Call: HORIZON-HLTH-2021-TOOL-06
Topic: HORIZON-HLTH-2021-TOOL-06-03
Funding Scheme: HORIZON Research and Innovation Actions (RIA)

Grant Agreement no: 101057062

AI powered Data Curation & Publishing Virtual Assistant

Deliverable No. 3.1

VA architecture (Application and Technical)

Approval by the European Commission Pending

Contractual Submission Date:	30/09/2023
Actual Submission Date:	29/09/2023
Responsible partner:	P9: GND

Funded by
the European Union

Grant agreement no.	101057062
Project full title	AIDAVA - AI powered Data Curation & Publishing Virtual Assistant

Deliverable number	**D3.1**
Deliverable title	**VA architecture (Application and Technical),**
Type[1]	R
Dissemination level[2]	PU
Work package number	WP3
Work package leader	P9-GND
Author(s)	Bela Bihari, Daniel Dallos, Lorant Ferencz, Botond Kiss, Zoltan Lazar, Alpar Tana (P9-GND) Isabelle de Zegher (P2 - b!lo) Remzi Celebi, Louis Powell, Shervin Mehryar, Ensar Emir Erol (P1-UM)
Reviewer(s)	Petros Kalendralis (P1 - UM) Markus Plass (P7 - MUG) Eno-Martin Lotman (P8 - NEMC) Emmanuel Benoist (P13 - MIDATA) Dan Bayley (P14 - DME)
Keywords	virtual assistant, data curation, technical architecture, microservice

Document History

Version	Date	Description
V1	05.09.2023	Document ready for internal review
V2	30.09.2023	Final version

Table of Contents

[1] **Type**: Use one of the following codes (in consistence with the Description of the Action):

R:	Document, report (excluding the periodic and final reports)
DEM:	Demonstrator, pilot, prototype, plan designs
DEC:	Websites, patents filing, press & media actions, videos, etc.

[2] **Dissemination level**: Use one of the following codes (in consistence with the Description of the Action)

PU:	Public, fully open, e.g. web
SEN:	Sensitive, limited under conditions of the Grant Agreement

Contents

List of Definitions

The definitions used in the deliverable are based on the AIDAVA Glossary [ref].

Executive Summary

The objective of the AIDAVA project is to prototype an intelligent virtual assistant that will maximise automation in data curation & publishing of heterogeneous personal health data while empowering individual patients when automation is not possible due to lack of contextual information. The solution includes a **backend** and a **frontend** described in the solution design (*see Deliverable D2.3. Solution Design)*.

This deliverable focuses on the technical and data architecture of the AIDAVA prototype and on its deployment, integration and testing with the different evaluation sites.

As the consortium intends to develop a reusable prototype, we first clarify the difference between product and prototype and confirm the importance of taking into account product constraints in the technical architecture to ensure reuse. We also define a set of architecture principles that guided the elaboration of the deliverable.

The technical framework relies on a microservices-based structure, encompassing numerous satellite applications and curation tools. These components will be seamlessly integrated to facilitate the automation process using predefined workflows that incorporate workflow orchestration tools. Additionally, the architecture encompasses connectivity with various medical partners. In this context, the system will acquire input data from file shares or databases and establish connections with health data intermediaries, receiving data either through API endpoints or by utilising SDKs.

The data architecture expands on the components identified in the solution design deliverable, from an implementation perspective.

As the AIDAVA project aims to test the solution in real life with real patients consenting to manage and curate their data, an important part of this deliverable relates to integration in the different evaluation sites, the needed hardware as well as deployment and testing.

This technical architecture is the consolidation of 1 year of efforts across different teams. It provides the consortium with a solid description of the solution that needs to be implemented to successfully meet the objectives of the project. While there are challenges ahead, there is confidence that the first generation (G1) of the prototype can be successfully developed and deployed. The technical architecture - and this document - will be updated for Generation 2 (G2) of the prototype, taking into account the results of the evaluation by patients and clinical sites, the need to integrate more powerful NLP curation tools and an improved human computer interaction front end developed in other work packages of the project.

1 Introduction

This document provides a detailed overview of how the various elements of the system work together to achieve its intended functionality, performance, security, and scalability. It serves as a reference point for developers, engineers, architects, and other stakeholders involved in the design, implementation, and maintenance of the system.

The present Technical Architecture Document encompasses:

1. **System Overview:** An introduction to the system, its purpose, goals, and key features. This section provides a high-level understanding of what the system aims to achieve, building on the extensive work done on requirements in *Deliverable 1.3 Business Requirements* and on automation in *Deliverable 2.2. Data curation and publishing process*.
2. **Architecture Overview:** This section provides a detailed description of the overall system architecture, including the major components, their interactions, and the system's internal and external interfaces. It builds on the *Deliverable 2.3 Solution Design,* and expand on the C4 model representation of the system with the context, container and component level
3. **Satellite Applications:** Detailed descriptions of each major component next to the core system which are ensuring seamless operation, efficiency, and comprehensive monitoring in the system. This includes their functions, responsibilities, and dependencies on other components.
4. **Data Architecture:** A description of how data is stored, retrieved, and managed within the system - building on the different data stores described in *Deliverable 2.3. Solution Design* - and what type of data storages will be used throughout the system.
5. **Curation Tools:** This section primarily centers around the curation workflows employed for onboarding and data curation processes, as well as the tools selected for inclusion within the curation workflows described in *Deliverable 2.2. Details on data curation & publishing process*
6. **Integration:** An explanation of how data flows through the system, how the system integrates with external systems, services, or APIs, including any standards or protocols used for integration. This builds on the requirements on data sources identified in *Deliverable 1.1. Use cases description.*
7. **Hardware Specification:** Description of hardware specifications for the AIDAVA prototype, including docker requirements, access protocols and actual hardware configuration.
8. **Deployment and Testing:** Details about the deployment procedures with containerization, including description of unit tests and integration tests for both frontend and backend, to be executed in the different evaluation sites identified in Deliverable 1.1.

The intended audience for a Technical Architecture Document includes:

1. **Developers and Engineers:** Those directly involved in building and coding the system rely on the document to understand the system's architecture, design, and interactions.
2. **Solution Architects:** Architects use the document to validate design decisions, ensure alignment with the intended system goals, and guide the development team.
3. **Project Managers:** Project managers reference the document to understand the technical aspects of the project, monitor progress, and allocate resources effectively.
4. **Stakeholders:** The European Commission can review the document to gain a technical understanding of the system's capabilities and potential impact on the business.
5. **Future Developers and Maintenance Teams:** As the system evolves, new developers and maintenance teams can rely on the document to understand the system's architecture and design choices.

1.1 Prototype versus product

The AIDAVA project will deliver a prototype of a medical device. The intent of the project is to further develop this prototype into a product, if the result of the evaluation shows that the prototype can solve the problem we want to address and if the discussions with the Sustainability Advisory Board (SAB) confirm the envisioned market potential for the end product.

The main difference between a software product and a software prototype is their level of completion. A software product is a finished and ready-to-use piece of software that has been fully tested and debugged. A software prototype, on the other hand, is an early and incomplete version of a software product that is used to test and refine the design and functionality.

They are different types of prototypes, as explained below (source: Bard)

- **Throwaway prototypes** are low-fidelity prototypes created quickly and easily. They are used to test the feasibility of an idea or to gather feedback from users. Once the prototype has served its purpose, it is discarded and not used to develop the final product.
- **Reusable prototypes** are higher-fidelity prototypes created with the intention of being used to develop the final product. They are more expensive and time-consuming to create than throwaway prototypes, but they can save time and money in the long run by reducing the need for rework.
- **Evolutionary prototypes** are prototypes that are continuously improved and refined over time. They are typically used in agile development projects, where the requirements are constantly changing. Evolutionary prototypes can be either throwaway or reusable.
- **Incremental prototypes** are prototypes that are developed in stages. Each stage adds new functionality to the prototype until it meets the final requirements. Incremental prototypes are often used in waterfall development projects, where the requirements are well-defined at the beginning of the project.
- **Extreme prototyping** is a type of prototyping that is used to quickly and easily create a prototype that meets the user's needs. Extreme prototyping is often used in conjunction with agile development projects.

AIDAVA is developing a **reusable prototype** in 2 stages. Generation 1 (G1) is based on the business requirements acquired as part of Tasks 1.3 and developed with existing curation tools. Generation 2 (G2) is based on the same business requirements improved with some updates after evaluation of the G1 prototype; the curation tools are the ones developed in the projects in Task 5.1 (NLP models) and Task 5.2 (Verification and publishing tools). In addition G2 will be provided with an explainability module.

The fact that we develop a reusable prototype directly impacts the technical architecture which is the foundation of the whole system and cannot be easily modified between the prototype and the product. Concretely, while designing the technical architecture we took into account all requirements - even the ones deemed not needed for the prototype and not implemented during the project - to define the technical architecture of the AIDAVA prototype.

1.2 Architectural principles of AIDAVA

In the design and realisation of the architectural framework, several fundamental principles guide the development of the system:

1. **GDPR Compliance as Core:** At the foundation of the solution lies an unwavering commitment to GDPR compliance. Both the chosen technologies and the associated processes must adhere to the stringent guidelines set forth by the General Data Protection Regulation. This ensures that the handling of sensitive health data is done with the utmost consideration for patient privacy and data protection.
2. **Empowering Productization and Certification:** The architectural blueprint is crafted with an acute awareness of the potential for productization and certification. This strategic foresight underscores the system's scalability and adaptability, allowing it to evolve into a certified solution that can be readily deployed in a variety of contexts, providing consistent value while meeting rigorous quality standards.
3. **Personal Health Knowledge Graph (PHKG) Integration:** The cornerstone of the architectural vision revolves around the Personal Health Knowledge Graph. This intricate construct seamlessly assimilates the diverse health data of an individual, sourced from various repositories. This unified representation not only facilitates data sharing but also fosters a comprehensive understanding of the patient's health journey.
4. **Patient-Centric Consent-Based Transactions:** In the realm of patient data, consent takes centre stage. The architectural philosophy stipulates that every transaction related to identifiable patient data, particularly in the creation and evolution of the PHKG, hinges on the explicit consent of the patient. This empowers patients with control over their data, fostering a relationship of trust and transparency.
5. **Patient-Specific Source Knowledge Graph (SKG) Transformation:** Each data source finds its transformation through the creation of a Patient-Specific Source Knowledge Graph (SKG). These individualised constructs serve as foundational elements, linking and integrating seamlessly into the overarching PHKG. This approach addresses data source-level and integration-level intricacies, ensuring a harmonised and cohesive repository.
6. **Centralised Repository for Dynamic Curation:** The architectural framework envisions the availability of a centralised repository during the curation process. As new data sources (SKGs) are integrated, this central repository serves as a hub to address integration challenges. By doing so, the system streamlines the incorporation of new data sources, minimising disruption and optimising efficiency.
7. **Data Usage with Patient Consent:** The architectural landscape emphasises the importance of patient consent in all aspects of data use. Whether it involves data access or transfers to third parties, the guiding principle mandates that such actions are executed solely with the explicit and informed consent of the patient. This principle reinforces ethical data practices and engenders patient trust.

In essence, these architectural principles epitomise a harmonious fusion of patient-centricity, technological innovation, and regulatory compliance. By weaving these principles into the fabric of the system's design, the architecture not only embraces the complexity of health data management but also positions itself as a beacon of excellence in a landscape marked by evolving data regulations and patient expectations.

2 System Overview

AIDAVA is an innovative project that advances application development with its modern architecture, driving meaningful progress in the field. At its core, AIDAVA adopts microservices as the foundational building blocks, unleashing a host of benefits that traditional monolithic systems cannot match. By breaking the application down into smaller, independent microservices, AIDAVA ensures modularity, scalability, and maintainability.

Microservices architecture is the heart of AIDAVA's design philosophy. Instead of a single monolithic application, AIDAVA comprises numerous microservices, each catering to a specific business function or capability. This approach allows development teams to work on individual services independently, enabling rapid iteration, continuous deployment, and easy updates without affecting the entire system. The microservices architecture fosters a culture of collaboration, as cross-functional teams can focus solely on their respective microservices.

AIDAVA follows the choreography pattern for communication between its microservices. In this pattern, services interact directly with one another using lightweight messaging mechanisms, without relying on a centralised component like an orchestrator. This decentralised communication model brings numerous advantages, including enhanced fault tolerance, adaptability, and the ability to scale independently. The choreography pattern empowers AIDAVA's microservices to respond rapidly to changes in the system and external events, ensuring a highly flexible and dynamic architecture that can gracefully evolve with business requirements.

One of the fundamental principles that AIDAVA adheres to is the single responsibility principle for its microservices. Each microservice is designed to have a specific and well-defined responsibility, encapsulating a particular business capability or logic. By adhering to this principle, AIDAVA ensures that each service remains focused, maintainable, and easy to understand.
AIDAVA takes the container-first approach. Every microservice and its dependencies are packaged into lightweight, isolated containers using Docker technology. This approach ensures easy deployment of containers, allowing for consistent and reproducible deployment across different environments sites, including development, testing, and production. Containerization eliminates the dreaded "works on my machine" problem and streamlines the deployment process, enhancing collaboration and reducing the risk of configuration-related issues.

In addition to its microservices architecture and containerization, AIDAVA incorporates powerful workflow orchestration capabilities which enables the seamless execution of different curation tools.

With AIDAVA's workflow orchestration, the system can intelligently route data to specific curation tools based on predefined workflows described in Deliverable 2.2 - Details on data curation & publishing process. The workflow orchestration system ensures that each tool is invoked at the appropriate stage of the data pipeline, allowing for a comprehensive and structured data curation process.

The integration of workflow orchestration with microservices allows AIDAVA to adapt to changing requirements seamlessly. New curation tools can be added, modified, or removed without disrupting the overall system. The choreography pattern ensures that each microservice is aware of the changes and can interact with the updated workflow orchestration accordingly.

