6GSNS

Co-funded by the European Union

SAFE-6G

SAFE-6G

A Smart and Adaptive Framework for Enhancing Trust in 6G Networks

Deliverable D3.6: Metaverse use-cases development and user-centric configuration

Date: 30/06/2026

Version: V1.0

DISCLAIMER

Grant Agreement	101139031
Document number	D3.6
Document title	Metaverse use-cases development and user-centric configuration
Lead Beneficiary	NCSRD
Editor(s)	Charles Bailly (IMM) Alejandro Fornés (UPV)
Author(s)	Charles Bailly (IMM) Julien Castet (IMM) Vagelis Anagnostopoulos (INF) Panagiotis Koumaras (INF) Georgios Koumaras (INF)
Dissemination level	Public
Contractual date of delivery	30/06/2026
Status	Final
File name	SAFE-6G_D3.6_V1.0.pdf

Revision History

Version	
V0.1	TOC proposal, guidelines and templates
V0.2	TOC refinement and first round of contributions
V0.3	Second round of contributions, including new architectural figures
V0.9	Internal review performed (8BELLS, NCSRD), sent to final edition
V1.0	Final version ready to submission

GLOSSARY

Abbreviations/Acronym	Description
API	Application Programming Interface
CAPIF	Common API Framework
CoCo	Cognitive Coordinator
CPU / vCPU	Central Processing Unit / Virtual CPU
DT	Digital Twin
FSM	Finite-State-Machine
GPU	Graphics Processing Unit
GUI	Graphical User Interface
LLM	Large Language Model
LoTw	Level of Trustworthiness
MLOps	Machine Learning Operations
MSISDN	Mobile Station International Subscriber Directory Number
NLP	Natural Language Processing
RAM	Random Access Memory
RPC(s)	Remote Procedure Call(s)
TF	Trust Function
UC	Use Case
UDP	User Datagram Protocol
WP	Work Package
XAI	eXplainable AI
XR	Extended Reality

EXECUTIVE SUMMARY

This document belongs to the second round of WP3 deliverables. In contrast to the first round where one single document contained the report of all WP tasks, now each of them has produced their own deliverable. Specifically, D3.6: *Metaverse use-cases development and user-centric configuration*, reports the final outcomes (mostly software) of Task 3.5.

This Metaverse ecosystem comprises the implementation of two major types of components. First, the Unity use-case applications, namely the factory digital twin for UC1 and the XR formation in UC2. And second, the Metaverse manager, which makes the link between the Unity apps and the rest of the SAFE-6G architecture. Besides, this document also presents the final development and integration efforts related to the SAFE-6G Chatbot.

All these components have been iterated on during the project and reached their final versions. They are thus ready for the verification and validation activities being performed in WP5.

KEYWORDS

Metaverse, Virtual Worlds, User interfaces, LLM, Chatbot, User Intent, CAPIF.

TABLE OF CONTENTS

List of FIGURES

List of TABLES

1 INTRODUCTION

Figure 1 highlights in green the building blocks of the SAFE-6G architecture implemented in WP3. The final implementation of their components is reported in the concurrent deliverables from the Work Package. Particularly, this one focuses on two modules: the **AI-powered Chatbot** of the SAFE-6G framework, which client is embeddable in different user interfaces (XR headset, web browser, etc.), and acts as the users' entrypoint for the features offered by the framework; and the **Metaverse ecosystem** that serve as use cases for testing and validating the SAFE-6G framework. The modules addressed in this deliverable are stressed in yellow. The open-source code implemented related to them is available at the following repositories: https://gitlab.com/safe-6g/development/chatbot_api and https://gitlab.com/safe-6g/pilots/metaverse-manager. This deliverable is complimenting the documentation that it is available on those repositories, without repeating the technical instructions needed for developing and deploying their components. The scope of the deliverable is rather to summarise a high-level overview of their technical implementation, delving a bit deeper into details related to the use-case Unity-based applications, which code will be commercial.

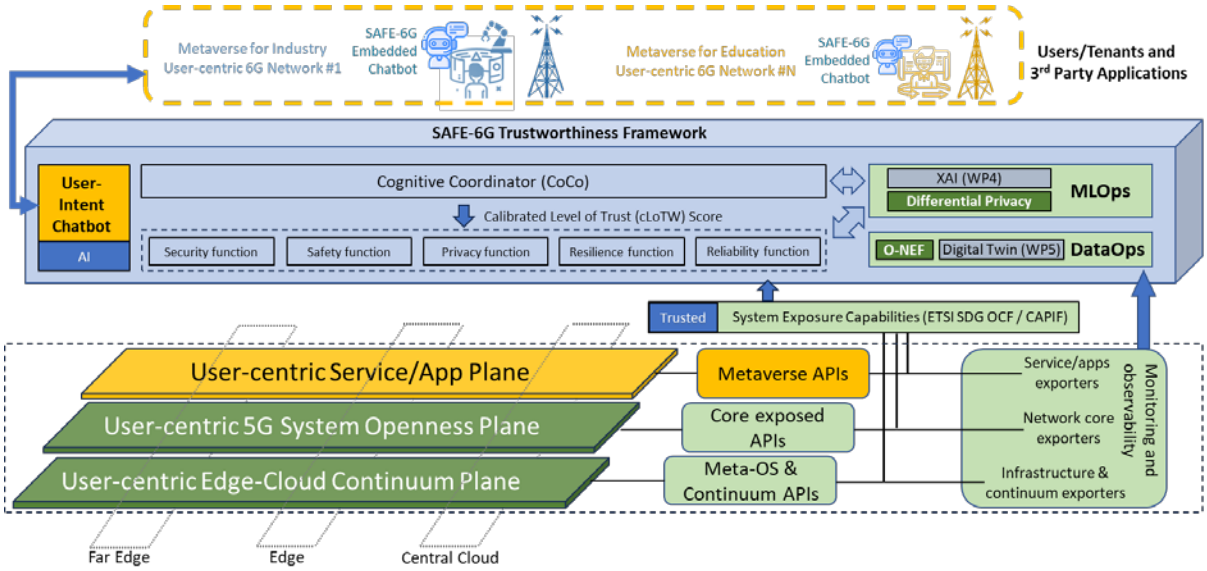


Figure 1: Building blocks and components of the SAFE-6G architecture

1.1 THE RATIONALE BEHIND THE STRUCTURE

The structure of this deliverable is as follows. After this introduction, Section 2 presents the development and integration efforts related to the Unity Use Case (UC) applications. Then, Section 3 describes the final version of the Metaverse manager component, which serves a set of endpoints for managing specific aspects of the aforementioned use cases. Finally, section 4 does the same for the SAFE-6G Chatbot, the main interface of the SAFE-6G framework.

2 UNITY USE-CASE APPLICATIONS

As presented in D3.1, the Metaverse ecosystem is divided into two categories: 1) Unity use-case applications and 2) the Metaverse manager. **Three major iterations with different technical approaches and setups were conducted during the project** on both categories to converge to the final versions presented in this report. As shown in the figure below, T3.5 also included the development and integration of the SAFE-6G Chatbot.

