UNIVERSIDADE
LUSÓFONA

Researcher's Diary

Trabalho Final de curso

Relatório Intercalar 2º Semestre

Guilherme Frantz, a22204098

Engenharia Informática

Orientador: Luis Campos

Departamento de Engenharia Informática da Universidade Lusófona

Centro Universitário de Lisboa

Abril de 2026

Direitos de cópia

Acknowledgement

For my family who has supported me unconditionally.

Resumo

A gestão da produção científica numa instituição de ensino ou investigação é um processo fragmentado e repetitivo. Investigadores perdem tempo significativo a registar e a reformatar os mesmos dados para diferentes agências e relatórios institucionais. Simultaneamente, os responsáveis por departamentos e instituições necessitam de agregar manualmente esses mesmos dados para produzir relatórios organizacionais, sem dispor de ferramentas adequadas para o efeito.

Este Trabalho Final de Curso (TFC) propõe o desenvolvimento do Researcher's Diary: uma plataforma web de gestão hierárquica de produção científica, construída com um backend em Python/FastAPI e um frontend em React/Next.js, orquestrada com Docker Compose. O sistema implementa uma hierarquia institucional complementada por um sistema de visibilidade e acesso baseado em funções. Responsáveis de uma instituição ou departamento podem consultar e exportar relatórios agregados com base nos registos partilhados, reduzindo significativamente o esforço administrativo de reporte científico.

A inovação do projeto reside na combinação de uma arquitetura API-First com um modelo de dados hierárquico e especializado, que transforma dados individuais dispersos numa fonte única e estruturada, reutilizável para múltiplos formatos de saída.

Palavras chave: Gestão de Produção Científica, Hierarquia Institucional, FastAPI, React, Docker, API-First.

Abstract

Managing scientific production within a research institution is a fragmented and repetitive process. Researchers spend significant time recording and reformatting the same data for different funding agencies and institutional reports. At the same time, department and institution heads must manually aggregate this data to produce organizational reports, without adequate tooling to support them.

This Final Year Project (FYP) presents the development of Researcher's Diary: a hierarchical scientific production management web platform, built with a Python/FastAPI backend and a React/Next.js frontend, orchestrated with Docker Compose. The system implements an institutional hierarchy complemented by a visibility and access system based on roles. Department and institution management can browse and export aggregated reports based on shared records, significantly reducing the administrative overhead of scientific reporting.

The project's core innovation lies in combining an API-First architecture with a hierarchical, specialized data model that transforms scattered individual data into a single structured source, reusable across multiple output formats.

Key Words: Scientific Production Management, Institutional Hierarchy, FastAPI, React, Docker, API-First.

Index

List of Figures

List of Tables

List of Abbreviations

API	Application Programming Interface
CLI	Command Line Interface
CRUD	Create, Read, Update, Delete
JWT	JSON Web Token
LEI	Licenciatura em Engenharia Informática
ORM	Object-Relational Mapping
REST	Representational State Transfer
SaaS	Software as a Service
RBAC	Role-Based Access Control

Introduction

Characterization of the problem

Scientific institutions operate under a constant reporting obligation. Funding agencies such as FCT[FCT] and the European Commission's Horizon Europe[HOREU] programme require detailed, formatted accounts of an institution's research output. Internally, department and institution management must compile this same information for strategic planning, accreditation processes, and annual activity reports.

At the individual level, researchers are forced to treat every grant application, progress report, or curriculum update as a new data-entry task. The same publication list is manually re-entered and re-formatted across platforms because each requires a distinct structure. This redundancy consumes time that could be spent on research, and introduces data inconsistency across versions.

At the institutional level, heads of departments face an aggregation problem. To produce a departmental or institutional report, they must collect data from each individual researcher and manually compile it into a unified document.

The Federal Demonstration Partnership (FDP) Faculty Workload Survey, conducted across over 13,000 principal investigators in 111 U.S. research institutions, found that researchers spend an average of 42% of their federally-funded research time on administrative tasks rather than conducting actual research[FDP12]. Of this, report preparation alone accounts for approximately 8% of total research time. The National Science Board's Task Force on Administrative Burdens identified report preparation, financial management, and effort reporting as the most burdensome categories[NSB14]. On the same note, a nationwide survey of Australian university academics confirmed that administrative demands associated with core research activities remain high, both in frequency and time intensity, to the detriment of actual academic work[ASHEIM25].

Existing tools fail to address the full scope of this problem. Reference managers such as Zotero[ZOTDEV] handle citation metadata effectively but are limited to publications and offer no institutional aggregation. Public profiles like ORCID[ORCID] serve as visibility platforms rather than private management tools, and do not support organizational hierarchies. General-purpose tools such as Notion[NTON] or spreadsheets offer flexibility but lack specialized academic data models, role-based access control, and structured export capabilities. None of these tools provide a unified platform where individual researchers manage their career data privately while institutional leaders access aggregated, visibility-controlled views for reporting.

Objectives

The primary objective of this project is to develop a hierarchical scientific production management platform that serves as a centralized source of structured career metadata for researchers, departments, and institutions. The specific objectives are:

- To design and implement a relational database model representing an institutional hierarchy with role-based access control and record visibility;
- To develop a REST API using FastAPI[FASTAPI] that enforces permission-scoped data access, allowing researchers to manage their own records while department and institution management access aggregated views;
- To implement full CRUD operations for all core data entities;
- To implement a visibility system that allows researchers to control which records are shared at the institutional level;
- To implement aggregated report generation endpoints that compile department-level and institution-level scientific production summaries;
- To implement a structured data export mechanism for individual researchers;
- To develop a frontend interface using React[REACT] and Next.js[NEXTJS] that reflects the hierarchical permission model with role-appropriate views;
- To ensure full system modularity and reproducibility using Docker Compose[DOCKER].

Document Structure

- In **Introduction**, the problem domain, objectives, and scope of the project are presented;
- In **Relevance and Viability**, the pertinence of the project and its technical, operational, and economic viability are analyzed;
- In **Comparative Analysis**, existing solutions, the innovation proposal, and business opportunities are discussed;
- In **Specification and Modeling**, functional and non-functional requirements are detailed alongside relevant data models;
- In **Developed Solution**, the implemented architecture, technologies, interface, and academic scope are described;
- In **Methodology and Planning**, the development methodology and project timeline are presented.

***Github*:** https://github.com/DEISI-ULHT-TFC-2025-26/TFC-DEISI2207-Researchers-Diary

Demonstration Video (*Youtube*): https://www.youtube.com/watch?v=E5EL9MsZ9a4

Relevance and Viability

Relevance

This project addresses a well-documented and universal challenge within the academic and scientific community: the administrative burden of managing and reporting scientific production data.

The problem impacts two distinct stakeholder groups. For individual researchers, the lack of a centralized management tool means that career data is scattered across disconnected platforms, each with its own format. This forces researchers to duplicate effort for every new report or application. The FDP Faculty Workload Survey found that over 13,000 principal investigators reported spending 42% of their research time on administrative requirements, with report preparation and post-award administration among the most time-consuming tasks[ROCK09]. The direct cost of this burden was estimated at $97 million in principal investigator salary support diverted from research to administration across the surveyed population alone.

For department and institution management, the problem is compounded by the need to aggregate individual data into organizational reports. Without a structured system, this process relies on manual collection which is both slow and error-prone.

The key relevance of Researcher's Diary lies in addressing both sides of this problem within a single platform. At the individual level, it provides a private workbench where researchers can register and organize their career metadata in a structured, reusable format. At the organizational level, it introduces a hierarchical visibility system that enables department and institution heads to access and export aggregated data without requiring manual collection from each researcher. This dual-purpose approach transforms the data management process from a repetitive, per-report task into a centralized, query-based operation.

Viability

The scope of this project presents robust viability across technical, operational, and economic dimensions.

Technical Viability

The proposed technology stack represents the modern industry standard for scalable, maintainable web systems. All components are mature, open-source, and well-documented. The system stores only text-based metadata rather than large files, which significantly reduces the complexity and cost of storage and bandwidth management.