2.1 Main components and their roles (Core)

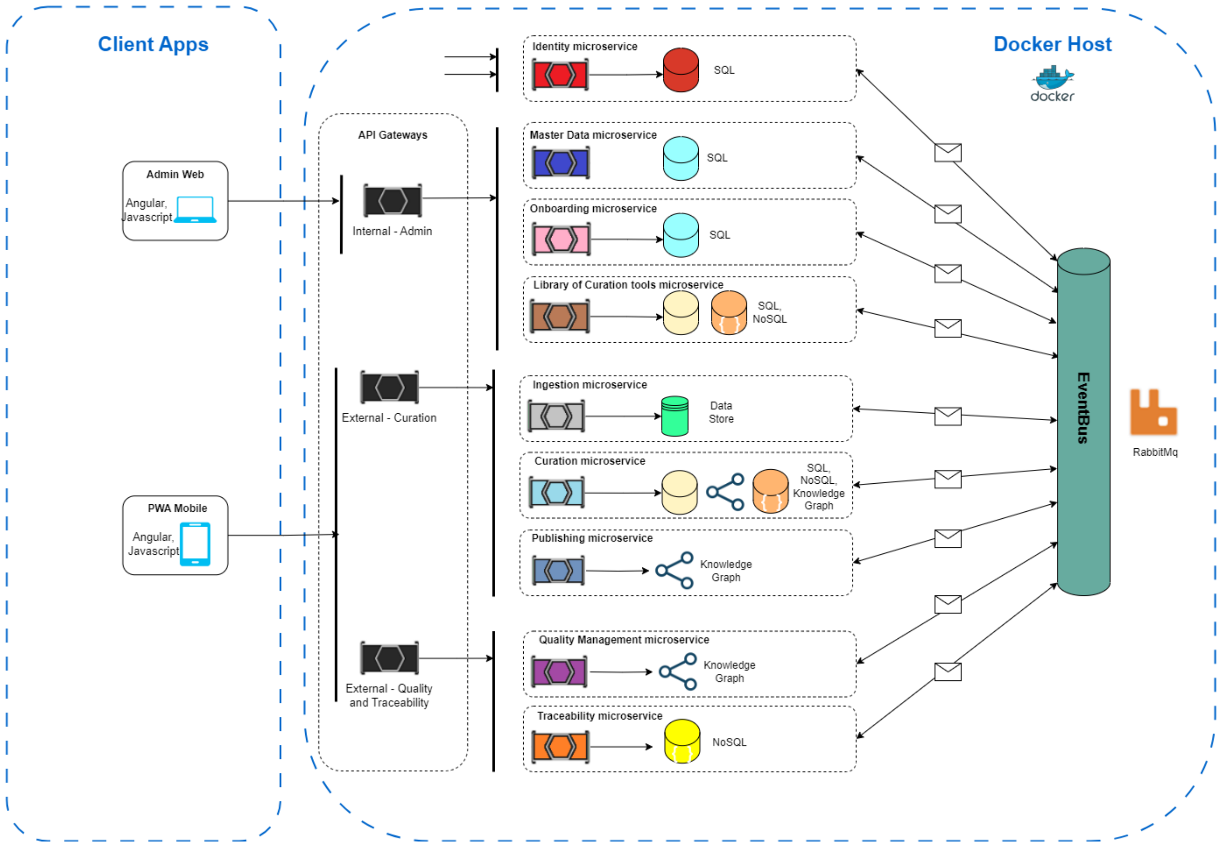


Figure 1. System Overview

2.1.1 Client apps

The Client Apps component encompasses two distinct applications that seamlessly interact with the microservices architecture via the API Gateways (Ocelot). These applications provide engaging user interfaces tailored for various platforms, enhancing accessibility and user experience.

The Web Application serves as a browser-based interface, intended to be used by administrators of the system, to change configurations and settings.

The Progressive Web App (PWA) for Mobile Phones, patients can start the curation of their data, answer questions raised by the system and curators can review, modify or even answer questions which are not applicable for the patients.

Both the Web Application and the Progressive Web App (PWA) share a common interaction pattern with the API Gateway. They initiate API requests through user actions, such as button clicks or form submissions. Ocelot then intelligently routes these requests to the appropriate microservices, facilitating seamless data flow between the client apps and the underlying services.

Furthermore, both applications can potentially leverage the Event Bus component to deliver real-time updates or notifications, contributing to dynamic and interactive user experiences.

2.1.2 API Gateways

The API Gateway serves as the entry point for the previously mentioned client apps and external requests into the microservices ecosystem, providing a centralised endpoint for routing and managing incoming traffic. In the AIDAVA architecture, Ocelot is the chosen solution for implementing the API Gateway.

Ocelot simplifies the routing of client requests to specific microservices based on predefined rules. It acts as a traffic cop, directing incoming requests to the appropriate endpoints while abstracting the complexity of the underlying microservices architecture. This simplifies API management and enhances overall system performance.

Ocelot offers key features and responsibilities that enhance the overall system. It enables dynamic routing, allowing developers to configure routes and route templates, which empowers efficient distribution of incoming requests to the relevant microservices. Additionally, Ocelot supports load balancing across instances of microservices, enhancing scalability and fault tolerance.

Authentication and authorization policies can be enforced by Ocelot, ensuring secure access to microservices. Furthermore, it is capable of caching responses, thereby reducing redundant requests to backend services and improving overall performance.

In terms of interactions with other components in the AIDAVA architecture, Ocelot communicates with the underlying microservices through HTTP requests, forwarding incoming API calls to the appropriate service endpoints.

2.1.3 Microservices

Microservices are a fundamental architectural pattern in the AIDAVA project, emphasising the decomposition of the application into small, independent, and loosely coupled services. Each microservice encapsulates a specific business capability or functionality, allowing for independent development, deployment, and scalability.

Communication between microservices is facilitated through the Event Bus, a central component that enables seamless data exchange and real-time notifications.

Microservices interact with both relational and non-relational databases to store and retrieve data. The relational databases, such as Postgres, offer structured data storage for critical and well-defined schemas. Non-relational databases, including MongoDB , provide flexible data storage options, suitable for scenarios requiring scalability and diverse data structures.

In the AIDAVA architecture, microservices act as the building blocks of the system, working collectively to provide a comprehensive and scalable solution. Their communication through the Event Bus and utilisation of both relational and non-relational databases contribute to the adaptability and responsiveness of the entire microservices ecosystem.

2.1.4 Service communication

Communication between microservices in the AIDAVA architecture happens through the Event Bus, a crucial component that enables efficient and decoupled data exchange. The Event Bus serves as the central hub for transmitting events, notifications, and messages across the microservices ecosystem.

Microservices leverage the Event Bus to achieve seamless interaction and real-time communication. Microservices publish events to the Event Bus, indicating significant actions, state changes, or updates. These events encapsulate data and relevant context, enabling other services to stay informed without direct coupling.

Other microservices can subscribe to specific events of interest, expressing their intention to be notified whenever a particular type of event occurs. This subscription mechanism allows microservices to respond dynamically to changes in the system.

The Event Bus decouples the sender and receiver of events, eliminating direct dependencies between microservices. This decoupling enhances modularity, flexibility, and overall system resilience.

The Event Bus further interacts with other components in the AIDAVA architecture. It collaborates closely with the API Gateway, allowing microservices to communicate with external client applications. Events generated by client actions can be propagated to microservices through the Event Bus.

3 Architecture overview: Formal representation in C4 model

The C4 model offers a structured method for modelling software architecture, employing a series of diagrams to represent a system's context, containers, and components. Its intent is to offer a straightforward, adaptable, and scalable framework, facilitating the clear communication and comprehension of a system's architecture among diverse stakeholders. Frequently adopted in agile and DevOps settings, it supports rapid and effective interaction and teamwork among developers, architects, and interested parties.

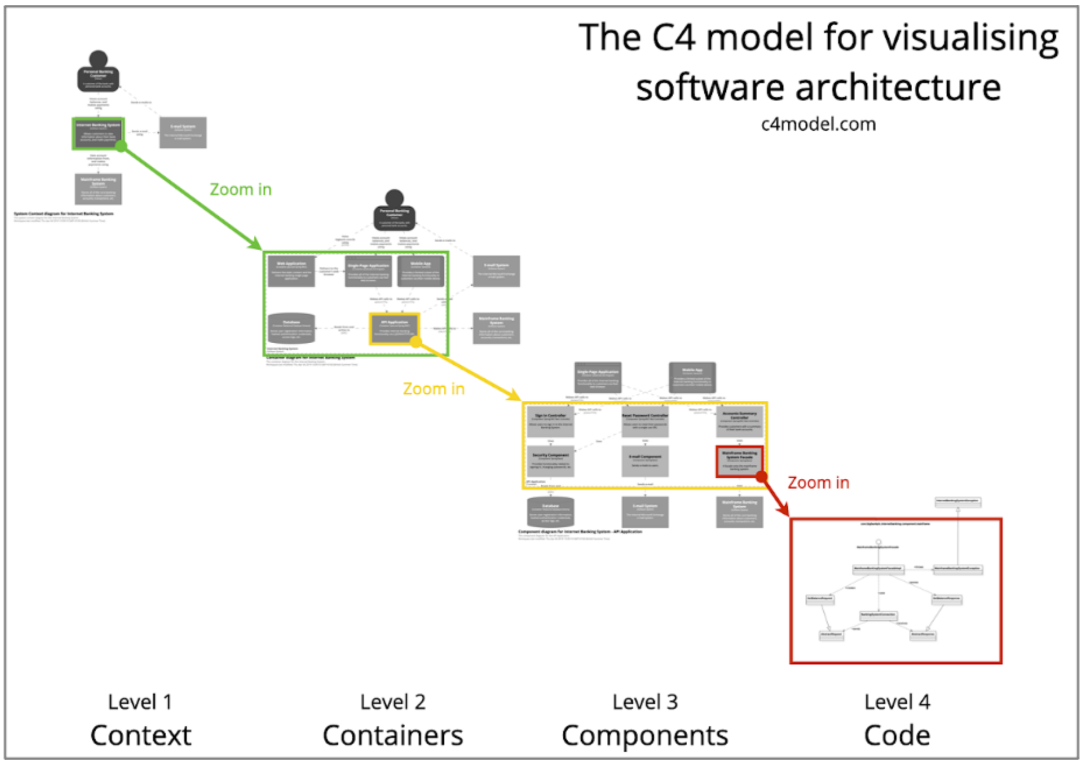


Figure 2. Levels of C4 description

3.1 Context and Container level

The C4 model's Context and Container levels provide a strategic view of software architecture. The Context level focuses on the system's external actors and their interactions, offering a high-level perspective. On the other hand, the Container level delves into the major software components and their interactions, providing a more detailed understanding of the system's architecture.

For an in-depth explanation of the C4 model's Context and Container levels, you can refer to the document titled "D2.3 Solution Design." This document offers comprehensive insights into these levels and how they contribute to effective software architecture visualisation.

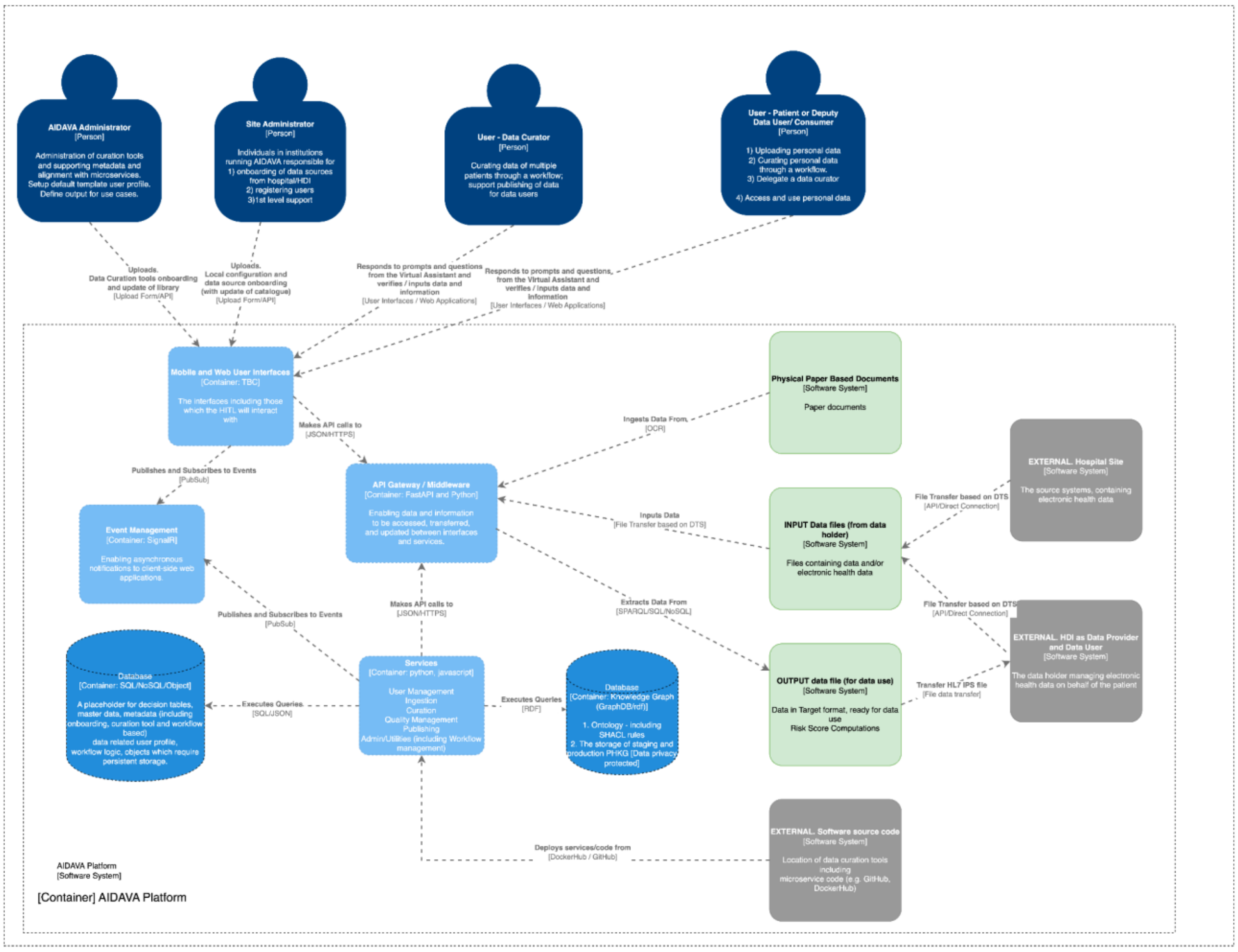

AIDAVA Administrator
[Person]
Administration of curation tools and supporting metadata and alignment with microservices. Setup default template user profile. Define output for use cases.
Site Administrator
[Person]
Individuals in institutions running AIDAVA responsible for 1) onboarding of data sources from hospital/HDI 2) registering users 3)1st level support
User - Data Curator
[Person]
Curating data of multiple patients through a workflow; support publishing of data for data users
User - Patient or Deputy Data User/ Consumer
[Person]
1) Uploading personal data 2) Curating personal data through a workflow. 3) Delegate a data curator 4) Access and use personal data
Uploads. Data Curation tools onboarding and update of library [Upload Form/API]
Uploads. Local configuration and data source onboarding (with update of catalogue) [Upload Form/API]
Responds to prompts and questions from the Virtual Assistant and verifies / inputs data and information [User Interfaces / Web Applications]
Responds to prompts and questions, from the Virtual Assistant and verifies / inputs data and information [User Interfaces / Web Applications]
Mobile and Web User Interfaces
[Container: TBC]
The interfaces including those which the HITL will interact with
Makes API calls to [JSON/HTTPS]
Publishes and Subscribes to Events [PubSub]
API Gateway / Middleware
[Container: FastAPI and Python]
Enabling data and information to be accessed, transferred, and updated between interfaces and services.
Event Management
[Container: SignalR]
Enabling asynchronous notifications to client-side web applications.
Physical Paper Based Documents
[Software System]
Paper documents
Ingests Data From, [OCR]
Inputs Data [File Transfer based on DTS]
Extracts Data From [SPARQL/SQL/NoSQL]
INPUT Data files (from data holder)
[Software System]
Files containing data and/or electronic health data
EXTERNAL. Hospital Site
[Software System]
The source systems, containing electronic health data
File Transfer based on DTS [API/Direct Connection]
File Transfer based on DTS [API/Direct Connection]
EXTERNAL. HDI as Data Provider and Data User
[Software System]
The data holder managing electronic health data on behalf of the patient
Transfer HL7 IPS file [File data transfer]
OUTPUT data file (for data use)
[Software System]
Data in Target format, ready for data use
Risk Score Computations
Publishes and Subscribes to Events [PubSub]
Makes API calls to [JSON/HTTPS]
Database
[Container: SQL/NoSQL/Object]
A placeholder for decision tables, master data, metadata (including onboarding, curation tool and workflow based) data related user profile, workflow logic, objects which require persistent storage.
Executes Queries [SQL/JSON]
Services
[Container: python, javascript]
User Management Ingestion Curation Quality Management Publishing Admin/Utilities (including Workflow management)
Executes Queries [RDF]
Database
[Container: Knowledge Graph (GraphDB/rdf)]
1. Ontology - including SHACL rules 2. The storage of staging and production PHKG [Data privacy protected]
Deploys services/code from [DockerHub / GitHub]
EXTERNAL. Software source code
[Software System]
Location of data curation tools including microservice code (e.g. GitHub, DockerHub)
AIDAVA Platform
[Software System]
[Container] AIDAVA Platform

