbines and another for solar PV generation. Both models are based on the Temporal Fusion Transformer (TFT) architecture. For the solar PV model, performance is evaluated using multiple metrics, including MAE, daytime-specific absolute error (AE_{day}, computed only during daylight hours), peak error between predicted and actual values, RMSE, and validation loss. For the wind energy model, evaluation is conducted using MAE, RMSE, and validation loss.

Each model is trained under different configurations, including varying learning rates and hidden layer sizes, to identify the optimal parameter set. The best-performing model is then selected for inference. During deployment, the trained model is integrated into the application and used to generate predictions, which are subsequently compared against actual observed values, specifically, the next-hour energy production that was not available during prediction. This enables assessment of real-world predictive accuracy and validation of model generalization. Furthermore, the inference results are analyzed in relation to the training error metrics to ensure consistency between offline evaluation and real-time performance.

This Figure 76 illustrates the solar energy predictions and demonstrates how the model captures the patterns of sunrise and sunset, including the corresponding rise and decline in energy production. It also includes a comparison between the predicted values and the actual next-hour energy output, reinforcing the previously discussed evaluation results.

**Figure 76 Actual vs Predicted Solar Energy**

The following images, represents the same comparison for the energy being produced by wind turbines.

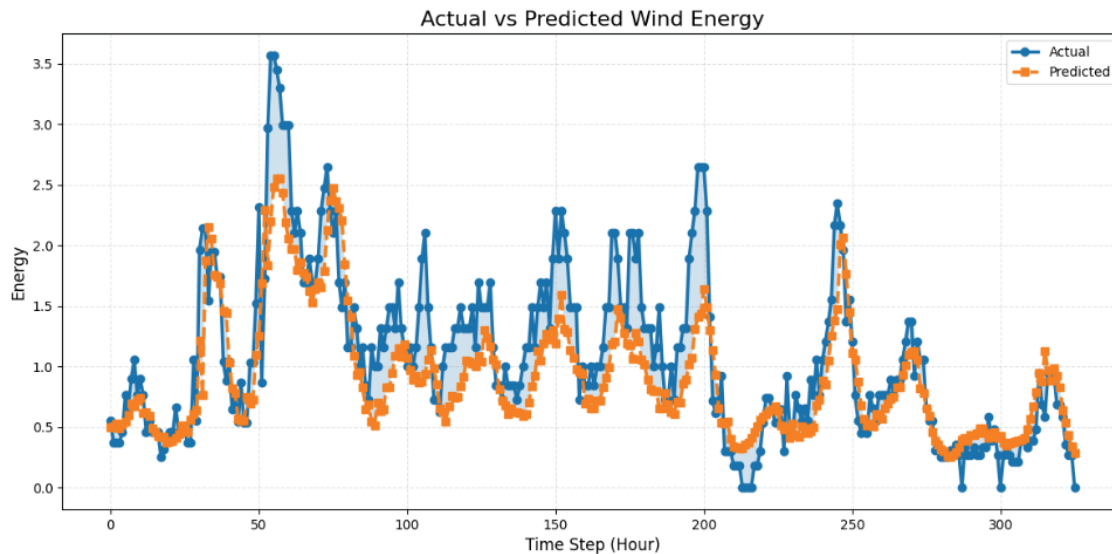


Figure 77 Actual vs Predicted Wind Energy

5.2.1.5 Connection to MLOps

The Arctic Energy Controller is integrated with the project's MLOps framework defined in WP3 to support the full lifecycle of the AI model, including data versioning, model training and evaluation. Training is performed within the MLOps platform using curated historical datasets, where experiments are tracked and validated using standardized metrics such as MAE and RMSE. Model versioning is maintained to ensure traceability between different training iterations. Once a model reaches the required performance level, it is exported from the MLOps environment and deployed within the application runtime. During operation, the system supports periodic retraining using newly collected data uploaded into the framework to improve prediction accuracy and adapt to seasonal variations in Arctic conditions. Monitoring tools are used to track model performance degradation and trigger retraining when necessary.

In terms of deployment the Arctic Energy Controller AI application is deployed as a containerized microservice within the testbed infrastructure using Docker. This container encapsulates the trained AI model, along with its preprocessing components and RESTful API services built with FastAPI, ensuring a self-contained and portable deployment unit. The system follows a modular architecture that enables seamless integration with multiple components across the infrastructure. It connects to a central database to retrieve input data, interfaces with the Arctic Traffic Controller to obtain predicted traffic information and communicates with the Remote Intelligent Controller (IC), NApp, and Network IC to execute energy optimization decisions. To support scalability, portability, and efficient integration, the containerized service is orchestrated within the testbed environment. Additionally, continuous integration and continuous deployment (CI/CD) pipelines managed through GitLab automate the processes of building, testing, and deploying the application, ensuring consistent and reliable updates. The Arctic Energy Controller AI-App is being deployed and tested within the Finnish testbed environment. The system demonstrates stable and reliable operation under varying simulated Arctic conditions.

Key outcomes include:

- Accurate short-term prediction of renewable energy availability
- Effective decision-making for energy mode selection (full-power, energy-saving, emergency)
- Improved utilization of renewable energy sources and reduced reliance on backup systems
- Seamless integration with the Arctic Traffic Controller and triplet architecture components
- Logs from the system indicate consistent API communication between modules and successful execution of energy management decisions at the remote site. Initial evaluation must soon show that the system can maintain continuous network operation while optimizing energy consumption.

- Further validation and performance tuning will be conducted in Phase 2 to enhance robustness and predictive accuracy.

5.2.1.6 Development, Deployment and Results

The application is developed in Python using modern AI and backend libraries. Core dependencies include PyTorch for deep learning modeling, FastAPI for REST API communication, and Docker for containerization. The codebase is maintained in a GitLab repository. The system follows an MLOps framework, enabling reproducible training, model versioning, and deployment pipelines. However, the trained model will be downloaded from the framework and will be used in the software in another environment.

The Arctic Energy Controller AI-App has been successfully developed and deployed as a containerized microservice within the testbed environment. The system integrates multiple components, including data ingestion, prediction models, decision making logic, and REST API communication, enabling end-to-end operation. At the current stage, validation has been conducted using synthetic input data to verify system functionality and integration.

These tests confirm that the application correctly processes input data, executes the prediction pipeline, and generates appropriate energy management decisions. From a modeling perspective, the AI components, solar PV and wind prediction models based on the Temporal Fusion Transformer (TFT) architecture, have been trained and evaluated offline using historical datasets. The models demonstrate promising predictive performance according to standard evaluation metrics such as MAE and RMSE. As illustrated in the final validation figures, the models are able to accurately capture energy production patterns and provide reliable predictions during inference when evaluated on historical (non-real-time) data.

However, integration of these models into the live system has so far been validated only under controlled conditions using simulated inputs, rather than real-time operational data. It is important to note that full system validation under real-world Arctic conditions has not yet been conducted. Future work will focus on testing the application with real data streams, performing comprehensive end-to-end evaluation, and quantifying improvements in energy efficiency, system reliability, and decision-making performance.

The actual real-world results cannot yet be provided, as the AI APP1 has not yet been tested in real-world scenarios.

5.2.2 Arctic Traffic Controller AI-App

5.2.2.1 Overview

The Arctic Traffic Controller AI-App is responsible for forecasting network traffic demand. It uses time-series regression models to predict traffic load and later on in AEC, these predictions will be translated into energy consumption estimates. The output is provided to the Arctic Energy Controller to support coordinated energy optimization decisions.

5.2.2.2 Architecture

The Arctic Traffic Controller is built as a modular forecasting service consisting of data ingestion, preprocessing, time-series prediction model, and API communication layer. It retrieves historical network data from the database and processes it using a regression-based forecasting model. The predicted traffic values are transmitted to the Arctic Energy Controller through REST APIs. The design follows the same triplet architecture principles ensuring compatibility with the rest of the system.

5.2.2.3 Data Preprocessing

[Network traffic data is aggregated and cleaned from raw x-app inputs stored in the database through n-app. Temporal smoothing and feature extraction (hourly/daily patterns) are applied. Data normalization is performed before training the forecasting model.

5.2.2.4 Training, Evaluation and Inference of AI Models

The model is trained using historical traffic datasets using supervised regression techniques. Performance is evaluated using MAE and RMSE. During inference, the model produces hourly traffic predictions which will be converted into energy consumption estimates for downstream use.

The forecasting system is based on two separate supervised regression models implemented using LightGBM. One model is trained to predict downlink traffic, while the second model is trained to predict uplink traffic for the next-hour horizon. Both models are trained using historical network and radio access data, including features derived from:

Radio and channel quality indicators (e.g., SINR, RSRP, RSRQ) Throughput and bitrate measurements
Temporal features such as hour of day, day of week, weekday/weekend indicators, and holiday flags
Aggregated traffic statistics from the network layer. These features enable the models to capture both radio condition-driven variations and temporal usage patterns, such as differences in traffic demand during working hours, weekends, and holidays. The LightGBM models were selected due to their efficiency, robustness to heterogeneous feature types, and strong performance on tabular time-series regression problems.

5.2.2.5 Connection to MLOps

The model is being trained using the project's MLOps framework defined in WP3 and being evaluated and retrained in the same platform. When training is done, model will be downloaded and used in software environment.

5.2.2.6 Development, Deployment and Results

After training, the finalized models are exported and deployed into the ATC for inference. The service is packaged in Docker containers and integrated into a dedicated server. During inference, the Arctic Traffic Controller retrieves last-hour historical data from the database, adds necessary features to them for precise prediction and processes them through the trained LightGBM models, and generates hourly predictions for both uplink and downlink traffic. These predictions will be sent to the AEC to make the final decision of the system via REATAPI. Preliminary internal tests confirm that the forecasting pipeline is stable and produces consistent predictions under simulated scenarios. Nevertheless, real-world validation is still pending until full system integration with live network infrastructure is completed. Therefore, the actual real-world results cannot yet be provided, as the AI APP1 has not yet been tested in real-world scenarios.

5.2.3 AI-Apps under Development for Phase 2

In Phase 2, the Finnish Cluster will extend the current system with capabilities to improve robustness and decision intelligence. These developments will focus on enhancing predictive accuracy, introducing anomaly detection mechanisms, and improving system adaptability under extreme Arctic conditions.

5.2.3.1 Arctic Energy Control

The AI-App *Arctic Energy Controller* is responsible for the primary decision-making process of the system, determining how energy resources are managed at remote sites operating in harsh Arctic conditions. Its objective is to ensure continuous network operation while optimizing the use of available energy sources. The application dynamically selects between different operational modes such as full power, power-saving mode, or emergency mode(rarely), depending on the predicted availability of energy and network demand.vThe AI-App operates as part of the triplet architecture and consumes input data from a centralized database. This data includes weather forecasts provided by FMI (Finnish Meteorological Institute), real-time measurements of solar and wind energy production at the remote site, and the current capacity levels of battery storage and hydrogen fuel cell. In addition, it receives predicted energy consumption values of the network from the Arctic Traffic Controller AI-App. Using a neural network-based model, the application estimates the amount of renewable energy that will be generated in the next hour and predicts the future state of the battery. Based on these predictions, it computes the expected energy balance by comparing the forecasted energy production with the anticipated consumption, which includes both traffic-dependent energy usage and the static energy

requirements of the site. Based on this analysis, the AI-App determines the optimal energy strategy. It can decide to use stored renewable energy, allow deeper battery discharge beyond standard state-of-charge limits when future generation is expected, or switch to hydrogen fuel cells when renewable sources are insufficient. The resulting decision is transmitted to the remote intelligent controller at the site for execution and to the network to enable appropriate reconfiguration of the radio infrastructure. The decision is also forwarded to the N-App, where it is stored in the database and made available to the V-App for visualization and monitoring purposes.

5.2.3.2 Arctic Traffic Control

The AI-App *Arctic Traffic Controller* is responsible for forecasting network traffic and estimating the associated energy consumption required to maintain communication services. The application relies on regression-based time series models to predict the traffic load of the network for the upcoming hour. Based on this prediction, it derives the corresponding energy consumption that will be required to handle the expected traffic demand. The AI-App operates within the triplet architecture and consumes network-related data stored in the database. This data is initially collected from the network through the X-App, transmitted to the N-App, and subsequently stored for processing. After performing traffic prediction and energy estimation, the Arctic Traffic Controller transmits the calculated energy consumption values to the Arctic Energy Controller via a REST API.

5.2.3.3 System Integration and Data Flow

The Finnish solution is based on a coordinated interaction between the AI-Apps and the components of the triplet architecture. Network data is collected through the X-App and forwarded to the N-App, which stores it in the database. Similarly, data from the remote site is gathered by the remote intelligent controller and transmitted to the N-App before being stored. The Arctic Traffic Controller retrieves network data from the database to predict traffic and estimate energy consumption, while the Arctic Energy Controller gathers environmental and system data and combines it with the predicted consumption values to determine the optimal energy strategy. The resulting decisions are transmitted to the network for configuration adjustments and to the remote intelligent controller for energy management actions. At the same time, all relevant data and decisions are stored by the N-App and made accessible to the V-App, which provides visualization and monitoring capabilities. Through this coordinated process, the two AI-Apps jointly enable intelligent energy budgeting and efficient utilization of renewable energy sources, ensuring reliable network operation in extreme Arctic environments. All communication between the apps happens via the REST APIs.

5.3 Greek Cluster

The Greek Cluster follows a phased development and deployment approach aligned with the WP5 trial execution (see Table 28). In Phase 1 (Trials 1 and 2), the focus is on validating the core intelligence and interoperability of the deployed AI-Apps within the triplet architecture, through the integration of the AI-App for Detection & Tracking and the AI-App for Wildfire Monitoring. These applications support situational awareness and real-time event monitoring capabilities in the Greek cluster use cases. In Phase 2 (Trials 3 and 4), the cluster will further extend its AI capabilities through the introduction of additional functionalities, including the AI-App for Virtual Fence and Prohibited Area Detection, while also refining and optimizing the existing AI-Apps based on trial outputs and real-world observations. The Phase 2 activities focus on improving robustness, scalability, integration maturity, and operational reliability across the deployed applications.

Table 28 Implemented and under development AI-Apps in the Greek cluster

Cluster Name	App Name	Phase (1or 2)
Greek	AI-App Detection & Tracking	Phase 1 & 2
Greek	AI-App for Wildfire Monitoring	Phase 1 & 2

Cluster Name	App Name	Phase (1or 2)
Greek	AI-App for Detection & Tracking	Phase 1 and 2
Greek	AI-App Virtual Fence and Prohibited Area Detection	Phase 2

5.3.1 AI-App Detection & Tracking

5.3.1.1 Overview

The AI-App for Detection and Tracking is responsible for providing AI-based perception capabilities within the Greek cluster, supporting the real-time analysis of video streams captured by surveillance assets such as UAVs and ground robotic platforms. Its main role is to detect and track objects of interest in live video feeds, generating structured outputs that can be consumed by the INUS Asset V-App to enhance situational awareness and support operational decision-making. In this way, the AI-App contributes to critical use cases such as search and rescue operations, perimeter surveillance, and wildfire-related field monitoring, where timely and reliable visual intelligence is required.

5.3.1.2 Architecture

The AI-App for Detection and Tracking follows a modular tracking-by-detection architecture designed to support real-time video analytics in the Greek cluster. The application comprises a video ingestion layer, a preprocessing stage, an object detection module based on YOLO11, a multi-object tracking module based on BoostTrack, and an output interface responsible for publishing structured metadata towards the INUS Asset V-App. Incoming video streams originating from field assets, such as UAVs and the robotic dog, are first acquired and prepared for analysis. They are then processed by the YOLO11-based detection module, which identifies objects of interest in each frame, while the tracking module associates detections across consecutive frames in order to preserve object identities over time and support continuous tracking of moving targets. The output of the AI-App consists of AI-generated metadata, including object classes, bounding box coordinates, confidence scores, and tracking identifiers, which can be forwarded to the V-App for visualization and operational use. The architecture seen in Figure 78 is designed to support deployment flexibility across edge and cloud resources, depending on the available infrastructure and the latency requirements of the operational scenario. Overall, the architecture enables the integration of real-time AI perception capabilities into the broader INUS system while supporting scalable and interoperable deployment.

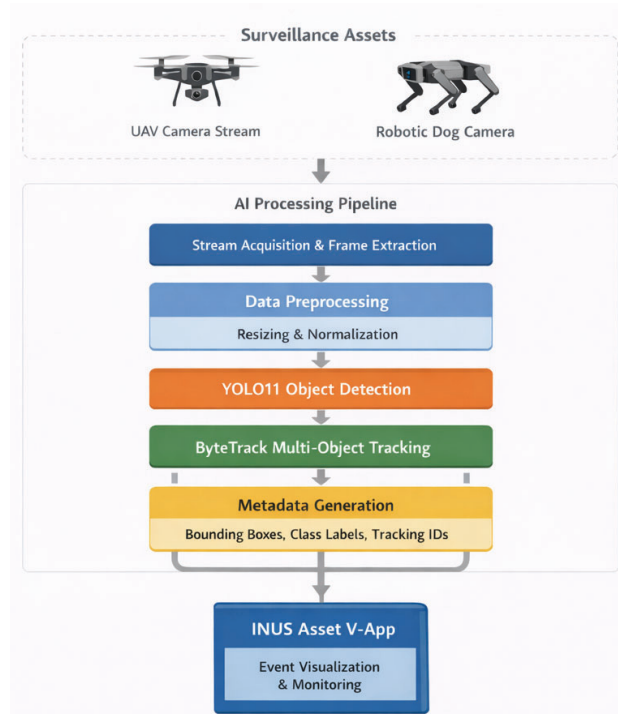


Figure 78 AI-App Detection and Tracking architecture

5.3.1.3 Data Preprocessing

The data preprocessing stage of the AI-App for Detection and Tracking is responsible for preparing the incoming video streams before they are processed by the object detection and tracking pipeline. In particular, the preprocessing workflow includes frame extraction from live video feeds, resizing and normalization of image data according to the input specifications of the YOLO11 detection model, and the preparation of frame sequences for subsequent tracking through ByteTrack. The preprocessing stage also handles heterogeneous stream characteristics, such as variations in image resolution, frame rate, and viewpoint, which are expected due to the use of both UAV-mounted and ground-based camera systems within the INUS framework. By transforming heterogeneous visual inputs into a consistent format, the preprocessing stage supports stable and efficient execution of the AI pipeline under real-time operational conditions. It therefore contributes to the robustness of the AI-App and to its ability to operate across different surveillance assets and deployment conditions.

5.3.1.4 Training, Evaluation and Inference of AI Models

The AI-App for Detection and Tracking relies on a tracking-by-detection computer vision pipeline based on YOLO11 for object detection and ByteTrack for multi-object tracking in streaming video data. Inference is primarily executed at the edge or fat edge, in line with the project architecture, so as to satisfy real-time operational requirements and minimise the latency associated with remote processing. This deployment model supports near real-time perception on live video feeds acquired from UAVs and ground robotic platforms in outdoor environments. At the current stage, the adopted pipeline is supported by evidence reported in the relevant technical literature, where YOLO11-based models have shown a favourable balance between inference speed and detection performance for UAV-related object detection tasks, while ByteTrack has demonstrated efficient detection association across successive frames for real-time tracking. In this respect, the AI-App is designed to support a practically deployable and resilient object detection and tracking capability for live field operations, rather than focusing exclusively on generic benchmark optimisation. The current evaluation and validation activities are centred on representative live-stream conditions and real-time demonstrations of object detection and tracking. In future work, the pipeline will be further refined through custom training and transfer learning processes in order to better address aerial viewpoints, small object sizes,

scene complexity, and project-specific target classes. Continued evaluation will also focus on improving robustness, tracking stability, and runtime efficiency under diverse environmental and operational conditions.

5.3.1.5 Connection to MLOps

The current production deployment focuses on stable real-time inference and event dissemination, while enhanced MLOps integration is planned for Phase 2. The next development phase will focus on model versioning, scheduled evaluation workflows, and controlled model promotion mechanisms.

5.3.1.6 Development, Deployment and Results

The AI-App for Detection and Tracking has been developed as a containerised AI-based video analytics service following a tracking-by-detection pipeline tailored to the operational requirements of the INUS system, namely real-time perception from UAV and robotic platforms in complex outdoor environments. The implementation is based on **YOLO11** for object detection and **ByteTrack** for multi-object tracking, enabling the identification of objects of interest in individual frames and the maintenance of persistent target identities across successive frames. This design supports the generation of structured outputs, including bounding boxes, class labels, confidence scores, and tracking identifiers, which can be exploited by higher-level INUS services for event generation, target localisation, and operational monitoring. From an implementation perspective, the AI-App processes live video streams originating from surveillance assets such as UAVs and the robotic dog, and applies a modular perception pipeline comprising video ingestion, preprocessing, frame-by-frame inference, tracking, and metadata generation. The implementation has been designed to support execution on edge or fat-edge compute resources, in line with the overall system architecture and the real-time requirements of the Greek cluster. In this way, the AI-App supports low-latency operation while reducing dependence on remote cloud processing for time-critical perception tasks. The development effort has focused on ensuring interoperability with the INUS Asset V-App, so that the generated detection and tracking outputs can be consumed in a structured way for operator-facing visualisation and situational awareness. Overall, the AI-App development follows a modular and service-oriented approach, combining real-time AI inference, tracking continuity, and integration with the broader INUS application workflow under realistic trial conditions.

The AI-App for Detection and Tracking is deployed as a containerised application following a cloud-native approach, in line with the broader deployment principles adopted in the Greek cluster. Depending on the operational scenario and the latency constraints of the perception task, the AI-App can be deployed across edge, fat-edge, or cloud resources. In the current implementation, inference is executed primarily at the edge or fat edge, so as to support near real-time video analytics on live streams originating from UAVs and ground robotic platforms. This deployment model enables efficient integration with the surveillance assets and with the INUS Asset V-App, while also supporting portability, maintainability, and simplified lifecycle management within the testbed infrastructure. In addition, the containerised approach facilitates future scaling and service evolution as the operational requirements of the AI-App expand in terms of connected assets, supported scenarios, and analysed video streams.

The development and deployment of the AI-App for Detection and Tracking resulted (Figure 79 and Figure 80) in an operational AI-based perception component capable of supporting real-time object detection and tracking on live video streams within the Greek cluster. During the initial trial phase, the AI-App successfully demonstrated real-time video analysis on feeds originating from surveillance assets, producing structured outputs that were consumed by the INUS Asset V-App for operator-facing visualisation and situational awareness support. The obtained results confirmed the feasibility of integrating edge-based AI video analytics into the broader INUS operational workflow under realistic trial conditions. The current implementation provides object-level outputs including bounding boxes, class labels, confidence scores, and tracking identifiers, thereby enabling the continuous observation of detected targets across successive frames. These initial results establish a solid baseline for the further evolution of the AI-App in Phase 2, where the focus will be placed on improving robustness, extending support to additional object classes and sensing platforms, and strengthening the integration with the rest of the V-/N-/AI-App ecosystem.

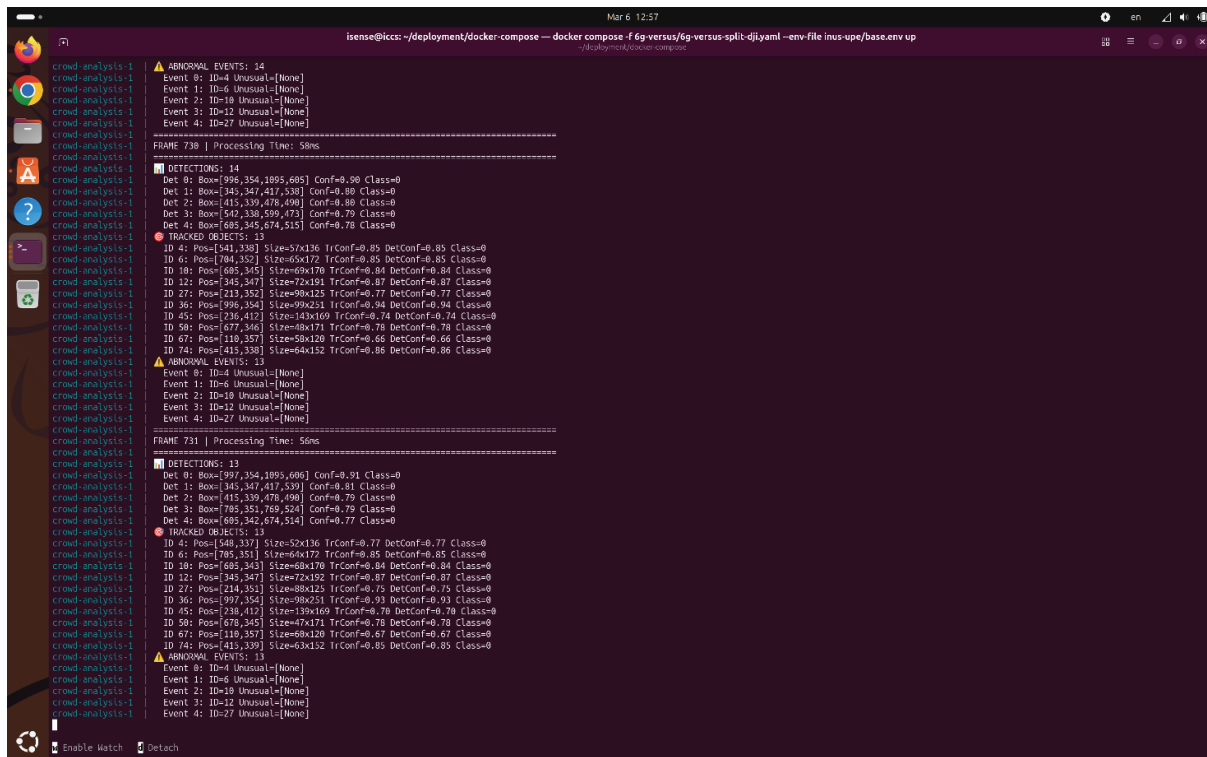


Figure 79 AI-Apps detection logs

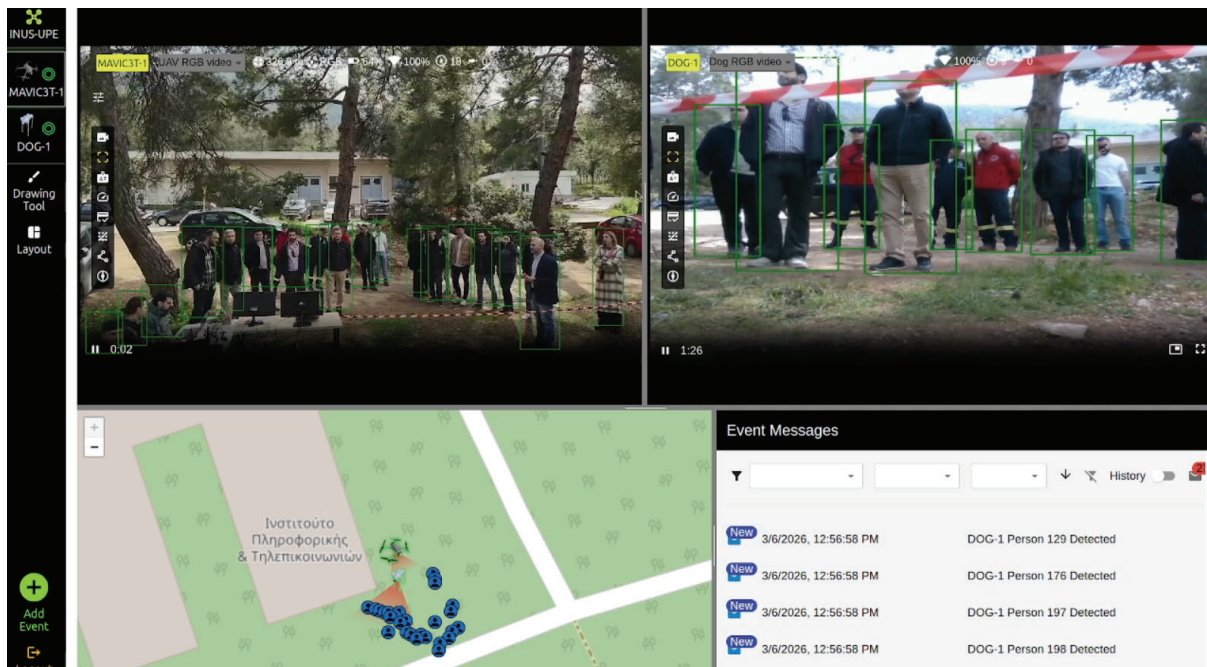


Figure 80 Real-time human detection and tracking from UAV and Robotic dog video streams

5.3.2 AI-App for Early Wildfire Monitoring

5.3.2.1 Overview

The AI-App for Early Wildfire Monitoring is an AI-driven edge application designed for real-time smoke detection from continuous RTSP video streams originating from field-deployed cameras. The application

continuously ingests live video, performs GPU-accelerated inference using a YOLO-based object detection model, and applies temporal persistence logic to distinguish true smoke events from transient artifacts. Upon validation of an event, the AI-App generates structured MQTT metadata, stores image and video evidence on NAS storage, and transmits mobile alerts through Pushover.

5.3.2.2 Architecture

The AI-App architecture follows a containerized GPU inference design, optimized for real-time smoke detection from multiple RTSP streams. A dedicated Docker container running on an NVIDIA-enabled GPU server receives live video from field-installed PTZ cameras over the 10 Gbps dark fiber connection between NCSRD and OTE infrastructure. The internal architecture consists of stream ingestion, frame preprocessing, YOLO-based smoke inference, temporal event persistence logic, evidence generation, MQTT dissemination, and mobile alert notification through Pushover. The architecture is designed to support real-time AI processing, distributed deployment, interoperability with vertical services, and future extensibility toward MLOps orchestration.

5.3.2.3 Data Preprocessing

The smoke detection model was trained on a heterogeneous wildfire smoke image dataset of approximately 8,000–10,000 original images, collected from multiple complementary sources:

- publicly available wildfire and smoke datasets
- internet-scraped smoke and wildfire imagery
- historical images from project-owned deployed cameras
- synthetically generated forest smoke scenes
- smoke scenarios generated from a custom Unreal Engine forest simulation environment

The dataset includes a single target class: smoke. All images were manually annotated by the development team using LabelImg (<https://github.com/HumanSignal/labelImg>) in YOLO TXT format, generating one annotation file per image. To ensure robust model development and unbiased evaluation, the original dataset was split into:

- 80% training
- 10% validation
- 10% test

The training subset was used for model learning, the validation subset for hyperparameter tuning and convergence monitoring, and the test subset for final performance evaluation and threshold selection. To improve generalization across diverse weather, forest, and camera conditions, multiple augmentation strategies were applied, including:

- random crops
- brightness and contrast changes
- horizontal flipping
- mosaic augmentation
- scaling and translation transforms
- random erasing
- HSV color perturbations

Following augmentation, the effective training corpus expanded to approximately 20,000 images, significantly improving dataset diversity and enabling the model to better generalize across real-world smoke patterns, lighting conditions, and forest environments. The final training process used 1920×1920 image resolution, YOLO-native augmentation pipelines, and distributed multi-GPU execution across 7 GPUs.