The hierarchical permission system and visibility-scoped data access have been fully implemented and validated. FastAPI provides automatic data validation via Pydantic schemas and generates interactive API documentation (OpenAPI/Swagger UI), ensuring testability and maintainability. SQLModel bridges the ORM and validation layers with a single model definition, reducing code duplication. The modular containerized architecture allows for future incremental extensions without affecting the existing system.

Operational Viability

The application targets two user profiles: individual researchers managing their own data, and institutional leaders accessing aggregated views. Both profiles involve digitally-literate professionals in academic environments. The system is designed with a low learning curve: researchers interact with familiar CRUD forms, while heads access role-appropriate views that appear automatically based on their permission level.

For institutional adoption, the Docker Compose orchestration ensures that the entire stack can be deployed on any compatible server for a private and secure implementation.

Economic Viability

The entire stack is open-source, minimizing development and licensing costs. Operational expenses remain negligible because the application stores lightweight text metadata rather than heavy files, making low-tier cloud hosting or institutional self-hosting highly effective strategies. The Docker-based deployment model also reduces infrastructure setup costs for institutions that already use container orchestration.

Comparative Analysis

Existing Solutions

The existing academic software ecosystem is fragmented into multiple separate and disconnected layers.

Public Profiles (ORCID, ResearchGate) are designed for visibility and global interconnection[RGATE]. Their primary function is to serve as public showcases rather than private management tools. Data entry is manual and slow, and they do not support organizational hierarchies, role-based access, or institutional reporting.

Reference Management Tools (Zotero, Mendeley) are excellent for managing bibliographies[ZOTDEV]. However, their functionality is limited to publications. They do not support the management of other career aspects such as grant proposals, projects, teaching history, or administrative service. They offer no institutional aggregation capabilities.

General-Purpose Software (Excel, Notion) offers total flexibility but lacks academic-specific business logic. They cannot validate metadata against academic schemas, enforce visibility rules, generate role-scoped reports, or provide structured API-based exports without extensive manual configuration.

Institutional Research Information Systems (Pure, Converis) address the institutional aggregation problem but are enterprise-grade platforms with significant licensing costs and implementation complexity. They are typically deployed at large research universities and are not accessible to smaller institutions, individual departments, or independent researchers.

Benchmarking

Table 1 - Benchmarking

Feature	Researcher's Diary	Zotero	ORCID	Notion	Pure
Full Career Scope	X	-	X	X	X
Institutional Hierarchy	X	-	-	-	X
Role-Based Access Control	X	-	-	-	X
Visibility-Based Sharing	X	-	-	-	~
Global Discover	-	-	X	-	X
Specialized Metadata	X	X	X	-	X
API-First	X	X	X	-	~
Private Workbench	X	X	-	X	-
Open-source	X	X	-	-	-
Low Implementation Cost	X	X	X	X	-

Innovation Proposal

Researcher's Diary differentiates itself by combining features that are individually available in different tools but are not offered together in any existing solution.

- **Hierarchical Scientific Management:** The system models an institutional hierarchy, enabling both individual and organizational data management within a single platform. This bridges the gap between personal research tools and enterprise CRIS systems.
- **Visibility-Controlled Data Sharing:** Each researcher controls which records are visible to their organization. This preserves individual privacy while enabling institutional aggregation for reporting purposes.
- **Aggregated Report Generation:** Department and institution heads can generate structured reports that follow the specific formats required by principal organizations and agencies, and compile scientific production data across their respective units.
- **API-First Architecture:** All functionality is exposed via a RESTful API. This ensures the system can be used via a web browser, terminal scripts for advanced automation, or integrated with other institutional systems.
- **Unified Metadata Inventory:** It serves as a single structured source for all career metadata, independent of any specific output format.
- **Structured Data Export:** Researchers can select specific records and export them, enabling programmatic reuse in other tools or reporting pipelines.

Business Opportunities and Applicability

The dual-purpose nature of the platform opens multiple economic pathways.

An **institutional SaaS model** could offer institutions a hosted platform with per-member pricing, providing immediate value through aggregated reporting and reduced administrative overhead. Since the application stores lightweight metadata, hosting costs per institution remain minimal.

An **open-source self-hosted model** allows institutions with existing Docker infrastructure to deploy and maintain their own instance at near-zero cost. This approach is particularly attractive for publicly funded universities that prefer to keep research data on their own servers.

A **freemium individual model** could offer researchers a free personal tier for managing their own data, with premium features for institutional aggregation and advanced exports. The low storage requirements make a generous free tier economically viable.

Specification and Modeling

Key Functional Requirements

Observation: The complete flowchart diagrams for all remaining use cases are included in Annex A.

UC07 - Manage (CRUD) Core Records

This use case represents the foundational data management capability of the system.

Brief Description: Provides tools to Create, Read, Update, and Delete records across all four entity types: Publications, Projects, Proposals, and Experiences.

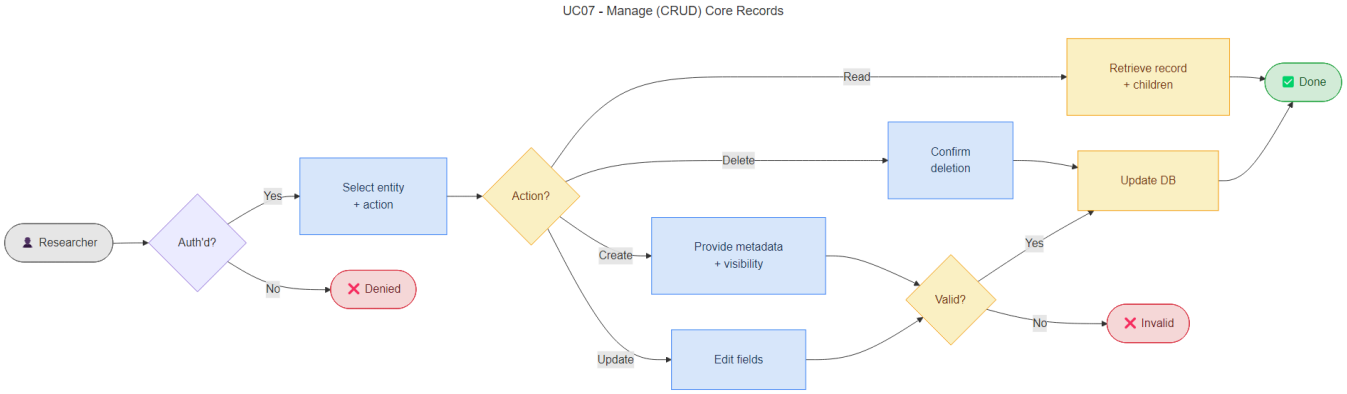
Actors: Researcher, System.

Pre-conditions: The researcher is authenticated. For Update and Delete actions, the researcher must be the owner of the record.

Acceptance Criteria: The system enforces entity-specific schema constraints. Visibility settings are applied correctly during creation and updates.

Post-conditions: The record is created, modified, or removed in the database.

Figure 1 - UC07



UC08 - Set Record Visibility

This use case is central to the hierarchical data sharing model.

Brief Description: Allows a researcher to change the visibility for any of their records between "private" (visible only to the owner) and "institution" (visible to authorized heads within their institution).

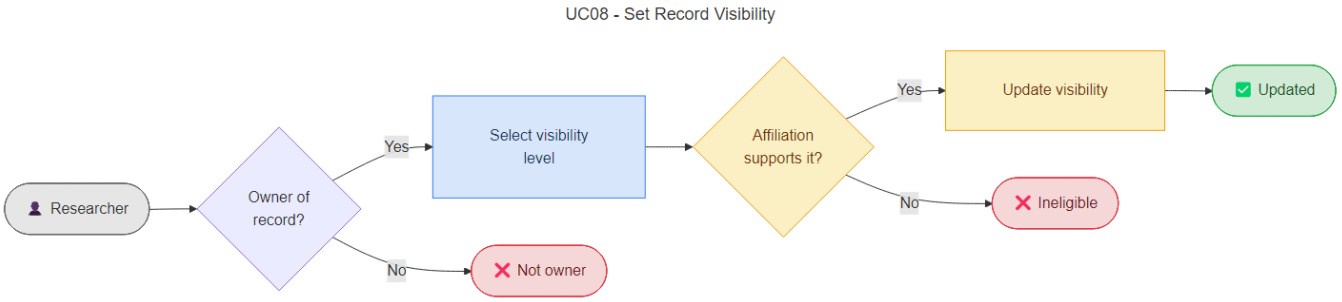
Actors: Researcher, System.