Figure 3. AIDAVA – C4 Container level

3.2 Components level

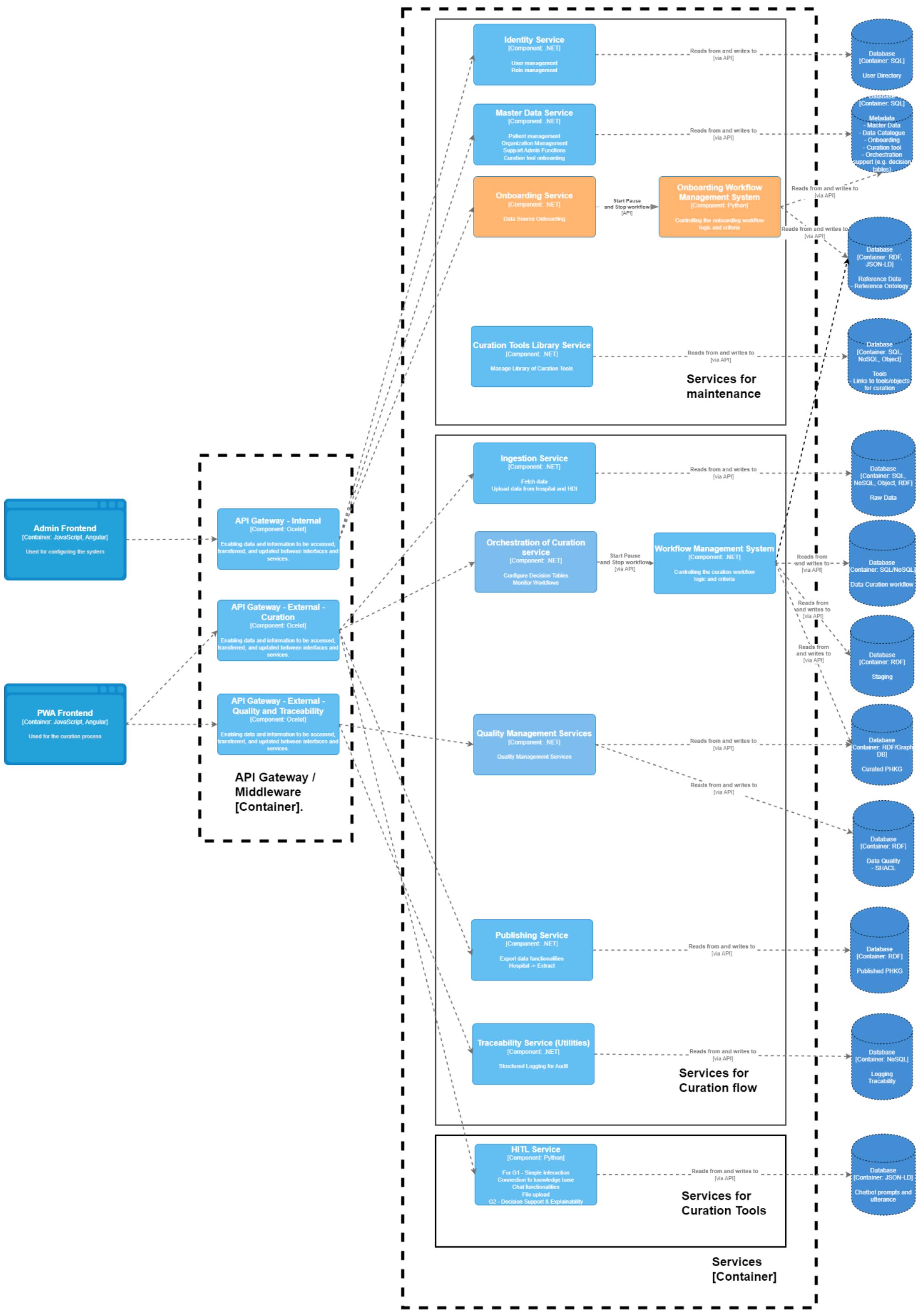

Admin Frontend
[Container: JavaScript, Angular]
Used for configuring the system
PWA Frontend
[Container: JavaScript, Angular]
Used for the curation process
API Gateway - Internal
[Component: Ocelot]
Enabling data and information to be accessed, transferred, and updated between interfaces and services.
API Gateway - External - Curation
[Component: Ocelot]
Enabling data and information to be accessed, transferred, and updated between interfaces and services.
API Gateway - External - Quality and Traceability
[Component: Ocelot]
Enabling data and information to be accessed, transferred, and updated between interfaces and services.
API Gateway / Middleware [Container].
Identity Service
[Component: .NET]
User management
Role management
Master Data Service
[Component: .NET]
Patient management
Organization Management
Support Admin Functions
Curation tool onboarding
Onboarding Service
[Component: .NET]
Data Source Onboarding
Start Pause and Stop workflow [API]
Onboarding Workflow Management System
[Component: Python]
Controlling the onboarding workflow logic and criteria
Curation Tools Library Service
[Component: .NET]
Manage Library of Curation Tools
Services for maintenance
Ingestion Service
[Component: .NET]
Fetch data
Upload data from hospital and HDI
Orchestration of Curation service
[Component: .NET]
Configure Decision Tables
Monitor Workflows
Start Pause and Stop workflow [via API]
Workflow Management System
[Component: .NET]
Controlling the curation workflow logic and criteria
Quality Management Services
[Component: .NET]
Quality Management Services
Publishing Service
[Component: .NET]
Export data functionalities
Hospital -> Extract
Traceability Service (Utilities)
[Component: .NET]
Structured Logging for Audit
Services for Curation flow
HITL Service
[Component: Python]
For G1 - Simple Interaction
Connection to knowledge base
Chat functionalities
File upload
G2 - Decision Support & Explainability
Services for Curation Tools
Services [Container]
Reads from and writes to [via API]
Database [Container: SQL] User Directory
Database [Container: SQL] Metadata - Master Data - Data Catalogue - Onboarding - Curation tool - Orchestration support (e.g. decision tables)
Database [Container: RDF, JSON-LD] Reference Data - Reference Ontology
Database [Container: SQL, NoSQL, Object] Tools - Links to tools/objects for curation
Database [Container: SQL, NoSQL, Object, RDF] Raw Data
Database [Container: SQL/NoSQL] Data Curation workflow
Database [Container: RDF] Staging
Database [Container: RDF/Graph DB] Curated PHKG
Database [Container: RDF] Data Quality - SHACL
Database [Container: RDF] Published PHKG
Database [Container: NoSQL] Logging Tracability
Database [Container: JSON-LD] Chatbot prompts and utterance

Figure 4. AIDAVA – C4 Components level

At this level, we first consider the distinct client applications that form part of the architecture. These client applications represent the user interfaces and interfaces through which users interact with the system. Understanding their roles and responsibilities is vital for comprehending the overall system behaviour.

The next part is the API gateways. These gateways serve as essential intermediaries, enabling controlled access to every user of the AIDAVA system. It is not allowed to communicate directly with the microservices only through the api gateways. They play a role in managing communication between clients and the backend services, ensuring security, load distribution, and sometimes even protocol translation.

Zooming further in, we explore the different microservices that constitute the backbone of the system. Each microservice encapsulates specific functionality, promoting modularity and maintainability. These microservices are interconnected by major components that facilitate communication and data flow, forming the intricate web of the application's logic.

Within this ecosystem, various types of databases come into play. SQL databases, NoSQL databases, and Graph databases interconnect to store and manage data efficiently. SQL databases handle structured data, NoSQL databases manage unstructured or semi-structured data, while Graph databases excel at managing highly interconnected data models.

Taking a step back, we observe the strategic grouping of microservices into distinct categories. Maintenance services handle tasks related to system health, updates, and monitoring. Curation flow services manage the flow of data and processes, ensuring that the system operates seamlessly. Curation tools, as microservices, provide specialised functionalities for refining and enhancing data quality.

Incorporating the C4 model's component level perspective enables a comprehensive understanding of how these diverse elements interact and contribute to the architecture's overall behaviour. This abstraction layer enhances communication among architects, developers, and stakeholders, fostering efficient collaboration and informed decision-making throughout the software development lifecycle.

4 Satellite applications

Within the AIDAVA project, Satellite Applications play a pivotal role in bolstering and enhancing the overall microservices architecture. These applications act as supporting elements, working in tandem with the core components to ensure seamless operation, efficiency, and comprehensive monitoring.

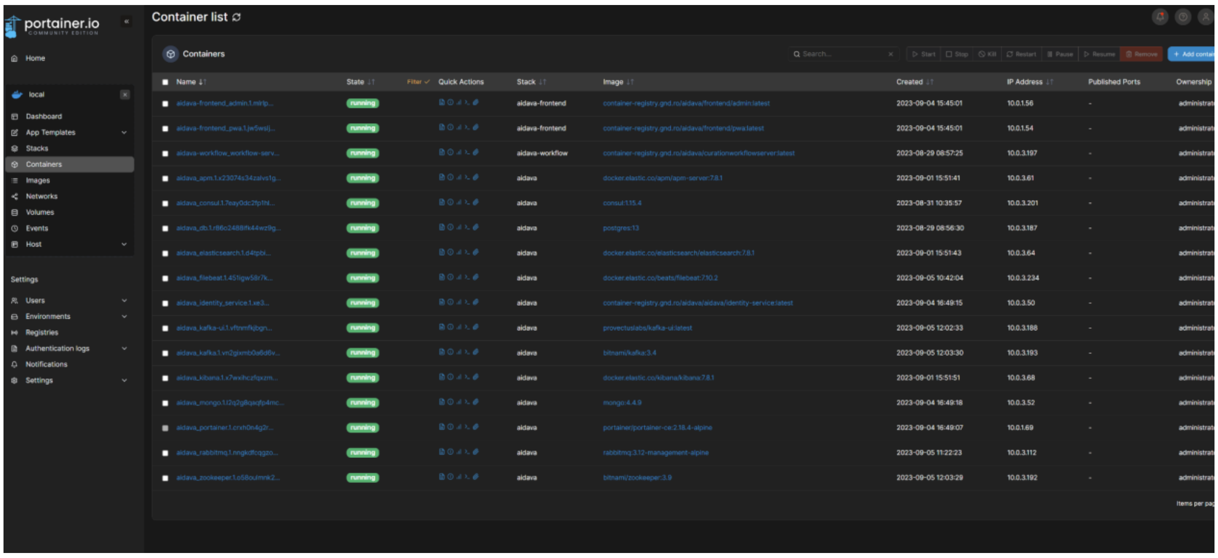
Satellite Applications are designed to contribute to the robustness and reliability of our architecture. They encompass a diverse range of functionalities, including monitoring, state management, and logging, which are vital for maintaining the health and performance of the entire system.

One of the core aspects of Satellite Applications is monitoring. Given the extensive number of Docker images and microservices involved, it is crucial to have a comprehensive monitoring system in place. This enables real-time tracking of the system's behaviour, resource utilisation, and overall performance. Monitoring provides insights into potential issues, bottlenecks, and anomalies, facilitating proactive measures to ensure optimal operation and responsiveness.

Another significant role of Satellite Applications is in managing the state of various components. State management encompasses maintaining consistent configurations, settings, and data across different instances and environments. By centralising and controlling the state, Satellite Applications contribute to the stability and uniformity of the architecture, reducing potential inconsistencies and operational challenges.

Additionally, Satellite Applications excel in logging, a critical element for diagnosing issues, tracking activities, and ensuring accountability. Proper logging mechanisms ensure that important events, errors, and activities are captured, enabling effective troubleshooting and analysis.

