

Project Acronym: HosmartAI
Grant Agreement number: 101016834 (H2020-DT-2020-1 – Innovation Action)
Project Full Title: Hospital Smart development based on AI



This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 101016834

DELIVERABLE

D4.1 – Platform Architecture Design and Open APIs

Dissemination level:	PU -Public
Type of deliverable:	R -Report
Contractual date of delivery:	31 January 2022
Deliverable leader:	ITCL
Status - version, date:	Final – v1.0, 2022-01-31
Keywords:	Platform architecture, Open APIs

This document is part of a project that has received funding from the European Union's Horizon 2020 research and innovation programme under agreement No 101016834. The content of this document reflects only the author's view and the European Commission is not responsible for any use that may be made of the information it contains.

The document is the property of the HosmartAI consortium and shall not be distributed or reproduced without the approval of the HosmartAI Project Coordination Team. Find us at www.hosmartai.eu.

Executive Summary

This document presents the first version of the HosmartAI Platform Architecture and Open APIs. The conceptual architecture from D1.5 and T1.3 outputs are analysed, identifying common layers between pilots, and transforming the requirements into elements to be added to the final architecture.

Also, data inputs and outputs for each pilot are analysed and a common API is defined to standardize the flow of information between the pilots and the platform. This part of the project is focused on the identification of all the key components of the platform and how the interconnection is made.

Deliverable leader:	Sergio Chico, Daniel Lozano (ITCL)
Contributors:	Sergio Chico, Daniel Lozano (ITCL), Pauline Loygue (GC), Luca Gilardi (EXYS)
Reviewers:	Makis Karadimas, Giannis Sarantidis (INTRA), Pauline Loygue (GC)
Approved by:	Athanasios Poulakidas, Irene Diamantopoulou (INTRA)

Document History			
Version	Date	Contributor(s)	Description
0.1	2021-09-23	Sergio Chico (ITCL)	Document creation
0.2	2021-10-13	Daniel Lozano, Sergio Chico. (ITCL)	First version of sections for the ToC, Executive Summary.
0.3	2022-01-14	Pauline Loygue (GC) Daniel Lozano, Sergio Chico (ITCL) Luca Gilardi (EXYS)	Review Introduction, Architecture elements description and diagram, APIs-OpenAPIs, corrections. Data security and privacy (Section 2.2)
0.4	2022-01-21		Reviewed, Pre-Final Version
1.0	2022-01-31	Athanasios Poulakidas, Irene Diamantopoulou (INTRA)	Final version for submission

Table of Contents

Executive Summary.....	2
Table of Contents.....	4
List of Figures	5
List of Tables	6
Definitions, Acronyms and Abbreviations	7
1 Introduction	8
1.1 Project Information	8
1.2 Document Scope	10
1.3 Document Structure.....	10
2 Requirement’s analysis	11
2.1 Platform requirements.....	11
2.1.1 Introduction	11
2.1.2 AI Software integration.....	16
2.1.3 Blockchain integration	17
2.1.4 The HosmartAI Frontend	19
2.1.5 Data integration	21
2.2 Data security and privacy	22
2.2.1 Security requirements	22
2.2.2 Data protection methods.....	24
2.2.3 Traceability of the information	29
2.3 Edge Cloud.....	30
2.4 Open API.....	31
2.4.1 API Documentation	31
2.4.2 API Specification	32
2.4.3 API Definition	32
2.4.4 API Visualization.....	32
2.4.5 Third party OpenAPI platform elements	33
3 HosmartAI Architecture Design	39
3.1 Principal elements.....	39
3.2 HHub.....	47
3.2.1 Marketplace	47
3.2.2 Co-creation Hub	48

3.2.3	Benchmarking	52
3.2.4	Dashboard	56
4	Conclusion	57
5	References	58

List of Figures

Figure 1:	Conceptual Architecture.	11
Figure 2:	HosmartAI Platform.	14
Figure 3:	Benchmarking service.	15
Figure 4:	AI Development flow.	17
Figure 5:	Blockchain service architecture.	18
Figure 6:	Health Level Seven Standards.....	21
Figure 7:	Edge Cloud infrastructure description.....	31
Figure 8:	OpenAPI YAML example.	33
Figure 9:	API Visualization tool (Swagger).	33
Figure 10:	JupyterHub OpenAPI.....	34
Figure 11:	Discourse OpenAPI.....	35
Figure 12:	Slack OpenAPI.	36
Figure 13:	Acumos LUM OpenAPI (YAML).	37
Figure 14:	Acumos LUM OpenAPI (Swagger).....	38
Figure 15:	HosmartAI Architecture Diagram.	39
Figure 16:	Docker infrastructure.....	41
Figure 17:	Ansible infrastructure.	42
Figure 18:	Nginx infrastructure.	43
Figure 19:	Acumos AI platform.	44
Figure 20:	Keycloak data flow.	46
Figure 21:	Consul infrastructure.	47
Figure 22:	JFrog Artifactory Infrastructure.	50
Figure 23:	SonarQube Infrastructure Pipeline.	51
Figure 24:	Jenkins Pipeline.....	52
Figure 25:	JupyterHub infrastructure.	54
Figure 26:	Kafka infrastructure.	55

List of Tables

Table 1: The HosmartAI consortium.	9
Table 2: List of needs and related requirements for data protection.	24
Table 3: List of needs and related requirements for data selection.	25
Table 4: List of needs and related requirements for data verification.	26
Table 5: List of needs and related requirements for authentication and access control.	27
Table 6: List of needs and related requirements for event management.	28
Table 7: List of needs and related requirements for traceability of the information.	29
Table 8: Docker requirements.	40
Table 9: Ansible requirements.	41
Table 10: Nginx requirements.	43
Table 11: Acumos requirements.	44
Table 12: ROS requirements.	45
Table 13: Keycloak requirements.	45
Table 14: Consul requirements.	46
Table 15: Drupal requirements.	47
Table 16: .NET Blazor requirements.	48
Table 17: Discourse requirements.	49
Table 18: JFrog Artifactory requirements.	49
Table 19: Slack requirements.	50
Table 20: SonarQube requirements.	51
Table 21: Jenkins requirements.	52
Table 22: Python Benchmarking libraries requirements.	52
Table 23: Benchmarking database driver requirements.	53
Table 24: JupyterHub requirements.	53
Table 25: Kafka requirements.	55
Table 26: MongoDB requirements.	56
Table 27: Angular requirements.	56

Definitions, Acronyms and Abbreviations

Acronym/ Abbreviation	Title
API	Application Programming Interface
DoA	Description of Action
EHR	Electronic Health Record
HHub	HosmartAI Hub
HL7	Health Level 7
HL-FHIR	Health Level 7 – Fast Healthcare Interoperability Resources
KPI	Key Performance Indicator
RAF	Reference Architecture Framework
SME	Subject Matter Expert
WP	Work Package

Term	Definition
Consortium	Group of beneficiaries that have signed the Consortium Agreement and the Grant Agreement (either directly as Coordinator or by accession through the Form A).
Consortium Agreement	Contractual document signed by all the beneficiaries (and not the EC), explaining how the Consortium is managed and works together.
Deliverable Leader	Responsible for ensuring that the content of the deliverable meets the required expectations, both from a contractual point of view and in terms of usage within the project. Is also responsible for ensuring that the deliverable follows the deliverable process and is delivered on time.
Description of Action	Annex 1 to the Grant Agreement. It contains information on the work packages, deliverables, milestones, resources and costs of the beneficiaries, as well as a text with a detailed description of the action. The DoA is made of Part A (structured data collected in web forms and workplan tables) and Part B (text document describing the action elements).
Dissemination	EC term for communication of information to a wide audience.
Grant Agreement	Contractual document which defines the contractual scope of the HosmartAI project. It is signed between the EC and the beneficiaries.

1 Introduction

1.1 Project Information



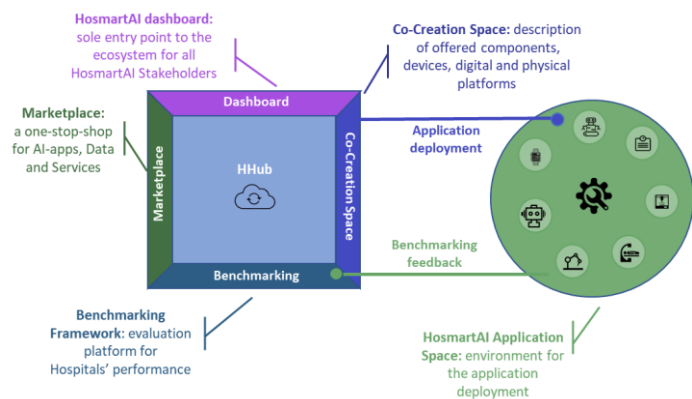
The HosmartAI vision is a strong, efficient, sustainable and resilient European **Healthcare system** benefiting from the capacities to generate impact of the technology European Stakeholders (SMEs, Research centres, Digital Hubs and Universities).



The HosmartAI mission is to guarantee the **integration** of Digital and Robot technologies in new Healthcare environments and the possibility to analyse their benefits by providing an **environment** where digital health care tool providers will be able to design and develop AI solutions as well as a space for the instantiation and deployment of AI solutions.

HosmartAI will create a common open Integration **Platform** with the necessary tools to facilitate and measure the benefits of integrating digital technologies (robotics and AI) in the healthcare system.

A central **hub** will offer multifaceted lasting functionalities (Marketplace, Co-creation space, Benchmarking) to healthcare stakeholders, combined with a collection of methods, tools and solutions to integrate and deploy AI-enabled solutions. The **Benchmarking** tool will promote the adoption in new settings, while enabling a meeting place for technology providers and end-users.



Eight Large-Scale Pilots will implement and evaluate improvements in medical diagnosis, surgical interventions, prevention and treatment of diseases, and support for rehabilitation and long-term care in several Hospital and care settings. The project will target different **medical** aspects or manifestations such as Cancer (Pilot #1, #2 and #8); Gastrointestinal (GI) disorders (Pilot #1); Cardiovascular diseases (Pilot #1, #4, #5 and #7); Thoracic Disorders (Pilot #5); Neurological diseases (Pilot #3); Elderly Care and Neuropsychological Rehabilitation (Pilot #6); Fetal Growth Restriction (FGR) and Prematurity (Pilot #1).

To ensure a user-centred approach, harmonization in the process (e.g., regarding ethical aspects, standardization, and robustness both from a technical and social and healthcare perspective), the

living lab methodology will be employed. HosmartAI will identify the appropriate instruments (KPI) that measure efficiency without undermining access or quality of care. Liaison and co-operation activities with relevant stakeholders and **open calls** will enable ecosystem building and industrial clustering.

HosmartAI brings together a **consortium** of leading organizations (3 large enterprises, 8 SMEs, 5 hospitals, 4 universities, 2 research centres and 2 associations – see Table 1) along with several more committed organizations (Letters of Support provided).

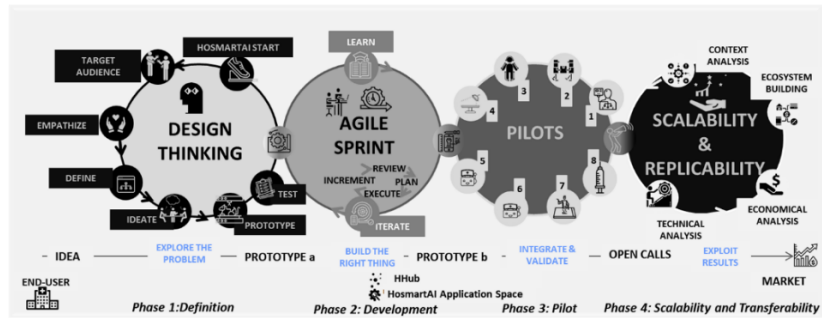


Table 1: The HosmartAI consortium.

Number ¹	Name	Short name
1 (CO)	INTRASOFT INTERNATIONAL SA	INTRA
1.1 (TP)	INTRASOFT INTERNATIONAL SA	INTRA-LU
2	PHILIPS MEDICAL SYSTEMS NEDERLAND BV	PHILIPS
3	VIMAR SPA	VIMAR
4	GREEN COMMUNICATIONS SAS	GC
5	TELEMATIC MEDICAL APPLICATIONS EMPORIA KAI ANAPTIXI PROIONTON TILIATRIKIS MONOPROSOPIKI ETAIRIA PERIORISMENIS EYTHINIS	TMA
6	ECLEXYS SAGL	EXYS
7	F6S NETWORK IRELAND LIMITED	F6S
7.1 (TP)	F6S NETWORK LIMITED	F6S-UK
8	PHARMECONS EASY ACCESS LTD	PhE
9	TERAGLOBUS LATVIA SIA	TGLV
10	NINETY ONE GMBH	91
11	EIT HEALTH GERMANY GMBH	EIT
12	UNIVERZITETNI KLINICNI CENTER MARIBOR	UKCM
13	SAN CAMILLO IRCCS SRL	IRCCS
14	SERVICIO MADRILENO DE SALUD	SERMAS
14.1 (TP)	FUNDACION PARA LA INVESTIGACION BIOMEDICA DEL HOSPITAL UNIVERSITARIO LA PAZ	FIBHULP
15	CENTRE HOSPITALIER UNIVERSITAIRE DE LIEGE	CHUL
16	PANEPITIMIAKO GENIKO NOSOKOMEIO THESSALONIKIS AXEPA	AHEPA
17	VRIJE UNIVERSITEIT BRUSSEL	VUB
18	ARISTOTELIO PANEPITIMIO THESSALONIKIS	AUTH
19	EIDGENOESSISCHE TECHNISCHE HOCHSCHULE ZUERICH	ETHZ
20	UNIVERZA V MARIBORU	UM

¹ CO: Coordinator. TP: linked third party.

Number ¹	Name	Short name
21	INSTITUTO TECNOLÓGICO DE CASTILLA Y LEON	ITCL
22	FUNDACION INTRAS	INTRAS
23	ASSOCIATION EUROPEAN FEDERATION FOR MEDICAL INFORMATICS	EFMI
24	FEDERATION EUROPEENNE DES HOPITAUX ET DES SOINS DE SANTE	HOPE

1.2 Document Scope

The deliverable aims to provide the first version of the HosmartAI architecture, based on the outcome of the D1.5 and the analysis of the technical requirements for each pilot. Also, this deliverable provides other key elements, like security, data privacy and interoperability methods to complete HosmartAI architecture.

1.3 Document Structure

This document is comprised of the following chapters:

Chapter 1 presents an introduction to the project and the document.

Chapter 2 analyses the requirements and defines the architecture features.

Chapter 3 explains and describes the final version of the HosmartAI architecture.

Chapter 4 provides some concluding remarks.

2 Requirement’s analysis

2.1 Platform requirements

2.1.1 Introduction

With the aim of creating a solid, functional and useful platform to accomplish all the proposed features of the external elements, information from other tasks of the project will be gathered and an analysis of the platform technical requirements and platform related components will be done.

