D2.3 DEVELOPMENT INFRASTRUCTURE AND INTEGRATION GUIDELINES

D2.3: Development Infrastructure and Integration Guidelines

WP2

31-05-2023

Funded by
the European Union

Version	1.0
WP	WP2
Dissemination level	Public
Deliverable lead	SYN
Authors	Stavroula Bourou, Drakoulis Petros, Konstantoudakis Konstantinos, Mpiliousis Stefanos, Papadopoulos Georgios, Zarpalas Dimitrios, Yusuf Can Semerci, Apostolos Maniatis
Reviewers	Nikos Achilleopoulos; Manuel Toledo; Astik Samal, Spiros Borotis, Olga Chatzifoti, Alberto Casanova, Nikolaos Skoulikas
Abstract	This deliverable provides the reader an overview of VOXReality components, how they interact with each other and the process diagrams per use case. Moreover, it describes the integration guidelines as well as the integration and validation methodology that will be followed during the VOXReality project. Finally, the deliverable contains the development infrastructure and the integration tools that will be followed.
Keywords	Integration, Interactions, Process Diagrams, Continuous Integration, Continuous Delivery, Validation
License	This work is licensed under a Creative Commons Attribution-No Derivatives 4.0 International License (CC BY-ND 4.0). See: https://creativecommons.org/licenses/by-nd/4.0/

Dissemination Level	
PU	Public
PP	Restricted to other programme participants (Including the Commission Services)
RE	Restricted to a group specified by the consortium (Including the Commission Services)
CO	Confidential, only for members of the consortium (Including the Commission Services)

Nature	
PR	Prototype
RE	Report
SP	Specification
TO	Tool
OT	Other

Version History

Version	Date	Owner	Author(s)	Changes to previous version
0.1	31-03-2023	SYN	Stavroula Bourou	ToC
0.2	05-05-2023	SYN	Stavroula Bourou	First inputs to the deliverable
0.3	10-05-2023	CERTH, UM, SYN	Drakoulis Petros, Konstantoudakis Konstantinos, Mpiliousis Stefanos, Papadopoulos Georgios, Zarpalas Dimitrios, Yusuf Can Semerci, Apostolos Maniatis	Input to section 2.1
0.4	18-05-2023	SYN	Stavroula Bourou	Additional inputs to the deliverable
0.5	22-05-2023	SYN	Stavroula Bourou	Peer-review version ready and additional updates
0.5.1	24-05-2023	MAG	Astik Samal, Spiros Borotis, Olga Chatzifoti, Alberto Casanova, Nikolaos Skoulikas, Nikos Achilleopoulos	Review and addition
0.5.2	29-05-2023	VRDAYS MAG	Manuel Toledo Nikos Achilleopoulos	Review
0.6	30-05-2023	SYN	Stavroula Bourou	Updates based on peer review comments and overall enhancements

Table of Contents

List of Abbreviations & Acronyms

API	:	Application Programming Interface
ASR	:	Automatic Speech Recognition
CD	:	Continuous Deployment
CI	:	Continuous Integration
DevOps	:	Development, Operations
DS	:	Dialogue System
HTML	:	HyperText Markup Language
HTTP	:	HyperText Transfer Protocol
IoT	:	Internet of Things
JSON	:	JavaScript Object Notation
KPI	:	Key Performance Indicator
ML	:	Machine Learning
NDI	:	Several non-destructive inspection methods
NLG	:	Natural Language Generation
NLP	:	Natural Language Processing
NLU	:	Natural Language Understanding
NMT	:	Neural Machine Translation
OAS	:	OpenAPI Specification
OS	:	Operating System
PaaS	:	Platforms as a Service
POV	:	Point of view
REST	:	REpresentational State Transfer
SAST	:	Static Application Security Testing
SCM	:	Source Code Management
SDLC	:	Software Development Lifecycle
TDD	:	Test-Driven Development
VCM	:	Version Control System
VFX	:	Visual Effects
VL	:	Vision Language
VR	:	Virtual Reality
WP	:	Work Package
XML	:	Extensible Markup Language
XR	:	eXtended Reality

List of Figures

List of Tables

Executive Summary

The purpose of this deliverable is to determine the integration and validation processes to be followed in the VOXReality project as well as to provide the integration guidelines while following the Continuous Integration and Continuous Delivery practice. In addition, a description of the components that will be developed during the project is presented aiming to highlight the main functionalities of VOXReality as well as the components' interactions and data formats. Process diagrams of the various tasks that will be carried out in pilots are also included. Furthermore, the D2.3 analyzes the development infrastructure requirements that will be needed to integrate and validate the VOXReality components.

1 Introduction

Integration and validation procedures are of crucial significance for every system that intents to be successful while efficiently fulfilling its purpose and role. A noteworthy solution that aims to be realised in a system or a platform or even a service is materialized as the outcome of proper integration and validation activities. Through these activities all the development outcomes are fused together into a unified and effective system.

To successfully guide the process of integrating different software systems, components and modules into a unified system is needed to set up the integration guidelines. Specifically, the guidelines include all the needed principles, best practices, and standards to help developers to design and implement an integrated system that is scalable and easy to maintain. The integration guidelines can bring several benefits to the project including improved code quality, reduced costs and time as well as better consistency.

The integration guidelines contain information and advice on the following topics:

- System architecture, providing an overview of the structure, behavior, and functionality. The goal of system architecture is to ensure that the system is well-designed, efficient, and scalable, and that it meets the needs of its users.
- Data formats and protocols, defining the practices for exchanging data between different components, including the usage of standard data formats and communication protocols. Based on this, developers can ensure that their systems can exchange data with other systems seamlessly and reliably.
- Testing and validation, determining the procedures to efficiently test and validate the integrated system. Those procedures may include unit, integration and performance testing and ensure that the software system or application is functioning correctly, efficiently, and reliably.

The aim of VOXReality is to generate a set of pre-trained models that combine language and AI vision, as well as a set of applications using those models to demonstrate innovation in various sectors. Together with those models the VOXReality consortium will provide the means and methods to use those AI models, such as inference codes, etc., as well as the AI tools to perform "once-for-all" training, finetune and optimize those AI models on different applications by stakeholders and third parties' users.

This deliverable describes the integration guidelines that should be followed by the VOXReality AI engineers and developers to ensure that the developed system behaves correctly and efficiently. Initially a description of VOXReality components is given, accompanied with information regarding the interaction of those components, including the data format. Additionally, the processes that would be followed with use cases are provided via process diagrams. Originate from those architectures, the scope of the project and the desired outcomes as well as an efficient integration and validation strategy is defined for VOXReality project. Additionally, the integration guidelines as well as tools and frameworks for integration and testing activities, within the project's CI/CD lifecycle, are introduced in this document. In the end, the development infrastructure that is utilized in the VOXReality project is presented. It should be mentioned that the integration methodology that is presented in this deliverable is closely connected with the work is conducted in task 4.1 "Model deployment and serving”, which its output and results will be presented in D4.1 “Model deployment analysis V1” in M17 and in the D4.2 “Model deployment analysis V2” in M32.

1.1 Intended Audience

The intended audience of this deliverable is mainly the consortium of the VOXReality project. In detail, this document provides the VOXReality architectures per use case and introduces to the technical partners of the consortium, the integration and validation environment of the VOXReality system as well as the corresponding guidelines that need to be followed through the development, integration and validation lifecycle of the project. Additionally, this document is useful for third parties’ users, including those who will be participating at project’s Open Calls as well as researchers who want to explore the VOXReality capabilities by utilizing its AI models and developed outcomes.

1.2 Relations to other activities

As already stated, integration and validation activities act as the foundation of the system development process. Consequently, this document is strongly connected with all the technical WPs (3-4). Moreover, it is also related to WP5 that consists of the validation work, dictated by the VOXReality Pilots cases. Finally, it is also related with WP6 and WP7 in the context of impact creation and Open Calls, respectively.

1.3 Document Structure

The present deliverable introduces the VOXReality process diagrams per use case and presents the principals on the Integration guidelines, the integration and validation plan, the frameworks and tools that will be used and finally introduces the development infrastructure.

Section 1 provides an introduction of the deliverable’s intended audience as well as an overview of its content. *Section 2* presents the VOXReality system, through the analysis of components and their interactions. Additionally, it presents some the process diagrams of each use case that describe the flow of the information between the VOXReality platform components. *Section 3* provides the integration guidelines aimed at establishing common development procedures that will significantly facilitate and accelerate the interaction between development teams, as well as the integration of diverse software pieces into a common VOXReality framework. Moreover, it contains information regarding modular software design, RESTful architecture, source code and API documentation. *Section 4* presents information regarding VOXReality integration plan. Specifically, it details the integration and validation methodology that will be used as well as the respective phases and integration time-plan. *Section 5* presents VOXReality integration framework and tools. Particularly it describes the DevOps practices that will drive the development and integration activities of the project and on the relevant CI/CD pipeline. Moreover, source code management procedures and the containerization concept as well as the VOXReality development environment are introduced. *Section 6* summarizes the conclusions that are deduced from the current document.

2 VOXReality System Overview

VOXReality integrates language- and vision-based AI models, aiming to tackle the challenges associated with human-to-human and human-to-machine interaction in XR space. To achieve this, VOXReality utilizes NLP and CV advancements to create robust AI models. The multi-modal information is exchanged between the modalities with unidirectional or bidirectional ways to drive AR and VR, enabling natural human interactions with the backends of XR systems and creating multi-modal XR experiences from the combination of vision and sound information.

Particularly, powerful multi-tasking NLP models that are adjustable to different languages and expressions while they can consider the surrounding context, are implemented. Additionally, based on language and visual information, visually grounded language models are built providing valuable information about the surrounding. Finally, by utilizing the outputs of the aforementioned AI models and additional knowledge, a context-aware dialogue system is developed, which creates well-grounded conversations, provides navigation guidelines and assistance to the user via XR. Specifically, four main components are implemented in VOXReality: Automatic Speech Recognition (ASR), including the Error Correction component, Neural Machine Translation (NMT), Vision language Models (VL) and Dialogue System (DS).

The developed AI models will be used to create immersive XR experience. Specifically, those models will be deployed and validated in three use cases: VR Conference, Augmented Theatre and Training Assistant, which have been described in D2.1 “Definition and Analysis of VOXReality Use Cases V1” [1].

This section contains a brief description of VOXReality AI components that are developed in the project. Additionally, information about how these components interact with each other is provided, focusing on input and output data format and language. Finally, process diagrams and analysis of each use case are presented.

2.1 VOXReality Components

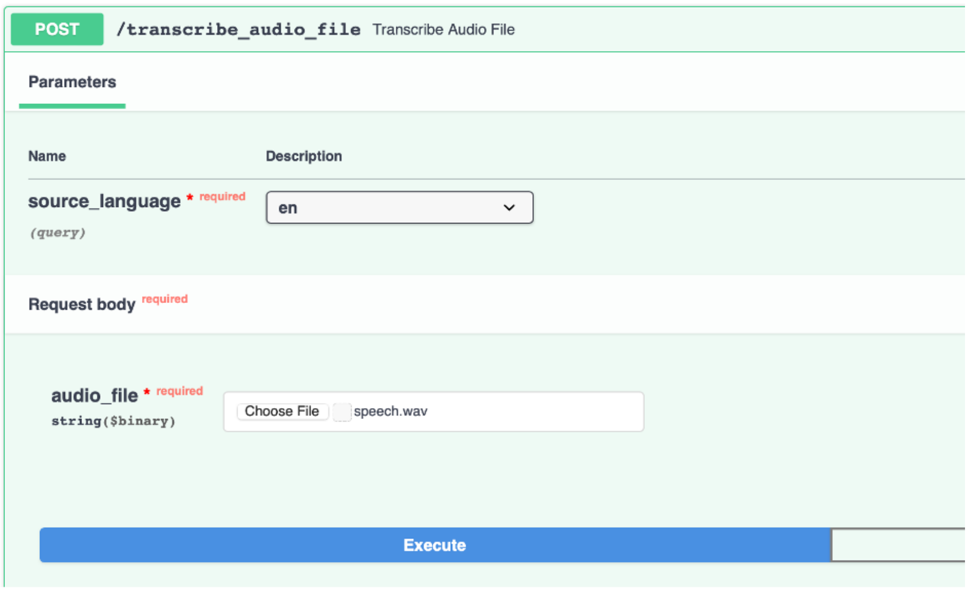
In this subsection, the main AI components of VOXReality are analyzed as well as information about their interaction.

2.1.1 Automatic Speech Recognition

The role of the Automatic Speech Recognition (ASR) component is to generate transcriptions of user speech. It operates by taking audio files in the ".wav" format and producing transcriptions in the language spoken by the individuals. This component incorporates an advanced deep learning model that has been trained using the VOXReality languages, namely English, German, Greek, Dutch, Spanish, and Italian. It is packaged within a docker container, equipped with RESTful API capabilities.

To access the ASR functionality, the API requires a POST request containing the audio file in ".wav" format as it is depicted in Figure 1. The source language is automatically detected, but the request can also specify the source language. Figure 2 shows the resulting transcription text is provided in the JSON format.

Additionally, the ASR component considers the visual context provided in English text from the Visual-Language models (Section 2.1.4) to enhance the accuracy of the transcription.


POST /transcribe_audio_file Transcribe Audio File

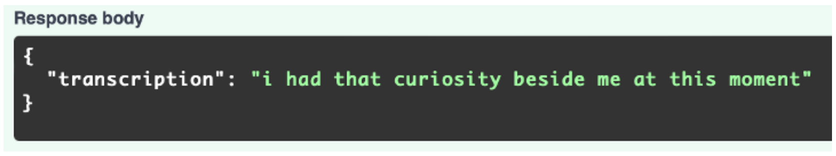
Parameters

Name	Description
source_language * required (query)	en

Request body required

audio_file * required string($binary) Choose File speech.wav

Figure 1: POST call providing the audio file to ASR.


Response body

```
{
  "transcription": "i had that curiosity beside me at this moment"
}
```

Figure 2: Transcription of input audio file.

2.1.1.1 Error Correction

The Error Correction component is responsible for improving the grammatical and syntactical accuracy of the transcriptions generated by the ASR component. It takes text-based sentences as input and produces corrected text in the language spoken by the users. Similar to the ASR component, it utilizes a cutting-edge deep learning model trained with the VOXReality languages and is included in the ASR docker bundle.

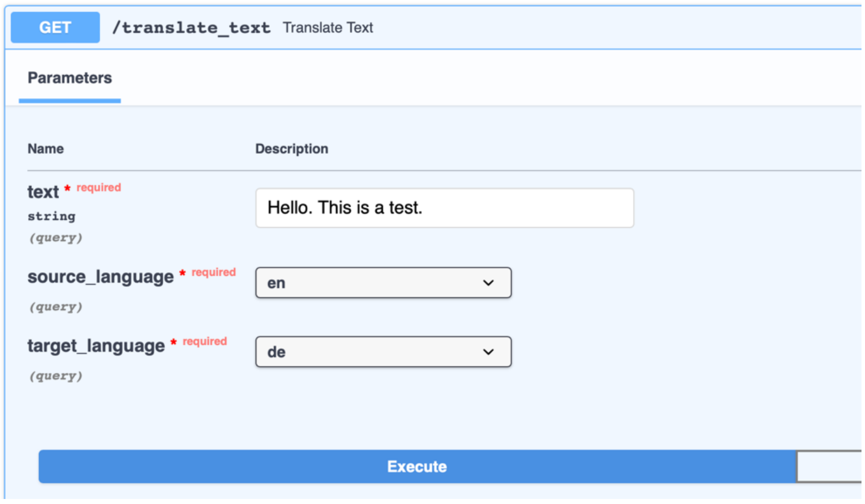
The Error Correction process occurs automatically after the ASR component generates the initial transcription from the input audio. The ASR component's response in the JSON format is the corrected version of the original transcription.