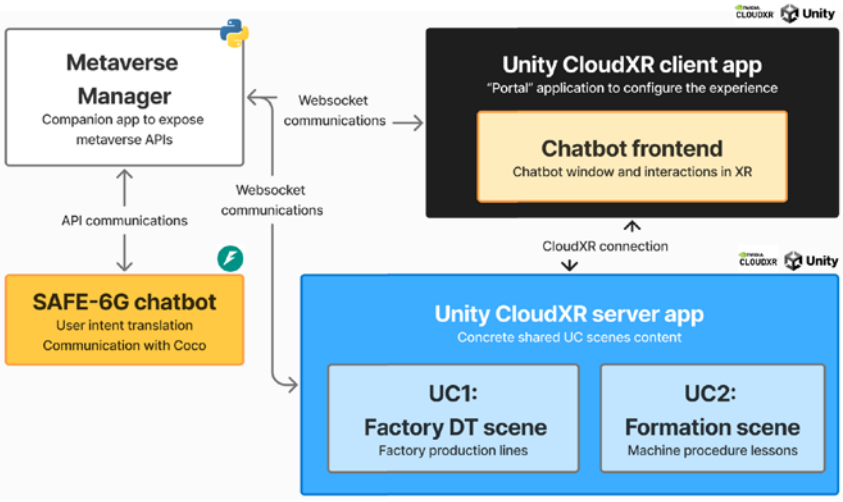


Figure 2: Overview of the components developed and integrated during T3.5

The Metaverse use-case applications are the XR applications end-users focus on, i.e., the **factory digital twin for UC1 and the XR formation in UC2**. The applications were developed with the Unity framework (in C#) to be able to run on XR headsets and be compatible with equipment such as haptic gloves and haptic suits. They are also based on specific NVIDIA modules to render the application on a workstation and send the resulting video stream to the XR headset of the user.

One of the major evolutions of the final release compared to the initially envisioned design concerns CloudXR[1]. CloudXR is a GPU-accelerated streaming platform from NVIDIA. It allows to run an XR application on a dedicated workstation, streaming the result to a target XR headset connected as client. This way, autonomous (i.e., non-tethered) headsets can easily display heavy applications since all the rendering is offloaded to another workstation.

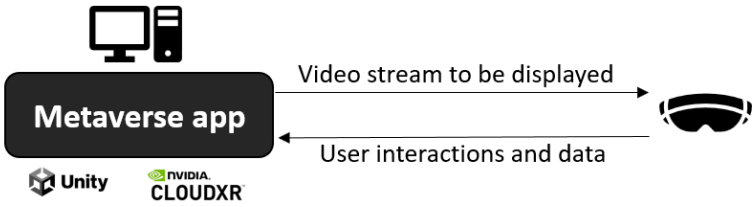


Figure 3: Illustration of the main concept of CloudXR. Running an XR application on a workstation and streaming the resulting frames to the target client headset

During the initial design steps of the project, CloudXR was only envisioned for UC1. This was motivated by the fact that the factory Digital Twin scene was too computationally demanding to be rendered directly on the headset. However, design iterations revealed that opening the CloudXR connection

[1] NVIDIA CloudXR documentation: https://developer.nvidia.com/topics/ai/xr/cloudxr-sdk

right from the start was not the optimal solution. Instead, it was more relevant for TFs to first configure the network to match the user intent before granting access to the streamed UC content through CloudXR. It also opened more possibilities related to the cloud-continuum positioning of Metaverse components and motivated a different split of the Unity applications. Instead of having one Unity application per use-case (as initially envisioned), there is in the final release a CloudXR client and a CloudXR server.

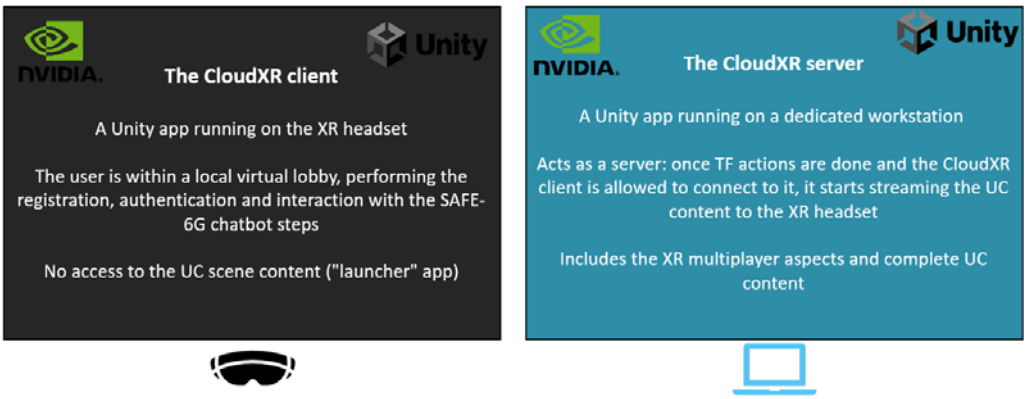


Figure 4: Overview of the split between CloudXR client and CloudXR server

First, the **CloudXR client** application directly runs on the end-user XR headset. It allows users to be in a local lobby and perform preliminary steps like registering themselves and interacting with the SAFE-6G Chatbot to express their needs. Second, a **CloudXR server** application runs the UC scenes (factory DT or machine formation) in another workstation. After the XR user ends the conversation with the Chatbot and the network reconfigured itself to match their requests, the connection with the CloudXR server opens. The XR headset starts receiving the video stream from the CloudXR server, allowing the user to access the shared XR UC scenes for UC1 or UC2.

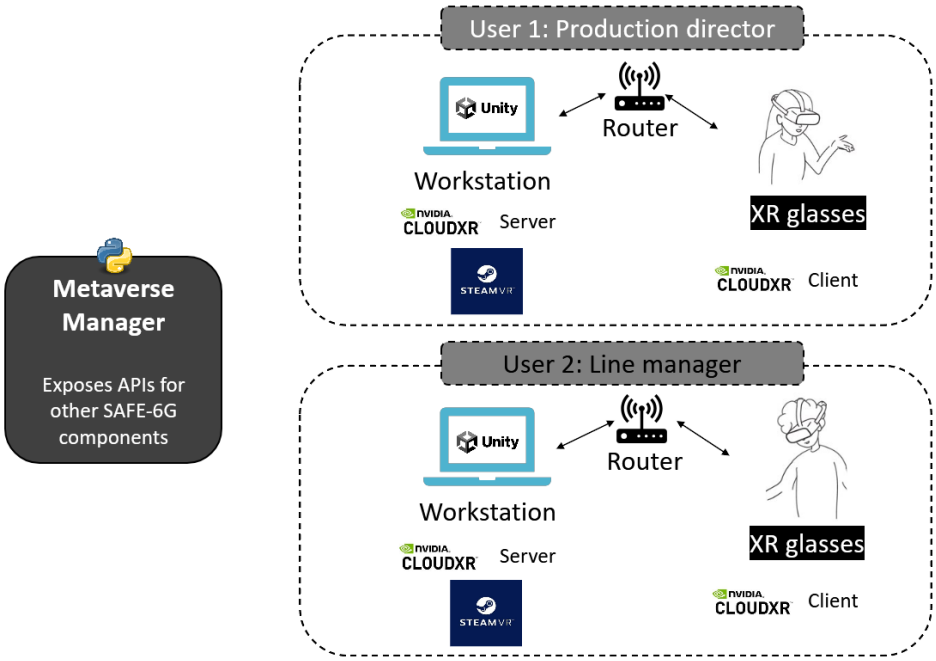


Figure 5: Example of CloudXR setup for UC1. The XR glasses/headsets cannot connect directly to the 5G network. Instead, they need to be connected to a router acting as Wi-Fi hotspot. The Metaverse manager can be deployed elsewhere in the network to be able to communicate with both the workstations and the XR equipment

2.1 CLOUDXR CLIENT AND SERVER APPLICATIONS

Several requirements, needs and desirable features were identified from end-users and reported in D2.3. The iterative development and integration of Metaverse components was guided by them, with specific focus on:

- **REQ-USER-Both-F-M-1**: Identity assessment of users before granting them access to the UC content (registration and authentication steps before accessing the Chatbot).
- **REQ-USER-Both-NF-R-1:** Easy connection to the SAFE-6G network (simple few steps to be performed in a local XR lobby before accessing UC content).
- **REQ-USER-Both-NF-M-2**: Security and trustworthiness measures not disturbing significantly the work of end-users (quick access to UC content, all initial steps performed within the lobby).
- **REQ-USER-CATEGORY-Both-NF-O-4**: Understandable SAFE-6G system decisions for novice end-users (high-level answers from Chatbot, XAI).
- **REQ-USER-UC1-F-R-2**: Role-based access to factory data (Differentiated Netcode capabilities based on role, role manager component in both Unity apps + Metaverse manager end-user management).
- **REQ-USER-UC2-F-R-1**: Letting instructors punctually adjust the network configuration (Chatbot remains available after the initial requests).
- **REQ-USER-UC2-F-M-4**: Continuous authentication of technician users (automatic disconnection can be triggered when the user removes the headset to prevent passing it to someone else).

