ISABEL

MINIMIZE COSTS AND MAXIMIZE BENEFITS IN THE TRANSITIONARY EUROPEAN REGIONAL LABOUR MARKETS

mInimize coSts and mAximize Benefits in the transitionary European regional Labour markets

Deliverable No 2.5 — ISABEL System Architecture

WP2: Conceptualisation, Co-creation Methodology, and System Architecture

Funded by
the European Union

DOCUMENT ID	D2.5
DUE DATE	30/06/2026
DELIVERY DATE	30/06/2026
WORK PACKAGE	WP2
TASK(S)	T2.5
DISSEMINATION LEVEL	PU
VERSION	V2.0
AUTHOR/S	CERTH
FILENAME	ISABEL_D2.5_System_Architecture_V2.docx

Abstract:

This deliverable defines the system requirements and architecture of the ISABEL. Starting from the storyline and the user requirements gathered during the project's co-creation activities, it specifies the platform's components, layers and design decisions, and provides the architectural reference for the development and validation phases that follow.

Views and opinions expressed in this document are those of the author(s) and do not necessarily reflect those of the European Union or REA. Neither the European Commission nor the granting authority can be held responsible for them.

Funded by
the European Union

Contributors

NAME	ORGANISATION
Michalis Angelou	CERTH
Nicholas Vretos	CERTH
Konstantinos Marthoglou	CERTH
Maria Kaklidou	CERTH
Vasileios Solachidis	CERTH
Nikos Kapitsinis	UCPH

Peer Reviewers

NAME	ORGANISATION
Elena Lasarte	UO
Rafał Muster	US

Versions

Version	Date	Author / Organisation	Description
0.1	05/2026	CERTH	Initial draft and structure.
1.0	06/2026	CERTH	First version of the document
2.0	06/2026	CERTH	Updated following review comments

Funded by
the European Union

Contents

Executive Summary 6
1. Introduction 7
    1.1 Purpose and Objectives 7
    1.2 Architecture at a Glance 7
    1.3 Key Design Decisions 9
2. Requirements Gathering and Analysis 10
    2.1 Requirements elicitation – co-creation 10
    2.2 Use Cases and Scenarios 11
    2.3 Documentation and Traceability 12
3. From Requirements to Functional Specifications 13
    3.1 Prioritisation (MoSCoW) 13
    3.2 Mapping Requirements to Features 16
    3.3 Functional Requirements Specification 21
    3.4 Non-Functional Requirements 21
4. Architecture Vision 22
    4.1 Purpose 22
    4.2 Scope and Boundaries 22
    4.3 Design Principles and Constraints 22
5. Architecture Overview and Data Flows 23
    5.1 High-Level Architecture 23
    5.2 Layers and Responsibilities 23
    5.3 Major Components 25
    5.4 Technology Stack 25
    5.5 The Data Pipeline 26
    5.6 Request and Tool-Call Flow 26
    5.7 Detailed Architecture Views 26
6. Presentation Layer 29
    6.1 Frontend Architecture 29
    6.2 Module Interfaces 29
    6.3 Hosting 29
    6.4 Performance 29
7. Gateway, Services and Data 30
    7.1 OpenWebUI as Middleware 30
    7.2 The ISABEL Tool Server and MCP 30
    7.3 Agentic Logic 30
    7.4 Authentication and Authorisation 30
    7.5 Data Validation and the Medallion Lake 31
    7.6 Logging and Monitoring 31
8. Deployment and Infrastructure 32
9. Conclusion 33
References 33
Appendices 34
    Glossary 34
    Related Deliverables 34

Funded by
the European Union

List of Figures

Figure 1. Layered system architecture: the custom frontend, the OpenWebUI gateway, the ISABEL tool and agent server (MCP), and the Medallion data-lake. 8
Figure 2: Platform concept: the functional layers, the central communication and storage components, and the SNAP, DPWA and FARO/ROMO module flows. 24
Figure 3: Detailed reference views: the OpenWebUI gateway stack, the authentication flow, the Medallion data flow, and the component layout. 28
Figure 4: Deployment: the on-premises super-node — dual RTX 5090 GPUs working in tensor-parallel mode for LLM inference, with MinIO-backed Medallion storage on a local NVMe array. 32

List of Tables

Table 1: Schedule of elicitaiton activities 10
Table 2: Requirements areas mapped to architectural responses 10
Table 3: High-level use cases 11
Table 4: Detailed user scenarions 11
Table 5: Requirements traceability matrix. 12
Table 6: MoSCoW prioritisation of all user requirements. 13
Table 7: User requirements mapped to features and components 16
Table 8: Main technology choices 25

List of Abbreviations

Acronym	Definition	Acronym	Definition
API	Application Programming Interface	OIDC	OpenID Connect
CI/CD	Continuous Integration / Continuous Deployment	OLAP	Online Analytical Processing
DPWA	Digital Policy Writer Assistant	PII	Personally Identifiable Information
GIS	Geographic Information System	RAG	Retrieval-Augmented Generation
GDPR	General Data Protection Regulation	REST	Representational State Transfer
JWT	JSON Web Token	SNAP	Smart oNe-stop digitAl data shoP
KPI	Key Performance Indicator	SSO	Single Sign-On
MinIO	(object storage product name)	TLS	Transport Layer Security
OCR	Optical Character Recognition	XAI	Explainable AI

Funded by the European Union

Executive Summary

This deliverable, D2.5 “ISABEL System Architecture”, sets out the technical design of the ISABEL platform and explains how the requirements gathered during the project’s co-creation work are turned into something that can be built. It is the reference document that the development, testing and integration teams will work from during the rest of the project.

ISABEL serves as a research and decision support platform for researchers and policy-makers. It brings together statistics and documents from many European and national sources, harmonises them, and lets policy-makers and researchers explore skill gaps and employment trends through dashboards and a conversational interface. Because the data is sensitive and the analysis feeds policy-makers, two requirements shaped almost every design choice: keep the data inside the ISABEL project and the European environment in general, and keep the reasoning behind every result transparent and checkable.

The architecture is organised around four layers. A custom in-house developed frontend gives each type of user a workspace suited to their job. A local OpenWebUI deployment will stand as the middleware: it handles user accounts, exposes a documented API, and brokers every call to tools and to the language model. A set of ISABEL-built services – some also packaged as Model Context Protocol (MCP) tools and some as services, and a FastAPI tool server - carries out the actual work, from querying data to running the SNAP and FAROROMO modules. Underneath, a Medallion approach data-lake (Bronze, Silver and Gold tiers) stores raw and immutable inputs, refined data and the analysis-ready knowledge that the platform serves back.

Two decisions are worth highlighting up front. First, the platform runs its own language model - Mistral-Small-24B-Instruct-2501, served with vLLM across the two GPUs using tensor parallelism - on hardware at CERTH premices, so that no data is sent to commercial AI services. Second, the frontend never talks to tools, data or the model directly; everything goes through OpenWebUI, which gives the project a single place to apply authentication, access control and logging.

The rest of the document moves from requirements to implementation. Part 1 records the user requirements, how they were prioritised, and how they map onto concrete features. Part 2 describes the architecture in detail — the layers and components, the data pipeline, the request flow, security, and the hardware the system runs on. Diagrams accompany the sections where a picture is clearer than prose. Together these provide a complete and practical basis for the development and validation phases that follow.

Funded by
the European Union

1. Introduction

1.1 Purpose and Objectives

ISABEL helps researchers and policy-makers make sense of regional labour markets that are changing quickly. The data behind these markets is fragmented: it lives in different national registries, in statistical portals such as Eurostat and ELSTAT, and in long PDF reports. ISABEL's job is to pull this material together, make it consistent, and present it in a way that supports concrete decisions about skills and employment, and on top of that help researchers do their work better and faster.

From an engineering point of view, that goal translates into a few practical objectives. The platform has to ingest data from many sources without manual reformatting each time. It has to store that data so that its origin and processing history stay auditable. It has to expose analysis and a natural-language interface that non-technical users can actually use. And it has to do all of this while keeping sensitive data under European control. The sections that follow describe how the design meets each of these objectives.

1.2 Architecture at a Glance

The platform is built as four cooperating layers. Users work in a custom frontend. The frontend talks to OpenWebUI, which acts as the middleware and gateway for the whole system. OpenWebUI in turn calls ISABEL's tools and services and, when needed, the local language model. All of these read from and write to a Medallion approach data-lake. Figure 1 shows how the layers fit together, from the frontend through the gateway and the MCP-based tool and agent server down to the Gold-tier stores.