2.1.2 Neural Machine Translation

The Neural Machine Translation (NMT) component handles the translation of text from one VOXReality language to another. It utilizes a state-of-the-art deep learning model trained with the VOXReality languages and is bundled into a docker container with RESTful API capabilities.

To utilize the NMT component, the API is accessed via a GET request containing the text sentences in the desired language and specifying the target language for translation, as it is visualized in Figure 3. The source language is automatically determined, but it can also be explicitly provided in the request. The translated text is provided in the JSON format, aligned with the target language specified in the request. Figure 4 depicts the translated output.

Similar to the ASR component, the NMT component leverages the visual context provided by English text from the Visual-Language models (Section 2.1.4) to enhance the translation process.

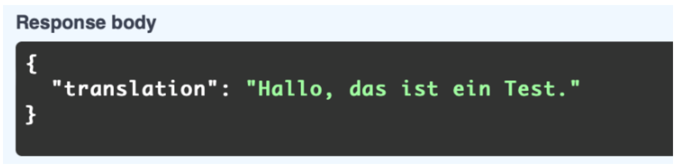


GET /translate_text Translate Text

Parameters

Name	Description
text * required string (query)	Hello. This is a test.
source_language * required (query)	en
target_language * required (query)	de

Figure 3: GET call providing the text file to Neural Machine Translation.



Response body

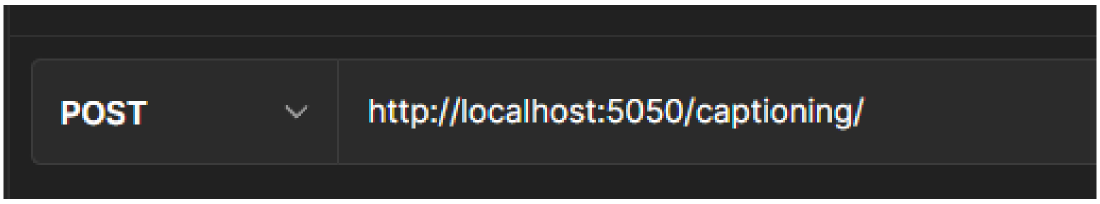
```
{
  "translation": "Hallo, das ist ein Test."
}
```

Figure 4: Translated output of Neural Machine Translation.

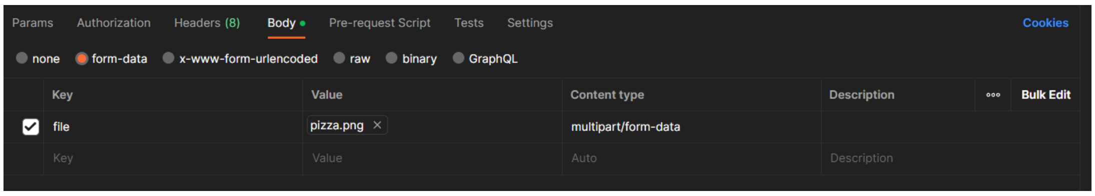
2.1.3 Vision Language Models

The Vision Language (VL) models to be developed will be bundled as docker containers, instantiated on-demand on CUDA compliant infrastructure of sufficient capacity. Their functionality will be accessible via REST API calls to distinct, task-specific, network endpoints. The envisioned VL tasks are the three following: 1) Scene Captioning 2) Visual Question Answering 3) Scene Spatial Description. The output of these models forms the "visual context" of the VOXReality pipeline to be used by various other components.

1) Scene Captioning: It refers to the task of generating the textual description or captions of a visual scene or image. The caller makes a POST call providing an image from the user's point of view to be captioned. Figure 5 shows POST call, while the Figure 6 presents the input image. The system returns a JSON with the answer in it as it is visualized in Figure 7. A visual example of this interaction follows:



POST http://localhost:5050/captioning/



Key	Value	Content type	Description
file	pizza.png	multipart/form-data	
Key	Value	Auto	Description

Figure 5: POST call providing the user's image to Scene Captioning VL task.

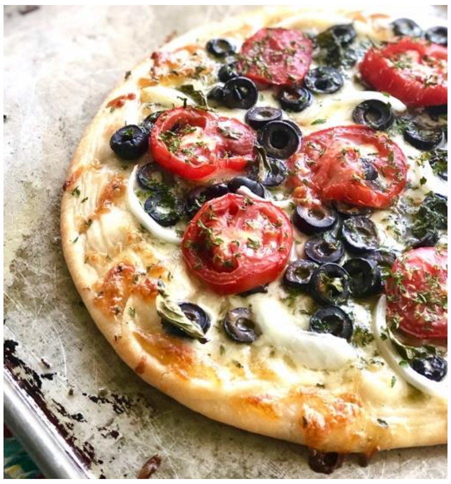
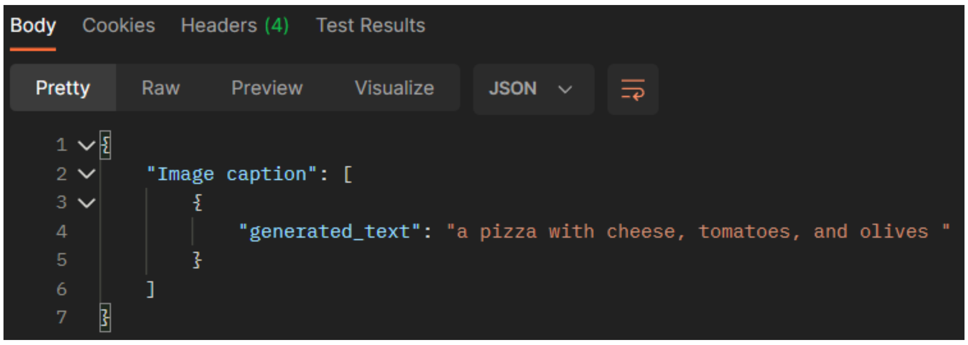


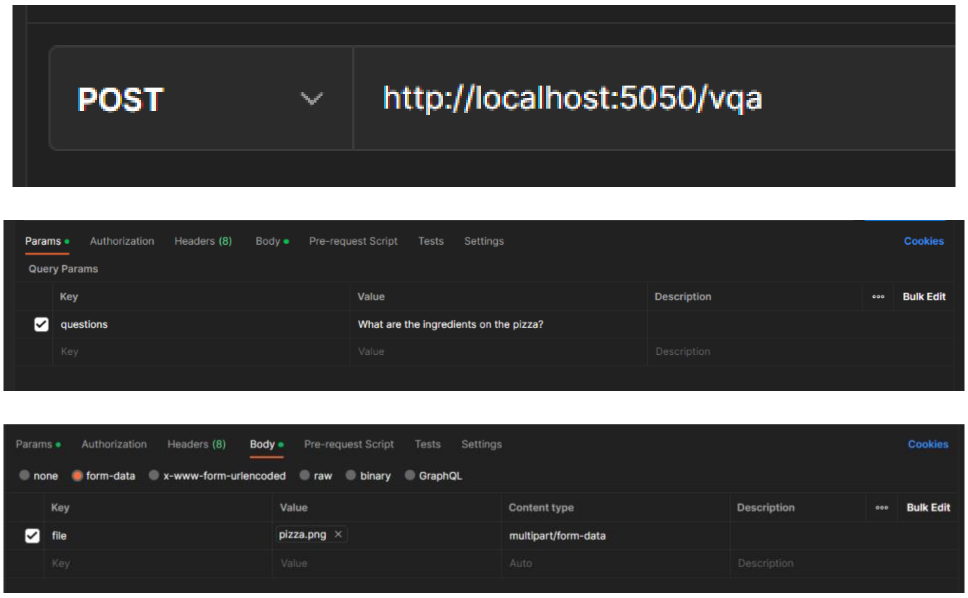
Figure 6: Input image to Scene Captioning Vision Language VL task.



```
{
    "Image caption": [
        {
            "generated_text": "a pizza with cheese, tomatoes, and olives "
        }
    ]
}
```

Figure 7: Output textual description of Scene Captioning VL task.

2) Visual Question Answering: This task attempts to correctly answer questions in natural language regarding visual content. The caller makes a POST call providing both an image from the user's point of view and a question to be answered based on the content of this image. The POST call with the requested question based on a specific image is depicted in Figure 8 while the image is demonstrated at Figure 9. The system returns a JSON with the answer in it as it is shown in Figure 10. A visual example of this interaction follows:


POST http://localhost:5050/vqa

Params ● Authorization Headers (8) Body ● Pre-request Script Tests Settings Cookies

Query Params

	Key	Value	Description
✓	questions	What are the ingredients on the pizza?	
	Key	Value	Description

Params ● Authorization Headers (8) Body ● Pre-request Script Tests Settings Cookies

none [✓ form-data] x-www-form-urlencoded raw binary GraphQL

	Key	Value	Content type	Description
✓	file	pizza.png	multipart/form-data	
	Key	Value	Auto	Description

Figure 8: POST call providing the user's image and the question to Visual Question Answering VL task.

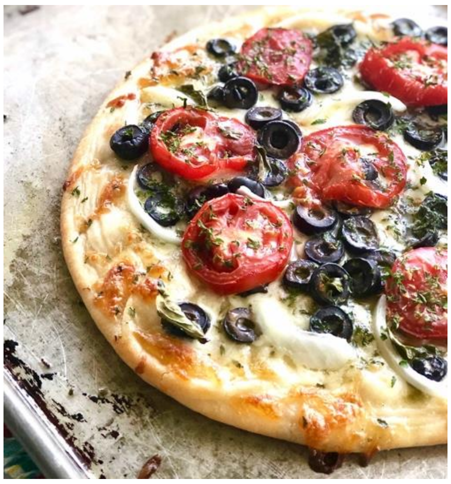
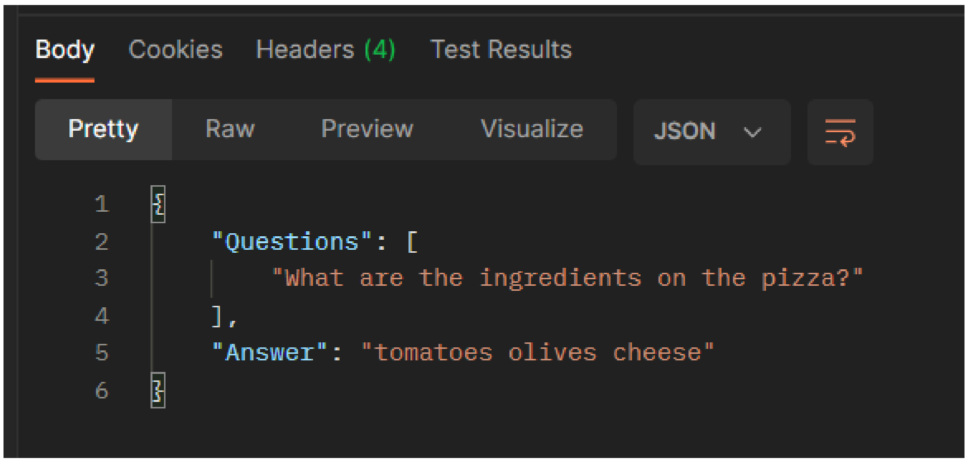


Figure 9: Input image to Visual Question Answering VL task.

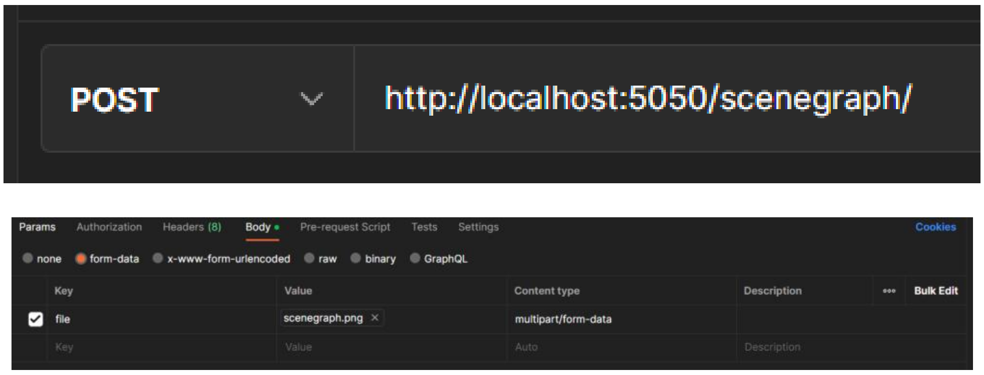

Body Cookies Headers (4) Test Results

Pretty Raw Preview Visualize JSON

```
{
    "Questions": [
        "What are the ingredients on the pizza?"
    ],
    "Answer": "tomatoes olives cheese"
}
```

Figure 10: Output answer to Visual Question Answering VL task based on question.

3) Scene Spatial Description: This task refers to the process of understanding and describing the spatial relationships between objects of visual scene. The caller makes a POST call providing an image from the user's point of view to be captioned as it is visualized in Figure 11. The input image is shown in Figure 12. The model identifies the visible objects and provides a number of sentences describing their spatial relationships as it is presented in Figure 13. A visual example of this interaction follows:



```
POST    http://localhost:5050/scenegraph/
```

	Key	Value	Content type	Description
✓	file	scenegraph.png	multipart/form-data	
	Key	Value	Auto	Description

Figure 11: POST call providing the user's image to Scene Spatial Description VL task.

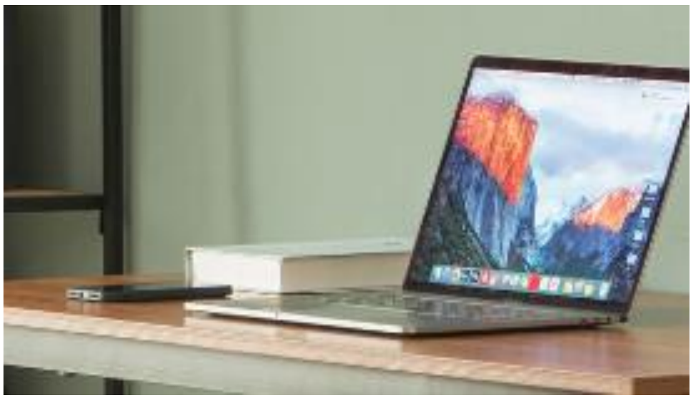
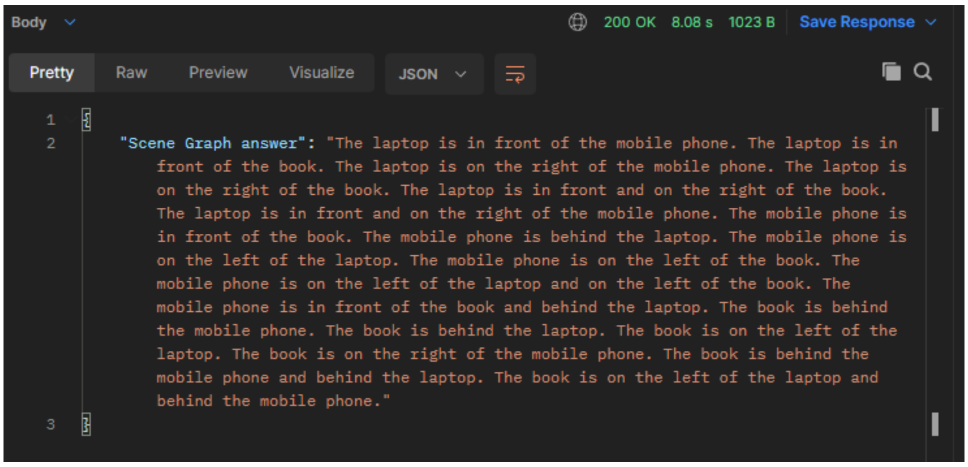


Figure 12: Input image to Scene Spatial Description task of Vision Language Models



200 OK 8.08 s 1023 B

```
{
    "Scene Graph answer": "The laptop is in front of the mobile phone. The laptop is in
        front of the book. The laptop is on the right of the mobile phone. The laptop is
        on the right of the book. The laptop is in front and on the right of the book.
        The laptop is in front and on the right of the mobile phone. The mobile phone is
        in front of the book. The mobile phone is behind the laptop. The mobile phone is
        on the left of the laptop. The mobile phone is on the left of the book. The
        mobile phone is on the left of the laptop and on the left of the book. The
        mobile phone is in front of the book and behind the laptop. The book is behind
        the mobile phone. The book is behind the laptop. The book is on the left of the
        laptop. The book is on the right of the mobile phone. The book is behind the
        mobile phone and behind the laptop. The book is on the left of the laptop and
        behind the mobile phone."
}
```

Figure 13: Output textual description to Scene Spatial Description task of Vision Language Models

Note that, in later stages of the project, video may be introduced as an input option as well.