Based on the listed requirements and the natural evolution of the project implementation, the **features included in the final release (Rel-C) for the Unity CloudXR client** includes:

- A virtual lobby where the user can learn more about SAFE-6G and the TFs (see Figure 6).

Figure 6: Capture of the local lobby from the Unity CloudXR client app. At this step, the user can hover each planet to learn more about each of the SAFE-6G TF

- Connection to the Metaverse manager and initial user registration.
- Interaction with the Chatbot frontend in XR, using either the virtual keyboard or through voice commands. The user requests are transmitted to the SAFE-6G Chatbot backend through the Metaverse manager.
- Launch of the connection with the CloudXR server.

The features included in the final release **(Rel-C) for the Unity CloudXR server application** include the role management components, the multiplayer components and the two shared UC scenes:

- **Role manager components**. A set of C# components present in both UC scenes. Their role is to enforce data and interaction policies based on user roles. Users can only visualize and interact with XR content (3D objects, GUIs, XR controller interactions…) which corresponds to their role while the rest remains hidden and not accessible. This applies both to local content (i.e. content already present in the application) and networked interactions (for instance, trying to trigger a factory DT update for UC1).
- **Multiplayer components.** The CloudXR server app includes custom Netcode[2] scripts to handle the multiplayer features for shared XR scenes. It regroups the multiplayer session discovery services, user management (connection and disconnections, for instance after blacklisting a user) and scene synchronizations through Netcode Remote Procedure Calls (RPCs).
- **Factory DT scene for UC1**. The scene simulates the digital twin of a truck factory with two production lines. One of the production lines is stopped because issues were detected on the robotic arms present on it. Production director and Line manager users can collaborate in XR to investigate and solve the issues in the production line, agreeing on the pieces to replace on the different machines. When a solution was defined, they update the factory DT, which would then trigger the change request on the real factory.

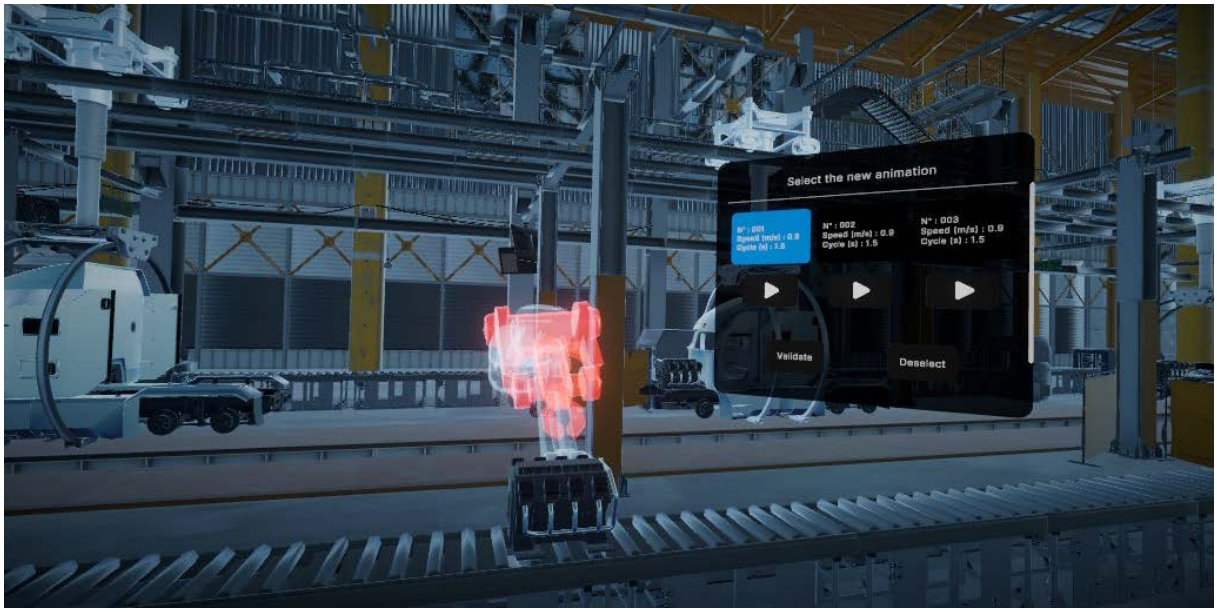


Figure 7: Capture from the factory DT scene from UC1 where the user can visualize a faulty piece

[2] Netcode documentation:
https://docs.unity3d.com/Packages/com.unity.netcode.gameobjects@2.7/manual/index.html

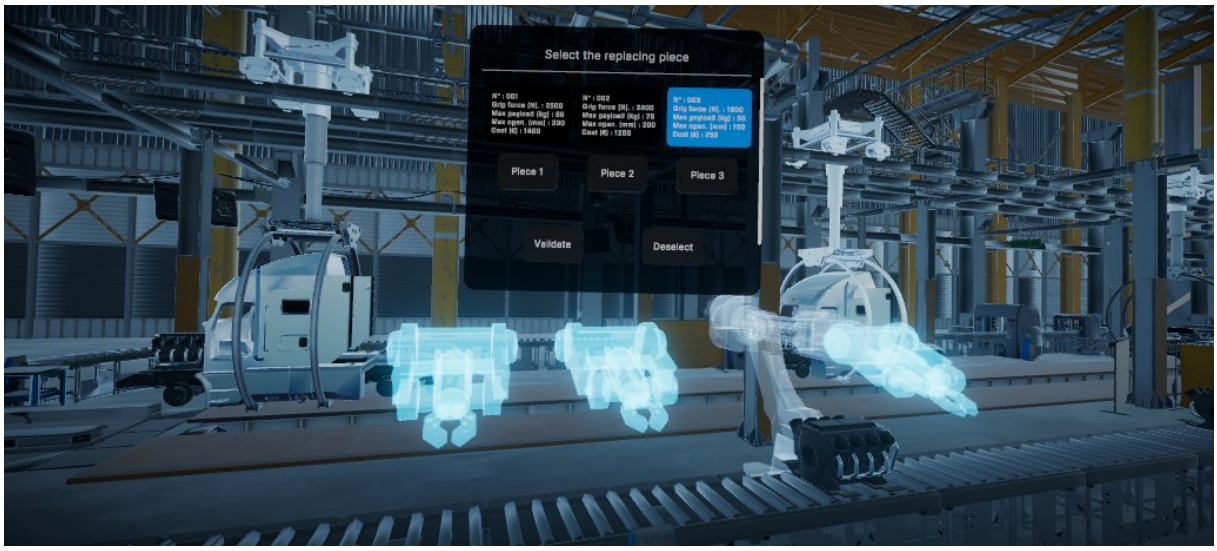


Figure 8: Capture from the factory DT scene from UC1 where the user can choose replacements for faulty pieces and define new behaviours to fix the production line

To represent the autonomous and persistent factory Digital Twin, a first instance of the application runs as Netcode host. This instance represents the "ground truth" of the factory state, called hereafter the *Simulated Twin*. Once allowed to access UC content, Production director and Line manager users will thus connect to the factory shared scenes as Netcode clients. This way, both types of users can collaborate in real time and synchronize scene content through Netcode RPCs, but the Simulated Twin keeps control over the scene synchronization and factory updates. For instance, it allows to run additional checks upon receiving RPCs about sensitive content, as illustrated in Figure 9.