Pre-conditions: The researcher is authenticated and owns the record. For "institution" visibility, the researcher must belong to an institution.

Acceptance Criteria: Visibility settings are applied correctly.

Post-conditions: The record's visibility is updated and access is immediately scoped accordingly.

Figure 2 - UC08



UC09 - View Department Records

This use case demonstrates the institutional aggregation feature.

Brief Description: Allows a Department Head to browse records made visible by members of their department.

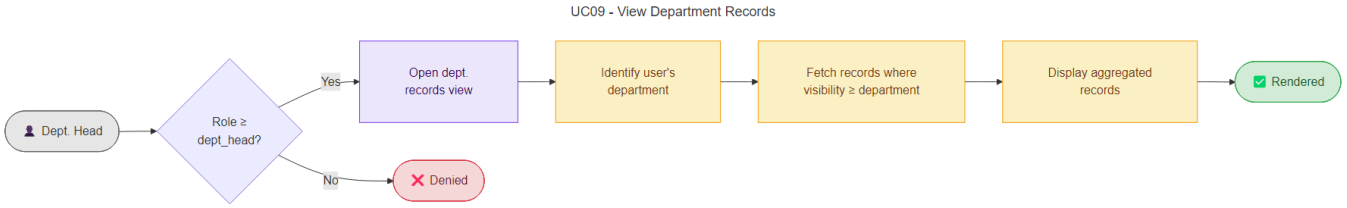
Actors: Department Head, System.

Pre-conditions: The user is authenticated with role "department_head" or higher and is associated with a department.

Acceptance Criteria: Only data with correct visibility and access permissions are displayed.

Post-conditions: Department-wide records view is rendered for the authorized user.

Figure 3 - UC09



Non-Functional Requirements

Table 2 - Non-Functional Requirements

ID	Title	Priority	Description
NFR01	Enforce Modularity	High	Clear separation between layers using containerized services.
NFR02	Enforce Reproducibility	High	Full stack reproducible via Docker Compose.
NFR03	Enforce Portability	High	Deployable on any Docker-compatible host.
NFR04	Enforce API-First Architecture	High	All functionality accessible via REST API.
NFR05	Decouple Interface	High	Frontend communicates exclusively via REST API.
NFR06	Validate Input Consistency	High	Schema-based validation at the API layer.
NFR07	Minimize Storage Usage	High	Text-based metadata only.
NFR08	Implement Rate Limiting	High	API rate limiting for abuse prevention.
NFR09	Enforce Type Safety	High	Typed models (SQLModel) and TypeScript frontend.
NFR10	Automate API Documentation	High	OpenAPI specification active for all endpoints.
NFR11	Ensure Interface Responsiveness	High	Functional across desktop and mobile.
NFR12	Display Immediate Feedback	High	Immediate visual feedback for all user actions.
NFR13	Ensure Accessibility	High	Semantic HTML and keyboard navigation.
NFR14	Hash Passwords	High	Salted t hashing for all passwords.

Relevant Models

Database Model

Figure 4 - Database Model

Developed Solution

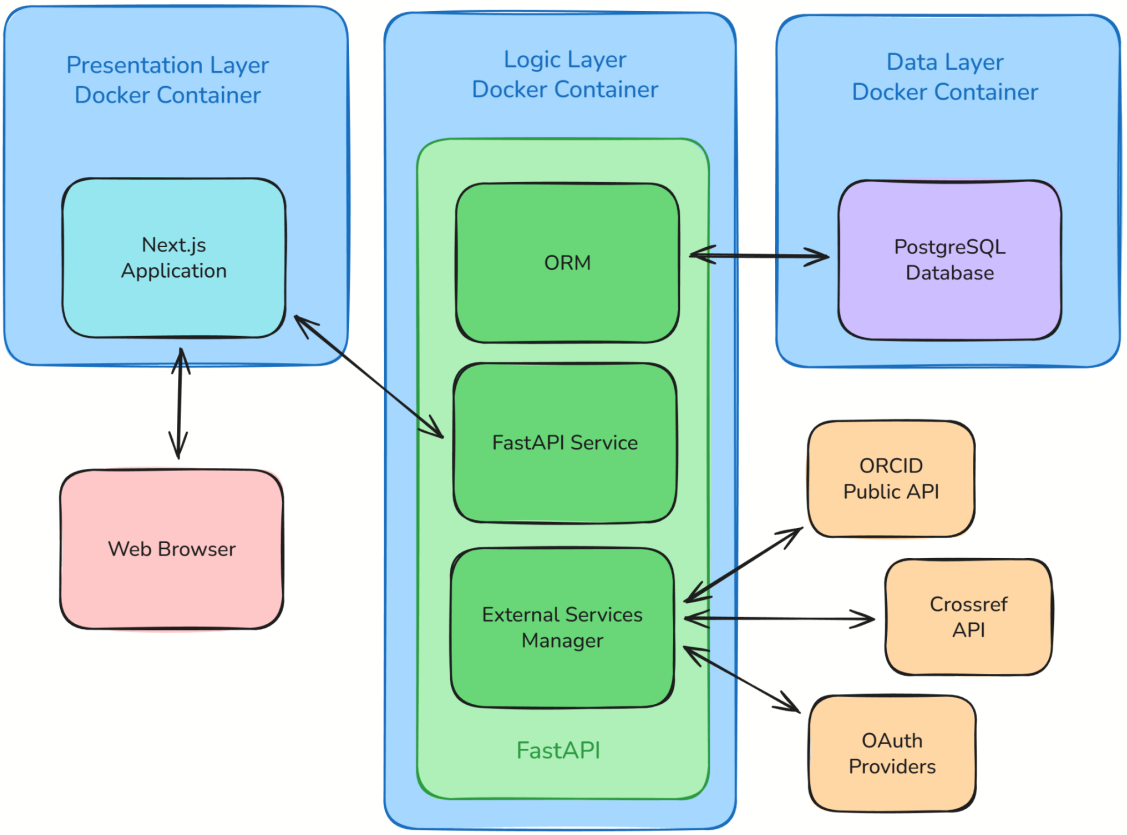
Introduction

Researcher's Diary is a distributed, modular web application that functions as a hierarchical scientific production management platform. It serves researchers as a private career workbench for managing academic metadata, and it serves department and institution heads as a reporting tool for aggregating and exporting scientific production data across their organizational units.

The system separates the Data, Logic, and Presentation layers into distinct containerized services, ensuring that the backend logic is agnostic to the interface and that each component can be developed, tested, and scaled independently.

Architecture

Figure 5 - Application Architecture



The application follows a Three-Tier Containerized architecture, orchestrated by Docker Compose with three services:

Data Layer (PostgreSQL) Responsible strictly for data persistence. Runs in a dedicated container with a health check, a named volume for data durability, and environment-based configuration.

Logic Layer (FastAPI / Python) Handles authentication, data validation, business logic, ORM operations, and API documentation. Runs in a dedicated container that depends on the database service being healthy.

Presentation Layer (Next.js / React) Consumes the FastAPI endpoints via REST to render the user interface. Runs in a dedicated container that depends on the backend service. Uses TailwindCSS for styling and TypeScript for type safety.

The three containers communicate over Docker's internal network. The entire stack is started with a single *`docker-compose up --build`* command.

Technologies and Tools

The technology stack was selected to balance an agile development process with long-term maintainability and robustness.

FastAPI was chosen as the backend framework over alternatives such as Django or Flask for its native asynchronous support (critical for non-blocking database operations), automatic OpenAPI documentation generation, and Pydantic-based request validation[FASTAPI].

SQLModel was chosen as the ORM layer over raw SQLAlchemy or Django ORM because it unifies the database model and the API validation schema into a single class definition. This eliminates the common pattern of maintaining separate ORM models and Pydantic schemas, reducing code duplication and potential inconsistency[SQLMODEL].

