DAIsy
Developing AI ecosystems improving diagnosis and care of mental diseases
DAIsy
24 partners
5 countries
ITEA Call 2021
Smart health

DAIsy – Developing AI ecosystems improving diagnosis and care of mental diseases

ITEA 4 – 21016

Work package 5 (WP5)

Platform Development

Deliverable 5.1. Development of common components and integration services of platform

Document type	: Deliverable
Document version	: No. 1
Document Preparation Date	: Feb 2024
Classification	: Confidential
Due Date	: Dec 2025

HISTORY

Document version #	Date	Remarks
1	April 2024	Initial version
2	February 2026	Final version

Table of Contents

1 Introduction

This DAIsy project deliverable is a testament to the modular and versatile design approach adopted by the development team. Each component and service crafted for this initiative not only fulfils its initial use case but also possesses the flexibility to be adapted for alternative applications. This adaptability is a significant advantage, ensuring that the project's outputs have a broader impact and a prolonged operational lifespan.

In essence, the DAIsy project's components and services are not just solutions to present challenges but are also stepping stones for future innovations. They embody a forward-thinking philosophy where each deliverable is a building block for the next, paving the way for a legacy of sustainable and scalable technology solutions.

The various security mechanisms of the individual components described are discussed in detail in Deliverable 5.4, where the protective measures and security architecture are comprehensively documented.

2 Therapy assistant – Backend Service

2.1 Functionality

The DAIsy Backend forms the central foundation for processing and providing all relevant data. It handles patient synchronisation by regularly transferring information from the hospital management system to the therapist dashboard without making any changes to the source system. This ensures data integrity and allows therapists to track what information was available to them for diagnosis at any given time.

In addition to synchronisation, the backend manages all data sent by the patient app, including mood information and the results of module processing. This information is reliably stored and immediately available for therapeutic work. In addition, a locally embedded AI service for progress prediction is integrated into the backend, which analyses the collected data and supports therapists with forecasts of possible developments in the course of therapy.

Through the close integration of data storage, synchronisation and AI-supported analysis, the backend ensures that therapists can use both current and historically traceable information. This not only increases the quality of patient care, but also ensures the transparency and traceability of therapeutic decisions.

The backend design also considers clinical usage constraints by ensuring that synchronized and processed data reflect the clinical context at specific points in time. This enables consistent interpretation of patient information during diagnosis and follow-up, without altering source systems or introducing ambiguity in therapeutic decision-making.

2.2 Interfaces

The DAIsy therapist dashboard backend retrieves patient data from the hospital management system via an HL7 FHIR interface. This data is stored in a connected MariaDB, which only the service has access to in order to ensure the security and integrity of the information. For retrieval via the therapist dashboard, the backend provides a REST interface that enables secure and efficient access to patient information.

In addition to this core functionality, the backend maps additional interfaces via REST. This includes integration with Keycloak, which ensures central authentication and authorisation and reliably secures access to the dashboard and the patient app. Data flows from the patient apps are also processed via REST: mood data and module processing results are stored directly in the backend and are available to therapists in real time. In addition, the AI service for predicting progress is also connected via a REST interface, so that the analysis and prognosis functions are seamlessly integrated into the backend architecture.

Through this combination of standardised interfaces, secure data storage and modular expandability, the backend ensures a robust, scalable and transparent infrastructure that meets both clinical requirements and technical integrity.

2.3 Dependencies

The service is based on ASP.NET Core and utilises its integrated resources such as dependency injection, middleware pipeline and configuration management, ensuring optimised and efficient operation. The use of the integrated Entity Framework Core enables direct access to the connected MariaDB without the need for external ORM libraries. For secure communication, the token-based authentication and HTTPS enforcement mechanisms included in ASP.NET Core are used, eliminating the need for additional security frameworks.

Since all interfaces – including REST endpoints for the therapist dashboard, patient apps, AI service and integration with Keycloak – are based on the native web API functionalities of ASP.NET Core, no external dependencies are necessary. This reduces the complexity of deployment and greatly simplifies maintenance, as updates and security patches can be applied centrally via the ASP.NET Core framework.

2.4 Architecture

The architecture of the DAIsy therapist dashboard backend is modular and connects clinical data sources, patient apps and AI-supported services in a scalable and secure infrastructure.