5.3.2.4 Training, Evaluation and Inference of AI Models

The smoke detection model was developed using YOLOv8-Large and trained through transfer learning using pretrained weights on a custom wildfire smoke dataset. Training was executed for 80 epochs using 1920×1920 image resolution, batch size 28, and distributed multi-GPU execution across 7 NVIDIA GPUs, enabling efficient convergence on the large-scale smoke dataset. The training process demonstrated stable convergence, with both training and validation losses decreasing consistently over the epochs. The final model achieved strong performance for the smoke class, with:

- Precision ≈ 0.78
- Recall ≈ 0.74
- mAP@0.5 ≈ 0.784
- mAP@0.5:0.95 ≈ 0.40
- Best F1-score ≈ 0.75
- Optimal confidence threshold ≈ 0.404

The precision-recall, F1-confidence, and loss curves were used to validate convergence stability and select the operational confidence threshold used in production deployment. During inference, the AI-App follows a multi-stage real-time smoke validation workflow:

1. Continuous RTSP ingestion from the field camera stream
2. YOLO inference on every incoming frame
3. If smoke is detected above the selected confidence threshold, the system initiates a strict consecutive-frame temporal persistence counter
4. The detection must remain valid for X consecutive seconds without missing even a single frame
5. Optional false-positive diagnostics and similarity heuristics may be executed in parallel for observability and future retraining support
6. Once the temporal persistence threshold is satisfied:
 - a. the event image snapshot is generated
 - b. the event video clip is created
 - c. both artifacts are stored on NAS
 - d. structured MQTT metadata is published
 - e. mobile alerts are sent through Pushover

This inference strategy was intentionally designed to reduce transient false positives caused by environmental motion, cloud artifacts, lighting fluctuations, dust, or temporary smoke-like patterns, while ensuring reliable early wildfire detection. The deployed inference service runs as a GPU-accelerated Docker container on NVIDIA A10 hardware, maintaining real-time performance while processing live RTSP streams.

5.3.2.5 Connection to MLOps

The current production deployment focuses on stable real-time inference and event dissemination, while enhanced MLOps integration is planned for Phase 2. The next development phase will focus on model versioning, scheduled evaluation workflows, and controlled model promotion mechanisms.

5.3.2.6 Development, Deployment and Results

The AI-App was developed in Python 3.10 as a GPU-accelerated containerized computer vision service, optimized for real-time wildfire smoke detection from live RTSP streams. The implementation is based on NVIDIA CUDA 11.8, PyTorch 2.5.1, and Ultralytics YOLOv8-Large, enabling efficient inference on NVIDIA A10 GPUs.

The main technology stack includes:

- PyTorch – <https://pytorch.org/>
- Ultralytics YOLOv8 – <https://docs.ultralytics.com/>
- OpenCV – <https://opencv.org/>

- NumPy – <https://numpy.org/>
- Paho MQTT – <https://pypi.org/project/paho-mqtt/>
- Requests – <https://requests.readthedocs.io/>
- PyOWM – <https://pyowm.readthedocs.io/>
- Docker – <https://www.docker.com/>
- NVIDIA CUDA – <https://developer.nvidia.com/cuda-toolkit>

Due to IPR protection and planned exploitation activities, the source code is not uploaded to the shared GitLab repository. From an implementation perspective, the AI-App was developed as a modular detection pipeline composed of:

- RTSP stream ingestion
- GPU inference execution
- temporal event persistence logic
- false-positive diagnostics
- NAS evidence generation
- MQTT dissemination
- Pushover mobile alerting
- runtime diagnostics and observability overlays

The live inference workflow operates as follows:

1. The application ingests the RTSP stream continuously from the field camera
2. YOLO inference is executed on every incoming frame
3. If smoke is detected, the system starts a strict consecutive-frame persistence counter
4. The alert is triggered only if smoke remains continuously detected for X seconds without missing even one frame
5. Optional false-positive diagnostics and similarity heuristics can run in parallel without blocking alert generation
6. Once the persistence threshold is reached:
 - a. the event snapshot image is saved
 - b. the event video clip is generated
 - c. both are stored on NAS
 - d. MQTT metadata is published
 - e. Pushover mobile alerts are transmitted

This strict persistence-based design was intentionally implemented to reduce transient false positives caused by clouds, lighting reflections, moving objects, or dust artifacts. The model was trained using YOLOv8-Large pretrained weights, with 80 training epochs, 1920 resolution, batch size 28, and 7-GPU distributed training, as documented in the training configuration. The selected inference threshold was derived from the training evaluation curves, where the deployed operating point achieved:

- Precision ≈ 0.78
- Recall ≈ 0.74
- mAP@0.5 ≈ 0.784
- Best F1-score ≈ 0.75
- Optimal confidence threshold ≈ 0.404

For Phase 1, the AI-App focuses on stable real-time inference. MLOps-based retraining, experiment tracking, and model lifecycle automation are planned for Phase 2.

In terms of deployment the AI-App was deployed as a containerized GPU-enabled Docker service on an NVIDIA server located within OTE infrastructure. The deployment uses the image based on CUDA 11.8, with NVIDIA GPU passthrough and direct access to mounted NAS storage for evidence persistence. The deployment follows a single-container-per-camera model, where each container instance is launched with:

- dedicated RTSP source
- camera ID

- site metadata
- mounted NAS evidence directory
- optional runtime configuration override
- assigned GPU device visibility

This architecture enables deterministic scaling across multiple cameras and GPUs while preserving low-latency inference performance. The AI-App was successfully implemented, containerized, and validated as a real-time GPU inference service for wildfire smoke detection. The provided screenshots and logs confirm:

- successful CUDA initialization
- healthy Docker runtime state
- GPU device assignment
- RTSP stream acquisition
- continuous frame processing
- stable inference execution
- NAS mount validation
- successful MQTT broker connectivity
- successful alerting pipeline execution

The accompanying figures include in

Figure 81 the training convergence plots, in Figure 82 F1-confidence threshold curves, in Figure 83 Docker deployment validation screenshots, in Figure 84 CUDA runtime logs, and in Figure 85 GPU utilization evidence, collectively demonstrating the successful end-to-end implementation of the AI-App from model development to production-ready containerized inference.

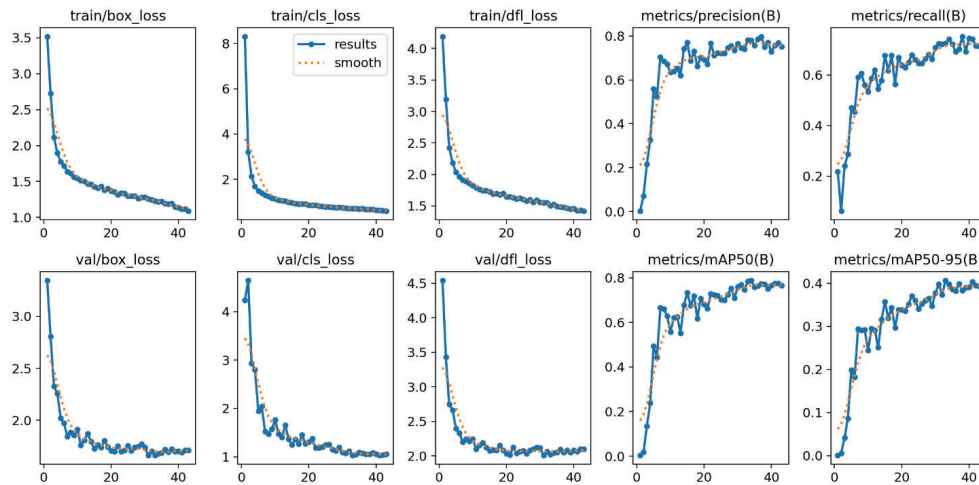


Figure 81 YOLOv8 smoke detection model training convergence curves, showing training/validation losses and precision, recall, and mAP evolution over 80 epochs

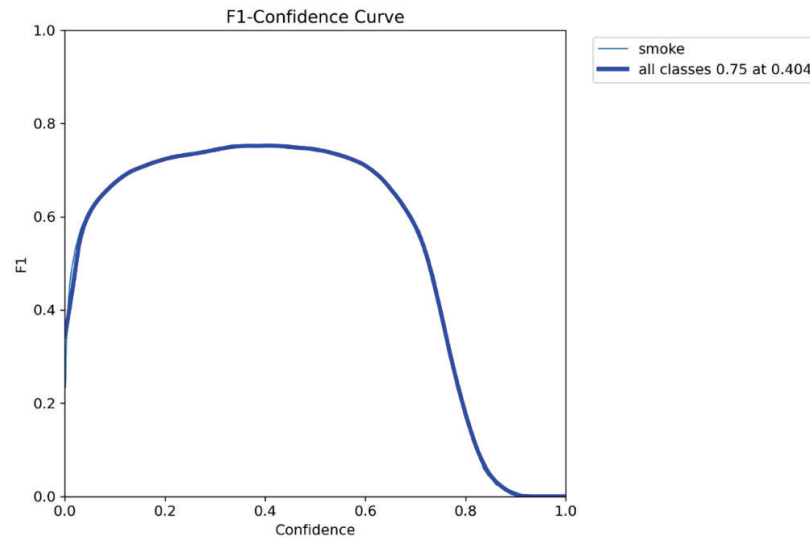


Figure 82 F1-confidence curve used for operational threshold selection of the smoke detection model during deployment

```
csmsote@gpuserver:~$ sudo docker ps -a --filter "name=wildfire-aiapp-Demokritos-1"
CONTAINER ID   IMAGE                                COMMAND                  CREATED        STATUS        PORTS        NAMES
324c1323c099   172.16.0.20:5005/dev-projects/wildfire-ai-app/wildfire-detection-aiapp:2.0.0   "/opt/nvidia/nvidia_..."  11 days ago   Up 5 minutes (healthy)   wildfire-aiapp-Demokritos-1

csmsote@gpuserver:~$ sudo docker inspect wildfire-aiapp-Demokritos-1 --format '{{.Name}}\n{{.Config.Image}}\n{{.State.Status}} (health: {{if .State.Health}}{{.State.Health.Status}}{{else}}n/a{{end}})\n{{.State.StartedAt}}\n{{.HostConfig.RestartPolicy.Name}} (count: {{.RestartCount}})\nCPUs: {{if .HostConfig.NanoCpus}}{{.HostConfig.NanoCpus}} CPUs (unlimited){{else}}unlimited{{end}}\nMemory: {{if .HostConfig.Memory}}{{.HostConfig.Memory}} bytes (unlimited){{else}}unlimited{{end}}\nGPU: {{range .HostConfig.DeviceRequests}}{{range .DeviceIDs}}{{.}} {{end}}{{end}}\nCmd: {{json .Config.Cmd}}\nMounts: {{range .Mounts}}{{.Source}} -> {{.Destination}}\n{{end}}'
Name:          /wildfire-aiapp-Demokritos-1
Image:         172.16.0.20:5005/dev-projects/wildfire-ai-app/wildfire-detection-aiapp:2.0.0
Status:        running (health: healthy)
Started:       2026-04-01T08:18:23.907831500Z
Restart:       unless-stopped (count: 2)
CPUs:          0 CPUs (unlimited)
Memory:        0 bytes (unlimited)
GPU:           7
Cmd:           ["python", "detect_from_IPcam.py", "--site_name", "Demokritos", "--cam_id", "1"]
Mounts:        /data/Wildfire_Detections -> /data/Wildfire_Detections
               /data/Wildfire_Configs -> /configs

csmsote@gpuserver:~$ sudo docker stats wildfire-aiapp-Demokritos-1 --no-stream --format "table {{.Name}}\t\t{{.CPUPerc}}\t\t{{.MemUsage}}\t\t{{.NetIO}}\t\t{{.PIDs}}"
NAME                CPU %    MEM USAGE / LIMIT   NET I/O   PIDS
wildfire-aiapp-Demokritos-1  198.86%  1019MiB / 125.6GiB   94.8MB / 893kB   361

csmsote@gpuserver:~$ sudo nvidia-smi -i 7 --query-gpu=index,name,memory.used,memory.total,utilization.gpu,temperature.gpu,power.draw,power.limit --format=csv
index,name,memory.used [MiB],memory.total [MiB],utilization.gpu [%],temperature.gpu, power.draw [W], power.limit [W]
7, NVIDIA A10, 1400 MiB, 23028 MiB, 65 %, 76, 130.96 W, 150.00 W

csmsote@gpuserver:~$ nvidia-smi -i 7
Wed Apr  1 11:24:54 2026

+-----+
| NVIDIA-SMI 535.288.01                  Driver Version: 535.288.01   CUDA Version: 12.2   |
+-----+
| GPU  Name           Persistence-M   Bus-Id        Disp.A     Volatile Uncorr. ECC  |
| Fan  Temp            Perf             Pwr:Usage/Cap     Memory-Usage     GPU-Util    Compute M. |
|              |              |              |              |              |              |
+-----+-----+
|  7  NVIDIA A10      Off              00000000:CC:00:0 Off         1440MiB / 23028MiB      65%          Default  |
|  0%   76C           P0              147W / 150W           |                      |
+-----+-----+
+-----+
| Processes: |
| GPU   GI       CI           PID    Type    Process name                        GPU Memory |
| ID    ID       ID           |              |              |              |
+-----+-----+
|  7    N/A      N/A       3753938    C       python                             1432MiB |
+-----+-----+
```

Figure 83 Containerized deployment validation of the AI-App, showing healthy Docker runtime state, mounted NAS volumes, assigned GPU device, and NVIDIA A10 utilization metrics

```

=====
== CUDA ==
=====

CUDA Version 11.8.0

Container image Copyright (c) 2016-2023, NVIDIA CORPORATION & AFFILIATES. All rights reserved.

This container image and its contents are governed by the NVIDIA Deep Learning Container License.
By pulling and using the container, you accept the terms and conditions of this license:
https://developer.nvidia.com/ngc/nvidia-deep-learning-container-license

A copy of this license is made available in this container at /NGC-DL-CONTAINER-LICENSE for your convenience.

[CLEANUP] No orphaned files found
Site: Demokritos
Prefecture: Athina
Stream number: 1
Camera: Hikvision

CUDA is available: True
YOLO device: cuda:0
...CONNECTING TO BROKER...
[INFO] Connected to broker
Start Receiving frames ...!
[2026-04-01T08:18:38] frame=100, chunk_frames=100, avg_fps=9.24, avg_pre_ms=10.85, avg_infer_ms=48.25, avg_post_ms=0.94, detections=0, non_detections=100
[2026-04-01T08:18:45] frame=200, chunk_frames=100, avg_fps=13.48, avg_pre_ms=10.89, avg_infer_ms=47.89, avg_post_ms=0.86, detections=0, non_detections=100
[2026-04-01T08:18:53] frame=300, chunk_frames=100, avg_fps=13.23, avg_pre_ms=11.09, avg_infer_ms=47.69, avg_post_ms=1.00, detections=0, non_detections=100
[2026-04-01T08:19:00] frame=400, chunk_frames=100, avg_fps=13.39, avg_pre_ms=10.43, avg_infer_ms=48.09, avg_post_ms=0.81, detections=0, non_detections=100

```

Figure 84 AI-App runtime validation logs showing CUDA initialization, MQTT broker connectivity, RTSP stream acquisition, and continuous real-time frame inference

The screenshot shows the MQTT Explorer interface. On the left, there's a sidebar with 'versus-master' and a 'New Subscription' button. The main area displays a message from the topic 'testing/wildfire' with QoS 0. The message content is a JSON object containing metadata and sensor data.

```

Topic: testing/wildfire  QoS: 0

{"application_id": "Wildfire-detection", "site_id": "Demokritos", "site_name": "Demokritos", "cam_name": "Hikvision", "cam_lat": "37.996140", "cam_long": "23.818520", "direction": "N/A", "cam_id": "1", "cam_ip": "10.220.2.45", "cam_ui_port": "80", "heartbeat": 1, "model": "YOLO", "prefecture": "Athina", "object": "fire", "detection_status": "not detected", "rtsp_string": "rtsp://admin:c0sm0te123@10.220.2.45/ISAPI/Streaming/channels/101", "frames_per_second": 12.823527903122278, "temperature": 13.63, "humidity": 78.0, "wind_speed": 12.35, "weather_status": "Clouds"}

2026-04-02 13:47:18:678

```

Figure 85 MQTT message example that AI-app sends

5.3.3 AI-App for Location Awareness

5.3.3.1 Overview

The Location Awareness AI-App analyses user equipment (UE) mobility patterns in order to support situational awareness during emergency events such as wildfire detection. The application receives hazard alerts from the monitoring infrastructure and combines them with network-derived UE location reports to estimate whether device activity in the affected area corresponds to a meaningful event.

5.3.3.2 Architecture

The AI-App follows a modular, event-driven architecture within the triplet (AI-App, N-App, V-App), supporting continuous ingestion of UE location updates and real-time processing. At the core of the system lies a temporal deep learning model based on a causal 1D CNN, designed to capture user mobility patterns over time. The model processes sliding windows of $T = 25$ -time steps, using features such as dwell time, consecutive unsafe ticks, cell changes, and unsafe area indicators. The network consists of stacked causal

convolutional layers with increasing dilation, followed by global pooling and a fully connected layer that outputs a risk probability. A fixed threshold is applied to generate alert decisions. This architecture enables efficient real-time inference while capturing both short-term and long-term temporal behaviour.

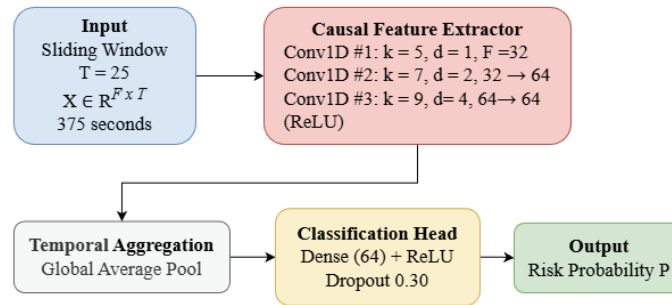


Figure 86 AI-App for Location Awareness-AI-Model Architecture

5.3.3.3 Data Pre-Processing

The model is trained on a time-series dataset that was generated utilising real measurements in the Athens B5G testbed, that captures UE mobility between safe and unsafe network cells, explicitly modeling user behavior over time. The dataset consists of multiple users (default: 100), each observed over approximately 150 time steps, with a sampling interval of 15 seconds, corresponding to periodic location updates from the N-App. This results in trajectories spanning roughly 30 minutes per user.

At each time step, the following features are generated:

- **dwel_s**: accumulated time spent in the current cell
- **consec_unsafe_ticks**: number of consecutive observations in unsafe cells
- **changes_{last_T}**: number of cell transitions within a recent time window (~5 minutes)
- **unsafe_frac_T**: fraction of time spent in unsafe cells within the window
- **is_unsafe_cell**: binary indicator of unsafe cell presence

The target label (risk_alert) is generated based on the duration of continuous exposure to unsafe cells. Specifically, when a user remains in an unsafe area, the probability of triggering an alert follows a sigmoid function centered at 300 seconds (~5 minutes). This probabilistic labelling introduces a smooth transition between normal and risky behaviour, avoiding unrealistic hard thresholds. For model training, the dataset is transformed into sliding windows of fixed length ($T = 25$ time steps), where each window represents a short temporal sequence of user activity. The label assigned to each window corresponds to the final time step, enabling the model to learn temporal dependencies leading to risk conditions. During preprocessing, continuous features are normalized using standard scaling, while binary features remain unchanged. The dataset is split at the user level into training, validation, and test sets, ensuring that sequences from the same user do not appear in multiple splits and preventing data leakage. This structured dataset design allows the model to learn realistic temporal patterns of mobility while ensuring consistency between training and real-time inference.

5.3.3.4 Training, Validation and Inference

The model is trained using a supervised learning approach on time-series windows extracted from the dataset. Each training sample corresponds to a sequence of length $T = 25$ time steps, with the label defined at the final time step of the window. The dataset is split at the user level into training, validation, and test sets (70% / 15% / 15%), ensuring that user trajectories are not shared across splits. Sliding windows are generated with stride 1, producing overlapping sequences that preserve temporal continuity. During training, continuous features are normalized using a StandardScaler, while binary features remain unchanged. The model is trained using binary cross-entropy loss, with class weighting applied to address class imbalance (higher weight on alert

cases). Optimization is performed using the AdamW optimizer, and early stopping is applied based on validation performance. Model selection is based on validation Average Precision (AP), which is suitable for imbalanced classification. After training, a decision threshold ($\tau \approx 0.75$) is selected either as a fixed value or based on validation performance (e.g., maximizing macro F1-score). For evaluation, the model is tested on unseen users, reporting metrics such as accuracy, precision, recall, macro F1-score, ROC-AUC, and confusion matrix. During inference, the AI-App operates in a real-time streaming mode, processing incoming UE location registration events from the N-App for Location Awareness. For each user, a stateful component maintains a sliding window of recent observations.

At each new event:

- Features are computed incrementally (dwell time, unsafe duration, transitions)
- A fixed-length window is updated
- Once sufficient history is available (T steps), the model performs inference

The input is normalized using the same scaler as in training and passed through the trained CNN model. The output is a probability score, which is compared against the threshold τ to produce a binary decision. If the predicted probability exceeds the threshold, a risk alert is triggered, otherwise no alert is issued. A warm-up phase is required before inference begins, ensuring that predictions are based on sufficient temporal context. This design enables continuous monitoring of user behaviour while maintaining consistency between offline training and real-time deployment.

5.3.3.5 Connection to MLOps

The current implementation focuses on stable real-time inference and system integration. Full MLOps integration is planned for Phase 2 and will include model versioning, monitoring, and automated retraining. The system is designed to remain compatible with centralized MLOps pipelines while supporting low-latency inference requirements.

5.3.3.6 Development, Deployment and Results

The AI-App is implemented in Python as a containerized service using FastAPI for communication and PyTorch for model inference. It interacts with the N-App through REST-based APIs and supports callback-based data delivery. It also receives wildfire alerts via MQTT communication with the V-App Wildfire Camera. The system is designed as a modular pipeline including:

- UE location data ingestion
- temporal feature computation
- sliding window management
- real-time model inference
- alert generation and logging

The service operates continuously and processes incoming events asynchronously, enabling scalable deployment across multiple users.

The AI-App is deployed as a containerized service in a cloud environment. It communicates with the N-App via REST APIs and callback mechanisms, enabling continuous reception of UE location updates. Each instance maintains per-user state and performs real-time inference. The architecture supports scalable deployment across multiple users while maintaining low latency.

The code is uploaded here: <https://gitlab.6g-versus.eu/wp4/ai-apps/greek-cluster/ai-app-location>

The performance of the proposed AI-App model was evaluated on the test set using multiple metrics, including confusion matrix, loss curves, and ROC analysis (Figures 87, 88, 89).

Classification Performance

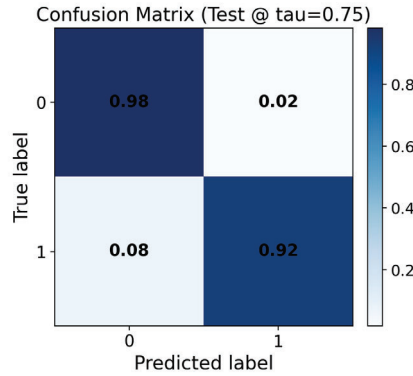


Figure 87 Confusion Matrix of the location awareness AI-App

The confusion matrix at threshold $\tau = 0.75$ shows strong classification performance:

- **True Negative Rate (class 0): 0.98**
- **False Positive Rate: 0.02**
- **True Positive Rate (Recall for class 1): 0.92**
- **False Negative Rate: 0.08**

This means that:

- The model almost never raises false alarms (only 2%)
- It successfully detects the majority of risky situations (92%)

Training Convergence

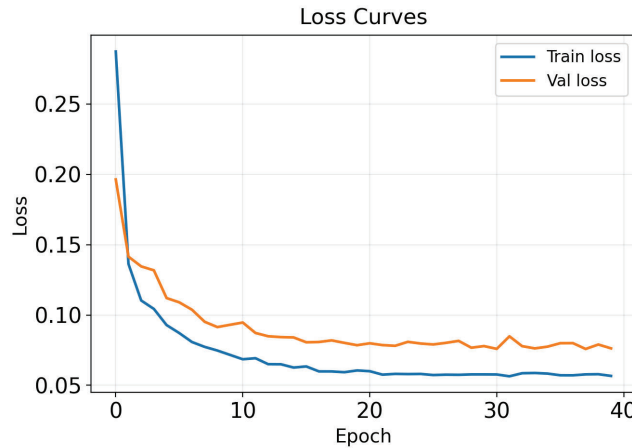


Figure 88 Training and Validation Loss Curves of the location awareness AI-App

The loss curves show stable convergence:

- Training loss decreases rapidly and stabilizes around **0.055–0.06**
- Validation loss follows a similar trend and stabilizes around **0.075–0.08**

This indicates:

- The model learns meaningful patterns early
- There is **no strong overfitting**, since validation closely follows training

ROC Analysis

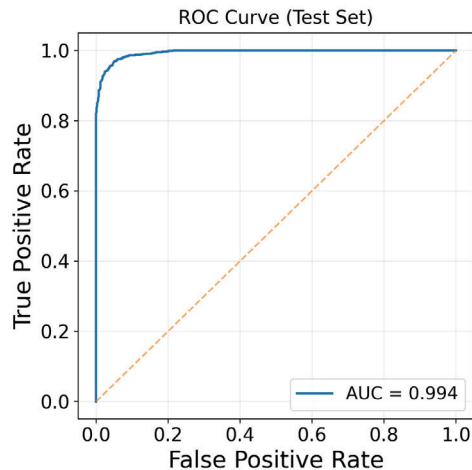


Figure 89 Training and Validation Loss Curves of the location awareness AI-App

The ROC curve demonstrates excellent discriminative ability:

- AUC = 0.994

This value is very close to 1, meaning:

- The model can almost perfectly separate safe from unsafe behaviour
- For almost any threshold, it achieves high true positive rate with very low false positive rate

5.3.4 9091AI-Apps under Development for Phase 2

5.3.4.1 AI-App for Early Wildfire Monitoring

For Phase 2, the AI-App for Early Wildfire Monitoring will focus on the integration of an MLOps layer based on MLflow and Airflow. The planned development includes storing and managing trained smoke detection models in MLflow, enabling model version control and simplified retrieval of the selected production model. In parallel, Airflow workers will be designed to schedule and orchestrate the execution of the AI-App on GPU-enabled worker nodes, replacing the current standalone Docker-based execution model with a more structured workflow-driven deployment approach. In addition, future work will also explore compatibility with the N-App for QoS-related functionalities, with the goal of enabling smoother interaction between the triplets.

5.3.4.2 AI-App for Location Awareness

The ongoing development of the AI-App focuses on the retraining of the model using newly collected data, with the goal of refining its performance, improving robustness, and better adapting to real-world conditions. In parallel, the work focuses on the finalization and stabilization of its integration within the overall system architecture. A key priority is the completion of the integration with the V-App wildfire camera, ensuring seamless event triggering. Further integration efforts are being carried out with the V-App INUS, aiming to extend the AI-App's capabilities in handling its outputs and strengthening its role within the broader ecosystem. These efforts are complemented by continued work on integrating the AI-App with MLOps pipelines, enabling systematic deployment and monitoring.

5.3.4.3 AI-App Detection & tracking

For Phase 2, the AI-App for Detection and Tracking will be further developed in order to improve the robustness and operational maturity of the current tracking-by-detection pipeline based on YOLO11 and ByteTrack. The next development stage will focus on improving the performance of the AI-App under more challenging field conditions, such as aerial viewpoints, varying object scales, scene clutter, and dynamic outdoor environments. In this context, future work will include further model refinement through custom training and transfer learning processes, as well as the possible extension of the supported object classes

according to the needs of the Greek pilot scenarios. In addition, Phase 2 will include further testing and validation under more complex and realistic operational scenarios, so as to assess the behaviour of the AI-App under conditions that more closely reflect actual field deployments. This will support a better understanding of the robustness of the detection and tracking pipeline in the presence of heterogeneous assets, moving targets, changing viewpoints, and more demanding environmental conditions.

5.3.4.4 AI-App Virtual Fence and Prohibited Area Detection

For Phase 2, the AI-App Virtual Fence and Prohibited Area Detection will be further developed in order to provide a more mature and robust AI-based capability for detecting intrusions, unauthorized presence, and activity within predefined virtual boundaries and restricted zones. Building on the initial prototype, the next development stage will focus on improving the reliability and accuracy of the detection pipeline under more demanding field conditions, including dynamic outdoor environments, varying viewpoints, partial occlusions, and heterogeneous sensing conditions. Emphasis will be placed on supporting more reliable identification of events related to virtual fence crossing and prohibited area violations in operational scenarios relevant to the Greek pilot. In addition, Phase 2 will include further testing and validation under more complex and realistic operational scenarios, so as to better assess the behaviour of the AI-App under conditions closer to actual field deployment. This will support the evaluation of the AI-App in the presence of multiple moving targets, changing scene configurations, and more challenging environmental conditions, while also enabling refinement of the event-generation logic associated with virtual perimeter monitoring. At the same time, further work will target tighter integration of the AI-App with the corresponding V-App and the broader application ecosystem, enabling more reliable alert generation, situational awareness support, and end-to-end interoperability within the Greek cluster. The AI-App will also be progressively aligned with the project MLOps framework in order to support improved model lifecycle management, monitoring, and future retraining based on operational data collected during the trials. These developments will support the transition of the AI-App from an initial prototype into a more robust, scalable, and deployment-ready AI service for Phase 2.

5.4 French Cluster

For Phase 1 (Trials 1 and 2), the AI-App RAN Optimization for Battery Lifetime Extension will be showcased for UC2. It improves sensor battery life by dynamically adapting RAN parameters (e.g., DRX) based on traffic and radio conditions, ensuring energy efficiency while maintaining reliable connectivity. For Phase 2 (Trials 3), more advanced AI-Apps will be introduced. The AI-App Sensors Virtualization will reduce reliance on physical sensors by clustering infrastructure and inferring environmental conditions. The AI-App Clogging Detection will identify potential clogging events in wastewater networks using sensor and rainfall data. The AI-App RAN Optimization for Battery Lifetime Extension will also be further enhanced and integrated by retraining the models. These applications are currently under development and will be validated during Phase 2 trials.

Table 29 Implemented and under development AI-Apps in the French cluster

Cluster Name	App Name	Phase (1or 2)
French	AI-App Sensors Virtualization	2
French	AI-App Clogging Detection	2
French	AI-App RAN optimization for battery lifetime extension	1 and 2 (Retraining)

5.4.1 AI-App RAN optimization for battery lifetime extension

5.4.1.1 Overview

A first version of this AI-App will be implemented for the first phase of trials, but the main results will be provided during phase 2. The AI application is designed to extend the battery lifetime of sensor nodes by intelligently reducing their power consumption while still ensuring optimal connectivity performance,

particularly in terms of data rate and latency. To achieve this balance, the AI dynamically configures key Radio Access Network (RAN) features such as Discontinuous Reception (DRX). These parameters are adjusted according to the actual traffic demand, allowing the network to avoid unnecessary energy expenditure during low-activity periods while maintaining reliable communication when data transmission is required for sensing purpose.