Funded by
the European Union

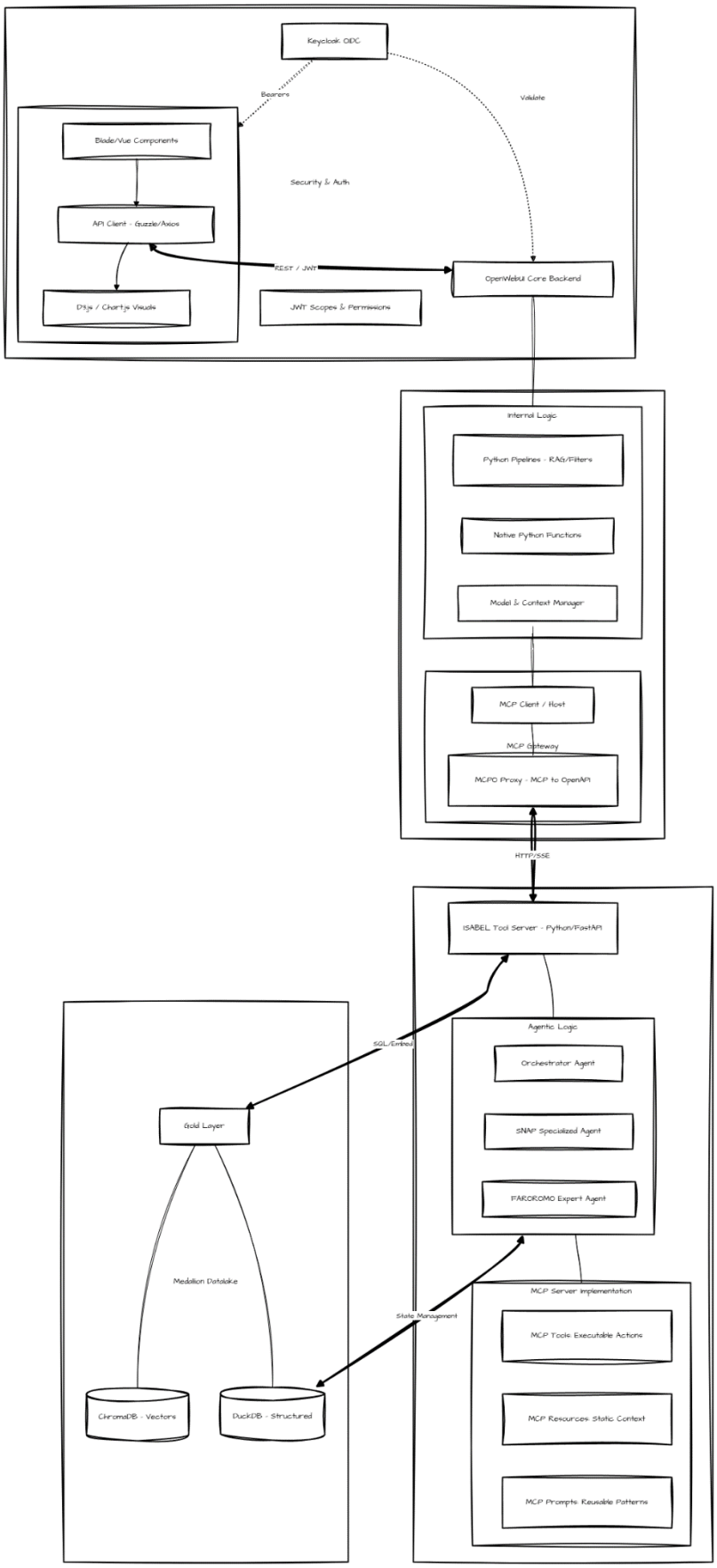


Figure 1. Layered system architecture: the custom frontend, the OpenWebUI gateway, the ISABEL tool and agent server (MCP), and the Medallion data-lake.

Funded by
the European Union

Rather than assemble the platform from generic cloud services, the project uses a focused, research-oriented stack hosted at CERTH's premises. OpenWebUI will provide ready-made building blocks the project would otherwise have to write itself: account and session management, an OpenAPI/Swagger interface, a client for language models, and a host for MCP tools. Building on top of it lets the team concentrate on the parts that are specific to ISABEL — the data tools, the analytical modules and the agents — instead of re-implementing infrastructure.

1.3 Key Design Decisions

A handful of decisions explain most of the architecture and are referenced throughout the document:

- **OpenWebUI as the single gateway.** The frontend has no direct route to tools, data or the model. Every request passes through OpenWebUI, which is where authentication, role checks and logging are enforced. This keeps security in one place and makes the system easier to reason about.
- **A local language model.** ISABEL runs Mistral-Small-24B-Instruct-2501 itself, served with vLLM on the two on-premises GPUs using tensor parallelism at CERTH. Sensitive labour-market data is never sent to third-party APIs, which is what makes the platform's data-sovereignty promise real rather than aspirational. [Jiang et al., 2023 — Mistral; won et al., 2023 — vLLM], [European Parliament, 2016 — GDPR]
- **Tools and services behind MCP.** ISABEL's capabilities are exposed as Model Context Protocol tools and through a FastAPI tool server. The frontend invokes them through the gateway, so new tools can be added without changing the user interface [Anthropic, 2024 — MCP].
- **A Medallion approach data-lake.** Data moves through Bronze (raw and immutable), Silver (cleaned, normalised and indexed, analysis-ready) and Gold (knowledge) tiers. The Silver tier holds DuckDB [Raasveldt & Mühleisen, 2019 — DuckDB] tables for analytics and curated, policy-ready knowledge assets. Semantic vector search is planned for a future release as the platform's embedding infrastructure matures. [Armbrust et al., 2021 — Medallion/Lakehouse]
- **People in the loop.** Automated processing is paired with expert review. A domain expert can inspect what the model produced, correct it, and approve it before it is promoted to the Gold tier and shown to end-users [Mosqueira-Rey et al., 2023 — HITL].

The remainder of the deliverable is in two parts. Part 1 covers the user requirements and how they become functional specifications. Part 2 describes the system architecture in detail.

2. Requirements Gathering and Analysis

This section explains how the project collected its requirements and how those requirements are documented so they can be traced through to the design. ISABEL was specified through co-creation: the people who will use the platform helped shape what it should do, and their feedback was filtered against the security and data-sovereignty constraints that the architecture has to respect.

2.1 Requirements elicitation – co-creation

Requirements were gathered in stages rather than in a single exercise. The first round took place at the project's first General Assembly in Copenhagen premises (on M4), where partners agreed the baseline architecture and the principle of on-premises hosting. Online workshops then brought in external experts and prospective users, which sharpened the user-interface needs and confirmed that experts would need a way to review and correct the platform's output. A dedicated session in London updated the project point of view on the approach and shared anew this vision with external experts and users. Along with these events, the project's Living Labs will provide a continuous stream of feedback as real users exercise the interface and the local model.

This feedback is what pushed the project away from generic cloud deployments and towards a local, controlled environment. The requirement that sensitive regional data must not leave the European ecosystem is the reason ISABEL runs its own language model, uses OpenWebUI for orchestration, and keeps its data in a Medallion lake on CERTH hardware. Table 1 lists the elicitation activities; Table 2 connects the main requirement areas to the parts of the architecture that answer them.

Table 1: Schedule of elicitaiton activities

Activity / event	Location	Audience	Main outcome
First General Assembly	UCPH	Project partners	Baseline architecture and initial tools and platform discussed.
Online co-creation workshops	Virtual	External experts, prospective users	UI/UX needs identified; need for expert review confirmed.
ISABEL platform session	London	Scientists, domain experts	FARO ROMO logic validated; Bronze/Silver/Gold lake adopted.
Living Labs (ongoing)	Regional hubs	Policy-makers, citizens, stakeholders, etc	Continuous testing of response times and model accuracy.

Table 2: Requirements areas mapped to architectural responses

Requirement area	Architecture component	Approach
High-volume data ingestion	Ingestion jobs	Scheduled GitLab CI/CD pipelines; asynchronous batch processing.
Secure access & data	Gateway + auth	Keycloak OIDC / OpenID Connect, JWT tokens,

Requirement area	Architecture component	Approach
sovereignty		RBAC; local LLM; TLS termination at the Nginx gateway.
Data harmonisation	Medallion lake	Bronze / Silver / Gold tiers; Silver indexed for retrieval.
Scientific accuracy	HITL validation	Experts approve model output before it reaches users.

2.2 Use Cases and Scenarios

A few representative use cases show how the layers cooperate in practice. They span the main kinds of user the platform serves: policy-makers asking questions, domain experts checking results, and the automated pipelines that keep the data current. Table 3 lists the headline use cases and Table 4 walks through the scenarios in more detail. These are note decided use cases and scenarios, since ISABEL will not have pilots to test UCs, yet they are indicative workflows to aid the elicitation procedure.

Table 3: High-level use cases

ID	Use case	Primary actor	Description
UC-001	Labour-market query	Policy-maker	Asks, in plain language, for a regional skill-gap related intervention and receives a sourced answer.
UC-002	Expert validation	Domain expert	Reviews a model-generated synthesis, corrects it, and approves it for publication.
UC-003	Automated ingestion	Platform	A scheduled job ingests new regional data and refreshes the indexes.

Table 4: Detailed user scenarions

Scenario	Use case	Pre-conditions	Main flow	Result
SCN-001	UC-001	User authenticated; Gold tier populated.	Policy-maker selects a region and indicator and asks for a 5-year forecast; the gateway routes the request to the KPI engine and retrieves supporting documents.	A trend chart with forecast intervals and source citations.
SCN-002	UC-002	Silver records await review; user has the reviewer role.	Expert opens the validation view, edits the extracted entities, and approves the record.	Record promoted to Gold; corrections stored for later model tuning.
SCN-003	UC-003	A source publishes a new quarterly dataset.	A scheduled job pulls the raw file (Bronze), cleans it (Silver), and regenerates embeddings.	Vector index refreshed; the run is logged.
SCN-004	UC-001	User has a local spreadsheet.	User drops the file into the ad-hoc query view; DuckDB loads it for the session alongside Gold data.	Results shown; session data discarded at logout.
SCN-005	UC-002	User is drafting a policy brief.	The brief co-pilot suggests evidence from the lake, drafts sections, and inserts citations.	A draft brief with every claim linked to a source.

Funded by the European Union

2.3 Documentation and Traceability

Because ISABEL's outputs can inform policy, the path from a raw record to a published insight has to be auditable. Each transformation in the data-lake is recorded, and corrections made by experts are kept so the team can see how a result was produced and, over time, use those corrections to improve the model. Table 5 is the requirements traceability matrix.

Table 5: Requirements traceability matrix

Req. ID	Requirement	Component	Test case	Status
REQ-DATA-01	Raw data is retained immutably for audit.	Bronze tier	TC-DAT-001	Approved
REQ-SEC-01	Sensitive data never reaches external AI APIs.	Local LLM + gateway	TC-SEC-004	Implemented
REQ-AI-01	Forecasts are verifiable by human experts.	HITL validation	TC-AI-002	Approved
REQ-INT-01	Background jobs do not slow the interface.	GitLab runners	TC-SYS-008	Implemented

Funded by
the European Union

3. From Requirements to Functional Specifications

This section turns the collected requirements into something the development team can act on. It records the full list of 68 user requirements, the priority assigned to each, and the platform feature that answers it. The requirements themselves are reproduced in full in the tables below; the narrative after each table explains the priorities and groups the requirements by area so the reasoning is easy to follow.