At its core is the backend, which is based on ASP.NET Core and operated in a Kubernetes cluster. This foundation enables containerised deployment, automatic scaling and self-healing in the event of failures. The backend ensures communication with the various components via REST interfaces: the therapist dashboard accesses patient information via a secure API, the patient apps send mood data and module processing results, Keycloak handles central authentication and authorisation, and the AI service for predicting progression is also integrated via REST.

The connection to the hospital management system is made via an HL7 FHIR interface, through which patient data is regularly retrieved and stored in a MariaDB. This database is accessible exclusively to the service and is scalable through replication and sharding, ensuring both data integrity and performance. During synchronisation, the existing data remains unchanged, ensuring the traceability of patient records at all times.

The architecture is deliberately designed to be stateless for REST communication, so that additional instances of the backend can be operated without complex synchronisation mechanisms. Monitoring and observability are integrated into the cluster environment and continuously provide metrics on response times, throughput, error rates and resource utilisation.

From a clinical perspective, the architecture supports traceability by preserving historical data states and ensuring that analytical outputs can be related to the corresponding clinical observations. This design choice enables safe integration of AI-supported services into routine mental healthcare workflows.

2.5 Performance Metrics

The DAIsy therapist dashboard backend is designed for high performance and continuous monitoring. The response times of the REST APIs are continuously measured to ensure low latency and smooth access for therapists and patients. At the same time, throughput is monitored in terms of processed requests per second, so that the system remains stable even with parallel access. CPU and memory resource utilisation within the Kubernetes pods provides important metrics that are used for automatic scaling decisions.

The database performance of MariaDB is evaluated based on query latencies, transaction rates and I/O operations, with replication and sharding forming the basis for stable processing of growing data volumes. The interfaces to the hospital management system via HL7-FHIR are also monitored for latency to ensure that synchronised patient data is up to date.

For the AI service for history prediction, model inference runtimes are crucial, as forecasts must be provided quickly even as data volumes increase. In addition, network metrics such as bandwidth, packet loss and latency between the backend, database and external interfaces are taken into account. Finally, the efficiency of the automatic scaling mechanisms in the Kubernetes cluster also plays a central role, as the time it takes to provision new pods during peak loads directly affects the user experience.

All these metrics are collected and visualised using monitoring and observability tools, so that deviations can be detected immediately and countermeasures initiated.

2.6 Scalability

The DAIsy therapist dashboard backend is already hosted in a Kubernetes cluster and is therefore designed from the ground up for scalability and reliability. The architecture utilises the containerisation of ASP.NET Core services, allowing new instances to be dynamically provisioned as the load increases. Since the REST interfaces to the dashboard, patient app, Keycloak and the AI service are implemented statelessly, additional pods can be operated without complex synchronisation mechanisms. MariaDB is extended through replication and sharding to efficiently process growing amounts of data, while HL7 FHIR data retrieval can be parallelised to connect multiple hospital management systems simultaneously.

Kubernetes orchestration handles automated load balancing, self-healing and rolling updates, ensuring continuous operation even with increasing user numbers or new function modules. For computationally intensive tasks such as AI-supported history prediction, dedicated resources such as GPU nodes can be integrated without affecting the core functions of the backend. Monitoring and observability tools are integrated into the cluster environment, enabling early detection of load peaks and automatic scaling of services.

This combination of containerised deployment, Kubernetes orchestration and modular interfaces keeps the DAIsy backend flexibly expandable and allows it to reliably serve both small facilities and large hospital networks without compromising the transparency or integrity of patient data.

2.7 Reliability and Fault Tolerance

The DAIsy therapist dashboard backend is designed for high reliability and fault tolerance to ensure stable and continuous operation in a clinical environment.

It is based on deployment in the Kubernetes cluster, which automatically monitors services and restarts them independently in the event of failures. Kubernetes provides self-healing by replacing faulty pods and updating deployments without interruption through rolling updates. The architecture is deliberately designed to be stateless for the REST interfaces to the dashboard, patient app, Keycloak and AI service, so that instances can be scaled horizontally or restarted at any time without losing session information.