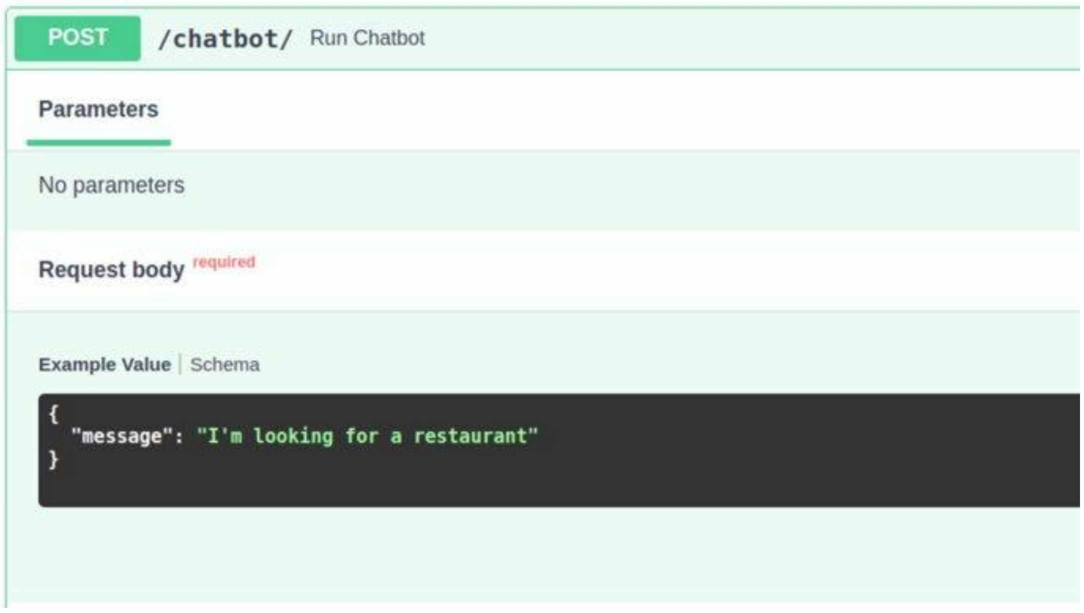
2.1.4 Dialogue System

The dialogue system model is a task-oriented system that is designed to generate human-like responses based on the user's input and the surrounding. By taking text inputs from multiple sources, the system will be capable of providing context-aware conversations, navigation instructions, information about the surrounding space and assembly instructions to the user. Additionally, the system will remember the conversation history, allowing it to provide relevant responses.

The dialogue system will be able to understand the input text using advanced Natural Language Understanding (NLU) model. Moreover, the system can manage the dialogue between the user and the system, keeping track of the history, the context as well as the state of conversation. The appropriate response to user's input is defined based on the current state of the dialogue. Finally, the natural-like response to the user is generated using robust Natural Language Generation (NLG) model.

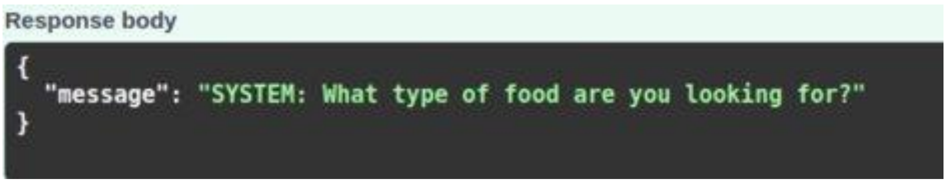
The automatic speech recognition (ASR) component will transcribe the user's speech into text, which the dialogue system is going to use to understand the user's intent. Also, the dialogue system could be connected to a knowledge base that provides external information. In addition to speech recognition, the dialogue system is going to also utilize text input from vision language models. These models provide descriptions of the environment, including information about the surrounding space and objects. By processing this information, the dialogue system could be capable of answering questions about the environment.

The dialogue system is built using state-of-the-art models that have been trained on publicly available datasets, enabling users to make reservations in multiple domains such as restaurants and hotels. The API system is built within a Docker container, allowing for easy deployment and scalability. The API is accessed through a POST request, with the user providing input in text format using the English language and the response from the dialogue system is returned in JSON format. Figure 14 depicts the POST call, containing the user's request, while the Figure 15 presents the response of the dialogue system.



```
{
  "message": "I'm looking for a restaurant"
}
```

Figure 14: POST call providing the user's request to Dialogue System.


Response body

```
{
  "message": "SYSTEM: What type of food are you looking for?"
}
```

Figure 15: Output response of Dialogue System.

Overall, the dialogue system is a key component of the VOXReality project, providing users with context-aware conversations, navigation, and assembly instructions. By combining inputs from speech and vision-based models, the system is able to understand the user's intent and provide relevant information about the environment. This makes the system a powerful tool for assisting users in a variety of tasks, from navigation to information retrieval.

2.1.5 Component Interactions

The VOXReality system consists of the components described in subsections 2.1.1 - 2.1.4 and accepts as input both audio, specifically speech, and visual data, which can be presented either on the scene or the user’s point of view (POV). The developed components interact with each other since the output of one component can be used as the input of another. It should be mentioned that the ASR, Error Correction and Neural Machine Translation can operate to any language between English, German, Greek, Dutch, Spanish and Italian, which are those languages that VOXReality focuses on, while Vision Language models and Dialogue System are functional for English.

An overview of the data flow between the different components is presented in Figure 16. Audio data is processed by ASR to generate transcriptions in the form of text. The audio data can be in any language between English, German, Greek, Dutch, Spanish and Italian. The transcription is in the same language as the audio data. Additionally, the ASR accepts as input the visual context produced by Vision Language Models to enhance the transcriptions. The Error Correction component is part of ASR, which accepts as input the transcription from ASR and provides an error-free text.

The Visual Language models receive as input the user’s point of view or the scene, being real or synthetic, process it and produce the “visual context” in the form of text data, which can be used by the ASR, Neural Machine Translation and Dialogue system to enhance their operation and assist them to generate a context-aware output.

Afterwards, the transcriptions are passed to Neural Machine Translation module, which translates it to the language of interest among the five languages that VOXReality focuses on. Additionally, the visual context produced by Vision Language Models can be utilized by this component to enhance the accuracy of translated text. If the user’s language is English and his/her intent is to interact with the Dialogue System, then the transcription is not processed by the NMT.

The Dialogue System accepts as input the English translated text from Neural Machine Translation as well as the visual context and produces an English response. If the user’s language is English, then this response is provided directly to him/her. Otherwise, it is processed through Neural Machine Translation to be translated to user's language.

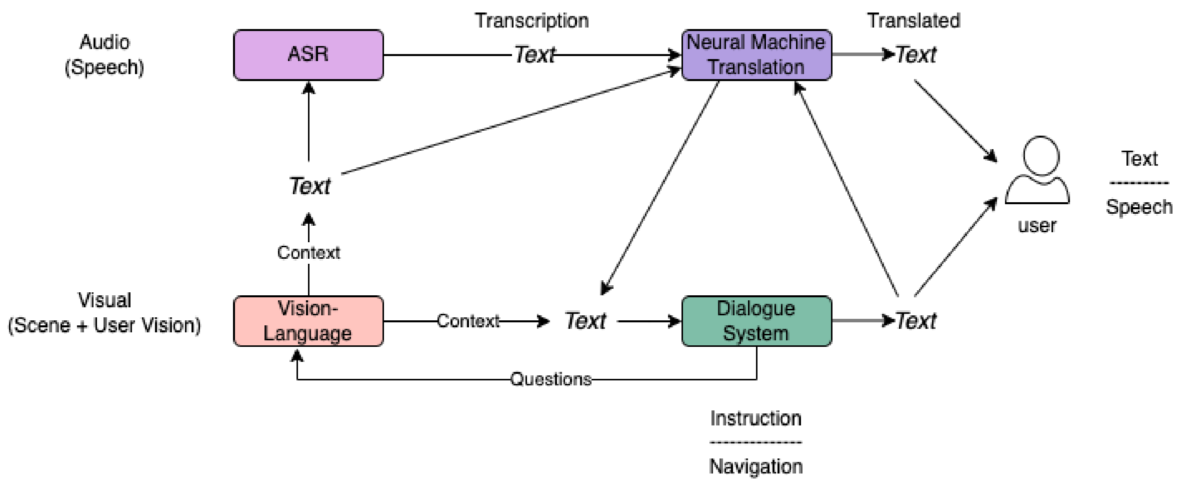


Figure 16: Overview of data flow between VOXReality components.

The various VOXReality components need to effectively communicate and exchange data to be integrated in a unified solution. Therefore, it is important to clearly define the input and output data format of each component. Moreover, it is crucial to determine the language/s that each component can operate with. The Table 1 summarizes this information.

Table 1: Input and output data format & operational language of VOXReality components.

Component	Input			Output		
	Data	Language	Format	Data	Language	Format
ASR	Audio	User's	wav	Text	User's	JSON
	Text (visual context)	English	JSON			
Error Correction	Text	User's	JSON	Text	User's	JSON
Neural Machine Translation	Text	User's	JSON	Text	Other user's or English	JSON
	Text (visual context)	English	JSON			
Vision Language Model	Image	N/A	jpg, png	Text (visual context)	English	JSON
	Text	English	REST POST			
Dialogue System	Text	English	JSON	Text	English	JSON
	Text (visual context)	English	JSON			

2.2 Process Diagrams of VOXReality Use Cases

This section provides the process diagrams of VOXReality use cases, highlighting the interactions among components. Table 2 presents for each use case the used VOXReality components. It is observed that at least two of the VOXReality components are utilized in every case, while the VR Conference contains all the components.

Table 2: VOXReality components used in each use case.

Use case \ Components	ASR	Neural Machine Translation	Vision language Models	Dialogue System
VR Conference	X	X	X	X
Augmented Theatres	X	X	X	
Training Assistant	X			X

2.2.1 VR Conference

The specific use case has as goal to provide language translation at a virtual conference and user navigation, that will be held in VR space. The language translation part of this use case can be characterized as user-to-user interaction since each user should be able to communicate with other users, that speak different language. The navigation experience can be identified as user-to-machine interaction since users will have the opportunity to assist the virtual agent in order to gain valuable information about the venue in VR space as well as the program of the conference events. For this reason, but also for better understanding of this use case, two process diagrams are presented, one for user-to-user and one for user-to-machine interaction.

User-to-user Interaction

The purpose of this case is to facilitate the communication between users who speak different languages, for instance user 1 speaks language 1 while user 2 speaks language 2. Users' language is any of the VOXReality operational language. Inputs are the user's speech and POV. Figure 17 shows the operations for processing the one user's input data and generate the translated output that is provided to another user.

Specifically, user 1 enters VR application via either computer or glasses. Audio data, which is the user's 1 speech, is processed by ASR module which produces the transcriptions. Additionally, the Error Correction component can produce an error-free version of those transcriptions. Then, the Neural Machine Translation accepts the (error-free) transcriptions and provides the translated version of the user's 1 speech. It should be mentioned that the Visual Language Models will provide visual context to ASR and Neural Machine Translation to enhance the accuracy of their performance. The translated output data is passed to VR application and the graphic overlays are created, which are demonstrated to user 2 via VR glasses of computer.

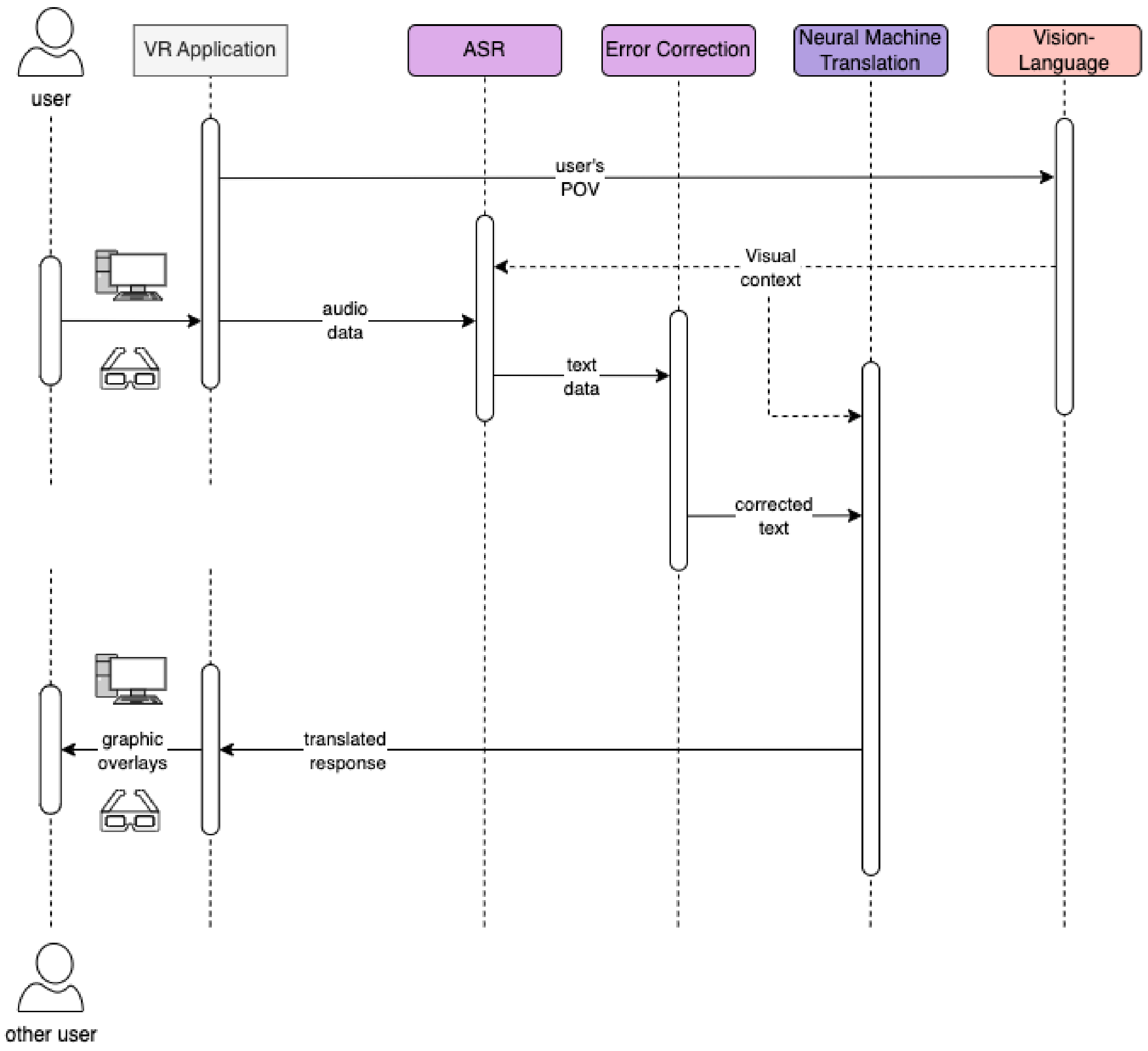


Figure 17: Process diagram for user-to-user interaction in VR Conference use case

User-to-machine Interaction

Regarding the user-to-machine interaction part of VR conference, the user requires to communicate with the system to derive useful knowledge about the VR venue and the program of events. Specifically, the user can ask the system about navigation instructions, programme-related details and questions about the surrounding place. Inputs in this case are user's speech and POV as well as the structural scene information derived from VR application. Figure 18 demonstrates the workflow of the components to build a successful user-to-machine interaction in VR conference use case.