PostgreSQL was chosen over lighter alternatives such as SQLite for its superior handling of concurrent connections, complex relational constraints, and production parity. Since the application is designed for institutional deployment with multiple simultaneous users, SQLite's single-writer limitation would be a constraint.

Next.js with **React** was chosen as the frontend framework for its server-side rendering capabilities, file-based routing, and strong TypeScript support. TailwindCSS provides utility-first styling that enables rapid UI development without custom CSS files.

Docker Compose was chosen as the orchestration tool to ensure full environment reproducibility. The entire stack is defined in a single `docker-compose.yaml` file and can be started with one command.

Bcrypt is used for password hashing, providing industry-standard salted hashing that protects against "rainbow table" attacks. **JWT** is used for stateless authentication, enabling the frontend to authenticate API requests without server-side session storage.

Development and Production Environment

The development environment runs entirely within Docker Compose. The *`docker compose.yaml`* file defines three services:

- **DB:** PostgreSQL with health checks and persistent volume
- **Backend:** Python 3.11 slim image, uvicorn server
- **Frontend:** Node.js with Next.js

On first startup, the backend automatically creates all database tables via SQLModel's *`metadata.create_all`*. The API documentation is automatically available via Swagger UI.

User Interface

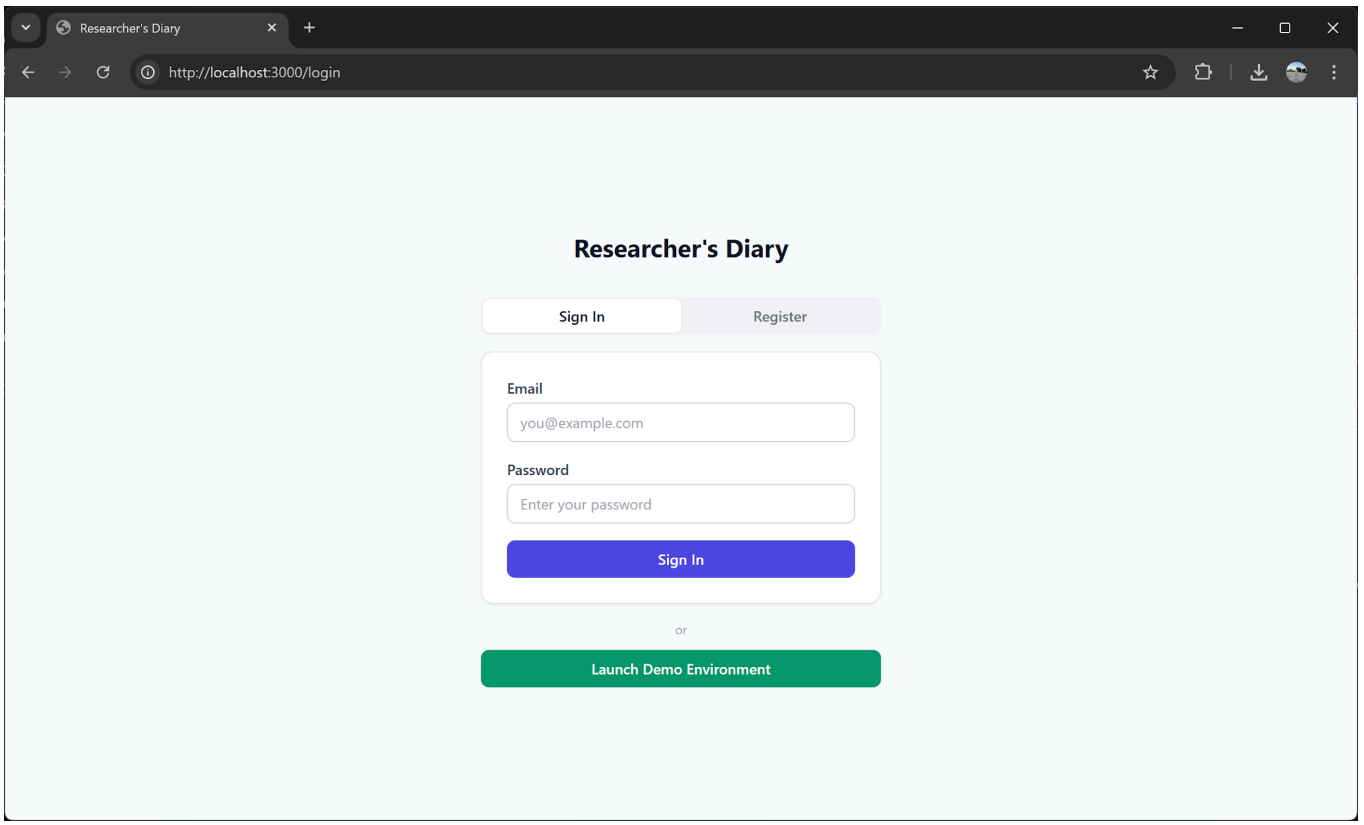
The current state of the application's frontend is functional and demonstrates the core workflows required. It is important to note that visual refinements and responsive adjustments will be iteratively implemented during the continuation of the development.

The interface adapts dynamically based on the authenticated user's role, demonstrating the database, API and frontend level permission and access verifications.

Login Page

The entrypoint of the application currently allows for regular email/password authentication. For demonstration purposes, a "Demo Environment" populates a complete hierarchy with sample data across multiple institutions.

Figure 6 - Login Page

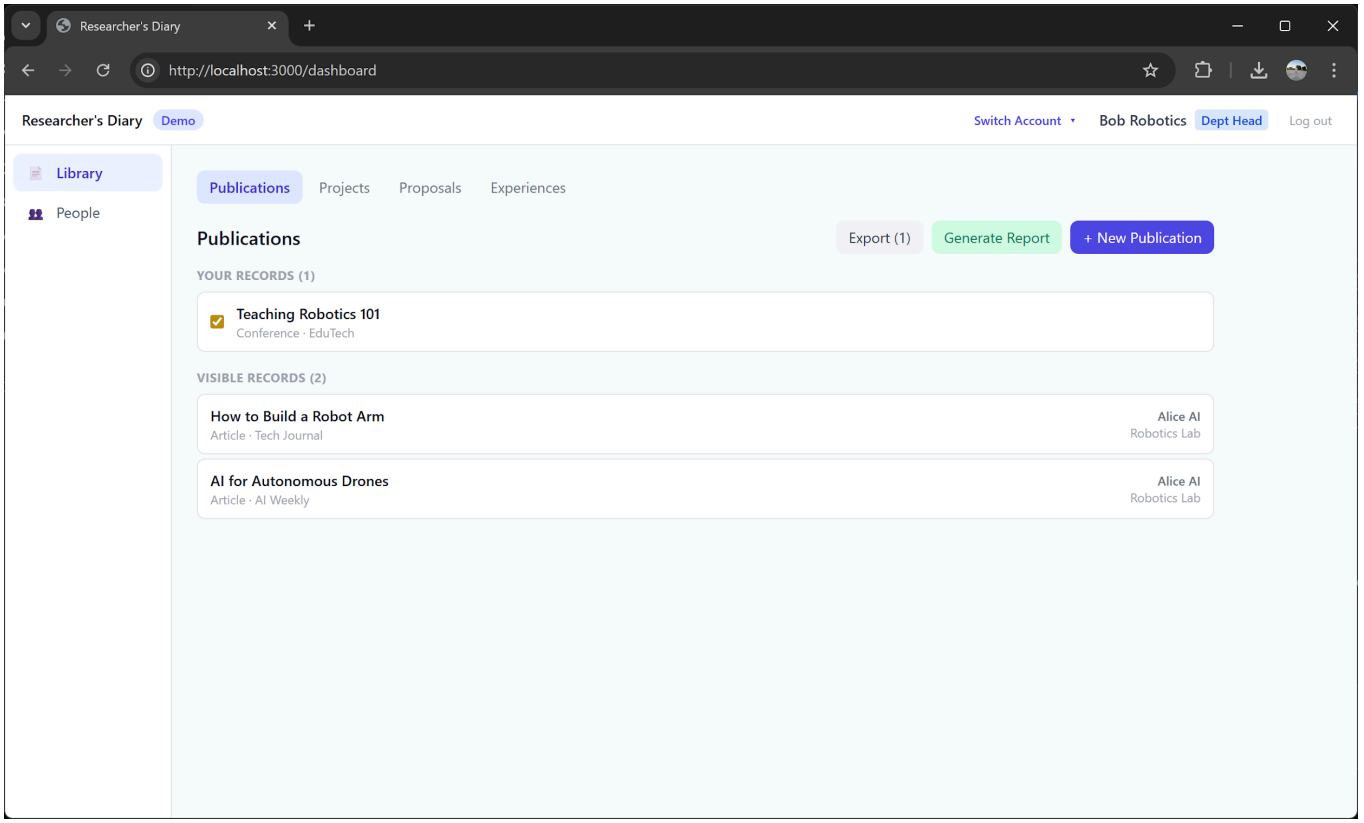


Library View

The primary workspace where researchers manage their records. Department heads and above also see a "Visible Records" section with shared data from their organizational unit.