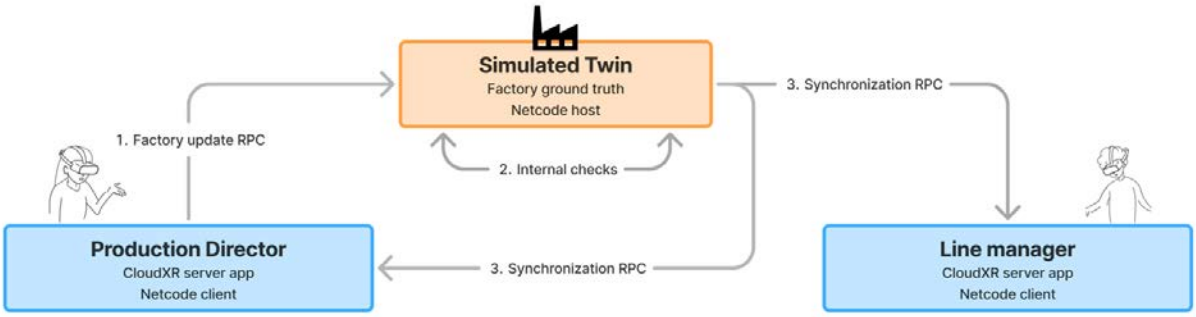


Figure 9: Example of shared scene synchronization through RPCs for UC1

XR formation scene for UC2

Technicians can learn how to operate a virtual industrial machine while Instructors provide assistance when needed. The procedures and the machine model were designed according to the industrial workshop from the mechanical engineering department of the University of Bordeaux.

Figure 10: Capture from the machine formation room from UC2. The Technician user is first selecting one of the courses, then is guided through several steps to learn how to operate the machine

One technical limitation found for this UC2 scene is the incompatibility of the ShariiingXR module[3]. As presented in D2.3, ShariiingXR was envisioned in the initial UC2 architecture to let Technicians share in real time the view of their headset with others. This would have allowed Instructors to easily monitor the progress of several Technicians at the same time to determine when help is required, even without wearing an XR headset. However, connectivity tests performed at NCSRD premises revealed that the underlying Web-RTC library used to manage the video streams was not compatible with a 5G setup. The ShariiingXR module was thus finally not included in UC2.

The Unity applications were developed to target Quest 3 headsets, which are recent XR devices with powerful video-see-through capabilities. They were thus built to target the Android platform. This means that they can quite easily be adapted to run on many other Android-based devices, like Pico headsets for instance. Nonetheless, alternative Windows-based builds were also created along the project for facilitating test purposes. These specific builds allowed partners without access to XR headsets to have a look at the Unity apps content and perform initial tests (connection to the Metaverse manager, interaction with the Chatbot…) at their own discretion instead of depending on hardware availability.

From an architectural point of view, both Unity applications use a Finite-State-Machine (FSM) approach with custom events. This design provides a high level of flexibility and scalability to adapt to potential new use-cases contents.

[3] IMMERSION Shariiing module website: https://www.immersion.fr/en/with-shariiing-xr-your-workspaces-have-never-been-so-immersive/

Several architectural changes have been made during the project to handle different setups and integration levels, for instance for demo events like at the EuCNC conference. With their 3.0 release, the Unity applications successfully reached their final stage for the project.

Category	*Status*
Version	Latest/3.0
Demo	https://www.youtube.com/watch?v=cvDM6dOr5fs
Deployment requirements	Not deployed through the SAFE-6G Meta-OS (not possible for Unity applications as are not Cloud Native). For the CloudXR server application, a Windows machine with a recent NVIDIA graphic card and at least 8 GB of RAM (16Gb recommended) is required. A 5 GHz, 50 + Mbps Wi-Fi band is also needed for using CloudXR[4].
Dependencies	Standalone components, but tightly coupled to the Metaverse manager. Summary of technologies used for their implementation: • Unity [5] (version 2022.3.18, C#) and Netcode (multiplayer management, UDP extension as communication protocol) • CloudXR (version 4.0) and SteamVR[6] (version 2.15.6) • Wit.ai[7] for voice commands (for Quest3 headsets)
License	Commercial (private, non-public)

Table 1: Summary of the status of the version 3.0 of the Unity UC applications

2.2 FINAL WORKFLOW FROM THE END-USER PERSPECTIVE

The final end-user workflow can be divided into four major steps:

1. Starting the Unity CloudXR client and performing the initial configuration steps
2. Interacting with the SAFE-6G Chatbot to refine requirements and user needs
3. Starting the connection to the CloudXR server
4. Working on the target UC scene

Step #1. First, the users start the CloudXR client application on their XR headset. They arrive in a local lobby where they can learn more about each of the Trust Function. They are then invited to select the UC they are interested in (factory DT or formation room) and to start the connection with the Metaverse manager. They can then proceed with the registration step, providing their name, and role to the Metaverse manager. Current XR headsets do not have SIM cards in them and thus cannot provide network-related id. Nonetheless, an Android hardware id is also sent to the Metaverse manager as an additional unique identifier for the given user.

Step #2. After the registration step, the user can start interacting with the Chatbot, using either the virtual keyboard or voice commands. Back and forth messages are exchanged until the Chatbot has a fine understanding of the user intent or until the user decides to end the conversation. The Chatbot will send the translated user intent to the Cognitive Coordinator (CoCo). Behind the scenes, the TFs apply their actions to align with the determined LoTw. Chatbot details are given in Section 4.

[4] CloudXR requirements: https://docs.nvidia.com/cloudxr-sdk/release/4.0/usr_guide/system_requirements. html
[5] Unity webpage: https://unity.com/es
[6] SteamVR webpage: https://store.steampowered.com/steamvr
[7] Wit.ai webpage: https://wit.ai/

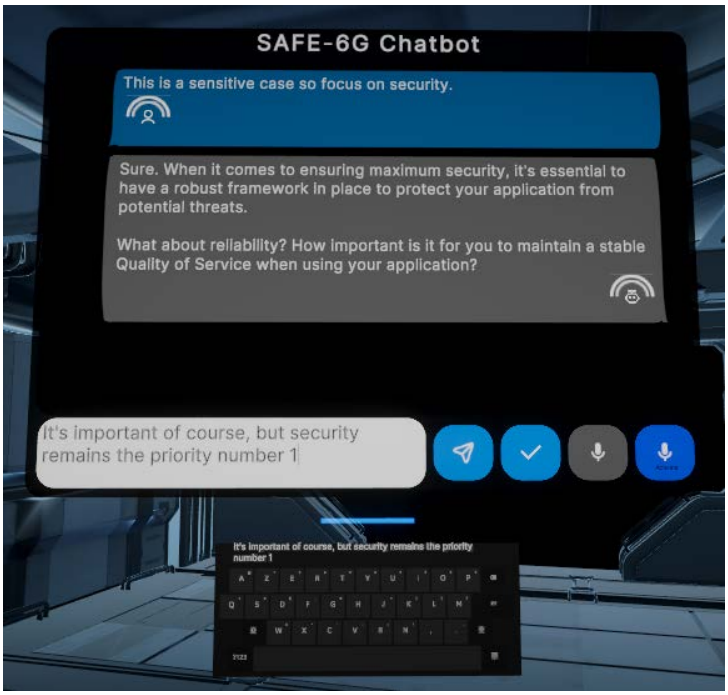


Figure 11: Capture of the Chatbot frontend in XR

Step #3. Once all TF actions are achieved, the user is notified through the Chatbot. The CloudXR server application is now accessible (from the network point of view). The user can connect to it through SteamVR, as shown in Figure 12. SteamVR makes the link between a wireless client headset and any OpenXR application (i.e., the CloudXR server here) running on the workstation.