4.1 Portainer



Container list

Containers

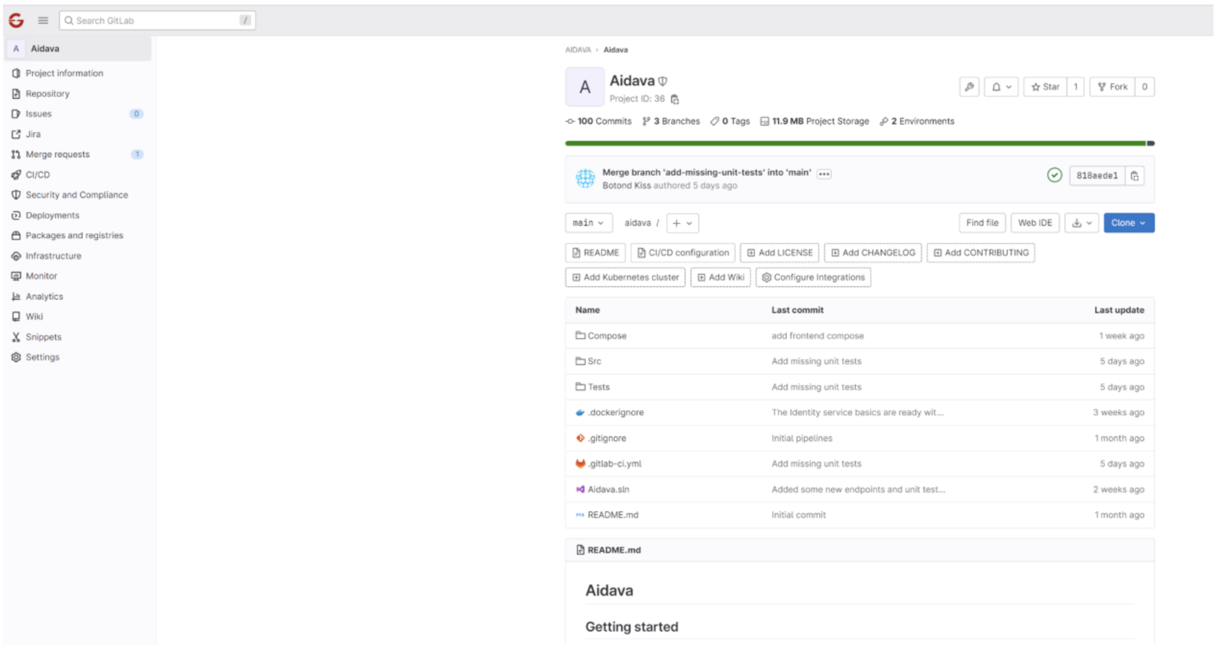
Name	State	Quick Actions	Stack	Image	Created	IP Address	Published Ports	Ownership
aidava-frontend_admin.1.mirlp...	running		aidava-frontend	container-registry.gnd.ro/aidava/frontend/admin:latest	2023-09-04 15:45:01	10.0.1.56	-	administrat
aidava-frontend_pwa.1.jw5wslj...	running		aidava-frontend	container-registry.gnd.ro/aidava/frontend/pwa:latest	2023-09-04 15:45:01	10.0.1.54	-	administrat
aidava-workflow_workflow-serv...	running		aidava-workflow	container-registry.gnd.ro/aidava/curationworkflowserver:latest	2023-08-29 08:57:25	10.0.3.197	-	administrat
aidava_apm.1.x23074s34zalvs1g...	running		aidava	docker.elastic.co/apm/apm-server:7.8.1	2023-09-01 15:51:41	10.0.3.61	-	administrat
aidava_consul.1.7eay0dc2fp1hl...	running		aidava	consul:1.15.4	2023-08-31 10:35:57	10.0.3.201	-	administrat
aidava_db.1.r86o2488ifk44wz9g...	running		aidava	postgres:13	2023-08-29 08:56:30	10.0.3.187	-	administrat
aidava_elasticsearch.1.d4tpbi...	running		aidava	docker.elastic.co/elasticsearch/elasticsearch:7.8.1	2023-09-01 15:51:43	10.0.3.64	-	administrat
aidava_filebeat.1.451igw58r7k...	running		aidava	docker.elastic.co/beats/filebeat:7.10.2	2023-09-05 10:42:04	10.0.3.234	-	administrat
aidava_identity_service.1.xe3...	running		aidava	container-registry.gnd.ro/aidava/aidava/identity-service:latest	2023-09-04 16:49:15	10.0.3.50	-	administrat
aidava_kafka-ui.1.vftnmfkjbgn...	running		aidava	provectuslabs/kafka-ui:latest	2023-09-05 12:02:33	10.0.3.188	-	administrat
aidava_kafka.1.vn2gixmb0a6d6v...	running		aidava	bitnami/kafka:3.4	2023-09-05 12:03:30	10.0.3.193	-	administrat
aidava_kibana.1.x7wxihczfqxzm...	running		aidava	docker.elastic.co/kibana/kibana:7.8.1	2023-09-01 15:51:51	10.0.3.68	-	administrat
aidava_mongo.1.l2q2g8qaqfp4mc...	running		aidava	mongo:4.4.9	2023-09-04 16:49:18	10.0.3.52	-	administrat
aidava_portainer.1.crxh0n4g2r...	running		aidava	portainer/portainer-ce:2.18.4-alpine	2023-09-04 16:49:07	10.0.1.69	-	administrat
aidava_rabbitmq.1.nngkdfcqgzo...	running		aidava	rabbitmq:3.12-management-alpine	2023-09-05 11:22:23	10.0.3.112	-	administrat
aidava_zookeeper.1.o58oulmnk2...	running		aidava	bitnami/zookeeper:3.9	2023-09-05 12:03:29	10.0.3.192	-	administrat

Figure 5. AIDAVA – Portainer

Portainer is an open-source, lightweight, and user-friendly management tool designed to simplify the deployment and management of Docker containers and container-based applications. As containers and containerization have become increasingly popular for application deployment, tools like Portainer have emerged to provide a graphical interface that makes managing containers more accessible to users of all skill levels. At its core, Portainer serves as a container management platform, offering a web-based interface that allows users to interact with Docker and Kubernetes clusters effortlessly.

Portainer is shipped as an integral part of the AIDAVA, playing a key role in the seamless operation of the prototype. This is ensured by providing system operators real-time monitoring and management capabilities of running containers, oversight of inter-container communication and effective management of data volumes.

4.2 Gitlab



Aidava
Project ID: 36

100 Commits 3 Branches 0 Tags 11.9 MB Project Storage 2 Environments

Merge branch 'add-missing-unit-tests' into 'main'
Botond Kiss authored 5 days ago
818aede1

Name	Last commit	Last update
Compose	add frontend compose	1 week ago
Src	Add missing unit tests	5 days ago
Tests	Add missing unit tests	5 days ago
.dockerignore	The Identity service basics are ready wit...	3 weeks ago
.gitignore	Initial pipelines	1 month ago
.gitlab-ci.yml	Add missing unit tests	5 days ago
Aidava.sln	Added some new endpoints and unit test...	2 weeks ago
README.md	Initial commit	1 month ago

README.md

Aidava

Getting started

Figure 6. AIDAVA – Gitlab

GitLab is a comprehensive web-based platform for version control and collaboration that enables software development teams to manage their source code, track changes, collaborate on projects, and streamline the software development process. It provides a complete DevOps lifecycle, integrating various tools and functionalities into a single unified environment. GitLab is often referred to as a "single application for the entire DevOps lifecycle" because of its extensive set of features.

At the core of GitLab is its version control system, based on Git. Git is a distributed version control system that allows developers to keep track of changes in their codebase over time. It enables multiple developers to work on the same project simultaneously and manage code versions efficiently. GitLab leverages Git's capabilities and adds a layer of collaboration features on top, making it easier for developers to collaborate effectively on codebases. GitLab's feature set includes various components that cater to different stages of the software development lifecycle, such as Code Review and Collaboration features, Continuous Integration and Continuous Deployment and Container Registry for Docker images.

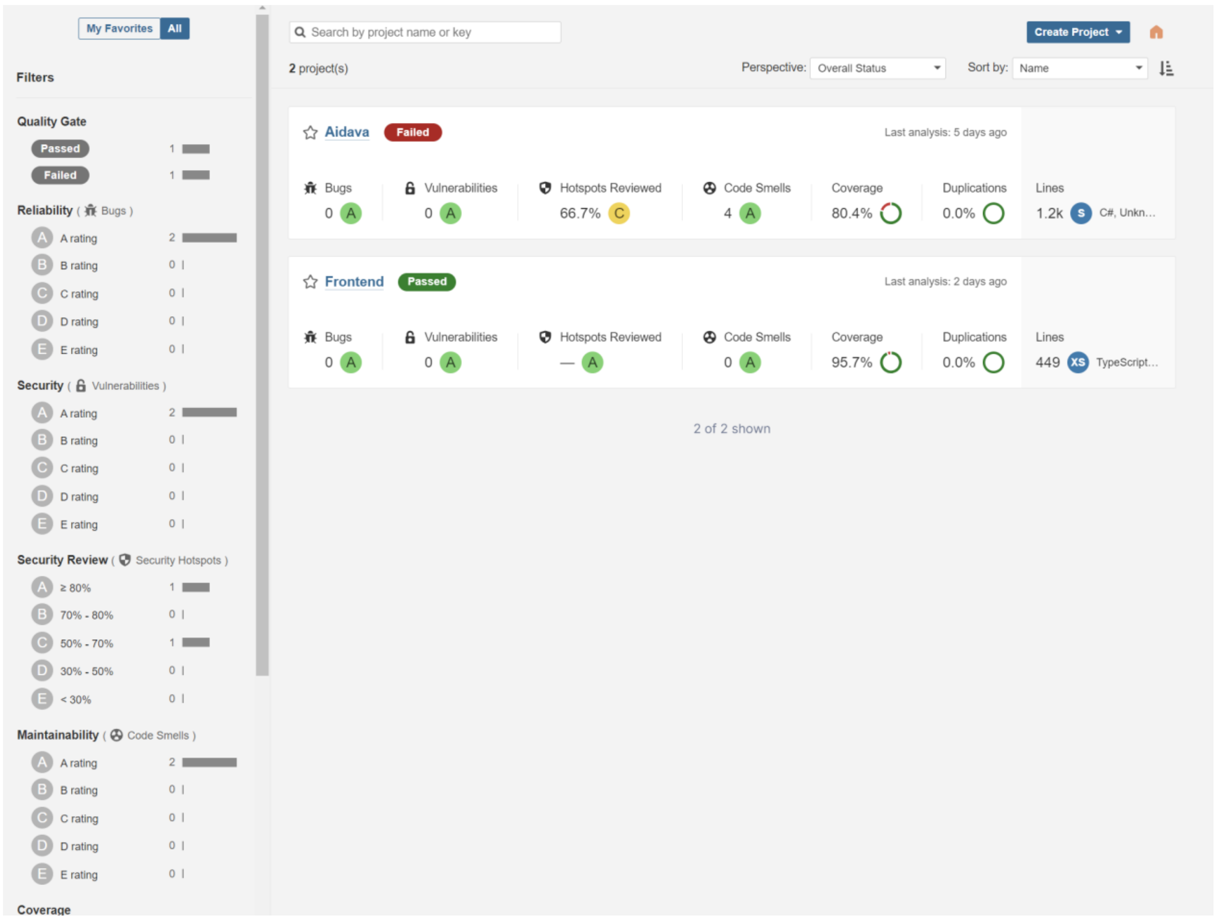
As GitLab's features have a significant role in the operation of the AIDAVA prototype, it won't be shipped during the deployment process. GitLab's on-premise version is hosted in GND's internal infrastructure and it is used as a version control, collaboration and CI/CD tool during the development cycle.

As a distributed version control system, it enables working on features of the AIDAVA prototype simultaneously, keeping a history of code changes over the development cycle.

As a collaboration tool, GitLab allows developers to conduct code reviews on each other's codes, thereby enhancing code quality and ensuring code maintainability in the long run.

GitLab also comes with feature rich, intuitive CI/CD capabilities, thus allowing developers to combine the version control capabilities with a series of CI/CD stages. During development, every commit (Git term, explanation needed?) triggers a well established sequence of CI/CD stages, such as linting (static code analysis), unit testing, end-to-end testing, SonarQube analysis (4.3). Upon finishing a feature, the sequence is extended by a build stage, responsible for creating Docker images, and a deployment stage.

4.3 SonarQube



Filters

Quality Gate

Passed	1
Failed	1

Reliability (Bugs)

A rating	2
B rating	0
C rating	0
D rating	0
E rating	0

Security (Vulnerabilities)

A rating	2
B rating	0
C rating	0
D rating	0
E rating	0

Security Review (Security Hotspots)

A ≥ 80%	1
B 70% - 80%	0
C 50% - 70%	1
D 30% - 50%	0
E < 30%	0

Maintainability (Code Smells)

A rating	2
B rating	0
C rating	0
D rating	0
E rating	0

Coverage

2 project(s) Perspective: Overall Status Sort by: Name

Aidava Failed — Last analysis: 5 days ago

Bugs	Vulnerabilities	Hotspots Reviewed	Code Smells	Coverage	Duplications	Lines
0 A	0 A	66.7% C	4 A	80.4%	0.0%	1.2k S C#, Unkn...

Frontend Passed — Last analysis: 2 days ago

Bugs	Vulnerabilities	Hotspots Reviewed	Code Smells	Coverage	Duplications	Lines
0 A	0 A	— A	0 A	95.7%	0.0%	449 XS TypeScript...

2 of 2 shown

Figure 7. AIDAVA – SonarQube

SonarQube is a robust and versatile platform designed to enhance code quality and maintainability across software development projects. Operating as a comprehensive code analysis and inspection tool, SonarQube aids development teams in identifying and rectifying issues within their codebase.

By conducting automated evaluations, SonarQube assesses source code for a range of factors including coding standards adherence, potential vulnerabilities, overall code complexity, and more. This process yields actionable insights that empower developers to proactively address issues, resulting in the production of more reliable, secure, and maintainable software. Moreover, SonarQube incorporates a quality gate mechanism, which allows projects to define specific criteria for code quality. If the codebase meets these pre-defined criteria, it's deemed suitable for further development stages; otherwise, it prompts developers to make necessary improvements before proceeding.

4.4 Consul

Consul by HashiCorp is a versatile tool designed to streamline the management of environments, particularly in the context of microservices and cloud-native architectures. It offers a range of features that simplify the complexities of handling large-scale distributed systems.

At its core, Consul provides a set of concepts that enable efficient communication and coordination between services. One of its key functionalities is service discovery, which allows services to automatically register themselves and locate other services within a network. This eliminates the need for manual intervention when services are added, removed, or scaled, ensuring seamless interactions between various components of an application.

In addition to service discovery, Consul offers health checking capabilities. Services can regularly report their health status, which Consul monitors. If a service becomes unhealthy, Consul can dynamically remove it from the pool of available services, preventing faulty instances from receiving traffic. This proactive health monitoring enhances the reliability and performance of applications.

Consul also features a distributed key-value store, which serves as a dynamic configuration management tool. This means that applications can store and retrieve configuration data in real time, without requiring application restarts. This flexibility supports agile development practices and allows for on-the-fly configuration updates, enhance agility and reduce downtime.

Consul offers features such as encrypted communication (TLS) and access control lists (ACLs). This safeguards communication between services and restricts access to sensitive data and resources. Additionally, Consul enables service segmentation, allowing us to control which services can communicate with each other, enhancing overall security posture.

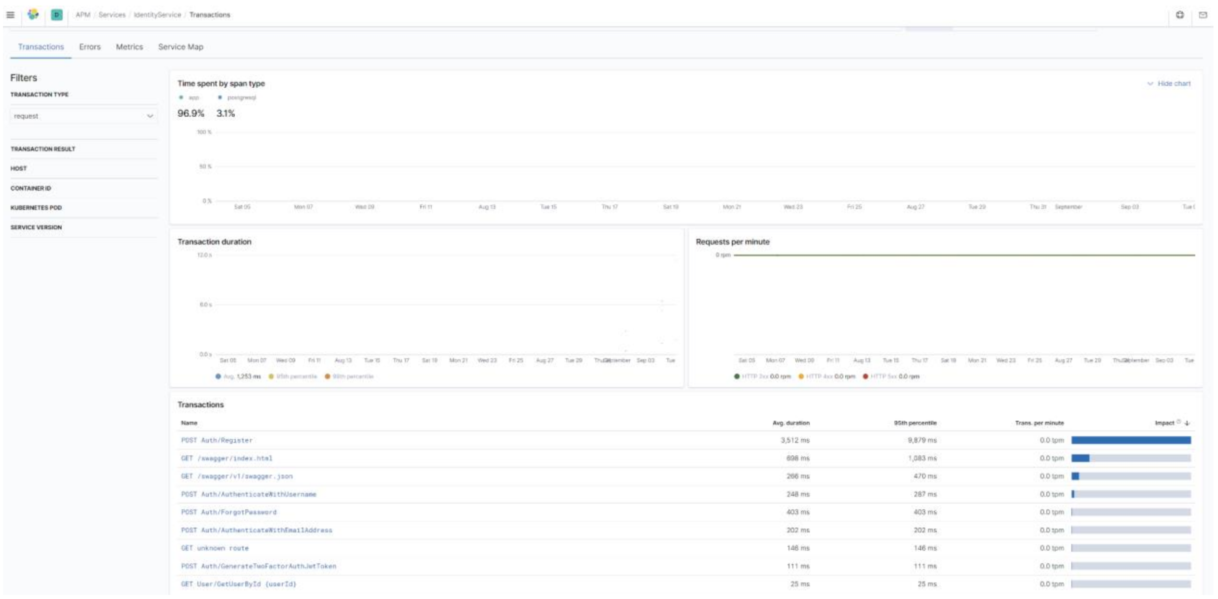
4.5 Elastic Search

Elasticsearch is an open-source, distributed search and analytics engine designed to handle and process vast volumes of data with remarkable speed and efficiency. Functioning as a robust and scalable full-text search platform, Elasticsearch is primarily utilised to index, store, and search through diverse types of data, ranging from structured to unstructured information.
At its core, Elasticsearch employs the concept of a distributed, schema-less JSON-based document store. It accommodates the seamless storage of a multitude of documents, each of which can vary in structure while conforming to the underlying data model. This flexibility makes it particularly well-suited for scenarios where data formats may evolve over time or where heterogeneity in data structure is prevalent.