MariaDB is secured by replication, ensuring consistent data access even in the event of a node failure. In addition, sharding mechanisms can be used to distribute peak loads and keep database performance stable even as data volumes grow. Redundant connections are provided for HL7 FHIR data retrieval from hospital management systems, so that the failure of individual interfaces does not cause an interruption in the overall system.

The security and authentication mechanisms via Keycloak are also designed to be highly available. Centralised management of tokens and roles allows multiple instances to be operated, which are automatically taken over in the event of failures.

Monitoring and observability tools such as Prometheus and Grafana are integrated into the cluster environment and continuously provide metrics on system status, response times and error rates. This data enables automated scaling decisions and early detection of anomalies.

Reliability mechanisms are designed to support continuous clinical operation, ensuring that temporary service interruptions do not compromise data consistency or availability required for ongoing patient monitoring and care.

2.8 Deployment

Deployment as a microservice in a Docker container on a Kubernetes cluster

To deploy a service within a Docker container to a Kubernetes cluster, an image repository is essential. This repository will store the Docker images that Kubernetes will pull and run on its pods. Once the image repository is set up and linked to the Kubernetes cluster, the Kubernetes manifests is used to define the deployments, including the image to be used. The service is equipped with a Swagger interface. Swagger, also known as OpenAPI, provides a developer-friendly framework for describing the API of the service. This facilitates easier integration and control by providing a clear contract for the API and interactive documentation.

3 Modular creation of teaching content

3.1 Functionality

The task of this component is the creation of personalized e-learning content for patients involving a sophisticated process where content modules designed in WordPress are integrated with structures defined in NodeRed. This integration facilitates the generation of equivalent content that is optimized for a patient app, ensuring that the information is accessible and tailored to individual patient needs. The end goal is to provide a seamless educational experience that supports patients in managing their health effectively.

3.2 Interfaces

The integration of REST interfaces with WordPress and NodeRed facilitates a seamless communication process. WordPress provides the HTML structure, while NodeRed offers the JSON structure, ensuring that data is efficiently managed and presented. The eLearning views are dynamically generated within the patient app, providing an interactive and user-friendly experience. Furthermore, the patient's progress is consistently monitored and updated to the therapist's dashboard backend via REST interfaces, allowing for real-time tracking and management of the patient's development.

3.3 Dependencies

For the generation a Wordpress as well as a NodeRed instance is required, the synchronization of the various content is done in ASP.Net core service. the patient application runs in an Ionic framework.

Various software programs are required to publish newly created NodeRed nodes. Verdaccio is a private npm-Registry that specifies how the new nodes must be structured. The Catalogue-Builder converts the private npm-packages from Verdaccio into a format that NodeRed can understand. To ensure this, a program is required that processes the WordPress export. In this case, this was implemented using Java. The Java Program is based on Java Spring-Framework. Each of these individual components can be provided with Docker images.

3.4 Architecture

Figure 1 describes the communication and structure of the therapy dashboard and patient app. Questionnaires are created in WordPress and these questionnaires are to be displayed later in the patient app. However, it should be possible to edit the questionnaires using NodeRed.

The Diagram in Figure 2 shows exactly which intermediate steps are necessary to establish a connection between WordPress and NodeRed. The WordPress export files can be sent to a Java program via REST. There, this data is processed and then forwarded to Verdaccio, a package manager. From there, the data goes to the Catalogue -Builder. NodeRed can then access the Catalogue-Builder and install the required data. Verdaccio and the Catalogue-Builder must be used due to NodeRed restrictions to make your own packages usable for NodeRed.

NodeRed finally returns a JSON object representing the flow. The patient app can then use this JSON object to display the corresponding modified questionnaire. However, to ensure that all information from the WordPress export, such as links to images, etc., can also be transferred, the JSON object has been expanded to include the 'Context' field. This field contains all additional information required to recreate the questionnaire in the app.