The user enters the VR application and starts to communicate with the system. The user's POV is utilized by Vision Language Models that can provide visual context to ASR, Neural Machine Translation and Dialogue System. The user's speech, i.e., audio data, are processed by ASR and Error Correction modules to produce error-free transcriptions in the form of text. Afterwards, this information is passed to Neural Machine Translation component. If the user's language is English, then the Neural Machine Translation sent the text directly to Dialogue System. Otherwise, the transcriptions are converted to English by the Neural Machine

Translation component. The translated text is transmitted to Dialogue System which is able to assist the user with useful responses. Moreover, in the case that the user wants to be informed about its surrounding place by asking questions, the Vision Language models provides the relevant information to Dialogue system to accomplish the Question Answering task.

The Dialogue System will utilize the English transcription, the visual context, the structural scene information and predefined, external context to provide usefull explanations to the user. The predefined, external context, could be any materials which would provide specific responses the user, such as the programme, the venue map, etc. The output of Dialogue System is English textual response. If the user's language is not English, then the response will be translated to user's language.

At the end, the VR application will produce the graphic overlays which will be visualized by the VR glasses or computer to user. Those graphic overlays would contain the textual response of the system, visual cues and other interactive features that can help the user to better understand the response.

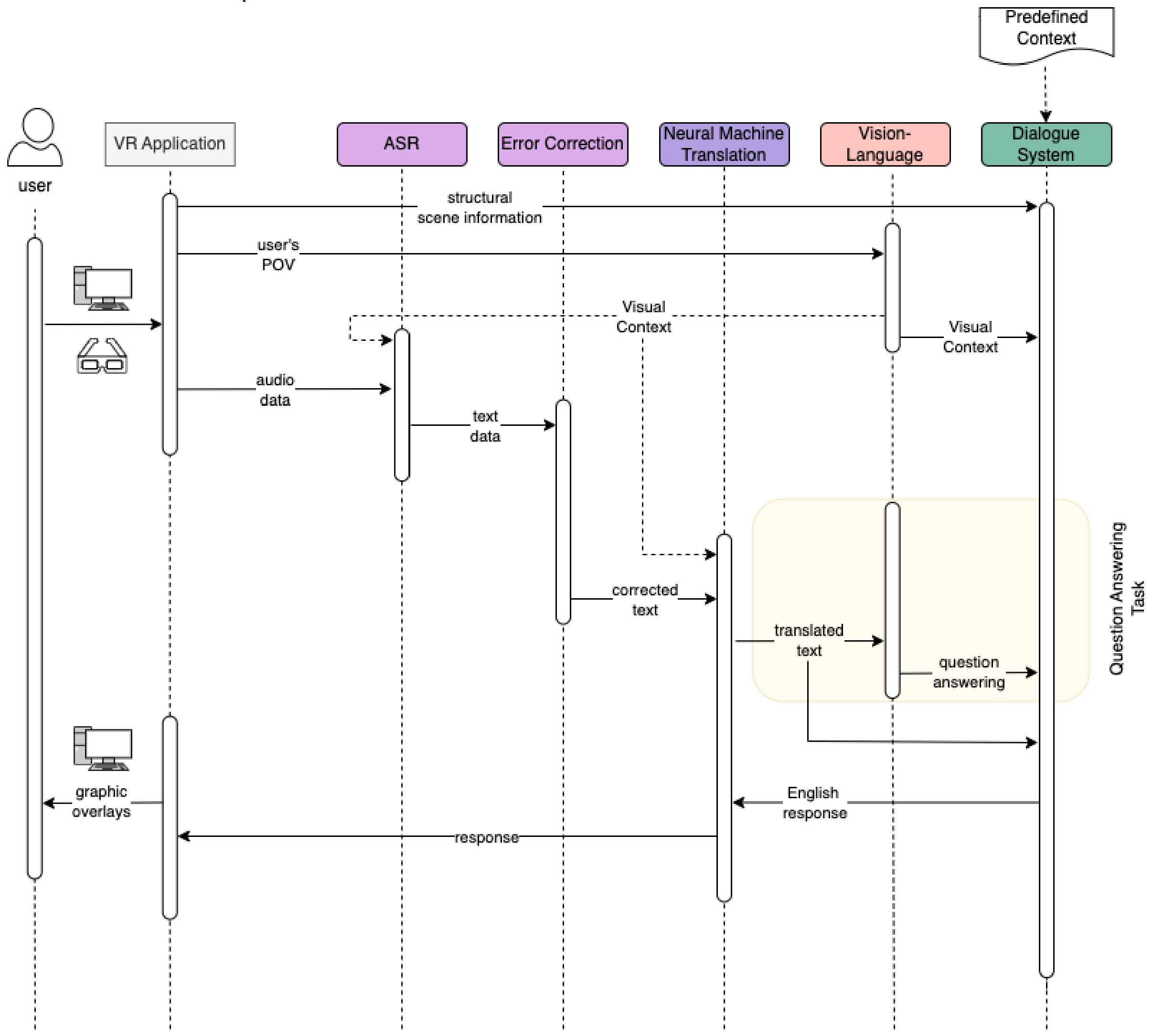


Figure 18: Process diagram for user-to-machine interaction in VR Conference use case

2.2.2 Augmented Theatres

In this use case, the VOXReality system will be used to enhance a theatrical play. This will be achieved by providing language translation and VFX experience to the user with AR glasses during the play. The inputs are both image recordings of the theater stage and audio recordings of each actor's and narrator's speech. Figure 19 presents the basic operations for processing audio and visual information by VOXReality components to generate immersive theatrical experiences.

The theater stage is continually captured by a fixed camera whose positioning will serve to provide unobstructed view to the scene extents. The scene view is processed by the Vision Language component which periodically provides descriptions of the scene, both semantic and spatial. The semantic information provided by the VL will be used to serve two needs. The first are accessibility needs through a feature, termed scene description, which generates automated textual descriptions of events happening in the scene. The second are theatrical literature needs through a feature, termed contextual information provision, which generates on demand background information about objects currently on stage. The background information for each object will be determined by the theatrical director. Finally, the spatial information can be used provisionally for the 3D placement of virtual object overlays, termed VFX.

Additionally, audio data, which is the actor's speech, is constantly recorded by each actor's fixed microphones. This data is processed by the ASR component, which generates the transcriptions. Moreover, the Error Correction component can produce an error-free version of those transcriptions. Afterwards, the Neural Machine Translation receives the (error-free) transcriptions and provides translated captions of an actor's speech. It should be mentioned that the Visual Language Models will provide visual context to ASR and Neural Machine Translation to enhance the accuracy of their performance.

The AR application receives input from the NMT and the VL and it is responsible to visualize both the VFX and captions to the user through AR glasses. The VFX will be triggered by a set of chosen keywords which will be scanned for in the transcript either as provided by the NMT (user preferred language) or the ASR (original language) and the scene description provided by the VL, and will be positioned based on predetermined positions and additional input from the VL. The VFX content and the exact conditions of their appearance and timing will be predetermined in conjunction with the theatrical partner.

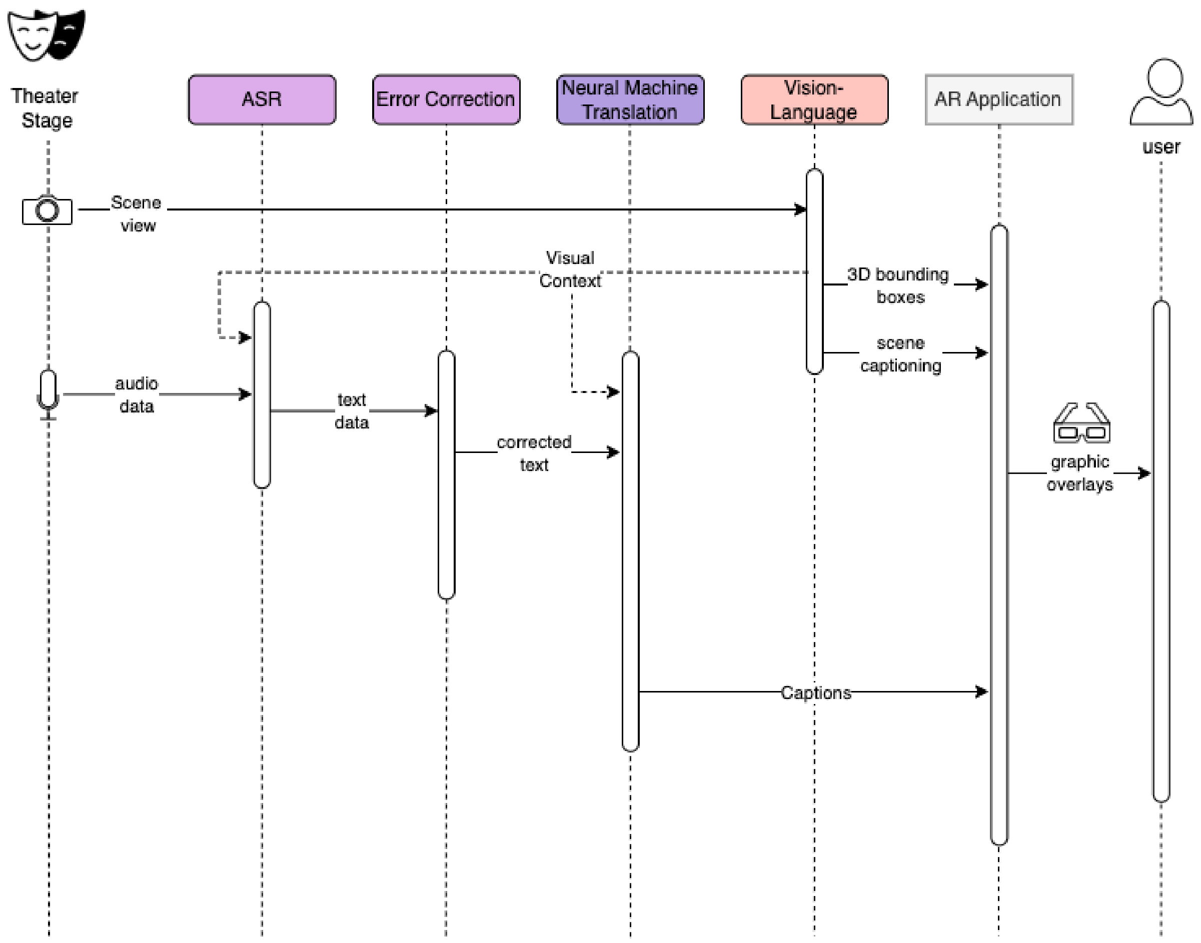


Figure 19: Process diagram of VOXReality components in augmented theatre use case.

2.2.3 Training Assistant

The purpose of this use case is to perform industrial machine assembly training using AR glasses. Specifically, users will assemble a virtual machine with virtual tools and objects in an Augmented Reality environment. Inputs in this use case will be user's audio data. Figure 20 describes the VOXReality operations for assisting a user in this specific task. The operational language of this use case is English.

The user's speech, i.e., audio data, are processed by ASR and Error Correction modules to produce error-free transcriptions in the form of text. Afterwards, the corrected text is transmitted to the Dialogue System which is able to assist the user with useful responses.

The Dialogue System will utilize the English transcription and the predefined external context to provide useful explanations to the user. The predefined, external, context could be any supporting materials which would provide specific responses about that use case to the user, such as instruction manuals, description of steps, etc. The output of Dialogue System will be English textual responses that can be used to assist the user to perform a specific task.

Eventually, the AR application will generate the graphic overlays which will be visualized via the AR glasses to user. Those graphic overlays would include the textual response, visual cues, and other interactive features to help trainees identify the correct tools and parts, navigate the assembly process, and troubleshoot any issues or errors that may arise.

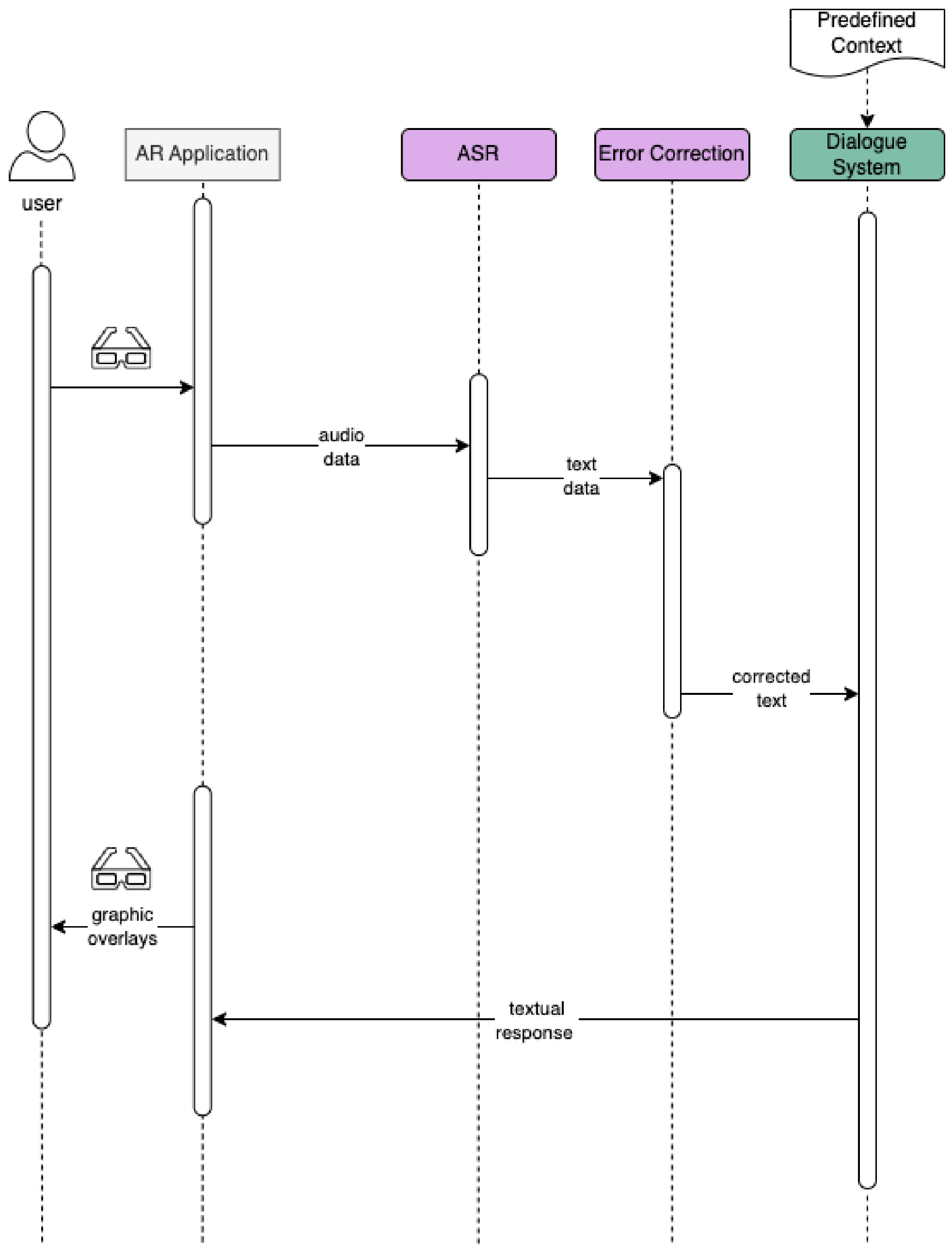


Figure 20: Process diagram of VOXReality components in training assistant use case

3 VOXReality Integration Guidelines

Common procedures that are applied during development may essentially enhance and accelerate the interaction and cooperation between development teams, as well as the integration of diverse software modules into a unified VOXReality system. The following subsections specify a set of suggested guidelines to be followed by VOXReality developers and, therefore, reinforce their interaction and communication.