5.4.1.2 Architecture

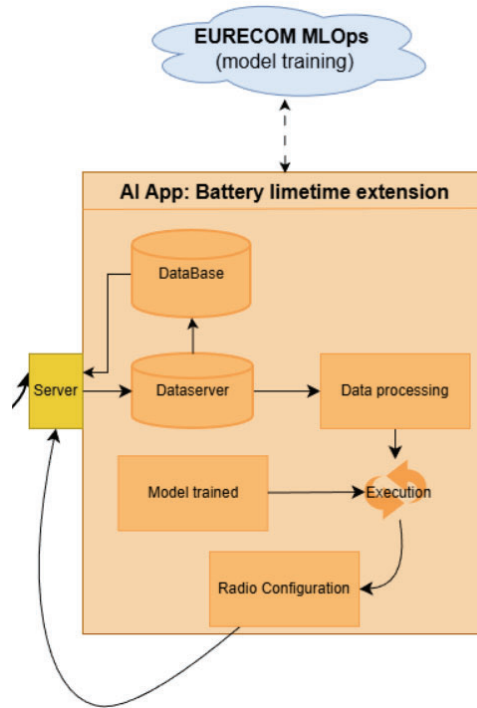


Figure 92 AI-App RAN optimization for battery lifetime extension

This AI-app, as seen in Figure 92, is composed of multiple blocks. The database contains all of radio measurements from the RAN that are essential to train the model and also to make the inference. The AI-App is used as a server, linked to the N-App to receive the data from the RAN and also to send a new radio configuration when needed. There is a data processing before executing the model. The model is trained in the MLOps platform, as MLOps ensures reliable and efficient ML model training. Then, the trained model is stored in the AI-App. That's why the inference is made locally, in the AI-App.

5.4.1.3 Data Preprocessing

In its initial implementation phase, sensor data traffic is emulated to replicate real-world conditions, specifically modelling the traffic patterns of field sensors used for watering green spaces. **Error! Reference source not found.**

5.4.1.4 Training, Validation and Inference

The AI model used for this first implementation is a DQN (Deep-Q-Network). The objective function is a weighted sum, composed of a latency, a power consumption and the number of correctly transmitted packets. Maximising the trade-off between those metrics will optimise the energy efficiency of the sensors.

The model has been trained and validated with the simulated traffic based on real traffic of the sensors and is represented Figure 85. The inference has been made on the simulated traffic. The traffic has multiple profiles and allows the training and testing datasets to be diverse.

5.4.1.5 Connection to MLOps

For the first trial, the AI-App will be connected to EURECOM MLOps in order to use it to train the model. It will also store the necessary data to train the model. The data will be based on radio measurements obtained in laboratory conditions.

The deployment will be made during the first trial. A first set of data has been stored in the MLOps and when it is ready, the ML code will be transferred as a containerized app, using Docker, to the MLOps. As it is on a development phase, more details on the results will be given on the next deliverable.

5.4.1.6 Development, Deployment and Results

This AI-App is developed in Python. It is not present in the GitLab since the code is proprietary and for privacy concerns cannot be shared.

In practice, the neural network acts as an agent. Its inputs are the traffic state (latency, throughput, Signal-To-Noise-Ratio), and is composed of 5 parameters (among 7 state parameters in total). The outputs are radio parameters that configure the DRX: *DRX_on*, *DRX_cycle*, *DRX_offset*) with 156 possible configurations.

This is an initial version of an AI model. Performances of this AI model are displayed hereafter. Next step is to compare these results with a new AI model trained on MLOps.

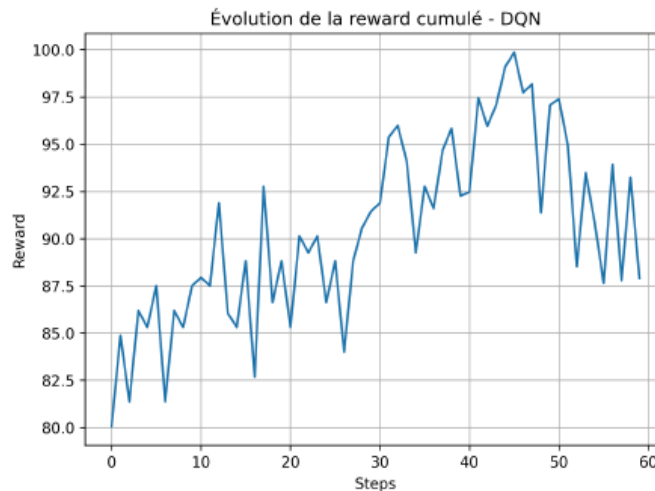


Figure 93 Evolution of the cumulated reward as a function of the steps

Figure 93 is obtained during the training of the DQN. The cumulated reward is the value to maximize. One can observe that it is increasing until step 45. This increase in reward can be attributed to two related factors:

- the gradual reduction in exploration (ϵ -greedy strategy), which leads the agent to choose optimal actions more often,
- the continuous improvement of the policy through learning.

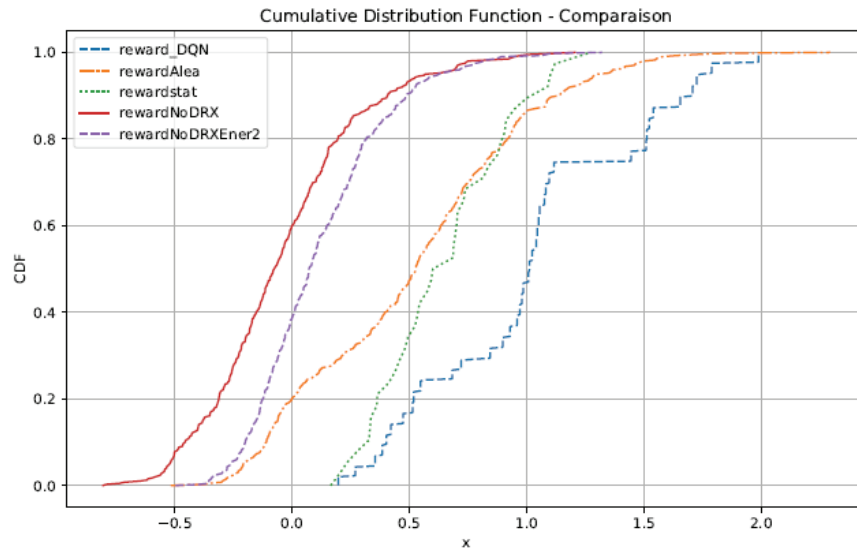


Figure 94 Cumulative Distribution Function on the reward/objective function

Figure 94 shows that the Cumulative Distribution Function of the reward obtained by the AI model (“reward_DQN”) is higher than all the other configurations. The configuration “rewardAlea” is one random DRX configuration, while “rewardstat” is an average of the random configurations. The worst configurations are “rewardNoDRX” and “rewardNoDRXEner2” that are respectively configurations without DRX with 50MHz of bandwidth and without DRX with 20MHz of bandwidth.

It shows that DRX is essential to maximise energy efficiency and that the DQN has maximised the trade-off between performance and energy consumption.

5.4.2 AI-Apps under Development for Phase 2

5.4.2.1 Enhanced battery lifetime of sensors

For this AI-App, the main objective during phase 2 will be to optimise the model. The latter will be compared to other models in order to choose the right one. A phase of retraining can also be important to adapt the model to various traffic conditions.

5.4.2.2 Sensors Virtualization

The AI-App “Sensors Virtualization” enables the virtualization of sensors in rainwater networks, improving infrastructure monitoring while reducing dependence on physical devices. Through AI-driven analytics and predictive modelling, utilities can limit large-scale sensor deployments and infer environmental conditions from a minimal set of hardware inputs. The first step of virtualization consists in clustering the gullies to form clusters with homogeneous behavior. The second phase involves defining cleaning intervention rules for gullies using sensor measurements collected from instrumented gullies. Observations of clogging levels recorded during maintenance operations are fed back into the model to improve the clustering. The AI-App operates as part of the triplet, consuming sensor measurements provided by the N-App as well as contextual data from the V-App. The AI-App is deployed in the cloud to ensure high availability and unique deployment site for all customers.

5.4.2.3 Clogging detection

The AI-App Clogging Detection provides an advanced service for detecting abnormal behavior in wastewater networks, with the objective of identifying potential clogging events hot spots. The application relies on the analysis of water level measurements collected from sensors installed in the network, combined with rainfall data that provide environmental context for interpreting hydraulic conditions. By correlating these data

sources, the AI-App is able to identify patterns that may indicate abnormal load of water leading to reduced hydraulic performance. In an initial phase, the inference service is deployed on a public cloud infrastructure in order to facilitate model development, testing, and large-scale data processing. This environment allows the training and validation of AI models using historical datasets and real-time sensor streams. Once validated, the service is containerized to ensure portability and reproducibility of the execution environment. The containerized AI service is then deployed on the EURECOM testbed, where it operates within the B5G experimentation infrastructure. The AI-App operates as part of the triplet architecture, interacting with the other application layers of the system. It consumes sensor measurements provided by the N-App, which handles data acquisition from the field devices, as well as contextual and aggregated data supplied by the V-App. By combining these data streams, the AI-App performs inference and generates indicators of potential clogging situations, supporting operators in monitoring the network and prioritizing maintenance interventions

5.5 Portuguese Cluster

For Phase 1 (Trials 1, 2 and 3), the cluster will showcase the Mobitrust AI-App (Scenario 1 and 2), Simple Object Detection System AI-App (Scenario 2) and the HazMat Vehicles Detector AI-App (Scenario 2). The Mobitrust AI-App provides advanced capabilities such as object detection and identification, and streaming optimisation to the Mobitrust platform. The Simple Object Detection System AI-App provides a simple object detection system that allows vehicle identification involving low-resources imaging hardware. The HazMat Vehicles Detector AI-App is a powerful image analysis system, supported by dynamically instantiated network and compute elements, supported by more powerful GPU-based hardware, which is activated on demand, after the initial triggering from the Simple Object Detection System AI-App. For Phase 2 (Trials 4, 5, 6, and 7), the cluster will add the IoT Device Energy Management AI-App (Scenario 1) and the Infrastructure Energy Management AI-App (Scenario 3). The IoT Device Energy Management AI-App aims to provide device and communication configuration (e.g. QoS) recommendations considering current and estimated device status (e.g. harvested energy availability) and information such as business objectives or rules (e.g. data collection frequency, IoT measurements) or energy-driven objectives (e.g. IoT device energy consumption, minimal battery levels thresholds). Regarding the Scenario 3 apps, during Phase 1 work focuses on the V-App baseline and infrastructure-side data collection/control functions; the AI-driven optimisation component is planned for Phase 2 through the Infrastructure Energy Management AI-App, which purpose is to act on vertical systems and optimising resources such as EV charging and public lighting.

Table 30 Implemented and under development AI-Apps in the Portuguese cluster

Cluster Name	App Name	Phase (1or 2)
Portuguese	Mobitrust AI-App	Phase 1
Portuguese	Simple Object Detection System AI-App	Phase 1
Portuguese	HazMat Vehicles Detector AI-App	Phase 1
Portuguese	IoT Device Energy Management AI-App	Phase 2
Portuguese	Infrastructure Energy Management AI-App	Phase 2

5.5.1 Mobitrust AI-App

5.5.1.1 Overview

The Mobitrust AI-App constitutes the intelligent component of the Mobitrust V-App (owned by OneSource), therefore, the effort within the AI-app scope is focused on the integration and extension of the triplet's architecture as described in the N-App 3.6. The AI-App is responsible for providing advanced capabilities such

as object detection and identification, and streaming optimisation. Initially embedded within the Mobitrust V-App, the AI functionalities were already fully developed.

5.5.1.2 Architecture

The Mobitrust AI-App consists of a decoupled component, previously integrated in the Mobitrust Platform, that follows the 6G-VERSUS architecture to process requests from the V-App.

5.5.1.3 Data Preprocessing

The Mobitrust AI-App processes individual frames in video streams from connected devices.

5.5.1.4 Training, Evaluation and Inference of AI Models

The AI models training and evaluation are not included in the 6G-VERSUS. This AI-App was included to support the validation and evaluation of the triplet..

5.5.1.5 Connection to MLOps

The Mobitrust AI-App does not integrate with the MLOps framework.

The Mobitrust AI-App is deployed as a cloud-native application. To ensure compliance with the project's triplet architecture, the AI-App was decoupled from the V-App and implemented as an independent entity. In its current form, the AI-App interacts autonomously with both the V-App and the N-App, following the triplet architecture, enabling real-time analytics, optimized video processing, and coordinated network-aware adaptations that leverage advanced B5G/6G capabilities.

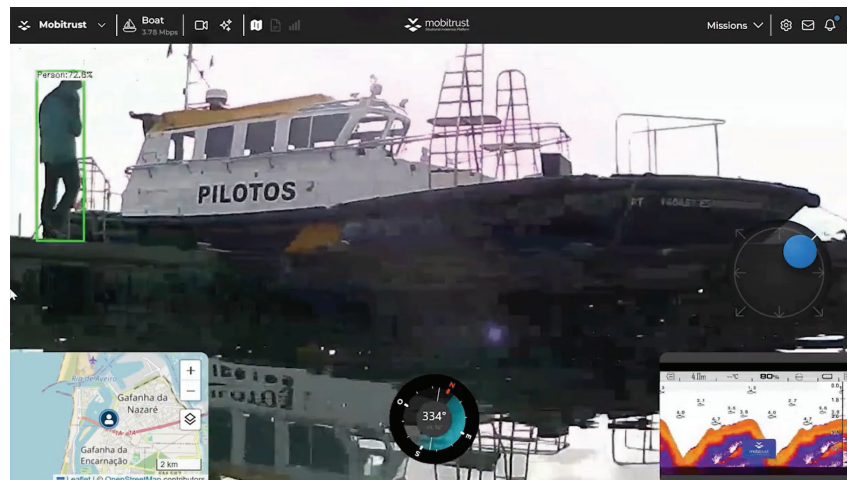


Figure 95 - Mobitrust AI-App in the Portuguese UC trial

5.5.1.6 Development, Deployment and Results

The Mobitrust AI-App development is not included in the 6G-VERSUS project, only it's integration with the triplet, and the clusters trials and pilots.

5.5.2 Simple Object Detection System AI-App

5.5.2.1 Overview

The Simple Object Detection System AI-App is aimed at detecting a vehicle approaching a specific area, using a low-resolution video stream (360p) with moderate hardware resource and energy consumption. Upon detecting a new vehicle, it triggers the instantiation of more powerful image characterisation procedures performed by the HazMat Vehicles Detector AI-App, enabling the acquisition of relevant vehicle identification

elements. Within this architecture, AgentA performs real-time truck detection on a 360p video stream using the default YOLOv11 model from Ultralytics (<https://docs.ultralytics.com/models/yolo11/>) with only the "Truck" class selected, generating a unique truck identifier and publishing a detection event to an Apache Kafka topic. These events act as the trigger mechanism for activating the subsequent high-resolution analysis stage, ensuring that advanced processing resources are only engaged on demand.

5.5.2.2 Architecture

The Simple Object Detection System AI-App is implemented as a single containerized component, AgentA, following an event-driven, state-machine architecture. AgentA connects to a 360p RTSP video stream provided by a MediaMTX streaming middleware server and continuously processes incoming frames using the YOLOv11 object detection model filtered to the truck class. A dedicated background thread (StreamManager) reads frames from the stream into a single-slot queue, always exposing the most recent frame to the inference loop to minimise latency. Upon detecting a truck with confidence above the threshold, AgentA generates a unique identifier (format TRK{uuid}) and publishes a JSON event to the per-gate Kafka topic truck-detected-{GATE_ID}, with the truck identifier encoded in the message header. The agent then enters an awaiting-reset state, suspending further detection until it receives a reset signal via the reset-agentA-{GATE_ID} topic relayed by the AI Gateway from the V-App (V_Brain component), or until the 60-second decision timeout elapses. The AI Gateway also acts as the Kafka-to-HTTP bridge between the AI-App and V-App layers. The Simple Object Detection System AI-App is closely related to the HazMat Vehicles Detector AI-App. The overall architecture is depicted in Figure 96.

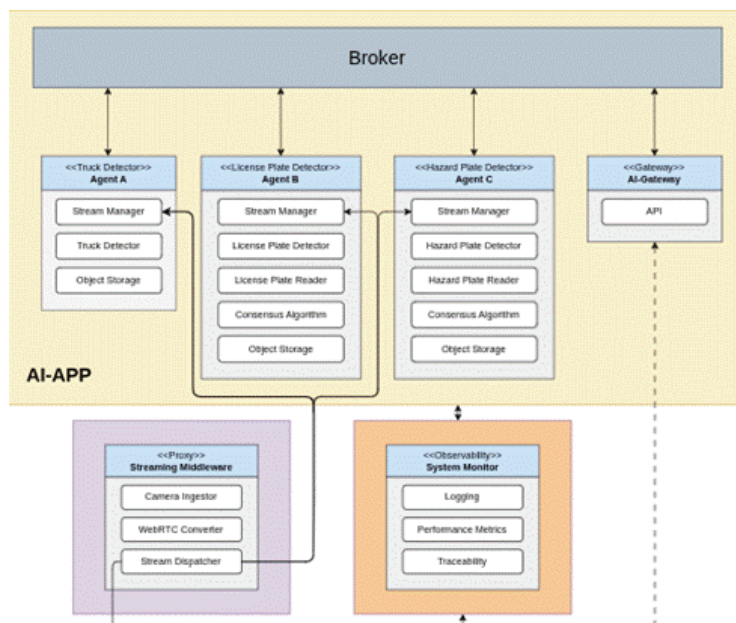


Figure 96 Technological architecture of Simple Object detection and Hazmat truck detection AI-Apps

5.5.2.3 Data Preprocessing

AgentA processes individual video frames extracted live from the RTSP stream. Each frame is passed directly to the YOLOv11 model without additional preprocessing, as the model handles input normalisation internally. The **StreamManager** component runs in a background thread and discards stale frames automatically, ensuring inference always operates on the most recent data.

5.5.2.4 Training, Evaluation and Inference of AI Models

AgentA uses a YOLOv11 model pre-trained on the COCO dataset and further fine-tuned for truck detection in port and logistics environments. The model file (**truck_model.pt**) is loaded at container startup from the configured **MODELS_PATH**. During inference, bounding boxes are filtered to class ID 7 (truck) with a

minimum confidence threshold of 0.75. Per-frame inference latency is tracked via a Prometheus histogram (`agent_a_inference_latency_seconds`) and total detection count via a counter (`agent_a_trucks_detected_total`).

5.5.2.5 Connection to MLOps

AgentA does not currently integrate with a MLOps platform. The model is loaded from a static file bundled with the container at startup. Future work may include integration with an MLOps framework for model versioning, automated retraining pipelines, and continuous performance monitoring.

5.5.2.6 Development-Deployment and Results

AgentA is implemented in Python using the following technologies:

- **Ultralytics YOLOv11** (<https://docs.ultralytics.com/models/yolo11/>) — object detection inference (truck class, ID 7 in COCO)
- **OpenCV** (<https://opencv.org/>) — video frame capture and stream management
- **confluent-kafka** (<https://github.com/confluentinc/confluent-kafka-python>) — Kafka producer and consumer via project-shared **KafkaProducerWrapper/KafkaConsumerWrapper**
- **MinIO Python SDK** (<https://www.min.io/>) — upload of annotated frames to S3-compatible object storage
- **Pydantic BaseSettings** (<https://docs.pydantic.dev/>) — environment-variable-based configuration with validation
- **prometheus_client** (https://github.com/prometheus/client_python) — Prometheus metrics exposition (inference latency, frames processed, trucks detected, detection confidence)

All service configuration is provided via environment variables (**KAFKA_BOOTSTRAP**, **MEDIAMTX_HOST**, **GATE_ID**, **MODELS_PATH**, etc.), following the Twelve-Factor App methodology. The service is packaged as a Docker container using a multi-stage Dockerfile.

The source code is available at the project GitLab repository (<https://gitlab.6g-versus.eu/wp4/ai-apps/portuguese-cluster>) under **src/AI_APP/agentA/**.

AgentA is deployed as a containerized service using Docker Compose. Two instances are deployed simultaneously, `agentA-1` for gate 1 and `agentA-2` for gate 2 (signaling the camera where the infraction may occur), each configured with its respective `GATE_ID` environment variable. Both instances share a dedicated NVIDIA GPU. Container images are built from source using multi-stage Dockerfiles and deployed on a dedicated GPU-enabled virtual machine in the testbed infrastructure. Container startup is sequenced via Docker Compose health checks: the Kafka broker must be healthy and the `kafka-init` service (responsible for topic creation) must complete successfully before AgentA containers start. For Trial 2, the full AI-App stack, including both AgentA instances, was deployed on the ITAV testbed, processing live 360p video feeds from gate cameras at the port demonstration environment and successfully triggering the HazMat detection pipeline in real time.

The Simple Object Detection System AI-App has been fully developed, integrated, and validated in conjunction with the associated V-App and the HazMat Vehicles Detector AI-App to ensure full functional interoperability. The first public demonstration took place on 25/3/2026 at the Port of Aveiro.

Figure 97 shows the initial truck detection using the YOLOv11 model on a 360p stream, acting as the trigger for the high-resolution analysis stage.



Figure 97 Initial truck detection

5.5.3 HazMat Vehicles Detector AI-App

5.5.3.1 Overview

The HazMat Vehicles Detector AI-App is a high-resolution image analysis system that is activated on demand, triggered by detection events from the Simple Object Detection System AI-App, and supported by GPU-accelerated hardware and 4K video streams. Upon activation, the system simultaneously identifies the vehicle license plate and any ADR hazard plates (displaying Kemler codes and UN numbers for dangerous goods classification) using parallel YOLO-based detection pipelines combined with PaddleOCR text extraction. A multi-frame consensus algorithm aggregates recognition results across up to 150 frames per activation (configured via Docker Compose), improving accuracy and robustness against transient occlusions or motion blur. Recognised codes, together with the corresponding cropped images, are published via Apache Kafka and stored in MinIO object storage, ensuring full traceability and seamless integration with the V-App decision engine. Once processing is complete, the system returns to a low-resource standby state, with network and compute resources released accordingly.

5.5.3.2 Architecture

The HazMat Vehicles Detector AI-App consists of two independently containerized agents, AgentB (licence plate detector) and AgentC (hazard plate detector), which are activated concurrently upon receiving a truck detection event from AgentA. Both agents inherit from a shared BaseAgent class that implements the common detection pipeline using the Template Method design pattern, ensuring code reuse and consistent behaviour. Upon consuming a truck-detected-`{GATE_ID}` event from Kafka, each agent independently connects to the high-resolution 4K RTSP stream, processes up to 150 frames (configured via Docker Compose), and applies a custom-trained YOLOv11 model to detect its respective target object (licence plate or ADR hazard plate). Detected image crops are extracted and passed to a PaddleOCR engine. The resulting text and confidence scores are fed into a ConsensusAlgorithm that performs character-level position voting across frames. Once consensus is reached (at least 80% of character positions decided) or the frame budget is exhausted, the best crop is selected via Levenshtein distance to the consensus text and uploaded to a per-agent MinIO bucket, while the final recognition result is published to the respective Kafka topic (`lp-results-{GATE_ID}` or `hz-results-{GATE_ID}`) with the truck identifier propagated in the message header. The AI Gateway relays these results to the V-App business logic layer via HTTP. The overall AI-App architecture is depicted in 98Figure 86.

5.5.3.3 Data Preprocessing

Image crops extracted from YOLO bounding boxes are preprocessed before OCR. The pipeline (implemented in `paddle_ocr.py`) converts each crop to greyscale, applies symmetric padding to produce a square image, and resizes to a consistent target dimension. This normalisation ensures reliable OCR performance across varying crop sizes. AgentC additionally restricts the OCR character set to the digits 0–9 and characters X/x (the only valid characters in Kemler codes and UN numbers), reducing character confusion errors.

5.5.3.4 Training, Evaluation and Inference of AI Models

AgentB uses a publicly available pre-trained YOLOv11 model for license plate detection (<https://github.com/bhaskrr/number-plate-recognition-using-yolov11/tree/main>), adapted for use within the pipeline without additional training. AgentC uses a custom YOLOv11 model trained on a proprietary dataset created by the project team, composed of orange square ADR hazard plates collected by the project team and covering a variety of viewing angles, distances, and lighting conditions. During inference, bounding boxes are filtered with a minimum confidence threshold of 0.4 for both agents (configurable via Docker Compose environment variables).

Table 31 Model Performance Metrics

Metric	Value	Description
Precision	95.50%	True positives / All predicted positives
Recall	91.35%	True positives / All actual positives
mAP@50	95.47%	Mean AP at IoU threshold 0.50
mAP@50-95	56.81%	Mean AP across IoU 0.50–0.95

The model developed by the team achieves a mAP@50 of 95.47% and precision of 95.50%, confirming reliable detection with minimal false positives. The recall of 91.35% shows the model captures the vast majority of hazard plates. The mAP@50-95 of 56.81% reflects acceptable performance under stricter localization thresholds. The normalized confusion matrix (Figure 99) provides detailed insight into classification performance per class, showing the proportion of correct predictions versus misclassifications. Figure 100 shows the evolution of training and validation losses alongside performance metrics throughout the training process. The training converged stably, with box loss decreasing from 0.97 to 0.74 and classification loss from 0.53 to 0.40, showing no significant overfitting.

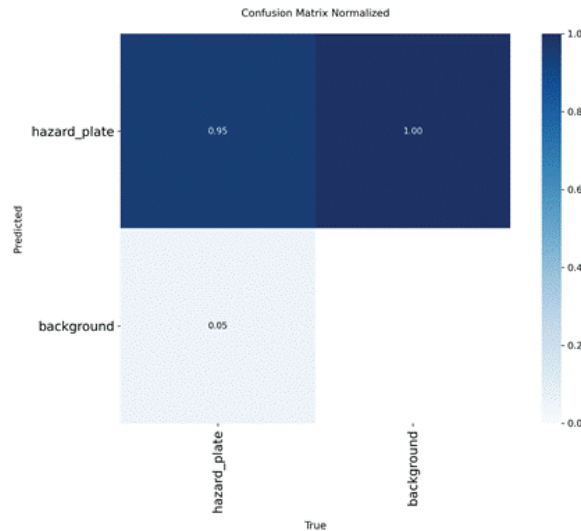


Figure 99 Normalized confusion matrix

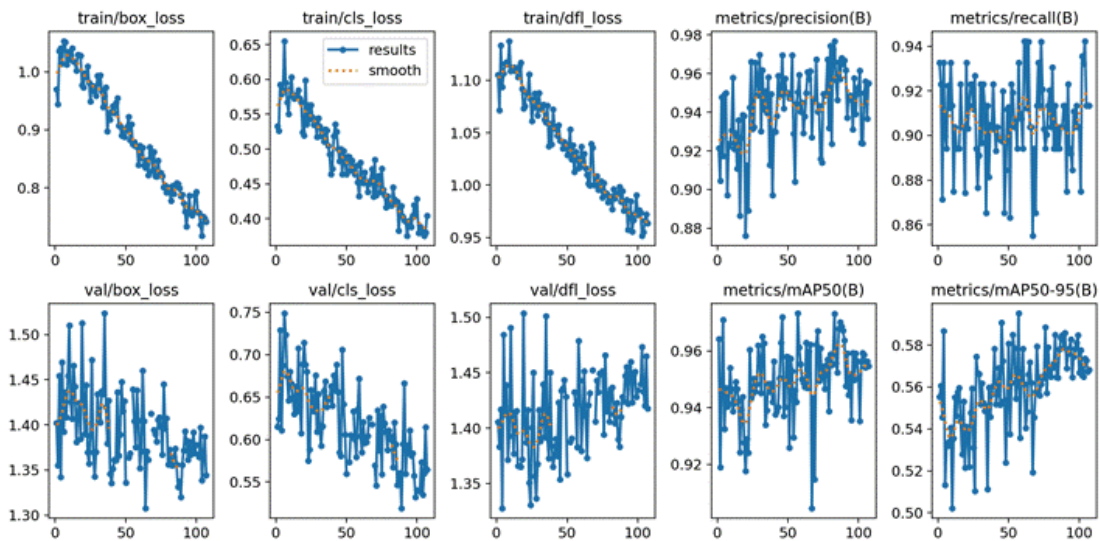


Figure 100 Training curves: loss functions (box, classification, DFL) and performance metrics (precision, recall, mAP) over 107 epochs

5.5.3.5 Connection to MLOps

AgentB and AgentC do not currently integrate with a MLOps platform. Models are loaded from static files at container startup. Future work may include integration with an MLOps framework to enable model versioning, automated retraining pipelines triggered by production data feedback, and systematic performance monitoring.

5.5.3.6 Development-Deployment and Results

AgentB and AgentC are implemented in Python using the following technologies:

- Ultralytics YOLOv11 (<https://docs.ultralytics.com/models/yolo11/>) — object detection models for licence plates (AgentB) and ADR hazard plates (AgentC)
- PaddleOCR (<https://github.com/PaddlePaddle/PaddleOCR>) — OCR text extraction from detected image crops

- OpenCV (<https://opencv.org/>) — video frame capture, image preprocessing, and bounding box annotation
- confluent-kafka (<https://github.com/confluentinc/confluent-kafka-python>) — Kafka consumer (truck-detected) and producer (lp-results / hz-results) via shared wrappers
- MinIO Python SDK (<https://www.min.io/>) — zero-copy in-memory upload of image crops to S3-compatible object storage with automatic lifecycle rules (7-day TTL for crops)
- Pydantic BaseSettings (<https://docs.pydantic.dev/>) — environment-variable-based configuration
- prometheus_client (https://github.com/prometheus/client_python) — Prometheus metrics (inference latency, frames processed, detections, OCR confidence)

The shared BaseAgent class (src/AI_APP/shared/src/base_agent.py) implements the full detection-OCR-consensus-publish pipeline. AgentB and AgentC implement only agent-specific abstract methods (model path, MinIO bucket, Kafka topic, message construction) while inheriting the pipeline from the base class. The PlateClassifier component (used in AgentB) applies HSV colour analysis and aspect-ratio checks to distinguish licence plates from ADR hazard plates and reject false detections. A StreamManager component runs in a background thread to provide always-fresh frames with automatic RTSP reconnection (up to 10 retries). Annotated frames are uploaded to MinIO asynchronously to avoid blocking the detection pipeline. Source code is available in the project GitLab repository (<https://gitlab.6g-versus.eu/wp4/ai-apps/portuguese-cluster/itav-aiapp>) under src/AI_APP/agentB/ and src/AI_APP/agentC/.

AgentB and AgentC are each deployed as two containerized instances via Docker Compose, namely **agentb-1/agentc-1** and **agentb-2/agentc-2**, each configured with the corresponding **GATE_ID** environment variable. The two instances reflect the two distinct use cases supported by the system: the first instance serves the vehicle access control use case, processing video from the entry gate to support gate opening decisions, and the second instance serves the infraction detection use case, processing video from a camera placed at a location where infractions may occur. Each instance operates its own independent agent pipeline, ensuring that both use cases are processed simultaneously and in isolation. All four instances share a dedicated NVIDIA GPU for faster YOLOv11 inference. Container images are built from multi-stage Dockerfiles and deployed on a dedicated GPU-enabled virtual machine in the testbed infrastructure alongside the AgentA instances, the Kafka broker and the AI Gateway. Container startup is orchestrated via Docker Compose health checks, ensuring that both Kafka broker availability and topic creation are confirmed before agent containers start. For Trial 2, AgentB and AgentC were deployed as part of the complete AI-App stack on the ITAV testbed, processing live 4K video feeds from cameras placed at the time of the trial simulating the real ones and successfully delivering license plate and hazard plate recognition results to the V-App in real time, enabling automated access control decisions for hazardous goods trucks

The HazMat Vehicles Detector AI-App has been fully developed, integrated and validated in conjunction with the associated V-App and the Simple Object Detection System AI-App to ensure full functional interoperability. The first public demonstration took place on 25/3/2026 at the Port of Aveiro. Figure 101 shows the high-resolution identification of the vehicle license plate by AgentB, utilized for the access control and infraction detection use cases.