3.1 Prioritisation (MoSCoW)

Requirements were prioritised with the MoSCoW method - Must, Should, Could and Won't - using the feedback from the co-creation workshops, and direct feedback from the project partners. The aim is to make sure the core of the platform is built and validated first. Of the 68 requirements, 26 are rated Must, 32 Should and 10 Could. No requirements came up that then were assigned as "Won't". Table 6 lists every requirement with its priority.

Table 6: MoSCoW prioritisation of all user requirements.

UR ID	Requirement	Description	Priority
UR_GEN_01	Authentication & Authorisation	Secure login with support for organisational roles.	Must
UR_GEN_02	User Personalisation	Users must have an account to store preferences (language, theme, default views).	Must
UR_GEN_03	System Scalability	Ability to handle increasing data volumes without performance loss.	Should
UR_GEN_04	Audit Logging	Full traceability of user actions for security compliance.	Must
UR_GEN_05	Accessibility	Ensure the platform is usable by people with disabilities.	Must
UR_GEN_06	Low Latency Performance	Dashboards and queries must load quickly.	Should
UR_GEN_07	API-First Architecture	The system should be extensible by third-party apps.	Could
UR_WRK_01	Project Initialisation	Ability to create a distinct project container with defined metadata (Goals, Scope).	Must
UR_WRK_02	Role-Based Templates	Initialize workspace layout based on user intent (Researcher vs. Policy Maker).	Should
UR_WRK_03	Context Persistence	Set project-wide preferences (e.g., Default Region) to avoid repetitive filtering.	Should
UR_WRK_04	Context-Sensitive Toolbars	Only tools relevant to the active view (Chart vs. Document) should be visible.	Must
UR_WRK_05	Adaptive Feature Visibility	Disable/Hide actions that cannot be performed on the specific selected object.	Should
UR_WRK_06	Contextual Tool Hints	Users need quick, in-app guidance on how to use specific tools.	Should
UR_WRK_07	Interactive Walkthroughs	Users need deep-dive tutorials and step-by-step guides for complex workflows.	Should
UR_WRK_08	Map Preferences	Store personal viewing preferences (default base map, color schemes).	Should
UR_DAT_01	Official Data Ingestion	Support ingestion from official authorities (ELSTAT, Eurostat, World Bank).	Must

Funded by the European Union

UR ID	Requirement	Description	Priority
UR_DAT_02	Unofficial Source Integration	Integration of unofficial sources (LinkedIn, social media, job portals).	Should
UR_DAT_03	Multi-Format Compatibility	Compatibility with CSV, Excel, JSON, XML, SQL.	Must
UR_DAT_04	Automated Cleaning	Automated cleaning of inconsistencies, duplicates, and syntax errors.	Must
UR_DAT_05	Data Imputation	Imputation techniques for missing data.	Should
UR_DAT_06	Outlier Management	Outlier management via statistical thresholds.	Should
UR_DAT_07	Transparency & Oversight	No silent deletions; oversight on cleaning actions.	Must
UR_DAT_08	Dataset Creation	Ability to create and save new custom datasets.	Should
UR_DAT_09	Metadata Management	Management of metadata (source origin, update history) for all datasets.	Must
UR_DAT_10	Ad-Hoc Data Virtualisation	Query temporary or local files without permanent ingestion.	Must
UR_DAT_11	Automated Report Scraper	Automatically download all files (PDF, XLSX) from a specific authority URL.	Must
UR_DAT_12	Intelligent Table Extraction	Locate and parse a specific table within unstructured documents (PDFs).	Must
UR_DAT_13	Temporal Aggregation	Automatically detect frequency (eg Monthly) and aggregate to Yearly if needed.	Must
UR_DAT_14	Spatial Data Import	Users need to import their own map files and geospatial data.	Should
UR_ANA_01	Core KPI Calculation	Calculate Unemployment, Vacancy Rate, Skills Mismatch, etc.	Must
UR_ANA_02	Time-Series Forecasting	Time-Series Analysis & Forecasting (ARIMA, Transformers).	Must
UR_ANA_03	Scenario Analysis	Scenario Analysis for policy changes and economic shifts.	Could
UR_ANA_04	External Scripting	Integration with R, Python, and SPSS.	Could
UR_ANA_05	AI Methodology Assistant	AI-assisted recommendation for selecting appropriate methodologies.	Could
UR_ANA_06	Policy Alignment	Tools to check compliance of trends with specific regulations.	Could
UR_VIS_01	Map Visualisation	Maps for visualisation of regional disparities.	Should
UR_VIS_02	Line Charts	Line charts for trends and forecasting visualisation.	Must
UR_VIS_03	Extended Charting	Extended Charts (Bar, Histogram, Scatter, Boxplot).	Should
UR_VIS_04	Interactive Dashboards	Interactive Dashboards with filtering and drill-down.	Must
UR_VIS_05	Export Capabilities	Export capabilities (PDF, Word, PNG, SVG).	Must
UR_VIS_06	FAROROMO GIS Interface	Access to a dedicated observatory for presenting data on a dynamic map.	Must
UR_VIS_07	Geospatial Policy Simulation	Visualize "What-If" policy simulations directly on the map.	Should
UR_VIS_08	Custom Layer Builder	Create, aggregate, and reuse custom map layers.	Should
UR_VIS_09	Map Presentation Mode	Save specific map setup as a "Presentation" to share.	Must
UR_REP_01	Geospatial Export	Export the map data (for GIS tools) and map image (for reports).	Must
UR_SEC_01	Explainable AI (XAI)	Explanation of AI models and predictions.	Should
UR_SEC_02	GDPR Compliance	Data Anonymisation & Governance.	Must

Funded by
the European Union

UR ID	Requirement	Description	Priority
UR_SEC_03	Bias Mitigation	Bias Mitigation for protected attributes	Should
UR_AI_01	Conversational Analysis	Conversational Data Analysis	Could
UR_AI_02	Research Simulation	Persona Generator & Simulator for pilot testing.	Could
UR_AI_03	Agentic Document Analysis	Agentic Paper Analysis (RAG) for summaries and Q&A.	Should
UR_AI_04	Domain Expertise	The system must understand specific economic/policy terminology.	Should
UR_AI_05	Drafting Assistance	Generate full policy drafts from outlines.	Could
UR_AI_06	Context Awareness	Reference existing laws/reports during generation.	Should
UR_AI_07	Citation & Sourcing	Claims must be linked to specific source documents.	Must
UR_AI_08	Summarisation	Summarize long policy papers into Executive Summaries.	Should
UR_AI_09	Tone Adjustment	Ability to rewrite text for different audiences.	Could
UR_AI_10	Template Compliance	Outputs must respect specific document formatting.	Could
UR_AI_11	Unified Multi-Source Querying	Ask Natural Language questions across mixed sources (Data-lake + Local + Temp) simultaneously.	Should
UR_AI_12	Speech-to-Text Transcription	Users need to convert audio recordings (interviews) into text for analysis.	Should
UR_AI_13	Thematic Analysis & Clustering	Users need to automatically identify recurring topics/themes across interview/chat logs.	Should
UR_AI_14	Sentiment & Emotion Analysis	Users need to gauge the mood and specific emotions in stakeholder interviews.	Should
UR_COL_01	Project Sharing	Ability to share projects and collaborate with teams.	Should
UR_COL_02	Map State Sharing	Share specific map presentations with others.	Must
UR_AI_15	Survey Quality Auditor	Users need to detect ambiguous phrasing, leading questions, and logical errors in questionnaires before launch.	Should
UR_AI_16	Representative Population Synthesis	Users need to simulate large-scale survey results using stratified sampling to match specific demographic distributions (census data).	Should
UR_AI_17	Holistic Data-lake Agent	Users need to query the entire Data-lake (structured stats + unstructured reports) via a single prompt to find supporting evidence.	Must
UR_AI_18	Policy Brief Co-Pilot	Users need a guided workflow that "drives" them to write a policy brief, automatically suggesting data points to back up their claims.	Should

The Must-have set defines the platform's backbone: secure access (UR_GEN_01, UR_GEN_04), ingestion from official sources and in multiple formats (UR_DAT_01, UR_DAT_03, UR_DAT_10–UR_DAT_13), data lineage (UR_DAT_09), the core analytics and forecasting (UR_ANA_01, UR_ANA_02), the interactive map and dashboards (UR_VIS_04, UR_VIS_06, UR_VIS_09), export (UR_VIS_05, UR_REP_01), GDPR handling (UR_SEC_02), citation of AI claims (UR_AI_07), the data-lake agent (UR_AI_17) and map-state sharing (UR_COL_02). These are the items without which the platform cannot deliver its purpose, and they are scheduled first.

Should-have items add depth - personalisation and role-based workspaces, data cleaning and imputation, extended charting, explainability and bias checks, and most of the AI knowledge and policy tooling - and are planned once the backbone is stable. Could-have items, such as scenario

Funded by
the European Union

simulation (UR_ANA_03), external scripting (UR_ANA_04), conversational analysis (UR_AI_01) and persona simulation (UR_AI_02), are valuable but deferred so they do not compete with core work.

3.2 Mapping Requirements to Features

Each requirement is answered by a specific platform feature and lives in a specific part of the architecture. Table 7 records that mapping for all 68 requirements: the requirement, the feature that satisfies it, and the component responsible. The proposed feature descriptions are the ones agreed during specification; the component column ties each to the layer that owns it.