3.1 Modular Software Design

Modular design and programming in software development focus attention on decomposing a program's functions into separate pieces or building blocks, each comprising all piece of code needed to implement a single attribute of the desired functionality [2]. Subsequently, integrating different software components together, the implementation formulates the system's complete range of capabilities.

Modular design is suitable for handling complexity and dependencies while dealing with remote working teams. The main concept of modular design refers to breaking down large and complex systems into simpler, smaller and more workable components. Modules present the appropriate methodology for abstracting arbitrary complexity, behind simple and straightforward interfaces. In this manner, functionality can be achieved incrementally, as the project advances.

In this respect, VOXReality will follow a modular design approach based on the following concepts:

- **Well-defined scope**: Each module must be well-defined and have a single, exclusive functionality. The implemented functionality should be as accurate and focused as possible, in order to avoid potential overlapping between the scopes of two or more software modules.
- **Precise interface/API**: In order to facilitate the usability of a module, the supported Application Programming Interface (API) should be defined and implemented in a crystal-clear, distinct and documented manner. Moreover, the interface needs to be comprehensive and minimal while it should cater for publishing to the system the predefined module's functionality avoiding possible misuse.
- **High degree of abstraction**: The coding specifics of a module should be private. Hence, modules should not reveal any functional details, interdependencies or their own data-structures. Moreover, the published API's functionality should be completely unrelated to the implementation details of a module, since updates in coding specifics should not influence API's functionality and therefore any of its clients.
- **Robust implementation**: The implementation mechanism of a module needs to be infallible, capable, well tested, and compact. For that reason, it is imperative that development procedure upholds best coding practices and exhaustive testing is executed.
- **Minimal interrelations**: he dependencies among modules augment the complexity of the integrated system. Therefore, an efficient modular system design minimizes the interactions between modules keeping the overall systems' complexity at low levels.

VOXReality adopts the modular design approach, for defining its architecture. Consequently, system functionality can be divided and assigned in components that will cover the requirements from different tasks and work packages. Consequently, modular design empowers different teams to develop in parallel their designated system segments, calling for the real-time cooperation between partner teams.

3.2 RESTful Architecture

A modern, widespread and standardized software architectural style for communication between computer systems is REST [3], which stands for Representational State Transfer. It utilises the synchronous HyperText Transfer Protocol (HTTP) in a stateless, client-server approach that facilitate the direct communication between the end-user point (client) and the backend point (server). RESTful API invokes the HTTP protocol's request types (POST, GET, PUT/PATCH, and DELETE) in order to implement user's requests for Create, Read, Update/Modify and Delete respectively.

The adoption of the RESTful architectural concept contains several notable advantages that are summarized below:

- **Lightweight:** Requires a limited bandwidth consumption, especially when JSON is selected for the RESTful API implementation.
- **Flexible and language independent**: Since it utilises the HTTP protocol it is format-agonistic and programmers can use XML, JSON, HTML, etc. Moreover, developers can easily utilise REST APIs without much processing overhead.
- **Scalable**: As a result of client server stateless communication, a module may be scaled up from the development team without much difficulty. Moreover, when the requested data from one of the queries are communicated properly, it is feasible to carry out a migration from the initial server to another or apply modifications on the database at any time without further complications. For instance, it is simple to employ different servers for hosting the front-end and the back-end of an application separately.

3.3 Documentation

Appropriate documentation expedites considerably development and integration activities. It can be regarded as a universal requirement for any project and should applied in every development and integration phase. With the help of software documentation, the functionality of software modules that were developed in the past can be easily comprehended from developers, even more if they have been written by someone else. Additionally, software documentation, substantially facilitates future updates, debugging and maintenance activities within module's code, where code "comments" help the programmer to understand the code's functionality and act accordingly. Summarizing, it is an excellent guideline to be applied during project's developing lifecycle.

3.3.1 Source Code Documentation

Working in collaboration with teams in different location and distributed code development responsibilities that can be modified over time, presents a high risk to generate source code that may be difficult to study and understand. If such a case occurs, it would considerably reduce the possibility of source code reusability, either to maintain or to extend/modify/update it. Consequently, developing code that is understandable and can be easily maintained by any team programmer or even from different teams forms a vital factor. A method to accomplish the above condition is to produce sufficient code documentation.

As software documentation can be characterized any written text or illustration that accompanies software code or is inserted inline in the source code. The documentation describes software's intended operation, how to use it, and may presents different meanings to people, according to their roles [4]. It is useful for the developers that update or debug program's code, as well as those who are willing to interface with it through API.

With the application of code documentation methodology, the development process will be enhanced with the following characteristics:

- **Knowledge sharing**: Usually plain code is not straight forward to understand and fails to provide the means for preserving the ideas, decisions and insights of the original programmers. In case that someone else is required to undertake a piece of software that was developed by someone else who is no longer member of the development team, meaningful code documentation can provide solutions regarding the transfer of programming logic to other team members.
- **Troubleshooting**: having useful documentation enables developers to gain a better understanding of code implementation specifics, speeding up the resolution process by quickly identifying the problematic code sections.
- **Extension**: Code documentation details dependencies between components/modules. This can be useful since it empowers the developers to make more informed decisions regarding the implementation of new functionalities or the integration with an external platform.

VOXReality developers are encouraged to perform appropriate source code documentation by following the guidelines below:

- **Make comments short and concise**: which can contribute to the readability of the comments, making them effective.
- **Journal the comments**: by including the name of the author and the date, any potential incompatibilities can be reported to the original developers.
- **Insert comments, above code**: comments should be positioned before the source code to which the developers provide documentation. While editing the code, comments may be displaced. If such a case occurs, the programmer should look after the text format and adapt comments position, correspondingly. According to this guideline, comments of functions should be inserted ahead of the function definition, explaining the logic of the function, along with the input and output parameters.
- **Maintain comments in sync with code**: While the maintenance/ update procedure is active, the source code keeps differentiate itself from previous versions, leading to the creation of inconsistences between the updated functionality of the code's current version and the content of the now obsolete comments. Since such discrepancies in source code documentation are hindering every code processing activity, consistent documentation should be realised by encouraging programmers to update comments in tandem with the respective source code.

3.3.2 API Documentation

API documentation can be regarded as a methodical explanation of the functionality that a specific API is publishing, along with appropriate information on how to instantiate this functionality. The purpose of providing such a documentation is to enable those that are going to use the respective API, to do so in a documented and secure manner, following the definitions and requirements of the API.

Meaningful API documentation, aside from the description of information regarding the process of testing, configuration and integration with the API, it should also analyse how such information will be provisioned in an easily accessible and usable manner. Since the procedure of maintaining an updated and consistent API documentation may become effort demanding, due to the continuous evolution of APIs supported functionality, a method that can be proved useful is the utilisation of online API documentation. Hence, VOXReality considers the adoption of OpenAPI Specification (OAS) [5].

As stated by the OpenAPI Specification [6] of the current version (3.1.0), OAS is a community-driven open specification under the umbrella of the OpenAPI Initiative, a Linux Foundation Collaborative Project. OAS defines a standard, programming language-agnostic interface description for HTTP APIs, such as the RESTful, that facilitates discovery and understanding of the service's capabilities, without source code knowledge, supplementary documentation or network traffic inspection. When a remote service is properly defined via OpenAPI, a "client" with a minimal amount of implementation logic, can understand and utilise the service successfully.

Well-known solutions for online documentation, following OAS, of RESTful APIs include Swagger [7] and apiary [8]. Moreover, online documentation can be facilitated with the help of API development frameworks, such as Django REST framework [9] or Spring Framework [10]. Regarding VOXReality implementation activities, partners' development teams are strongly recommended to document the developed APIs utilising online documentation tools (e.g. Swagger).

3.4 Open Source

Open source is considered as the source code that is made freely available for potential modification and redistribution [11]. Products that are benefit from open source code should include permission to use the source code, design documents or content of the product. Additionally, since open source grants permission for a piece of software to be used/reviewed/updated by any interested programmer, there is a solid opportunity to benefit from the knowledge of other developers as well as to participate to the group of potentially contributing developers and get involved in the further evolution of the aforementioned open-source code.

Open source and respective projects usually come with automated, asynchronous and lock-free workflows, making them a modern option. The constantly accelerating pace of technological progress, especially in the field of Information Technology (IT), led towards the adoption of open-source concept even from market dominant IT technological giants in order to actively participate, contribute and benefit from the new technological developments. Hence, IT technology leading corporations like Amazon Web Services, Google, IBM, Intel, SAP, Adobe, Microsoft etc. are significant contributors and active participants in many open-source projects and initiatives.

Finally, it should be mentioned that VOXReality components are anticipated to be available as Open Source, thus the appropriate course of action, regarding the application of standardized practices for the development of project's technical outputs, will be taken. Additionally, the VOXReality components will be meticulously and thoroughly documented provide information regarding each component's functionality, technical specification of utilised APIs, deployment information and instructions that describe its appropriate application.

4 Integration and Validation Methodology

This section analyzes the integration and validation methodology, including the roles involved in these processes and their responsibilities. Additionally, the integration and validation cycle as well as the various phases and the time-plan for these activities are presented. In VOXReality, the integration and validation methodology has as goal to guarantee that the delivered components are successfully implemented and integrated.

To better understand the integration and validation methodology of VOXReality, the outcomes of the project should be defined, which are:

1. Set of pre-trained AI models,
2. Codes and modules for building and use AI models, including inference codes that can be used with the VOXReality AI pre-trained models,
3. AI tools for:
    a. Sub-network extraction based on “once-for-all” training scheme,
    b. Finetuning and optimization

The developed VOXReality outcomes will be publicly available to third parties, including those who will participate in Open Calls. Specifically, those outcomes can be utilized by third parties to perform a variety of actions. For instance, third-party users can infer the pre-trained models using the appropriate inference codes, they can execute the AI tools and perform modification of the software codes to fit in their needs. Eventually, third-party users will be able to implement their own applications using the VOXReality components.

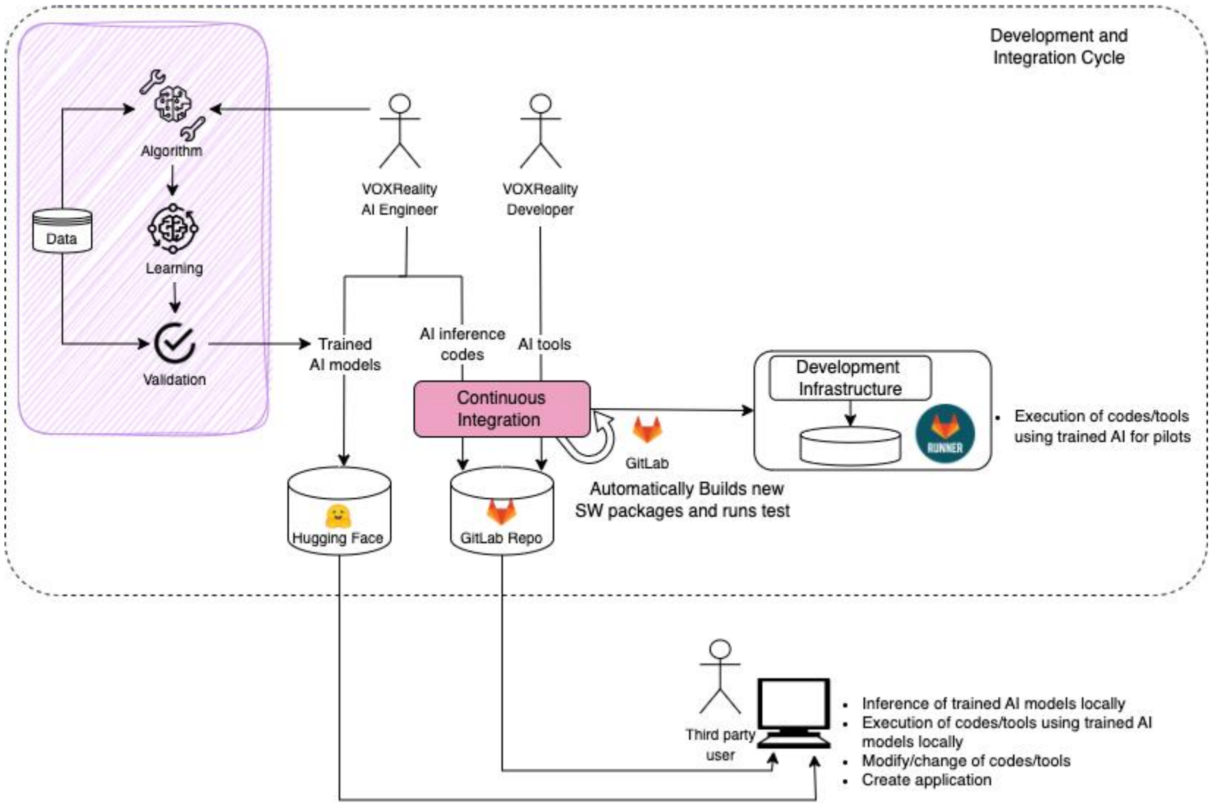
4.1 Roles and Responsibilities

The following roles have been identified in the integration and validation process:

1. **VOXReality development team**:
    a. AI Engineer, who is responsible to design and develop AI models and the suitable algorithms for the VOXReality project. Additionally, the AI engineer evaluates and optimizes the performance of those developed AI models. At VOXReality project, the AI engineer generates all the needed AI models as well as the codes that will be used to create those AI models and perform the inference of them.
    b. Software Developer, who are in charge of providing any software component needed for the project as well as to contribute to testing.
2. **Users**, who are the final actors of the deployed VOXReality applications. The various users will evaluate the VOXReality solutions through pilots, testing usability of the different functionalities as well as the user experience of the integrated system and reporting the test results.