One of Elasticsearch's key strengths lies in its capacity to provide near real-time search capabilities. By using a distributed architecture that divides data into shards and distributes them across a cluster of nodes, Elasticsearch ensures high availability and fault tolerance. Its inherent horizontal scalability allows it to handle large datasets with ease, while the built-in replication mechanisms contribute to data redundancy and reliability.

In the AIDAVA project, we set up the microservices to send their log data to Elasticsearch and establish index patterns for effective data organisation and storage. Defining mappings becomes crucial as it specifies the data formats for various log fields, enabling Elasticsearch to accurately comprehend and handle the information.

4.6 Kibana



Transactions

Name	Avg. duration	95th percentile	Trans. per minute	Impact
POST Auth/Register	3,512 ms	9,879 ms	0.0 tpm	
GET /swagger/index.html	698 ms	1,083 ms	0.0 tpm	
GET /swagger/v1/swagger.json	266 ms	470 ms	0.0 tpm	
POST Auth/AuthenticateWithUsername	248 ms	287 ms	0.0 tpm	
POST Auth/ForgotPassword	403 ms	403 ms	0.0 tpm	
POST Auth/AuthenticateWithEmailAddress	202 ms	202 ms	0.0 tpm	
GET unknown route	146 ms	146 ms	0.0 tpm	
POST Auth/GenerateTwoFactorAuthJwtToken	111 ms	111 ms	0.0 tpm	
GET User/GetUserById {userId}	25 ms	25 ms	0.0 tpm	

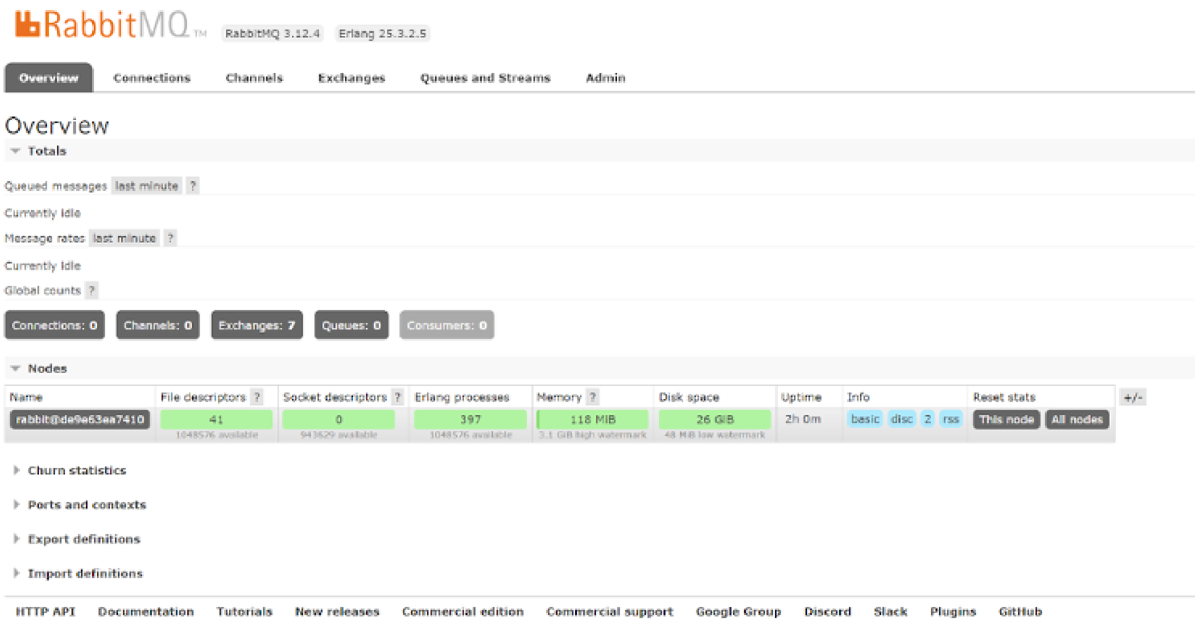
Figure 8. AIDAVA – Kibana

Kibana is an open-source data visualisation and exploration platform developed to work seamlessly with the Elasticsearch search and analytics engine. Positioned as a critical component within the Elastic Stack, Kibana enables users to transform complex datasets into insightful visualisations, dashboards, and reports.

At its core, Kibana simplifies the process of interacting with data stored in Elasticsearch by providing an intuitive web-based interface. This interface allows users to construct and customise a wide range of visual elements, including line charts, bar graphs, pie charts, maps, and more, using the data indexed in Elasticsearch. These visualisations facilitate the exploration and understanding of data trends, patterns, and anomalies.

Utilising Kibana in the project, we have the capability to execute searches, aggregations, and apply filters to our log data. By integrating diverse visualisations, we can create a comprehensive perspective of both the performance of our microservices and the associated log information.

4.7 RabbitMQ



Overview

Totals

Queued messages last minute ?

Currently idle

Message rates last minute ?

Currently idle

Global counts ?

Connections: 0 Channels: 0 Exchanges: 7 Queues: 0 Consumers: 0

Nodes

Name	File descriptors ?	Socket descriptors ?	Erlang processes	Memory ?	Disk space	Uptime	Info	Reset stats
rabbit@de9e63ea7410	41 1048576 available	0 943629 available	397 1048576 available	118 MiB 3.1 GiB high watermark	26 GiB 48 MiB low watermark	2h 0m	basic disc 2 rss	This node All nodes

Churn statistics

Ports and contexts

Export definitions

Import definitions

Figure 9. AIDAVA – RabbitMQ

RabbitMQ is an open-source message broker software that facilitates the transfer of data between different systems, applications, and services. This intermediary plays a pivotal role in ensuring that messages are efficiently routed, managed, and delivered, thereby contributing to enhanced scalability, reliability, and flexibility in diverse software environments.

RabbitMQ offers a myriad of benefits that elevate it to a central position in the realm of messaging and communication infrastructure. One of its key strengths lies in reliable message delivery, which is facilitated through various messaging patterns, including publish-subscribe, request-response, and point-to-point. This reliability ensures that messages reach their intended recipients without fail, contributing to data consistency and real-time responsiveness in applications.
Central to RabbitMQ's functionality are message queues. These queues serve as temporary storage for messages until they are ready to be processed. This decoupling of producers and consumers ensures smoother traffic management and enhanced fault tolerance, a critical aspect in maintaining system reliability.

Message persistence is another crucial feature of RabbitMQ. It can persist messages to disk, safeguarding valuable data in case of system failures and ensuring the durability of messages. This resilience contributes to the overall robustness of the communication infrastructure.

In microservices architecture, RabbitMQ's significance remains pronounced. Microservices, with their emphasis on independent, deployable services, demand effective communication and integration mechanisms. RabbitMQ excels in this domain by providing service decoupling, enabling microservices to interact without tightly integrated dependencies. This enhances maintainability and scalability within the microservices ecosystem.

Moreover, RabbitMQ's support for event-driven architecture aligns seamlessly with the requirements of microservices. Services can publish events and subscribe to others' events, facilitating real-time updates and responses to changes in the system. This event-driven approach is pivotal in achieving the desired flexibility and agility of microservices.

In the context of varying workloads that microservices can experience, RabbitMQ's load balancing capabilities prove invaluable. The message broker distributes tasks evenly, preventing service bottlenecks and bolstering system stability. Furthermore, RabbitMQ's fault tolerance features, including message persistence and replication, ensure that critical messages are not lost in the event of service failures. This attribute enhances the overall reliability and redundancy of microservices.

One of RabbitMQ's distinct advantages within microservices architecture is its ability to accommodate both synchronous and asynchronous communication needs. This adaptability gives the opportunity of optimising performance and responsiveness as per the specific requirements of each microservice.

In AIDAVA, the communication between its diverse microservices is orchestrated seamlessly through the utilisation of RabbitMQ as an event bus. Acting as an intermediary, RabbitMQ elegantly handles the transmission of messages from the service producers, who emit events, to the service consumers, who are vested in particular events.

In this ecosystem, events serve as the conduits of information, embodying the essence of noteworthy events in the system. These events are published by microservices to designated queues within RabbitMQ. Fellow microservices, keenly interested in specific events, subscribe to these queues to gain insight into the system's developments.

The elegance of this approach lies in its decoupling prowess. Microservices operate independently of one another, blissfully ignorant of their peers' existence. This cultivates an environment of scalability, agility, and easy maintenance, as each microservice functions autonomously without direct dependencies.

5 Data architecture

Data architecture is the design of data for use in defining the target state and the subsequent planning needed to achieve the target state. It is the planning of data and how it will be stored, consumed, integrated and managed by different data entities and IT systems, as well as any applications using or processing that data in some way. The different components have been introduced in Deliverable 2.3. Solution Design; this section focuses on the development aspect.

5.1 SQL database

A SQL (Structured Query Language) database is a structured and organised repository for storing and managing digital information. It employs a relational model, where data is organised into tables with rows and columns, each representing specific entities and their attributes. SQL databases enable efficient data storage, retrieval, and manipulation through a standardised language, SQL, which facilitates various operations like querying, inserting, updating, and deleting data. This powerful system ensures data integrity, consistency, and reliability by enforcing constraints and relationships between tables. SQL databases are widely used in applications ranging from business systems to web applications, offering a dependable and scalable solution for managing structured data.

5.1.1 UserDirectory

A user directory, also known as a directory service or identity management system, is a centralised database used to store and manage user-related information within a networked environment. This database contains user accounts along with their associated attributes, roles, and permissions. User directories play a critical role in security of AIDAVA by facilitating authentication and authorization processes, ensuring that users can securely access resources based on their designated privileges. They are commonly used in enterprises, educational institutions, and other organisations to maintain a structured and secure user management system.

In the case of AIDAVA the user directory will be storing authentication information for accessing the system, authorization information regarding roles and access levels and also profile information which may help describe the patient.

5.1.2 MasterData

Master data refers to the core and essential data elements that are critical for a business's operations and represent the foundational information about entities, objects, or subjects within an organisation. This data remains relatively stable over time and is typically shared across different departments and systems. Master data includes key reference information that serves as a common point of reference for various business processes and transactions.

In the context of AIDAVA, master data will keep information about things that will not need to change regularly. For example this information could be information about hospitals such as their unique identifiers in the system, name, location etc.

5.1.3 Library of Curation Tools

The Library of Curation Tools in the AIDAVA system is a comprehensive resource that streamlines data curation. It features a curated list of tools, complete with version and repository information. These tools are stored in a dual-container registry – one for basic curation and another with an added communication interface. This interface equips tools with predefined commands that seamlessly integrate with the AIDAVA system, enhancing data transformation, validation, and enrichment processes. This library optimises curation by ensuring compatibility, accessibility, and efficient interaction between tools and the system.

5.2 NoSQL

NoSQL databases, or "not only SQL" databases, are a category of database systems designed to handle and manage large volumes of unstructured or semi-structured data. Unlike traditional relational databases that rely on fixed schemas and SQL for querying, NoSQL databases offer more flexible data models tailored to specific use cases, including document-oriented, key-value, column-family, and graph databases. They prioritise high scalability, allowing data to be distributed across multiple servers for efficient horizontal scaling. NoSQL databases are well-suited for tasks demanding rapid growth, real-time processing, and diverse data formats, such as big data analytics, IoT, and content management systems.

5.2.1 Datalakes RAW, Staging, Curated, Published

"Datalakes RAW, Staging, Curated, Published" refers to a data processing and storage framework commonly used in data management pipelines. This framework organises data at various stages of processing, from its raw form to curated and published versions.

In the RAW stage, data is ingested into the data lake without significant processing or transformation. This is the initial state of the data, often directly sourced from various systems, applications, or external sources. The data might be in its original format, which could be structured, semi-structured, or unstructured.

In the Staging stage, the raw data is cleansed, standardised, and transformed. This might involve data validation, filtering, and structuring to prepare it for further processing. Staging helps ensure data quality and consistency before it moves on to more advanced processing stages. The data output from this stage is consistent with RDF (Resource Description Framework). RDF provides a variety of syntax notations and data serialisation formats including JSON-LD (JSON for Linked Data).

After the data has been staged, it enters the Curated stage. Here, data is refined and organised in a way that is suitable for analysis, reporting, and other business operations. In AIDAVA's use case, during this stage data is ready to be converted to Knowledge Graph.

In the Published stage, the curated data is made available for consumption by end-users, applications, or analytical tools. The format and structure of this data is dictated by each use case, specifically the requirements of the platform or destination in which it will be published

5.2.2 EventStore

Event sourcing is a software architectural pattern that revolves around capturing and storing the state changes of an application as a sequence of immutable events. Instead of solely recording the current state of an application's data, event sourcing focuses on preserving the history of how that state was arrived at through a series of events.

In event sourcing, every action or state change within the application is represented as an event object. These events are then stored in a chronological order within an event store or log. The current state of the application can be reconstructed by replaying these events sequentially.

5.3 GraphDB

Graph DB is a robust and high-performance semantic graph database tool designed to efficiently store, manage, and query complex interconnected data. It enables users to model and represent their data as a graph, where entities are nodes and relationships between entities are edges, allowing for flexible and intuitive data representations. For many downstream tasks, this representation of data provides strong usability due to rich semantic and reasoning power. GraphDB excels in handling large-scale semantic data by providing powerful graph-based querying, reasoning and inference capabilities, making it a valuable asset for applications ranging from knowledge management and linked data representation to data integration and exploration.

6 Curation tools

The heterogeneous nature of health data sources, formats and structures, coupled with the potential for such data to be incomplete, missing key information or missing context, presents a clear challenge for the AIDAVA system when integrating patient data into a PHKG.

In the context of AIDAVA, specifically for G1, curation tools refer to a pre-defined collection of various open source software and applications, integrated into each site’s AIDAVA deployment. Given the microservices architecture approach, each tool will run as its own service, with a predefined interface specifying the data interchange and protocols.