Table 7: User requirements mapped to features and components

UR ID	Need	Mapped feature	Component
UR_GEN_01	Secure login with support for organisational roles.	SSO & RBAC Gateway: Integration with Enterprise Single Sign-On (SSO) and a granular Role-Based Access Control (RBAC) system.	Gateway + RBAC service; container host
UR_GEN_02	Users must have an account to store preferences (language, theme, default views).	User Profile Service: A persistent storage layer linked to the User ID that retains configuration states (language, dark/light mode, default dashboard).	Gateway + RBAC service; container host
UR_GEN_03	Ability to handle increasing data volumes without performance loss.	Cloud-Native Auto-Scaling: Architecture deployed on Docker containers capable of horizontal scaling.	Gateway + RBAC service; container host
UR_GEN_04	Full traceability of user actions for security compliance.	A tamper-proof logging service that records data access, changes, and model executions.	Gateway + RBAC service; container host
UR_GEN_05	Ensure the platform is usable by people with disabilities.	WCAG 2.1 Compliance Layer: UI framework adherence to accessibility guidelines (contrast, screen readers).	Gateway + RBAC service; container host
UR_GEN_06	Dashboards and queries must load quickly.	In-Memory Caching (Redis): Caching strategies to ensure dashboard visuals render in under 2 seconds.	Gateway + RBAC service; container host
UR_GEN_07	The system should be extensible by third-party apps.	Exposure of core platform functionality via secure, documented APIs.	Gateway + RBAC service; container host
UR_WRK_01	Ability to create a distinct project container with defined metadata (Goals, Scope).	A multi-step UI to instantiate a new project workspace, defining title, description, timeframe, and tags for future retrieval.	Custom frontend; project & UI-state services
UR_WRK_02	Initialize workspace layout based on user intent (Researcher vs. Policy Maker).	Bootstrapper that configures UI layout. "Researcher Mode" loads data grids/code; "Policy Mode" loads dashboards/drafting tools.	Custom frontend; project & UI-state services
UR_WRK_03	Set project-wide preferences (e.g., Default Region) to avoid repetitive filtering.	Configuration module where users define "Project Constants" (e.g., Region="Thessaly") applied automatically to all new charts/queries.	Custom frontend; project & UI-state services
UR_WRK_04	Only tools relevant to the active view (Chart vs. Document) should be visible.	Dynamic Component Mounter: A UI framework state manager that unmounts irrelevant toolbars. (e.g., The "Text Formatting" ribbon is hidden when the user is focusing on a Data Grid).	Custom frontend; project & UI-state services
UR_WRK_05	Disable/Hide actions that cannot be	Selection-Based Action Guard: Logic that listens to	Custom frontend;

UR ID	Need	Mapped feature	Component
	performed on the specific selected object.	user selection events; if a static data point is selected, "Time-Series Forecast" buttons are strictly hidden to prevent errors.	project & UI-state services
UR_WRK_06	Users need quick, in-app guidance on how to use specific tools.	Smart Tooltip System: Hover-activated information cards displaying brief usage instructions or mini-GIFs demonstrating the tool's function.	Custom frontend; project & UI-state services
UR_WRK_07	Users need deep-dive tutorials and step-by-step guides for complex workflows.	A help module offering "Tour Guides" (overlay interactive steps showing users where to click) and a searchable library of video tutorials.	Custom frontend; project & UI-state services
UR_WRK_08	Store personal viewing preferences (default base map, color schemes).	Settings module that remembers the user's preferred "Base Map" (Satellite vs. Topo) and color palettes.	Custom frontend; project & UI-state services
UR_DAT_01	Support ingestion from official authorities (ELSTAT, Eurostat, World Bank).	Configurable ETL module with pre-built API adapters for official endpoints.	Ingestion jobs; parsers; Bronze layer
UR_DAT_02	Integration of unofficial sources (LinkedIn, social media, job portals).	External Data Harvesting Engine: Web scraping agents and public API aggregators for real-time trends.	Ingestion jobs; parsers; Bronze layer
UR_DAT_03	Compatibility with CSV, Excel, JSON, XML, SQL.	Multi-Format Ingestion Parser: Engine to normalize structured and semi-structured inputs into a unified schema.	Ingestion jobs; parsers; Bronze layer
UR_DAT_10	Query temporary or local files without permanent ingestion.	Ephemeral Data Engine (DuckDB): In-memory processing layer allowing users to drag-and-drop a file (CSV, Excel) for "Session-only" querying.	Ingestion jobs; parsers; Bronze layer
UR_DAT_11	Automatically download all files (PDF, XLSX) from a specific authority URL.	Recursive Web Crawler: A scraping module that accepts a target URL, detects file links, and batch-downloads them.	Ingestion jobs; parsers; Bronze layer
UR_DAT_12	Locate and parse a specific table within unstructured documents (PDFs).	Layout Analysis Engine: A CV/NLP parser that scans documents for specific visual patterns/headers to identify and extract the correct table.	Ingestion jobs; parsers; Bronze layer
UR_DAT_13	Automatically detect frequency (eg Monthly) and aggregate to Yearly if needed.	Time-Frequency Inference Engine: Logic layer that detects frequency from metadata and applies Sum/Mean aggregation to match target schema.	Ingestion jobs; parsers; Bronze layer
UR_DAT_14	Users need to import their own map files and geospatial data.	Geospatial Uploader: Support for ingesting standard GIS formats (Shapefile, KML, GeoJSON) to overlay external boundaries.	Ingestion jobs; parsers; Bronze layer
UR_DAT_04	Automated cleaning of inconsistencies, duplicates, and syntax errors.	Automated Sanitisation Pipeline: Rules-based processor for detecting and correcting schema violations.	Cleaning pipeline; HITL validation
UR_DAT_05	Imputation techniques for missing data.	Algorithmic Imputation Toolkit: Statistical engine (Mean/Median, KNN, MICE) to resolve data gaps.	Cleaning pipeline; HITL validation
UR_DAT_06	Outlier management via statistical thresholds.	Statistical Anomaly Detector: Automated scanner using Z-score/IQR to flag abnormal data points.	Cleaning pipeline; HITL validation
UR_DAT_07	No silent deletions; oversight on cleaning actions.	Human-in-the-Loop Validation: Staging area for stewards to review/approve cleaning actions before commit.	Cleaning pipeline; HITL validation
UR_DAT_08	Ability to create and save new custom datasets.	No-Code Dataset Builder: Visual interface to merge variables into saved "Custom Data Packages".	Lineage tracker; dataset builder

Funded by
the European Union

UR ID	Need	Mapped feature	Component
UR_DAT_09	Management of metadata (source origin, update history) for all datasets.	Data Lineage Tracker: System logging "Chain of Custody" (origin, time, transformations) for data points.	Lineage tracker; dataset builder
UR_ANA_01	Calculate Unemployment, Vacancy Rate, Skills Mismatch, etc.	Dynamic KPI Engine: Processor with pre-built formulas to compute standard labor market indicators.	KPI engine; forecasting; script sandbox
UR_ANA_02	Time-Series Analysis & Forecasting (ARIMA, Transformers).	Predictive Modeling Suite: Library supporting ARIMA, SARIMA, and Autoformers for trend projection.	KPI engine; forecasting; script sandbox
UR_ANA_03	Scenario Analysis for policy changes and economic shifts.	Counterfactual Simulation Engine: "What-If" tool to tweak variables (e.g., GDP) and view metric impacts.	KPI engine; forecasting; script sandbox
UR_ANA_04	Integration with R, Python, and SPSS.	External Script Sandbox: Dockerized environment for running custom R/Python analysis scripts.	KPI engine; forecasting; script sandbox
UR_ANA_05	AI-assisted recommendation for selecting appropriate methodologies.	Methodology Recommender System: Wizard that suggests models based on data properties (e.g., seasonality).	KPI engine; forecasting; script sandbox
UR_ANA_06	Tools to check compliance of trends with specific regulations.	Semantic Policy Overlay: Module extracting targets from PDFs to overlay "Target vs Actual" on charts.	KPI engine; forecasting; script sandbox
UR_VIS_01	Maps for visualisation of regional disparities.	Geospatial Analytics Layer: Interactive GIS component for choropleth and heat maps (NUTS-2/3).	Web-GIS; dashboard canvas; charting
UR_VIS_06	Access to a dedicated observatory for presenting data on a dynamic map.	Interactive Web-GIS Engine: Leaflet/Mapbox-based observatory allowing users to plot indicators across administrative boundaries with full interactivity.	Web-GIS; dashboard canvas; charting
UR_VIS_07	Visualize "What-If" policy simulations directly on the map.	Simulation Overlay Layer: Dynamic rendering mode connected to Simulation Engine (UR_ANA_03) to recolor regions based on scenarios.	Web-GIS; dashboard canvas; charting
UR_VIS_08	Create, aggregate, and reuse custom map layers.	Layer Manager & Aggregator: Toolset to toggle visibility, adjust opacity, and aggregate data from regions into custom zones.	Web-GIS; dashboard canvas; charting
UR_VIS_09	Save specific map setup as a "Presentation" to share.	Interactive Map Snapshot: Ability to save map state (coordinates, layers, filters) as a named "View" for sharing.	Web-GIS; dashboard canvas; charting
UR_VIS_02	Line charts for trends and forecasting visualisation.	Time-Series Widget: Charting component optimized for historical lines and forecast intervals.	Web-GIS; dashboard canvas; charting
UR_VIS_03	Extended Charts (Bar, Histogram, Scatter, Boxplot).	Advanced Charting Library: Suite supporting distribution, correlation, and variance analysis.	Web-GIS; dashboard canvas; charting
UR_VIS_04	Interactive Dashboards with filtering and drill-down.	Custom Dashboard Canvas: Drag-and-drop builder with global filtering and drill-down capabilities.	Web-GIS; dashboard canvas; charting
UR_VIS_05	Export capabilities (PDF, Word, PNG, SVG).	High-Fidelity Reporting Service: Renderer for converting dashboards to static PDF/SVG or editable Word docs.	Web-GIS; dashboard canvas; charting