4.2 Integration and Validation Cycle

This subsection describes the integration and validation activities and strategies of VOXReality AI models and software components. Figure 21 shows the validation and integration cycle.



Development and Integration Cycle
Algorithm
Data
Learning
Validation
VOXReality AI Engineer
VOXReality Developer
Trained AI models
AI inference codes
AI tools
Continuous Integration
Development Infrastructure
Execution of codes/tools using trained AI for pilots
GitLab
Automatically Builds new SW packages and runs test
Hugging Face
GitLab Repo
Third party user
Inference of trained AI models locally
Execution of codes/tools using trained AI models locally
Modify/change of codes/tools
Create application

Figure 21: Development and integration cycle

In VOXReality AI Engineers build robust, state-of-the-art AI models that can be used for ASR, multilingual translation, visual grounding of language models as well as in dialogue system for various tasks. To achieve this, some specific steps must be followed. Initially, it is important to gather the relevant data for each task. Those datasets will be accordingly processed and prepared, including data cleaning, data augmentation and feature engineering. The next step is to select the appropriate AI model architecture and parameters for the task and then to train the AI model using the processed data. After the model has been trained, it is validated on a separate set of data to evaluate its performance based on specific metrics.

The validation of AI models is a crucial step in the development and deployment of AI models, which guarantee that the model can make accurate predictions on new, unseen data. Through this process, issues, limitations, and biases in the models are identified while it can be ensured that the model will perform well on real-world data. To evaluate the performance of AI models, appropriate metrics will be defined. The choice of metrics depends on each specific task and the goal of the developed models. In VOXReality, the developed AI models will be validated at laboratory environment, using openly available dataset and the AI Engineer that creates the AI models is responsible to validate the model and to be sure that it can achieve satisfactory results. All this information about AI models validation, including the selected metrics and the achieved performances, will be provided at D3.1 "Advanced AI multi-modal for XR analysis V1" in M15 and at D3.2 "Advanced AI multi-modal for XR analysis V2" in M30.

The VOXReality AI Engineers and developers perform the above-described steps to train AI models at their own servers, utilizing their internal hardware and software resources. Once the processes of validation and refinement are done, the VOXReality AI Engineers will upload the trained models to Hugging Face[1], which is a platform for sharing and deploying natural language processing (NLP) models. Hugging Face provides a centralized location for developers and AI engineers to share and easily access pre-trained NLP models. When an AI model is uploaded to Hugging Face, it is available to other developers and AI Engineers to use it and build their own applications. Detailed information about the sharing of trained VOXReality models will be provided at the D4.1 “Model deployment analysis V1” at M17 as well as at the D4.2 “Model deployment analysis V2” at M32.

Additionally, software components will be implemented during the project. Specifically, software codes and modules that are used to build the AI models and to perform the inference of them will be developed by AI Engineers. Moreover, various AI tools will be created, focusing on the sub-network extraction based on “once-for-all” training scheme, finetuning of the trained VOXReality AI models as well as their subsequent optimization afterwards.

DevOps software development practices, such as Continuous Integration and Deployment, will be utilized to assure agile and smoothly development of components, integration and testing. In VOXReality project, it is decided to use GitLab as the main continuous integration tool and for software components versioning. A detailed description of the DevOps practices applied in VOXReality is provided in section 5. To achieve a successful development and integration of various components, laboratory testing could be performed. The aim of the tests is to identify issues as early as possible, to ensure compatibility, to improve reliability and at the end to enhance performance. Specifically, VOXReality will consider the following laboratory tests, which may include:

- **Unit testing** focuses on verifying the functionality of individual components of the software system. Those kinds of tests are typically automated and are designed to check if the tested component behaves as expected under various conditions and inputs. The purpose of those tests is to identify defects as early as possible in the development process. It may be performed in isolation from the rest of the system which varies based on the stage of the software development process. In VOXReality, separate unit tests will be planned and executed in each technical work package and the owner of each component will be responsible to create the unit tests.
- **Integration testing** aims to test the interactions and interfaces between different components of the system. The purpose of those tests is to verify that the various components of the system can work together as expected without any errors or defects. Therefore, the focus of those tests is on the integration points between different components. Typically, the integration tests are performed after separate components have been tested in isolation using unit tests.
- **Performance testing** evaluates the performance and responsiveness of a software system under various workload and stress conditions. The purpose of those tests is to identify bottlenecks and other performance issues of the system as well as to determine its capacity and scalability. To perform those tests, it is needed to run a series of tests and simulations to measure how well the system under different conditions, which in the case of VOXReality those conditions can be heavy load, many users connected in the applications, extended use. The tests may measure various aspects of performance, such as response time, resource usage.

[1] https://huggingface.co/

Pilots test the system in real-world environment and gather feedback from the users to identify any issues or areas for improvement of the XR applications but also to collect information about their overall experience. Specifically, the aim of VOXReality pilots is to validate the usability and the user experience of XR applications. Specifically, the usability of XR application refers to how user-friendly and effective the application is in providing an immersive experience within XR space. Regarding the user experience of XR applications, this term refers to the overall quality of the user's interaction, within XR space.

In VOXReality, two pilot phases will take place. During the pilots, the developed system is provided to a limited number of users for evaluation and feedback. The users are asked to use the system in realistic setting and then to give their feedback in some questionaries. In VOXReality, two pilots are planned to be performed, the 1st during M19-20 and the 2nd during M31-M35. The scenarios of pilots will be presented with details at D2.4 "Organisational preparation for VOX pilot scenarios and PRESS analysis V1" in M10 and at D2.5 "Organisational preparation for VOX pilot scenarios and PRESS analysis V2" in M26.

Finally, the third parties' users can access the publicly available VOXReality code using the GitLab and the pre-trained VOXReality AI models via Hugging Face. Specifically, they can access the public repositories of GitLab and download the codes, while they can find and use the corresponding AI models through Hugging Face. Those research outputs, both VOXReality software codes and AI models can be modified and used by third parties to develop their own applications. During VOXReality project, third parties will use the VOXReality outcomes via Open Calls to further extend the use cases and the application domains.

In VOXReality open calls phase, third parties' users will utilize and validate the components, providing feedback about their functionalities. During this phase, it is possible for the VOXReality technical partners to perform refinements of software components based on the feedback. Additionally, the evaluation remarks and observations from third parties' users will be contained and discussed in the D6.3 "Lessons learned, success stories and exploitation plan update" in M36. In the case of unsolved issues from the validation of components during the open calls phase, the D6.3 will present possible reasons that may have contributed to this happening, potential solutions as well as other recommendations and actionable suggestions in order to avoid undesired behaviors.

4.3 Integration and Validation Phases and Time-plan

The VOXReality project will be executed over a period of 36 months, organized in three (3) phases, following a spiral iterative development process. VOXReality's phased approach is presented in Figure 22, with each one feeding into the next to continually refine the project's results and activities. During the project, two pilot phases will be carried out. The VOXReality components will be ready prior to the beginning of each pilot phase. The initial version of VOXReality components (ver. 0) will be released within the second phase and especially after the end of first pilot phase. The final version of components (ver. 1) will be released within the third phase, after the end of second and final pilot. The three phases that combine and structure these activities efficiently in order to reach the desired outcomes are the following:

- **Phase I – Design**: The first phase of the project spans over the first ten months and will focus on the early design activities of the research activities as well as the challenges as defined by the use cases that they will seek to address. The requirements will be analysed in order to retrieve the technical and systemic requirements for each data-driven model and the envisaged application, which will then drive the design and lead to the definition of the VOXReality models' architecture and data formats. In this phase, also the specific test scenarios, in which new technology will be validated, and the goals of these tests will be defined. Finally, within this phase the technical team of VOXReality will set-up the integration and development environment (i.e. tools, source code repositories, issue reporting, etc.) that will be used during the project's lifetime.
- **Phase II – Core Developments**: The second phase of the project lastes ten months and focuses on the core development of the VOXReality models. Within this phase, the initial version of VOXReality models are developed and delivered. Before delivery, the models will be technically verified and validated on a functional, technical and workflow basis by the VOXReality technical team. At the end of this phase, the first set of trials and validations will be performed by end-users, including functionality, and user experience tests for both the VOXReality models and deployed applications.
- **Phase III – Maturation**: The final, third phase of VOXReality, will span the last sixteen months of the project and will include the refinements and improvements of the VOXReality models and delivered applications, based on the feedback received from the first validation phase. Following the paradigm of second phase, a thorough technical assessment of the models and applications will be conducted before their final validation with end-users in the second pilot for ensuring their robustness and usability. The phase will conclude with the final version of VOXReality models, XR applications as well as final pilots.

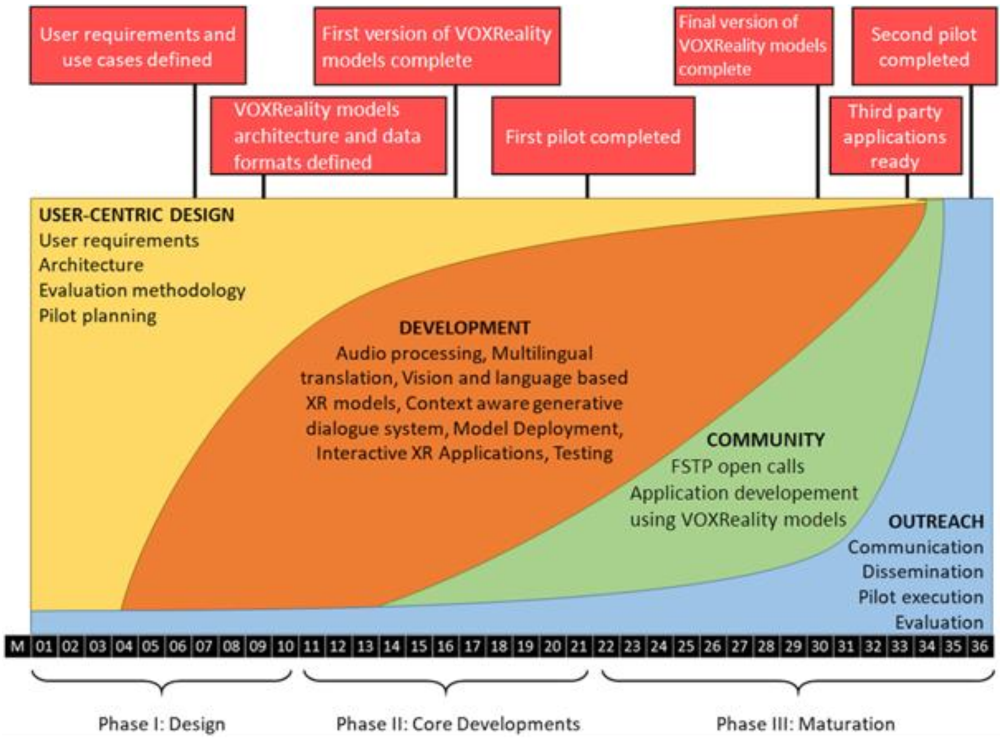

User requirements and use cases defined
First version of VOXReality models complete
Final version of VOXReality models complete
Second pilot completed
VOXReality models architecture and data formats defined
First pilot completed
Third party applications ready
USER-CENTRIC DESIGN
User requirements
Architecture
Evaluation methodology
Pilot planning
DEVELOPMENT
Audio processing, Multilingual translation, Vision and language based XR models, Context aware generative dialogue system, Model Deployment, Interactive XR Applications, Testing
COMMUNITY
FSTP open calls
Application developement using VOXReality models
OUTREACH
Communication
Dissemination
Pilot execution
Evaluation
M 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36
Phase I: Design
Phase II: Core Developments
Phase III: Maturation

Figure 22: VOXReality's phased approach

Overall, two major development cycles are foreseen, each one composed of requirement collection, technology design, research and development activities and user testing and evaluation. Smaller iterative steps will complement the two major cycles, following a SCRUM[2] methodology procedure that will reduce the overall time of development and enable timely problem resolution, while supporting the regular communication of progress and outcomes within the consortium.

5 Development Infrastructure and Integration Tools

VOXReality adapts today's modern business environments. Specifically, VOXReality recognizes that it is imperative for product development teams to maintain an optimal workflow. Having a well-tuned workflow not only keeps a team productive, but it also helps them deliver software that is reliable and in a timely manner. Therefore, VOXReality implements DevOps processes which are critical for the involved project's teams to maintain the outcome of the project. To this end, VOXReality uses multiple environments to ensure that the resulted solution is rigorously tested before it is deployed and made available to users. These multiple environments consist of the following:

- ***Development Environment:*** It will be the first line of defence against bugs for VOXReality and the deployed vertical solutions. This environment will allow VOXReality developers to deploy their code and test any newly implemented features. Any bugs found will be dealt with before re-deploying for further testing. The process will be iterated until the code is ready for the end users.
- ***Production Environment:*** Once the code will be thoroughly tested, it will be then deployed to production where it will be made available to end-users.

5.1 DevOps Approach

According to the traditional software development lifecycle (SDLC) it is common for different software teams to develop code working in isolation from the operators that cater for the maintenance of the system's execution environment managing servers and applications. Developers focused almost exclusively on how to implement the software system that would fulfil the user requirements while operators centred their attention on avoiding any rapid changes without proper safeguards that could destabilize the operation of the production system. With the arrival of modern DevOps practices the above segmentation is changed.

DevOps can be defined as a collection of cultural philosophies, practices, and tools that boost an organization's ability to deliver applications and services at an accelerated rate, outpacing organizations that implement and improve products using traditional software development and infrastructure management processes. Consequently, with the adoption of DevOps strategy, organizations can build, test and release software much faster and safer than before, while keeping customers satisfied and compete more effectively in the market.

[2] https://www.scrum.org/

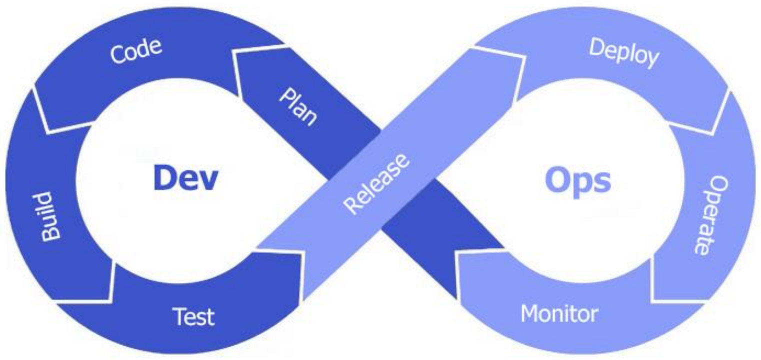


Figure 23: DevOps Lifecycle Stages

In order to illustrate DevOps lifecycle procedure, the process can be divided into phases which form the DevOps pipeline as illustrated in Figure 23. Specifically, the tools and processes used throughout the various stages can be described as follows:

- **Plan**: The stage covers everything that happens before the developers start writing code. Requirements and feedback from stakeholders and customers is used to build a product roadmap to guide future development.
- **Code**: The code development takes place at this stage. The development teams use tools and plugins to streamline the development process, eluding security flaws and lousy coding practices.
- **Build**: Subsequent to the coding phase completion, the development team requests the merging of the new code with the shared codebase. The request triggers an automated process which builds the codebase.
- **Test**: Following the successful completion of the previous stage, the build is deployed to the test environment, where a series of manual and automated tests like user acceptance test, security test, integration testing, performance testing are performed.
- **Release**: By this stage, the build is ready for deployment into the production environment. Since the build passed the testing procedure, the operations team schedules updates or sends several versions to production, according to the organizational requirements.
- **Deploy**: The build is released into production environment.
- **Operate**: The operations team administers hosting configuration in order to make the new version accessible to users. Moreover, provisions for the collection of user feedback information are utilized.
- **Monitor**: At the "final" stage, the operations team monitors data collected from customer behavior, application performance, errors along with potential bottleneck in the DevOps workflow and feedbacks appropriate input to the development team initiating a new lifecycle iteration.