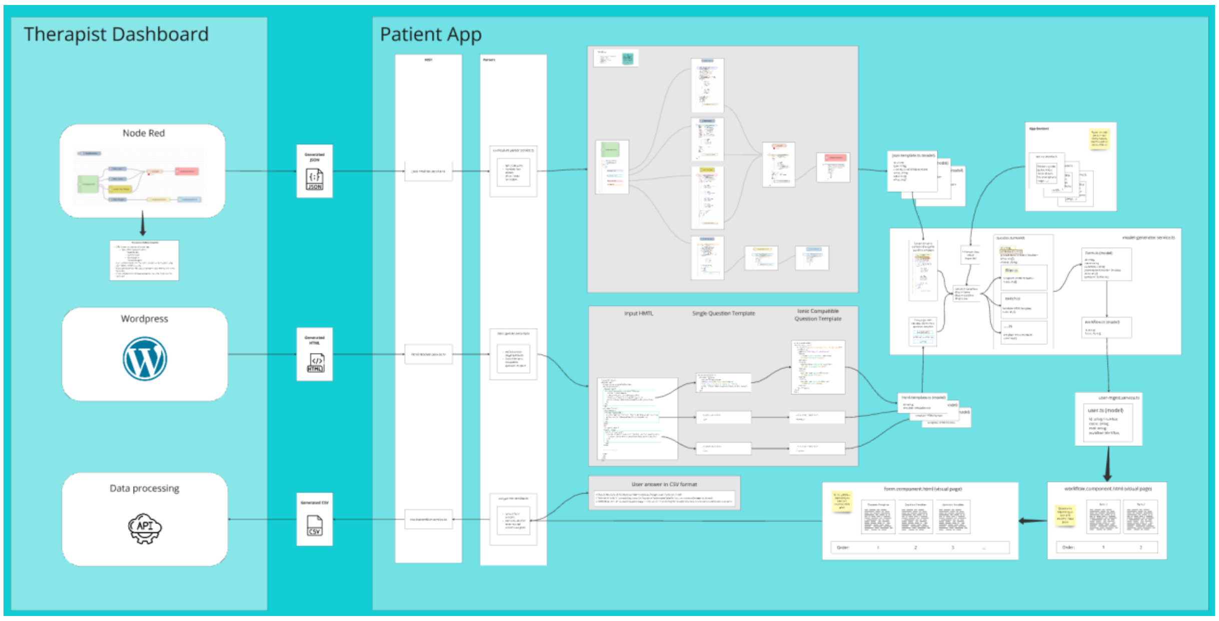


Figure 1: Architecture

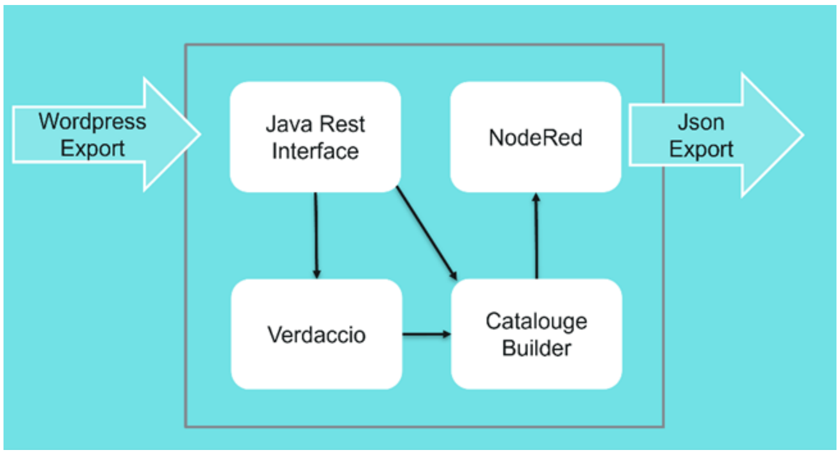


Figure 2: NdoeRed Architecture:

3.5 Performance Metrics

Due to the long processing chain from WordPress to NodeRed, measures were taken to keep latency as low as possible. To avoid having to load each package individually into Verdaccio, which can take quite some time with hundreds of packages, individual nodes were combined into 49 packages. 49 is a restriction imposed by Verdaccio

3.6 Reliability and Fault Tolerance

When examining the structure of the software solution, two potential sources of error immediately become apparent. One is the transition from WordPress to NodeRed, and the other is NodeRed itself.

All services used to transfer the WordPress export to NodeRed could fail due to unexpected errors, but since they are hosted in a Kubernetes cluster, they should restart automatically in the event of a software crash. The standardized WordPress export should not cause any errors during processing.