For that, the design of the first version of the architecture is based on the diagram of the conceptual architecture, that has the following elements:

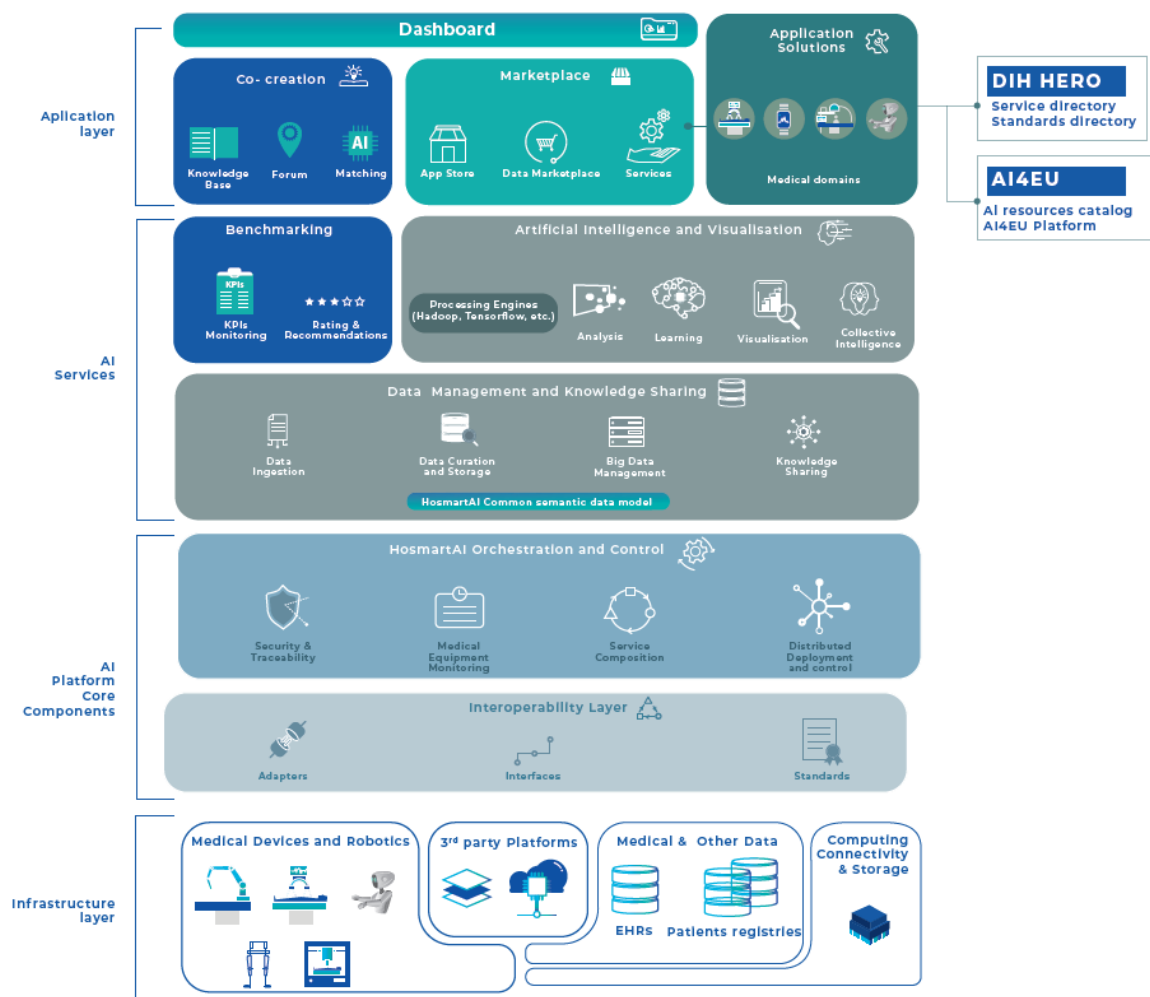


Figure 1: Conceptual Architecture.

During the development of this task, changes to the architecture will be made as necessary to fulfil all the new requirements discovered in previous tasks.

2.1.1.1 *Application layer*

2.1.1.1.1 *Marketplace*

The HosmartAI Marketplace includes all components, devices, services, data sources, platforms, etc. that are available for exploitation. The project's marketplace will support the development of applications with access to technologies, but also with training material, documentation, and other useful data.

The user interface displayed in the dashboard will be a Web-based catalogue of items, where each item will have a name, a description, and some additional specific information fields.

The different items can be Components, devices, services, data sources, platforms, datasets, etc.

Feedback from end users through the benchmarking tool should be able to be used in the marketplace for sorting/filtering/recommendation.

The benchmarking process is specific for each application, employing a predefined set of variables that are representative of the concrete case of use, and producing a particular set of measurements. Therefore, the comparisons of two resources should only be possible when using the same input variables.

Other filters can be applied to search through items, like filter items by name, type, benchmarking results, etc.

The HosmartAI Marketplace will be able to pull resources from the AI4EU project that will complement the already available ones. Also, the marketplace will provide information about already available services in digital innovation hubs such as DIH-HERO.

One of the most important assets to support the development and proper execution of AI algorithms is the specific data set to be used by third party developers in Open Calls. The HosmartAI Marketplace will provide those as well as the ability for stakeholders coming from both the demand side and the supply side to participate.

The marketplace must meet certain characteristics to fulfil the objectives set by the project. It must be able to serve any service or application in a reasonably low time and in a way that is understandable for the system that is making the request.

The connection needs are high, since it will have many requests and high information traffic, so the connection channels must be wide.

2.1.1.1.2 *Co-creation*

The HosmartAI Co-creation Space is used during the co-creation process, in which healthcare stakeholders, service advisors and providers can select together the most appropriate set of tools, devices, components and data sources. To do this, they can also examine the existing resources and their performance through the HosmartAI Benchmarking Framework. Many of those can be already deployed at hospitals, primary care centres and care homes and thus provide valuable information that can be used to set expectations and improve goals.

The Co-creation Space can be used for the set of standards, APIs, interoperability mechanisms and data models that will be employed in the process of creating a new HosmartAI enabled application, some examples of co-creation platforms that can be incorporated into the HosmartAI platform can be AI4EU (Acumos), GAIA-X or OPEN DEI.

2.1.1.1.3 Dashboard

The HosmartAI Dashboard will provide the main entry point to the HosmartAI platform.

The Dashboard will be accessed via web browser. The interface will be device-independent, making it able to access the system with different web browsers, and different devices with different Operative Systems, both mobile and desktop.

The technologies used to develop the dashboard will be ASPX.NET (Devexpress XAF), Blazor pages (C# .NET/WebAssembly) and PHP (for WordPress or other content management system).

Security should be a priority on the dashboard, as this is the place where the users interact with the platform. Security elements like privacy, data integrity, authentication and authorization should be taken into consideration in the development of the platform. Authentication and authorization can be addressed by using JWT tokens.

The Dashboard architecture will be decoupled, offering communication with different modules via gRPC services, a REST API or both. This underlying API layer will map the functions of the benchmarking, marketplace, and co-creation to be presented to the users.

The Dashboard will also offer access to each user's personal account and allow them to perform administration tasks to customize their participation to Co-creation, Benchmarking and Marketplace.

2.1.1.1.4 Application solutions

Application solutions that will be developed will be able to combine different types of tools and frameworks for their main functionality regarding its inner workings, but also interfaces to interact with end-users.

Here we have the different interfaces for users to operate the different applications. We include here the scheduler interface for example. As well as other interfaces like the ones used for the application catalogue, recommendations, or other purposes. All of them will be hosted by the HosmartAI platform.

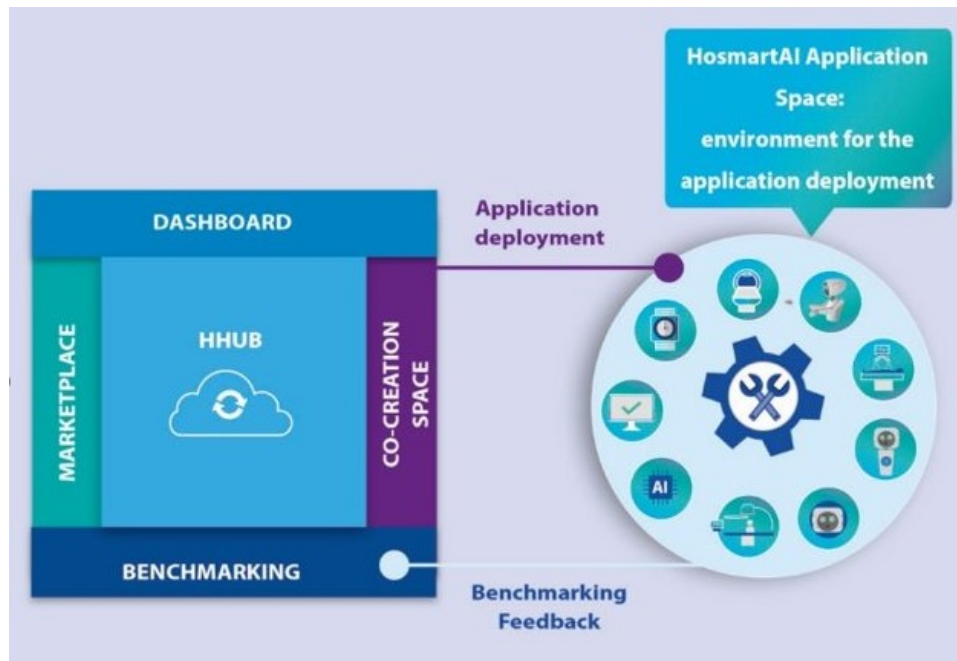


Figure 2: HosmartAI Platform.

2.1.1.2 AI Services

2.1.1.2.1 Benchmarking

The HosmartAI Benchmarking Framework assessment will help identify ways to improve healthcare services and applications through establishing measurable indicators and putting feedback mechanisms in place.

Due to the necessity of processing a large amount of data that the project presents, along with the specific needs of visualization and data mining, Apache Spark will be the solution. This is a system that allows the execution of large amounts of calculations in parallel, distributed among several nodes, and is specially designed for the study of relationships between variables, their filtering, and the application of algorithms.

The system needs Python for executing all the tasks, so will include pyspark library also to work along Apache Spark. For this integration, Anaconda is the right tool to provide necessary Python implementation and a series of libraries that allow certain calculations to be carried out in parallel within the processors, which will also be useful in the project development. In addition, Anaconda allows the development and implementation of a wide variety of ML algorithms.

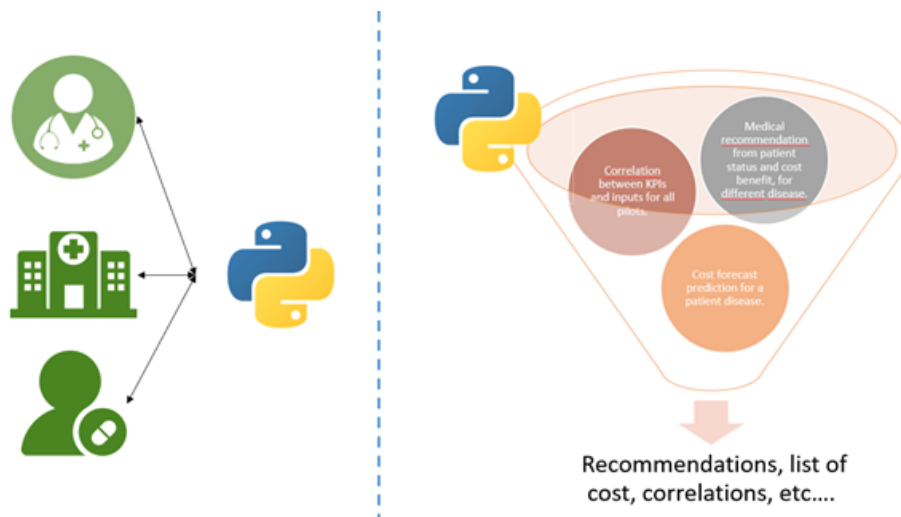


Figure 3: Benchmarking service.

Finally, Mongo database is needed for the data storage necessary for the execution of our codes. MongoDB is not resource intensive, at the same time it can deal with relatively large amounts of data, which makes it well suited for the need of the platform.

2.1.1.2.2 Artificial Intelligence and Visualization

Artificial Intelligence algorithms can be used to automate image processing and analysis of medical data to produce an augmented visualization of data to include analysis results in a way that makes it easy to interpret by medical practitioners. One example is the algorithm to support AI-augmented echocardiography applications and automate image processing and quantitative analysis of Left Ventricular (LV) function.

All the models and datasets can be onboarded within the platform and will be offered in the application catalogue so that they can be used by anyone who need the information to do their tasks.

2.1.1.2.3 Data Management and Knowledge Sharing

The goal of Data Management will be that of addressing and enabling Big Data, as well as facilitating knowledge sharing through data translation mechanisms that use common standards. An example of this is the development of tools to map the electrical activity of the heart by companies that record ECG signals with the goal of integrating with various sources and standardizing the data set across different models and device types.

2.1.1.3 AI Platform core components

This layer consists of:

- A large set of adapters that enable the integration and interoperability of applications for smart hospitals. The specifications for these adapters include:
 - o Common interfaces, for example based on OpenAPI
 - o Data pipelines utilizing the publish-subscribe pattern to distribute information
 - o Common standards

- Security and traceability services enabling privacy preservation that will be embedded throughout the entire operational system lifecycle
- Common tools abstracting the medical device monitoring functionalities
- Tools to compose a microservices-based application
- Tools and environment to deploy and control the microservices-based applications

2.1.1.4 Infrastructure layer

This layer includes the infrastructure, such as the following:

- Assets that are used to store data as well as train and execute AI algorithms. The data might originate from Electronic Health Records (EHR), Hospital Information Systems (HIS) or other sources of non-clinical data, such as lifestyle and demographics.
- Medical devices and robotics.
- Added value third-party platforms which can offer high-level APIs, extra features, and libraries.
- Other computing, connectivity, and Big Data storage infrastructure.

2.1.1.4.1 Computing, Connectivity and Storage

Data collected by medical devices might need to be stored in cloud-based servers to support telehealth consultations. Also, the data from many different devices need to be backed up in a common cloud to avoid information lost in case of hard disk corruption.

2.1.2 AI Software integration

AI tools and algorithms will be available for development through JupyterHub or Acumos AI Platform.

Through **JupyterHub**, a user can create a server that will be able to execute Python code. Commands can also be directed to the system console to install additional libraries, such as `"pip install --extra-index-url https://partner.com/pypi partner-library1"`. This is done by using the special character `"%"` at the front of the command to be redirected to the system (`"%pip install..."`). Preinstalled libraries can enable the usage of the following libraries and tools:

- Scikit-Learn
- Apache Spark
- TensorFlow
- Keras
- Pandas
- Kafka
- ROS

The user can then experiment with these libraries and produce code that creates the intended output. The Jupyter Notebook that contains the code and the resulting output can be saved and shared with other users. Also, through the usage of system commands the code can be packaged to another format (e.g., tarball) and be used to create a HosmartAI Application Model which in turn can be onboarded to the HosmartAI Marketplace. The same code can

also be pushed to the GIT repo of the project and, through the CI/CD process, deploy an Application in the HosmartAI Application Space.