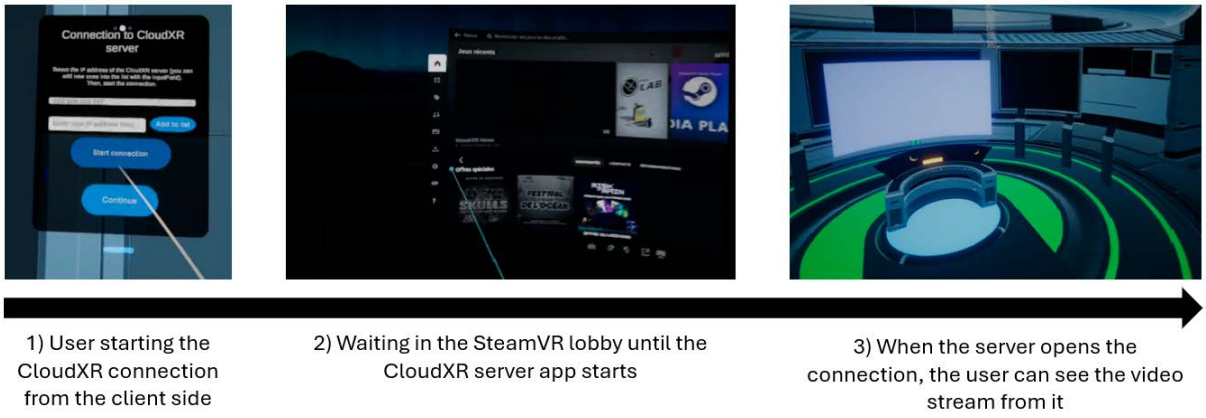


Figure 12: Overview of Step#3: establishing the connection with the CloudXR server application

Step #4. The user is now seeing the streamed frames from the CloudXR server application and can interact with it as if it was running on the headset directly. To reach the UC scene and collaborate in XR with other connected persons, the user simply must connect to the corresponding Netcode lobby. This way, the user will join the shared multiplayer room and automatically synchronize in real time the target UC scene content.

3 METAVERSE MANAGER

The Metaverse manager is a companion application made to expose APIs and facilitate testing phases since the Unity applications cannot be containerized and deployed like other components. It is not involved on the multiplayer aspects of XR UC nor acting as server for it. This is handled by the Unity app directly. The Metaverse manager is a lightweight application based on Python. It communicates directly with the Unity UC apps and makes the link with the rest of the SAFE-6G architecture (in particular, the Chatbot backend). Thanks to its web Graphical User Interface (GUI), it is possible to see the list of connected end users or to simulate some Metaverse API calls even without XR headset running.

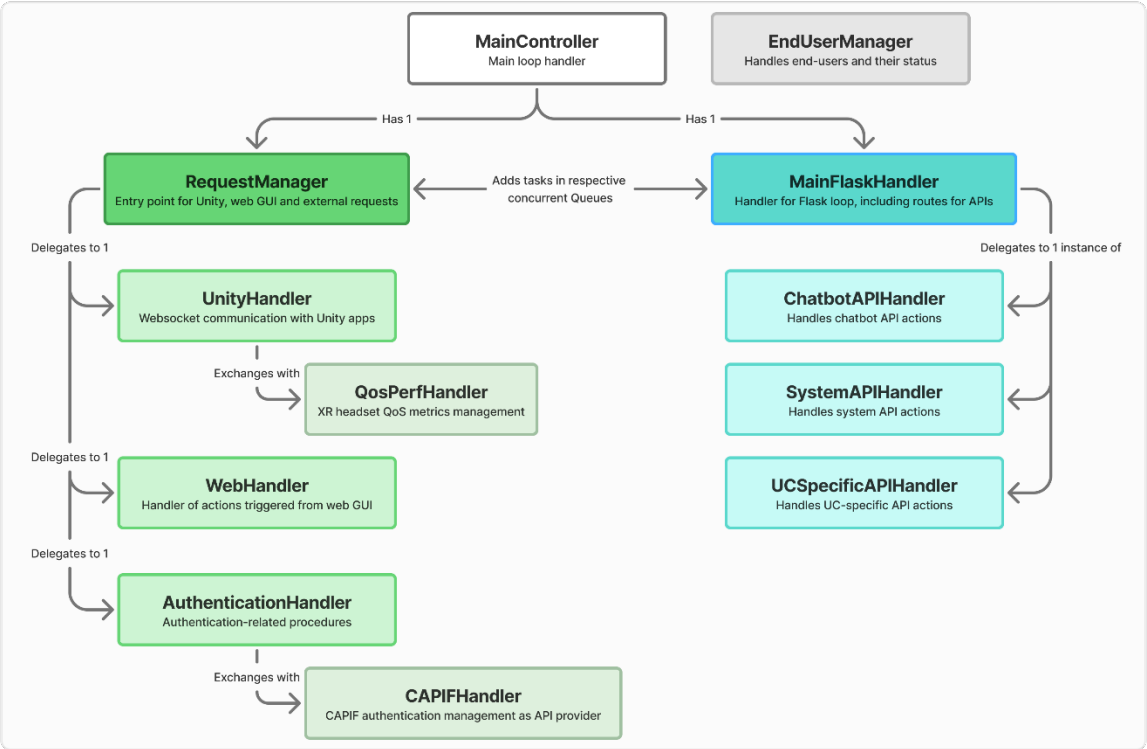


Figure 13: Overview of the main components of the final Metaverse manager architecture

As with the Unity applications, three different iterations of the Metaverse manager were developed following the requirements defined in D2.3. In addition to the requirements mentioned in the previous section, the Metaverse manager particularly addresses:

- **REQ-USER-Both-NF-R-1:** Easy connection to the SAFE-6G network (The Metaverse manager handles all the communications with SAFE-6G components).
- **REQ-USER-CATEGORY-Both-F-R-1**: Protection the Chatbot from input attacks (as intermediate component, the Metaverse manager can check early user inputs and sanitize them if needed).

In the last release **(Rel-C), the Metaverse manager provides the following features**:

- End-user management options. This includes viewing current users and their status (connected, registered, authenticated…). It is also possible to create simulated users and modify their status on the fly.

- High-level logs to facilitate checking actions and consequences on both itself and on the Unity app side.
- Debug and Metaverse API testing features to be able to trigger API calls, simulate effects as if a Unity app was running and get back an answer.

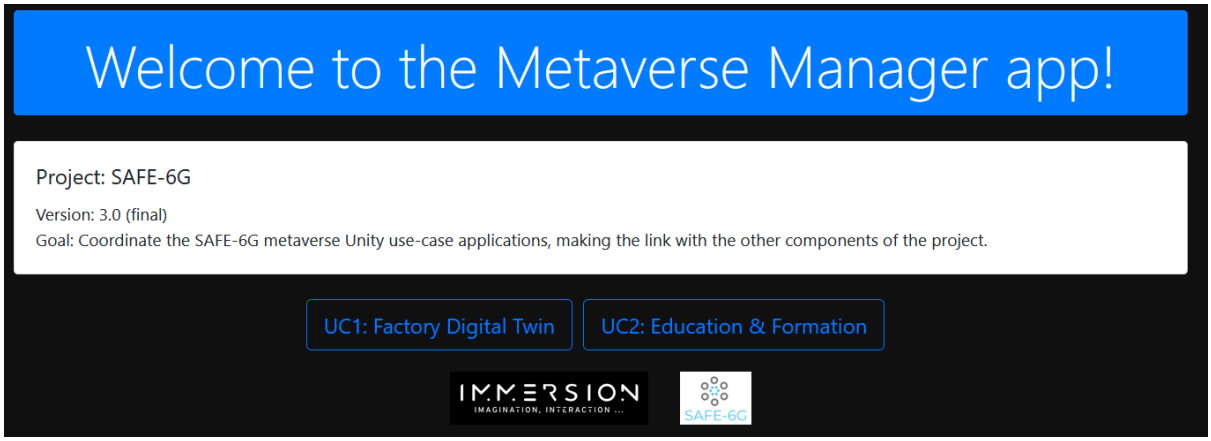


Figure 14: Capture of the home page of the Metaverse manager web GUI

Besides, with the new split of the Unity applications (CloudXR client and server), the Metaverse manager was also adapted to correctly make the link between the two applications and trigger Metaverse API related actions on the proper application. Besides, as it exposes APIs useful for SAFE-6G components, it was also updated to handle properly the CAPIF-registration procedure to make API discoverable and usable through CAPIF.