Funded by the European Union

UR ID	Need	Mapped feature	Component
UR_REP_01	Export the map data (for GIS tools) and map image (for reports).	Dual-Stream Map Export: Functionality to download visual map (PNG/PDF) and spatial data (CSV/GeoJSON).	Reporting service; dual-stream export
UR_SEC_01	Explanation of AI models and predictions.	Model Interpretability Layer: SHAP/LIME integration for feature importance charts and justifications.	PII masking; XAI; fairness auditor
UR_SEC_02	Data Anonymisation & Governance.	PII Masking & Governance Module: Filter to detect/hash PII and enforce retention policies.	PII masking; XAI; fairness auditor
UR_SEC_03	Bias Mitigation for protected attributes	Algorithmic Fairness Auditor: Diagnostic tool to stress-test models for bias against protected groups.	PII masking; XAI; fairness auditor
UR_AI_01	Conversational Data Analysis	Generative Data Interface: LLM engine translating natural language to SQL/Python for data retrieval.	Generative interface (text-to-SQL)
UR_AI_02	Persona Generator & Simulator for pilot testing.	Synthetic Persona Simulator: Configurable interface to simulate survey responses using LLM personas.	Persona & population synthesis
UR_AI_03	Agentic Paper Analysis (RAG) for summaries and Q&A.	Research RAG System: Autonomous agents for indexing PDFs, summarizing corpora, and citation-based Q&A.	RAG / agents; topic & speech models
UR_AI_04	The system must understand specific economic/policy terminology.	Fine-Tuned Policy Adapter: LoRA model fine-tuned on EU/OECD corpora for correct terminology/tone.	Drafting engine; citation agent; retrieval
UR_AI_05	Generate full policy drafts from outlines.	Structured Drafting Engine: Workflow expanding bullet points into structured drafts (Problem->Options).	Drafting engine; citation agent; retrieval
UR_AI_06	Reference existing laws/reports during generation.	Contextual Retrieval Pipeline: Retrieval of relevant chunks from user library to seed the LLM context.	Drafting engine; citation agent; retrieval
UR_AI_07	Claims must be linked to specific source documents.	Automated Citation Agent: Module verifying claims and inserting clickable footnotes to sources.	Drafting engine; citation agent; retrieval
UR_AI_08	Summarize long policy papers into Executive Summaries.	Long-Form Summarizer: Map-Reduce algorithm to digest large PDFs into concise summaries.	Drafting engine; citation agent; retrieval
UR_AI_09	Ability to rewrite text for different audiences.	Adaptive Tone Controller: UI to adjust system prompt complexity (Formal vs. Simplified).	Drafting engine; citation agent; retrieval
UR_AI_10	Outputs must respect specific document formatting.	Document Rendering Engine: Service populating strict .docx/LaTeX templates with generated content.	Drafting engine; citation agent; retrieval
UR_AI_11	Ask Natural Language questions across mixed sources (Data-lake + Local + Temp) simultaneously.	Hybrid Router Agent: Orchestrator that parses questions, determines source (DB/File), and synthesizes a combined answer.	RAG / agents; topic & speech models
UR_AI_12	Users need to convert audio recordings (interviews) into text for analysis.	ASR & Diarization Engine: Integration of an Automatic Speech Recognition (ASR) model (e.g., Whisper) that transcribes audio with timestamps and distinguishes between different speakers.	RAG / agents; topic & speech models

Funded by the European Union

UR ID	Need	Mapped feature	Component
UR_AI_13	Users need to automatically identify recurring topics/themes across interview/chat logs.	Topic Modeling Engine: An NLP module (BERTopic/LDA) to cluster text segments into thematic groups and visualize frequency.	RAG / agents; topic & speech models
UR_AI_14	Users need to gauge the mood and specific emotions in stakeholder interviews.	Emotion Detection Classifier: Classification layer that scans text to score sentiment polarity and categorize emotional states.	RAG / agents; topic & speech models
UR_COL_01	Ability to share projects and collaborate with teams.	Collaborative Workspace: Real-time environment for shared views, editing, and threaded commenting.	Shared workspace & view generator
UR_COL_02	Share specific map presentations with others.	Shared View Generator: Generate persistent URLs or read-only tokens to share live map states with non-users or team members.	Shared workspace & view generator
UR_AI_15	Users need to detect ambiguous phrasing, leading questions, and logical errors in questionnaires before launch.	Survey QA Agent: An NLP diagnostic tool that parses the questionnaire file (JSON/Word), flagging "double-barreled" questions, cognitive biases, and broken skip-logic paths.	Persona & population synthesis
UR_AI_16	Users need to simulate large-scale survey results using stratified sampling to match specific demographic distributions (census data).	Synthetic Population Generator: A sampling engine where users define quotas (e.g., Age, Gender). The system generates N distinct LLM agents strictly adhering to these quotas to answer surveys en masse.	Persona & population synthesis
UR_AI_17	Users need to query the entire Data-lake (structured stats + unstructured reports) via a single prompt to find supporting evidence.	Text-to-Lake Orchestrator: A ReAct (Reasoning + Acting) agent that breaks down a complex prompt, queries SQL/Vector DBs simultaneously, and synthesizes a grounded answer.	RAG / agents; topic & speech models
UR_AI_18	Users need a guided workflow that "drives" them to write a policy brief, automatically suggesting data points to back up their claims.	Interactive Brief Wizard: A "Socratic" AI assistant that guides the user through standard sections (Problem, Evidence, Options), actively scanning the Data-lake to suggest relevant evidence to insert.	Drafting engine; citation agent; retrieval

Requirements by area

Grouping the requirements by area shows how they distribute across the architecture:

- **Platform foundations (UR_GEN).** Single sign-on and RBAC, user profiles, audit logging, accessibility and caching. These are served by the gateway and the auth service and underpin every other feature.
- **Workspace management (UR_WRK).** Project setup, role-based layouts, context persistence and in-app guidance. These shape the PROGRESSUS-built frontend and its state handling.
- **Data ingestion and quality (UR_DAT).** Connectors for official and unofficial sources, multi-format parsing, table extraction, cleaning, imputation and lineage. These map to the ingestion jobs and the Bronze and Silver tiers.
- **Analytics and visualisation (UR_ANA, UR_VIS, UR_REP).** KPI calculation, forecasting, the interactive Web-GIS, dashboards, charting and export. These are the analytical engine and the visualisation components, drawing on the Gold tier.
- **Trust and security (UR_SEC).** Explainability, GDPR handling and bias mitigation, implemented as cross-cutting services applied to model output.

Funded by
the European Union

- **AI capabilities (UR_AI).** Conversational analysis, retrieval-augmented document work, policy drafting with citations, transcription and thematic analysis. These run as MCP tools and agents behind the gateway, using the local model and the vector index.
- **Collaboration (UR_COL).** Project sharing and shareable map states, provided by the collaborative workspace and view generator.

3.3 Functional Requirements Specification

The functional specification translates the requirements above into the concrete behaviour the system must exhibit. In ISABEL this means describing how data moves through the pipeline, how a user request is served, and what each tool is expected to do. Because the platform is organised around the gateway, most functional requirements take the same shape: a user action arrives at OpenWebUI, which authenticates it, calls one or more MCP tools or the local model, and returns a result. The detailed behaviour of the data pipeline, the request flow and the individual services is specified in Part 2, where each component is described alongside the requirements it satisfies.

3.4 Non-Functional Requirements

Non-functional requirements describe how well the system behaves rather than what it does. For ISABEL the important ones are:

- **Performance.** Dashboards and queries should feel responsive; heavy work is pushed off the request path into background jobs so the interface stays quick.
- **Security and privacy.** Access is controlled with Keycloak and RBAC, traffic is encrypted, personal data is masked, and the local model keeps sensitive data off external services.
- **Reliability and maintainability.** Services are loosely coupled and tested, so a component can be updated or replaced without taking the platform down.
- **Observability.** Metrics, logs and traces — including GPU and pipeline telemetry — are collected so problems can be found and fixed quickly.
- **Interoperability.** Open formats and a documented API let ISABEL exchange data with existing European systems and aggregate local trends into a wider picture.

Funded by
the European Union

4. Architecture Vision

4.1 Purpose

The architecture exists for one main purpose: design a system that let the users (researchers and policy-makers) work on regional labour markets, affected by the twin transition (digital and green) quickly and with confidence in the result. Everything in the design serves that aim - a frontend that fits how each user works in the everyday line of work, a gateway that keeps access and logging in one place, tools that do the analytical work, and a data-lake that keeps the inputs and outputs organised and traceable.

4.2 Scope and Boundaries

ISABEL mainly integrates with existing data sources rather than replacing them. It reads from national registries, statistical portals and document archives, harmonises what it finds, and serves it back through its own interface. Additionally, it gives the users the ability to create custom, personalized datasets, that better serves the task to be performed each time, also consolidating or creating evidence ready for policy making. Within that boundary, the platform is responsible for ingestion, storage, analysis, the user interface, and the security that surrounds them.

The platform also sets a clear internal boundary: the frontend is a client of OpenWebUI and nothing more. It cannot reach the tools, the data-lake or the model except through the gateway. This keeps the attack surface small and means that access rules and audit logging only have to be enforced in one place.

4.3 Design Principles and Constraints

The design follows a small set of principles, most of which fall out of the data-sovereignty requirement:

- **Keep data local.** Inference and storage mainly run on CERTH hardware; no data is sent to commercial AI services.
- **Mediate everything through the gateway.** OpenWebUI is the only entry point to tools, data and the model.
- **Keep services loosely coupled.** Tools and modules can be developed, deployed and replaced independently.
- **Make results checkable.** Outputs carry citations, and experts can review and approve them before publication.

The main constraints are equally concrete: the platform must integrate with legacy databases rather than replace them; updates must not interrupt access for users who depend on the platform; and cross-border data must be handled in line with European data-protection rules, which is why personal data is masked and aggregated before it informs wider analysis.