There are also *Export* and *Generate Report* actions available based on user role.

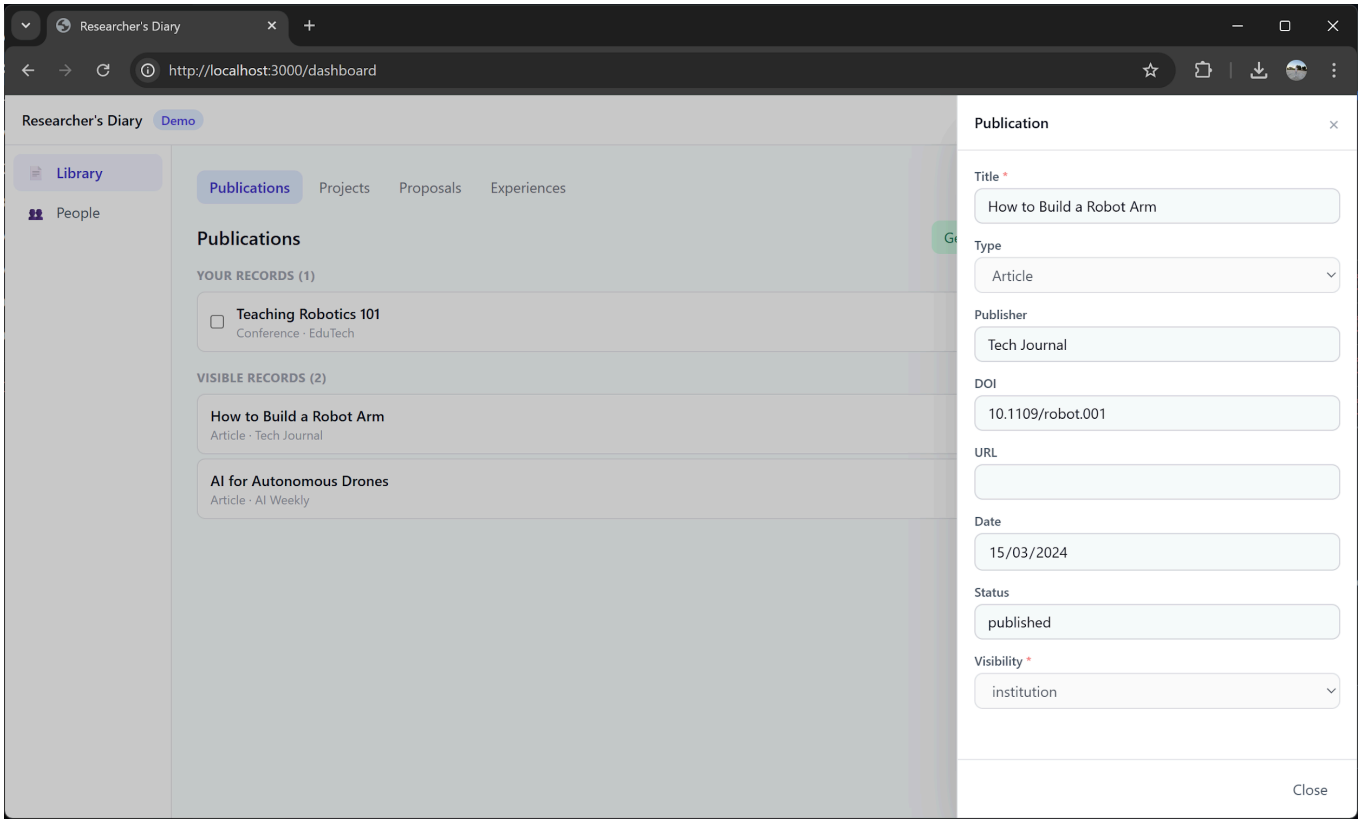
Figure 7 - Library View



Library with Record View

A slide out panel for creating, editing or viewing available record data. Ownership of a record is necessary for *edit* actions.

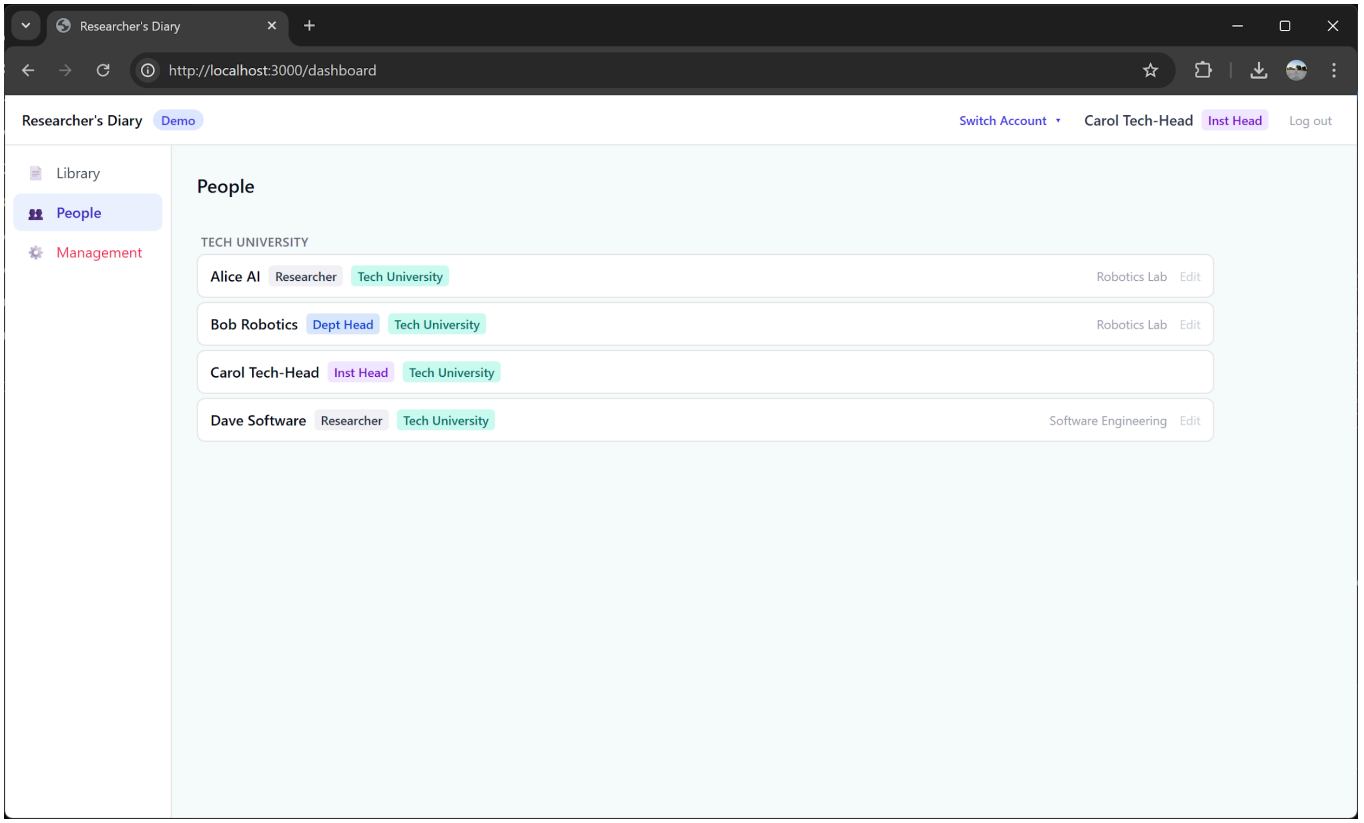
Figure 8 - Library with Record View



People View

Allows the visualization and management (based on role permission) of a user's entire institutional hierarchy.