To login into JupyterHub, someone can use the same credentials that they use for the HHub provided special permissions concerning the usage of JupyterHub have been granted. As shown in Figure 4, this can be achieved through the integration of JupyterHub and HHub with Keycloak.

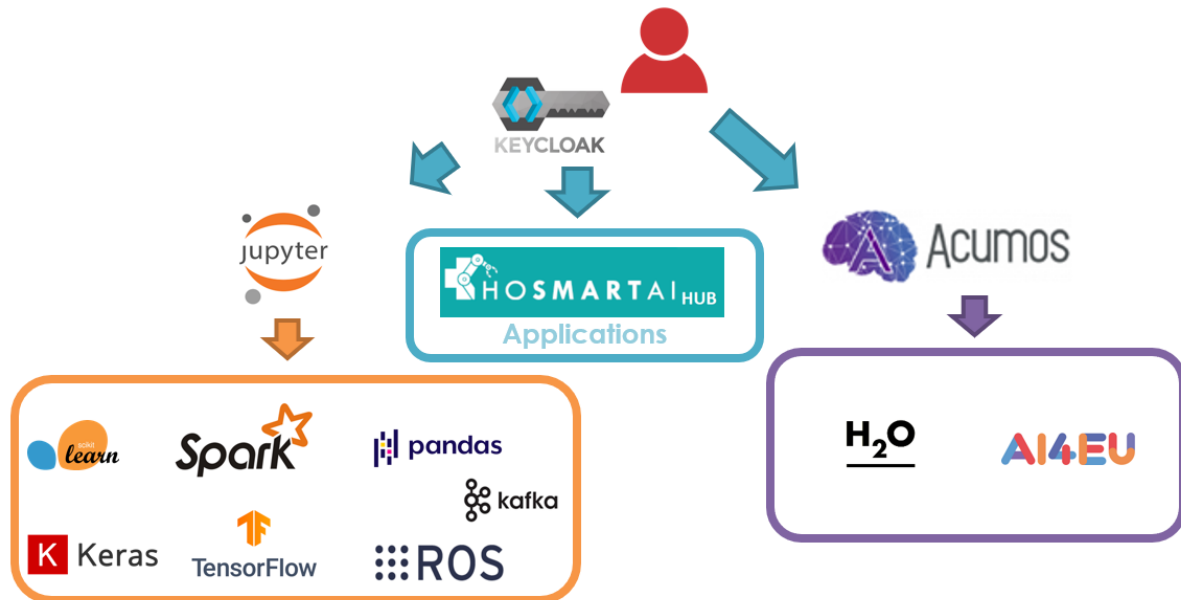


Figure 4: AI Development flow.

Through the **Acumos AI Platform**, the user can utilize existing AI4EU models by onboarding them to the Acumos Platform that is part of HosmartAI Platform architecture. The same can be done with H2O.ai models, which are supported by the Acumos platform as well. The applications that are part of the HHub, either as listings on the Marketplace or as deployed application in the HosmartAI Application Space can then potentially refer or point to applications that have been onboarded to Acumos or developed on JupyterHub.

2.1.3 Blockchain integration

The blockchain service will be deployed on-premise (at the edge) on a dedicated Internet infrastructure that will not interfere with the hospital or care facility's network.

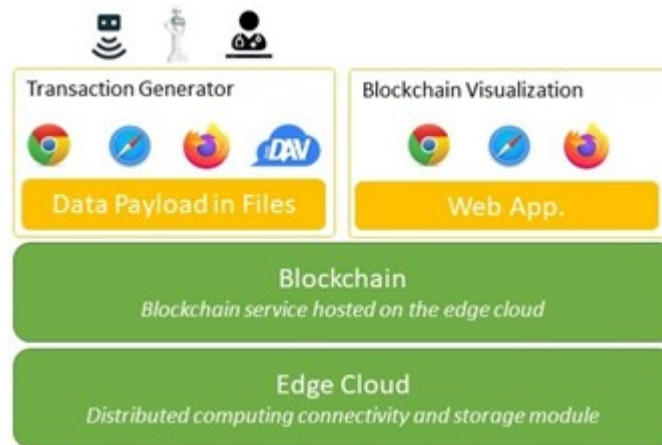


Figure 5: Blockchain service architecture.

The figure above (Figure 5) describes the architecture of the service.

- The first layer is the *Edge Cloud*. The edge cloud is an on-premise computing connectivity and storage module composed of edge nodes dedicated to the blockchain. The nodes will also offer Wi-Fi and Ethernet interfaces for IP connectivity. This module contains an embedded web server NGINX. Interaction with the Blockchain is done through WebDAV interface. A description of the edge cloud infrastructure is detailed in Section 2.3.
- The second layer is the *Blockchain*. The blockchain service is hosted, processed and synchronized among each node of an edge cloud. Blockchain features are detailed in the WP2 deliverables.
- The *Transaction Generator* layer consists of the data that will feed the blockchain. Robots and IoT devices deployed in Pilots #5 and #6 will generate data. Data that are worth to be traced is packaged into files. Files are uploaded manually to the blockchain with a simple web browser or automatically via a WebDAV interface.
- The *blockchain visualization* layer is a web interface that one may use for watching and navigating through the blockchain, getting blocks and transactions details, etc.

2.1.3.1 Technology definition

Blockchain is a technology for storing and transmitting non-modifiable information. The blockchain consists of a distributed, secure and decentralized database and is composed of data structures called blocks. Each block contains a set of transactions that are brought to the blockchain to be integrated - a transaction can be defined as a piece of information that can represent any kind of information. Once a new block of data is formed, it is linked to the previous block by including the reference of the latter in its metadata (header), thus giving a chain structure. An encryption of blocks against each other makes the blockchain a non-modifiable system. The blockchain operates over a distributed network of nodes with networking, computing and storage capacity. Each node owns a copy of the blockchain and can contribute to it.

2.1.3.2 Preferred network for the project

The network chosen for the project is a dedicated computing connectivity and storage module called the edge cloud described in Section 2.3.

2.1.3.3 Added value of working with blockchain

Blockchain aims at creating trust among Parties that wish to share data (e.g., patient with hospital, hospitals among each other, hospital vs third party, etc.). Instead of calling for a Third Party to control transactions in a centralized way, the Parties rather use a distributed system called blockchain.

The main advantage of the blockchain is that the log of the data shared among the Parties will be replicated over all the nodes of the edge cloud and possibly on a cloud device to avoid that someone could modify the data. Integrity of the data is the property that the blockchain guarantee. Parties owning one or more nodes can equally visualize, verify, and participate to the blockchain.

2.1.4 The HosmartAI Frontend

The HosmartAI Frontend should be the single point of entrance for every user, regardless of the usage scenario.

2.1.4.1 Presentation - UI/UX requirements for the platform to fulfil the user needs

The HosmartAI Frontend should be accessible from any place and any device. Therefore, it should be implemented as a web application running within a browser. There are countless frameworks and technologies to develop a web application, however not many fit the diverse requirements imposed to HosmartAI Frontend. As far as UI/UX requirements are concerned, the Frontend should be presented itself as a single page application (SPA). This is not only for aesthetics but for performance too. Instead of navigating back and forth between links, the user is within a unified environment where he can access all the components of HosmartAI.

Equally important is to use an open-source framework with strong support, well – documented and avoid proprietary, black-box type of solutions as much as possible. The entire Frontend application should be able to be hosted in either Windows or Linux or MacOS systems.

All things considered, the technology of choice for the implementation of HosmartAI Frontend is Blazor. The name Blazor is a combination/mutation of the words Browser and Razor (the .NET HTML view generating engine). The innovation is that instead of having to execute Razor views on the server in order to present HTML to the browser, Blazor is capable of executing these views on the client. Of course, it supports executing SPAs on the server.

Blazor does not require any kind of plugin installed on the client in order to execute inside a browser. Blazor either runs server-side, in which case it executes on a server and the browser acts as a dumb terminal, or it runs in the browser itself by utilising WebAssembly.

Because WebAssembly is a web standard, it is supported on all major browsers, which means also client-side Blazor apps will run inside a browser on Windows/Linux/Mac/Android and iOS.

WebAssembly (abbreviated Wasm) is a safe, portable, low-level code format designed for efficient execution and compact representation. Its main goal is to enable high performance applications on the Web, but it does not make any Web-specific assumptions or provide Web-specific features, so it can be employed in other environments as well. [REF-01]

Interfacing with external modules can be implemented in two basic ways:

- API integration
- Direct hosting

API integration suggests that the functionality of attached modules will be exposed via API calls, either implemented in REST or gRPC. Direct hosting means that there will be a link to the module / system. Such an example is the user forum which will be included.

2.1.4.2 Security

We can identify two security planes: a user-plane and a system-plane. At the user-plane, we want to avoid an unauthorized user entering the system or any user accessing components he is not supposed to. At the system-plane, the main requirement is to avoid access to any component of the HosmartAI infrastructure. An example would be an unauthorized call to one of the APIs.

In order to address the security of user access, a time-proven solution such as SSO (Single – Sign On) could be used and open standards such as OAUTH. Such a system is Keycloak which is an open-source Identity and Access Management solution supported by RedHat. It supports, apart from SSO, integration with LDAP and Active Directory, standard protocols such as OpenID Connect, OAuth 2.0 and SAML 2.0. It also features central management and most importantly REST API for easy integration with other systems.

Regarding system-wide security, there are two main solutions. The first is related with a system such as Keycloak which can issue a JSON Web Token (JWT) that is used between communicating parties, specifically to address the safety between API calls. In effect, the calling party contains the token that proves that is an authorized and legit entity to call other HosmartAI endpoints.

The second solution relies mostly on network security: strict firewall rules can safeguard endpoints from unauthorized access. Of course, these two approaches could co-exist and complement each other.

2.1.4.3 Adaptability

The Frontend will be able to provide differentiated functionality depending on the type of user. This will be achieved by a set of rules that will dictate the type of resources and APIs a user has the right to access. These rules can be expressed in a custom JSON format and have the form of a configuration file. The decoupled integration via APIs also provides some flexibility towards potential changes which in practical terms means that potential changes in a module, will not require system-wide changes.

2.1.5 Data integration

Data integration methods should be clear to manage all the data in an efficient way. The 3D geometry is received and generated by the cardiac mapping system and if available a list of targets for the cardiac ablation. The data received will be used during the navigation and not be stored at the end of the navigation task. No clinical data will be generated by the robotic platform as we do not have sensors that provide feedback.

Develop an interface to mapping system and process EP map with AI in order to improve map quality and minimize damage to the heart from said procedure. Then evaluate/validate the improved EP map. This is the data that will be received for robotic navigation.

2.1.5.1 HL7

HL7 standards or HL7 protocols indicate how information is organized and communicated between two components. These standards define the language, structure and types of data required for seamless integration between health systems.

HL7 is a set of standards for the application level. That is, it is defined at OSI level 7 because it is specifically a protocol for data exchange at that level.

Level 7 provides applications with the ability to access lower layer protocols, and defines the protocols that applications use to exchange data. This means that HL7 messages could be exchanged via TCP, or FTP, or HTTP.



Figure 6: Health Level Seven Standards.

2.1.5.2 Image/genome data integration.

By improving and automating the process of data collection, while preserving patient data privacy, a platform for research will be effectively provided. This includes (i) pseudonymization routines that comply with current privacy legislation and adhere to ethical approval; (ii) automated import workflows for each type of data from online or hospital information systems; (iii) fine-grained user access and logging to ensure data protection; (iv) interface to launch processing pipelines directly from the platform. Based on the invaluable experience of the ICT department of UZ Brussel, a modular approach will be adopted by separating the import routines as much as possible, so that extension to other sources and pathologies is possible.

The database will be integrated into a joint platform, which will enable (i) access to integrated patient data, also available off-site by pseudonymization of data; (ii) the automated extraction of clinically relevant findings from imaging and molecular data from parallel research; (iii)

predictive ML based models using multifactorial patient information, and the associated uncertainties of such models.

2.2 Data security and privacy

Data security and privacy are of paramount importance in the today technological panorama, where communication of information is performed on a global scale and where huge amounts of data are exchanged in fractions of a second, therefore easily exposing sensible information to be stolen or damaged during cyber-attacks. The health and medical sector are even more touched, because of the sensitivity of the information exchanged in relation to the privacy necessities. The HosmartAI system and platform, covering the health domain and involving different pilots and installations geographically spread on a vast area, must therefore be duly securitized.

Platform's security, privacy as well as data protection and traceability general approach and requirements for the HosmartAI system were already investigated in the deliverable '*D2.1 Design of Common AI, Benchmarking and Security Pillars*'.

In this section, security, privacy and traceability needs and provisions to the platform to fulfil all the security requirements are defined and described, based on the outcomes reached so far and on the HosmartAI platform architecture presented in this deliverable (Chapter 3). As the status of the project's implementation is getting more mature, and more details about the final deployed architecture are emerging, a more implementation-approach about security than in the previous deliverables can be taken into account here. In particular, the single components of the architectures, as well as the interrelations and communications links among them and with the extern, are considered in relation to the security, privacy and traceability needs.

2.2.1 Security requirements

Security requirements were defined in detail in '*D1.5 HosmartAI Platform Conceptual Architecture*' and put in the project's JIRA Technical Requirements. Basically, the key concepts about security/privacy/traceability needs that emerged during the investigation stage can be summarized by the non-functional requirements described there, as follows:

- **Tools:** the tools ensuring security, privacy and traceability should be employed in the system, and should be open source, when possible.
- **Data processing:** the processing (in particular by the AI algorithms) of the collected information (datasets) within the system will be also performed using open-source tools, when possible.
- **Laws, standards and regulations:** the security, privacy and traceability subsystem should comply with applicable laws and regulations of the EU and Member States (or with any other recognized regulations), when possible.
- **Scalability:** The system should be scalable in regard with the security, privacy and traceability needs.

- **Encrypted communications:** the communication over public networks within the system and between the system and its users will be securitized by encryption, when necessary.
- **System continuity:** system continuity and resilience should be guaranteed against attacks on one or more subcomponents, against falsified or corrupted data and from physical damaging of one or more of its components.

About the functional requirements, the main indications and provisions employed in and come out from the study can be summed up as follows:

- **Reference architecture:** the system's architecture of reference subjected to securitization analysis is the HosmartAI's platform architecture described in Chapter 3 of the present document (refer in particular to the block diagram of Section 3.1).
- **Security audits:** The deployed infrastructures (platform and pilots) and data should be periodically submitted to security audit campaigns. A planning of the audit sessions should therefore be prepared. Appropriate security tests (described in D2.1) are also planned in the audits.
- **Event monitoring:** digital events (data exchange over networks, data storing, etc.) should be constantly monitored. A SIEM system, monitoring the whole infrastructure by producing system and events logs, is strongly suggested. In that case, ad-hoc hardware/software agents (probes) can also be deployed in strategic points of the infrastructure.
- **Data selection, filtering, limiting and access:** data exchanges should be filtered and limited, lowering the risk that sensitive information is stolen or damaged. Also, access policies to exchanged information must be provided (sensible data should be also anonymized or pseudo-anonymized, depending on the cases). The design of the data protection policies will be based on the content of deliverable '*D6.7 Data Management Handling Plan*'.
- **Authentication and access control:** user access and authentication and system-wide securitized access must be guaranteed (these are also described in Section 2.1.4.2 of this document).
- **Data traceability:** criteria of point-to-point traceability on certain data (parameters) that are required to be traced will be defined in policies based on the content of deliverable '*D6.7 Data Management Handling Plan*'.