Funded by
the European Union

5. Architecture Overview and Data Flows

5.1 High-Level Architecture

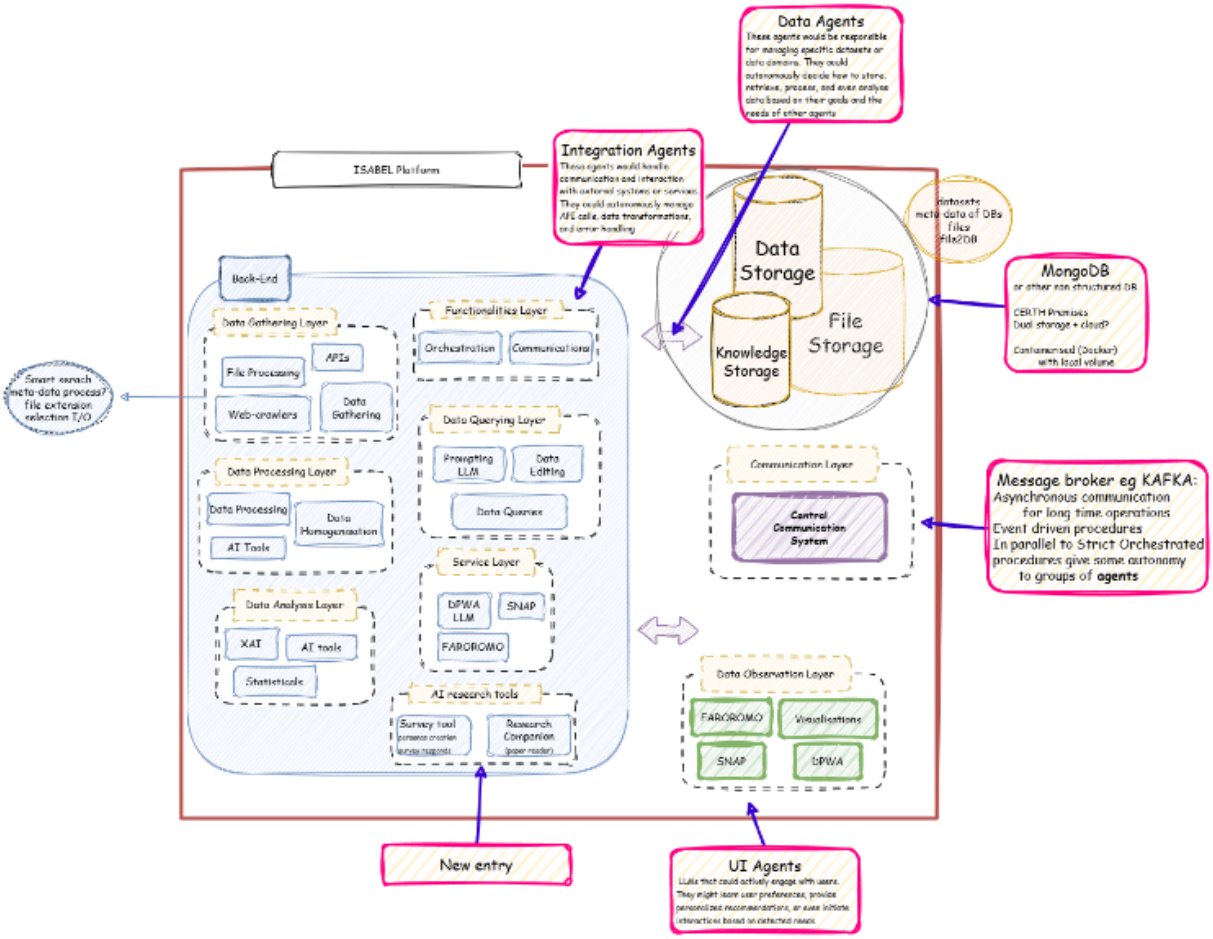
Figure 1 in Section 1.2 introduced the four layers. To restate them here: the custom frontend is what users see; OpenWebUI is the middleware that authenticates requests and brokers calls; the ISABEL tool and agent server does the analytical work; and the Medallion data-lake stores everything. The value of this arrangement is that each layer has one clear job, and the boundaries between them are simple — the frontend speaks REST to the gateway, the gateway speaks MCP and OpenAPI to the tools, and the tools speak SQL and structured queries to the lake.

5.2 Layers and Responsibilities

- **Presentation layer.** The frontend built by the ISABEL team (Vue components, D3.js and Chart.js for visuals) that provides role-specific workspaces, charts, maps and dashboards. It consumes OpenWebUI's backend API and the Data-lake API.
- **Gateway layer.** OpenWebUI, which manages accounts and sessions, exposes the OpenAPI/Swagger interface, hosts the MCP client, runs internal RAG [Lewis et al., 2020 — RAG] and filter pipelines, and brokers calls to the model.
- **Service layer.** The ISABEL tool server (Python/FastAPI) and its MCP tools, resources and prompts, together with the orchestrator and the SNAP and FARAROMO agents.
- **Data layer.** The Medallion approach data-lake - Bronze, Silver and Gold — using MinIO as the underlying object store, DuckDB for analytics, and two PostgreSQL instances for app and identity data. Semantic vector search (ChromaDB) is planned for a future release.

Underneath this four-layer view sits a more detailed functional decomposition that the project developed during design. It breaks the work into data-gathering, processing, analysis, querying, observation and communication functions, and shows how the three main analytical modules — SNAP, FARAROMO and DPWA — each move data from a repository through processing and on to their own views. Figure 2 captures this concept and the per-module flows.

Funded by
the European Union



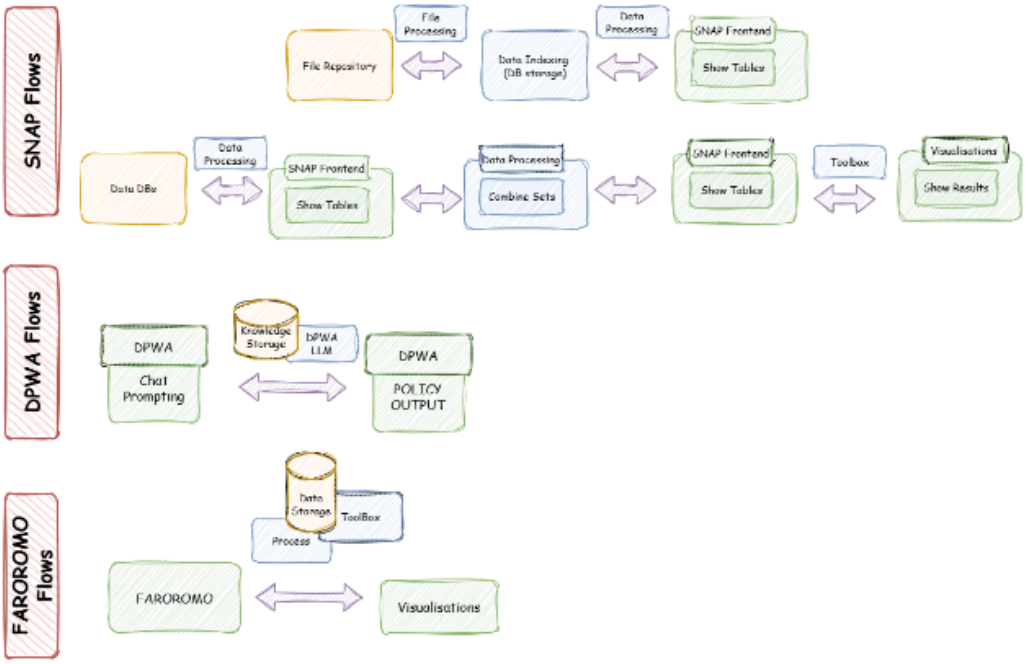


Figure 2: Platform concept: the functional layers, the central communication and storage components, and the SNAP, DPWA and FARO/ROMO module flows.

Funded by
the European Union

5.3 Major Components

- **OpenWebUI (middleware).** Provides the capabilities the project would otherwise have to build: user management, a documented API, a language-model client, and a host for MCP tools. It is the hub through which the frontend reaches everything else.
- **ISABEL tool server.** A FastAPI service that implements the platform's tools and also exposes them over MCP if needed. It is where ingestion, data queries, analytics and the module logic live.
- **Agents.** An orchestrator agent breaks a complex request into steps; a specialised FARAROMO agent handles forecasting and Web-GIS work; and additional tool-calling agents retrieve and synthesise data from the SNAP data-lake infrastructure.
- **Modules (SNAP, FARAROMO, DPWA).** The platform's three primary analytical products: SNAP (Smart oNe-stop digitAl data shoP) is the data-lake and data-access infrastructure; FARAROMO provides forecasting and Web-GIS; and DPWA (Digital Policy Writer Assistant) helps policy-makers draft evidence-based policy briefs from Gold-layer content.
- **Local model.** Mistral-Small-24B-Instruct-2501 served with vLLM across both GPUs (tensor parallelism, 32k context, tool-calling enabled), exposed through an OpenAI-compatible API. No data leaves the premises.

5.4 Technology Stack

Table 8 lists the main technologies and why each was chosen. The selection reflects the project's priorities: keep inference local, lean on OpenWebUI for infrastructure, and use lightweight, file-based data engines that run well on a single powerful machine.

Table 8: Main technology choices

Area	Technology	Why
Frontend	Framework: Nuxt 4 (SSR Mode) Language: TypeScript (Strict) UI Framework: Nuxt UI v4 Styling: Tailwind CSS v4 Icons: Lucide	Built and maintained by the PROGRESSUS team; consumes OpenWebUI backend API and Data-lake API.
Middleware / gateway	OpenWebUI + Nginx	OpenWebUI provides user management, OpenAPI, model client, MCP host, RAG pipelines. Nginx handles TLS termination, path-based routing, and acts as the single external entry point.
Tools & services	Python, FastAPI, MCP (MCPO proxy) + Data-lake API	ISABEL tool server and custom Data-lake API (FastAPI); tools exposed uniformly via MCP.
Language model	Mistral-Small-24B-Instruct-2501 via vLLM	24B-parameter instruction model; tensor parallelism across both GPUs; 32k context; tool-calling enabled. No data leaves the premises.
Object storage	MinIO (S3-compatible)	On-premises S3-compatible store; holds all Bronze, Silver and Gold Medallion files. No external cloud dependency.
Analytics store	DuckDB	Fast OLAP queries over local Parquet files, no separate server. Exposed to tools via DuckDB-MCP.
Vector store	ChromaDB / FAISS (planned)	Semantic vector search. Not yet deployed; current release uses keyword search over file names and metadata.