Figure 9 - People View

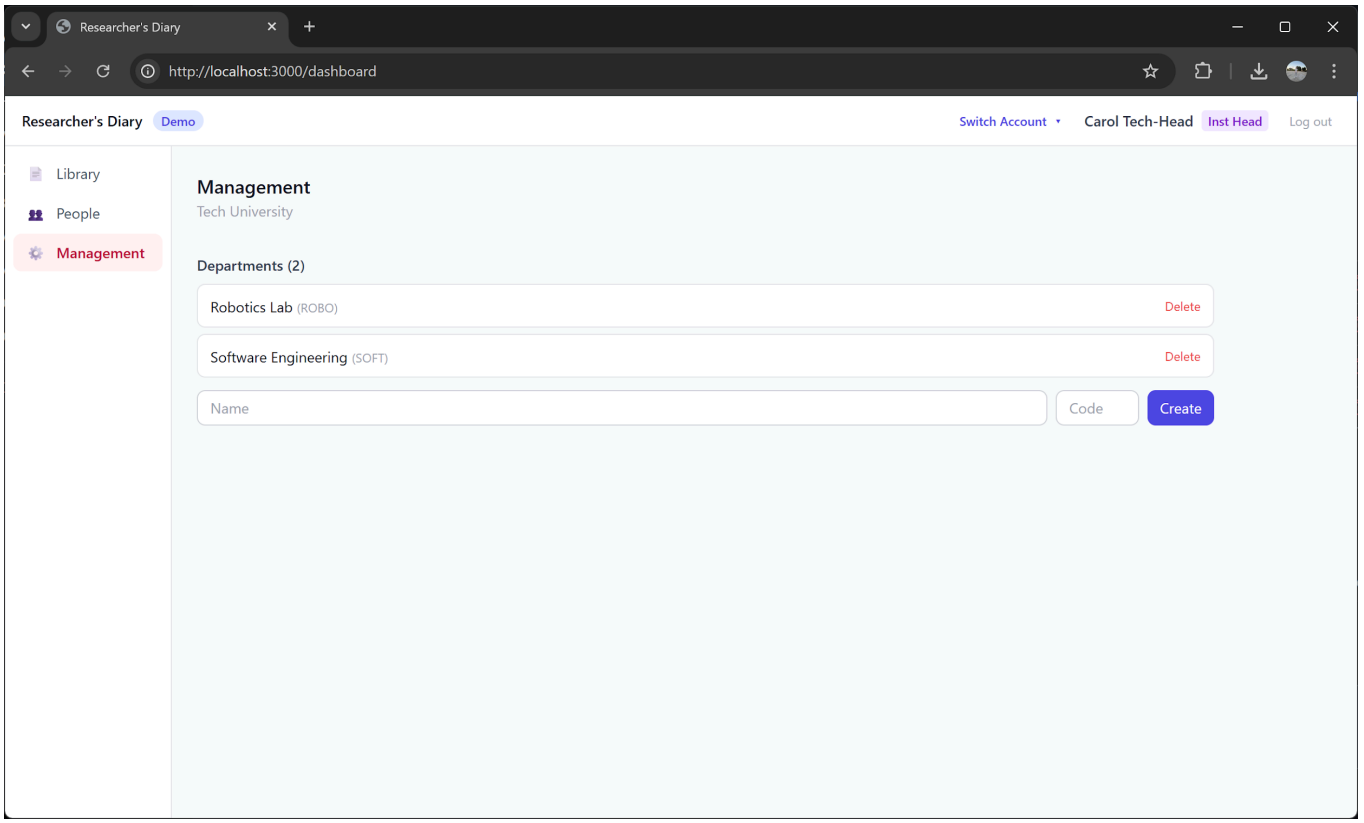


Management View

Platform-level management for creating institutions and departments, and assigning users to institutional structures.

Multi-institution management is available at *Admin* level, while multi-department management is available at *Institution-Head* level

Figure 10 - Management View



Continuous Development

While the core hierarchical engine and containerized architecture of the application are fully operational, additional features and refinements are scheduled to be iteratively and continuously implemented during the conclusion of the project.

UI/UX Refinements

The current frontend demonstrates functional workflows for all roles, but incremental features and *Quality of Life* adjustments, such as a further detailed Hierarchical people management or record creation will be implemented.

Report and Export Engines

The generation of reports that directly comply with the specific formatting requirements of major agencies and organizations is expected with the implementation of specialized export templates.

External Integrations

ORCID integration for the import of existing data and user authentication via GoogleAuth should further improve a user's *Time to Live*.

Academic Scope

This project applies knowledge from multiple areas of the Computer Engineering curriculum:

Web Programming: Full-stack development with a RESTful HTTP architecture, asynchronous request handling, and client-server communication via JSON APIs.

Databases: Relational modeling and normalization of a multi-table schema with foreign key relationships, cascade behaviors, and query optimization via role-scoped filtering.

Software Engineering: Requirements engineering with functional and non-functional requirements, iterative development methodology, and version control via Git.

Information Systems: Analysis of academic workflows and institutional reporting processes, identification of stakeholder requirements at multiple organizational levels, and design of a role-based access control system.

Security: Implementation of authentication, authorization, and data privacy controls.

Systems Architecture: Containerized microservice architecture with Docker Compose, environment-based configuration, and service dependency management.