Moving to the details, in the following two chapters, data protection and traceability needs are mapped to the security technical requirements, and for each of them, it is reported the employable methods and tools, as well as the specification to which architecture's components they can be applied.

2.2.2 Data protection methods

2.2.2.1 Audit on data and infrastructures

Table 2: List of needs and related requirements for data protection.

Need	Description	Related functional requirements (in black), and non-functional requirements (in green) [as defined in JIRA]
SN1	<i>Audit on data and infrastructures:</i>	TR-36 Monitoring of the research data datasets <i>TR-46 Open-source tools for security, privacy and traceability subsystem</i> <i>TR-48 Compliance of the security, privacy and traceability subsystem with regulations</i>

2.2.2.1.1 Methods and Tools

- Periodic audit campaigns on all platform components and pilot infrastructures, applying security and penetration testing methodologies and well-established standards (D2.1, Section 5.3.3).
- Envisioned tools:
 - o Security Onion,
 - o Wireshark,
 - o OWASP,
 - o Zenmap,
 - o Kali Linux NetHunter.

2.2.2.1.2 Applicability to platform components

1. All HHUB's components:

- HosmartAI Dashboard
 - o User Interface
 - o Public Space
- User Access Management
 - o Keycloak
 - o User Database (MySQL)
 - o User Management (Spring boot)
- Benchmarking Framework
 - o Benchmarking Libraries and Services
 - o Database (mongoDB)

- o Open API (benchmarking data)
- Co-creation space
 - o Common AI pillars
 - o Data Management
 - o Co-creation tools
 - o Application deployment
- Marketplace
 - o Content Management (Drupal)
 - o Other services

2. All pilot applications and sites

2.2.2.2 Data selection

Table 3: List of needs and related requirements for data selection.

Need	Description	Related functional requirements (in black), and non-functional requirements (in green) [as defined in JIRA]
SN2	<i>Data selection: limit data export to the ones required by AI processing</i>	TR-41 Data filtering and limiting <i>TR-47 Open-source tools for processing the collected data</i> <i>TR-49 Scalability in regard with the security, privacy and traceability</i> <i>TR-50 Encryption of communication over public networks</i> <i>TR-51 System continuity</i>

2.2.2.2.1 Methods and Tools

Design and application of Filtering Policies based on:

- defined requester services
- specific API endpoint (Open API)

Provision of data protection policies: the design of the policies with the specification of the exchange limits for each dataset will be based on the content of deliverable 'D6.7 Data Management Handling Plan'.

2.2.2.2.2 *Applicability to platform components*

- 1. HHUB’s components:**
 - Benchmarking Framework
 - o Open API
 - Co-creation space
 - o Common AI pillars
 - o Application deployment
 - Marketplace
- 2. All pilot applications and sites**

2.2.2.3 *Data verification*

Table 4: List of needs and related requirements for data verification.

Need	Description	Related functional requirements (in black), and non-functional requirements (in green) [as defined in JIRA]
SN3	<i>Data verification/integrity methods and anonymization or pseudonymization</i>	TR-42 Integrity verification of research datasets TR-46 <i>Open-source tools for security, privacy and traceability subsystem</i> TR-48 <i>Compliance of the security, privacy and traceability subsystem with regulations</i> TR-49 <i>Scalability in regard with the security, privacy and traceability</i> TR-51 <i>System continuity</i>

2.2.2.3.1 *Methods and Tools*

- Data path and traceability, definition of plausibility criteria.
- Increase in robustness and resiliency to errors.
- Use of hashing techniques to guarantee anonymization and integrity.
- Possible use of Blockchain techniques (e.g., Ethereum blockchain) to guarantee data integrity over time (Section 2.1.3 of this deliverable).

- Provision of data protection policies: the design of the policies with the indication of the need of anonymization or pseudonymization for every dataset will be based on the content of deliverable ‘D6.7 Data Management Handling Plan’.

2.2.2.3.2 *Applicability to platform components*

1. **HHUB’s components:**

- HosmartAI Dashboard
 - Public Space
- Benchmarking Framework
 - Database (mongoDB)
 - Open API
- Co-creation space
 - Data Management
 - Application deployment
- Marketplace

2. **All pilot applications and sites**

2.2.2.4 *Authentication and access control*

Table 5: List of needs and related requirements for authentication and access control.

Need	Description	Related functional requirements (in black), and non-functional requirements (in green) [as defined in JIRA]
SN4	<i>Forcing authentication and access control</i>	TR-43 Validation of the origin of data <i>TR-46 Open-source tools for security, privacy and traceability subsystem</i> <i>TR-48 Compliance of the security, privacy and traceability subsystem with regulations</i> <i>TR-50 Encryption of communication over public networks</i> <i>TR-51 System continuity</i>

2.2.2.4.1 *Methods and Tools*

- Base user authentication at the HosmartAI Frontend is performed by the Keycloak open-source identity and access management tool, already described in this deliverable in Section 2.1.4.2.

- System-wide securitized access is performed by two methods:
 - Keycloak, able to issue a JSON Web Token (JWT)
 - Deploying firewalling solutions at specific endpoints together with well-defined firewalling rules. Examples of hardware firewalls that can be employed are:
 - **EXYS9000-EFS** (endpoint tunnelling and firewalling system),
 - Snort,
 - Suricata,
 - SonicWall firewalls,
 - CUJO AI,
 - OPENSense,
 - Sophos.

2.2.2.4.2 *Applicability to platform components*

1. **HHUB's components:**

- HosmartAI Dashboard
 - User Interface
 - Public Space
- User Access Management
 - Keycloak
 - User Database (MySQL),
 - User Management (Spring boot)
- Benchmarking Framework
 - Open API
- Co-creation space
 - Application deployment

2. **All pilot applications and sites**

2.2.2.5 *Event Management*

Table 6: List of needs and related requirements for event management.

Need	Description	Related functional requirements (in black), and non-functional requirements (in green) [as defined in JIRA]
SN5	<i>Event Management</i>	TR-44 Monitoring of digital events (event logs) <i>TR-46</i> <i>Open-source tools for security, privacy and traceability subsystem</i> <i>TR-50</i> <i>Encryption of communication over public networks</i>

2.2.2.5.1 *Methods and Tools*

- Collection of data logs from all system components using syslogs, ad-hoc hardware/software agents installed in the infrastructure (probes) and SSL based connections to detect anomalies.
- Envisioned tools - SIEM platforms:
 - o **EXYS7900** (Enterprise grade SIEM with hardware/software agents)
 - o LogRhythm,
 - o NextGen,
 - o Splunk,
 - o AlienVault,
 - o OSSIM.

2.2.2.5.2 *Applicability to platform components*

1. All HHUB's components:

- HosmartAI Dashboard
 - o User Interface
 - o Public Space
- User Access Management
 - o Keycloak
 - o User Database (MySQL)
 - o User Management (Spring boot)
- Benchmarking Framework
 - o Benchmarking Libraries and Services
 - o Database (mongoDB)
 - o Open API
- Co-creation space
 - o Common AI pillars
 - o Data Management
 - o Co-creation tools
 - o Application deployment
- Marketplace
 - o Content Management (Drupal)
 - o Other services

2. All pilot applications and sites

2.2.3 Traceability of the information

Table 7: List of needs and related requirements for traceability of the information.

Need	Description	Related functional requirements (in black), and non-functional requirements (in green) [as defined in JIRA]
SN6	<i>Traceability in heterogeneous and aggregated datasets:</i>	TR-45 Data traceability

		<p>TR-46 <i>Open-source tools for security, privacy and traceability subsystem</i></p> <p>TR-49 <i>Scalability in regard with the security, privacy and traceability</i></p> <p>TR-51 <i>System continuity</i></p>
--	--	---

2.2.3.1 Methods and Tools

- Definition of adequate policy and criteria of traceability on data (parameters) that are required to be traced (for instance, complete traceability of a health parameter from source to destination, tracking and storing all the intermediate passages, or only some pre-declared intermediate passages (D2.1, Section 5.3.4.1). The design of the policies with the indication of the traceability needs will be based on the content of deliverable ‘D6.7 Data Management Handling Plan’.
- Envisioned tools: use of Blockchain technology (e.g., Ethereum blockchain, as well as the HosmartAI’s blockchain implementation, Section 2.1.3 of this deliverable) to ensure a point-to-point and very solid traceability environment for particular data and parameters’ requirements.

2.2.3.2 Applicability to platform components

1. HHUB’s components:

- User Access Management
 - User Database (MySQL)
- Benchmarking Framework
 - Database (mongoDB)
 - Open API
- Co-creation space
 - Data Management
 - Application deployment
- Marketplace

2. All pilot applications and sites

2.3 Edge Cloud

The edge cloud is an edge-based computing connectivity and storage module provided for some applications inside the whole ecosystem to connect robots and other IoT devices without interfering with the hospital’s infrastructure and to host the blockchain service near end-devices.

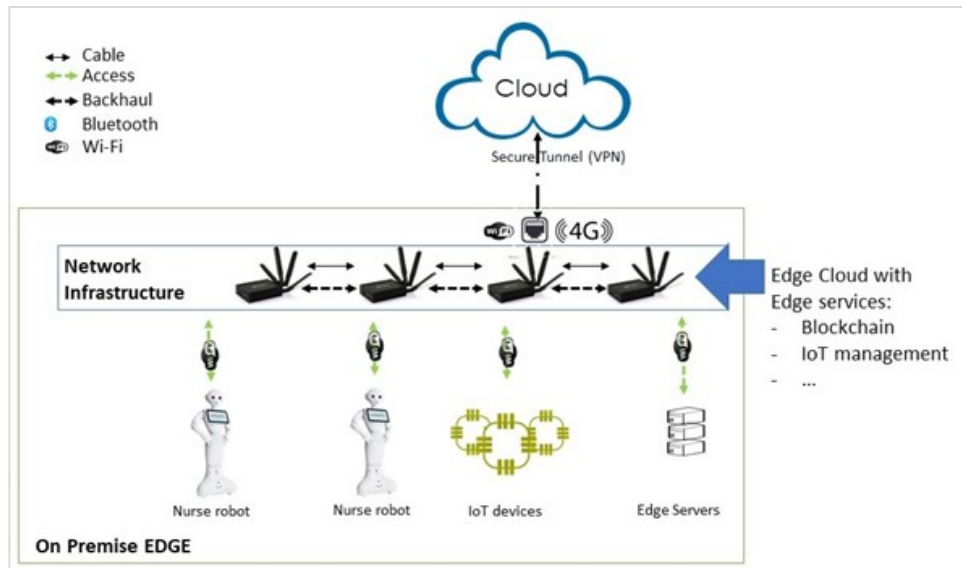


Figure 7: Edge Cloud infrastructure description.

The figure above (Figure 7) describes the Edge Cloud architecture:

- The *Network Infrastructure* is composed of a set of embedded nodes with networking, storage and computing capabilities. Nodes communicate with each other using a mesh networking protocol and create a local Internet and edge cloud infrastructure that is autonomous from any external structure. The edge cloud host edge-based services for performing various tasks such as IoT management, local data sharing and the blockchain. Nodes connected with each other recognize being part of the same network and contributing to the same blockchain through secure authentication.
- Robots, IoT devices and other edge servers can connect to the edge cloud for enjoying a dedicated networking infrastructure and sharing their data among the edge cloud and the blockchain.
- The edge cloud can be connected to remote servers and edge clouds operating at multiple locations for creating the equivalent of a large cloud distributed at the edges.

All equipment is deployed on-premises, at the care facility, rather than on distant clouds for data sovereignty and service resiliency to Internet shut down.

2.4 Open API

In order to interconnect both internal elements and to give external access to certain services, it is necessary to define a set of accesses. To do this, a standard must be adopted, so that all calls are the same and it is easier to add or maintain all services. OpenAPI is the specification of choice, and we will talk more about it below.

2.4.1 API Documentation

API documentation is essentially the reference manual for an API, it tells API consumers how to use the API. API documentation is meant for humans, usually developers, to read and understand. Providing documentation that is well-designed, comprehensive, and easy to

follow is crucial when it comes to ensuring developers have a great experience with the API. Also, a great developer experience (DX) means a better chance for API success. Good documentation also helps decrease the time it takes to onboard new API consumers.

API documentation should provide an example of every call, every parameter, and responses for each call. It should include code samples for commonly used languages such as Java, JavaScript, PHP, and Python. Documentation should provide an explanation for each API request and examples of error messages. It's also important that API documentation is actively maintained and always up to date.

2.4.2 API Specification

API specification is a term that is often used interchangeably with API definition. While these terms have many similarities, they are different entities. An API specification provides a broad understanding of how an API behaves and how the API links with other APIs. It explains how the API functions and the results to expect when using the API.

An API specification explains how the API behaves and what to expect from the API. It contains the list of actions that the API offers, with the definition of the objects, values and parameters needed to call each method, what REST verbs can be used, and the relationships between different objects.

2.4.3 API Definition

An API definition is like an API specification that provides an understanding of how an API is organized and how the API functions. An API definition provides information about how the API functions, how it links with other APIs, and the expected results in a machine-readable format. It focuses on defining the API and outlining the structure of the API.

An API definition is often used as a baseline for automated tools. API definitions can be used to generate API documentation, code samples, and SDKs automatically.

API definitions can also be imported into a mock server for virtual API testing. Among the many tools for mock server and API testing that allow to import an API definition file are SoapUI and SwaggerHub.

An API definition can be used to power automated tools that can improve the quality of the documentation, compile client API libraries or generate unit tests for the API.

2.4.4 API Visualization

For listing the principal services, a visualization tool will be used, in this case Swagger. The API is defined in a YAML or JSON file, with all the data, like the method, parameter, users, security, and all the remaining features regarding the API.


```

info:
  description: "Description about HosmartAI API"
  version: "0.0.1"
  title: "HosmartAI"
host: "hosmartai.swagger.io"
basePath: "/v0"
tags:
- name: "Marketplace"
  description: "Communication with marketplace for interact with application and services"
- name: "Co-Creation"
  description: "Interact with continous integration/development tools"
- name: "Benchmarking"
  description: "Operations about benchmarking"
schemes:
- "https"
- "http"
paths:
  /market/addOpinion:
    post:
      tags:
      - "Marketplace"
      summary: "Add a opinion to a service"
      description: ""
      operationId: "addOpinion"
      consumes:
      - "application/json"
      - "application/xml"

```

Figure 8: OpenAPI YAML example.

The above lines (Figure 8) define the API, but the tools translate it to a human readable interface.