Area	Technology	Why
Document processing	Docling (CPU, GPU-accelerated planned)	Extracts structured Markdown from PDFs, DOCX, XLSX and other formats. Currently CPU-bound; GPU acceleration is a planned enhancement.
Identity	Keycloak (OIDC / OAuth 2.0), JWT	Centralised SSO and access control. OpenWebUI validates tokens; first-login auto-provisioning by email.
Databases	PostgreSQL 16 (×2), Redis / Valkey	Two isolated PostgreSQL instances: OpenWebUI app data and Keycloak identity data. Redis / Valkey for ephemeral WebSocket sessions only.
Pipelines	GitLab CI/CD (self-hosted)	Versioned, auditable background jobs without a Kubernetes cluster.
Observability	Portainer	Unified container management and built-in monitoring.

5.5 The Data Pipeline

Data flows through the data-lake in three tiers, all backed by MinIO — an S3-compatible object store that runs entirely on the CERTH node. The Bronze tier holds raw inputs exactly as they arrive — spreadsheets, JSON, exported tables and PDF reports — so there is always an immutable record of the original. The Silver tier is where the work happens: Docling extracts content from each file and produces normalised Markdown and Parquet representations. The Gold tier holds curated, validated knowledge assets and the DuckDB tables the platform queries for analytics. Semantic vector search (ChromaDB) is planned but not yet active; the current release uses keyword matching across file names and metadata.

Two points are worth stressing. Promotion from Silver to Gold is not automatic for analytical output: an expert reviews and approves it first, which is how the platform keeps its results trustworthy. And the heavy work — bulk ingestion, OCR, re-indexing — runs as scheduled GitLab jobs, so it never competes with live user requests. The data flow through the tiers is shown, alongside the other detailed views, in Figure 3 (Section 5.7).

5.6 Request and Tool-Call Flow

A typical request shows how the layers cooperate. The user asks a question in the frontend; the request reaches OpenWebUI, which checks the user's token and role; the gateway then calls the appropriate MCP tool on the ISABEL server; the tool queries DuckDB and the Silver/Gold layers for data and context; that context goes back to the gateway, which sends it together with the prompt to the local model; and the model's answer, with citations, is returned to the frontend to display.

Synchronous flows like this one power the interactive parts of the platform. Anything slow or bulky is handled asynchronously instead: ingestion, transformation and indexing run as background jobs, which keeps the interface responsive even while large datasets are being processed. The orchestration and tool-call routing are visible in the lower panels of Figure 3.

5.7 Detailed Architecture Views

Figure 3 brings together the detailed views the project produced during design. Read clockwise, it shows the OpenWebUI gateway stack with its user-interface, security and MCP layers; the authentication path through Keycloak and the OpenWebUI token-validation layer; the Medallion data flow from the Bronze landing tier through Silver refinement to the Gold consumption tier;

and the component layout that places the React/Next.js user layer, the OpenWebUI and vLLM middleware, the MCP server layer and the SNAP data layers in relation to one another. It is denser than the other figures because it consolidates several perspectives on one canvas; it is included here as the authoritative design reference, and the high-resolution image can be magnified for detail.

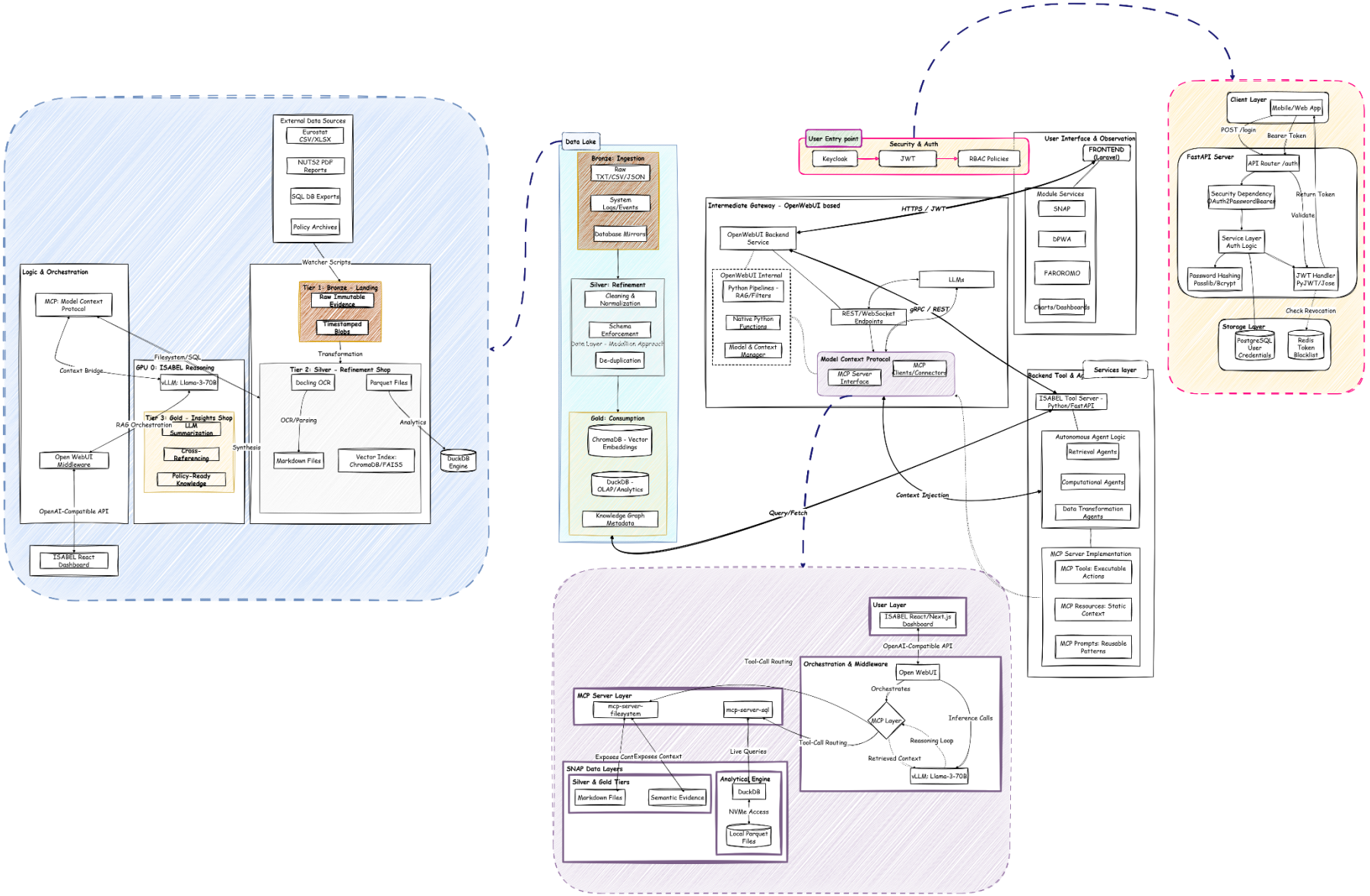


Figure 3: Detailed reference views: the OpenWebUI gateway stack, the authentication flow, the Medallion data flow, and the component layout.

6. Presentation Layer

6.1 Frontend Architecture

The frontend is built and maintained by the ISABEL team. It consumes two platform APIs: OpenWebUI's backend REST/WebSocket API for model interactions, user sessions and knowledge-base queries, and the custom Data-lake API for browsing, searching and extracting Medallion-layer content. Its technology stack (Vue components, D3.js and Chart.js for visuals, Axios for HTTP) is owned by PROGRESSUS; the CERTH backend is technology-agnostic from the frontend's perspective, exposing only documented API endpoints.

6.2 Module Interfaces

The interface adapts to the user. A researcher sees data grids and analytical tools; a policy-maker sees dashboards and drafting aids. The analytical modules each have their own components:

- **SNAP.** The Smart oNe-stop digitAl data shoP — the platform's unified data-access and storage infrastructure. It stores, sorts and homogenises datasets from the project's data-collection activities and makes them available to researchers and tools through the Data-lake API.
- **FARAROMO.** Forecasting dashboards and the interactive Web-GIS, including scenario inputs and comparative trend views.
- **DPWA.** The Digital Policy Writer Assistant. It draws on Gold-layer knowledge assets to help policy-makers draft evidence-based policy briefs and recommendations grounded in the verified datasets the platform provides.

6.3 Hosting

The frontend can be hosted for high availability and quick delivery to users across Europe, while the parts that touch sensitive data — the gateway, the tools, the model and the lake — run on CERTH's hardware. The FARAROMO component is hosted and maintained by the University of Western Macedonia because of its ArcGIS dependencies. The pieces stay connected through the gateway's API.

6.4 Performance

Responsiveness comes mainly from keeping slow work off the request path and from caching. Background jobs handle ingestion and indexing; the gateway's internal pipelines and the Gold-tier stores are tuned for fast reads; and telemetry highlights any slow query or endpoint so it can be addressed before users notice.

Funded by
the European Union

7. Gateway, Services and Data

This section looks more closely at the layers behind the frontend: the OpenWebUI gateway, the ISABEL tool server and its agents, the security model, the data validation process and the monitoring that keeps it all healthy.

7.1 OpenWebUI as Middleware

OpenWebUI is the centre of the architecture. It manages users and sessions, exposes a documented OpenAPI/Swagger interface, and acts as the client for the language model [Open WebUI, 2024 — OpenWebUI]. It also hosts the MCP client and an MCPO proxy that presents MCP tools as OpenAPI endpoints, and it runs internal Python pipelines for retrieval and filtering. In short, it is the layer that turns ISABEL's collection of tools and a raw model into a coherent service the frontend can call.

7.2 The ISABEL Tool Server and MCP

ISABEL's own capabilities live in a FastAPI tool server and are exposed through the Model Context Protocol. MCP organises them into three kinds of building block: tools, which are executable actions such as querying data or running an analysis; resources, which provide static context the model can draw on; and prompts, which are reusable patterns for common tasks. Because these are reached only through the gateway, the team can add or change tools without touching the frontend, and access to every tool is governed by the same authentication and role checks.