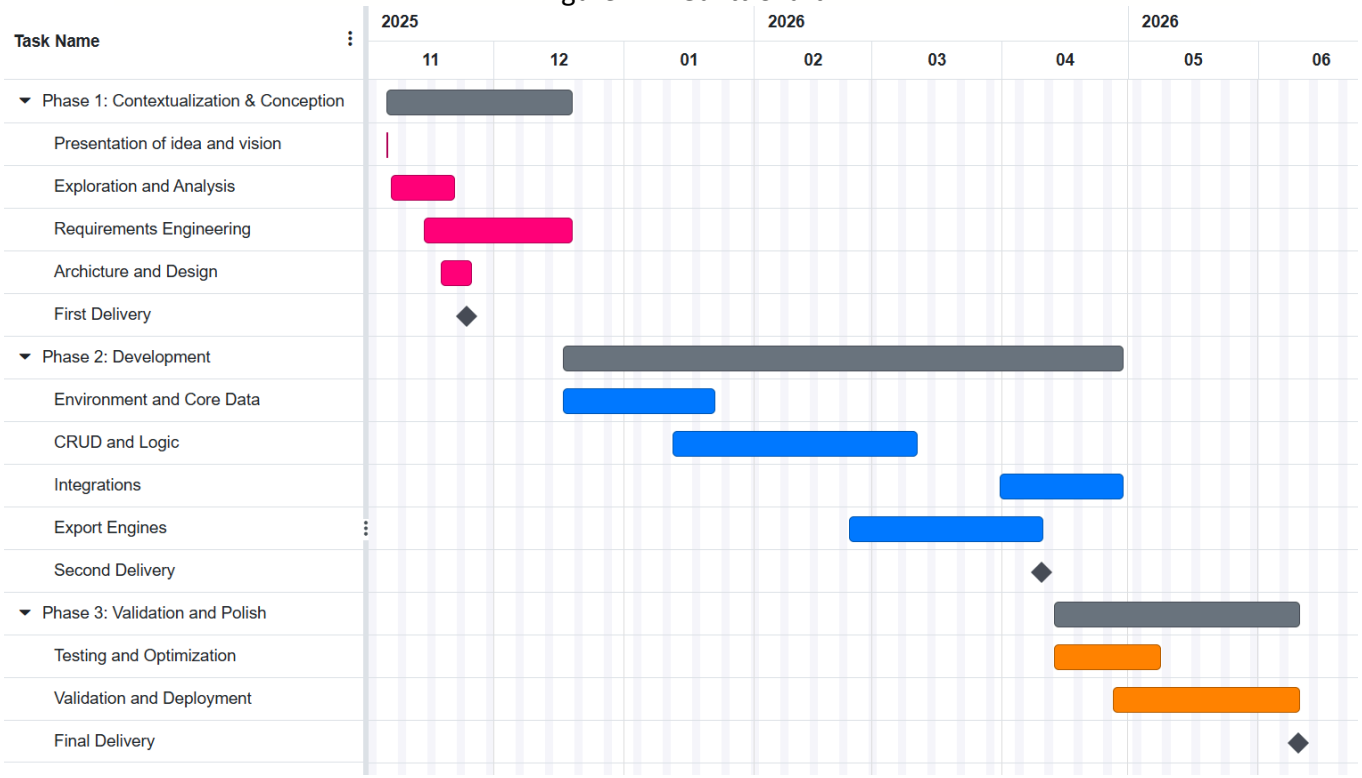
Methodology and Planning

Methodology

An iterative and incremental development methodology was adopted, guided by AGILE principles[AGILE]. The project was divided into focused development cycles, with each cycle producing a functional increment of the system. Requirements were continuously refined based on feedback from the first delivery evaluation and ongoing discussions with the project advisor.

Planning

Figure 11 - Gantt Chart



Bibliography

[DEISI24] DEISI, Regulamento de Trabalho Final de Curso, Out. 2025.

[FASTAPI] FastAPI Documentation, https://fastapi.tiangolo.com/

[NEXTJS] Next.js Documentation, https://nextjs.org/docs

[SQLMODEL] SQLModel Documentation, https://sqlmodel.tiangolo.com/

[REACT] React Documentation, https://react.dev/

[DOCKER] Docker Documentation, https://docs.docker.com/

[ORCIDAPI] ORCID Public API, https://info.orcid.org/what-is-orcid/services/public-api/

[ORCID] What is ORCID, https://info.orcid.org/what-is-orcid/

[RESP23] Asheim, B. T., Laudal, T. and Mykletun, R. J. (Eds.), *Practicing Responsibility in Business Schools: Implications for Teaching, Research, and Innovation*, Cheltenham, UK, 2023, https://www.elgaronline.com/edcollchap-oa/book/9781035313174/book-part-9781035313174-19.xml

[FCT] Fundação para a Ciência e a Tecnologia, https://www.fct.pt/en/

[HOREU] Horizon Europe, https://research-and-innovation.ec.europa.eu/funding/funding-opportunities/funding-programmes-and-open-calls/horizon-europe_en

[APS13] Association for Psychological Science, "Tidy Desk or Messy Desk? Each Has Its Benefits," *APS News Release*, Aug. 6, 2013, https://www.psychologicalscience.org/news/releases/tidy-desk-or-messy-desk-each-has-its-benefits.html

[BIBTEX] BibTeX Resource, https://www.bibtex.org/

[RGATE] About Research Gate, https://www.researchgate.net/about?_tp=eyJjb250ZXh0Ijp7ImZpcnN0UGFnZSI6InB1YmxpY2F0aW9uIiwicGFnZSI6ImluZGV4IiwicG9zaXRpb24iOiJnbG9iYWxGb290ZXIifX0

[ZOTDEV] Zotero for Developers, https://www.zotero.org/support/dev/start

[AGILE] Agile Methodology, https://www.scrumportugal.pt/agile-methodology/

[WRD] Microsoft Word, https://www.microsoft.com/en-us/microsoft-365/word

[NTON] Notion, https://www.notion.com/product

[RGATE] About ResearchGate, https://www.researchgate.net/about

[FDP12] Schneider, S. L., Ness, K. K., Rockwell, S., Shaver, K., and Brutkiewicz, R., "2012 Federal Demonstration Partnership (FDP) Faculty Workload Survey: Research Report," Federal Demonstration Partnership, 2014, https://sites.nationalacademies.org/cs/groups/pgasite/documents/webpage/pga_087667.pdf

[NSB14] National Science Board, "Reducing Investigators' Administrative Workload for Federally Funded Research," NSB-14-18, National Science Foundation, 2014, https://www.nsf.gov/pubs/2014/nsb1418/nsb1418.pdf

[ASHEIM25] Kenny, J., Fluck, A., et al., "Administrative burden in Australian universities: Insights into dimensions and drivers from a nationwide survey," Science and Public Policy, Oxford University Press, 2025, https://doi.org/10.1093/scipol/scaf029

[ROCK09] Rockwell, S., "The FDP Faculty Burden Survey," Research Management Review, vol. 16, no. 2, pp. 29-44, 2009, https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2887040/

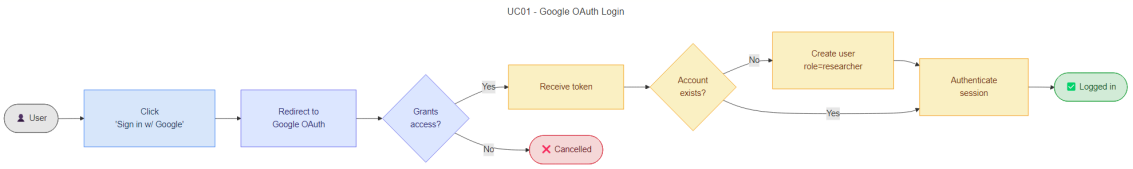
Glossary

LEI	Licenciatura em Engenharia Informática
ULHT	Universidade Lusófona de Humanidades e Tecnologias
DEISI	Departamento de Engenharia Informática e Sistemas de Informação
TFC	Trabalho Final de Curso
FYP	Final Year Project
BibTeX	Bibliographic TeX
ORCID	Open Researcher and Contributor ID
REST	Representational State Transfer