Figure 101 High-resolution identification of license plate

Figure 102 shows the detection of ADR orange plates by AgentC for the extraction of Kemler codes and UN numbers to identify dangerous goods.



Figure 102 Detection of ADR orange plates

5.5.4 AI-Apps under Development for Phase 2

5.5.4.1 IoT Device Energy Management AI-App

The role of this AI-App is to recommend a QoS energy profile to be used for an IoT device to achieve the best possible continuous data collection and transmission based on the current and predicted state of the battery (accounting for the availability of energy harvesting capabilities). To achieve this, the AI app leverages the AI/MLOps framework to operate two engines, one for forecasting and one for recommendations. The forecasting engine ingests energy state data from IoT devices and other data (e.g. standard energy harvesting rate, meteorological data from external APIs) to predict the future energy state of each device. The recommendation engine then processes these forecasts and based on the available QoS profiles, selects the most appropriate profile for a given period. Currently, an API for orchestrating the interactions between the AI-app, V-app, and MLOps Framework is being developed, alongside the Airflow DAGs required to manage

these workflows. Once complete, the focus will shift to the development of the model and recommendation engine, as well as any necessary support features, such as data processing.

5.5.4.2 Infrastructure Energy Management AI-App

The AI-App for Scenario 3 will act as an intermediary layer between the AI/ML framework and the N-App and V-App components. Its main role is to orchestrate communication, receiving processed outputs from the AI/ML framework and forwarding actionable insights to the relevant X-Apps, which are responsible for interacting with the underlying hardware infrastructure. The AI/ML framework ingests data from solar panels and EV chargers, including energy production, consumption patterns, and operational status. Based on this data, the framework generates predictions and optimization strategies aimed at maximizing the use of clean energy, particularly by aligning EV charging demand with renewable energy availability. These results are then transmitted to the AI-App, which distributes them accordingly to the V-App. The V-App subsequently translates these high-level optimization decisions into control actions on the EV chargers, enabling dynamic and efficient energy management. In this context, the AI-App plays a central role as the intelligence enabler and coordination point of the overall system. Beyond simple message relaying, it ensures that the outputs generated by the AI/ML framework are properly interpreted, contextualized, and translated into actionable commands aligned with operational constraints and system priorities. It also handles interoperability between heterogeneous components, guaranteeing seamless integration across the 6G architecture.

5.6 Spanish Cluster

The Spanish Cluster contributes several AI-based applications across both trial phases. During Phase 1, selected AI-Apps are introduced as mock-ups or initial trial components, while Phase 2 expands their role through full implementation and deployment in additional trial activities. As shown in Table 32, the Spanish Cluster includes applications focused on water quality decision support, reliable water treatment, immersive AGV driving with autonomous missions, and automatic surveillance, demonstrating a progressive approach to the development and validation of AI solutions across the project. It should be noted that no corresponding AI-App has been defined for the Multi-connectivity Protocol N-App or the Unmanned Vehicle Control V-App within the Spanish Cluster.

Table 32 Implemented and under development AI-Apps in the Spanish cluster

Cluster Name	App Name	Phase (1or 2)
Spanish cluster	DSS for water quality AI-App	Phase 1 & 2
Spanish cluster	Immersive driving of an AGV with autonomous missions AI-App	Phase 1 & 2
Spanish cluster	Automatic surveillance AI-App	Phase 2
Spanish cluster	DSS for reliable water treatment AI-App	Phase 1 & 2

5.6.1 DSS for water quality AI-App

5.6.1.1 Overview

DSS for water quality AI-App is the core of the DSS for water quality. Its main function is to process near real-time data from water-quality sensors in order to support the optimisation of water mixing, combining reclaimed water with other sources, such as groundwater, so that the blended water remains suitable for crop irrigation.

5.6.1.2 Architecture

The architecture of DSS for water quality AI-App, as shown in Figure 103 is centred on its role as the core intelligence component of the triplet. The AI-App receives water-quality measurements through the sensing pipeline, processes these data in the cloud, and applies decision-support logic to assess whether the available water is suitable for irrigation and how reclaimed water should be mixed with other sources. The architecture is aligned with the AI-App role in the triplet, as it separates data ingestion, analytical processing, alert generation, and interaction with the V-App, while ensuring confidentiality in the management of sensitive operational data.

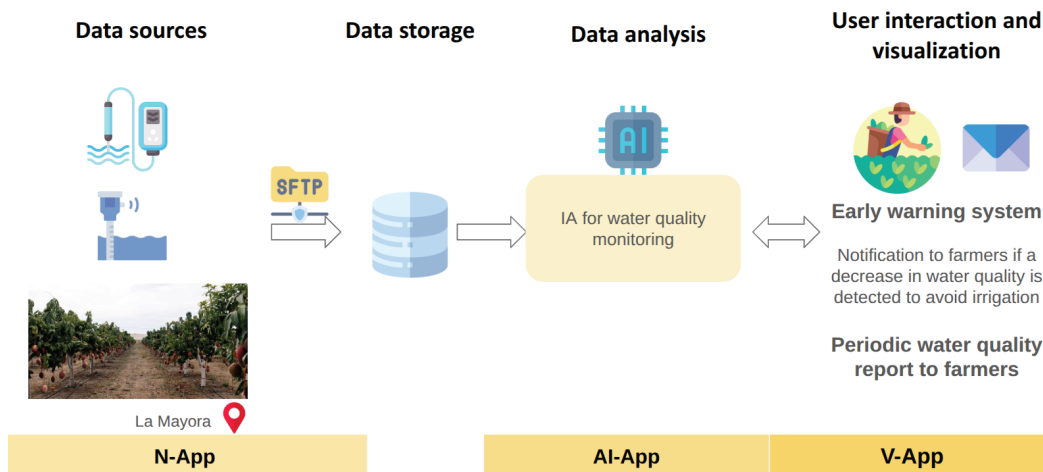


Figure 103 Architecture of the DSS for water quality

5.6.1.3 Training, Evaluation and Inference of AI Models

At the current stage, the training and evaluation of the AI models have not yet started, because they depend on the availability of real sensor data from the upcoming field deployment. Once the sensors are installed by CSIC and sufficient data have been collected, the development of the water mixing model will begin. This model will be designed to support decisions on the combination of reclaimed water with other water sources so that the resulting blended water remains suitable for crop irrigation. The evaluation phase will define appropriate performance metrics according to the specific modelling approach finally adopted, while the inference phase will use near real-time sensor inputs to assess irrigation suitability and generate the corresponding notification triggers and contextual outputs for the V-App.

5.6.1.4 Connection to MLOps

For privacy reasons due to the treatment of confidentiality-sensitive data, the application is not planned to be integrated within the MLOps solution.

5.6.1.5 Development-Deployment and Results

The development of DSS for water quality AI-App is focused on implementing the core cloud-based decision support logic for the water quality use case. The data ingestion pipeline has already been designed and implemented, and the water-quality sensors are planned to be installed in the coming weeks by CSIC. Once the sensors are installed and data collection begins, the development of the water mixing model will start using real measurements collected from the monitored environment. The AI-App is planned to be developed in a way that supports ingestion of sensor data, analytical processing, generation of irrigation suitability assessments, production of alert triggers for the V-App, and aggregation of historical water-quality information. Since the application has no strict latency constraints, it is intended for cloud deployment, while maintaining strict confidentiality controls for potentially sensitive information associated with wells, ponds, and irrigation points. The data preprocessing stage of DSS for water quality AI-App will start once the water-quality sensors are installed and begin producing measurements. At that stage, the collected sensor data will

be prepared for subsequent modelling and decision-support tasks. The preprocessing workflow is expected to include validation of incoming measurements, handling of missing or inconsistent values, synchronization of data streams from different sensing points, and preparation of structured input datasets for the development of the water mixing model. As the final characteristics of the collected data will depend on the sensor deployment and operating conditions, the exact preprocessing procedures will be refined once real field data become available.

A key aspect of the development of DSS for water quality AI-App is the treatment of confidentiality-sensitive data. Although the application is deployed in the cloud and does not have strict latency constraints, it must ensure appropriate protection of data that may be associated with specific owners, agricultural assets, or operational routines. Therefore, confidentiality and controlled handling of contextual information are considered fundamental design principles of the application.

DSS for water quality AI-App will be deployed as a containerized application in the cloud-based testbed infrastructure. The containerized deployment approach, using Docker, will support portability, modular integration, and maintainability of the application within the overall triplet. This cloud deployment model is suitable for the application because the use case does not impose strict latency constraints, while still requiring dependable processing and secure handling of sensitive data. At the current stage, the main result achieved for DSS for water quality AI-App is the design and implementation of the data ingestion pipeline. The field water-quality sensors are planned to be installed in the coming weeks by CSIC, and once they are operational the collection of real measurements will begin. The development of the water mixing model, as well as its training, evaluation, and inference validation, will start after sufficient sensor data have been gathered.

5.6.2 Immersive driving of an AGV with autonomous missions AI-App

5.6.2.1 Overview

This AI-App provides real-time video analytics capabilities to support the immersive remote driving use case. It processes video streams received from an external PTZ camera and performs object and pattern recognition using computer vision techniques. The application extracts frames from incoming video streams and applies AI-based detection models to identify relevant patterns. The results are stored and made available for monitoring and analysis, enabling enhanced situational awareness and data-driven decision-making.

5.6.2.2 Architecture

The AI-App is designed as a two-service architecture deployed on a dedicated virtual machine. The first service is responsible for video ingestion and preprocessing, while the second service handles AI-based inference and result storage. The image ingestion system is implemented through a service running on the AI machine, which accesses the camera's API via IP address. A GET request is made using curl to an endpoint on the camera's server, causing the camera to take a picture and return it in the response. The images obtained periodically are stored in a local directory for later processing. The second service is a background daemon that continuously monitors the image directory. When new frames are detected, the service applies object detection algorithms using an OpenCV-based pipeline integrated with a YOLO model. Detection results are stored in an external InfluxDB database, while annotated images are saved locally. This modular architecture enables efficient processing and clear separation between data ingestion and AI inference components.

5.6.2.3 Data Preprocessing

The preprocessing process includes periodic image extraction and file management to ensure that only relevant and recent data is processed. After each processing cycle, the input directory is cleared to avoid redundant processing and to maintain system efficiency.

5.6.2.4 Training, Evaluation and Inference of AI Models

The AI-App uses a YOLO-based model for object and pattern detection. The model is integrated into the inference pipeline using OpenCV and is applied to each extracted frame. Inference is performed periodically as new frames are detected. When a target pattern is identified, the system records the detection results and

generates annotated images with bounding boxes. These outputs can be used for monitoring, analysis, and validation of system performance.

5.6.2.5 Connection to MLOps

In this implementation, a full MLOps pipeline has not been incorporated. This decision is justified by the scope and maturity level of the current prototype, which focuses on validating the feasibility of real-time image analytics within the immersive remote driving use case. The AI-App relies on a pre-trained YOLO model integrated into a lightweight inference pipeline, without the need for continuous model retraining, versioning, or automated deployment workflows. The operational requirements of the use case are currently satisfied through static model deployment and periodic manual updates when needed. Additionally, the system operates in a controlled testbed environment, where data variability and model drift are limited, reducing the necessity for continuous training and automated lifecycle management. However, the architecture has been designed with modularity in mind, allowing future integration with MLOps frameworks if required. This would enable capabilities such as automated retraining, model versioning, performance monitoring, and continuous deployment in more advanced or production-level scenarios.

5.6.2.6 Development, Deployment and Result

The AI-App has been developed as a combination image capture and AI-based inference services. The image ingestion component is implemented using a dedicated service from the camera, which handles periodically photo reception and storage. The inference component is implemented in Python using OpenCV and a YOLO-based object detection model.

The system follows a modular approach, separating image acquisition and AI processing into independent services. A background daemon is used to enable continuous monitoring and processing of incoming data. The application is designed to operate efficiently within a virtualized environment and can be containerized for deployment.

The AI-App is deployed on a dedicated virtual machine within the testbed infrastructure. The solution consists of two services: the image extraction service and the AI inference daemon. The InfluxDB database is deployed on a separate virtual machine and is used to store detection results. The entire system can be containerized to facilitate deployment and scalability within the 5G environment. The AI-App successfully processes incoming images and performs real-time object detection as shown in Figure 104. The system is capable of identifying target patterns and storing the results in a database, as well as generating annotated images. The solution demonstrates stable operation and efficient processing of video data. The integration with InfluxDB enables monitoring and analysis of detection events.



Figure 104 Person detection with YOLO

5.6.3 DSS for reliable water treatment AI-App

5.6.3.1 Overview

DSS for reliable water treatment AI-App is the core Decision Support System (DSS) of the reliable water treatment use case. Its main function is to analyse multi-sensor data streams from the Roquetas de Mar wastewater treatment plant, combining water-quality sensor measurements and camera feeds to detect anomalies, assess process performance, and support the assurance of treated water quality. The application is deployed in the cloud, as it does not impose strict latency constraints, and image data are handled in dedicated, access-restricted environments to support GDPR compliance, since plant workers may appear in captured images. In addition, the use case has already evaluated the ColiMinder (CMI-02) online microbiological analyser as a potential early-warning source for E. coli detection in reclaimed water.

5.6.3.2 Architecture

The architecture of the application is centred on its role as the intelligence core of the triplet (see Figure 105). The application receives heterogeneous input data from the plant monitoring environment, combining water-quality sensor streams and image data from camera feeds. These inputs are processed in the cloud by analytical and AI-based modules that detect anomalies, assess process performance, and generate outputs for the other triplet components. When abnormal conditions are identified, the AI-App sends notification triggers and related alert context to the V-App so that plant operators can be informed.

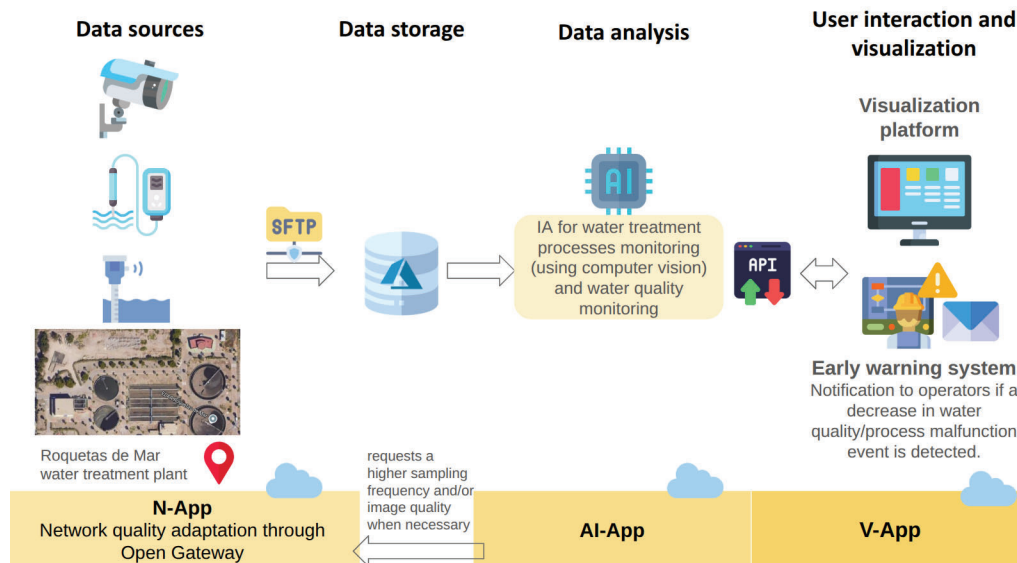


Figure 105 Architecture of the DSS for reliable water treatment

The architecture is aligned with the AI-App role in the triplet, separating data ingestion, analytics, alert generation, and network-oriented adaptation support, while also incorporating access-restricted handling of image data for GDPR compliance. A previous evaluation also considered the possible use of the ColiMinder analyser for online microbiological monitoring; however, given the lack of robust correlation with the reference method, this signal is not currently treated as a validated decision variable, and the future integration of any online *E. coli* sensor remains open.

5.6.3.3 Data Preprocessing

The data preprocessing workflow of the application will support heterogeneous data streams coming from water-quality sensors and camera systems. For sensor data, preprocessing will include validation, cleaning, synchronization, and formatting of time-series measurements. For image data, preprocessing will include image selection, formatting, annotation, and preparation of datasets suitable for computer vision model development. Since the cameras are still to be installed, the final preprocessing pipeline for image data will be refined once real plant images become available. If online microbiological sensing is incorporated in future phases, its preprocessing workflow will also need to account for validation against reference measurements and interpretation according to its proven operational meaning. Based on the completed evaluation, ColiMinder cannot currently be used as a direct proxy for *E. coli* concentration in the reclaimed water tank.

5.6.3.4 Training, Evaluation and Inference of AI Models

At the current stage, the training and evaluation of the AI models have not yet started, as they depend on the installation of the cameras and the subsequent creation of the image dataset. Once the cameras are deployed and sufficient data have been collected, the dataset will be built and used to train the AI models for anomaly detection and process performance assessment. The final model development strategy may include both sensor-based and vision-based analytical components, depending on the characteristics of the collected data and the operational needs of the wastewater treatment plant. The evaluation phase will define suitable performance metrics according to the selected modelling approaches, while the inference phase will process incoming plant data to detect abnormal conditions. The completed ColiMinder study provides an important boundary condition for future model development: its signal cannot currently be treated as a validated indicator of *E. coli* contamination in the reclaimed water tank. Moreover, it has not yet been decided whether a different online *E. coli* sensor will be integrated into the AI-App in future phases.

5.6.3.5 Development, Deployment and Results

The development of the AI-App focuses on implementing in Python the core analytical and decision-support components of the reliable water treatment DSS. So far, the data ingestion pipeline has been designed and implemented, providing the basis for integrating heterogeneous plant data sources. The cameras required for the computer vision solutions will be installed in the coming weeks, after which the image dataset will be built for training the AI models. The application is being designed to support ingestion of water-quality sensor streams and camera feeds, anomaly detection, process performance assessment, generation of notification triggers and alert context for the V-App, and generation of connectivity configuration intents for the N-App.

As part of the already completed work, ColiMinder (CMI-02) was evaluated as a possible online early-warning system for *E. coli* detection in reclaimed water at the Roquetas de Mar plant (see Figure 107), with the aim of supporting compliance with RD 1085/2024 for A.A quality water. The study compared ColiMinder measurements against the reference method IDEXX Colilert-18 and included equipment installation, commissioning, laboratory testing, plant trials, and two prevalidation campaigns, together with monitoring of residual chlorine, UV transmittance, UV dose, turbidity, pH, and conductivity. The results showed an estimated limit of detection of 3.5 mMFU/100 ml and a limit of quantification of 30 mMFU/100 ml, both worse than the manufacturer's reported performance. Although the first prevalidation campaign showed that reducing chlorine dosage increased *E. coli* concentrations measured by the reference method, the correlation between ColiMinder and Colilert-18 remained weak ($R^2 = 0.153$, see Fig 106). In the second prevalidation campaign, the reference method detected 0 NMP/100 ml, while ColiMinder still reported activity in the range of approximately 169 to 234 mMFU/100 ml. The study therefore concluded that the device is not recommended as an early-warning system for *E. coli* in the reclaimed water tank under current operating conditions. Accordingly, ColiMinder is not currently considered a validated operational input for the AI-App. At this stage, it is also not yet defined whether a different online *E. coli* sensor will be incorporated as an input in later phases of the application.



Figure 106 A. Components of the equipment installed in the laboratory. B. Equipment installed in the water treatment plant.

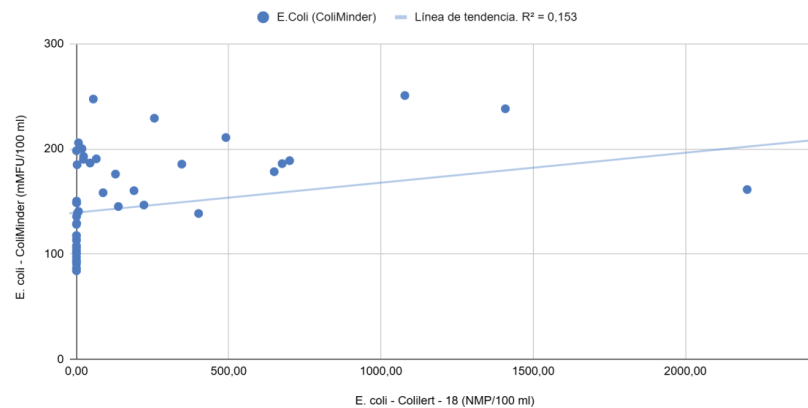


Figure 107 Correlation between E. coli concentration in the IDEXX Colilert method and enzymatic activity recorded on the ColiMinder instrument.

An important aspect of the development of DSS for reliable water treatment AI-App is the controlled handling of camera data. As plant workers may appear in captured images, image-based processing must be carried out in dedicated, access-restricted environments in order to support GDPR compliance. Another relevant consideration is the role of online microbiological sensing. The completed evaluation of ColiMinder showed that, although the device can provide continuous enzymatic activity measurements, it did not demonstrate a sufficiently robust relationship with the reference E. coli method under real plant conditions. Consequently, it is not currently considered a validated input for the AI-App, and the possible inclusion of a different online E. coli sensor remains to be defined.

At the current stage, the main result achieved for DSS for reliable water treatment AI-App is the design and implementation of the data ingestion pipeline. In parallel, a dedicated study has already assessed the feasibility of using ColiMinder (CMI-02) as an online early-warning system for E. coli in reclaimed water at the Roquetas de Mar plant. The study included laboratory tests, plant trials, and two prevalidation campaigns against the IDEXX Colilert-18 reference method. Its main conclusion was that no robust operational correlation could be established under current plant conditions, and therefore the device is not recommended at this stage as a validated early-warning system for E. coli in the reclaimed water tank. At present, no decision has yet been made on whether another online E. coli sensor will be incorporated into the AI-App.

5.6.4 AI-Apps under Development for Phase 2

In Phase 2, DSS for water quality AI-App will focus on starting the model development phase once the field sensors are installed and sufficient data have been collected. This includes preparing the datasets, defining the water-mixing model, establishing the corresponding evaluation methodology, and integrating the resulting decision-support outputs with the V-App for notifications and information delivery.

In Phase 2, the Immersive Remote Driving AI-App will focus on detecting and counting the number of people present in front of the camera. This information will then be communicated to the N-App, enabling it to adjust the network slice accordingly. The goal is to create a responsive system where network resources are adapted in real time based on the observed environment. Regarding DSS for reliable water treatment AI-App, in Phase 2 it will focus on completing the sensing setup, in particular the installation of the cameras for the computer vision solutions and building the image dataset required for model training. Further work will address the development and evaluation of the anomaly-detection models, the consolidation of the AI outputs for alert generation, and the integration of these outputs with the V-App and N-App in the target deployment environment. In Phase 2, the Automatic Surveillance & Detection in Farming Operation AI-App will focus on advancing the multimodal analysis pipeline that combines radio sensing information from the 5G system with LiDAR data from the robot in order to detect relevant events in farming environments. The main objective will be to convert variations observed in the radio channel and LiDAR measurements into reliable event detection

outputs, such as abnormal situations, unexpected presence in the supervised area, or other relevant surveillance-related events. These detection results will be returned to the associated V-App to support automated response mechanisms. At the current stage, the AI-App is in early development, with data reception from both 5G and LiDAR sources already operational and the ground truth system in place. Ongoing work in Phase 2 is focused on testing transformer-based models for the fusion and interpretation of these sensing inputs, including the evaluation of different training checkpoints and model versions (e.g., epoch 70, epoch 67, and best-performing weights). This work will provide the basis for the first integrated detection results to be demonstrated in the next phase of the farming surveillance use case.

6 Triplet Integration

The integration and validation framework of the project is structured to support the progressive convergence of V-App, AI-App, and N-App components across the six experimental clusters. It follows a Network-Aware Application paradigm, enabling a continuous and adaptive verification cycle that evolves from controlled laboratory validation to comprehensive end-to-end (E2E) service evaluation over 5G/6G infrastructures.

At the project requirements, a valid service deployment is characterized by the successful interaction between at least one V-App and one N-App through standardized interfaces. While the integration of AI-Apps is not mandatory, it is actively encouraged to enhance service intelligence by combining network telemetry with vertical-specific data, enabling advanced analytics and closed-loop automation. The integration lifecycle is organized into four complementary stages and applies four the two phase of trials, allowing flexible progression depending on the readiness of individual components and environments:

Stage 1: Initial Unit Testing and Containerization: This stage focuses on software readiness and portability. All V-/AI-/N-App components are containerized (e.g., using Docker) and validated for correct deployment, execution, and exposure of basic APIs.

Stage 2: Registry Integration and Connectivity Verification: This stage addresses service exposure and interoperability mechanisms. It includes integration with CAPIF where applicable, covering API registration, discovery, and authorization procedures, as well as the validation of initial 5G control-plane interactions for N-Apps.

Stage 3: Functional E2E Integration: This stage validates service-level workflows under realistic conditions. It involves the orchestration of applications across edge and cloud environments, ensuring data-plane connectivity between V-Apps and 5G user equipment (UEs), including diverse endpoint devices such as sensors, UAVs, and robotic platforms. Both real and synthetic data sources may be used to exercise the system.

Stage 4: System Optimization and Pilot Readiness: The final stage focuses on performance tuning, stability, and service robustness. It consolidates previous integration efforts and prepares the system for large-scale validation activities, serving as a transition toward operational trial execution.

This staged approach provides a structured yet adaptable pathway for integration, allowing each experimental environment to progressively refine its service capabilities while contributing to the overall system validation objectives.

6.1 Bulgarian Cluster

The Bulgarian-cluster triplet integration combines the **V-App**, **N-App**, and **AI-App** components across two complementary environments: the **A1 operational pilot deployment environment in Bulgaria** and the **UCY 5G-OAI / OPAL-RT pre-piloting environment in Cyprus**. The A1 environment provides the target distributed deployment setup, with central-side components deployed in Sofia and regional-side components deployed in Varna, while the UCY environment provides a controlled laboratory setup for validating the PMU-based monitoring, control, and closed-loop actuation workflow before full pilot execution.

In the current alpha-stage integration, the **V-App layer** provides the main monitoring and control path. The V-App Central receives PMU measurements, supports monitoring through the WAM function, and performs central control through the WAC function. The V-App Regional Controller receives control requests from the WAC and forwards active and reactive power setpoints toward the DER/model interfaces. This path has already been validated in the UCY pre-piloting environment using the OPAL-RT power-system digital twin and the UCY 5G-OAI communication chain.

The **N-App for Network QoS Monitoring and Improvement** is part of the central-side triplet architecture. It interfaces with the underlying 5G Core Network and exposes network APIs through OpenCAPIF, acting as the bridge between the network layer and the V-App Central. The **AI-App** is being integrated progressively: a dummy version is included in the A1 deployment architecture, while the functional AI-App is under development in the UCY environment to provide short-term frequency trajectory prediction and support future WAC decision-making.

This combined A1 + UCY setup is used to de-risk the Bulgarian pilot, validate interoperability across the triplet components, and prepare broader end-to-end validation activities.

6.1.1.1 Sequence Diagram

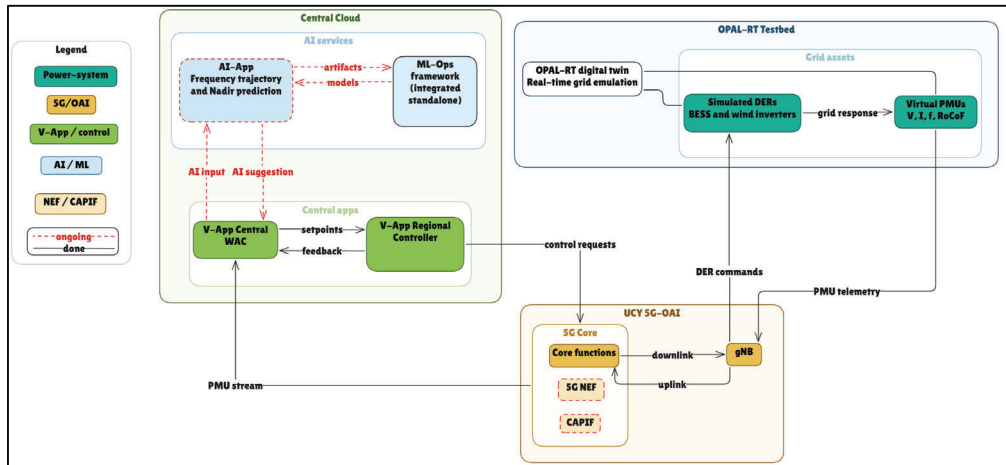


Figure 108 Bulgarian-cluster triplet integration architecture in the UCY pre-piloting environment, showing the OPAL-RT power-system testbed, UCY 5G-OAI transport and exposure layer, V-App Central WAC, V-App Regional RC, AI-App, ML-Ops services, and the N-App / netw

Although the figure mainly illustrates the UCY pre-piloting validation path, including the PMU, V-App, RC, AI-App and ML-Ops data flows, the **N-App for Network QoS Monitoring and Improvement** is also part of the Bulgarian triplet integration, but in the **A1 testbed environment**. The N-App is not part of the UCY laboratory validation path shown in the figure, but it is used in the A1 central-side deployment to interface with the underlying 5G Core Network and provide the network-facing application layer between the Core Network and the V-App Central. Network APIs are exposed through **OpenCAPIF**, which enables controlled access to NEF-related services.

In the A1 architecture, the N-App receives network performance information from the A1 performance and monitoring platform, provides network data to the V-App Central, and issues requests toward the Core Network. For Phase 1, the N-App functionality based on the **Monitoring Event API** has been tested and integrated with the V-App Central. The N-App is implemented as a Python application using FastAPI, Unicorn, and the OpenCAPIF SDK, with the N-App invoker and NEF service deployed as Docker containers through Helm charts in Kubernetes.

In Phase 2, the N-App will be extended to leverage A1 5GC Network Exposure capabilities and expose NEF network APIs, specifically the QoS Network API and the Event Exposure Network API, focusing on the following event types: Loss of Connectivity, UE Reachability, and PDN Connectivity Status.

6.1.1.2 Communication mechanisms

The Bulgarian-cluster triplet uses a combination of communication mechanisms depending on the layer of interaction.

1. PMU / synchrophasor data transport

At measurement level, synchronized PMU telemetry is transported from the power-system layer toward the application layer. In the UCY environment, this transport is provided through the 5G-OAI chain (USRP, gNB, and 5G Core). The PMU-related data include synchronized measurements such as voltage, current, frequency, and RoCoF.

2. V-App Central – V-App Regional control communication

The control interaction between the V-App Central – WAC and the V-App Regional Controller – RC is implemented through REST APIs over HTTP, with JSON payloads carrying the control request and status / capability information. This is a key implementation aspect because the WAC and RC are designed to operate in different data centres rather than on the same machine.