The specific curation tools have been tested and identified to support the curator (e.g. a patient) to find, verify and transform their data in a coherent and meaningful manner with as little human intervention required as possible.

The functionalities and capabilities of each curation tool act as solutions to specific interoperability issues as documented in AIDAVA deliverable 2.1 “Details on data curation & publishing process”. A given tool will only be utilised when a patient’s data meets certain pre-defined criteria, and acts as a bridge to transformation into the PHKG. Given the wide range of potential issues in the structure, content and context of the data AIDAVA will receive, it is possible that during the process of curating any one data source for one patient, multiple curation tools are ‘triggered’.

6.1 Onboarding workflow

As detailed in deliverable 2.1; the "Data source onboarding" process involves gathering information on a specific data source prior to the curation process. The more contextual information we can capture during this process, the higher the potential for automation, reducing the burden for both the patient and the curator.

The onboarding workflow itself defines the set of inputs required and individual steps the site administrator will follow to collect the necessary information to support curation of data from a specific source in future. During the onboarding process, the AIDAVA Site Administrator, with the support of the Data Expert curator, will check the validity of the underlying schema, identify tools that can support the curation workflow, and define the mapping with the AIDAVA Reference Ontology.

The output from the onboarding workflow for a given source are entries in the data catalogue and other metadata at the core of the AIDAVA system.

6.1.1 Onboarding tools

There are a number of software and tools available, both open-source and licenced which can provide one or more of the functionalities required for the onboarding process. These have been identified during the literature review in Task 2.1, and introduced in Deliverable 2.2

In this section, we provide a brief overview of some of these software and tools. Each will be investigated and tested for suitability in the G1 implementation of AIDAVA.

Tesseract:
Tesseract is an open-source OCR (Optical Character Recognition) engine that converts images of printed text into machine-readable text. In the context of AIDAVA, its application would be to extract text from images or scanned documents.

PDF Validator:
PDF Validator is a service or tool designed to verify the integrity and compliance of PDF documents with the PDF specification. It may perform checks to ensure that the PDF files adhere to the standards and guidelines for valid PDF documents.

Talend Open Studio:
Talend Open Studio is an open-source data integration software. It acts as an ETL (Extract, Transform, Load) tool which in the context of AIDAVA would be used to build data pipelines including transformations (e.g. mappings) and preprocessing steps (e.g. unit and measurement splitting) and support standardisation of DTS (Data Transfer Specifications) of heterogeneous structure and format into required output standards.

Onto Refine:
Onto Refine, also known as OpenRefine, is an open-source tool for cleaning and transforming data. It assists in cleaning, restructuring, and enriching datasets, enhancing data quality and usability.

6.2 Curation workflow

The chosen tool for our curation workflow is ELSA version 3. To enhance its efficiency, we have divided the workflow logic into two distinct sections for better management and execution.

The first section is the Workflow Server. Within this section, we've developed a web application that provides a user-friendly interface for editing various workflows. Users can conveniently manipulate the workflow components through the graphical user interface. Additionally, the Workflow Server enables the importing and exporting of workflows using JSON files, promoting seamless sharing and collaboration.

In this section, each step within a workflow is associated with a designated activity task. This ensures that each element of the workflow is executed according to its intended function. When a workflow is initiated, it triggers the creation of a workflow instance. This instance serves as a guide for the execution process, orchestrating the steps in the workflow.

The Workflow Executor Service forms the second section of our curation workflow architecture. This service assumes the responsibility of managing the execution of individual workflow steps through webhooks. The advantage of this containerized solution is the ability to concurrently run multiple instances of the Workflow Executor Service, thereby enhancing the overall speed and efficiency of the curation process.

One of the key functions of the Workflow Executor Service is to interact with the containerized curation tools. This interaction is established through well-defined connections, allowing the curation tools to be systematically called and utilised. This systematic approach ensures that the curation tools are optimally employed in alignment with the workflow steps.

In summary, the curation workflow, powered by ELSA version 3, is designed with a clear division between the Workflow Server and Workflow Executor Service. This separation enhances the organisation, execution, and parallelization of the curation process, ultimately leading to a more streamlined and efficient workflow for our curation needs.

6.2.1 Curation tools

There are a number of software and tools available, both open-source and licenced which can provide one or more of the functionalities required for the curation process. These have been identified during the literature review in Task 2.1, and introduced in Deliverable 2.2

In this section we provide a brief overview of some of these software and tools. Each will be investigated and tested for suitability in the G1 implementation of AIDAVA.

Graph DB:
Graph DB refers to a graph database, a type of database designed to store and manage data as interconnected nodes and edges, allowing for efficient graphical representation and querying of complex relationships between data points.

Tabula:
Tabula is an open-source tool designed to extract tabular data from PDF documents. It extracts data from tables in a PDF document and enables exporting into a structured data format (e.g. comma separated values (.csv) or Microsoft Excel (.xslx)).

PDFMiner:
PDFMiner is a text extraction tool for PDF documents. It enables programmatically extracting textual information from PDF files for further processing or analysis.

Translator (NodeNormalization):
Translator with NodeNormalization is a component that assists in standardising or normalising data in a consistent format. This tool can support interoperability.

UCUM Web Services:
UCUM Web Services likely refers to a set of web services that provide functionality related to the Unified Code for Units of Measure (UCUM). UCUM is a standardised code system for representing units of measurement in healthcare and other domains.

UCUM-LHC Converter:
There are a number of open-source implementations including the UCUM-LHC Converter which facilitate conversion and standardisation of measurement units, for example in pathology reports.

7 Integration

AIDAVA curation tool in the ingestion part primarily focuses on streamlining the data management process. In this part, it aims to establish a seamless method for gathering various types of information. An integral aspect of this endeavour involves the integration of data from medical partners and Health Data Intermediary (HDI). This collaborative effort ensures that vital medical insights and information are efficiently incorporated into the system's data store.

One of the functionalities of AIDAVA is the capability to both receive and transmit files and data. This two-way communication is crucial for maintaining a dynamic flow of information between different stakeholders. Whether it's obtaining crucial medical records or dispatching essential findings, this bidirectional data exchange plays a pivotal role in the project's success.

Once the data is collected, AIDAVA orchestrates a systematic process of storing this information in the RAW Data Store. This repository serves as the foundation, housing the diverse array of data that the project handles. The Data Transfer Specification (DTS) is then employed to organise and structure this data, making it more accessible and actionable for further analysis and interpretation.

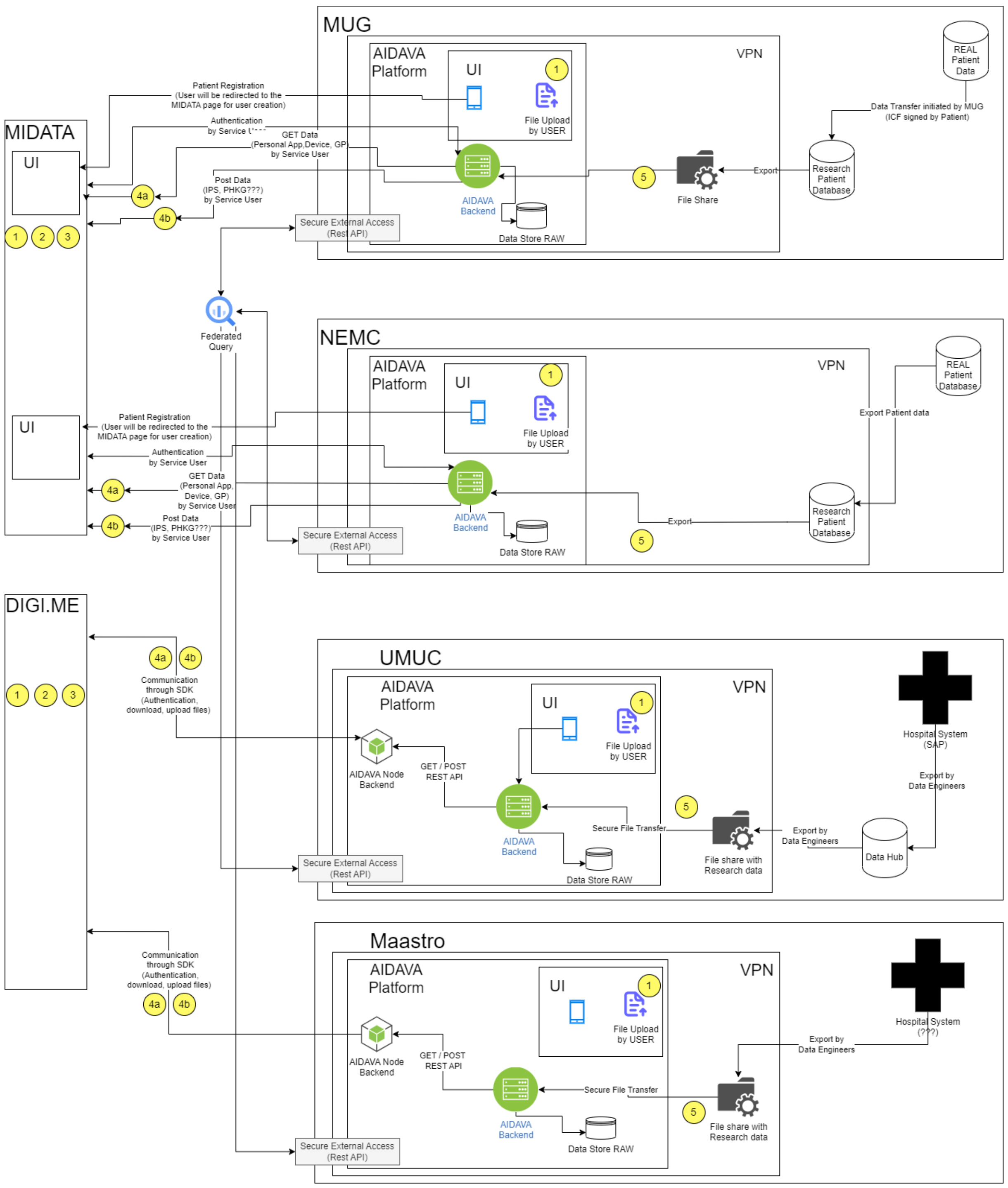

MUG
AIDAVA Platform
UI
1
File Upload by USER
VPN
REAL Patient Data
Data Transfer initiated by MUG (ICF signed by Patient)
Research Patient Database
Export
File Share
5
AIDAVA Backend
Data Store RAW
Secure External Access (Rest API)
Patient Registration (User will be redirected to the MIDATA page for user creation)
Authentication by Service User
GET Data (Personal App,Device, GP) by Service User
Post Data (IPS, PHKG???) by Service User
4a
4b
MIDATA
UI
1
2
3
Federated Query
NEMC
AIDAVA Platform
UI
File Upload by USER
VPN
REAL Patient Database
Export Patient data
Research Patient Database
Export
AIDAVA Backend
Data Store RAW
Secure External Access (Rest API)
Patient Registration (User will be redirected to the MIDATA page for user creation)
Authentication by Service User
GET Data (Personal App, Device, GP) by Service User
Post Data (IPS, PHKG???) by Service User
DIGI.ME
Communication through SDK (Authentication, download, upload files)
UMUC
AIDAVA Platform
UI
File Upload by USER
VPN
AIDAVA Node Backend
GET / POST REST API
AIDAVA Backend
Secure File Transfer
Data Store RAW
File share with Research data
Export by Data Engineers
Hospital System (SAP)
Export by Data Engineers
Data Hub
Secure External Access (Rest API)
Maastro
AIDAVA Platform
UI
File Upload by USER
VPN
AIDAVA Node Backend
GET / POST REST API
AIDAVA Backend
Secure File Transfer
Data Store RAW
File share with Research data
Hospital System (???)
Export by Data Engineers
Secure External Access (Rest API)
Communication through SDK (Authentication, download, upload files)

Figure 10. AIDAVA - Medical partners - Connections - Technical

7.1 Integration with Medical partners

7.1.1 MUG, UMUC, Maastro

At Med Uni Graz (MUG), Maastricht University Medical Centre (MUMC) and Maastro clinic the data management process involves several interconnected steps. The data is exported and channelled into designated file shares. These file shares are located within a secure Virtual Private Network (VPN) environment, ensuring the confidentiality and integrity of the information.

Upon reaching the file shares, the data undergoes the phase of ingestion. This pivotal step is orchestrated by the AIDAVA backend, which acts as the catalyst for transferring the files into the RAW Data Store. The RAW Data Store serves as a repository, housing the unprocessed and unaltered data in its pristine form, ready for subsequent analysis and curation.

7.1.2 NEMC

The North Estonia Medical Centre (NEMC) hosts a database on a local server and provides access via secure VPN, channelling data for the project. Extracted into files, this process is managed by the AIDAVA backend, which transfers them to the RAW Data Store. This streamlined journey ensures data security and optimised medical insights for operational excellence.

7.2 Integration with HDI

7.2.1 MIDATA

The integration of AIDAVA with MIDATA is a multi-step process designed to enhance healthcare services. AIDAVA uses the MIDATA API to connect with the platform. Users register on both AIDAVA and MIDATA. AIDAVA stores the unique MIDATA user ID after registration. A pre-registered service account allows AIDAVA to securely download patient data. AIDAVA then analyses the data, generating insights. It compiles an International Patient Summary (IPS). This IPS is uploaded to the user's MIDATA profile through the MIDATA API, enriching their healthcare record.

7.2.2 Digi.Me

The integration between AIDAVA and DIGIME involves the establishment of a dedicated microservice within AIDAVA for DIGIME's functionalities. This microservice utilises DIGIME's SDK solution to seamlessly connect the two systems. Users are required to register in the DIGIME app, after which AIDAVA stores their registered IDs. AIDAVA also possesses a pre-registered service account within DIGIME. This service account facilitates secure data downloads for patients, enabling accurate information transfer. At the culmination of data curation, the International Patient Summary (IPS) is generated and seamlessly uploaded back to DIGIME using its SDK.

8 Hardware Specifications

In planning the deployment of the Prototype, it's essential to start with a clear set of assumptions and infrastructure requirements. These assumptions and infrastructure needs will serve as the foundation for a seamless transition from G1 to G2 without significant changes. Let's break down these key elements.

8.1 Assumptions

The Prototype's architecture will remain consistent and unchanged from Generation 1 (G1) to Generation 2 (G2). This stability is crucial for ensuring a smooth transition and minimal disruptions during the upgrade.

8.2 Infrastructure Requirements

8.2.1 Docker Containerization

All essential components of the prototype, including services, databases, workflows, and the knowledge graph, will be packaged and deployed within Docker containers. This containerization will be efficiently managed using our designated docker-compose file. It is imperative that Docker is installed on the hosting servers to facilitate this deployment strategy.

8.2.2 Access Protocols

Secure Shell (SSH) and Remote Desktop Protocol (RDP) access will be needed for administration and maintenance tasks on the hosting servers. These access methods are essential for ensuring the Prototype's operational integrity.