One of the major benefits of DevOps adoption is the improved communication between development and operations teams. This leads towards faster and more efficient deployment processes, as well as improved quality and accuracy of software. Furthermore, DevOps can minimize the amount of time spent on manual tasks, which can increase resource availability for more important tasks. Additionally, to the above, there is number of benefits from DevOps application such as:

- Predictability: Significantly lower failure rate for new releases.
- Maintainability: Effortless recovery from new release crashing since earlier versions can be restored as needed.

- Quality: Incorporating infrastructure issues can enhance software development quality.
- Resiliency: Software system is more stable, secure, and changes are auditable.
- Time to market: Streamlined software delivery significantly scales down time to market period.
- Cost-efficiency: Improved cost-efficiency during software development.

On the other hand, one of the most important challenges regarding DevOps application is the achievement of close collaboration between different teams of the consortium. Developers and IT professionals must join forces to implement and deploy software, while ensuring that the software meets the needs of the customer. Additionally, DevOps teams must ensure successful software maintenance and update procedures throughout the whole product development lifecycle.

5.2 CI/CD

Continuous Integration (CI) is a developer technique for code integration while maintaining a working system through the whole process. The methodology incorporates small changes to the system by integrating frequently (usually at least daily) on the mainline by means of appropriate tools supporting automation with lots of automated tests. This empowers different developer groups to work on shared code and boosts the awareness regarding the development status and quality of the system. Furthermore, Continuous Integration (CI) usually counts on developers to implement Test-Driven Development (TDD) with constant refactoring practice. When a developer is unit-test-driving his/her code, it is made certain that the local software copy implements the specified requirements.

Continuous Deployment (CD) abstracts to the automated deployment of every updated -release- version of an under-development system to the production environment. Successive to the continuous integration practice that is described above, when a system release advances to a certain maturity level, certified by specific, predefined conditions, CD gets to work at updating the previously deployed running version of the system automatically, minimizing upgrading downtime.

Integrated together, CI/CD formulates a pipeline that accepts new developments and deploys an updated running version of a system in the designated execution environment. In VOXReality a CI/CD environment is already organized in GitLab. Figure 24 displays the GitLab's CI/CD pipeline in an abstract level.

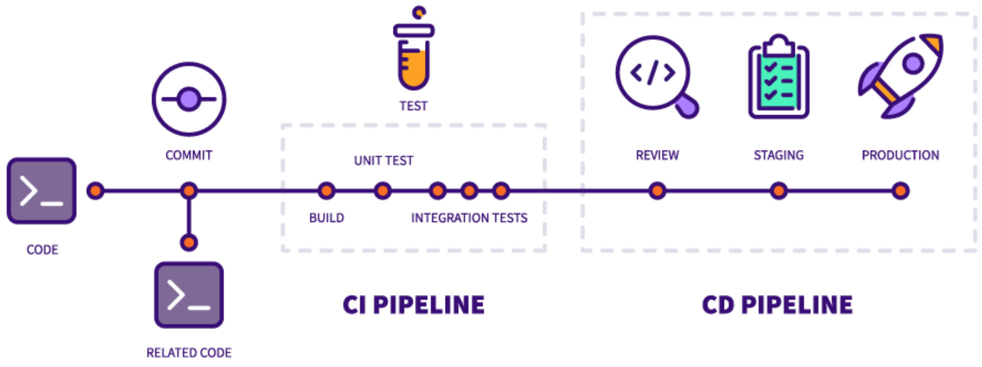


Figure 24: General CI/CD pipeline – Source: about.gitlab.com

The suggested CI/CD pipeline operation is clarified in more detail in Figure 25. More specifically the main phases of the GitLab's CI/CD procedure that is detailed in the figure, can be described as follows:

- A particular piece of software is completed and wishes to be integrated to a module functionality.
- The developer specifies unit tests for the relevant piece of code regarding the evaluation of the modules' output results.
- The above-mentioned code developments are committed from their local repository to a branch of remote GitLab repository that is a dedicated development branch defined in the CI infrastructure.
- The developer applies for code merging.
- The CI/CD pipeline procedure, for the specific project, is triggered. The preconfigured unit test(s) are performed from CI platform.
- In case of successful unit tests, the merge request is accepted, and the newly committed source code gets part of the code main branch.
- In any other case, the merge request is rejected signalling the implementation of appropriate code updates in order to fully satisfy unit and system testing procedures.
- The process advances to the CD part, that builds and deploys the package.
- Following the successful CD phase, the new system version is running on the development infrastructure and is subjected to user testing/production.

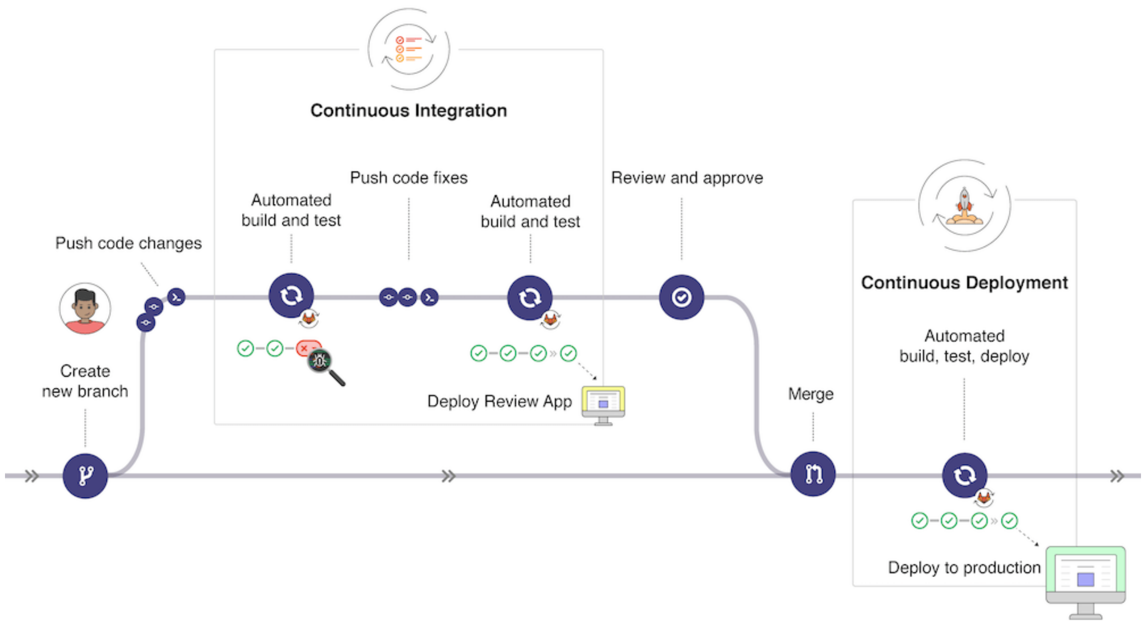


Figure 25: VOXReality GitLab's CI/CD pipeline steps - Source: docs.gitlab

Focusing to CD part of pipeline, if all the tests are passed successfully, the code is considered ready to be deployed. The CD part of pipeline is responsible to automate the packaging and to prepare the software components for deployment. In VOXReality, the outcomes of this step are docker images, which are pushed to the VOXReality DockerHub. Then, the docker containers are deployed to development server.

Moreover, the VOXReality CI/CD pipeline has been further enhanced towards security processes. Specifically Static Application Security Testing (SAST) is applied. This type of testing is used to check the code without its execution, allowing the developers to find security vulnerabilities in the source code early in the early stages of the development. The overall VOXReality pipeline is illustrated in Figure 26.

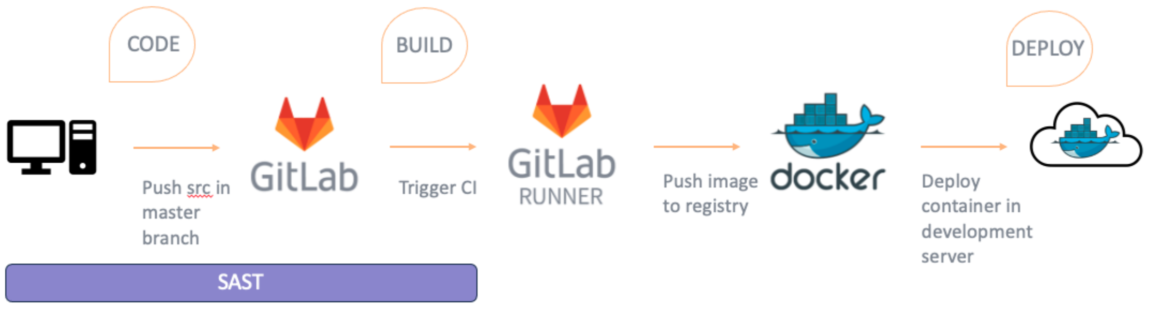


Figure 26: VOXReality CI/CD pipeline

5.3 Source Code Management

Source code management (SCM) is the procedure of tracking source code modifications. Maintaining an updated log of the modifications applied to the common codebase enables programmers, developers and testers to verify that they are always working with appropriate and up-to-date code, while enabling fast conflict resolution during code integration from diversified development teams. SCM practice is also known as Version Control System (VCM).

The most prevalent VCM is Git[3] which is an open-source distributed software for tracking changes in any compilation of files. Web platforms, originating from Git management software, that are available for hosting project source codebases, include GitLab[4] and GitHub[5], that support Git repository management as well as code reviews, issue tracking, wikis etc. The two aforementioned platforms above offer a complete solution for software management in a single stand-alone web platform that provide for projects' entire software development lifecycle. Regarding VOXReality, source code management will be based on Git, the underlying system behind GitLab, which is defined as the suitable CI/CD platform for the project.

5.4 Containers

Containerization introduces an Operating System (OS)-level virtualization that simplifies the deployment and execution of distributed applications. Containers differentiate from a "traditional" virtual machine solution since they virtualize, as far as the operating systems (OS) layer, while VMs, through a hypervisor, virtualize also physical hardware. Consequently, they offer a lightweight alternative that involves encapsulating an application in a container with its own operating system, they are portable and may hosted reliably from different computing environments, while multiple containers may share the same OS kernel. The term "container" comes from the logistics field where a packaging container refers to a large, metallic box that

[3] http://git-scm.com/
[4] https://about.gitlab.com/
[5] https://github.com/

"encapsulates" smaller crates, designed for easy and fast loading and unloading during transportation or storage.

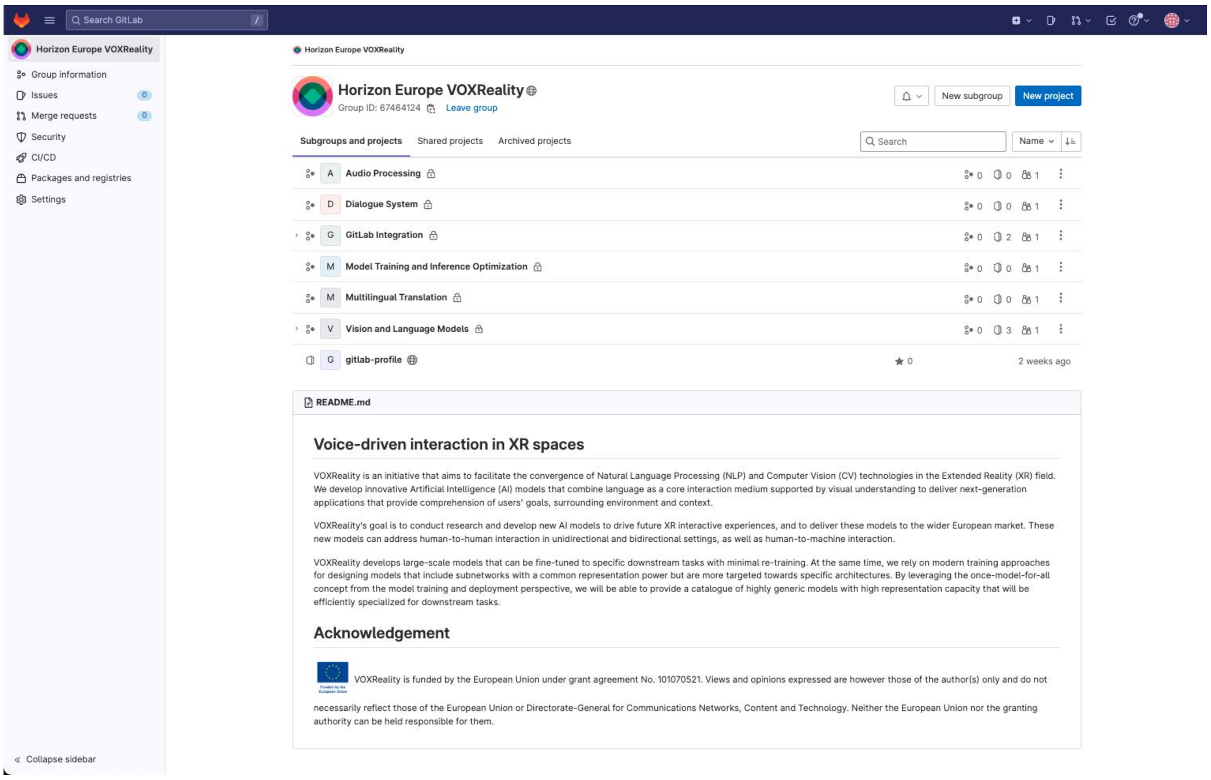
Among the first and most popular containerization technology applications today is Docker [6]. Docker follows the platform as a service (PaaS) concept, that uses OS-level virtualization to deliver software in packages. A Docker container image is a lightweight, standalone, executable package of software that incorporates everything (code, runtime, system tools, system libraries and settings) needed to run an application [12]. Moreover, open-source systems for automating deployment, scaling, and management of containerized applications, suitable for hosting Docker containers, like Kubernetes (K8s) [13] are available for application.

VOXReality conforming to the modular architecture scheme that is described above, selects Docker for the project's containerization framework and strongly advice the technical developers of the project to utilize containers for the containerization of their implemented software modules. Eventually, VOXReality development infrastructure will host the containerized modules of the project.

5.5 VOXReality CI/CD Framework Instantiation

The GitLab CI/CD framework has been set up and configured within the public GitLab instance for the VOXReality project. The official GitLab group of VOXReality is entitled "Horizon Europe VOXReality" and is publicly accessible at https://gitlab.com/groups/horizon-europe-voxreality. Figure 27 provides a snapshot of the VOXReality's GitLab repo. The group contains 6 subgroups that represent the main thematic entities that will be developed in the VOXReality project.