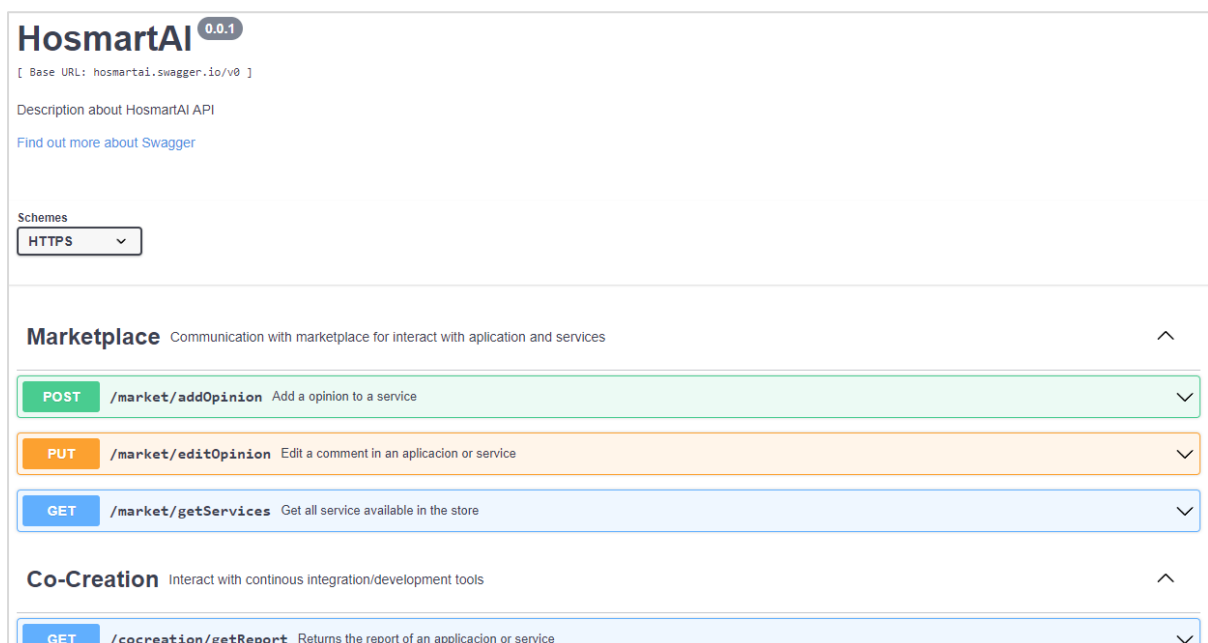


Figure 9: API Visualization tool (Swagger).

2.4.5 Third party OpenAPI platform elements

Some platform elements already have an OpenAPI specification. These APIs can be used to connect internal platform elements or to create an API Gateway to give external access to these HosmartAI services.

2.4.5.1 Jupyterhub OpenAPI

JupyterHub OpenAPI enables the management of users and groups of users. It also can be used for authorization, service listing and user's notebook servers' management.

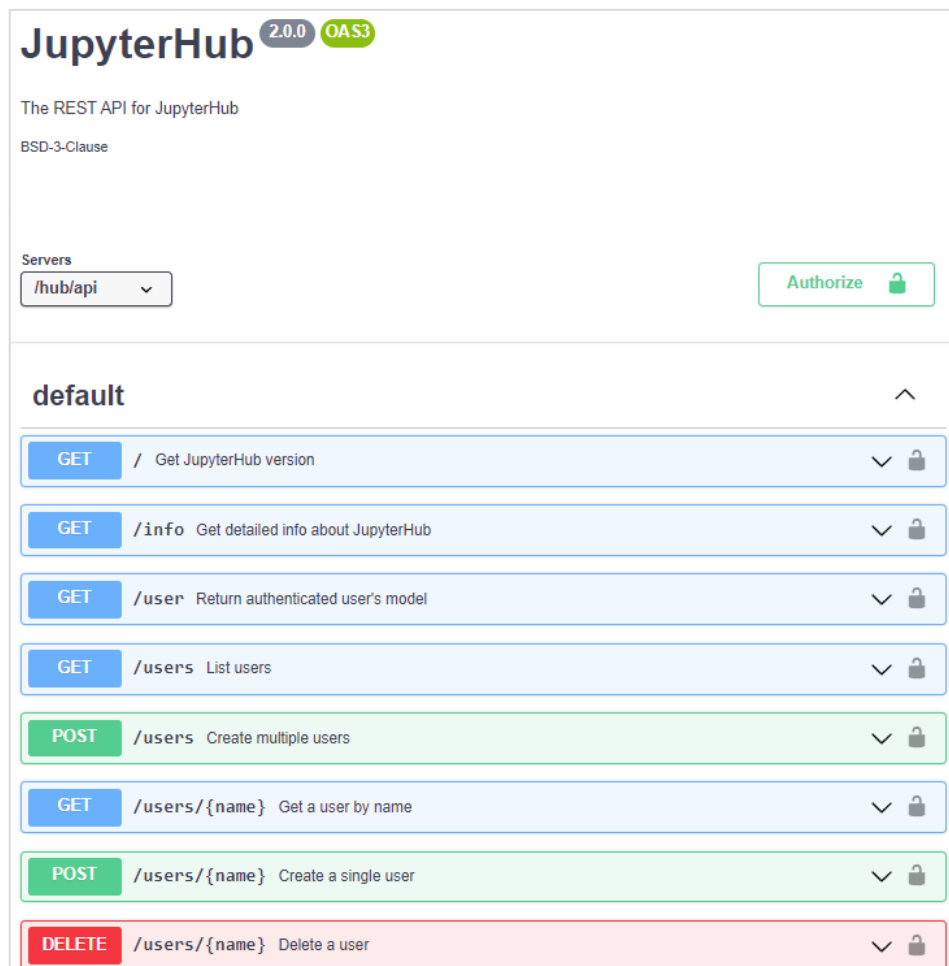
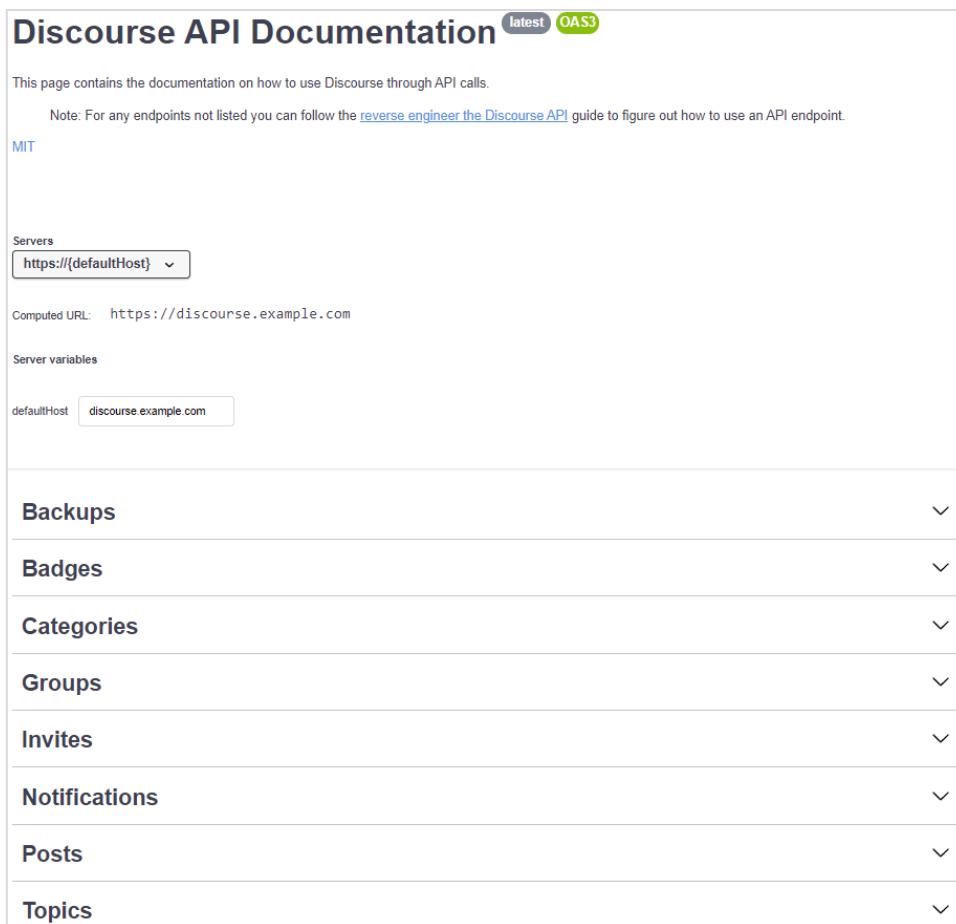


Figure 10: JupyterHub OpenAPI.

2.4.5.2 Discourse OpenAPI

Discourse OpenAPI can be used to manage all the discourse info, like creating backups, creating users, creating posts, sending private messages, uploading files and other actions.

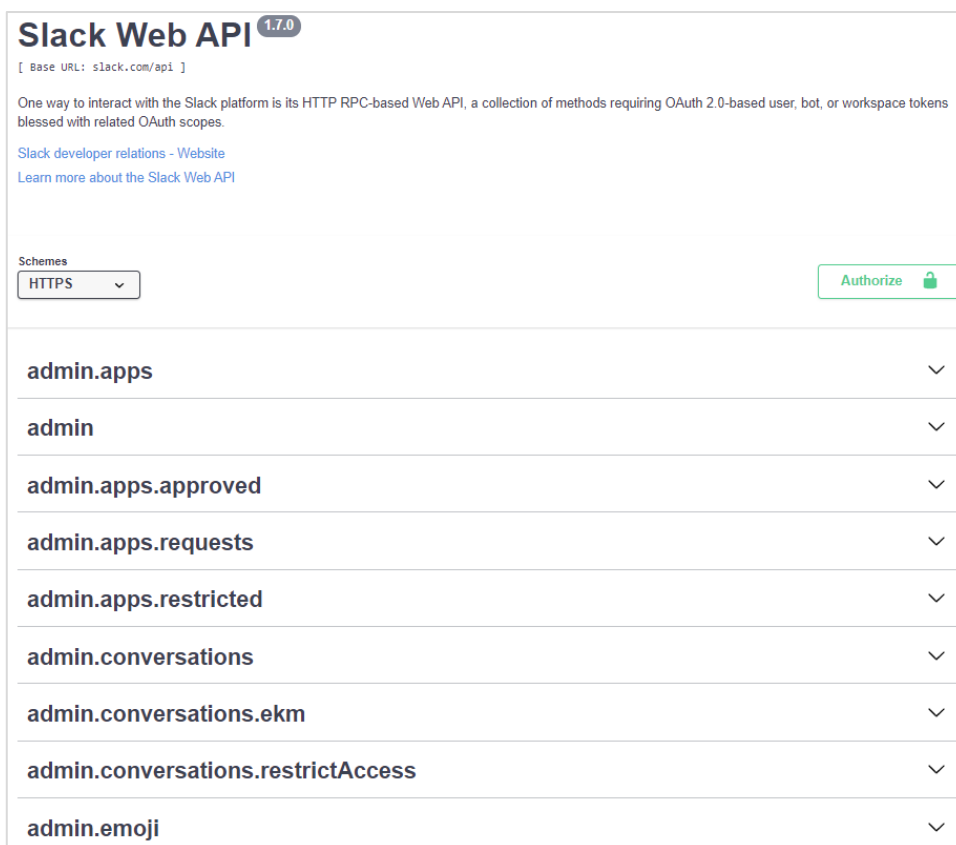


The screenshot shows the 'Discourse API Documentation' page. At the top, there is a title 'Discourse API Documentation' with 'latest' and 'OAS3' badges. Below the title, a paragraph states: 'This page contains the documentation on how to use Discourse through API calls.' A note follows: 'Note: For any endpoints not listed you can follow the [reverse engineer the Discourse API](#) guide to figure out how to use an API endpoint.' There is a 'MIT' license indicator. Under the heading 'Servers', there is a dropdown menu showing 'https://{defaultHost}' and a 'Computed URL: https://discourse.example.com'. Under 'Server variables', there is a text input field for 'defaultHost' containing 'discourse.example.com'. At the bottom, there is a table of contents with expandable sections: Backups, Badges, Categories, Groups, Invites, Notifications, Posts, and Topics.

Figure 11: Discourse OpenAPI.

2.4.5.3 Slack OpenAPI

Slack has an OpenAPI that can be used to manage all the information of the web, like creating users, uploading files, managing chats, approve the use of apps and other actions.



Slack Web API ^{1.7.0}
[Base URL: slack.com/api]

One way to interact with the Slack platform is its HTTP RPC-based Web API, a collection of methods requiring OAuth 2.0-based user, bot, or workspace tokens blessed with related OAuth scopes.

[Slack developer relations - Website](#)
[Learn more about the Slack Web API](#)

Schemes
HTTPS

Authorize

admin.apps	▼
admin	▼
admin.apps.approved	▼
admin.apps.requests	▼
admin.apps.restricted	▼
admin.conversations	▼
admin.conversations.ekm	▼
admin.conversations.restrictAccess	▼
admin.emoji	▼

Figure 12: Slack OpenAPI.

2.4.5.4 Acumos OpenAPI

Acumos has an OpenAPI for a module called License Usage Manager (LUM).

This API can be used to manage licence agreements for software elements of the Acumos platform.

```
1 # =====
2 # Copyright (c) 2019-2020 AT&T Intellectual Property. All rights reserved.
3 # Modifications Copyright (C) 2019 Nordix Foundation.
4 # =====
5 # Licensed under the Apache License, Version 2.0 (the "License");
6 # you may not use this file except in compliance with the License.
7 # You may obtain a copy of the License at
8 #
9 #   http://www.apache.org/licenses/LICENSE-2.0
10 #
11 # Unless required by applicable law or agreed to in writing, software
12 # distributed under the license is distributed on an "AS IS" BASIS,
13 # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
14 # See the License for the specific language governing permissions and
15 # limits under the License.
16 # =====LICENSE_END=====
17 ---
18
19 openapi: '3.0.3'
20
21 info:
22   version: "0.0.0"
23   title: "License Usage Manager (LUM) API"
24   description: RESTfull-ish API for License Usage Manager - json based
25   license:
26     name: "Apache 2.0"
27     url: "http://www.apache.org/licenses/LICENSE-2.0.html"
28   contact:
29     name: Acumos License Management
30     url: https://wiki.acumos.org/display/LM/
31
32 externalDocs:
33   description: LUM docs
34   url: https://docs.acumos.org/en/latest/submodules/license-usage-manager/docs/index.html
35
```

Figure 13: Acumos LUM OpenAPI (YAML).

License Usage Manager (LUM) API 0.0.0 OAS3

RESTfull-ish API for License Usage Manager - json based

[Acumos License Management - Website](#)

[Apache 2.0](#)

[LUM docs](#)

Servers

/ - root ▼

swid-tag	CRUD on swidTag - software-identity tag with licenseProfile attached to it	▼
asset-usage-agreement	CRUD on assetUsageAgreement that is the collection of permissions/rights-to-use	▼
asset-usage-agreement-restriction	CRUD on assetUsageAgreementRestriction with all the subscriber company restrictions against the asset-usage agreement	▼
asset-usage	request the entitlement on the asset usage or record an event	▼
asset-usage-tracking	reports of all requests with decisions per supplier	▼
info	LUM healthcheck and info	▼
admin	LUM admin to see and change the settings	▼

Figure 14: Acumos LUM OpenAPI (Swagger).