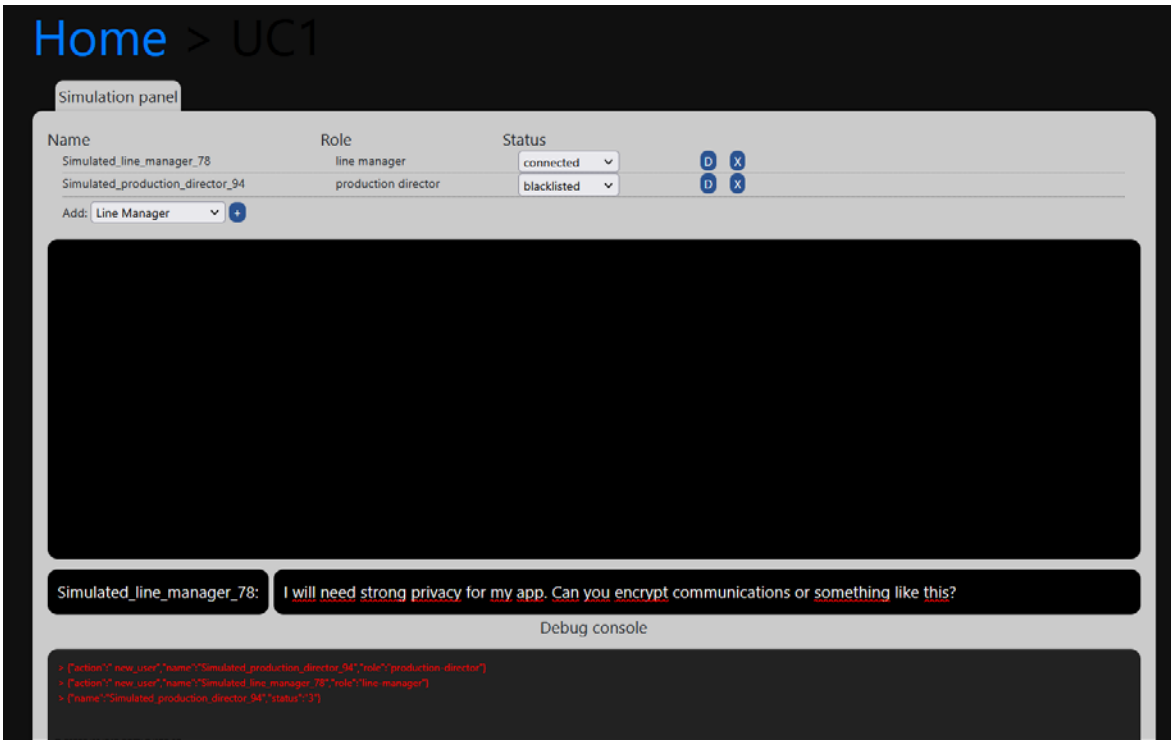


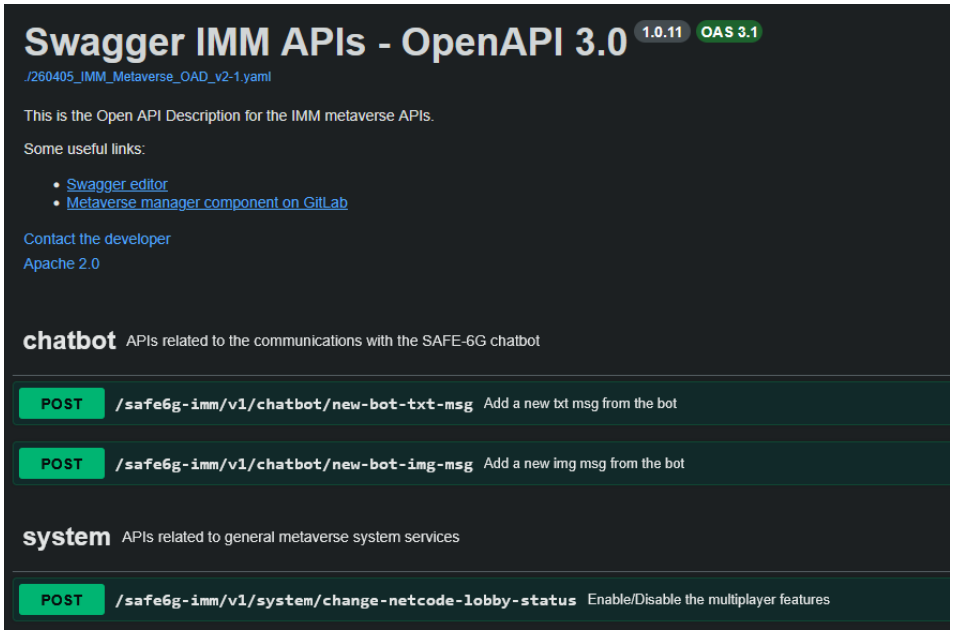
Figure 15: Screenshot of the Metaverse manager web GUI. Two simulated users have been created thanks to the provided widgets. The production director has also been blacklisted

The Metaverse manager is a Python application based on Flask and Bootstrap for its web GUI. It also uses websockets to communicate with Unity applications. The initial SocketIO module was eventually replaced by custom websockets to avoid being too limited by the C# wrapper implementation. The initial Docker container deployment was also replaced by a K8s deployment at NCSRD premises.

Category	*Status*
Version	latest/3.0
Documentation	https://gitlab.com/safe-6g/pilots/metaverse-manager/-/blob/main/README.md?ref_type=heads
Source code	https://gitlab.com/safe-6g/pilots/metaverse-manager
Docker images	https://hub.docker.com/r/safe6g/metaverse-manager
Helm charts	metaverse-manager-0.1.0.tgz
Open API Swagger	Metaverse API Swagger link - http://178.170.49.141:8081/#/
Demo	NA
Deployment requirements	1GB RAM, 0.5 vCPU, Recent web browser Ports 5001 and 9991
Dependencies	Standalone components, but interacts with the Unity apps, the SAFE-6G Chatbot backend. It uses the following technologies: • Python 3.10, Flask (3.1) • Requests module (2.26) • Bootstrap for the web GUI
License	MIT licence

Table 2: Summary of the status of the version 3.0 of the Metaverse manager

The Metaverse manager exposes several APIs to let other SAFE-6G components get data from the Unity UC applications and trigger effects on them. The Metaverse APIs are grouped into three categories: Chatbot-related APIs, system APIs and UC-specific APIs. These APIs are further described in D3.4.



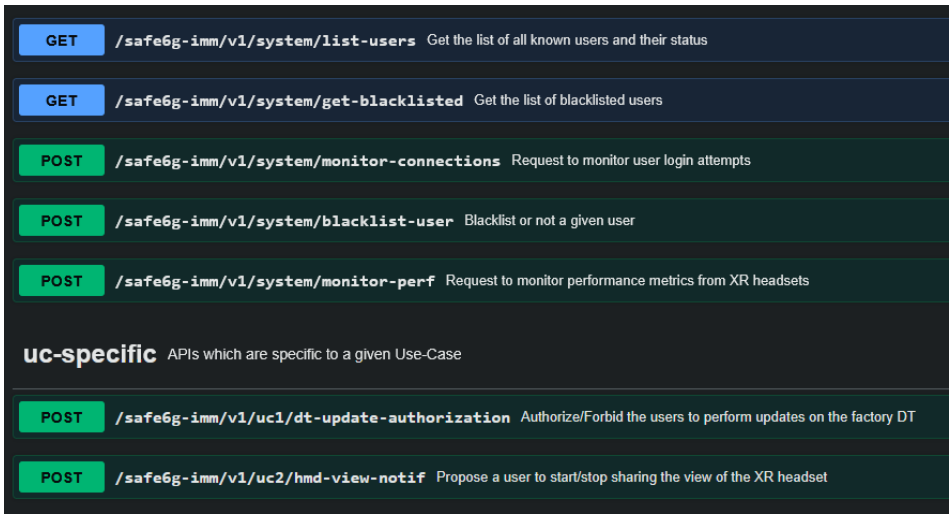


Figure 16: Overview of the Metaverse APIs exposed by the manager

4 SAFE-6G CHATBOT

Within the final SAFE-6G end-user workflow, the Chatbot provides the conversational interface through which users express and refine their trust-related requirements before accessing the target XR use-case scene. The component acts as the user-facing entry point for translating natural-language requests into trust-related intents that can be processed by the SAFE-6G framework. The implementation of the SAFE-6G Chatbot was guided by the following requirements identified in D2.1:

- **REQ-CHAT-UI-NF-R-01:** The Chatbot should provide an accessible and comprehensible interface for non-technical users, enabling interaction with the SAFE-6G framework without requiring specific technical expertise.
- **REQ-CHAT-TRU-F-M-02:** The Chatbot must communicate recognized user intents to the Cognitive Coordinator, which is responsible for computing the required level of trust across dimensions such as safety, security, privacy, resilience and reliability.
- **REQ-CHAT-UI-F-M-03:** The Chatbot must provide a user-friendly conversational interface and use NLP techniques to parse and recognize user intents related to the SAFE-6G trustworthiness dimensions.
- **REQ-CHAT-ADA-NF-M-05:** The Chatbot must be adaptable to different user requirements and use-case contexts, handling diverse inputs and providing appropriate responses according to the detected intent.
- **REQ-CHAT-UX-F-R-06:** The Chatbot should generate automated responses based on user input and predefined trust-oriented interaction logic, improving the efficiency of the interaction.
- **REQ-CHAT-IME-F-M-07:** The Chatbot must be capable of integration with immersive environments, allowing users in XR-based industrial production and training scenarios to interact with it seamlessly.
- **REQ-CHAT-DAT-F-M-08:** The Chatbot should expose and access endpoints that allow data exchange with Metaverse applications, including receiving end-user inputs and returning generated responses, while the Metaverse applications remain responsible for XR input/output handling.
- **REQ-CHAT-DAT-F-M-09:** The Chatbot must include an API interface for communicating with the Cognitive Coordinator, sending recognized user intents and receiving trust-related results or status information.

Based on these requirements and on the final integration workflow adopted for the Metaverse use cases, the final release of the SAFE-6G Chatbot provides the **following features (Rel-B)**:

- Conversational user interaction, allowing end-users to express and refine trust-related requirements through natural-language dialogue.
- Service-oriented Chatbot API, exposing HTTP-based endpoints for user interaction, request/response validation, session handling and integration with other SAFE-6G components.

- Intent-recognition pipeline, classifying user requests into the SAFE-6G trustworthiness categories of Security, Privacy, Safety, Reliability and Resilience, while filtering unrelated or out-of-scope inputs.
- LLM-based automated response generation, using intent-specific prompting and recent conversation context to provide trust-aware responses to the user.
- Cognitive Coordinator integration, preparing and submitting structured user metadata and classified intent information so that the corresponding trust-related actions can be coordinated within the SAFE-6G framework.
- Testing and debugging support, including a lightweight web client, session reset/debug operations and Swagger/Open API documentation for easier integration by developers and other SAFE-6G components.

4.1 EVOLUTION FROM THE INITIAL DESIGN AND FINAL RUNTIME FLOW

Compared with the initial design, the final SAFE-6G Chatbot implementation evolved from a standalone conversational interface concept into an integrated backend component of the SAFE-6G trust-management workflow. The main change concerns the role of the Chatbot in the overall architecture: it no longer represents only a user-facing dialogue prototype, but acts as the component responsible for translating natural-language user requests into structured trust-related intent information that can be consumed by the rest of the SAFE-6G framework.

A second important evolution concerns the separation of responsibilities between the Chatbot and the surrounding Metaverse components. The XR application and the Metaverse Manager handle the immersive interface, user registration and communication with the Metaverse environment, as described in the previous sections. The Chatbot backend is limited to the conversational and trust-intent processing logic: receiving user messages through its API, maintaining the dialogue context, identifying the relevant trustworthiness dimension, generating an appropriate response and preparing the information required by the Cognitive Coordinator.

At architectural level, the final runtime flow can therefore be summarized from the Chatbot perspective as follows. The Chatbot receives a user message and sender metadata from the integration layer, processes the message through its intent-recognition and response-generation pipeline, updates the session context and returns a trust-aware response to the calling component. Once the dialogue contains sufficient trust-related information, the Chatbot submits the structured intent information to the Cognitive Coordinator. The concrete API-level realization of this flow is summarized in Figure 17 and detailed in the following paragraphs.

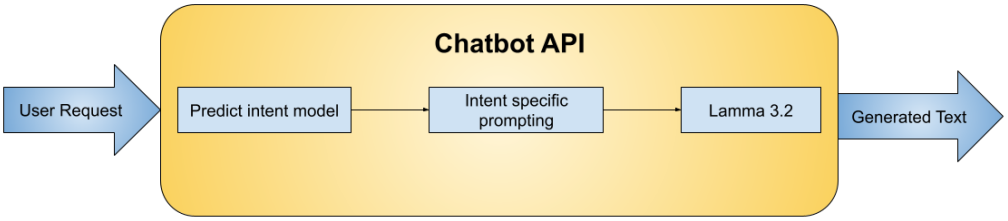


Figure 17: Chatbot API processing flow from user request to intent prediction, intent-specific prompting, Llama 3.2 response generation and generated text output

The intent-recognition step shown in the processing flow is implemented through the Predict Intent module, a dedicated component responsible for classifying incoming user requests according to the SAFE-6G trustworthiness functions. The model behind this component is trained through the SAFE-6G MLOps workflow, where the training and evaluation process can be managed using the project's MLOps tooling. Once trained, the resulting SetFit model artifacts are stored and made available for deployment through the model-serving workflow.

The dataset used to train and refine the intent-recognition model was created through an iterative process involving the SAFE-6G consortium partners responsible for the different Trust Functions. Initially, partners provided representative examples of potential user requests related to their respective functions, such as Security, Privacy, Safety, Reliability and Resilience. These seed examples were then expanded and diversified to enrich the dataset with additional phrasings, request styles and usage scenarios. During Chatbot testing and integration activities, further user inputs and observed interaction patterns were collected and used to continuously improve the dataset, making the intent-recognition component more robust against variations in natural-language requests.

For deployment and inference, the trained Predict Intent model is served through the SAFE-6G MLOps model-serving repository dedicated to the Chatbot. This repository packages the SetFit-based intent-recognition model as a service that can be consumed by the Chatbot backend. The serving component loads the trained model artifacts, exposes a prediction endpoint, validates incoming inference requests and returns the predicted intent class to the Chatbot pipeline.

At the intent-recognition stage shown in Figure 17, the Predict Intent API maps each user message to one of the SAFE-6G trustworthiness categories: Security, Privacy, Safety, Reliability or Resilience. If the message does not match any of these trustworthiness functions, it is assigned to an irrelevant or out-of-scope class, preventing unrelated user input from being forwarded as a valid trust-function request. This classification is performed using a trained SetFit model based on the `sentence-transformers/all-MiniLM-L6-v2` base model. The predicted intent is then used to select the appropriate trust-oriented system prompt and guide the response generation process. Responses are generated through an Ollama-hosted Llama 3.2 model, allowing the Chatbot to provide contextual, intent-specific answers while maintaining recent conversation history as part of the model context.

The service maintains lightweight in-memory session state per user, identified through the sender information included in the request. This enables the Chatbot to preserve conversational continuity, track the active trust function, store recent user-assistant turns and advance through the trust-function sequence during the interaction. The API also supports explicit reset and debug operations, which are useful during testing, integration and demonstration scenarios.

For Cognitive Coordinator communication, the Chatbot builds a structured payload containing the user metadata and the classified intent history and submits it to the configured CoCo endpoint once sufficient trust-related input has been collected. The response from the Cognitive Coordinator is then appended to the Chatbot output, allowing the user-facing interface to present calculated trust-related results in the same conversational flow.