3. Regional controller – DER / model interface communication

At the regional side, the RC exchanges telemetry and control references with DER interfaces or OPAL-RT model interfaces. In the current engineering and validation setup, this may include field-oriented or model-based interfaces, including Modbus/TCP where applicable.

4. Central-side interaction with N-App and AI-App

At the central side, the V-App layer provides monitoring and application data that can be consumed by the N-App and the AI-App. In the present status, the N-App is part of the A1 central deployment, while the AI-App is under active functional integration in the UCY environment.

5. User access and UI communication

Access to the web interfaces of the deployed applications is performed through the internal pilot network, with VPN-based access control in the A1 infrastructure.

6.1.1.2.1 API-based communication

The main API-based communication currently implemented at application level is the **WAC–RC communication path**.

<u>REST Method</u>	<u>Resource URI</u>	<u>Body Parameter</u>	<u>Description</u>
<u>GET</u>	<u>/rc/<rc_id>/capability</u>	<u>None</u>	Returns the available flexibility / capability of the addressed regional controller.
<u>GET</u>	<u>/rc/<rc_id>/status</u>	<u>None</u>	Returns the current status of the addressed RC, including aggregated telemetry and asset-level information where relevant.
<u>POST</u>	<u>/rc/<rc_id>/dispatch</u>	<u>delta p,</u> <u>optionally delta q</u>	Sends the active-power and optionally reactive-power dispatch request from the WAC to the RC.
<u>GET</u>	<u>/rc</u>	<u>None</u>	Returns the list of available regional controllers or their summary information.

In the current software implementation, these APIs are used by the WAC to:

- retrieve **regional capability / flexibility** information,
- retrieve **regional status** and response information,
- submit **dispatch commands** representing the corrective response required from the regional layer.

The payloads exchanged through these APIs are **JSON-based** and support the distributed operation of the central and regional services across physically separated environments.

6.1.1.2.2 Other communication methods

Synchrophasor / PMU telemetry

The PMU-related communication path carries synchronized electrical measurements from the power-system layer to the application layer. The data flow includes quantities such as:

- timestamp,
- voltage phasors,
- current phasors,

- frequency,
- RoCoF.

In the UCY setup, these streams originate from the OPAL-RT-emulated power system and are transported through the 5G-OAI infrastructure toward the computing nodes hosting the applications.

DER-side telemetry and control exchange

The regional side exchanges telemetry and control references between the RC and the DER or DER-model interfaces. The exchanged information includes:

- active power setpoints,
- reactive power setpoints,
- measured active/reactive power,
- state-of-charge and resource availability where applicable.

Depending on the test configuration, this interaction can be implemented through **model-based interfaces** or **Modbus/TCP-based communication**.

Deployment and access environment

Although not an application-layer protocol, the deployment environment itself is important for the integration status. All applications are containerized and deployed through Docker, Helm, and Kubernetes, while access to A1-hosted web interfaces is controlled through VPN-based connectivity.

The current results show that the Bulgarian cluster has already established the main deployment and integration foundations required for the triplet architecture.

In the A1 environment, the central-side and edge-side applications are already distributed across two data centres, with:

- Sofia hosting the central-side deployment,
- Varna hosting the regional-side deployment,
- all services deployed as containerized applications through Helm and Kubernetes.

In the UCY pre-piloting environment, the following results have already been achieved:

- real-time emulation of the Bulgarian use-case grid in OPAL-RT,
- transfer of PMU telemetry over the UCY 5G-OAI infrastructure,
- ingestion and storage of streaming PMU measurements by the V-App Central (WAM & WAC),
- reception of control requests by the V-App Regional Controller,
- forwarding of active and reactive power setpoints toward the simulated DER units,
- validation of the closed-loop actuation path back to the simulated DERs.

These results demonstrate that the distributed deployment concept of the Bulgarian triplet is already in place and that the V-App control and monitoring path has been functionally validated in the UCY environment. The AI-App / ML-Ops functional integration remains ongoing and is the next major step toward broader end-to-end validation.

Supporting evidence for these results includes:

- the deployment status of the A1 Sofia and Varna environments,
- the UCY pre-piloting integration architecture,
- the previously provided V-App UI snapshots,
- and the technical repositories and deployment artifacts maintained in the project GitLab.

6.1.1.3 Results

The current results show that the Bulgarian cluster has already established the main integration foundations required for the triplet architecture. The achieved results can be separated into deployment results in the A1 environment, functional validation results in the UCY pre-piloting environment, N-App integration results, and ongoing AI-App / ML-Ops integration results.

In the A1 environment, the central-side and edge-side deployment model has been established across two data centres. The Sofia data centre hosts the central-side components, including the V-App Central – WAM, the N-

App, and the AI-App dummy version, while the Varna data centre hosts the regional-side V-App Regional – monitoring DERs. The applications are deployed as containerized services using Docker, Helm, and Kubernetes, confirming the initial distributed deployment baseline of the Bulgarian triplet.

In the UCY pre-piloting environment, the power-system and control-chain interaction has been functionally validated. The OPAL-RT environment emulates the Bulgarian use-case grid and generates PMU-like synchrophasor measurements, which are transported through the UCY 5G-OAI infrastructure toward the application layer. These streams are received by the V-App Central layer and used for monitoring, event detection, WAC processing, and closed-loop validation.

The V-App control path has also been validated in the UCY setup. The WAC receives and processes the PMU measurements, evaluates the grid condition, and sends control requests to the V-App Regional Controller. The RC receives the dispatch request, evaluates the available regional capability, and forwards active and reactive power setpoints toward the simulated DER units in OPAL-RT. This confirms the functional closed-loop path from PMU measurements to WAC decision-making, RC dispatch, and DER/model response. The V-App file confirms that the WAC is implemented as a Python control service with PMU ingestion, WAC processing, control actions, dashboard supervision, and WAC–RC APIs, while the RC acts as the regional execution layer communicating upward with WAC and downward with DER/model interfaces.

For the N-App, the current result is that the N-App for Network QoS Monitoring and Improvement has been developed as a Python application using FastAPI and Uvicorn, with integration support through OpenCAPIF SDK. The N-App invoker and NEF service are deployed as Docker containers through Helm charts in Kubernetes. The Phase 1 functionality based on the Monitoring Event API has been tested and integrated with the V-App Central, while the QoS Network API and the Event Exposure Network API integration in A1 5GC network is planned for Phase 2.

For the AI-App, the functional integration is still ongoing. The current result is that the AI-App development path has been established: synthetic datasets are generated from MATLAB/Simulink simulations, the data are connected to the ML-Ops workflow using Airflow and MLflow, and preliminary model-development activities have started. The AI-App is intended to consume streaming PMU-derived data and provide short-term frequency trajectory and nadir predictions to support the WAC decision-making process in Trial 2.

Overall, the current results demonstrate that the Bulgarian triplet has reached the following integration status: the distributed deployment baseline is established in A1; the PMU-to-V-App-to-RC-to-DER closed-loop path is validated in UCY; the N-App Monitoring Event functionality is tested and integrated with V-App Central; and the AI-App / ML-Ops path is under active integration toward the next trial.

6.2 Finnish Cluster

The Finnish cluster integrates the V/AI/N-Apps and other applications within the UOULU 5G test network testbed. All the applications are run on a dedicated edge server in a Docker containerized environment using micro-service architecture. Communications between the applications take place over a shared Docker network within the server (see Figure 109). depicts communication between all the relevant system applications.

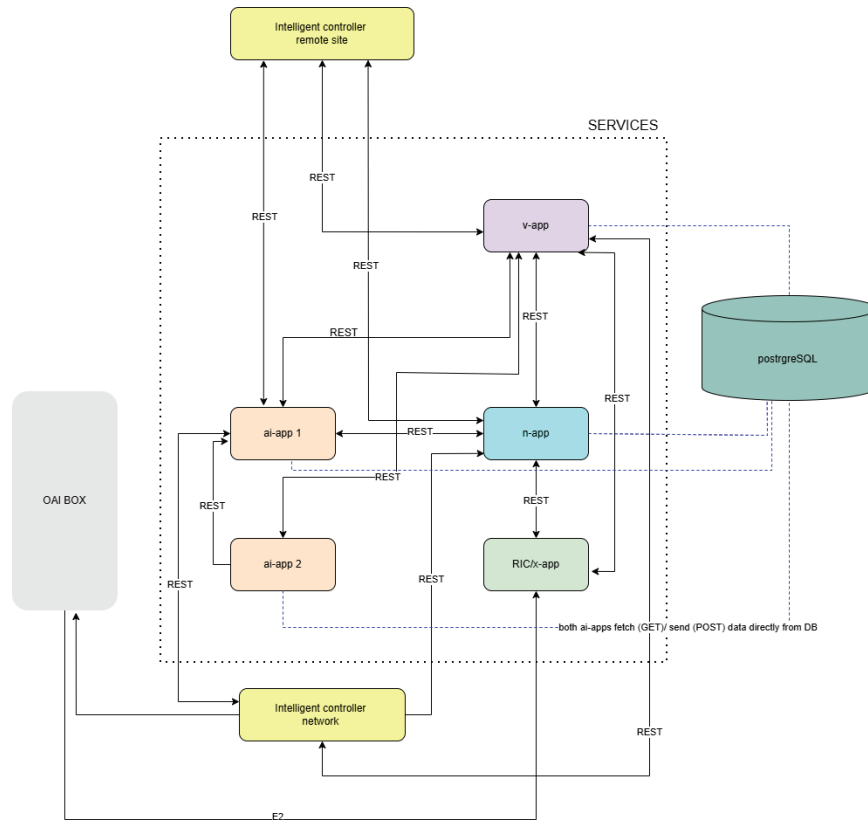


Figure 109. Inter-communication of main applications

V-App is responsible for providing web-based user interface and dashboard for controlling our energy self-sustainable 5G base station solution. It controls the start/stop sequence of other components and acts a system error reporter to the user in case of faults within other triplet components. V-App also shows various data to the user, e.g. weather forecast information, energy produced (solar panels, hydrogen fuel cell) and consumed by the whole system. V-App runs timers to command N-App to fetch weather forecast data periodically and to command X-App, remote intelligent controller, and network remote controller to send (energy) measurement data to N-App for storing them to a database.

There are two AI-Apps in the Finnish cluster's solution. AI-App1 with obtained data (remote side, weather forecast data, predicted network traffic from AI-App2) makes two decisions: 1) changes in remote side for example to turn on/off hydrogen fuel cell. 2) choose a configuration for BENETEL and database. AI-App1 communicates with N-App and sends decisions to it for storing the decision in database. AI-App2 in other hand is responsible for using data from X-App, type of the network (business, domestic, etc) and calendar data (vacations, peaks) as well as time of the day to predict network traffic and send the prediction to AI-App1. N-App is responsible for receiving data from components (AI-App1, remote intelligent controller, network intelligent controller, X-App) and storing the data in the database. On a regular interval V-App requests N-App to fetch newest weather forecast information via an API provided by Finnish Meteorological Institute and storing it in the database.

Other components participating in the inter-component communications are remote intelligent controller, network intelligent controller, and X-App. Remote intelligent controller waits requests from V-App to aggregate various energy consumption and production data and send them to N-App. Remote intelligent controller also enforces the AI decision (energy budgeting) upon request from AI-App1. Network intelligent controller measures the power consumption of the component in a virtual environment and server and upon request from V-App reports the measurements to N-App, which stores the data in the database. Upon request from AI-App1, the network intelligent controller enforces the AI decision to reconfigure OAIBOX. X-App

Receives network statistics data from OAIBOX and gNodeB per UE and upon request from V-App sends the data to N-App for storing them in the database.

6.2.1.1 Sequence Diagram

V-App runs a periodical 1-hour internal timer when the system is up and running. Each hour V-App request X-App, network intelligent controller, and remote intelligent controller to aggregate network and energy consumption/production measurement data from the last hour and send it to N-App for storing the data in the database. Figure 110 shows the sequence flow of V-App requesting each application for 1-hour measurement data.

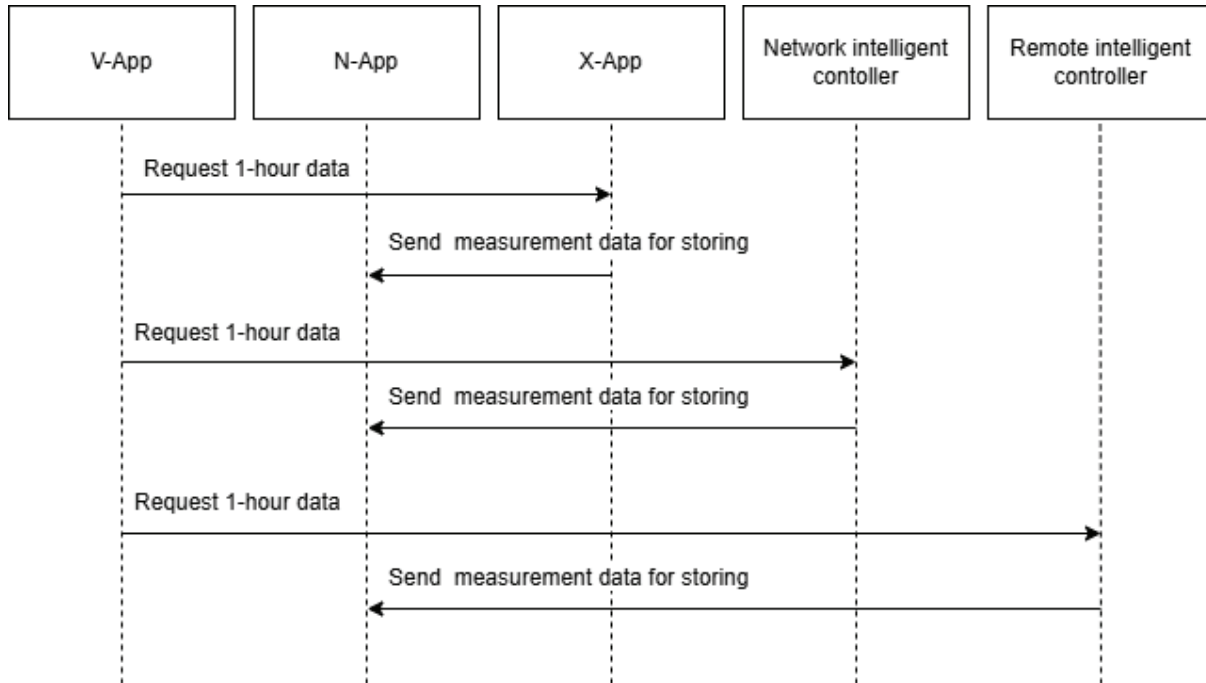


Figure 110. Handling of measurement data

After V-App has requested X-App, network intelligent controller, and remote intelligent controller to send the most recent aggregated measurement data to the N-App and the database, V-App informs AI-App1 and AI-App2 that new measurement data is ready in the database. AI-Apps then fetch the data and based on the data AI-App2 reaches a decision, which is sent to AI-App1. Then AI-App1 in turn creates a final decision and based on it adjusts network parameters in network intelligent controller X-App and energy production parameters in remote intelligent controller. After this AI-App1 sends the decision data to N-App for storing in the database. N-App stores the data and informs V-App about the availability of the new decision. V-App then updates the user interface and shows the decision information on the web page. Figure 111 illustrates the sequence of AI decision handling in our system.

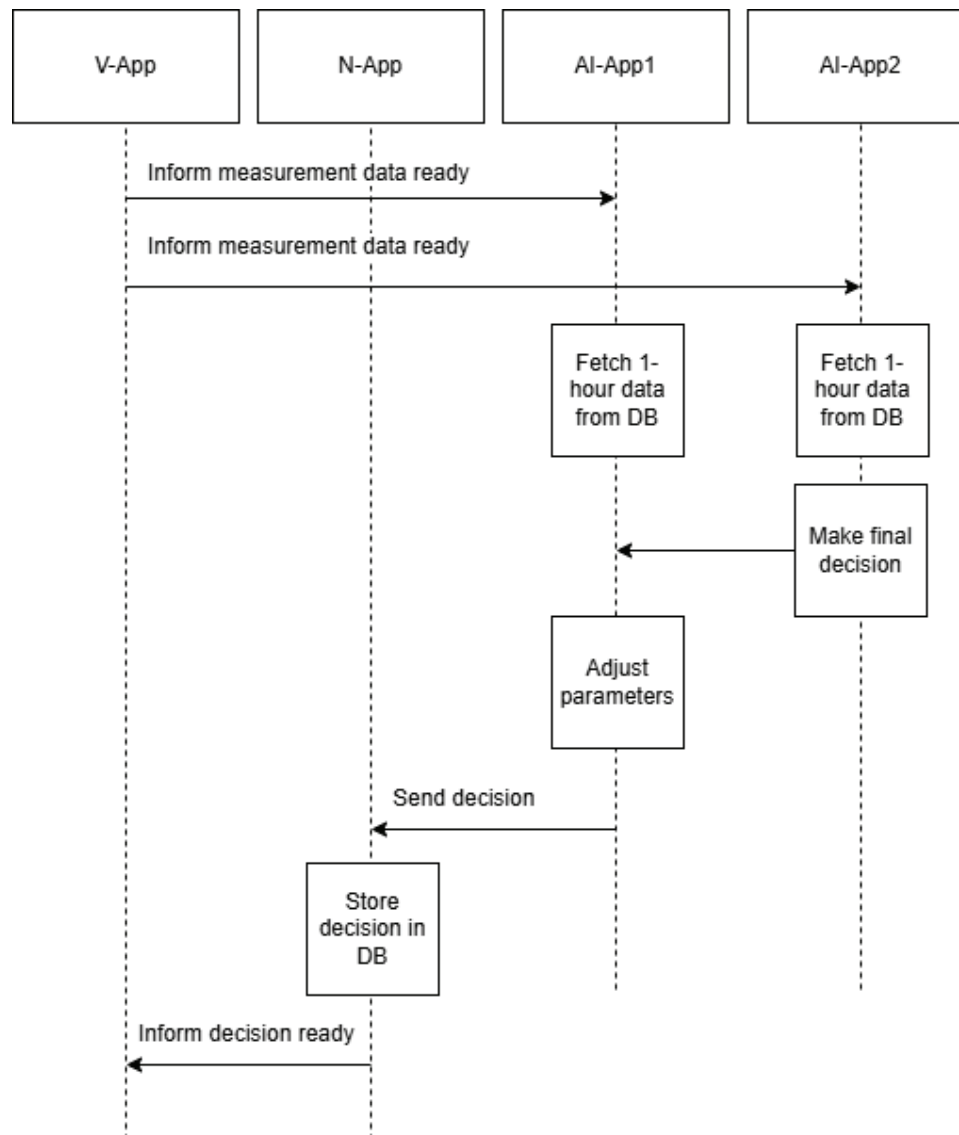


Figure 111. Sequence of AI decision handling

6.2.1.2 Communication mechanisms

All applications communicate with each other using REST APIs. Mainly three response codes are used: 200 – indicates successful request, 400 – indicates problem with request format, e.g. JSON payload, and 500 – indicates recipient could not successfully handle the request due to an error. In case of an error (response codes 400 and 500), description of the error is put in the response payload (Content-Type: application/text).

6.2.1.2.1 API-based communication

The most relevant end-points exposed by the triplet applications are presented in the table below.

Table 33 End-points exposed by the triplet applications

REST Method	Resource URI	Body Parameter	Description
POST	/fmi	None	N-App: fetch weather forecast information from FMI and store it in the database

POST	/ai-decision	{ "timestamp": 1772709600000, "hydrogen_on": true, "battery_soc_limit": 30.5, "pv_power_kw": 2.4, "wind_power_kw": 5.1, "site_load_kw": 6.0, "energy_surplus_kw": 1.5, "predicted_traffic_load": 0.82, "oai_box_configuration": { "mimo": "4x4", "bandwidth_mhz": 100, "tdd_ratio": "DDDDDDDFUU" }, "oai_box_energy_state": "power saving" }	N-App: receive AI decision and store it in the database
POST	/network-ic-data	{ "measurements": [{ "time": 1768820040004, "device": "Output-1", "energy": 231.76 }, { "time": 1768820041004, "device": "Output-2", "energy": 1231.76 }] }	N-App: receive measurement data from network intelligent controller and store it in the database
POST	/remote-ic-data	{ "storage": { "soc": { "value": 50.5, "timestamp": 1774953603680 }, "soh": { "value": 32, "timestamp": 1774953603680 }, "sensing": { "windspeed": { "value": 3.4, "timestamp": 1774953603680 }, "wind_direction": { "value": 65.7, "timestamp": 1774953603680 }, "solar_irradiance": { "value": 34.3, "timestamp": 1774953603680 }, "temperature_outside": { "value": -19.4, "timestamp": 1774953603680 }, "temperature_battery": { "value": 15.4, "timestamp": 1774953603680 }, "temperature_hydrogen": { "value": 20.4, "timestamp": 1774953603680 }, "temperature_devicebox": { "value": 17.4, "timestamp": 1774953603680 }, "production": { "wind": { "value": 30.4, "timestamp": 1774953603680 }, "solar": { "value": 50.4, "timestamp": 1774953603680 }, "hydrogen": { "value": 399.4, "timestamp": 1774953603680 }, "consumption": { "radiounit": { "value": 60.1, "timestamp": 1774953603680 }, "microwave_link": { "value": 60.1, "timestamp": 1774953603680 }, "hybrid_inverter": { "value": 60.1, "timestamp": 1774953603680 }, "hvac": { "value": 60.1, "timestamp": 1774953603680 }, "switch": { "value": 60.1, "timestamp": 1774953603680 }, "hydrogen_cell": { "value": 60.1, "timestamp": 1774953603680 }, "raspberrypi": { "value": 60.1, "timestamp": 1774953603680 }, "sensors": { "value": 60.1, "timestamp": 1774953603680 }, "netio": { "value": 60.1, "timestamp": 1774953603680 } } } }	N-App: receive measurement data from remote intelligent controller and store it in the database
POST	/xapp-data	{ "ue_count": 2, "ue": [{ "ue_id": 2, "timestamp": 1754303881569, "ul_mcs_mean_last_hour": 7, "dl_mcs_mean_last_hour": 8, "ul_bler_mean_last_hour": 0.008863, "dl_bler_mean_last_hour": 0.0, "sinr_mean_last_hour": 22.5, "rsrp_mean_last_hour": -73, "rsrq_mean_last_hour": -10.5, "ul_throughput_bytes_mean_last_hour": 233742, "dl_throughput_bytes_mean_last_hour": 233742, "ul_throughput_bytes_max_last_hour": 234002, "dl_throughput_bytes_max_last_hour": 234002 }] }	N-App: receive network statistics data from xApp and store it in the database

		"dl_throughput_bytes_max_last_hour": 234002}}}	
POST	/ai-decision-ready		V-App: receives a notification from Napp that the AI processing result is ready to be visualized
POST	/traffic	{"ul_throughput_bytes_mean_next_hour" = 2000 "dl_throughput_bytes_mean_next_hour" = 2000 }	AI-App2 (Arctic Traffic Controller): sends predicted traffic load for the next hour to Arctic Energy Controller

6.2.1.3 Results

Our solution is still in active development phase and no integrations into the bare metal server has been made yet. We are currently developing the triplet and other applications separately and for some applications (namely N-App) local testing has been concluded so far. Before the integration of the whole system, we are soon ready to conduct the remaining local tests as the other applications reach maturity.

Whole system-wide integration and deployment will be carried out by Trial 2. We will report results for triplet integration once we reach maturity for Trial 2. We are currently starting our integration activities and for that N-App and Network intelligent controller have already been deployed on the server. Further, X-App has been installed and run on OAIBOX/RIC. However, system-wide testing/integration has not been carried out yet.

6.3 Greek Cluster

6.3.1.1 Sequence Diagram

The sequence diagram in Figure 112 illustrates in detail the set of processes, data and interactions among the various applications composing the Greek cluster that was tested during phase 1 trials. These interactions collectively define the operational workflow for wildfire monitoring, which subsequently transitions into search and rescue activities. Components depicted in gray are already integrated into the overall sequence; however, their full operational capabilities will be validated during the Phase 2 trial. This upcoming phase along with an updated diagram will demonstrate the end-to-end execution of the complete operational scenario, including the active involvement of first responders and the assessment of real-time coordination between all participating entities. The scenario starts with the cameras capturing live video streams (Step 1), which are sent to the wildfire monitoring AI application. The AI processes the video and, when smoke is detected, it generates a fire alert (Step 2) based on the estimated smoke probability.

This alert is then passed to the Wildfire Command & Control application, where it can be checked either through the user interface (Step 3). Once the fire is confirmed, the workflow moves to the victim's localization phase. Here, the AI application for location awareness is triggered (Step 6), using a user device connected to the 5G network. The 5G Core, through the location services N-App, sends notifications about the device's connectivity and movement within the cell of interest. Based on this, the AI estimates whether a person is likely to remain in a specific area for a specific interval. Next, the system generates and ranks possible victim locations (Step 7), which are sent to the INUS Asset Management platform. This platform uses AI-based services for tracking, virtual fencing, and detecting if someone is in a restricted or dangerous area. After that, aerial assets (mUAV) are assigned to check the identified locations (Step 8). The UAV moves toward these points and explores them to confirm the presence of victims (Step 9), while sending back live video (Steps 10–11) for further analysis at the edge and other connected components. Meanwhile, system performance and end-to-end metrics are collected (Step 5.2) and visualized through Grafana.

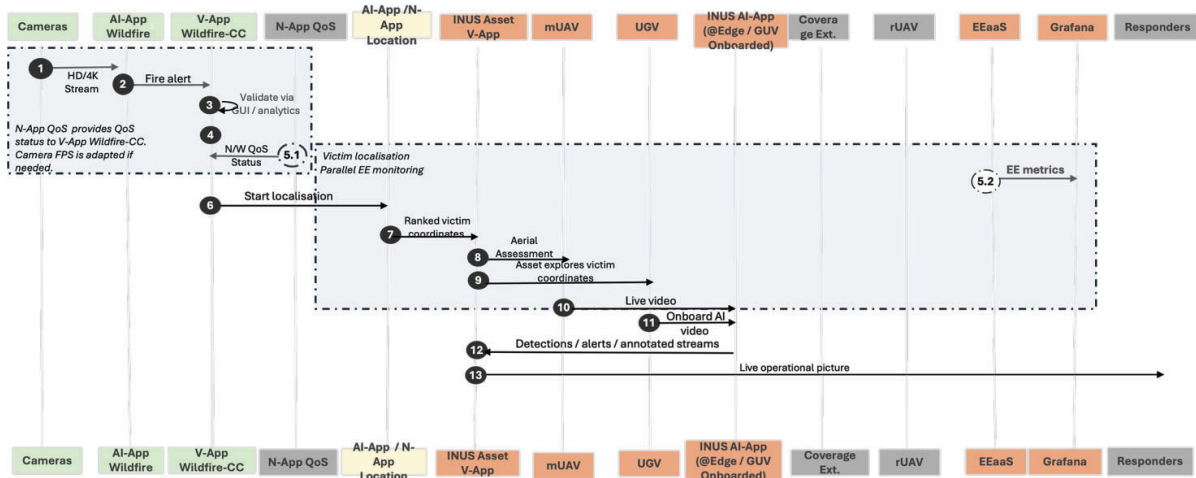


Figure 112 Sequence diagram for the interaction of the components in the Greek use case

6.3.1.2 Communication mechanisms

Complementing the sequence diagram, Figure 113 below shows the connections and the overall architecture regarding the integration of the components tested during Trial 1. As can be seen from the Figure, the triplet components are distributed between the two sites of the Athens B5G testbed (NCSRD, OTE). The system relies on a combination of communication mechanisms to support the interaction between components and ensure real-time operation. The **INUS assets**, such as the mUAV and robotic dog, connect to the infrastructure through **5G connectivity**. Video data generated by camera is transmitted to the AI-App using **RTSP/HTTP protocols**, allowing continuous streaming for real-time analysis. For service-level interaction, **REST APIs** are used mainly between the N-App location and the AI-App location, enabling the exchange of network-related information from the Monitoringevent API, as well as communication with monitoring services. Real-time wildfire telemetry from the AI-App to the V-App is decoupled through a central MQTT broker: the AI-App publishes structured JSON messages; the V-App runs multiple MQTT subscribers on the same topic family, updates a camera registry, and pushes updates to the operator GUI via Server-Sent Events (SSE). Where NAS is mounted at both ends, detection images and MP4 evidence are written by the AI pipeline and referenced by filename in MQTT so the V-App can correlate alerts with files without sending full video on every update. This ensures efficient and scalable event distribution, allowing the V-App to receive updates instantly and react accordingly.

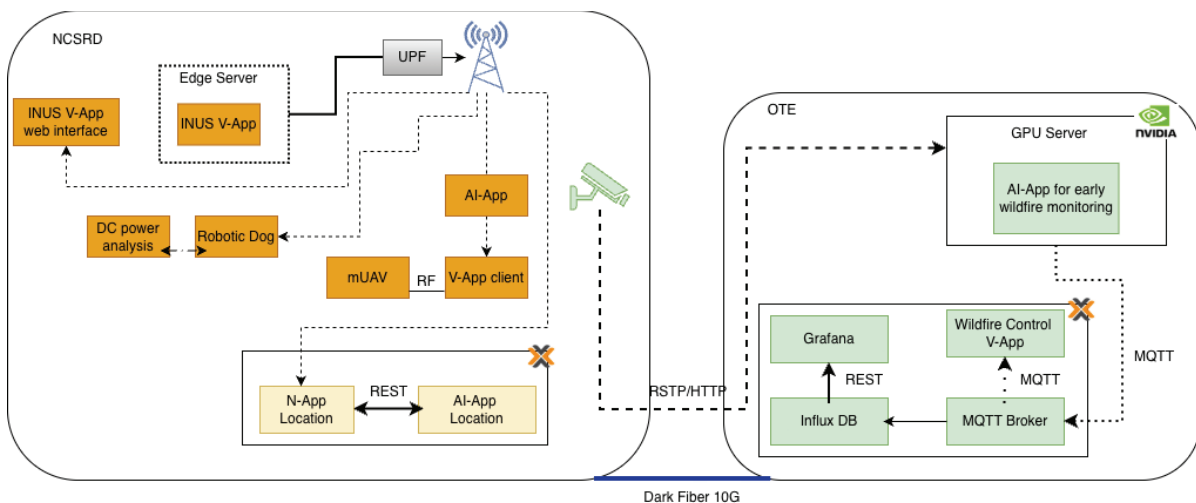


Figure 113 Inter-communication of main components in the Athens B5G testbed

6.3.1.2.1 API-based communication

Table 34 End points exposed by the N-App for location

REST Method	Resource URI	Body Parameter	Description
POSTxx	/3gpp-monitoring-event/v1/{scsAsId}/subscriptions	{"msisdn":"<UE_ID>","notificationDestination":"<AI callback>","monitoringType":"LOCATION_REPORTING","locationType":"LAST_KNOWN_LOCATION"}	AI-App creates a subscription for UE location monitoring
GET	/3gpp-monitoring-event/v1/{scsAsId}/subscriptions	-	Retrieve all active subscriptions
GET	/3gpp-monitoring-event/v1/{scsAsId}/subscriptions/{subscriptionId}	-	Retrieves details of a specific subscription
DELETE	/3gpp-monitoring-event/v1/{scsAsId}/subscriptions/{subscriptionId}		Deletes a subscription and stops monitoring

Table 35 End points exposed by AI-App

REST Method	Resource URI	Body Parameter	Description
POST	https://api.pushover.net/1/messages.json	token, user, message, and optional fields such as url, attachment	AI-App → Pushover: operator push notification when a wildfire detection is confirmed.