Since NodeRed is an established and thoroughly tested software, it is unlikely that major errors will occur here. Due to the rather complex entry barrier of NodeRed, users may be more prone to making mistakes when using it. To provide support here, four help tours have been created and integrated to facilitate the initial use of NodeRed.

3.7 Deployment

Deployment as a microservice in a Docker container on a Kubernetes cluster

To deploy a service within a Docker container to a Kubernetes cluster, an image repository is essential. This repository will store the Docker images that Kubernetes will pull and run on its pods. Once the image repository is set up and linked to the Kubernetes cluster, the Kubernetes manifests is used to define the deployments, including the image to be used. The service is equipped with a Swagger interface. Swagger, also known as OpenAPI, provides a developer-friendly framework for describing the API of the service. This facilitates easier integration and control by providing a clear contract for the API and interactive documentation.

Deploying an Ionic app on Android

Deploying an Ionic app on Android is a multi-step process that includes installing Android Studio, creating a release build of the app, signing the APK file, optimizing it, and finally publishing it to the Google Play Store. It is important that every step is done carefully to ensure that the app works correctly and complies with the guidelines of the Google Play Store. The use of commands such as 'ionic cordova build', 'keytool', 'jarsigner' and 'zipalign' are crucial. After optimizing the APK file and filling in all the necessary information in the Google Play Console account, the app can be made accessible to users worldwide.

4 FitSprite Nutrition Service

4.1 Functionality

The system is designed to support doctors in monitoring patients with eating disorders by tracking their dietary habits, while also providing patients with insights into their physical activity and food intake. One of the key features is the automatic estimation of calorie intake through image analysis. To achieve this, three Deep Learning (DL) models are employed:

- Food recognition model – identifies the food items present in an image.
- Segmentation model – isolates the food from the background.

- Depth estimation model – generates a depth map of the image to improve portion size estimation.

All models are developed in Python using the TensorFlow framework.

An AI-powered insight generator further enhances the system by (1) creating meaningful insights from patient data, and (2) continuously adapting to user preferences. This ensures that recommendations remain relevant and actionable. By combining activity and nutrition data, the system delivers personalized guidance to help patients improve their health outcomes.

4.2 Interfaces

The FitSprite Nutrition Service interacts with the broader system through well-defined interfaces.

- Data input: Patients submit information via the mobile application.
- Data transmission: Input is sent to the database using secure web API calls.

This setup ensures seamless communication between patients, clinicians, and the backend system.