In addition to the XR-based Chatbot interface, a lightweight web-based Chatbot client was developed for testing, debugging and demonstration purposes. This client allows developers and integrators to manually define the sender metadata used by the Chatbot API, including the username, status, role, hardware identifier and MSISDN and to submit natural-language messages directly to the Chatbot backend. The generated Chatbot responses are displayed in the chat panel, making it possible to inspect the conversation flow without launching the full XR application. The interface also provides a dedicated control for submitting the accumulated intent information to the Cognitive Coordinator, which facilitates integration testing of the end-to-end workflow from user input to trust-related intent processing.

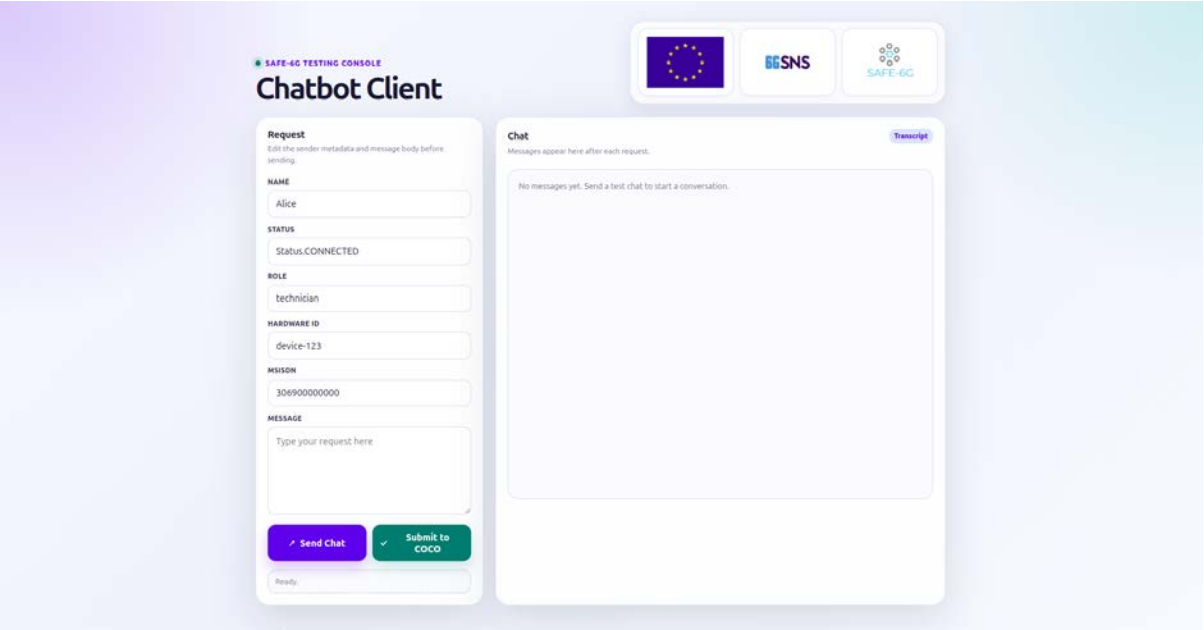


Figure 18: Chatbot Client testing interface

In addition to internal testing and integration activities, the SAFE-6G Chatbot was presented and demonstrated through official project trial and showcasing activities at EuCNC & 6G Summit 2025 in Poznań and EuCNC & 6G Summit 2026 in Málaga. These events provided an important opportunity to demonstrate the Chatbot as a user-facing component of the SAFE-6G framework, highlighting its role in enabling natural-language interaction with the trust-management workflow. During the demonstrations, the Chatbot was showcased as the entry point through which users can express trust-related requirements, refine their intent, and trigger the corresponding SAFE-6G processing chain. The participation in these major European 6G events supported the dissemination and validation of the Chatbot implementation in realistic trial and demonstration settings.

The implementation is containerized for reproducible deployment. The Docker setup separates the Chatbot API container from the Ollama service container, with environment-driven configuration for the LLM endpoint, model name, intent model path, conversation limits and Cognitive Coordinator URLs. Since the LLM model name is provided through configuration, the response-generation backend can be easily adapted to use a different or larger Ollama-supported model when higher reasoning capacity or response quality is required.

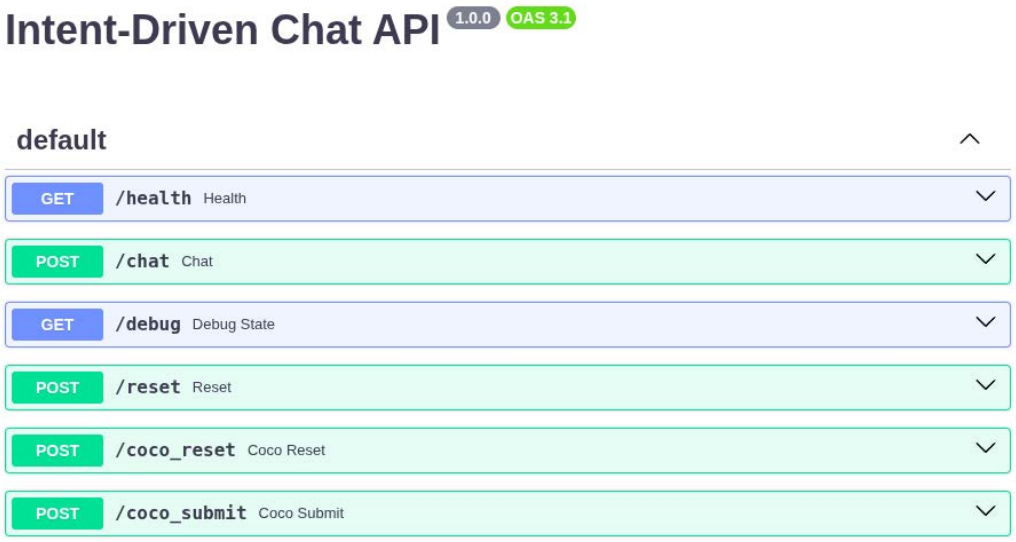


Figure 19: Swagger view of the Chatbot API endpoints implemented for testing, debugging, and Cognitive Coordinator integration

Table 3 summarizes the implementation status of the Chatbot API, including the available documentation and source-code references, API description, deployment requirements and dependencies required for its integration with the SAFE-6G framework.

Category	***Status***
Version	Latest/4.0
Documentation	https://gitlab.com/safe-6g/development/chatbot_api/-/blob/main/README.md?ref_type=heads
Source code	https://gitlab.com/safe-6g/development/chatbot_api
Open API Swagger	https://gitlab.com/safe-6g/development/chatbot_api/-/blob/main/openapi.json?ref_type=heads
Deployment requirements	CUDA-capable GPU recommended for LLM inference, CPU-only execution is supported for testing but results in slower response times. A recent web browser is required only for the Chatbot client testing interface.
Docker Images	https://hub.docker.com/r/safe6g/ollama-chatbot-service, https://hub.docker.com/r/safe6g/chatbot
Helm charts	chatbot-api-4.0.0.tgz
Dependencies	Standalone component, but interacts with the Metaverse Manager and the Cognitive Coordinator.
Licence	Apache Licence 2.0

Table 3: Summary of the status Chatbot API

Overall, the Chatbot API operationalizes the SAFE-6G Chatbot concept as an executable integration component. It provides a concrete backend service for intent-driven trust interaction, LLM-based response generation, session-aware dialogue management and Cognitive Coordinator communication.

5 CONCLUSION

This document reported the outcomes of T3.5, mostly related to the final release of the software assets developed, tailored or updated, related to the Metaverse ecosystem and the Chatbot. With the release of the latest versions, the SAFE-6G Chatbot, Metaverse manager and the Unity UC applications successfully reached their final state and are now integrated within the SAFE-6G pipeline. Only potential minor refinements are expected to be required during the validation activities from WP5.