Table 36 AI-App MQTT topics

MQTT operation	Topic	Message content	Description
PUBLISH	monitor_testing/wildfire	JSON: site/camera ids, detection_status, frame counters, FPS, optional weather fields, optional image_filename; periodic messages act as heartbeat, alert path includes detection metadata when a fire event is confirmed.	AI-App → broker: telemetry and status so consumers can refresh registry, map risk, and drive alerts.
SUBSCRIBE	monitor_testing/wildfire	JSON: site/camera ids, detection_status, frame counters, FPS, optional weather fields, optional image_filename; periodic messages act as heartbeat,	V-App: ingest messages, update in-memory state, feed SSE to the GUI.

		alert path includes detection metadata when a fire event is confirmed.	
--	--	--	--

6.3.1.3 Results

Regarding the N-App for location (invoker application) the main functionality focuses on the proper interaction with the core network via invoker's application interaction with any external application that asks for network location data in a way that an invoker securely requests that data on behalf of the external application from NEF API by leveraging the CAPIF Framework for securely access the NEF's API endpoints that has been exposed as well through CAPIF After the invoker application receive any response from the provider application, in our case NEF MonitoringEvent API, passes it in the external application that initially asked for location information data. The N-App has been tested during the trial connecting a device to the Amarisoft RAN and Open5GS core in order to consume the Monitoring event API. The relevant logs can be seen below including the information of the API. That information is forwarded to the AI-App for location-aware analysis.

```
INFO: 10.220.2.106:56234 - "POST /invoker-app/v1/location HTTP/1.1" 202 Accepted
2026-03-06:10:42:18,353 INFO [helper.py:56] Extracting callback URL from payload
2026-03-06:10:42:18,354 INFO [helper.py:22] Building monitoring event subscription request with current_loc_enabled=False
2026-03-06:10:42:18,354 INFO [location_fetcher.py:41] Constructed monitoring event request body: accuracy=None externalId=None msisdN='001010123456789' monitoringType='LOCATION_REPORTING' maximumNumberOfReports=None monitorExpireTime=None locationType='LAST_KNOWN'
2026-03-06:10:42:18,354 INFO [location_fetcher.py:44] Sending monitoring event subscription request to http://nef:8000/3gpp-monitoring-event/v1/1/subscriptions with payload: {'accuracy': None, 'externalId': None, 'locationType': 'LAST_KNOWN', 'monitoringType': 'LOCATION_REPORTING', 'monitoringType': 'LOCATION_REPORTING', 'locationType': 'LAST_KNOWN_LOCATION'} and headers: {'Accept': 'application/json', 'Content-Type': 'application/json'}
2026-03-06:10:42:18,373 INFO [helper.py:113] Sending POST request to http://nef:8000/3gpp-monitoring-event/v1/1/subscriptions with payload: {'accuracy': None, 'externalId': None, 'locationType': 'LAST_KNOWN', 'monitoringType': 'LOCATION_REPORTING', 'monitoringType': 'LOCATION_REPORTING', 'locationType': 'LAST_KNOWN_LOCATION'} and headers: {'Accept': 'application/json', 'Content-Type': 'application/json'}
2026-03-06:10:42:18,373 INFO [helper.py:113] Status Code: 200
2026-03-06:10:42:18,373 INFO [helper.py:119] Success: {'msisdN': '001010123456789', 'locationInfo': {'cellId': '001234502', 'trackingAreaId': '001234502', 'locationType': 'LAST_KNOWN', 'monitoringType': 'LOCATION_REPORTING', 'eventTime': '2026-03-06T10:42:18.373Z'}, 'locationType': 'LAST_KNOWN', 'monitoringType': 'LOCATION_REPORTING', 'eventTime': '2026-03-06T10:42:18.373Z'}
2026-03-06:10:42:18,398 INFO [location_fetcher.py:54] Last known location reporting is enabled, proceeding with immediate data report.
2026-03-06:10:42:18,398 INFO [location_fetcher.py:56] Sending response to callback URL http://10.220.2.106:8004/receive with body {'msisdN': '001010123456789', 'locationInfo': {'cellId': '001234502', 'trackingAreaId': '001234502', 'locationType': 'LAST_KNOWN', 'monitoringType': 'LOCATION_REPORTING', 'eventTime': '2026-03-06T10:42:18.398Z'}, 'locationType': 'LAST_KNOWN', 'monitoringType': 'LOCATION_REPORTING', 'eventTime': '2026-03-06T10:42:18.398Z'}
2026-03-06:10:42:18,407 INFO [helper.py:118] Status Code: 200
2026-03-06:10:42:18,408 INFO [helper.py:119] Success: {'status': 'ok', 'msisdN': '001010123456789', 'risk_prob': 0.9404494762420654, 'risk_prob': 0.9404494762420654}
2026-03-06:10:42:18,408 INFO [location_fetcher.py:77] Task registry after removal: {}
```

Figure 114 N-App location (invoker app) logs

AI-App for location Awareness and N-App Location

Live inference from lab environment testing integration between AI-App for Location Awareness and N-App Location

```

[Inference] UE=001010123456789 cell=001234501 dwell=0.0s consec_unsafe=1 unsafe_
frac_T=1.00 changes_T=0
→ WARM-UP (not enough history)
[Cycle] UE=001010123456789 cycle=1 posts_in_cycle=1
INFO: 10.220.2.181:43060 - "POST /receive HTTP/1.1" 200 OK
[Inference] UE=001010123456789 cell=001234501 dwell=499.7s consec_unsafe=25 unsa
fe_frac_T=1.00 changes_T=0
→ READY | prob=0.830 | pred=1 (tau=0.75)
*** ALERT: risk_alert=1 for UE 001010123456789
[Cycle] UE=001010123456789 cycle=9 posts_in_cycle=1
INFO: 10.220.2.181:39588 - "POST /receive HTTP/1.1" 200 OK

```

Figure 115 Inference Results

During real-time operation, the AI-App processes UE events sequentially using a sliding window. Initially, it enters a warm-up phase, where no prediction is made until enough history (T steps) is collected, avoiding unreliable decisions. After that, each new event is evaluated by the model, which outputs a probability compared against the threshold ($\tau = 0.75$). If exceeded, a risk alert is triggered. For example, sustained presence in an unsafe area leads to a high probability (e.g., 0.83) and correctly generates an alert, showing that the system detects persistent risk rather than isolated events.

6.3.1.3.1.1 INUS Asset V-App and AI-App Detection & Tracking

The integration between the **INUS Asset V-App** and the **AI-App for Detection and Tracking** was also initially validated in the Greek cluster environment. The validation focused on the processing of live or near-real-time video streams from robotic assets and cameras, and on the forwarding of AI-generated information to the INUS platform for operational visualization and situational awareness.

During the tests, the AI-App successfully processed incoming video frames and produced detection and tracking outputs, including detected objects, bounding boxes, confidence scores, tracking IDs, and abnormal-event indications. The runtime logs confirmed that the AI-App was able to analyse consecutive frames in near real time, extract object-level information, assign tracking identifiers, and identify abnormal crowd-related events when relevant.

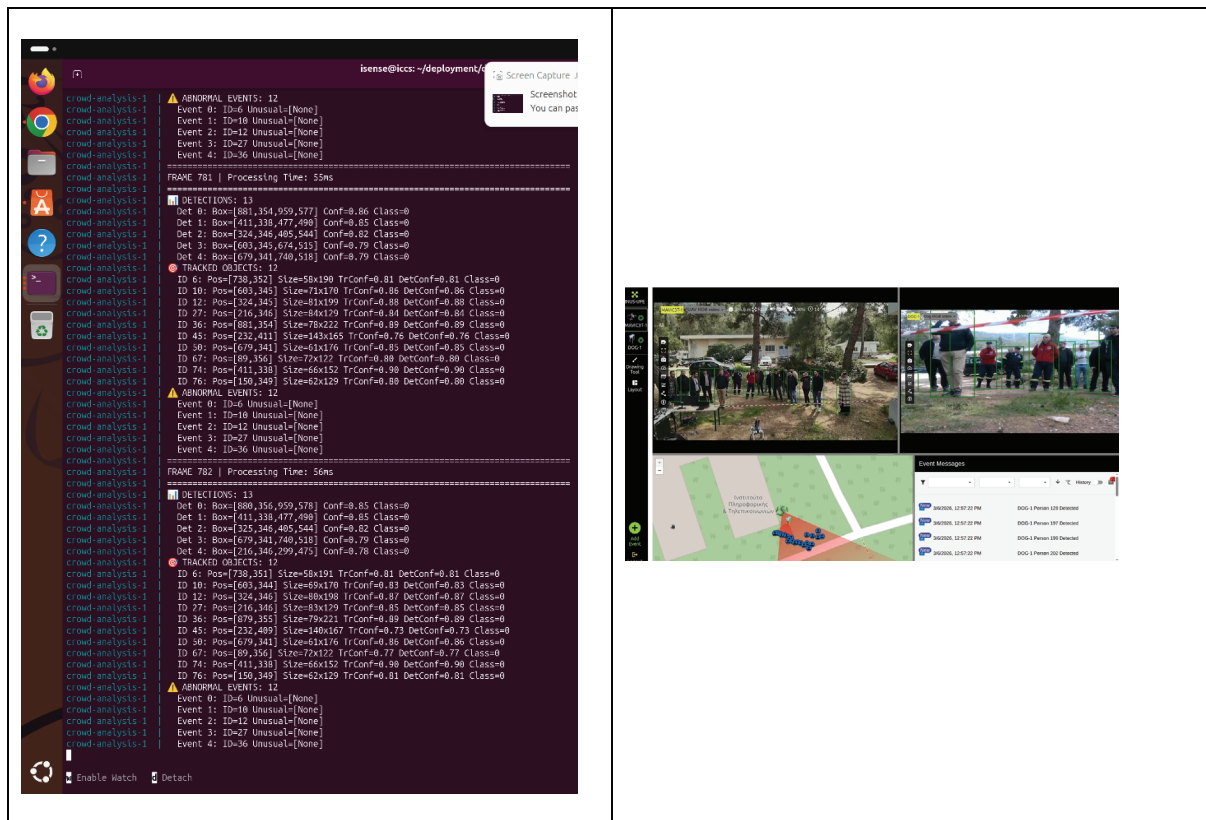


Figure 116 AI-App Detection and Tracking runtime logs

The generated AI outputs are intended to be consumed by the INUS Asset V-App in order to enrich the operator dashboard with AI-assisted information. Through this integration, the INUS V-App can combine asset monitoring, live video feeds, map-based visualization, and AI-generated detections into a common operational picture. This supports improved situational awareness during emergency-response and search-and-rescue scenarios, where UAVs, robotic assets, and fixed cameras may provide complementary visual information from the field.

Overall, the initial results confirm the feasibility of the INUS and AI-App Detection & Tracking integration. The AI-App demonstrated the capability to process video streams and generate structured detection/tracking metadata, while the INUS Asset V-App provides the operational layer where this information can be visualized and exploited by the end user. Further integration work will focus on improving the end-to-end data flow, enhancing the visualization of AI events in the INUS dashboard, and validating the workflow with additional robotic assets during Phase 2 trials.

6.4 French Cluster

The French cluster integrates the V/AI/N-Apps within the EURECOM (EUR) 5G experimental facility, relying on an OpenAirInterface (OAI) 5G Standalone Core Network and heterogeneous RAN environments, including OAI-based deployments and a RAN platform controlled through the Thales SIX GTS France (TSGF) RIC for AI-driven RAN experimentation (See Figure 117). The facility provides extreme-edge and edge computing environments. Edge services will be deployed and managed through the EURECOM Web Portal, which enables lifecycle management of containerized network functions (CNFs). Cloud resources are hosted externally by Greencityzen (GREEN), where several V-Apps and AI-Apps are deployed. At the Extreme Edge, the V-App Soil Sensor has been integrated with a Greencityzen soil sensor and 5G RedCap UE (Quectel RG255C series) operating over the OAI FR1 RAN. End-to-end data-plane validation has been successfully performed toward the cloud through the V-App B5G Connector, enabling telemetry ingestion by the V-App Environmental Indicators Dashboard and V-App Abnormal Watering Detection applications. One sensor–

RedCap pair is currently operational in the testbed, while three additional RedCap devices are already integrated, awaiting additional sensors to complete a four-node sensing deployment.

At the edge–cloud interaction layer, the V-App B5G Connector is currently deployed in the GREEN cloud environment for initial validation and will be integrated in the EURECOM edge environment in the next phase. This component will integrate NEF and NWDAF CAPIF clients, enabling secure exposure of network capabilities and analytics through the CAPIF framework, which has already been deployed and validated in the EURECOM 5GC environment, including CAPIF Core, Provider, and Invoker functions. Other V-Apps are deployed in the cloud because there are no strict real time constraint and data is usually mastered by municipalities which have their own requirements regarding deployment.

On the N-App side, the NEF–CAPIF client has been fully implemented and validated in the EURECOM laboratory. It operates as a lightweight Application Function (AF) implemented in Go that performs CAPIF onboarding, service discovery, OAuth2 authentication, and subscription to NEF Monitoring Event APIs. The application supports concurrent device monitoring and retrieves events such as UE Reachability, PDN Connectivity Status, and Location Reporting, enabling real-time monitoring of the Greencityzen sensor devices. The NWDAF–CAPIF client integration is currently ongoing, enabling future exposure of network analytics through the 3GPP service-based architecture.

Regarding AI components, two AI applications developed by GREEN and operating at the vertical level are being integrated with the corresponding V-Apps: AI-App Sensors Virtualization and AI-App Clogging Detection. Both are initially deployed in the GREEN cloud environment, with the AI-App Clogging Detection planned to be migrated to the EURECOM edge environment in a second phase. In parallel, the AI-App RAN Optimization for Battery Lifetime Extension, developed by TSGF, interacts with the RAN through the N-App RAN/AI-App Interface. This component is integrated as an xApp within the Thales RIC controlling the gNB, enabling AI-driven dynamic reconfiguration of RAN parameters. Validation tests of this closed-loop control mechanism are currently ongoing.

For AI lifecycle management, EURECOM's MLOps platform has been deployed on a Kubernetes cluster and exposes a Training API through the AI Fabric orchestration layer, with MLflow used for experiment tracking and model registry, and MinIO for dataset and artifact storage. This platform will be used by TSGF to train and deploy models for the AI-based RAN optimization use case. At the RAN access layer, the facility integrates 3GPP Release-17 RedCap support and a Two-Step RACH implementation merged into the OAI codebase, enabling low-power and low-latency access procedures for massive IoT devices. RedCap connectivity has been validated using commercial Quectel RG255C devices with the OAI 5G SA network. The Two-Step RACH procedure has been validated using OAI UE and gNB implementations, while validation with commercial hardware UEs will be performed once compatible devices become available on the market. This integration enables end-to-end validation of the vertical stack, from sensor devices and RedCap connectivity at the extreme edge, through 5G network exposure and analytics (NEF/NWDAF), up to AI-driven RAN optimization and cloud-based environmental data analytics services within the French cluster testbed infrastructure.

6.4.1.1 Sequence Diagram

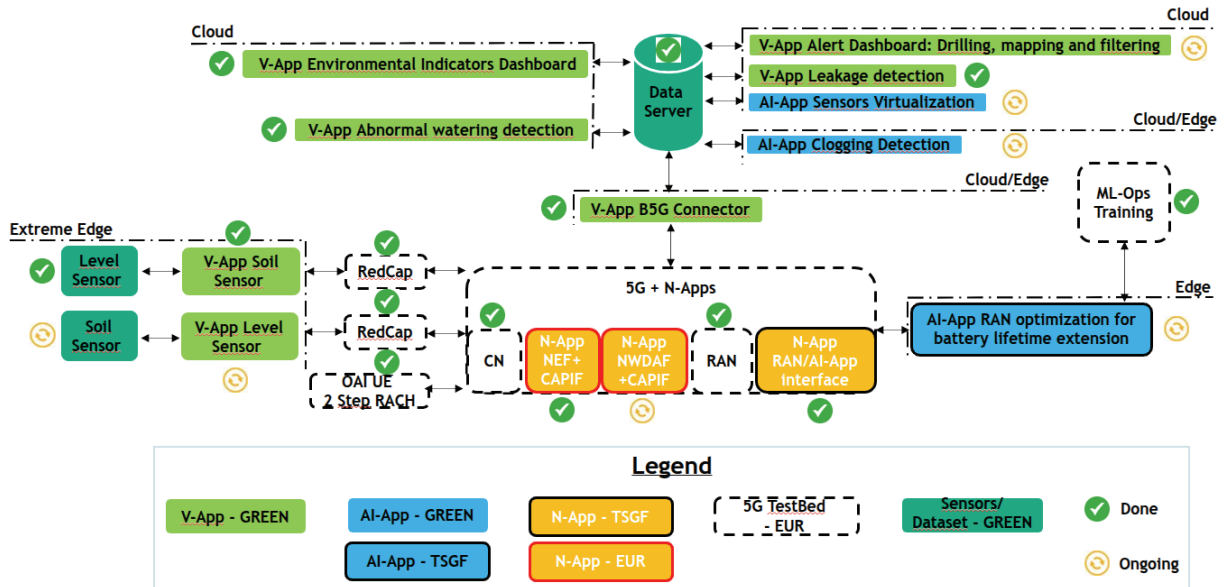


Figure 117 French 5G Extreme Edge-to-Cloud Testbed Integrating Sensors, V/AI/N-Apps, MLOps, and CAPIF

6.4.1.2 Communication mechanisms

The French cluster relies on a combination of standardized and lightweight communication mechanisms:

- **REST APIs (HTTP/HTTPS)**
Used for:
 - CAPIF interactions (onboarding, discovery, security)
 - NEF Monitoring Event APIs
 - ML-Ops training APIs for AI-App
- **5G Service-Based Interfaces (SBI)**
Used internally within the OAI 5G Core (e.g., AMF, SMF, NEF, NWDAF).
- **User Plane (GTP-U over UDP/IP)**
Used for transporting sensor data from RedCap/2 Step RACH UEs to V-Apps via UPF.
- **WebSocket Interface**
Used for interaction between the N-App RAN/AI-App Interface (xApp) and the RAN.

6.4.1.2.1 API-based communication

Table 37 End points of triplet integration in French cluster

REST Method	Resource URI	Body Parameters	Description
GET	/data/uplink?device={id}&data={payload}&time={ts}&key={k}	N/A	Sensor data uplink from Greencityzen device via RedCap modem using AT command AT+SEND (HTTP GET request)
GET	/getauth	Basic Auth (username,password)	Retrieve CAPIF access token
POST	/onboardedInvokers	APIInvokerEnrolmentDetails	Onboard N-App as CAPIF invoker
GET	/service-apis?apiInvokerId={id}&apiName=NEF	N/A	Discover NEF APIs via CAPIF

PUT	/trustedInvokers/{apiInvokerId}	ServiceSecurity	Activate security context with NEF
POST	/security/{securityId}/token	client id, grant type, scope	Request OAuth2 token
POST	/3gpp-monitoring-event/v1/{afId}/subscriptions	externalId, monitoringType, notificationDestination	Create NEF monitoring subscription
DELETE	/3gpp-monitoring-event/v1/{afId}/subscriptions/{subscriptionId}	N/A	Delete NEF subscription
POST	/notify/{subscriptionId}	MonitoringNotification	Receive NEF asynchronous events
POST	/mlops/train	dataset reference, parameters	Trigger AI model training

6.4.1.2.2 Other communication methods

In addition to REST-based interactions, the French cluster integrates lower-layer and device-level communication mechanisms to support the end-to-end data flow from sensors to the 5G network and applications.

6.4.1.2.2.1 UART-based Communication (Sensor RedCap Modem)

At the extreme edge, communication between the Greencityzen sensing device (HB platform) and the 5G RedCap modem is implemented using a UART interface (3.3V). The communication follows a lightweight AT command protocol, enabling device control, data transmission, and reception.

UART specifications:

- Baud rate: 57600 b/s
- Data bits: 8
- Stop bits: 1
- Parity: None
- End of line: \r\n

6.4.1.2.2.2 AT Command Protocol

The HB device acts as a master and interacts with the RedCap modem using the following commands:

- **Initialization / readiness**

AT+RDY?

→ OK

- **Sleep mode**

AT+STB

→ OK

- **Uplink data transmission**

AT+SEND=http://url/data/uplink?device=ID&data=PAYLOAD&time=TIMESTAMP&key=KEY

→ SSS:YYY:OK

Where:

- SSS: HTTP status code (e.g., 200 for success)
- YYY: number of bytes received by the server
- device: unique sensor identifier
- data: encoded sensor payload
- time: timestamp
- key: authentication key

6.4.1.3 Results

6.4.1.3.1 Data Plane

In the French cluster, we have integrated four RedCap UEs and one 2-step RACH OAI UE. We connected a pair of soil sensors measuring humidity to a V-App (Soil Sensor) deployed at the extreme edge. We then tested the data plane using both RedCap and 2-step RACH to transmit payload data to the V-App B5G connector in the cloud through the 5G network. At this stage, one sensor is integrated; additional sensors will be incorporated later to establish multiple sensor-UE pairs within the French testbed.

RedCap Payload Transmission: Figure 118 shows the logs of the integrated RedCap system. Figure 119 presents the logs for one V-App, one sensor, and one RedCap UE.

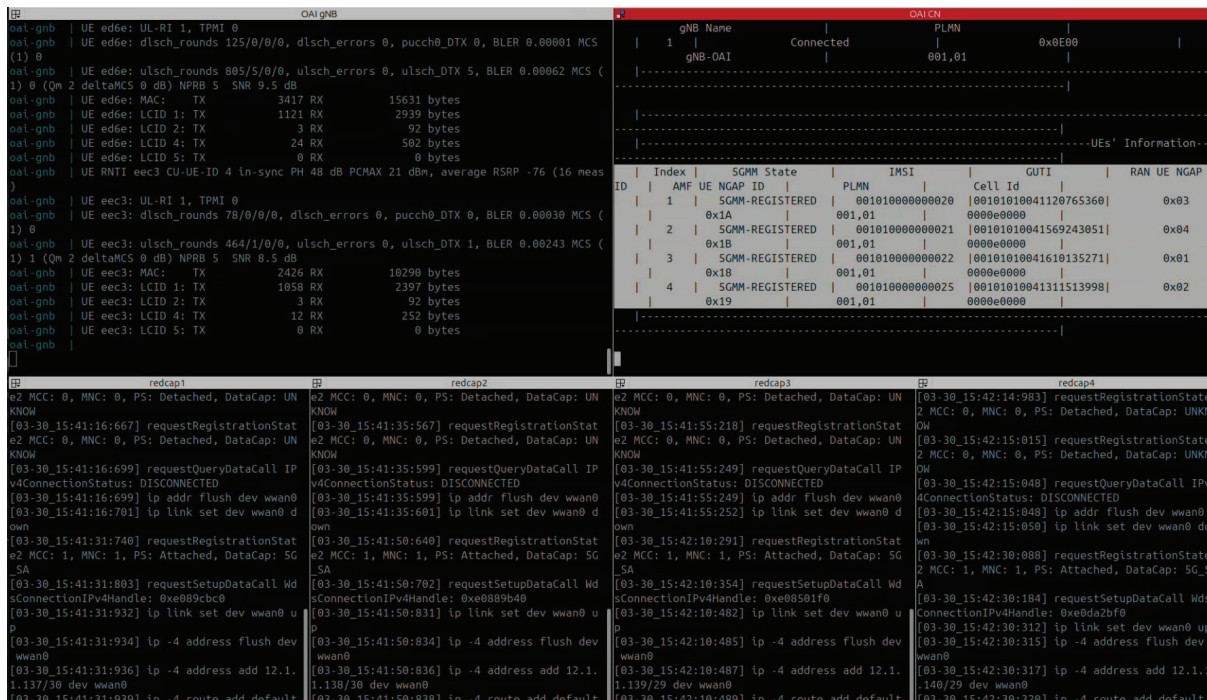


Figure 118 Four RedCap UEs integrated into the French testbed using the OAI 5G Standalone (SA) network



```

nfsd@sgfcellapc:~$ ./oai-gnb-usrp-OTA
[+] 4
UE 9e15: ulsch_rounds 175/0/0/0, ulsch_errors 0, ulsch_DTX 0, BLER 0.00000 MCS 0 0 (Qm 2)
deltaMCS 0 dB) NRPB 5 SNR 47.5 dB CCE Fail 0
UE 9e15: MAC: TX 5887 RX 2769 bytes
UE 9e15: LCID 1: TX 4457 RX 277 bytes
UE 9e15: LCID 2: TX 0 RX 0 bytes
UE 9e15: LCID 4: TX 6 RX 113 bytes

425504.914315 [NR_MAC] I Frame.Slot 640.0
UE RNTI 9e15 CU-UE-ID 1 ln-synr PH 48 dB PCMAX 20 dBm, average RSRP -97 (16 meas)
9e15: UL-R1 1, TPMI 0
UE 9e15: dlsch_rounds 62/0/0/0, dlsch_errors 0, pucch_DTX 0, BLER 0.01501 MCS 0 0 CCE fa
il 4
UE 9e15: ulsch_rounds 190/0/0/0, ulsch_errors 0, ulsch_DTX 0, BLER 0.00000 MCS 0 0 (Qm 2)
deltaMCS 0 dB) NRPB 5 SNR 48.0 dB CCE Fail 0
UE 9e15: MAC: TX 5127 RX 3131 bytes
UE 9e15: LCID 1: TX 4457 RX 277 bytes
UE 9e15: LCID 2: TX 0 RX 0 bytes
UE 9e15: LCID 4: TX 9 RX 167 bytes

[+]
[+] oai-gnb-usrp-OTA
[+] 250170.222956 [NR_MAC] I UE 0 RNTI 9e15 stats: SFR: 512.8, cumulated bad DCI 0
DL harq: 60/0
UL harq: 176/0 avg code rate 0.1, avg bit/symbol 2.0, avg per TB: (nb RBs 8.8, nb symb
ls 8.6)
250170.290098 [NR_PHY] I [UE 0] RSRP = -97 dBm
250170.379099 [NR_PHY] I [UE 0] RSRP = -96 dBm
250170.450904 [NR_PHY] I [UE 0] RSRP = -96 dBm
250170.539106 [NR_PHY] I [UE 0] RSRP = -96 dBm
250170.619101 [NR_PHY] I [UE 0] RSRP = -97 dBm
250170.699098 [NR_PHY] I [UE 0] RSRP = -98 dBm
250170.779100 [NR_PHY] I [UE 0] RSRP = -99 dBm
250170.859093 [NR_PHY] I [UE 0] RSRP = -100 dBm
250170.939097 [NR_PHY] I [UE 0] RSRP = -98 dBm
250171.019112 [NR_PHY] I [UE 0] RSRP = -96 dBm
250171.099085 [NR_PHY] I [UE 0] RSRP = -96 dBm
250171.179093 [NR_PHY] I [UE 0] RSRP = -96 dBm
250171.259085 [NR_PHY] I [UE 0] RSRP = -97 dBm
250171.339063 [NR_PHY] I [UE 0] RSRP = -98 dBm
250171.419087 [NR_PHY] I [UE 0] RSRP = -99 dBm
250171.499054 [NR_PHY] I [UE 0] RSRP = -99 dBm

[+]
[+] oai-gnb-usrp-OTA
[+] gNBs' Information
+-----+-----+-----+-----+-----+
| Index | Status | PLMN | Global Id |
+-----+-----+-----+-----+
| 1 | Connected | 001,01 | 0x0E00 |
+-----+-----+-----+-----+
| gNB-OAI | 001,01 |
+-----+-----+-----+-----+

[+] UEs' Information--
+-----+-----+-----+-----+-----+
| Index | AMF UE NGAP ID | SCGM State | PLMN | IMSI | Cell Id | GUTI | RAN UE NGAP |
+-----+-----+-----+-----+-----+
| 1 | 0x00 | SCGM-DEREISTERED | 001,01 | 000000000002 | 00101010041820419525 | 0000000000 | 0x01 |
+-----+-----+-----+-----+-----+
| 2 | 0x13 | SCGM-REGISTERED | 001,01 | 0000000000025 | 00101010041344369584 | 0000000000 | 0x01 |
+-----+-----+-----+-----+-----+

```

Figure 120 2-step RACH OAI UE successfully attached over the air (OTA) using two USRPs (one for the UE and one for the gNB), with an established PDU session and assigned IP address, in the OAI 5G Standalone (SA) network

Cloud Reception: Figure 121 illustrates the V-App Environmental Indicators Dashboard receiving payload data in the cloud.