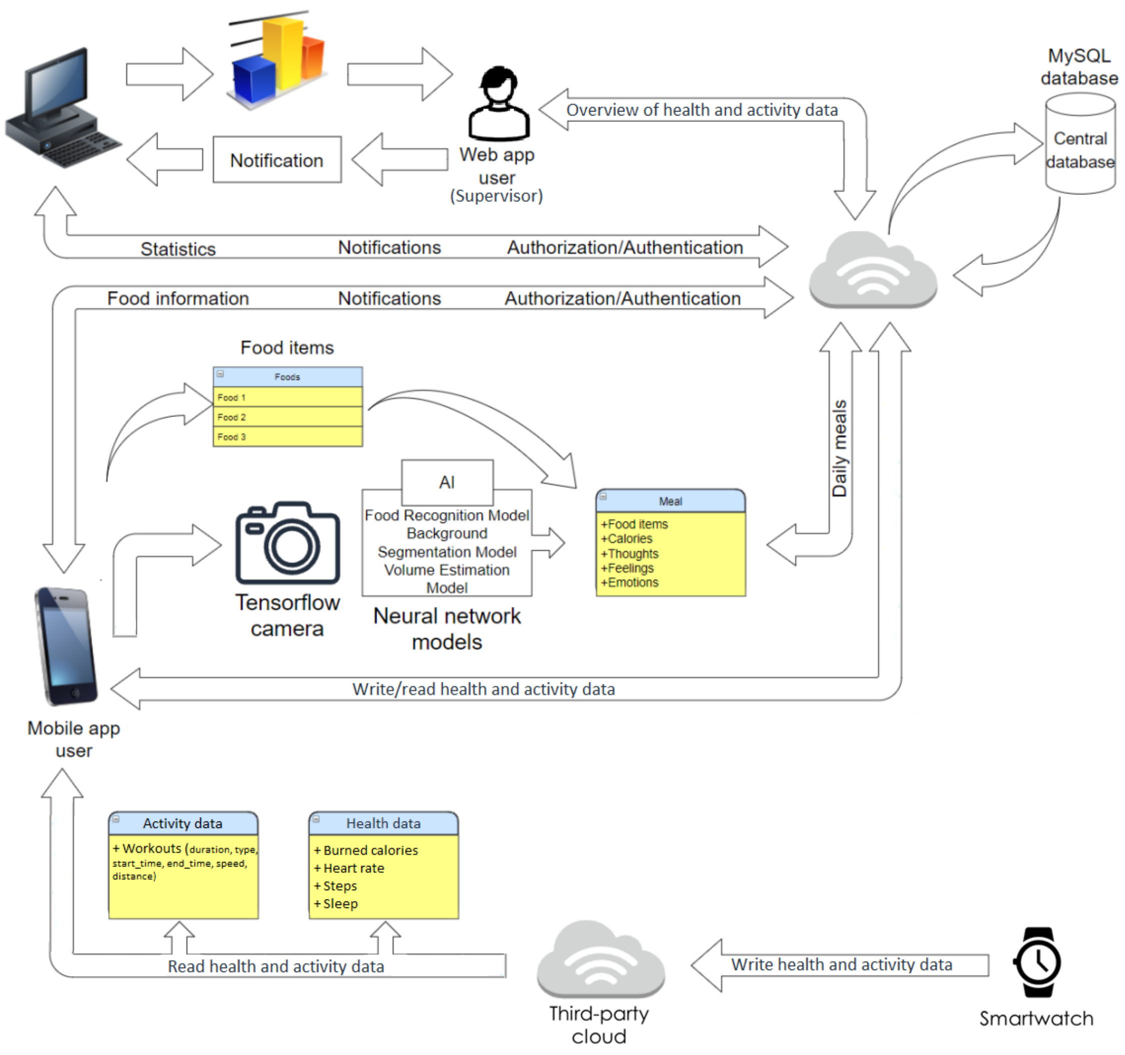
4.3 Dependencies

The system is built using TensorFlow for machine learning, React Native for mobile app, React for web portal, and PHP with REST APIs for backend services and MySQL for structured storage and retrieval.

. Additional libraries support state management, UI components, data visualization, localization, notifications, and performance optimization.

4.4 Architecture

The system's main components are: Deep Learning models for food recognition, mobile app to be used by the patients to record calorie intake and take over the physical activity data, backend system and the web portal for clinician to manage patients of the system.



4.5 Performance Metrics

The system is designed to deliver reliable and efficient performance:

UI rendering for the web navigation portal is maintained under 700 ms for smooth interactions.

4.6 Scalability

AI models: are deployed in Docker containers which makes it possible to spin up multiple instances of the same model when demand increases.

4.7 Deployment

The deployment of the system ensures accessibility across platforms:

- Mobile Application: FitSprite Nutrition is available on both the Apple App Store and Google Play Store.
- Web Application: accessible via https://nutrition-ai.nl

Access credentials:

- Username: 5M.Software / Password: 8b69598f

5 Major Depressive Disorder (Distinguishing Bipolar and Unipolar Depressive Disorder)

5.1 Functionality

This platform implements an end-to-end, orchestrated workflow for AI-supported analysis of MRI studies, and clinical information all integrated into a clinical environment and aligned with established imaging and access control standards.

Access to the platform is restricted to authorised users. Clinicians interact with the system through the Doctor Dashboard, where authentication and role-based access control ensure that only permitted users can initiate analyses and access patient-related information. Authentication is enforced consistently across user-facing components and backend services, providing controlled access to imaging data, processing workflows, and results.