7.3 Agentic Logic

More involved requests are handled by agents. An orchestrator agent interprets the request and decides which tools to call and in what order. Specialised agents handle particular domains: the FARO ROMO agent for forecasting and Web-GIS work, and agents that interface with the SNAP data-lake infrastructure. This division keeps each agent's job manageable and lets the platform combine them — for example, retrieving data with one tool, analysing it with another, and asking the model to summarise the result.

7.4 Authentication and Authorisation

Authentication is managed by Keycloak, which implements OAuth 2.0 and OpenID Connect [Sakimura et al., 2014 — OpenID Connect]. When a user logs in, Keycloak issues a signed identity token; OpenWebUI validates that token before granting access to any platform resource. OpenWebUI also handles automatic account provisioning on first login, matching users by email. Role assignments and access-control policies are managed centrally in Keycloak, keeping authorisation logic cleanly separate from the application layer. User identity data is stored in a dedicated Keycloak PostgreSQL instance, isolated from the application database. Redis / Valkey manages ephemeral WebSocket session state; it is not a persistent store and holds no security-sensitive data. The authentication flow is shown in the right-hand panel of Figure 3.

All external traffic enters through the Nginx reverse proxy, which handles TLS termination. Nginx enforces HTTPS on port 443 (redirecting plain HTTP) and routes requests to the correct upstream

service by path prefix. Backend services communicate over the platform's private Docker network and do not require individual certificate infrastructure.

7.5 Data Validation and the Medallion Lake

The data-lake was described in Section 5.5; what matters here is how its quality is guaranteed. Automated cleaning and storing in the Silver tier is the first step, but it is not the last. For analytical output, a domain expert reviews what the model produced — for instance a skill-gap synthesis — corrects anything that is wrong, and approves it. Only then is the result promoted to the Gold tier and made available to users as ISABEL knowledge. The corrections are kept, both as an audit trail and as material for improving the model later. The full data-lake specification is documented separately in Deliverable D5.3.

7.6 Logging and Monitoring

Monitoring covers both the background jobs and the gateway. The platform tracks pipeline success rates, GPU memory use, model throughput in tokens per second, and API latency. This telemetry does double duty: it surfaces problems quickly, and it provides the evidence the project needs to show the platform scales. The Living Labs are central to this — real use by citizens, researchers and policy-makers generates the load that tells the team where to tune the model and the pipelines.

Funded by
the European Union

8. Deployment and Infrastructure

ISABEL runs on a single high-performance workstation at CERTH rather than on a cluster. Concentrating everything on one well-specified machine removes network hops between components and keeps all data and inference inside one controlled environment. The node carries two NVIDIA RTX 5090 GPUs (64 GB of VRAM in total) and a fast NVMe storage array, on a Linux base. Figure 4 shows how the work is divided.

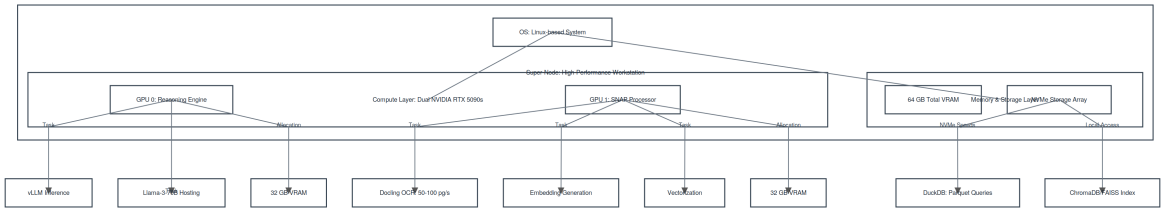


Figure 4: Deployment: the on-premises super-node — dual RTX 5090 GPUs working in tensor-parallel mode for LLM inference, with MinIO-backed Medallion storage on a local NVMe array.

The two GPUs work together in tensor-parallel mode to serve Mistral-Small-24B-Instruct-2501 via vLLM, which requires more VRAM than a single device can provide. Docling currently runs on the CPU; GPU-accelerated document processing is planned as document volumes grow. The NVMe array holds the MinIO object storage (Bronze / Silver / Gold layers), the DuckDB Parquet tables and the two PostgreSQL databases, all reachable at local speeds. This is a deliberate departure from a Kubernetes-style deployment: for a research platform of this size, a single tuned node is simpler to operate and keeps the data-sovereignty guarantees easy to reason about. Should demand grow, the same design can be replicated or extended onto additional nodes.

Funded by
the European Union

9. Conclusion

This deliverable has described how ISABEL is put together and why. The platform is organised around OpenWebUI as a single gateway, with a custom frontend in front of it, ISABEL-built tools and agents behind it, and a Medallion data-lake underneath. Running Mistral-Small-24B-Instruct-2501 on local hardware lets the platform keep sensitive labour-market data inside the European environment, and routing every request through the gateway keeps security and logging in one place.

In Part 1, and most specifically in §3, recorded the 68 user requirements, prioritised them, and mapped each to a feature and a component. Additionally, in Part 2 (§4, §5, §0, §7, §8) showed how those features are realised across the layers, through the data pipeline and request flow, and on the hardware the system runs on. Taken together, the document gives the development and validation teams a clear and practical basis for the work ahead, and a design that can grow as the project's needs do.

References

1. vLLM — local LLM inference engine, Kwon, W., Li, Z., Zhuang, S., Sheng, Y., Zheng, L., Yu, C. H., ... & Stoica, I. (2023). Efficient Memory Management for Large Language Model Serving with PagedAttention. In Proceedings of the 29th ACM Symposium on Operating Systems Principles (SOSP '23). ACM. https://doi.org/10.1145/3600006.3613165
2. Mistral language model, Jiang, A. Q., Sablayrolles, A., Mensch, A., Bamford, C., Chaplot, D. S., Casas, D. D. L., ... & Sayed, W. E. (2023). Mistral 7B. arXiv preprint arXiv:2310.06825. https://arxiv.org/abs/2310.06825
3. Retrieval-Augmented Generation (RAG), Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., ... & Kiela, D. (2020). Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. Advances in Neural Information Processing Systems, 33, 9459–9474. https://arxiv.org/abs/2005.11401
4. Model Context Protocol (MCP), Anthropic. (2024). Model Context Protocol: Open Standard for Connecting AI Assistants to Data Sources. https://modelcontextprotocol.io/
5. Medallion / lakehouse architecture, Armbrust, M., Ghodsi, A., Xin, R., & Zaharia, M. (2021). Lakehouse: A New Generation of Open Platforms that Unify Data Warehousing and Advanced Analytics. In Proceedings of CIDR 2021. https://www.cidrdb.org/cidr2021/papers/cidr2021_paper17.pdf
6. DuckDB — in-process analytical engine, Raasveldt, M., & Mühleisen, H. (2019). DuckDB: An Embeddable Analytical Database. In Proceedings of the 2019 ACM SIGMOD International Conference on Management of Data (pp. 1981–1984). ACM. https://doi.org/10.1145/3299869.3320212
7. Keycloak / OpenID Connect, Sakimura, N., Bradley, J., Jones, M., de Medeiros, B., & Mortimore, C. (2014). OpenID Connect Core 1.0. OpenID Foundation. https://openid.net/specs/openid-connect-core-1_0.html
8. Human-in-the-loop AI validation, Mosqueira-Rey, E., Hernández-Pereira, E., Alonso-Ríos, D., Bobes-Bascarán, J., & Fernández-Leal, Á. (2023). Human-in-the-loop machine learning: A state of the art. Artificial Intelligence Review, 56, 3005–3054. https://doi.org/10.1007/s10462-022-10246-w
9. GDPR and AI data sovereignty, European Parliament and of the Council. (2016). Regulation (EU) 2016/679 — General Data Protection Regulation. Official Journal of the European Union, L 119, 1–88. https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:32016R0679
10. OpenWebUI, Open WebUI. (2024). Open WebUI: User-friendly WebUI for LLMs. GitHub.

Funded by the European Union

Appendices

Glossary

Term	Meaning
OpenWebUI	The middleware and gateway that manages users, exposes the API and brokers calls to tools and the model.
MCP (Model Context Protocol)	A standard way to expose tools, resources and prompts to a model-driven system.
MCPO proxy	A component that presents MCP tools as OpenAPI endpoints.
Medallion lake	A three-tier data store: Bronze (raw), Silver (refined) and Gold (analysis-ready).
vLLM	A high-throughput server for running large language models locally.
Mistral-Small-24B-Instruct-2501	The 24B-parameter instruction-following model served locally by vLLM across both GPUs (tensor parallelism). Used for direct user interactions, tool-calling and Gold-layer knowledge synthesis.
DuckDB	An in-process analytical database used for fast queries over Parquet files.
MinIO	S3-compatible object store hosting all Bronze, Silver and Gold Medallion-layer files on the CERTH node. No external cloud dependency.
ChromaDB / FAISS (planned)	Vector stores for embeddings and semantic search. Not yet deployed in the current release; keyword-based search is in use.
Docling	An OCR and document-parsing tool that turns PDFs into clean text.
Keycloak	An identity provider supplying single sign-on, OIDC and tokens.
RBAC	Role-based access control: permissions tied to a user's role.
HITL	Human-in-the-loop: expert review and approval of automated output.
RAG	Retrieval-augmented generation: grounding model answers in retrieved data.
SNAP / FAROROMO / DPWA	ISABEL's three analytical products. SNAP (Smart oNe-stop digitAl data shoP) is the data-lake and data-access infrastructure. FAROROMO provides forecasting and Web-GIS. DPWA (Digital Policy Writer Assistant) helps policy-makers produce evidence-based policy briefs.

Related Deliverables

The detailed data-lake schemas and indexing rules are specified in Deliverable D5.3. This document (D2.5) should be read as the architectural reference that those and later technical deliverables build on.

Funded by the European Union