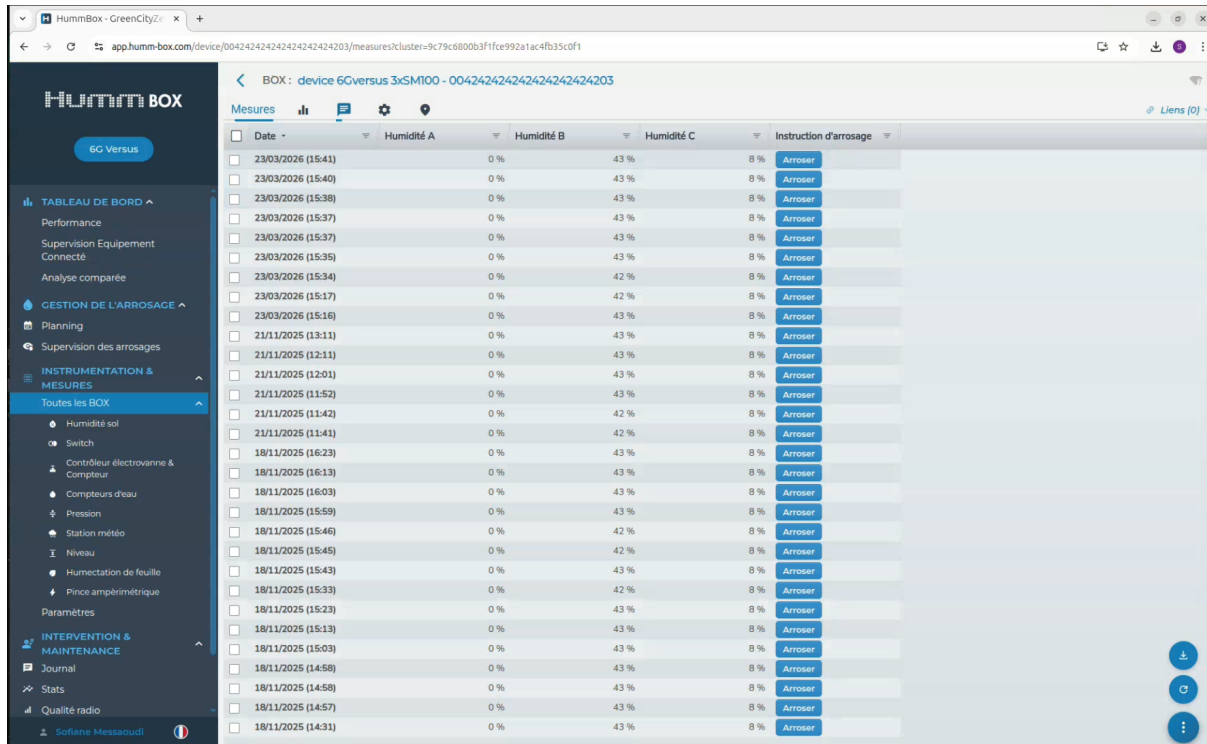


Figure 121 V-App Environmental Indicators Dashboard receiving payloads sent by soil sensors integrated with RedCap and 2-step RACH UEs

6.4.1.3.2 Control Plane

Here, we mainly focus on the integration and testing of the N-App (NEF + CAPIF) within the French testbed. In this setup, the client (i.e., the invoker) subscribes to monitoring events via NEF to receive information about UE status (i.e., UE reachability, PDU session, and location). Figure 122 shows the logs of the tests conducted in the French testbed laboratory. Further integration with the V-App B5G Connector will follow in the second phase.

```

netsoft@vm03:~/capif/services
CapiF Core Function
[{"timestamp": "23/03/2026 16:34:52", "level": "DEBUG", "logger": "service-apis-1", "function": "get_discoveredapls", "line": 52, "message": "Discovering services api s by: INVa7afbc95faf9c1fb018aa77ea70b05"}]
[{"timestamp": "23/03/2026 16:34:52", "level": "INFO", "logger": "service-apis-1", "function": "all_service_apis_get", "line": 80, "message": "Discovering service api s"}]
[{"timestamp": "23/03/2026 16:34:52", "level": "DEBUG", "logger": "service-apis-1", "function": "get_discoveredapls", "line": 52, "message": "Discovering services api s by: INVa7afbc95faf9c1fb018aa77ea70b05"}]
[{"timestamp": "23/03/2026 16:34:52", "level": "INFO", "logger": "capiF-security-1", "function": "cert_validation", "line": 34, "message": "CN: INVa7afbc95faf9c1fb018aa77ea70b05"}]
[{"timestamp": "23/03/2026 16:34:52", "level": "INFO", "logger": "capiF-security-1", "function": "trusted_invokers_api_invoker_id_put", "line": 167, "message": "Creatin g security context"}]
[{"timestamp": "23/03/2026 16:34:52", "level": "DEBUG", "logger": "capiF-security-1", "function": "create_service_security", "line": 249, "message": "Creating security c ontext"}]
[{"timestamp": "23/03/2026 16:34:52", "level": "DEBUG", "logger": "capiF-security-1", "function": "check_invoker", "line": 48, "message": "Checking api invoker with i d: INVa7afbc95faf9c1fb018aa77ea70b05"}]
CapiF Invoker
Event Time: 2026-03-23 16:34:52.701669063 +0000 UTC
[DeviceStatusSvc] 2026/03/23 16:34:52
P2DN Connection Information for redCap-25
Status: CREATED
APN: oat
P2DN Type: IPV4
IPV4 Address: 12.1.1.132
Event Time: 2026-03-23 16:34:52.701669063 +0000 UTC
[NefMonitoringEventClient] 2026/03/23 16:34:52 created LOSS_OF_CONNECTIVITY subscription 94 d16ad1-05d3-4518-8a1a-5090452dbcee for UE versus-app:Yy2Az0MoYnTFvtux5z1GlaqlrDT15k5jJbt56 ogG7N1P08ZLWkUT90PqONUF7LKR9zG8w9GuQeXmo
[DeviceStatusSvc] 2026/03/23 16:34:52
LossOfConnectivity for redCap-25
Reason: DEREGISTERED
Event Time: 2026-03-23 16:34:52.708982325 +0000 UTC
Core Network - NEF-CAPIF Provider
false lat:1.774283692e+09 iss:INVa7afbc95faf9c1fb018aa77ea70b05 jti:f37bf56d-15f2-413c-a45 b:757c6f5e5c8c nbfi:1.774283692e+09 res_owner_id:<nil> scope:3gpp#AEF2f72df4352bec2f0829255e b537c28:3gpp-monitoring-event sub:INVa7afbc95faf9c1fb018aa77ea70b05 type:access]
2026/03/23 16:34:52 Error when calling 'DefaultAPI.AefIdLogsPost': 401 Unauthorized Nis(*st ring=0xc00015790) (Nis(*string=0xc00015790))
2026/03/23 16:34:52 Full HTTP response: 401 Unauthorized 401 HTTP/1.1 1 map[Connection: [keep-alive] Content-Length:[109] Content-Type:[application/json] Date:[Mon, 23 Mar 2026 16 :34:52 GMT] Server:[nginx/1.27.1]] [{"title": "Unauthorized", "detail": "User not authorize d", "cause": "You are not the owner of this resource"}] 109 [] false false map[] 0xc0003121 20 0xc0002b42c0
2026/03/23 16:34:52 map[csrf:9c93b513-efb4-46ee-9aec-58f42a38519f exp:1.774284292e+09 fresh :false lat:1.774283692e+09 iss:INVa7afbc95faf9c1fb018aa77ea70b05 jti:f37bf56d-15f2-413c-a45 b:757c6f5e5c8c nbfi:1.774283692e+09 res_owner_id:<nil> scope:3gpp#AEF2f72df4352bec2f0829255e b537c28:3gpp-monitoring-event sub:INVa7afbc95faf9c1fb018aa77ea70b05 type:access]
2026/03/23 16:34:52 Error when calling 'DefaultAPI.AefIdLogsPost': 401 Unauthorized Nis(*st ring=0xc00056770b) (Nis(*string=0xc00056770b))
2026/03/23 16:34:52 Full HTTP response: 401 Unauthorized 401 HTTP/1.1 1 map[Connection: [keep-alive] Content-Length:[109] Content-Type:[application/json] Date:[Mon, 23 Mar 2026 16 :34:52 GMT] Server:[nginx/1.27.1]] [{"title": "Unauthorized", "detail": "User not authorize d", "cause": "You are not the owner of this resource"}] 109 [] false false map[] 0xc0003139 e0 0xc0002b4420
gNB
carota@batman:~/soflane$ docker compose -f docker-compose-redcap.yml up -d

```

Figure 122 N-App NEF + CAPIF integration and control-plane event subscription logs in the French testbed

6.5 Portuguese Cluster

For the Portuguese cluster, triplet integration is reported scenario-by-scenario in order to maintain explicit traceability between the use-case/scenario defined in the project.

6.5.1 Scenario 1 Triplet

This scenario is centered on energy-efficient, network-aware environmental monitoring for real-time pollution alerts. It enables the Port Administration to quickly identify potential pollution sources, reduce response times, avoid regulatory non-compliance, and manage environmental risks proactively.

The Port Environmental Monitoring V-App receives, processes, and stores data from smart buoys. This information is presented to vertical stakeholders, enabling them to take appropriate actions, such as modifying the energy profile used by the buoys or allowing such adjustments to be triggered automatically based on predefined thresholds. In upcoming trial phases, this behavior will evolve, with the V-App receiving recommendations for energy profile selection over specific time periods provided by the AI-App. The selected energy profile will then trigger an API request to the N-App, cascading subsequent requests to the 5G Core network and the smart buoys.

6.5.1.1 Sequence Diagram

Figure 123 presents a sequence diagram of the interaction between the Port Environmental Monitoring V-App and the QoS N-App where after receiving information from the smart buoys a change in the QoS profile for the device is triggered.

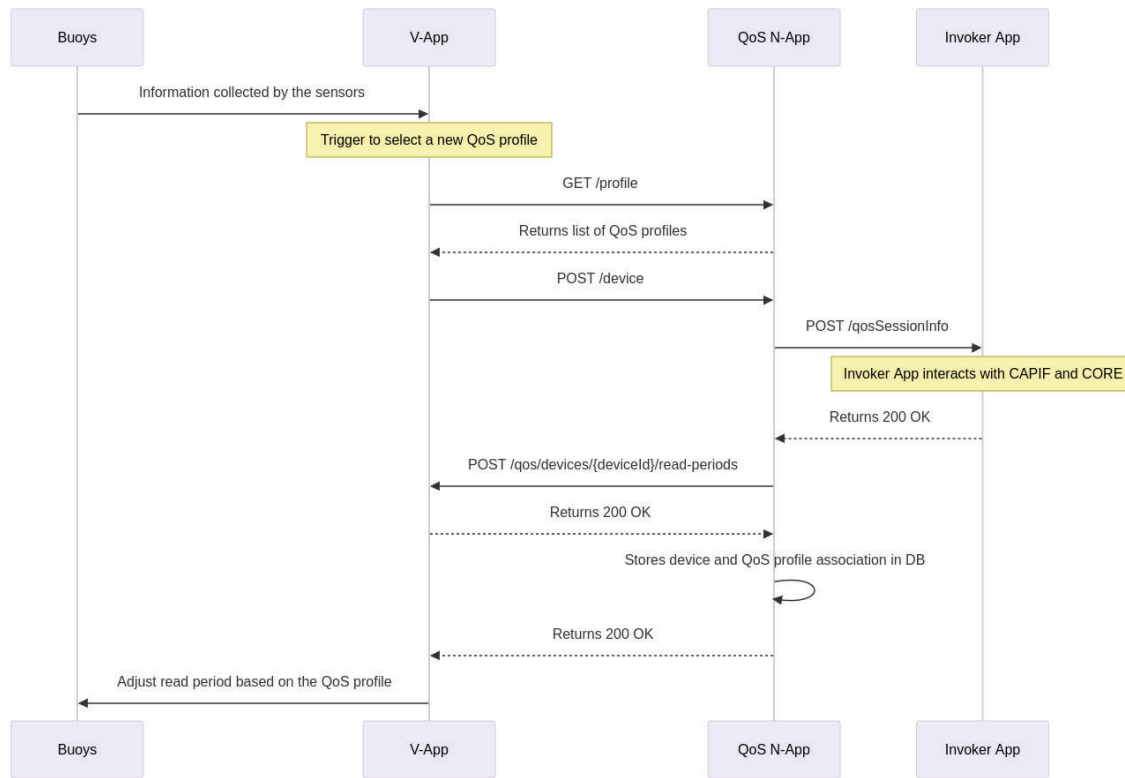


Figure 123 Simplified Scenario 1 sequence diagram

6.5.1.2 Communication mechanisms

All the components (V-App, N-App, CAPIF, NEF, and smart buoys) use REST APIs for communicating - additionally, communications between the V-App and smart buoys is done via websockets.

6.5.1.2.1 API-based Communication

Table 38 End points for triplet integration in the Portuguese cluster

REST Method	Resource URI	Body Parameter	Description
GET	/devices		List all the devices and their respective profiles.
POST	/devices	{ "name": "Device_A", "description": "Device for testing purposes.", "ue_ipv4_address": "127.0.0.1", "mac_address": "00:1B:44:11:3A:B7", "profile_id": 1 }	Create a new device-profile association, activating the QoS profile for the device.
GET	/device/{device_id}		Find a device by their id.
PUT	/device/{device_id}	{ "name": "Device_A", "description": "Device for testing purposes.", "ue_ipv4_address":	Update the device properties, like selecting a new QoS profile.

REST Method	Resource URI	Body Parameter	Description
		"127.0.0.1", "mac_address": "00:1B:44:11:3A:B7", "profile_id": 2 }	
DELETE	/device/{device_id}		Deletes de device-profile association, removing the QoS profile for the device.
GET	/profiles		List all the profiles and their properties.

Additional APIs are available for creating and modifying QoS profiles, however, since they are not used by the vertical and are intended for administrative purposes, they are not included in the table.

6.5.1.3 Results

The results of the interaction between the V-App and QoS N-App can be seen in Figures 124 and 125. Figure 124 presents the user changing the profile for a device using the V-App dashboard, while Figure 125 shows the logging information occurring between the V-App, on top, and the QoS N-App below.

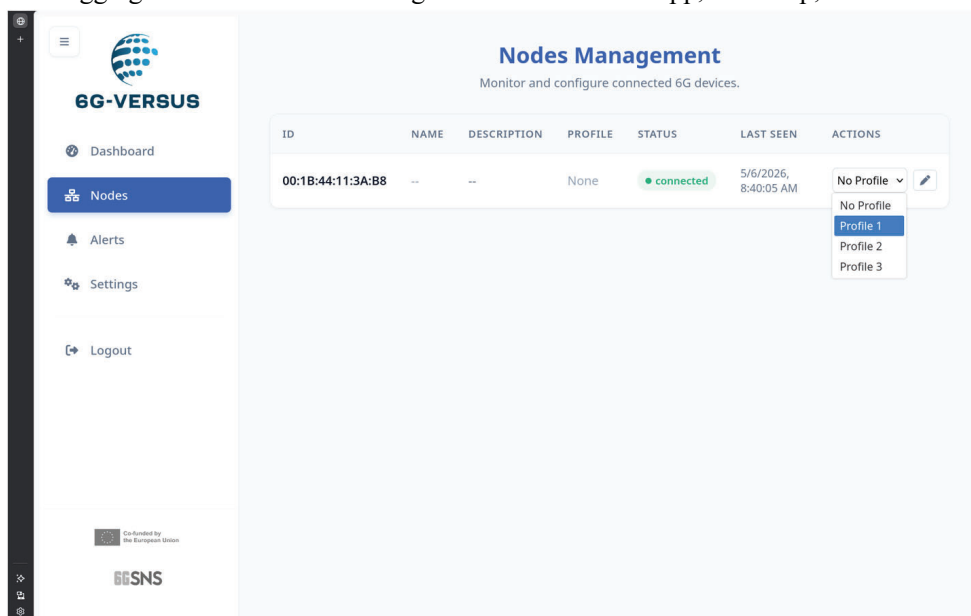


Figure 124. Manual change of a device profile.

```

[2026-05-06 09:39:43 +0100] [158329] [INFO] Using worker: uvicorn.workers.UvicornWorker
[2026-05-06 09:39:43 +0100] [158330] [INFO] Booting worker with pid: 158330
[2026-05-06 09:39:44,051 INFO] [main.py:9] Starting V-APP server
[2026-05-06 09:39:44,128 INFO] [iot_sensors.py:32] InfluxDB3 URL: http://influxdb3-core:8181
[2026-05-06 09:39:44,128 INFO] [iot_sensors.py:35] InfluxDB3 Database: telemetry
[2026-05-06 09:39:44,147 INFO] [qos.py:21] QoS N-App URL: http://127.0.0.1:8001
[2026-05-06 09:39:44,173 INFO] [qos.py:35] Successfully fetched profiles from QoS N-App
[2026-05-06 09:39:44 +0100] [158330] [INFO] Started server process [158330]
[2026-05-06 09:39:44 +0100] [158330] [INFO] Waiting for application startup.
[2026-05-06 09:39:44 +0100] [158330] [INFO] Application startup complete.
[2026-05-06 09:39:52 +0100] [158329] [INFO] Handling signal: winch
[2026-05-06 09:40:16,249 INFO] [qos.py:271] Received update request for device_id: 00:1B:44:11:3A:B8
[2026-05-06 09:40:16,250 INFO] [qos.py:274] Request body: {'profile_id': 1}
[2026-05-06 09:40:16,250 INFO] [qos.py:293] Current db_id for 00:1B:44:11:3A:B8: -1
[2026-05-06 09:40:16,250 INFO] [qos.py:296] Handling profile_id: 1
[2026-05-06 09:40:16,250 INFO] [qos.py:305] Fetching profile 1 from N-App...
[2026-05-06 09:40:16,258 INFO] [qos.py:307] Profile data received: {'description': 'This profile is meant to get a large number of data points quickly. It increases the amount of data collected, and decreases the time it takes to receive data. However, it uses more energy in the device.', 'mbr_uplink': 150000, 'mbr_downlink': 227000, 'gbr_uplink': 150000, 'gbr_downlink': 227000, 'period_uplink': 10, 'period_downlink': 10, 'read_period': 1, 'profile_id': 1}
[2026-05-06 09:40:16,258 INFO] [qos.py:321] Adding new device to N-App
[2026-05-06 09:40:16,305 INFO] [qos.py:367] Updated read periods list for device 00:1B:44:11:3A:B8: {'uplink_period': 10, 'downlink_period': 10, 'read_period': 1}
[2026-05-06 09:40:16,307 INFO] [qos.py:330] Updating local state for 00:1B:44:11:3A:B8

Updating device 00:1B:44:11:3A:B8 with body: {"profile_id": 1}
INFO: 127.0.0.1:34750 - "POST /api/devices/00%3A1B%3A44%3A11%3A3A%3AB8 HTTP/1.1" 200 OK
INFO: 127.0.0.1:34750 - "POST /api/imsi-list HTTP/1.1" 200 OK
INFO: 127.0.0.1:34750 - "POST /api/imsi-list HTTP/1.1" 200 OK
INFO: 127.0.0.1:34750 - "POST /api/imsi-list HTTP/1.1" 200 OK
INFO: 127.0.0.1:34750 - "POST /api/imsi-list HTTP/1.1" 200 OK
]
127.0.0.1 - - [06/May/2026 09:40:02] code 404, message File not found
127.0.0.1 - - [06/May/2026 09:40:02] "GET /favicon.ico HTTP/1.1" 404 -

2026-05-06 09:39:38,863 INFO [main.py:14] Log Filename Path: ./app/logs/app_logger
2026-05-06 09:39:38,863 INFO [main.py:15] QoS API URL: http://invoker-app.n-app.com/invoker-app/v1
2026-05-06 09:39:38,863 INFO [main.py:16] Database URL: sqlite:///default.db
2026-05-06 09:39:38,863 INFO [main.py:17] Creation of default profiles: True
2026-05-06 09:39:38,993 INFO [database.py:30] Database tables not found for sqlite:///default.db.
2026-05-06 09:39:38,993 INFO [database.py:31] Creating a new database for sqlite:///default.db.
2026-05-06 09:39:39,082 INFO [database.py:35] Database created successfully.
2026-05-06 09:39:39,096 INFO [main.py:34] Populating database with 3 default profiles.
INFO: Started server process [158065]
INFO: Waiting for application startup.
INFO: Application startup complete.
INFO: 127.0.0.1:42688 - "GET /profiles HTTP/1.1" 200 OK
INFO: 127.0.0.1:52544 - "GET /profiles HTTP/1.1" 200 OK
INFO: 127.0.0.1:35868 - "GET /profiles/1 HTTP/1.1" 200 OK
2026-05-06 09:40:16,306 INFO [router.py:301] Device characteristics:
{
  "ue_ipv4_address": "10.45.0.31",
  "mac_address": "00:1B:44:11:3A:B8",
  "profile_id": 1
}
2026-05-06 09:40:16,306 INFO [router.py:308] Profile characteristics:
{
  "description": "This profile is meant to get a large number of data points quickly. It increases the amount of data collected, and decreases the time it takes to receive data. However, it uses more energy in the device.",
  "mbr_uplink": 150000,
  "mbr_downlink": 227000,
  "gbr_uplink": 150000,
  "gbr_downlink": 227000,
  "period_uplink": 10,
  "period_downlink": 10,
  "read_period": 1,
  "profile_id": 1
}
2026-05-06 09:40:16,306 INFO [router.py:315] Successfully associated energy profile 1 with device 1.
INFO: 127.0.0.1:35868 - "POST /devices HTTP/1.1" 201 Created

```

Figure 125. Logging information of the Port Environmental V-App (top), and the QoS N-App (bottom).

6.5.2 Scenario 2 Triplet

Scenario 2 focuses on monitoring vehicles transporting chemical products and detecting road congestion, queues, accidents, and other incidents within the port area. The collected data will help the Port Authority assess traffic conditions, identify potential hazards, improve road safety, and optimize traffic management. Real-time alerts will also support faster responses to incidents, reduce disruptions, and strengthen regulatory compliance.

In this scenario, the Port Safety V-app plays the overall coordination of the process, including inter-App communication. The port safety V-App communicates with the Network Operations N-App and the two AI-Apps (Simple Object detection AI-App and HazMat vehicle detection AI-App) via REST APIs.

6.5.2.1 Sequence Diagram

Figure 126 provides a sequence diagram that represents the interaction flow between the AI/V/N-Apps involved in scenario 2, in the case where the system evolves from a low-energy state upon detection of a new vehicle approaching the port premises, executes the required reconfiguration of the network resources, and finally returns to the initial low-energy state.

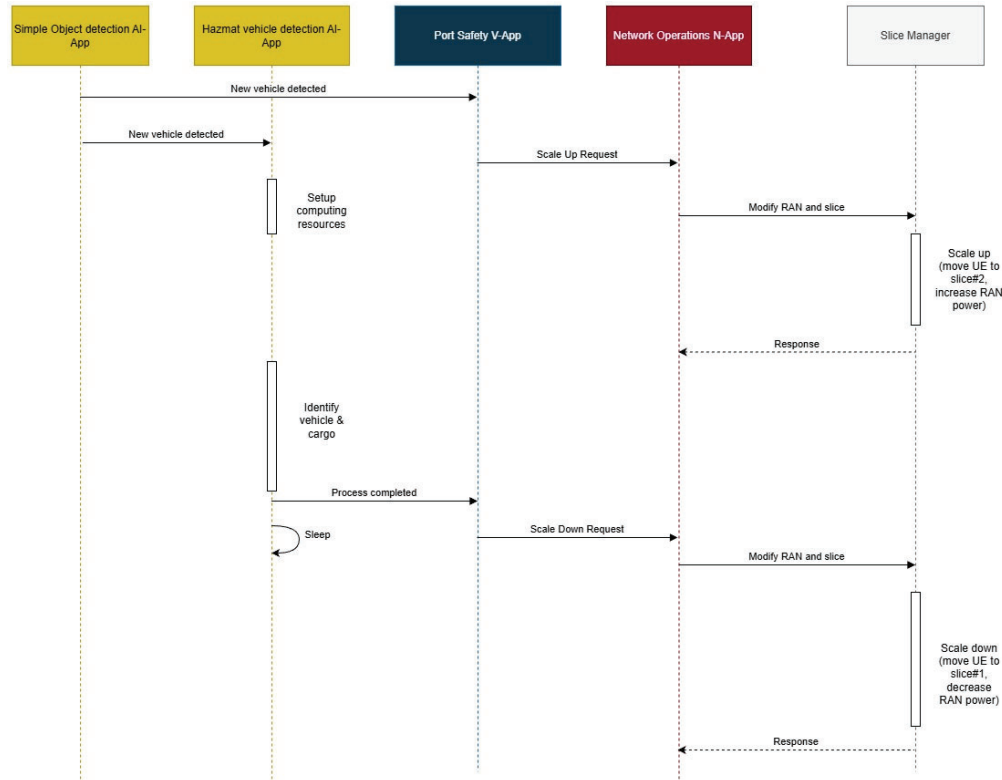


Figure 126 Scenario 2 typical sequence diagram

6.5.2.2 Communication Mechanisms

The REST APIs involved in the V/N-App/5G core communication have been described before, in section 3.6.2.

6.5.2.3 Results

The integration of N-App, V-App and AI-Apps in the context of the Portuguese Cluster scenario 2 was successfully validated and demonstrated in a real environment through Trial 2, which took place on 25/3/2026 in the premises of the Port of Aveiro. Overall system performance, stability and robustness had been verified and fine-tuned previously in the lab environment and on-site pre-trials. The V-App dashboard, shown in Figure 55, provided evidence of the successful integration of the triplet components in the multiple stages of the use case scenario.

6.5.3 Scenario 3 Triplet

The scenario 3 addresses energy-aware infrastructure and sustainable operations, aiming to improve energy efficiency across smart port environments. By using advanced network capabilities, real-time user equipment location data, and edge-based artificial intelligence, the system can dynamically manage services such as public lighting and electric vehicle charging according to user presence, mobility patterns, and energy availability. To this end, smart lighting allows streetlights to adjust their brightness according to activity in the area, saving energy while keeping public spaces safe and well lit, while, intelligent EV charging uses AI to

predict renewable energy availability, especially solar power, and schedule charging when clean energy is more accessible, helping reduce grid dependence and support more sustainable mobility.

The Triplet follows a layered architecture where each component has a specific role. The diagram in Figure 108 represents the interaction between all components. The N-App is responsible for interfacing with the network and providing contextual information, such as connectivity status and device location. The V-App acts as the data aggregation and control layer, ingesting energy-related data from solar panels and EV chargers and exposing dashboards as well as control capabilities. The AI-App functions as an orchestration layer, responsible for receiving outputs from the AI/ML framework and forwarding them to the V-App, without performing any additional processing or model inference. Communication between components is primarily based on REST APIs over HTTP, ensuring simple and interoperable integration across the system. The N-App exposes network-related information via API endpoints consumed by the V-App when required. The AI-App receives results from the AI/ML framework and transmits them to the V-App through dedicated endpoints, while the V-App acts as the execution layer, translating these inputs into control actions on EV chargers and updating visualization dashboards accordingly. Data exchanged between components is structured in JSON format to ensure lightweight and standardized communication.

6.5.3.1 Sequence Diagram

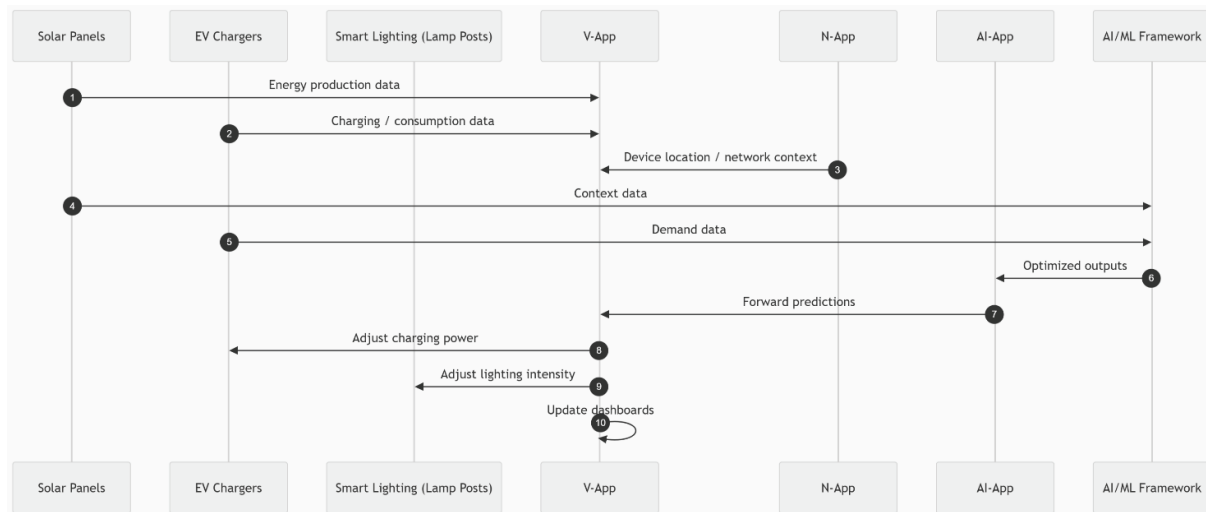


Figure 127 Scenario 3 triplet integration sequence diagram

6.5.3.2 Communication Mechanisms

6.5.3.2.1 API-based communication

Table 39 APIs utilized for Triplet

REST Method	Resource URI	Body Parameter	Description
GET	/MyLabs/OcppApi/chargingstations	X-API-Key	Get the last reading from all the charging stations

6.5.3.2.2 Modbus based communication

Method	URI	Parameters	Description
GET	/modbus/registers	address, count, ip, port, unit_id	Generic register read
GET	/modbus/model/{model_id}	model_id, base_addr, ip, port, unit_id	Read specific SunSpec model

6.5.3.3 Results

The V-App was able to ingest real-time data from solar panels and EV chargers, validating the integration with external energy infrastructure. This data was processed and visualized through preliminary dashboards (see Figure 128, 129, 130 and 131), providing visibility over energy production and consumption. In parallel, the N-App was successfully connected to the network, enabling the exposure of network-related capabilities that will support future context-aware functionalities, such as device location. Although the full triplet workflow (including AI-driven optimization) was not executed, the interaction between the deployed components confirmed the correct operation of the foundational layers of the architecture. In particular, the system validated the end-to-end flow from data acquisition to visualization, establishing a solid baseline for the integration of the AI-App in future phases. Application logs and system outputs confirmed stable data ingestion and processing, with no major issues observed. However, it should be noted that advanced functionalities such as AI-based decision-making and automated actuation were not yet implemented at this stage. Overall, the results demonstrate the readiness of the current system to evolve towards a fully integrated triplet architecture, supporting energy optimization use cases in future iterations.