3 HosmartAI Architecture Design

As a result of the analysis performed of all the different technical requirements, the outcome from other tasks and other elements that are considered necessary in the architecture, the first version of the final architecture can be defined.

3.1 Principal elements

The Following figure (Figure 15) pictures the elements of the HosmartAI platform and the connection between them. This diagram has been created using collaborative tools in the HosmartAI Confluence space.

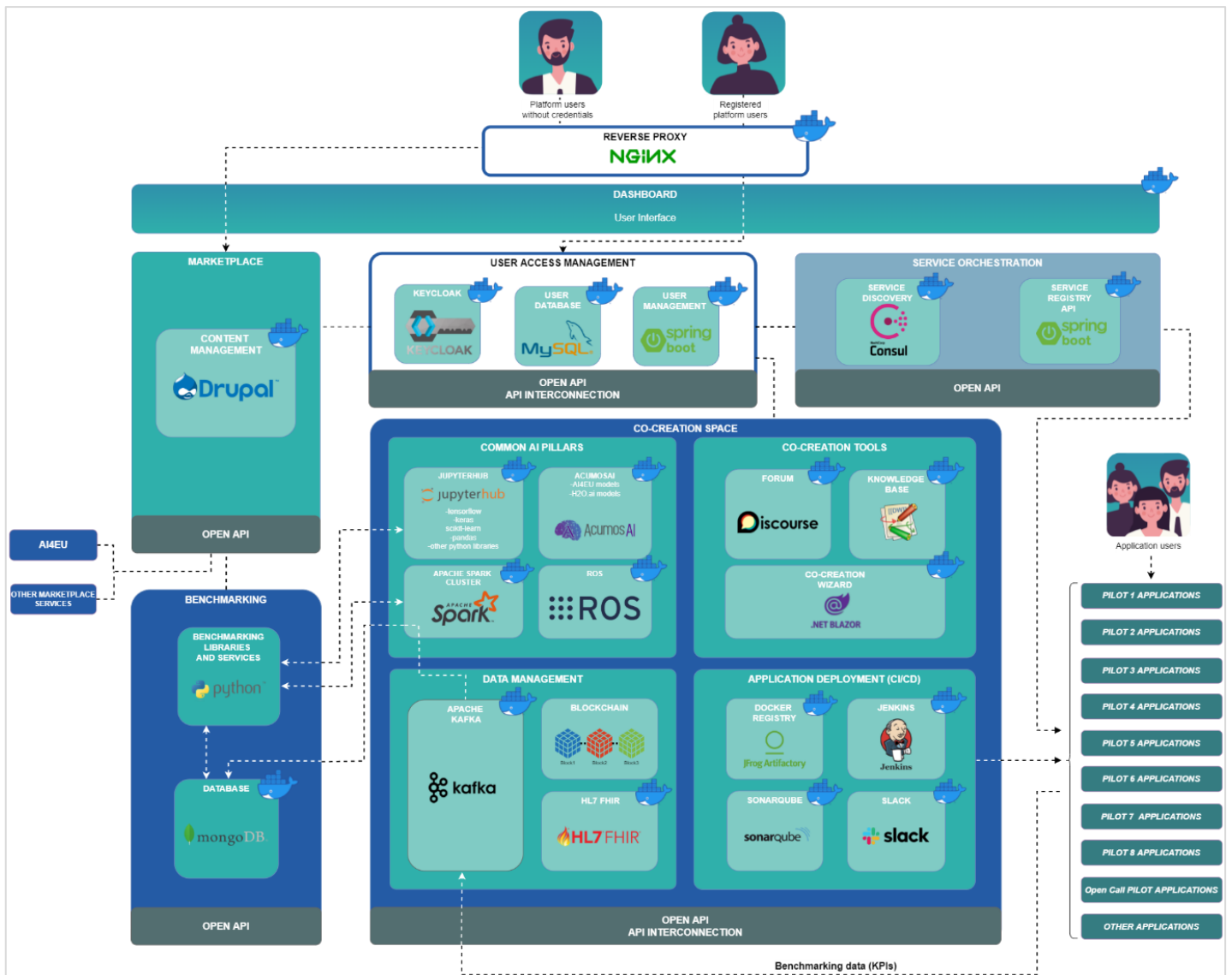


Figure 15: HosmartAI Architecture Diagram.

The principal elements presented in the first version of the final architecture are:

- Docker
- Ansible
- Nginx

- Acumos
- ROS
- Keycloak
- Consul

3.1.1.1 Docker

Docker is a set of platforms as a service (PaaS) product that use OS-level virtualization to deliver software in packages called containers. Containers are isolated from one another and bundle their own software, libraries, and configuration files. They can communicate with each other through well-defined channels. Because all the containers share the services of a single operating system kernel, they use fewer resources than virtual machines. [REF-02]

Table 8: Docker requirements.

Hardware elements
<ol style="list-style-type: none"> 1. 64-bit processor 2. > = 4 GB RAM 3. BIOS hardware virtualization support
Software elements
<ol style="list-style-type: none"> 1. Linux OS 2. Docker-compose 3. Python 4. Curl

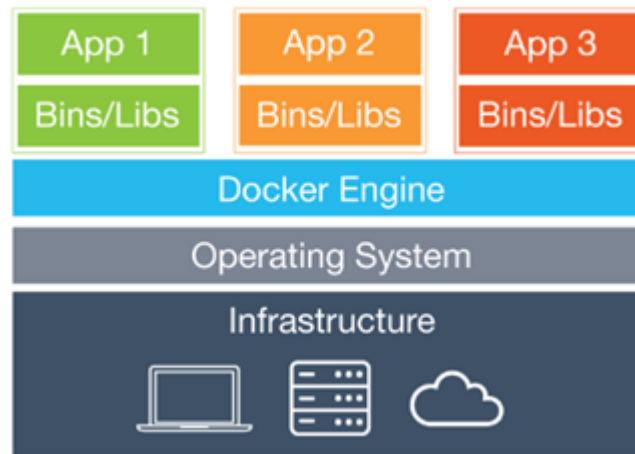


Figure 16: Docker infrastructure.

3.1.1.2 Ansible

Ansible is an open-source software provisioning, configuration management, and application-deployment tool enabling infrastructure as code. It runs on many Unix-like systems, and can configure both Unix-like systems as well as Microsoft Windows. It includes its own declarative language to describe system configuration. Ansible is agentless, temporarily connecting remotely via SSH or Windows Remote Management (allowing remote PowerShell execution) to do its tasks. [REF-03]

Table 9: Ansible requirements.

Hardware elements
<ol style="list-style-type: none"> 1. 2 GB RAM (4 GB RAM is recommended) 2. 20 GB hard disk space 3. 64-bit support required (kernel and runtime)
Software elements

1. Linux OS
 - a. Red Hat Enterprise Linux 6 64-bit
 - b. Red Hat Enterprise Linux 7 64-bit
 - c. CentOS 6 64-bit
 - d. CentOS 7 64-bit
 - e. Ubuntu 12.04 LTS 64-bit
 - f. Ubuntu 14.04 LTS 64-bit
2. Windows Server
 - a. 2012
 - b. 2016
 - c. 2019

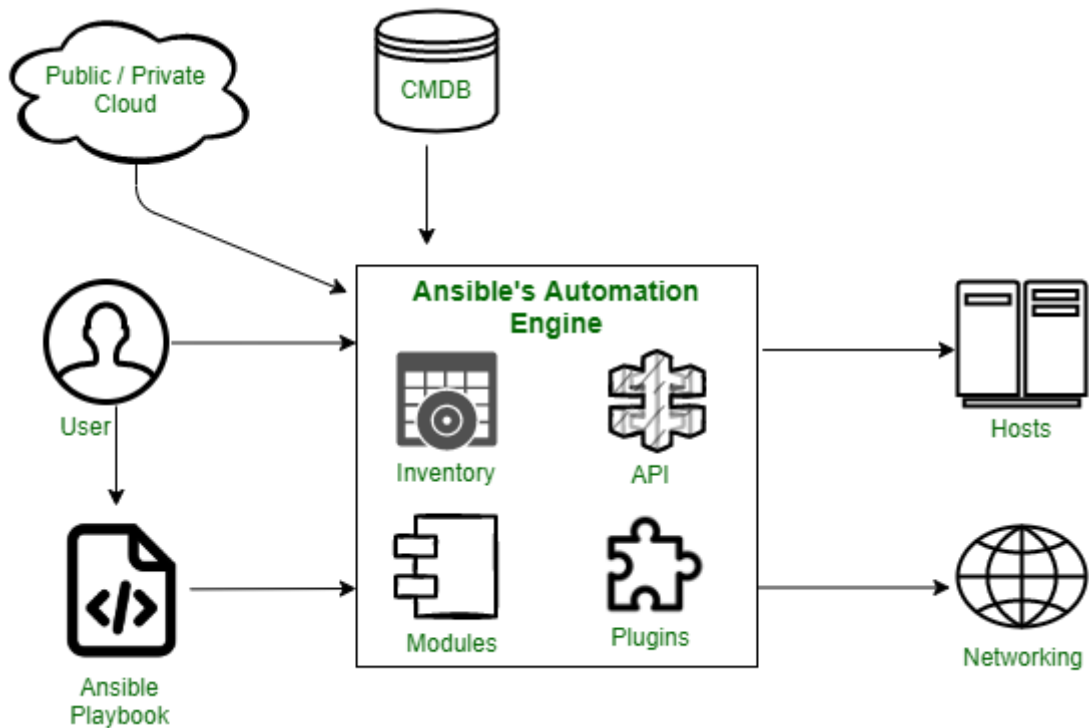


Figure 17: Ansible infrastructure.

3.1.1.3 Nginx

Nginx is a web server that can also be used as a reverse proxy, load balancer, mail proxy and HTTP cache. Nginx is free and open-source software, released under the terms of the 2-clause BSD license. A large fraction of web servers uses NGINX, often as a load balancer. [REF-04]

Table 10: Nginx requirements.

Hardware elements
<ol style="list-style-type: none"> 1. 8 GB RAM 2. CPU: 8-Core CPU @ 2.40 GHz or similar 3. Disk space: 155–255 GB
Software elements
<ol style="list-style-type: none"> 1. Linux OS 2. Database for operate

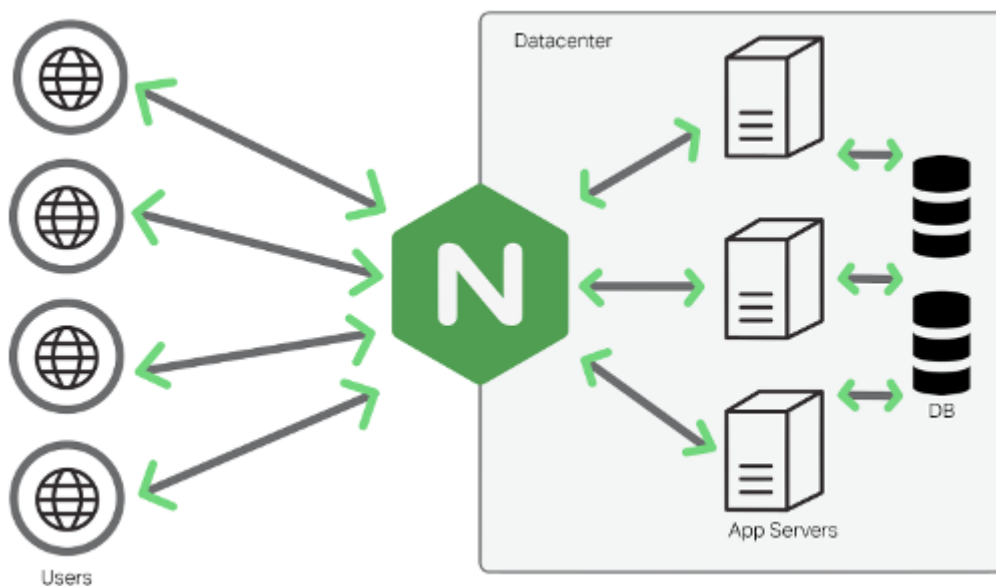


Figure 18: Nginx infrastructure.

3.1.1.4 Acumos

Acumos AI is a platform and open-source framework that makes it easy to build, share, and deploy AI apps. Acumos standardizes the infrastructure stack and components required to run an out-of-the-box general AI environment. This frees data scientists and model trainers to focus on their core competencies and accelerates innovation. [REF-05]

Table 11: Acumos requirements.

Hardware elements
<ol style="list-style-type: none"> 16GB RAM 100GB disk storage size Open application ports
Software elements
<ol style="list-style-type: none"> Linux OS or Windows OS or Mac OS Curl

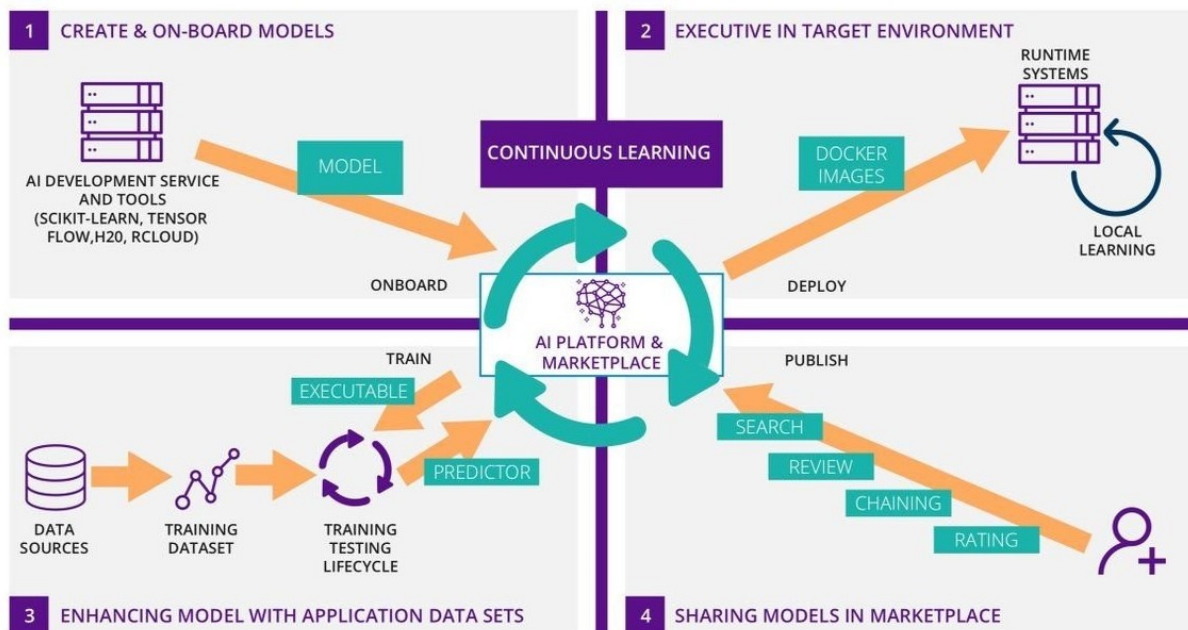


Figure 19: Acumos AI platform.