The workflow is initiated by a clinician via the Doctor Dashboard and proceeds as a controlled processing job. The system:

- Retrieves patient context from the hospital information system and imaging context from PACS, including study and series identifiers required to locate the MRI data.
- Registers the request as a work item in the Unified Procedure Step (UPS) framework to enable controlled workflow management. The creation and handling of UPS work items follows the DICOM standard, ensuring consistency with established imaging workflow definitions.
- Orchestrates the execution of processing stages through a dedicated orchestration component.
- Performs MRI–specific data preparation by retrieving raw DICOM data from PACS and producing the intermediate inputs required by the downstream feature extraction pipeline.
- Persists these produced inputs in a centralized repository, ensuring that subsequent processing operates on a consistent and traceable data set.
- Executes AI-based analysis on the derived feature representations to produce a diagnostic classification, confidence values, and information supporting result interpretation.
- Generates a DICOM Structured Report (SR) encapsulating the analysis outcome and stores it back into PACS as part of the clinical record.
- Enables clinical review through a Viewer that presents brain imaging together with ROI and/or tract overlays and analysis result summaries.
- Finalizes the UPS work item and notifies the Doctor Dashboard that processing has completed.

For image visualisation and review, the platform follows principles aligned with the **Invoke Image Display (IID)** standard, ensuring consistent presentation of imaging content and derived results without duplicating clinical image storage.

For internal processing and data organisation, the preparation and feature extraction stages follow conventions compatible with widely adopted neuroimaging data structuring practices, such as those defined by the **Brain Imaging Data Structure (BIDS)**, while preserving PACS as the authoritative clinical imaging system.

The overall workflow design is aligned with the principles of the IHE AI Workflow for Imaging (AIW-I) profile, supporting the integration of AI-based analysis into routine clinical imaging workflows in a controlled and interoperable manner.

5.2 Interfaces

Clinical and User Interfaces: Doctor Dashboard: initiates analysis requests and receives completion notifications.

Viewer: visualize brain imaging together with ROI/tract overlays and analysis result summaries.

Clinical System Interfaces:

Hospital Information System (HIS): provides patient and administrative context.

PACS: supplies MRI DICOM studies and series information and receives DICOM Structured Report outputs.

Orchestration Interfaces

UPS: manages work item registration, state transitions, and completion.

Orchestrator (Listener): subscribes to UPS work items, claims processing tasks, and coordinates downstream execution.

Data Interfaces:

Repository: persistence layer for the inputs produced during data preparation and consumed by the feature extraction stage.