8.2.3 Https Access

To enable external access to the User Interfaces or API endpoint from locations outside the hospital environment, HTTPS access is required. This secure access mechanism will be crucial for running federated queries securely and efficiently.
The Prototype will incorporate the necessary hospital certificate(s) into its configuration. This integration is vital for establishing trust and ensuring that the Prototype operates within the hospital's security and compliance framework.

8.2.4 Mail Server Access

Access to a dedicated mail server will be required to facilitate the sending of notifications to users. This functionality is essential for keeping users informed and engaged with the Prototype.

8.2.5 Hardware Specification

Hardware specification are only estimations

Resources	Minimum	Recommended	Comments
CPUs / vCPUs	4 core	8 core	Depending on the degree of parallel processing
RAM	24 GB	32 GB	Depending on the degree of parallel processing
Storage	1 TB	2 TB	Storage requirements depend on how many documents are imported.

9 Deployment & Testing

The "Deployment & Testing" phase in the AIDAVA project is a crucial step towards ensuring the reliability and quality of our codebase. This phase is powered by a robust CI/CD deployment process, enabling efficient and automated deployment of our software changes.

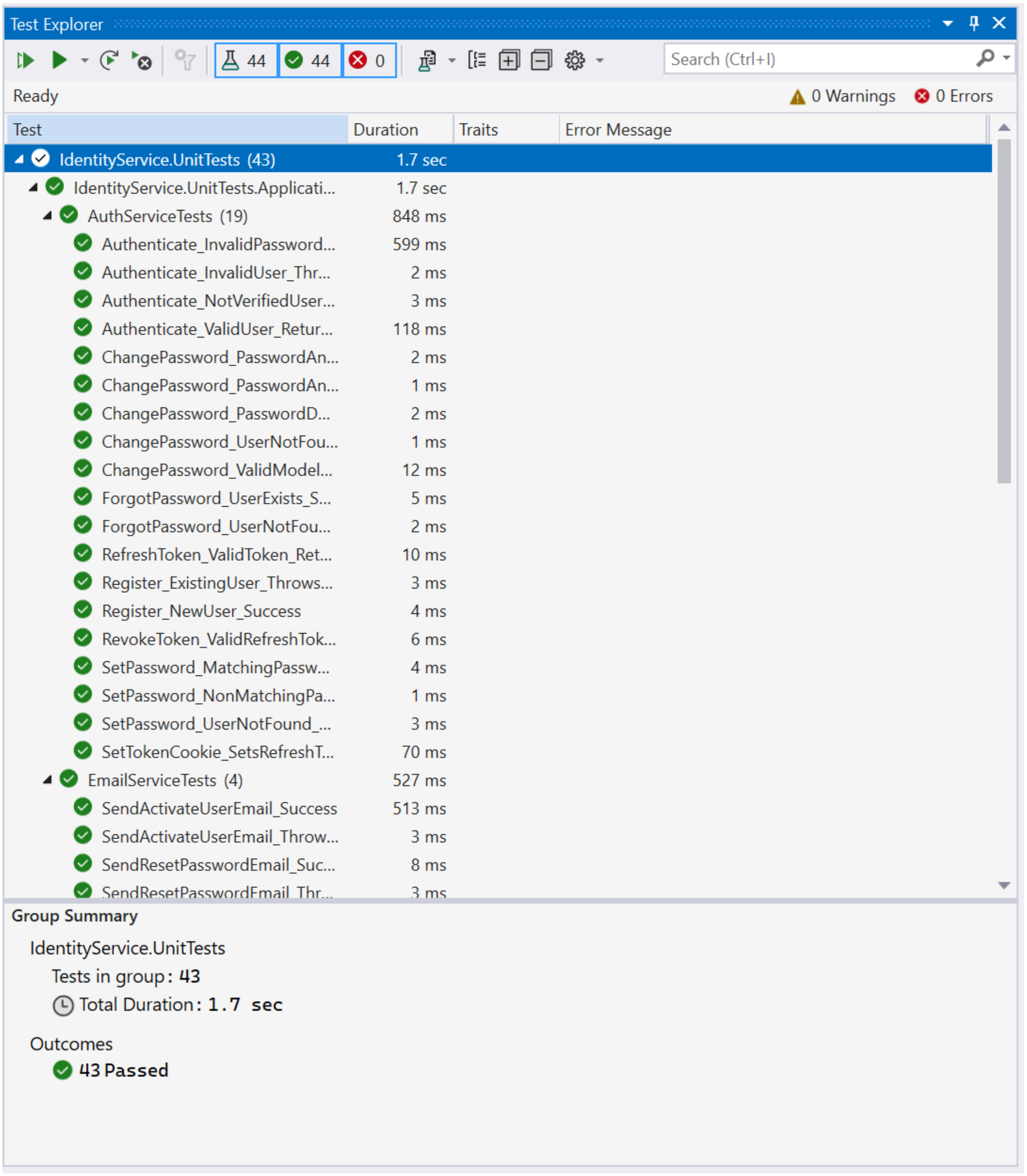
Our CI/CD deployment process automates the build and testing stages, streamlining the validation of code changes. During the CI phase, builds are generated, and tests, including unit and integration tests for the backend, and unit, integration, and end-to-end (e2e) tests for the frontend, are executed. This step guarantees that code modifications adhere to quality standards and prevent regressions.

The CD phase allows us to deploy our modifications with a single click, delivering changes to production or test environments seamlessly. Notably, modifications to volumes are excluded from deployment to safeguard data integrity.

Our comprehensive testing strategies ensure the integrity of both frontend and backend components. The backend undergoes rigorous unit and integration testing, covering individual units and component interactions. On the frontend, we conduct unit, integration, and end-to-end tests, guaranteeing robustness across all layers of our application.

Maintaining a high level of test coverage is a paramount goal in the AIDAVA project. We require test coverage to be consistently above 80% for both frontend and backend components. This stringent criterion ensures that our tests comprehensively exercise code paths, minimising the risk of undetected issues and enhancing the overall stability of our microservices architecture.

9.1 Backend - Unit, Integration tests


Test Explorer

44 44 0

Search (Ctrl+I)

Ready 0 Warnings 0 Errors

Test	Duration	Traits	Error Message
IdentityService.UnitTests (43)	1.7 sec		
IdentityService.UnitTests.Applicati...	1.7 sec		
AuthServiceTests (19)	848 ms		
Authenticate_InvalidPassword...	599 ms		
Authenticate_InvalidUser_Thr...	2 ms		
Authenticate_NotVerifiedUser...	3 ms		
Authenticate_ValidUser_Retur...	118 ms		
ChangePassword_PasswordAn...	2 ms		
ChangePassword_PasswordAn...	1 ms		
ChangePassword_PasswordD...	2 ms		
ChangePassword_UserNotFou...	1 ms		
ChangePassword_ValidModel...	12 ms		
ForgotPassword_UserExists_S...	5 ms		
ForgotPassword_UserNotFou...	2 ms		
RefreshToken_ValidToken_Ret...	10 ms		
Register_ExistingUser_Throws...	3 ms		
Register_NewUser_Success	4 ms		
RevokeToken_ValidRefreshTok...	6 ms		
SetPassword_MatchingPassw...	4 ms		
SetPassword_NonMatchingPa...	1 ms		
SetPassword_UserNotFound_...	3 ms		
SetTokenCookie_SetsRefreshT...	70 ms		
EmailServiceTests (4)	527 ms		
SendActivateUserEmail_Success	513 ms		
SendActivateUserEmail_Throw...	3 ms		
SendResetPasswordEmail_Suc...	8 ms		
SendResetPasswordEmail_Thr...	3 ms		

Group Summary

IdentityService.UnitTests

Tests in group: 43

Total Duration: 1.7 sec

Outcomes

43 Passed

Figure 11. AIDAVA - Backend - Unit, Integration tests

The "Backend - Unit, Integration Tests" segment is a pivotal aspect of the AIDAVA project, aimed at ensuring the robustness, reliability, and quality of the backend services developed using .NET 6 Core. This testing framework encompasses both unit tests and integration tests, collectively serving as essential safeguards against defects and regressions.

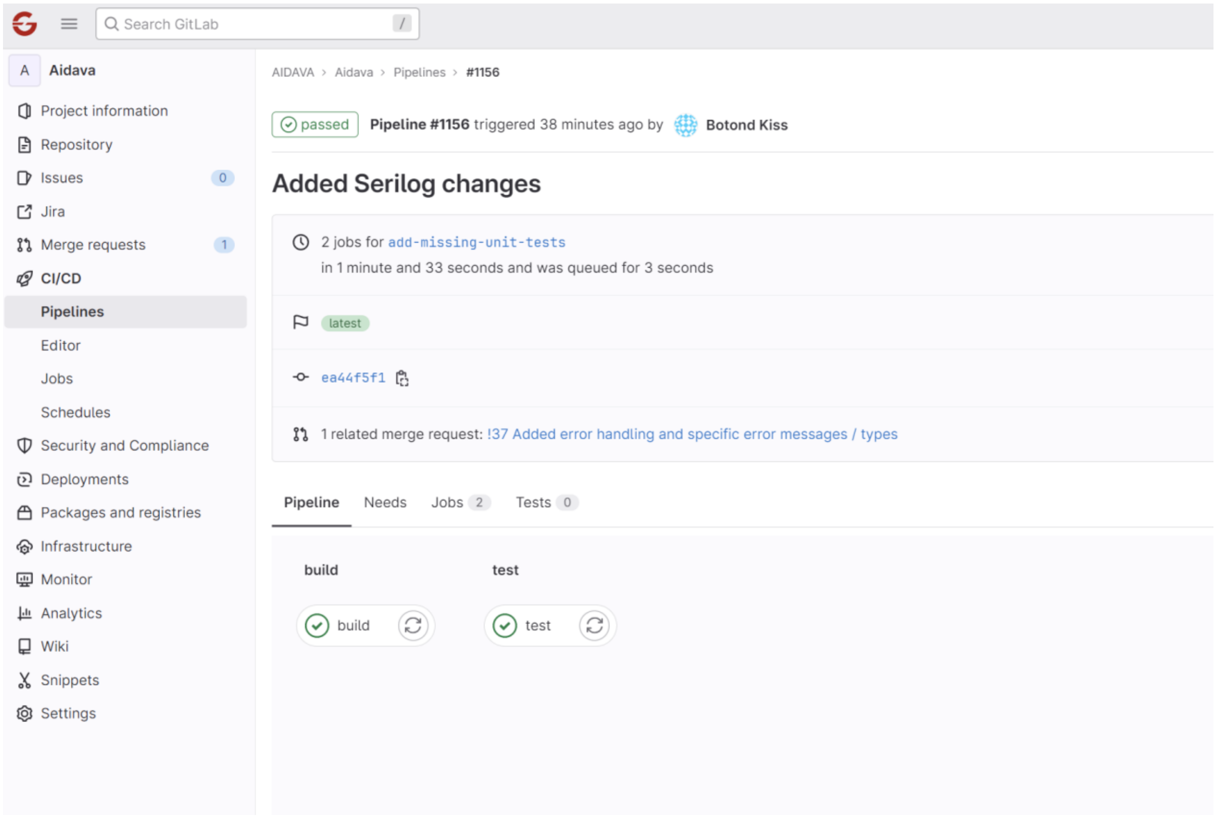
Unit tests are meticulously crafted to scrutinise individual units or components of the backend services in isolation. These tests evaluate the behaviour and functionality of specific methods, classes, or modules, verifying that they produce expected outcomes under various conditions. Unit tests are designed to be concise, focused, and self-contained, enabling developers to identify and rectify issues with precision.

Integration tests, on the other hand, assess the interoperability and collaboration between various backend components. These tests verify that different units, services, or modules work cohesively when integrated, simulating real-world scenarios and interactions. Integration tests identify potential bottlenecks, inconsistencies, or unexpected behaviours arising from the integration of multiple components.

A key aspect of integration tests in the AIDAVA project is the replication of actual runtime conditions, ensuring that the interactions between backend services are validated under realistic circumstances.

Both unit tests and integration tests contribute to the continuous improvement of the AIDAVA backend, fortifying the overall codebase and enhancing system stability. By embracing a comprehensive testing approach, the AIDAVA project reinforces its commitment to delivering high-quality backend services, ultimately translating into a more reliable and effective microservices architecture.

9.1.1 Testing tools - CI/CD - Focus on CI



Pipeline #1156 triggered 38 minutes ago by Botond Kiss

Added Serilog changes

2 jobs for add-missing-unit-tests
in 1 minute and 33 seconds and was queued for 3 seconds

ea44f5f1

1 related merge request: !37 Added error handling and specific error messages / types

Figure 12. AIDAVA - Backend - Continuous Integration (CI)

In the AIDAVA project, our Continuous Integration (CI) process stands as a crucial pillar in upholding code quality and ensuring the reliability of our microservices architecture.

The CI process comprises two core steps: build and tests. During the build phase, our CI pipeline adapts to the specific needs of each component. For the backend, it compiles and packages code changes into deployable artifacts. Meanwhile, the frontend code undergoes TypeScript compilation and resource bundling to prepare for testing and deployment.

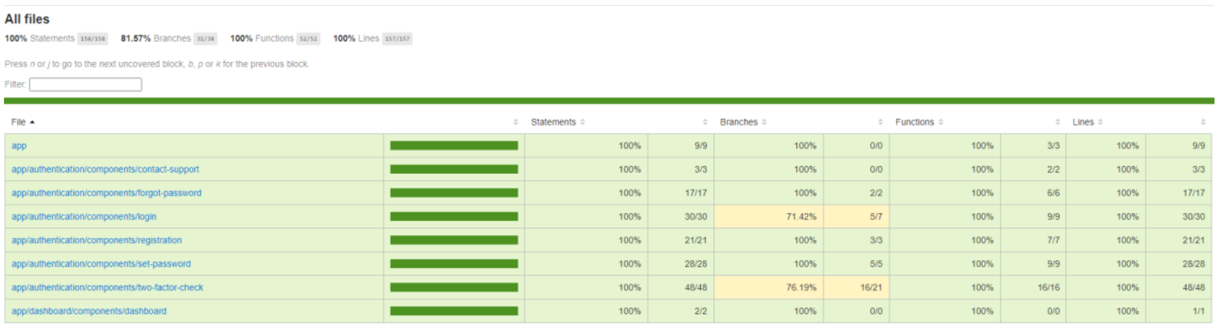
This standardised process guarantees that every code modification is properly prepared for rigorous testing and potential deployment.

The tests step is where code is subjected to a comprehensive battery of tests, including unit, integration, and end-to-end tests. These tests meticulously evaluate the functionality and interactions of our code, preventing regressions and verifying seamless integration with existing components.

A standout aspect of our CI process is the SonarQube code quality check. Triggered upon merging a branch into the main branch, SonarQube conducts comprehensive static code analysis. It detects code smells, vulnerabilities, and adherence to coding standards, enhancing the overall robustness, security, and maintainability of our codebase.

By effectively combining the build, tests, and SonarQube steps, our CI process serves as a gatekeeper for the quality of code changes. It ensures that only reliable, well-tested, and secure code reaches our main codebase, contributing to the overall stability and excellence of our codebase.