3.1.1.5 ROS

Robot Operating System is an open-source robotics middleware suite. Although ROS is not an operating system but a collection of software frameworks for robot software development, it provides services designed for a heterogeneous computer cluster such as hardware abstraction, low-level device control, implementation of commonly used functionality, message-passing between processes, and package management. Despite the importance of reactivity and low latency in robot control, ROS itself is not a real-time OS (RTOS). It is possible, however, to integrate ROS with real-time code. [REF-06]

Table 12: ROS requirements.

Hardware elements
1. Robot for running the OS
Software elements
1. Linux OS
2. Access to repositories to allow all type of libraries

3.1.1.6 Keycloak

Keycloak is an open-source software product that enables single sign-on (IdP) with Identity Management and Access Management for modern applications and services. This software is written in Java and supports by default the SAML v2 and OpenID Connect (OIDC) / OAuth2 identity federation protocols.

From a conceptual perspective, the intention of the tool is to facilitate the protection of applications and services with little or no encryption. An IdP allows an application (often called a Service Provider or SP) to delegate its authentication. [REF-07]

Table 13: Keycloak requirements.

Hardware elements
1. 512M of RAM
2. 1GB of disk space
Software elements
1. Java 8 JDK
2. zip or gzip and tar

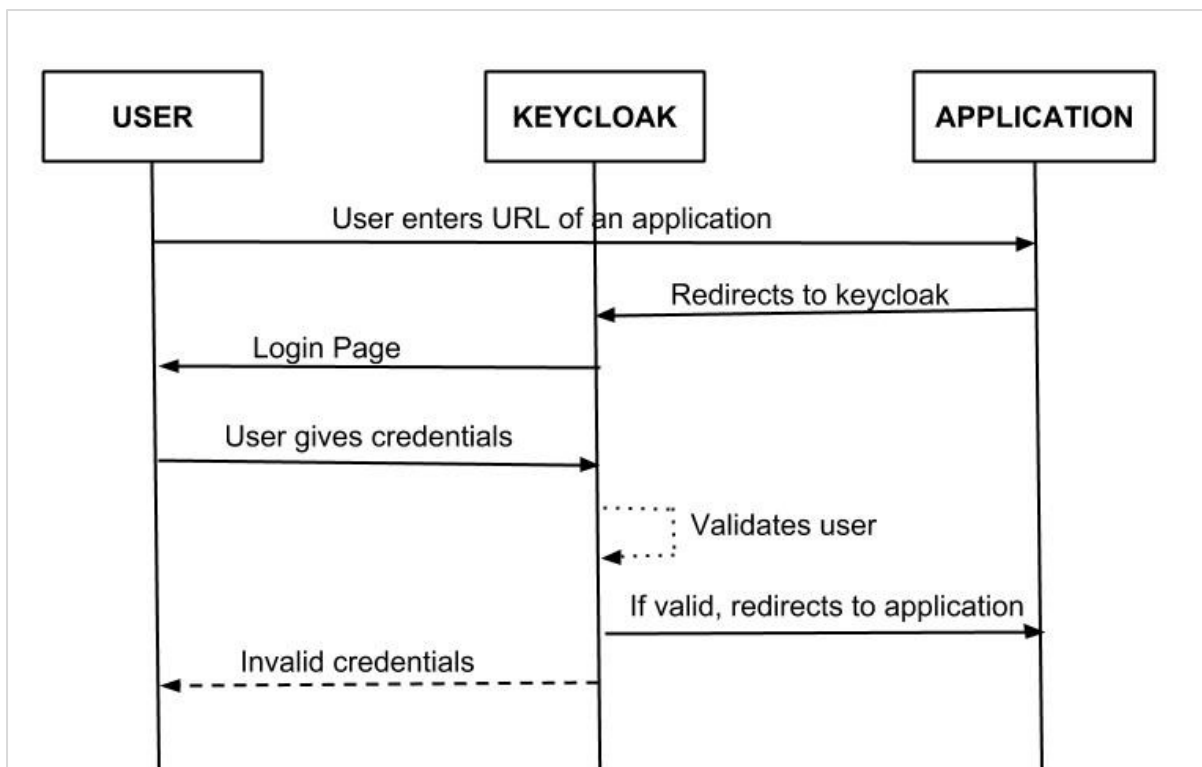


Figure 20: Keycloak data flow.

3.1.1.7 Consul

Consul is a networking tool that provides a fully featured service mesh and service discovery [REF-08].

Table 14: Consul requirements.

Hardware elements
<ol style="list-style-type: none"> 1. 2-4 core 2. 8-16 GB RAM 3. 100+ GB disk space
Software elements
<ol style="list-style-type: none"> 1. Linux OS or Windows OS 2. Curl on Linux 3. Chocolatey on Windows

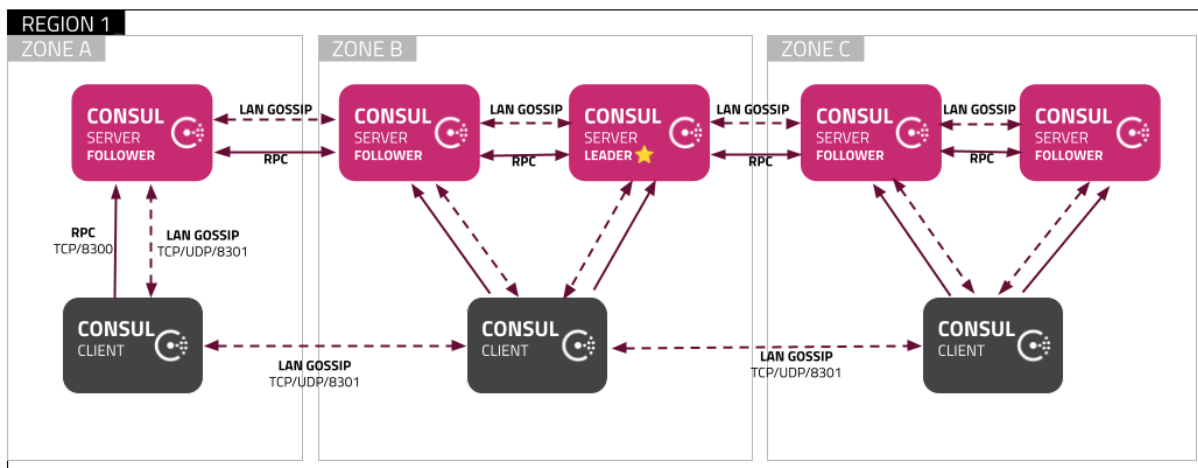


Figure 21: Consul infrastructure.

3.2 HHub

The main tools used for the development of the HHub (HosmartAI Hub) are listed below. These tools will manage the content displayed in the marketplace, the correct continuous integration and development in the Co-creation hub and measure the quality and performance of the system and its applications through benchmarking tools.

3.2.1 Marketplace

The marketplace will implement the project’s repository displaying its main results in terms of selected AI-applications, relevant data and services, including e.g., provider details, main features and integration constraints.

3.2.1.1 Drupal

Table 15: Drupal requirements.

Hardware elements
1. 60 MB disk storage size
Software elements
1. MySQL, MariaDB, or Percona Server
2. Nginx
3. PHP 5.2.5 or higher

3.2.2 Co-creation Hub

The Co-creation Hub is a group of components used for the development and creation of new HosmartAI solutions. It is composed of 2 main sets of elements: one of them is dedicated to the co-creation tools and the other to the continuous integration and development.

For the Co-Creation tools we have the following components:

- .Net Blazor
- Discourse

3.2.2.1 NET Blazor

Blazor is a framework for creating interactive client-side web user interfaces with .NET, using C# instead of JavaScript. It can run client-side C# code directly in the browser using WebAssembly. This allows for code and libraries in C# to be shared between the server-side and the client-side parts of the application. [REF-09]

Table 16: .NET Blazor requirements.

Hardware elements
1. Dedicated space for storage
Software elements
1. Visual Studio
2. .NET 6.0 SDK

3.2.2.2 Discourse

Discourse is an open-source Internet forum and mailing list management software application. The application is written with Ember.js and Ruby on Rails. PostgreSQL serves as its back-end database management system. From a usability perspective, Discourse breaks with existing forum software by including features recently popularized by large social networks, such as infinite scrolling, live updates, expanding links, and drag and drop attachments. The source code is distributed under the GNU General Public License version 2. Therefore, Discourse can be self-hosted by anyone. [REF-10]

Table 17: Discourse requirements.

Hardware elements
<ol style="list-style-type: none"> 1. 10 GB of free disk space. 2. Single-core CPU. 3. 64-bit processor 4. 1 GB RAM.
Software elements
<ol style="list-style-type: none"> 1. Linux OS 2. Docker

For continuous integration and development, the following tools will be used:

- JFrog Artifactory
- Slack
- SonarQube
- Jenkins

3.2.2.3 JFrog Artifactory

As the world’s first universal repository, JFrog Artifactory is the mission-critical heart of the JFrog Platform functioning as the single source of truth for all packages, container images and Helm charts, as they move across the entire DevOps pipeline. [REF-11]

Table 18: JFrog Artifactory requirements.

Hardware elements
<ol style="list-style-type: none"> 1. 4-core CPU 2. 4GB RAM 3. SAN backup recommended
Software elements
<ol style="list-style-type: none"> 1. Linux OS or Windows OS 2. Java

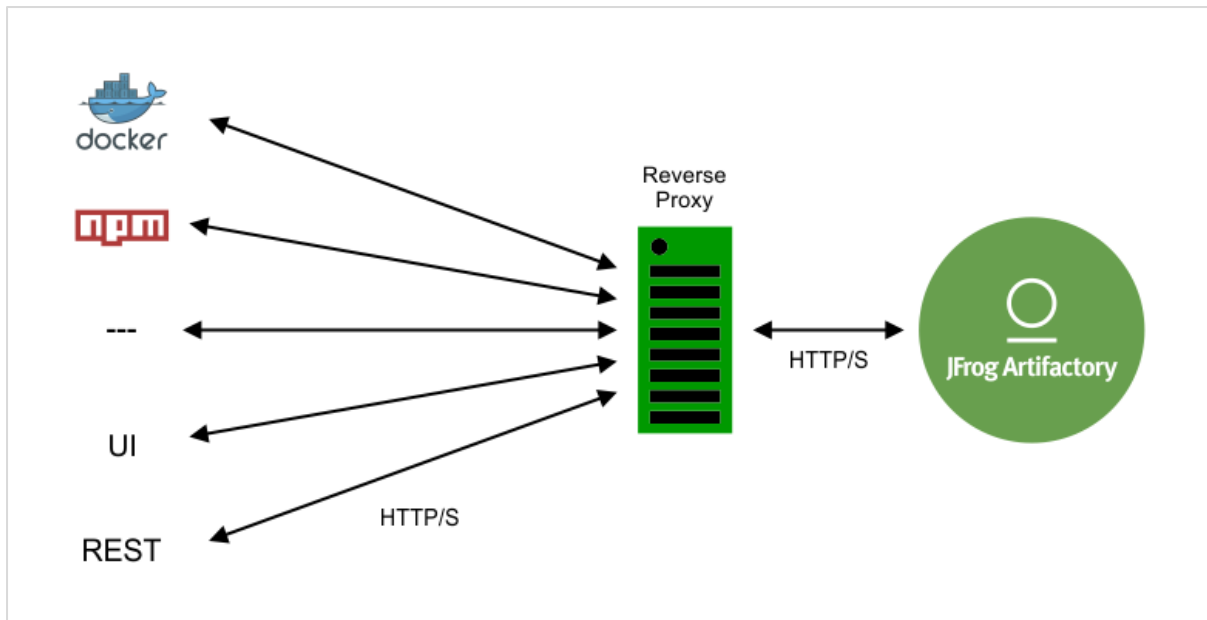


Figure 22: JFrog Artifactory Infrastructure.

3.2.2.4 Slack

Slack is a proprietary business communication platform. Slack offers many IRC-style features, including persistent chat rooms (channels) organized by topic, private groups, and direct messaging. [REF-12]

Table 19: Slack requirements.

Hardware elements
1. Dedicated space for storage
Software elements
1. Linux OS or Windows OS or Mac OS

3.2.2.5 SonarQube

SonarQube is an open-source platform for continuous inspection of code quality to perform automatic reviews with static analysis of code to detect bugs, code smells, and security vulnerabilities on many programming languages. SonarQube offers reports on duplicated code, coding standards, unit tests, code coverage, code complexity, comments, bugs, and security vulnerabilities. Also, can record metrics history and provides evolution graphs.

SonarQube provides fully automated analysis and integration with Maven, Ant, Gradle, MSBuild and continuous integration tools (Atlassian Bamboo, Jenkins, Hudson, etc.). [REF-13]

Table 20: SonarQube requirements.

Hardware elements
<ol style="list-style-type: none"> 1. 2GB RAM 2. 64-bit CPU
Software elements
<ol style="list-style-type: none"> 1. API Connection to retrieve and analyse the code 2. Java 3. Linux OS or Windows OS

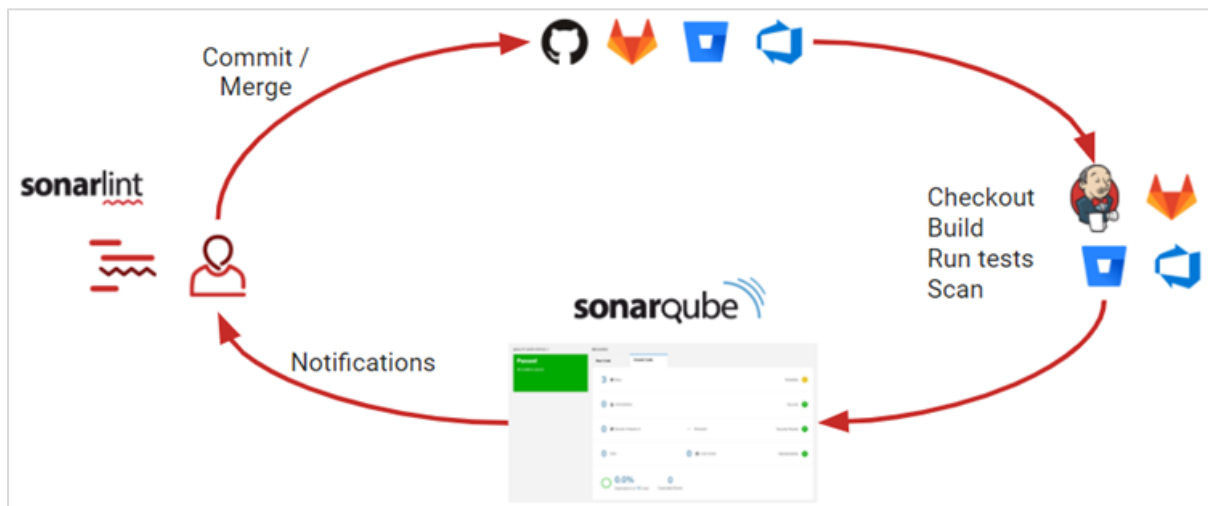


Figure 23: SonarQube Infrastructure Pipeline.