[6] https://www.docker.com



Audio Processing	0	0	1	
Dialogue System	0	0	1	
GitLab Integration	0	2	1	
Model Training and Inference Optimization	0	0	1	
Multilingual Translation	0	0	1	
Vision and Language Models	0	3	1	
gitlab-profile		★ 0		2 weeks ago

README.md

Voice-driven interaction in XR spaces

VOXReality is an initiative that aims to facilitate the convergence of Natural Language Processing (NLP) and Computer Vision (CV) technologies in the Extended Reality (XR) field. We develop innovative Artificial Intelligence (AI) models that combine language as a core interaction medium supported by visual understanding to deliver next-generation applications that provide comprehension of users' goals, surrounding environment and context.

VOXReality's goal is to conduct research and develop new AI models to drive future XR interactive experiences, and to deliver these models to the wider European market. These new models can address human-to-human interaction in unidirectional and bidirectional settings, as well as human-to-machine interaction.

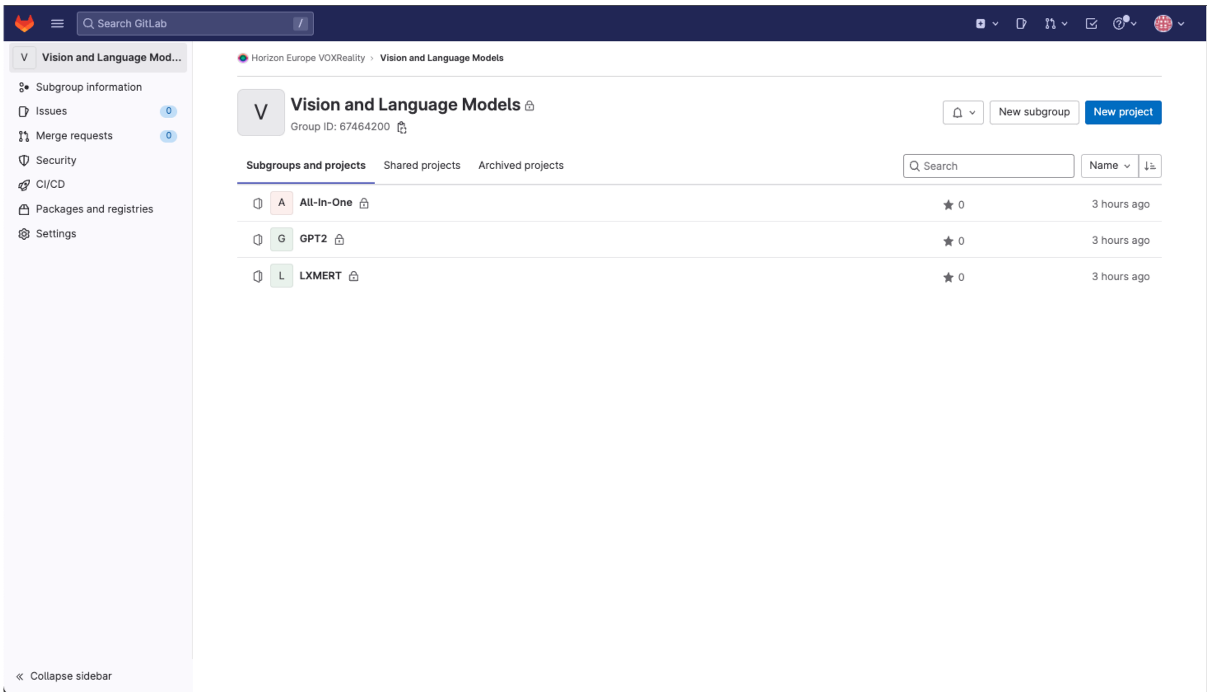
VOXReality develops large-scale models that can be fine-tuned to specific downstream tasks with minimal re-training. At the same time, we rely on modern training approaches for designing models that include subnetworks with a common representation power but are more targeted towards specific architectures. By leveraging the once-model-for-all concept from the model training and deployment perspective, we will be able to provide a catalogue of highly generic models with high representation capacity that will be efficiently specialized for downstream tasks.

Acknowledgement

VOXReality is funded by the European Union under grant agreement No. 101070521. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or Directorate-General for Communications Networks, Content and Technology. Neither the European Union nor the granting authority can be held responsible for them.

Figure 27: VOXReality GitLab Home page

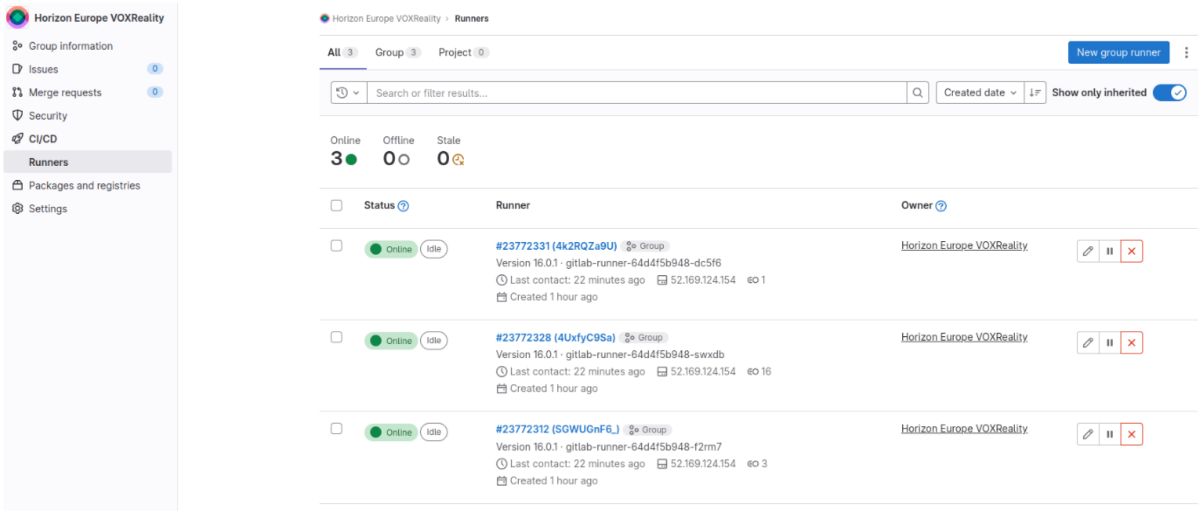
Within each subgroup, the development activities are organized in separate subgroups according to relevant tasks, which is a way to structure and organize repositories within a large project. Specifically, Figure 28 shows the content of the subgroup "Vision Language Models".



All-In-One	★ 0	3 hours ago
GPT2	★ 0	3 hours ago
LXMERT	★ 0	3 hours ago

Figure 28: VOXReality Vision and Language Models Subgroup.

VOXReality has configured the CI/CD pipeline that enables the automation of integration and deployment procedures using the GitLab CI/CD platform. The CI/CD pipeline is operated by three GitLab runners, which have been deployed and integrated in the development server. Figure 29 shows the GitLab runners on VOXReality. More information about the development infrastructure is provided in Section 5.6.

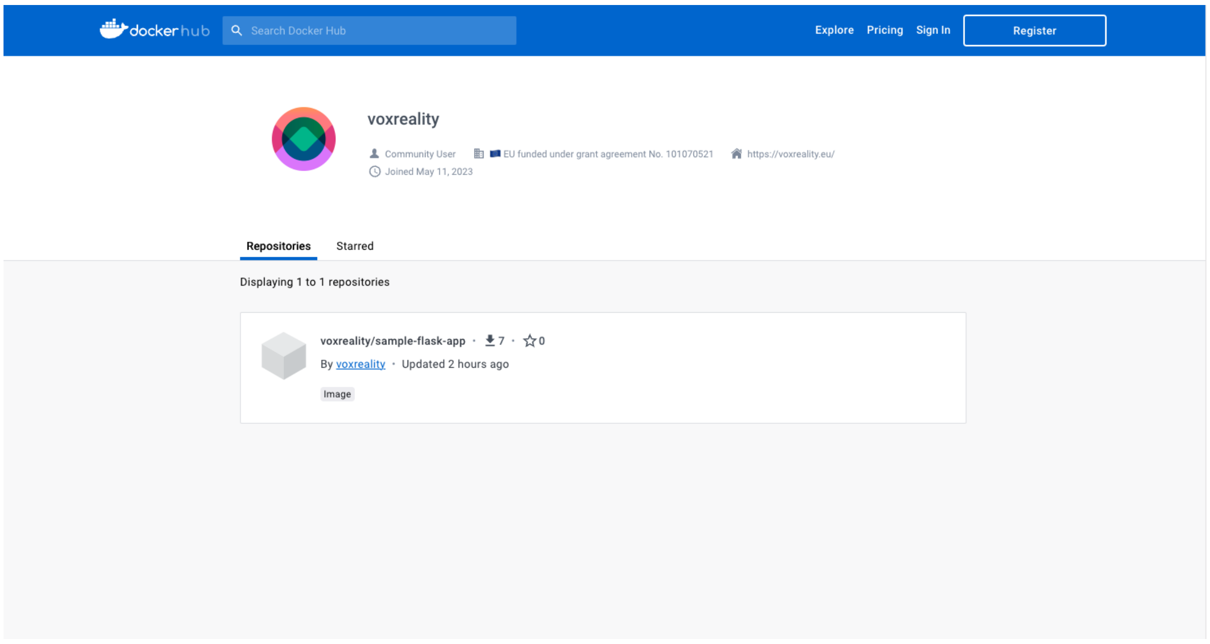


Online 3 ● Offline 0 ○ Stale 0

Status	Runner	Owner
Online Idle	#23772331 (4k2RQZa9U) Group Version 16.0.1 · gitlab-runner-64d4f5b948-dc5f6 Last contact: 22 minutes ago 52.169.124.154 1 Created 1 hour ago	Horizon Europe VOXReality
Online Idle	#23772328 (4UxfyC9Sa) Group Version 16.0.1 · gitlab-runner-64d4f5b948-swxdb Last contact: 22 minutes ago 52.169.124.154 16 Created 1 hour ago	Horizon Europe VOXReality
Online Idle	#23772312 (SGWUGnF6_) Group Version 16.0.1 · gitlab-runner-64d4f5b948-f2rm7 Last contact: 22 minutes ago 52.169.124.154 3 Created 1 hour ago	Horizon Europe VOXReality

Figure 29: VOXReality GitLab runners

VOXReality docker images are created based on the uploaded source code in GitLab and they are stored to the VOXReality docker repository, which has been established for the needs of the project. Guidelines will be provided to assist developers to trigger this procedure, while a sample application has been created in the VOXReality GitLab that can be used as example. The VOXReality profile in the DocherHub repository is available at https://hub.docker.com/u/voxreality. Figure 30 illustrates a screenshot of the VOXReality profile.



voxreality

Community User · EU funded under grant agreement No. 101070521 · https://voxreality.eu/
Joined May 11, 2023

Displaying 1 to 1 repositories

voxreality/sample-flask-app · 7 · 0
By voxreality · Updated 2 hours ago
Image

Figure 30: VOXReality docker image repository.

5.6 VOXReality Development Environment

VOXReality development infrastructure is utilized by consortium partners to validate and test the trained AI models as well as the developed AI tools. The infrastructure is based on the Microsoft Azure[7], which is a cloud computing platform with a significant and continuously expanding set of services that help DevOps operators to build solutions according to their business goals. Azure services support everything from simple to complex, based on simple web services for hosting the business presence in the cloud. It also supports running fully virtualized computers, enabling operators and developers to manage their custom software solutions. Apart from that, Azure provides a wealth of cloud-based services like remote storage, database hosting, and centralized account management, as well as new capabilities like artificial intelligence (AI) and Internet of Things (IoT) focused services. Azure currently offers more than 200 products and cloud services designed to help organizations to bring new solutions to life – to solve today's challenges and create the future. In this regard, it supports them to build, run, and manage applications across multiple clouds, on-premises, and at the edge, with the tools and frameworks of your choice.

The requirements defined by the VOXReality technical partners for the development purposes, set the route towards a selection of infrastructure with the following characteristics. These initial requirements were identified in a "pay-as-you-go" plan, to allow for further changes later, if required. Those requirements are presented in Table 3.

Table 3: Development server characteristics.

Category	Description
OS / Software	Ubuntu Linux
CPU	4 Cores
RAM	28 GB
Temporary Storage	180 GB
GPU	1 x T4

In general, the technical partners, that are developing the AI models and the various tools, considered as critical to base the development in a series of virtual machines (with four virtual CPUs) accompanied with a GPUs with at least 16GB of memory. The synthesis presented in the Table 3 is considered as ideal to run Machine Learning and Artificial Intelligence workloads utilizing CUDA, TensorFlow, PyTorch, Caffe and other frameworks. This selection allows the use of different operation systems (Windows, iOS, Linux, etc) to support effectively and efficiently the technical partners selections. Moreover, this selection providers adequate network configuration properties for the development of the VOXReality development services (bandwidth, protocols, configurations, etc.)

[7] https://azure.microsoft.com

6 Conclusion

Well-defined and meticulous planning is a prerequisite for successful project implementations. Projects that involve multiple teams working on various subsystems in different locations, such as VOXReality, require well-organized procedures to efficiently handle complex development tasks. Therefore, it is vital to carefully coordinate parallel development efforts and to facilitate the integration of subsystems into a unified system to accelerate their progress.

This document presents a description of VOXReality components as well as the process diagrams of each use case, describing the main interactions among them and specifying representative generic processes that will be implemented through the project. Moreover, the integration guidelines for the VOXReality project are presented, including recommendations for the interfaces' technologies and proper code, as well as API documentation, consistent to the open-source support of the project. Additionally, VOXReality's DevOps practices are introduced, in order to broaden the level of automation and therefore reduce the risk, as well as the time and effort required for integrating individual software modules, while facilitating proper methodologies within the integration and deployment process. Finally, the development environment is presented.

7 References

[1] VOXReality, "Definition and Analysis of VOXReality Use Cases V1," 2023.

[2] D. Davis, J. Burry and M. Burry, "Untangling parametric schemata: enhancing collaboration through modular programming," *Proceedings of the 14th international conference on Computer Aided Architectural Design, University of Liege, Liege,* 2011.

[3] C. Pautasso, E. Wilde and R. Alarcon, REST: advanced research topics and practical applications, Springer, 2013.

[4] A. Forward and T. C. Lethbridge, "The relevance of software documentation, tools and technologies: a survey," in *Proceedings of the 2002 ACM symposium on Document engineering*, 2002, pp. 26-33.

[5] OpenAPI, "OpenAPI Specification," OpenAPI, [Online]. Available: https://swagger.io/specification/. [Accessed April 2023].

[6] GitHub, "The OpenAPI Specification," [Online]. Available: https://github.com/OAI/OpenAPI-Specification. [Accessed April 2023].

[7] SmartBear, "Swagger," [Online]. Available: https://swagger.io/solutions/api-documentation/. [Accessed April 2023].

[8] Oracle, "apiary," [Online]. Available: https://apiary.io/. [Accessed April 2023].

[9] Django REST framework, "Django REST framework," [Online]. Available: https://www.django-rest-framework.org/. [Accessed April 2023].

[10] "Spring Framework," [Online]. Available: https://spring.io/. [Accessed April 2023].

[11] Wikipedia, "Open source," [Online]. Available: https://en.wikipedia.org/wiki/Open_source.

[12] Docker, "What is a Container?," Docker, [Online]. Available: https://www.docker.com/resources/what-container. [Accessed April 2022].

[13] kubernetes, "Production-Grade Container Orchestration," The Linux Foundation , 2020. [Online]. Available: https://kubernetes.io. [Accessed 12 2020].

VOXReality

Voice driven
interaction in XR spaces

Funded by
the European Union