9.2 Frontend -Unit tests, E2E tests


All files

100% Statements 158/158 **81.57%** Branches 31/38 **100%** Functions 52/52 **100%** Lines 157/157

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

Filter:

File ▲		Statements		Branches		Functions		Lines	
app	🟩	100%	9/9	100%	0/0	100%	3/3	100%	9/9
app/authentication/components/contact-support	🟩	100%	3/3	100%	0/0	100%	2/2	100%	3/3
app/authentication/components/forgot-password	🟩	100%	17/17	100%	2/2	100%	6/6	100%	17/17
app/authentication/components/login	🟩	100%	30/30	71.42%	5/7	100%	9/9	100%	30/30
app/authentication/components/registration	🟩	100%	21/21	100%	3/3	100%	7/7	100%	21/21
app/authentication/components/set-password	🟩	100%	28/28	100%	5/5	100%	9/9	100%	28/28
app/authentication/components/two-factor-check	🟩	100%	48/48	76.19%	16/21	100%	16/16	100%	48/48
app/dashboard/components/dashboard	🟩	100%	2/2	100%	0/0	100%	0/0	100%	1/1

Figure 13. AIDAVA – Frontend - Unit, E2E tests

After thorough analysis, Angular was selected as the frontend framework for developing the UI applications for the AIDAVA prototype. Some decisive aspects were the modular architecture, propagation of modern software development practices and out of the box automated testing support. Angular comes with built-in Jasmine and Karma integration for Unit testing, and low effort Cypress integration for end-to-end (E2E) testing.

Angular's built-in unit testing libraries (Karma, Jasmine) support the stability, functionality, and maintainability of applications built with the framework. By writing and executing automated unit tests, developers can examine individual components, services, and directives in isolation. This practice aids in detecting bugs early in the development process, thus allowing developers to confidently make changes to code without fearing unintended side effects, as the tests act as a safety net that quickly highlights regressions. Moreover, these tests serve as documentation, describing how various parts of the application are intended to function.

On the other hand, end-to-end (E2E) testing methodology involves simulating real user interactions and scenarios to validate the entire application workflow. By utilizing tools like the Cypress testing framework, developers can automate browser actions such as mouse clicks, form submissions, and navigation, mimicking how actual users would interact with the application. This process helps uncover issues related to user interface responsiveness, data flow, and integration between different components. E2E testing aids detecting bugs that might not be evident through unit testing and ensures that the application meets the user's expectations in terms of navigation, usability, and overall performance.

9.2.1 Testing tools - SorryCypress

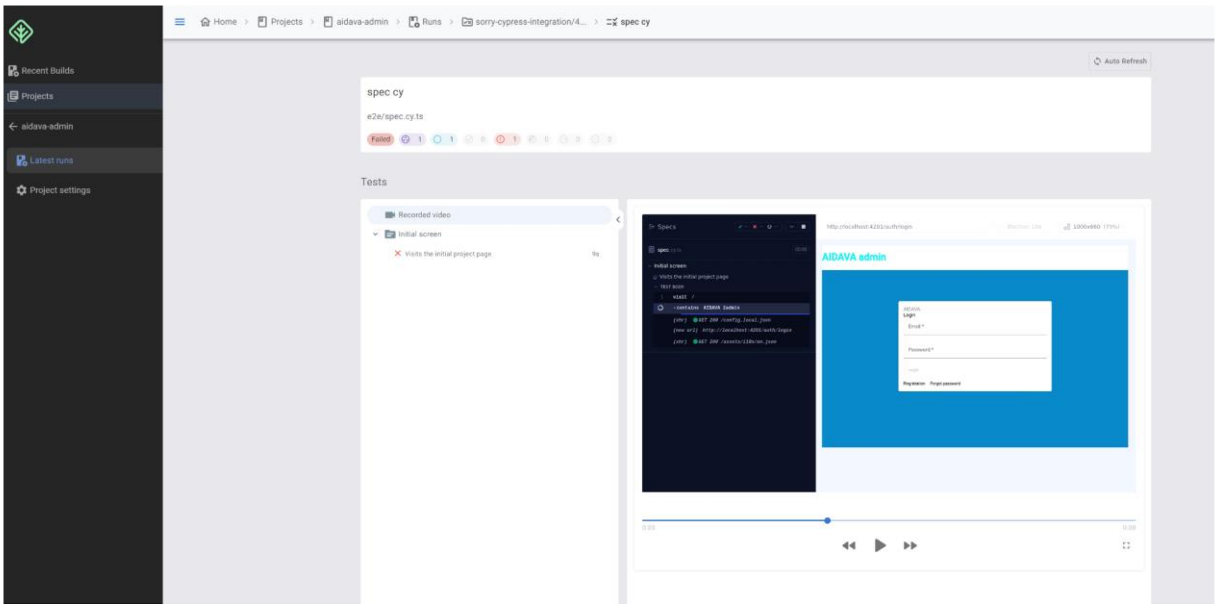


Figure 14. AIDAVA – SorryCypress

Although the Cypress testing framework can run end-to-end tests in a standalone manner, optimizing E2E test execution times in CI/CD pipelines without Cypress's Cloud service cannot be effectively achieved.
Cypress Cloud offers great parallelization, test debugging and analytics capabilities, however, it lacks a self-hosted version, raising data privacy concerns.

SorryCypress is an open-source, highly regarded alternative to Cypress Cloud, offering advanced capabilities for efficient and scalable end-to-end testing of web applications. It enables developers to execute Cypress tests across multiple machines in a distributed manner, thereby significantly reducing test execution times. Unlike Cypress Cloud, SorryCypress can be self-hosted, granting users greater control over their testing infrastructure and data privacy. This powerful tool also provides detailed test logs, and the ability to analyze test runs comprehensively. SorryCypress offers insightful dashboards for analysing E2E test runs by screenshots and video recordings, automatically captured by the Cypress framework during every test execution.

9.3 Container deployment

The "Container Deployment" phase within the AIDAVA project represents a pivotal step in our seamless and efficient software delivery process. This phase, facilitated by GitLab's robust Continuous Deployment (CD) capabilities, revolves around generating Docker images and storing them in GitLab's dedicated container registry.

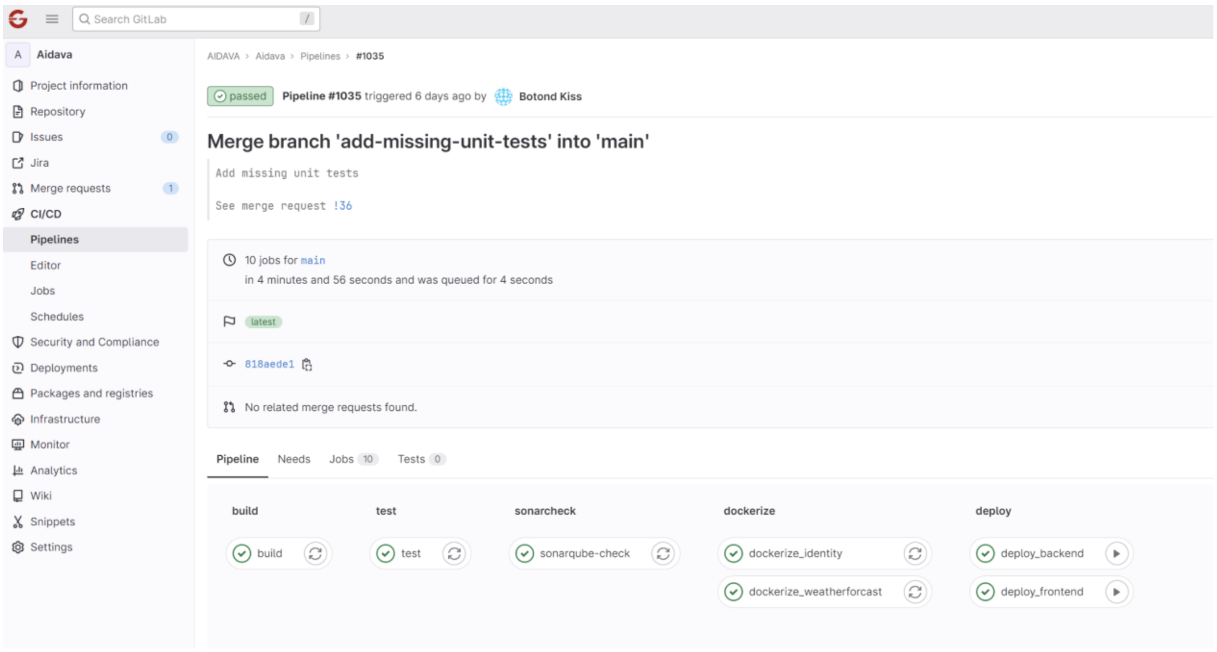
During this phase, we encapsulate the functionality of each microservice into Docker images, creating a self-contained and consistent environment for deployment. Each microservice results in two distinct Docker images: one associated with the specific commit hash and another with the 'latest' tag. This approach empowers us to maintain flexibility in deployment by having a snapshot of each microservice at its corresponding commit, while also having the latest version readily accessible.

GitLab's container registry acts as a centralised repository for these Docker images, enabling versioned storage and controlled access. This repository hosts a collection of immutable images, each representing a specific iteration of our microservices architecture.

By leveraging GitLab's CD capabilities, we seamlessly bridge the gap between code readiness and operational deployment. The Container Deployment phase, with its Docker image generation and registry utilisation, streamlines the process of packaging, distributing, and deploying our microservices with precision and ease.

This strategic approach ensures that our microservices are encapsulated, versioned, and accessible, promoting efficient management and enabling agile deployment strategies. The Container Deployment phase underscores our commitment to delivering a robust and agile microservices architecture within the AIDAVA project.

9.3.1 CI/CD - Focus on CD



passed Pipeline #1035 triggered 6 days ago by Botond Kiss

Merge branch 'add-missing-unit-tests' into 'main'

```
Add missing unit tests

See merge request !36
```

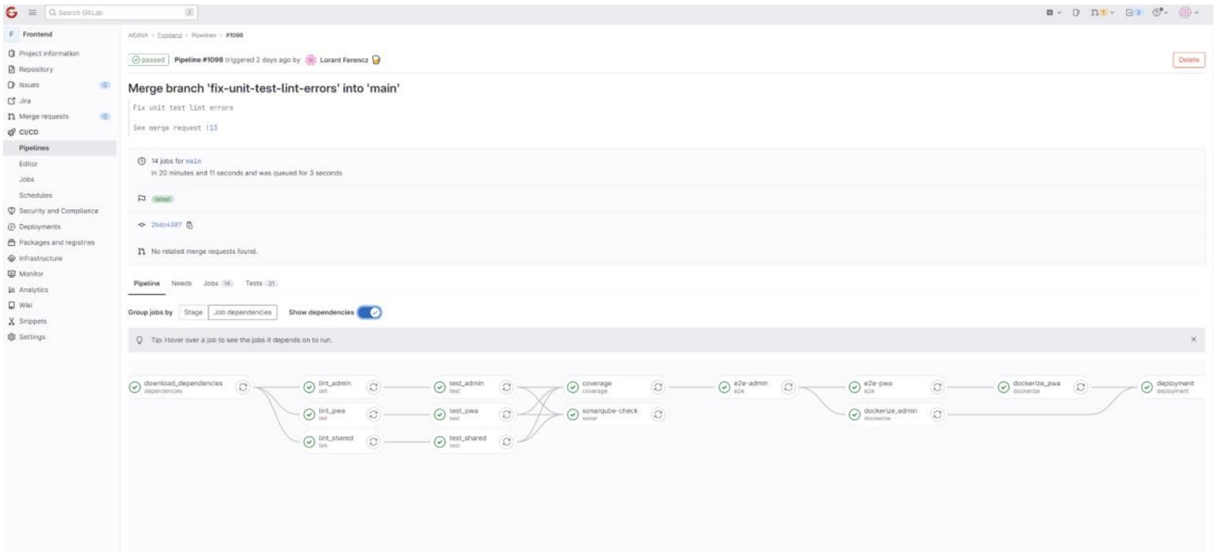
10 jobs for main
in 4 minutes and 56 seconds and was queued for 4 seconds

latest

818aede1

No related merge requests found.

Figure 15. AIDAVA – Backend deployment


passed Pipeline #1098 triggered 2 days ago by Lorant Ferencz

Merge branch 'fix-unit-test-lint-errors' into 'main'

```
Fix unit test lint errors

See merge request !13
```

14 jobs for main
in 20 minutes and 11 seconds and was queued for 3 seconds

2bdc4387

No related merge requests found.

Tip: Hover over a job to see the jobs it depends on to run.

Figure 16. AIDAVA – Frontend Deployment

The Continuous Deployment (CD) process within the AIDAVA project represents the culmination of our streamlined software delivery lifecycle. This phase, facilitated by GitLab's powerful capabilities, encompasses a sequence of strategic steps, each contributing to the efficient and controlled deployment of our microservices architecture.

At the heart of the CD process lies the generation of Docker images, capturing the essence of each microservice within a self-contained environment. These images are meticulously crafted to encapsulate both the frontend and backend components, ensuring consistent and reliable deployment.

Upon image generation, we utilise GitLab's container registry to securely store these Docker images. This centralised repository ensures versioned and controlled access to our microservice images, forming a crucial repository for operational deployment.
The CD process unfolds through a manual "Deploy" stage, where user interaction plays a key role. This stage is designed to provide flexibility and control, allowing users to manage deployments according to specific requirements. Within the "Deploy" stage, two elements stand out: the deployment of frontend images and the deployment of backend images. This dual approach permits separate deployment of frontend and backend components, catering to distinct needs and enhancing operational agility.

While user interaction is required, the "Deploy" stage empowers users to orchestrate the deployment process with precision and confidence. Whether deploying frontend, backend, or both, this manual step ensures that deployments are aligned with project objectives and business needs.

The CD process, with its Docker image generation, container registry utilisation, and manual "Deploy" stage, embodies our commitment to controlled, efficient, and strategic deployment. It seamlessly bridges development and operations, resulting in a robust and adaptable architecture within the AIDAVA project.

10 Conclusion

In conclusion, the proposed technical architecture represents a robust and scalable solution that is well-equipped to meet the demands of our project. Through the utilisation of microservices, integration of satellite applications, and the implementation of efficient curation tools, we have laid a strong foundation for automating our workflows. The architecture's seamless connection with diverse medical partners, whether through file shares, databases, or health data intermediaries, ensures our ability to collect and process data effectively.

As we move forward with the implementation of this technical architecture, we anticipate increased efficiency, flexibility, and reliability in our systems. It not only aligns with current industry best practices but also positions us to adapt and scale as our project evolves. By adhering to these architectural principles, we are well-prepared to achieve our goals and deliver exceptional results in our technology endeavours.

11 Next steps

The next steps are clear and achievable. The established architecture outlined above will be developed in *Deliverables 3.3, 3.4 and 3.5*.

Updates to the document will be made as necessary to reflect significant architectural changes. The primary goal remains centred on providing a robust solution for the AIDAVA project, ensuring it aligns with current standards.

Our focus is steadfast, with a commitment to delivering a dependable solution that meets project needs effectively.