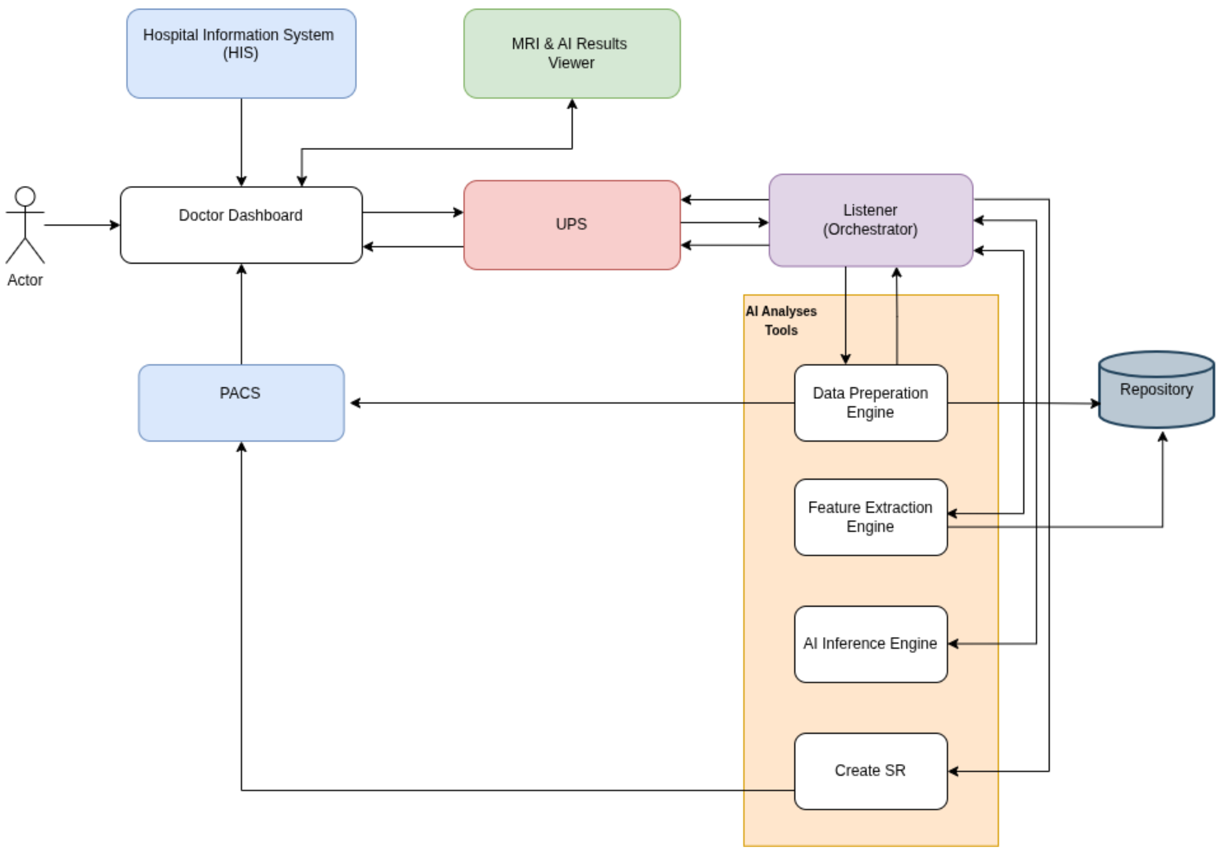
5.3 Dependencies

The platform relies on modular, Dockerized services and standard clinical infrastructure components:

- DICOM interoperability capabilities for reading MRI data from PACS and writing structured reporting results back.
- An MRI data preparation toolchain capable of transforming raw acquisitions into processing-ready inputs for feature extraction.
- A feature extraction component that produces standardized representations suitable for AI-based analysis.
- An AI analysis component that generates classification results, confidence values, and interpretation-related information.
- A DICOM Structured Report creation component to serialize analysis results into clinically compatible reports.
- A repository service for persistent storage of prepared inputs.
- A viewer component capable of rendering volumetric brain data together with overlays.

5.4 Architecture

The architecture follows a UPS-orchestrated, repository-centric processing model.



This architecture ensures clear separation of concerns while maintaining traceability across all processing stages.

5.5 Performance Metrics

The The performance characteristics of the platform are determined by the computational properties of its processing stages:

- End-to-end processing time is primarily dominated by MRI data preparation, which involves retrieval of raw DICOM data from PACS and generation of inputs required for feature extraction.
- Feature extraction and AI-analysis operate on repository-stored inputs and therefore execute with comparatively lower latency once preparation has completed.
- The final clinical output of the workflow is the DICOM Structured Report stored in PACS.
- When required, model-level evaluation metrics such as accuracy, area under the ROC curve, and F1-score can be associated with specific analysis configurations as contextual information. These metrics describe validated model behaviour and are not interpreted as patient-specific guarantees.

The Viewer supports interactive clinical inspection of imaging data, tract overlays, and analysis results in order to incorporate eXplainable AI (XAI) for improvement of tools acceptance.

5.6 Scalability

Scalability is achieved through controlled orchestration and parallel execution of processing stages:

- Multiple workflow instances can be executed concurrently under UPS control.
- Feature extraction stages can be executed in parallel by deploying multiple service instances when several studies are processed at the same time.
- Work distribution between the Orchestrator and processing services enables efficient use of available computing resources.

This design allows processing capacity to be increased incrementally without altering the overall workflow or clinical integration.

5.7 Reliability and Fault Tolerance

- UPS-based work item management provides controlled execution and clear completion semantics.
- Persisting prepared inputs in the repository prevents loss of processing state and supports reproducibility.
- Service isolation limits the impact of failures to individual processing stages.
- Persisting DICOM Structured Report results in PACS ensures that clinically relevant outputs remain available independently of user interface components.

5.8 Deployment

All platform components (Orchestrator, data preparation, feature extraction, structured reporting, repository access, and viewer/dashboard backends) are deployed as **Dockerized microservices** on on-premise, hospital-managed infrastructure.

This deployment approach enables:

- independent deployment and update of individual components,
- controlled service restarts and fault isolation,
- environment-specific configuration (e.g. PACS, UPS, and repository endpoints),
- flexible scaling by running multiple instances of selected processing services.