3.2.2.6 Jenkins

Jenkins is an open-source automation server. It helps automate the parts of software development related to building, testing, and deploying, facilitating continuous integration and continuous delivery. It is a server-based system that runs in servlet containers such as Apache Tomcat. It supports version control tools, including AccuRev, CVS, Subversion, Git, Mercurial, Perforce, ClearCase and RTC, and can execute Apache Ant, Apache Maven and sbt based projects as well as arbitrary shell scripts and Windows batch commands. [REF-14]

Table 21: Jenkins requirements.

Hardware elements
<ol style="list-style-type: none"> 1. 256 MB RAM (4GB recommended) 2. 1 GB of drive space (50GB recommended)
Software elements
<ol style="list-style-type: none"> 1. Linux OS

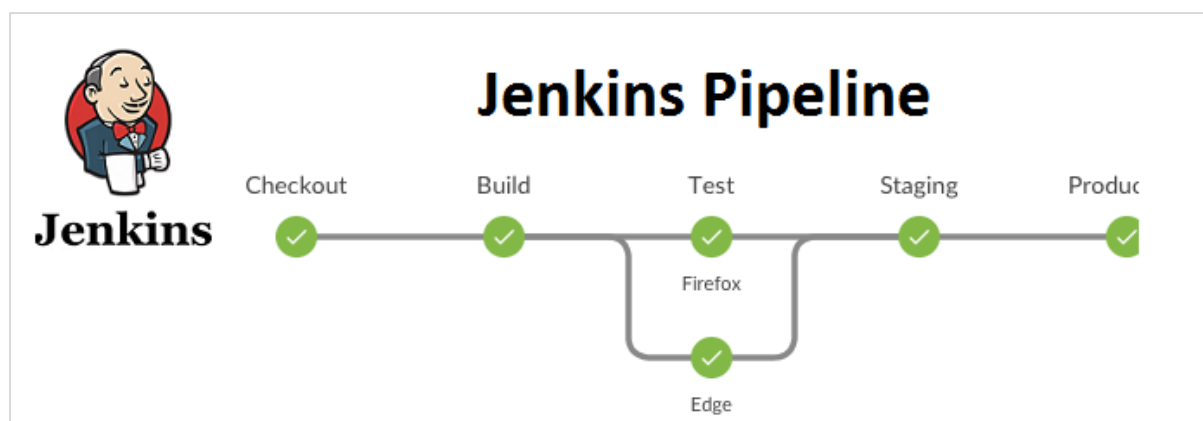


Figure 24: Jenkins Pipeline.

3.2.3 Benchmarking

Benchmarking is made up of a series of proprietary libraries that will be developed for this specific purpose. These libraries will be developed in Python and will use a series of databases to store data for later use. At the same time, Kafka will be used for the flow of information between components.

3.2.3.1 Python Benchmarking libraries

This part of the architecture brings all the libraries written in Python to perform all the desired actions.

Table 22: Python Benchmarking libraries requirements.

Hardware elements
<ol style="list-style-type: none"> 1. Dedicated space for storage 2. Ports for communication

Hardware elements
Software elements
<ol style="list-style-type: none"> 1. Python environment 2. Third-party libraries

3.2.3.2 Benchmarking database driver

Driver necessary for storage all the information relative to benchmarking and all its process.

Table 23: Benchmarking database driver requirements.

Hardware elements
<ol style="list-style-type: none"> 1. Dedicated space for storage 2. Ports for communication
Software elements
<ol style="list-style-type: none"> 1. Specific Driver

3.2.3.3 JupyterHub

JupyterHub is a multi-user Hub that spawns, manages, and proxies multiple instances of single-user Jupyter notebook servers, giving each user their own isolated environment. JupyterHub is used as a user interface for Benchmarking. Custom Python libraries can be imported to access and operate with the Benchmarking data. [REF-15]

Table 24: JupyterHub requirements.

Hardware elements
<ol style="list-style-type: none"> 1. Dedicated space for storage
Software elements

Hardware elements

1. Linux OS
2. Python 3.6 or greater
3. TLS certificate and key for HTTPS communication

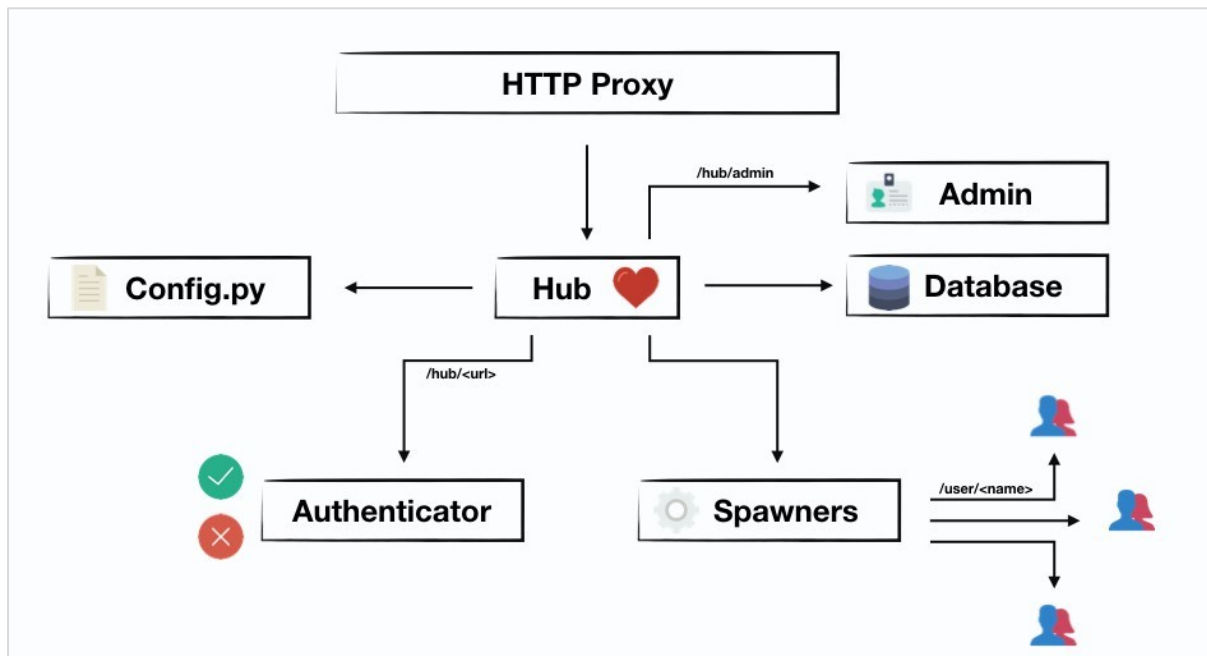


Figure 25: JupyterHub infrastructure.

3.2.3.4 Kafka

Apache Kafka is a framework implementation of a software bus using stream-processing. The project aims to provide a unified, high-throughput, low-latency platform for handling real-time data feeds. Kafka can connect to external systems (for data import/export) via Kafka Connect and provides Kafka Streams, a Java stream processing library. Kafka uses a binary TCP-based protocol that is optimized for efficiency and relies on a "message set" abstraction that naturally groups messages together to reduce the overhead of the network roundtrip. [REF-16]

Table 25: Kafka requirements.

Hardware elements
<ol style="list-style-type: none"> 1. 8 GB RAM 2. 4 CPU cores 3. 500 GB disk storage size 4. Network 1 GbE-10 GbE
Software elements
<ol style="list-style-type: none"> 1. Apache Kafka 2.3.1-0 2. Apache Zookeeper 3.4.14 3. Java 1.8 or later

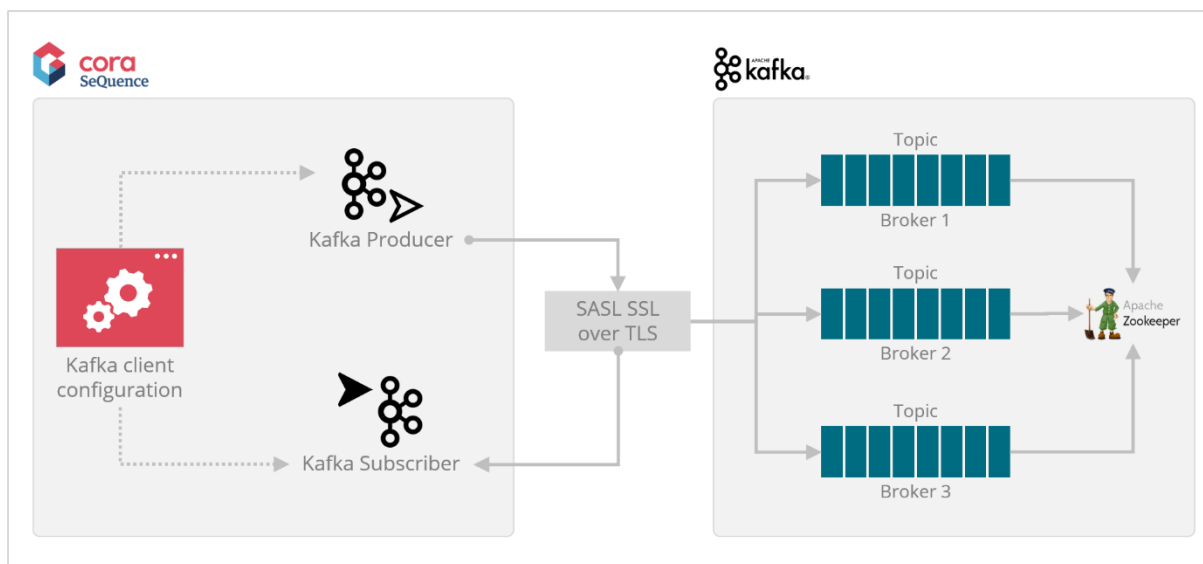


Figure 26: Kafka infrastructure.

3.2.3.5 MongoDB

MongoDB is a source-available cross-platform document-oriented database program. Classified as a NoSQL database program, MongoDB uses JSON-like documents with optional schemas. [REF-17]

Table 26: MongoDB requirements.

Hardware elements
<ol style="list-style-type: none"> 1. Dynamic space for storage 2. 1GB of RAM
Software elements
<ol style="list-style-type: none"> 1. Linux OS or MacOS or Windows OS

3.2.4 Dashboard

The dashboard is responsible for displaying all information to HosmartAI users. This information will be presented on a website, and for that reason Angular will be used for its development.

3.2.4.1 Angular

Angular is a TypeScript-based free and open-source web application framework led by the Angular Team at Google and by a community of individuals and corporations. Angular is a complete rewrite from the same team that built AngularJS. Angular is used as the frontend of the MEAN stack, consisting of MongoDB database, Express.js web application server framework, Angular itself (or AngularJS), and Node.js server runtime environment. [REF-18]

Table 27: Angular requirements.

Hardware elements
<ol style="list-style-type: none"> 1. Dedicated space for storage
Software elements
<ol style="list-style-type: none"> 1. NodeJS 2. NPM Package Manager

4 Conclusion

After an exhaustive review of all the elements that make up the architecture, it can be determined that powerful hardware and a large amount of disk storage space will be needed to house the large number of tools that the HosmartAI system requires.

To this end, the operating system on which the entire architecture will be based will be Linux, with the CentOS 7 distribution being chosen for the deployment of most of the tools. Exceptions may be made for tools that are not compatible, with Ubuntu being deployed for this specific task.

The main elements of the Artificial Intelligence part will be JupyterHub and Acumos. JupyterHub will be used for the management of libraries, while Acumos will be more oriented to the management of data models.

For the Blockchain section, a node will be deployed on the platform with an integrated NGINX module, in order to be able to exchange data in a secure and orderly way.

All the APIs needed within the system will be defined under the OpenAPI standard, being Swagger the default editor and viewer for this part.

For the correct access of the users to the system, NGINX will be used to carry out the redirections, and Keycloak together with an additional database to manage the users and their permissions.

The main element that will manage all the contents related to the Marketplace will be Drupal and the set of tools that will act within the co-creation space for the management of development and continuous integration, as well as knowledge management, has already been defined.

Kafka will be in charge of managing all the information flow and propagating it to the elements that need it.

Finally, the benchmarking section has defined a series of elements, which interconnected in a concrete way will play the role of evaluation of all the necessary tools and applications.

This document has defined the first version of this complex system, resulting in a solid, efficient architecture with the necessary elements to begin to support the whole system, leaving open the possibility of evolution in the future, as new applications and needs appear.

5 References

[REF-01]	WebAssembly introduction. (Accessed on 20/01/2022) https://webassembly.github.io/spec/core/intro/introduction.html
[REF-02]	Docker, Wikipedia. (Accessed on 20/01/2022) https://en.wikipedia.org/wiki/Docker_(software)
[REF-03]	Ansible, Wikipedia. (Accessed on 20/01/2022) https://en.wikipedia.org/wiki/Ansible_(software)
[REF-04]	Nginx, Wikipedia. (Accessed on 20/01/2022) https://en.wikipedia.org/wiki/Nginx
[REF-05]	Acumos website. (Accessed on 20/01/2022) https://www.acumos.org/
[REF-06]	ROS(Robot Operating System), Wikipedia. (Accessed on 20/01/2022) https://en.wikipedia.org/wiki/Robot_Operating_System
[REF-07]	Desde Linux. Keycloak: an open source identity and access management solution. (Accessed on 20/01/2022) https://blog.desdelinux.net/en/keycloak-an-open-source-identity-and-access-management-solution/
[REF-08]	Consul, Hashicorp. (Accessed on 20/01/2022) https://learn.hashicorp.com/collections/consul/getting-started
[REF-09]	Blazor, Microsoft. (Accessed on 20/01/2022) https://dotnet.microsoft.com/en-us/apps/aspnet/web-apps/blazor
[REF-10]	Discourse, Wikipedia. (Accessed on 20/01/2022) https://en.wikipedia.org/wiki/Discourse_(software)
[REF-11]	JFrog, Opsera. (Accessed on 20/01/2022) https://www.opsera.io/ecosystem/jfrog
[REF-12]	Slack, Wikipedia. (Accessed on 20/01/2022) https://en.wikipedia.org/wiki/Slack_(software)
[REF-13]	SonarQube, Wikipedia. (Accessed on 20/01/2022) https://en.wikipedia.org/wiki/SonarQube
[REF-14]	Jenkins, Wikipedia. (Accessed on 20/01/2022) https://en.wikipedia.org/wiki/Jenkins_(software)
[REF-15]	Jupyterhub documentation. (Accessed on 20/01/2022) https://jupyterhub.readthedocs.io/en/stable/
[REF-16]	Kafka, Wikipedia. (Accessed on 20/01/2022) https://en.wikipedia.org/wiki/Apache_Kafka
[REF-17]	MongoDB, Wikipedia. (Accessed on 20/01/2022) https://en.wikipedia.org/wiki/MongoDB
[REF-18]	Angular, Wikipedia. (Accessed on 20/01/2022) https://en.wikipedia.org/wiki/Angular_(web_framework)