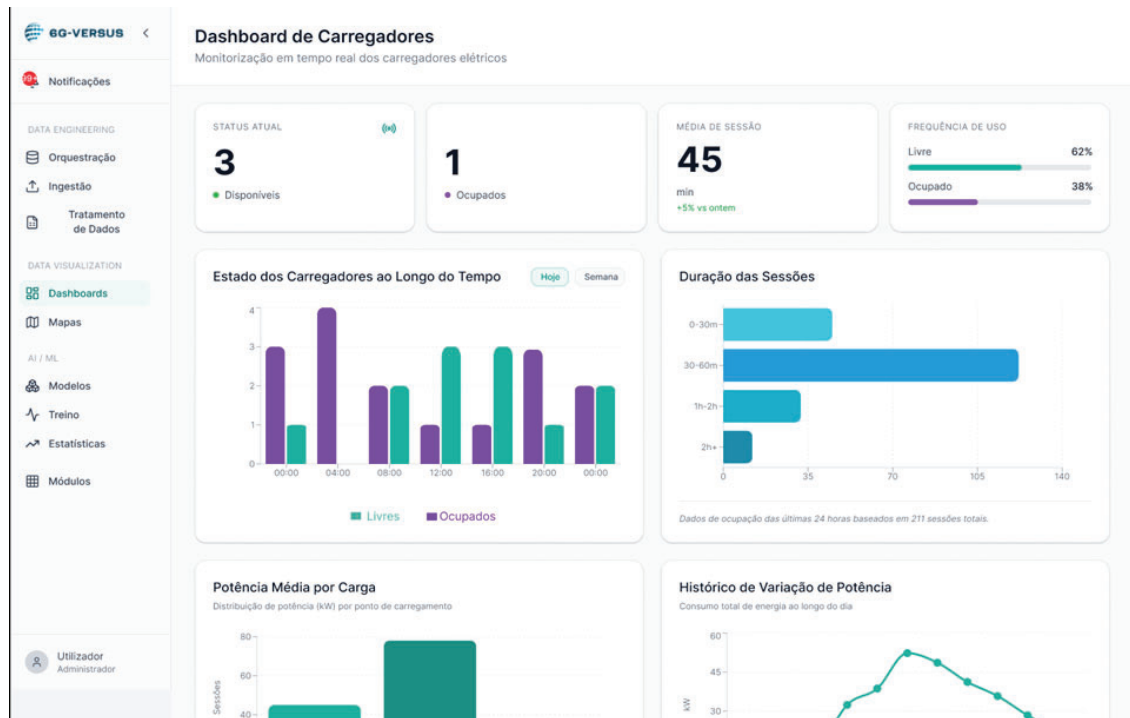


Figure 128 - Snapshot of the Infrastructure Energy Tracking V-App dashboard for EV Chargers 1



Figure 129 – Snapshot of the Infrastructure Energy Tracking V-app dashboard for EV Chargers 2

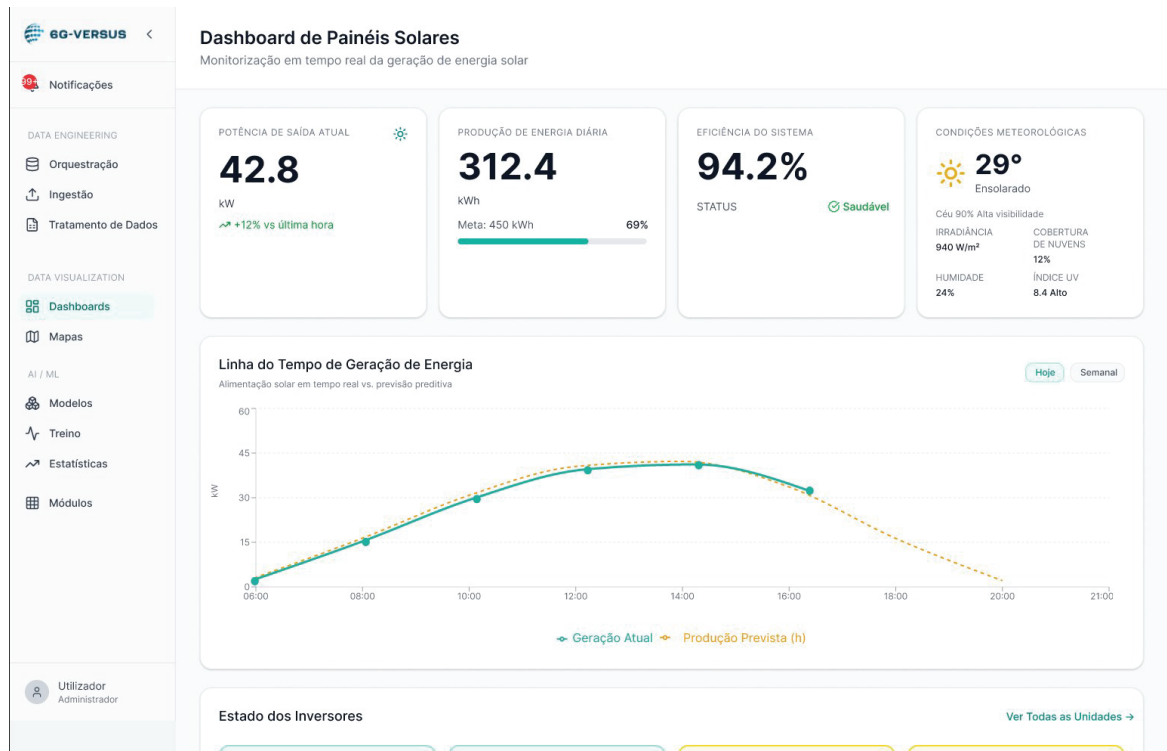


Figure 130 - Snapshot of the Infrastructure Energy Tracking V-App dashboard for Solar panels 1

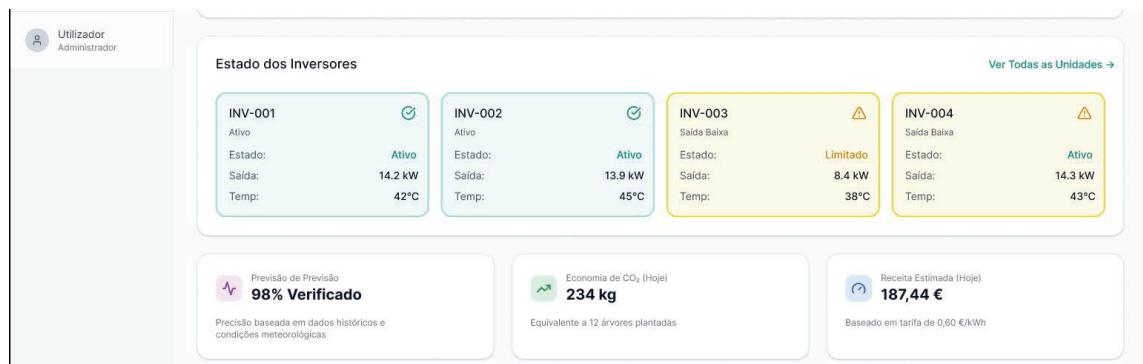


Figure 131 - Snapshot of the Infrastructure Energy Tracking V-App dashboard for Solar panels 2

6.6 Spanish Cluster

6.6.1 Activity 1: DSS for water quality

In the DSS for water quality triplet, the three application components play complementary roles to support continuous monitoring, decision support, and communication with end users. The N-App provides the connectivity support for the field-deployed water-quality sensors. Its function is to ensure reliable network attachment and stable transmission of measurements so that data can be delivered continuously and with suitable service quality conditions.. The AI-App acts as the core analytical component. It processes near real-time water-quality data collected from the sensors in order to assess whether the available water is suitable for irrigation and to support decisions on the mixing of reclaimed water with other water sources. Based on this analysis, the AI-App generates decision-support outputs, including alert triggers and the contextual information associated with the detected condition.vThe V-App provides the user-facing layer of the triplet. It receives the outputs generated by the AI-App and transforms them into information that is directly useful for farmers. In particular, it supports the delivery of early-warning notifications when water quality deteriorates, as well as periodic notifications on water-quality conditions. It also provides a cloud-based interface for

visualising current measurements, historical trends, and basic status indicators. Taken together, these components form an integrated workflow in which sensor data are reliably transmitted, analysed to derive water-quality assessments, and communicated to end users in an actionable form. This integration enables the triplet to support timely and informed decisions on irrigation water management.

6.6.1.1 Sequence Diagram

The following sequence of Fig 132 describes the target interaction planned:

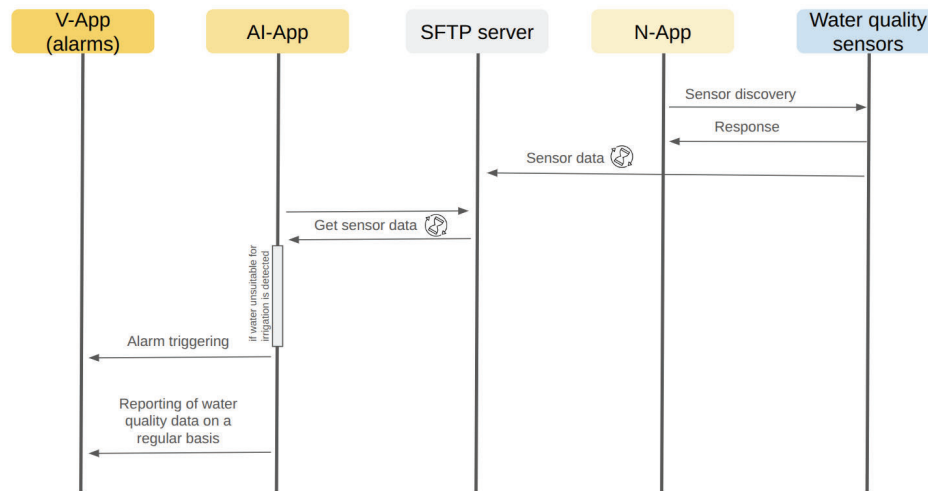


Figure 132 DSS for water quality sequence diagram

6.6.1.2 Communication mechanisms

The following communication mechanisms are considered in the current and planned integration:

- N-App ↔ Water quality sensors: the N-App performs sensor discovery through a REST-based interface to identify available water quality sensors and receive their capabilities. Once discovered, the sensors provide continuous or periodic water quality data to the N-App.
- N-App → SFTP server: the N-App transfers the collected sensor data to the SFTP server using file-based data exchange.
- AI-App ↔ SFTP server: the AI-App retrieves sensor data from the SFTP server via file transfer for subsequent processing and analysis.
- AI-App (analytics and decision-making): the AI-App processes the retrieved sensor data to assess water quality conditions and support decision-making.
- AI-App → V-App: the AI-App sends alarms and periodic reports to the V-App through a REST-based interface, enabling visualization, monitoring, and user notification.

6.6.1.2.1 API-based communication

The explicit V-App with N-App interface is still under definition. The table below summarizes the planned API-based communication for Phase 2.

REST Method	Resource URI	Body Parameter	Description
POST	analyze/water_quality/from_file	Water quality data (file)	Analyses the data from the water quality sensors and returns the results from the model

POST	Send_message/	Recipients, body message, file (optional)	Sends an email to the recipients (whether an alert or an informational message), with the body and file (optional) received as parameters.
------	---------------	---	--

6.6.1.3 Results

At the current stage, no results are yet available regarding the integration between the applications. The end-to-end workflow involving V-App ↔ AI-App ↔ N-App is scheduled for implementation in Phase 2, and results will be documented and reported once the integration has been completed and validated.

6.6.2 Activity 2: Unmanned Vehicle control V-App and Multi-connectivity protocol N-App

The interaction concerns the Unmanned Vehicle control V-App and the Multi-connectivity protocol N-App, since no AI-App is foreseen for this activity, together with OpenCAPIF and the HPE 5GC API as external supporting systems. The V-App is responsible for the remote control of the unmanned vehicle and its payloads, while the N-App is responsible for multi-path management. The envisaged interaction is that the V-App sends service-level requests to the N-App, for example to request enhanced treatment for remote control traffic. The N-App then applies the corresponding multi-connectivity decisions and, in the next implementation stage, invokes the relevant HPE 5GC APIs through OpenCAPIF. At present, both Apps have been demonstrated separately in the trial environment, while the explicit V-App ↔ N-App interface is planned for Trial 2 (Phase 1).

6.6.2.1 Sequence Diagram

The following sequence of Figure 133 describes the target interaction planned for Trial 2 (Phase 1).

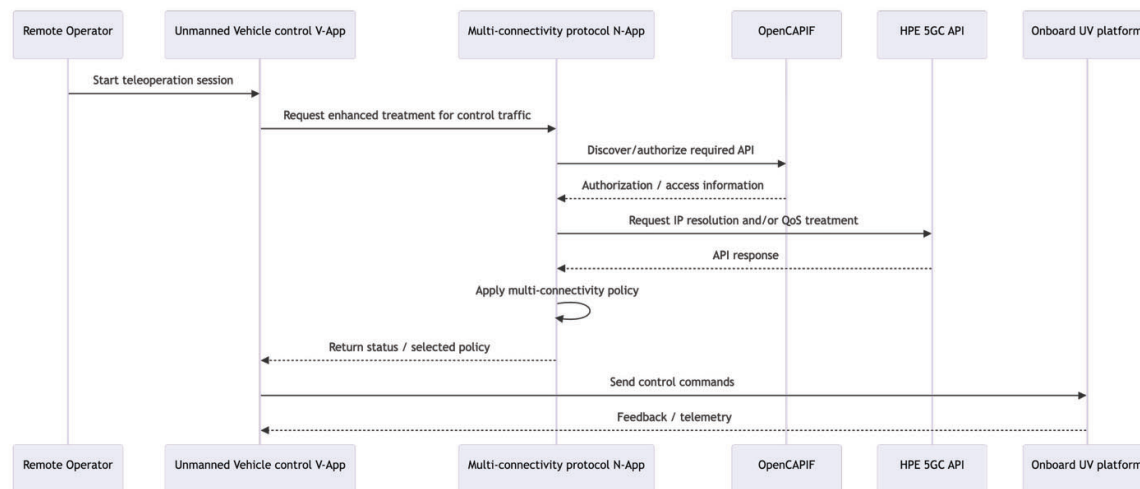


Figure 133. Unmanned Vehicle control V-App and the Multi-connectivity protocol N-App sequence diagram

6.6.2.2 Communication mechanisms

The following communication mechanisms are considered in the current and planned integration:

- V-App ↔ N-App: planned REST-based interface for service-level requests, such as bandwidth limiting or selecting specific 5QI for remote control traffic.

- N-App ↔ OpenCAPIF ↔ HPE 5GC API: planned API-based interaction, where the N-App acts as API invoker and accesses the selected 5GC capabilities through OpenCAPIF.
- Remote teleoperation: current V-App communication over 5G, using MAVLink channels in the drone-based implementation used in Trial 1 (Phase 1).
- Backup control: alternative control through the UV remote controller.
- Payload control: communication between the onboard PC or Raspberry Pi and the robotic arm through USB serial.
- Multi-connectivity management: internal protocol-specific communication (MTIP protocol) inside the N-App for path selection and path management.

6.6.2.2.1 API-based communication

The explicit V-App with N-App interface is still under definition. The table below summarizes the planned API-based communication for Trial 2. Note that the exact API definition will be finalized during the Trial 2 implementation.

Table 40 APIs utilized for Triplet

REST Method	Resource URI	Body Parameter	Description
POST	/remotecontrol/sessions	sessionId, vehicleId, trafficType	Start a remote control session and notify the N-App.
POST	/remotecontrol/sessions/{sessionId}/5qi	trafficType, requested5qi	Request enhanced treatment for unmanned vehicle or robotic arm control traffic.
GET	/remotecontrol/sessions/{sessionId}/status	None	Retrieve the current session and policy status.
DELETE	/remotecontrol/sessions/{sessionId}	None	Terminate the session and release the applied treatment.

6.6.2.3 Results

At the current stage, no results are available for the integration between Apps, since the V-App and the N-App have only been demonstrated separately during Trial 1 (Phase 1), and no AI-App is foreseen for this activity. The current results therefore relate only to the individual validation of each App in its corresponding environment. The integrated V-App ↔ N-App workflow is planned for Trial 2 (Phase 1), and the corresponding results will be reported once available.

6.6.3 Activity 3: Immersive driving of an AGV with autonomous missions

The interaction between components is driven by the V-App, which dynamically retrieves the AGV's IP address from the N-App using the IMSI, allowing automatic network configuration at runtime. Once connectivity is established, the V-App maintains real-time bidirectional communication with the AGV for control and video streaming. Work for the integration with the AI-App is planned to the phase 2. This integration ensures seamless interoperability between network-layer capabilities and application-layer services, enabling low-latency communication, dynamic adaptability, and enhanced operational intelligence within the immersive remote driving system.

6.6.3.1 Sequence Diagram

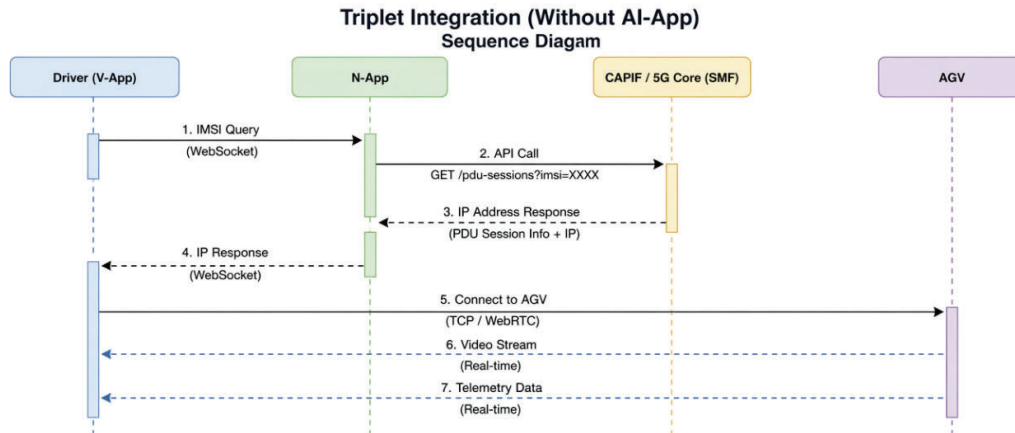


Figure 134 The Triplet integration in the Spanish use case

6.6.3.2 Communication mechanisms

The Triplet integration relies on a combination of real-time and asynchronous communication mechanisms optimized for low latency 5G environments:

- **WebSocket**
 - Used between V-App and N-App
 - Enables persistent, bidirectional, low-latency communication
 - Suitable for real-time IMSI-to-IP queries
- **REST APIs (HTTP)**
 - Used between N-App and 5G Core (SMF via CAPIF)
 - Provides access to PDU session and addressing information
- **TCP / WebRTC (or equivalent streaming protocol)**
 - Used between V-App and AGV
 - Supports real-time control commands and video streaming
- **HTTP / File-based ingestion**
 - Used by AI-App to retrieve images from camera endpoints
- **Internal file system + daemon processing**
 - Used inside AI-App for decoupling ingestion and inference
- **Database communication (InfluxDB)**
 - Used by AI-App to store detection results
 - Used by V-App (indirectly) for monitoring via Grafana

All communications are designed to ensure low latency, high availability, and scalability.

6.6.3.2.1 API-based communication

Table 41 N-App - 5G Core (SMF API)

REST Method	Resource URI	Body Parameter	Description
GET	/pdu-sessions	IMSI (query)	Retrieves PDU session information and IP address associated with a given IMSI

Table 42 V-App - N-App (WebSocket API)

Method	Endpoint	Payload	Description
WebSocket	/ws	{ "imsi": "string" }	Requests IP address associated with IMSI
WebSocket	/ws	{ "ip": "string" }	Response containing AGV IP address

6.6.3.2.2 Other communication methods

Video Streaming (AGV → V-App)

- Protocol: TCP / WebRTC (implementation-dependent)
- Data format: Video stream (e.g., H.264 encoded)
- Characteristics:
 - Real-time transmission
 - Multiple camera feeds
 - Low-latency requirement

Frame Transfer (Camera → AI-App)

- Method: HTTP GET (camera) or file-based transfer
- Example request: GET /capture
- Output: Image (JPEG/PNG)

AI Detection Output (AI-App → Database)

- Format: JSON
- Example payload:


```
{
  "timestamp": "2026-04-17T10:15:30Z",
  "object_detected": "person",
  "confidence": 0.92,
  "bounding_box": {
    "x": 120,
    "y": 80,
    "width": 60,
    "height": 140
  }
}
```
- Storage: InfluxDB
- Usage: Visualization and analytics in Grafana

Metrics Monitoring (V-App → Grafana via DB)

- Metrics collected:
 - Latency (RTT)
 - Control input rates
 - System performance indicators
- Flow: V-App → Metrics → Database → Grafana Dashboard

6.6.3.3 Results

The integrated Triplet (N-App, V-App, AI-App) planned for the 1st phase has been successfully deployed and validated within the 5G testbed at La Mayora.

Key Results

- Seamless Integration
 - The V-App dynamically retrieves AGV IP addresses via the N-App with no manual configuration.
 - CAPIF-based communication with the 5G Core is fully operational.

- Low-Latency Performance
 - Real-time control and video streaming achieved with stable latency.
 - WebSocket communication ensures immediate response for network queries.
- Reliable Video and Control
 - Continuous multi-camera streaming from AGV to driver.
 - Stable bidirectional control loop.
- AI-Based Analytics
 - AI-App successfully performs real-time object detection.
 - Detection results are stored and visualized effectively.
- Monitoring and Observability
 - Metrics and AI outputs are available in Grafana dashboards.
 - RTT measurements confirm system responsiveness.

Validation Evidence

- Logs confirming IMSI-to-IP resolution via N-App
- Successful API responses from SMF (/pdu-sessions)
- Stable WebSocket communication sessions
- Continuous video streaming and control logs
- AI detection outputs with bounding boxes
- Stored entries in InfluxDB
- Grafana dashboards showing real-time metrics

The system demonstrates a robust, scalable, and low-latency integration of network intelligence, immersive control, and AI analytics, validating the feasibility of the Triplet architecture for advanced 5G use cases. Figure 23 and Figure 64 show the logs of a successful interaction between N-App and V-App to obtain the AGV IP address through the IMSI number.

6.6.4 Activity 4: Automatic Surveillance & Detection in Farming Operation

This section describes the end-to-end integration of the **V-App / N-App / AI-App triplet** supporting the *Automatic Surveillance & Detection in Farming Operation* activity of the Spanish cluster. The description follows the same structure and level of detail adopted for the other Spanish activities in order to ensure consistency and homogeneity across Section 6.

The objective of this triplet is to enable network-aware radio sensing and surveillance in a farming environment, where UEs reporting SRS information are identified, monitored, and supported with appropriate network treatment to ensure reliable data delivery for detection and decision-making purposes.

6.6.4.1 Sequence Diagram

The operational sequence of the triplet is summarized as follows:

1. **SRS reporting and surveillance trigger:** UEs deployed in the farming area periodically transmit SRS information as part of the radio sensing workflow. These reports are associated with specific UEs participating in the surveillance activity.
2. **V-App request to N-App:** The *Automatic Surveillance & Detection in Farming Operation V-App* requests network assistance from the N-App in order to identify the UEs involved in the sensing workflow and to ensure suitable network conditions for their traffic.
3. **UE identification via 5G Core:** The N-App interacts with the 5G Core (HPE 5GC) to retrieve network-related information, in particular resolving the IP address associated with a given IMSI. This step enables the V-App to correctly associate sensing data with the corresponding UE.
4. **QoS request and prioritization:** Based on the surveillance requirements, the N-App requests differentiated network treatment (e.g., specific 5QI or higher priority) for the UEs reporting SRS information, ensuring adequate reliability and performance.

5. **Data forwarding to AI-App:** The V-App forwards the relevant sensing and contextual data to the AI-App, which performs detection and analysis functions related to the farming surveillance use case.
6. **Inference and feedback:** The AI-App processes the received data and returns inference results (e.g., detection outcomes or alerts) to the V-App for visualization and operational use.

This sequence enables a closed-loop interaction where network awareness, AI-based detection, and vertical application logic are combined into a single operational workflow.

6.6.4.2 Communication mechanisms

The triplet integration relies on the following communication mechanisms:

- **V-App ↔ N-App:** REST-based northbound APIs are used for requesting UE identification and differentiated network treatment. This interface allows the V-App to remain decoupled from low-level network details while still benefiting from network programmability.
- **N-App ↔ 5G Core:** The N-App interacts with the HPE 5GC through implementation-specific northbound APIs to retrieve UE-related information (e.g., IMSI-to-IP resolution) and to request QoS or policy adaptations. This interaction represents the southbound interface of the N-App.
- **V-App ↔ AI-App:** The V-App communicates with the AI-App through application-level APIs for data delivery and inference result retrieval. This interface supports the exchange of sensing data and detection outputs required by the surveillance logic.

The integration is aligned with the overall 6G-VERSUS triplet architecture, where the N-App acts as the mediation layer between the vertical logic and the network, while the AI-App provides intelligence and decision support.

6.6.4.3 Results

The triplet integration for the *Automatic Surveillance & Detection in Farming Operation* activity has been successfully validated in an emulated setup during Phase 1 trials. The main results can be summarized as follows:

- Successful **end-to-end interaction** between V-App, N-App, and AI-App, demonstrating the feasibility of the triplet-based approach for farming surveillance scenarios.
- Correct **UE identification and association**, achieved through IMSI-based queries to the 5G Core, enabling the V-App to accurately map sensing data to the corresponding devices.
- Effective **differentiated network treatment** for UEs reporting SRS information, confirming that the N-App can request and apply suitable QoS configurations to support surveillance-related traffic.
- Functional **AI-based detection workflow**, where sensing data provided by the V-App is processed by the AI-App and the resulting inference outputs are returned to the V-App for operational use.

These results confirm that the integrated triplet can support network-aware radio sensing and surveillance in farming environments. Further extensions and refinements of this activity, including deployment in the target testbed, are planned for Phase 2 and will be reported in D4.2.

6.6.5 Activity 5: DSS for reliable water treatment

In the DSS for reliable water treatment triplet, the AI-App acts as the core intelligence component, the N-App as the adaptive connectivity management component, and the V-App as the user-facing notification and visualisation component. The AI-App receives and analyses heterogeneous plant monitoring data, including water-quality sensor measurements and image-based information from camera systems. Based on this analysis, it detects anomalies, assesses process performance, and determines whether abnormal conditions require operator attention. When such conditions are identified, the AI-App sends notification triggers and alert context to the V-App, which is responsible for delivering early-warning notifications to plant staff and presenting relevant information through a simple visual interface. In parallel, the AI-App also issues connectivity-related

requirements to the N-App whenever the monitoring situation requires changes in communication performance. These requests may involve, for example, increasing available bandwidth, supporting higher image resolution, increasing capture frequency, or prioritising critical telemetry. The N-App translates these requirements into network-level actions by applying the corresponding connectivity and Quality of Service (QoS) configurations, thereby adapting communication conditions to the needs of the monitoring process. The V-App therefore provides the operational interface towards the end users, while the N-App provides the communication support required to sustain the sensing and monitoring capabilities of the system. Together, these interactions enable the triplet to combine intelligent analysis, adaptive connectivity, and timely operator notification within a coordinated decision-support framework for reliable water treatment.

6.6.5.1 Sequence Diagram

The following sequence of Figure 135 describes the target interaction planned.

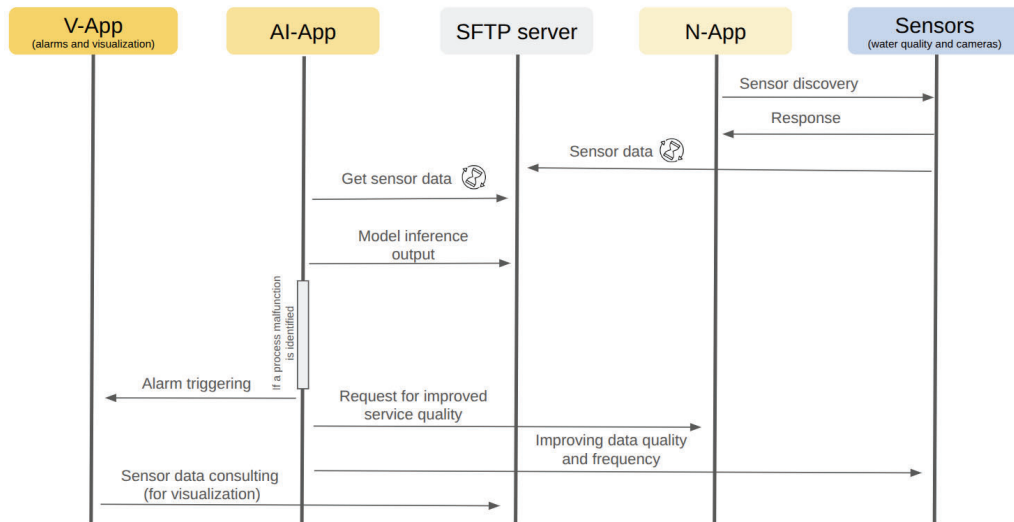


Figure 135 DSS for reliable water treatment sequence diagram

6.6.5.2 Communication mechanisms

The following communication mechanisms are considered in the current and planned integration:

- N-App ↔ Sensors (water quality and cameras): the N-App performs sensor discovery to identify available devices, and then the N-App provides network connectivity to the sensors.
- Sensors → SFTP server: Sensors send data to the SFTP server, which acts as a centralized storage layer for subsequent access and processing (REST-based interface).
- AI-App ↔ SFTP server: the AI-App retrieves data from the SFTP server, and the SFTP server responds with the corresponding data for analysis.
- AI-App → SFTP server: after processing, the AI-App stores the model inference output back in the SFTP server, making results available for other components.
- AI-App → V-App: if a process malfunction is identified, the AI-App triggers an alarm toward the V-App for visualization and user notification (REST-based interface).
- V-App ↔ SFTP server: the V-App performs sensor data consulting by accessing stored data (and potentially inference results) from the SFTP server for visualization purposes.
- AI-App → N-App: when needed, the AI-App issues a request for improved service quality through Open Gateway (e.g., higher sampling rate or higher image resolution, using REST-based interface).
- AI-App → Sensors: the AI-App enforces improvements in data quality and frequency by reconfiguring the sensors accordingly (REST-based interface).

6.6.5.2.1 API-based communication

REST Method	Resource URI	Body Parameter	Description
POST	analyze/image/from_file	Image (file)	Analizes image through computer vision and returns the output of the model
POST	analyze/water_quality/from_file	Water quality data (file)	Analyses the data from the water quality sensors and returns the results from the model
POST	Send_message/	Recipients, body message, file (optional)	Sends an email to the recipients (whether an alert or an informational message), with the body and file (optional) received as parameters.

6.6.5.3 Results

At the current stage, no results are yet available regarding the integration between the applications. The end-to-end workflow involving V-App ↔ AI-App ↔ N-App is scheduled for implementation in Phase 2, and results will be documented and reported once the integration has been completed and validated.

7 Conclusion

This deliverable presented the intermediate development status of the Network Applications (N-Apps), Vertical Applications (V-Apps), and AI Applications (AI-Apps) developed within the 6G-VERSUS project. It reported the progress achieved by the project partners across all clusters in the specification, implementation, deployment, and initial integration of the application components forming the 6G-VERSUS application triplets. Furthermore, the deliverable documented the current maturity of the developed software components, including available implementation references, deployment approaches, and demonstration results obtained during the initial development and validation activities.

As an initial/intermediate version of the WP4 development reporting, this deliverable reflects the current implementation status of the project's software components at this stage of the project lifecycle. The reported developments mainly correspond to the Phase 1 trial activities (Trials 1 and 2), while also introducing application components and functionalities currently under development for Phase 2 (Trials 3 and 4). The reported prototypes constitute the baseline versions that will be further refined, extended, and validated during the upcoming integration and trial phases.

Building upon the developments presented, the next project phases will focus on advancing the maturity of the application components, completing the remaining developments, and performing their large-scale validation within the 6G-VERSUS experimentation environments. The final and fully mature versions of the developed applications, together with the complete reporting of their implementation, integration, and validation, will be presented in Deliverable D4.2 at the end of the project.

8 Bibliography

- [1] 3GPP, "TS 29.522: 5G System; Network Exposure Function (NEF) Northbound APIs; Stage 3," [Online]. Available: <https://www.3gpp.org/dynareport/29522.htm>.
- [2] 3GPP, "TS 29.520: 5G System; Network Data Analytics Services; Stage 3," [Online]. Available: <https://www.3gpp.org/dynareport/29520.htm>.

- [3] 3GPP, "TS 23.286: Application layer support for Vehicle-to-Everything (V2X) services; Functional architecture and information flows," [Online]. Available: <https://www.3gpp.org/dynareport/23286.htm>.
- [4] 3GPP, "SA6 accelerates work on new verticals," June 2019. [Online]. Available: <https://www.3gpp.org/news-events/3gpp-news/sa6-verticals>.
- [5] 3GPP, "Network Automation Enablers in 5GS," May 2025. [Online]. Available: <https://www.3gpp.org/technologies/nae-5gs-ct3>.
- [6] 3GPP, "TS 33.122: Security aspects of Common API Framework (CAPIF) for 3GPP northbound APIs," [Online]. Available: <https://www.3gpp.org/dynareport/33122.htm>.
- [7] "5G PPP Software Networks WG, "Network Applications: Opening up 5G and beyond networks," Vols. <https://5g-ppp.eu/wp-content/uploads/2022/10/Software-Network-WG-Network-Applications-2022.pdf>, 2022.
- [8] "5G PPP Software Networks WG, "Network Applications: Opening up 5G and Beyond networks," July 2023. [Online]. Available: <https://zenodo.org/records/8134429>.
- [9] 5. P. A. WG, "The 6G Architecture Landscape – European perspective, White Paper," December 2022. [Online]. Available: https://5g-ppp.eu/wp-content/uploads/2022/12/6G-Arch-Whitepaper_v1.0-final.pdf.