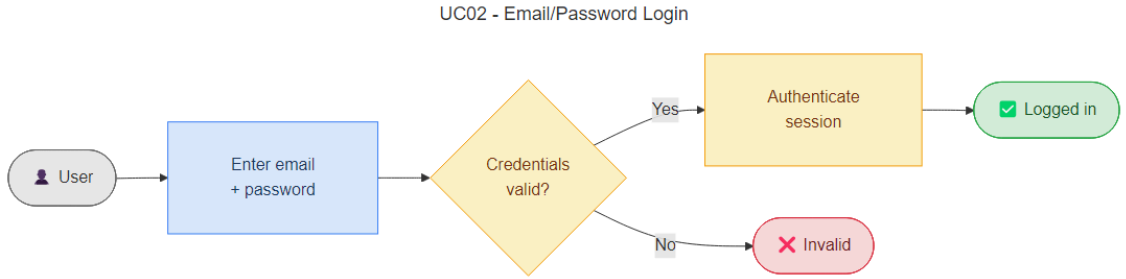
Annex A

Functional Requirements Diagrams

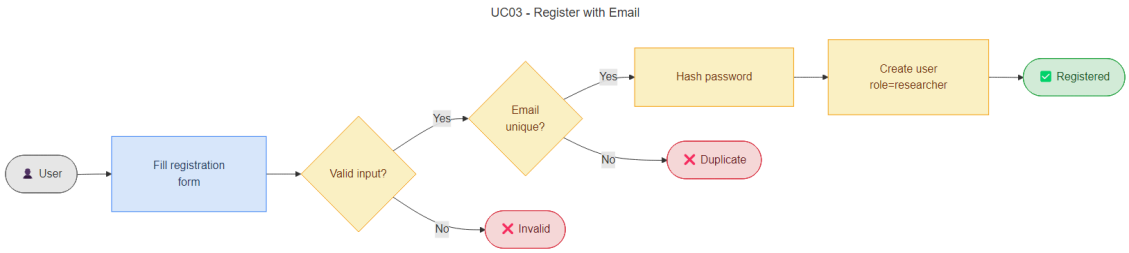
UC01



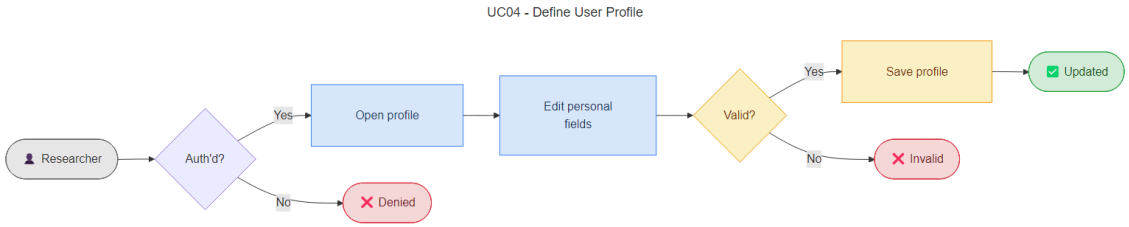
UC02



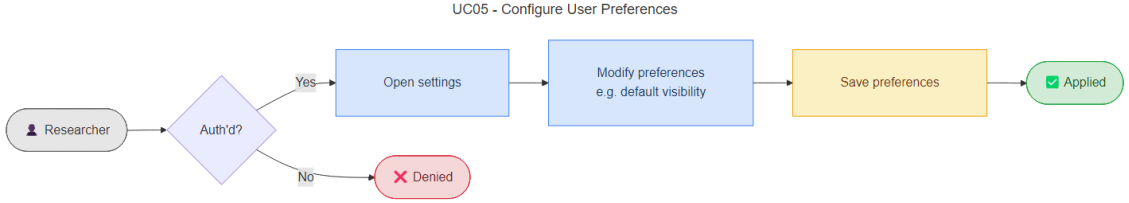
UC03



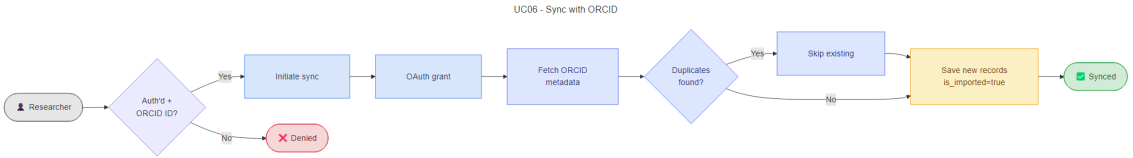
UC04



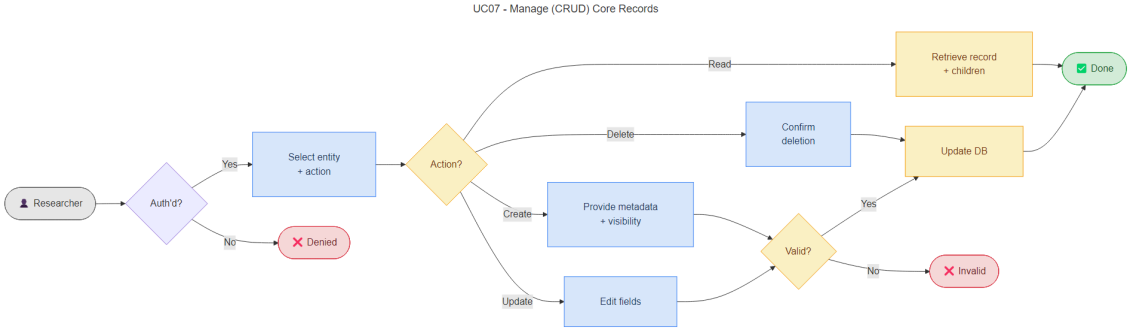
UC05



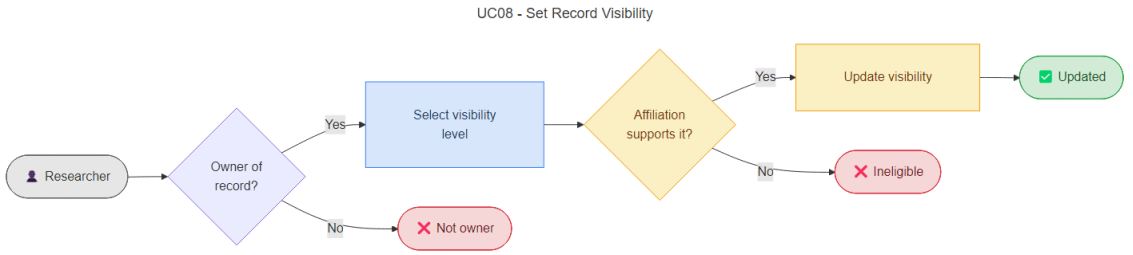
UC06



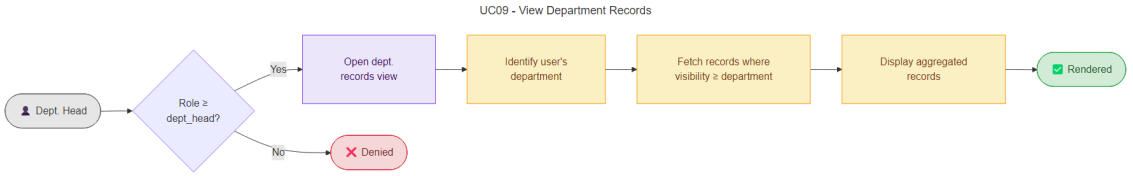
UC07



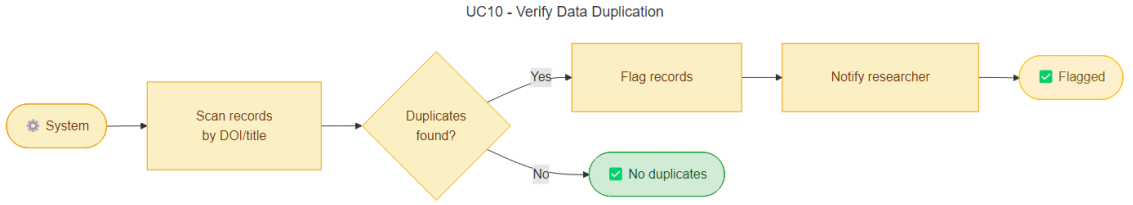
UC08



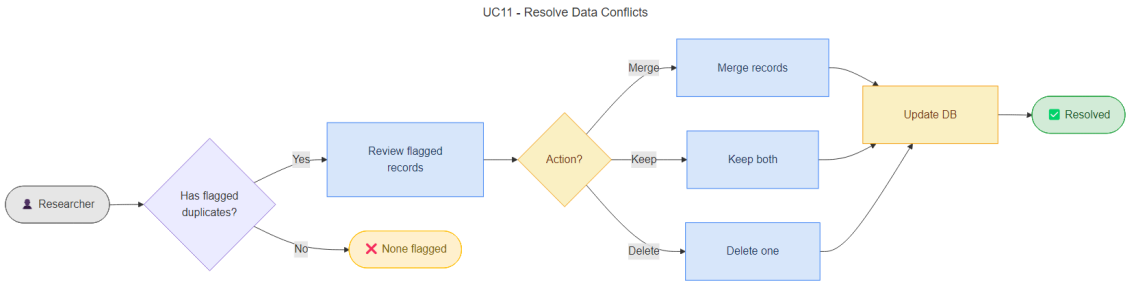
UC09



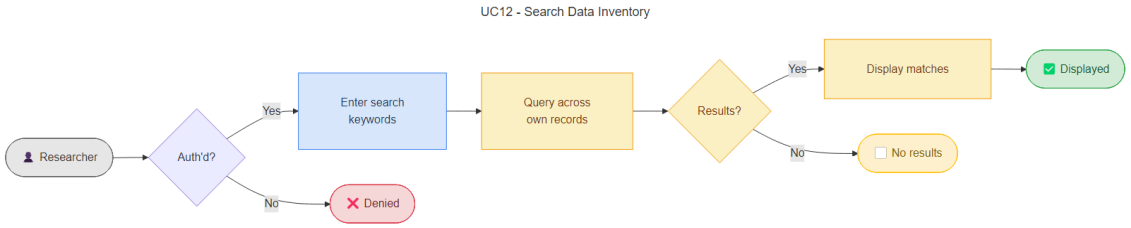
UC10



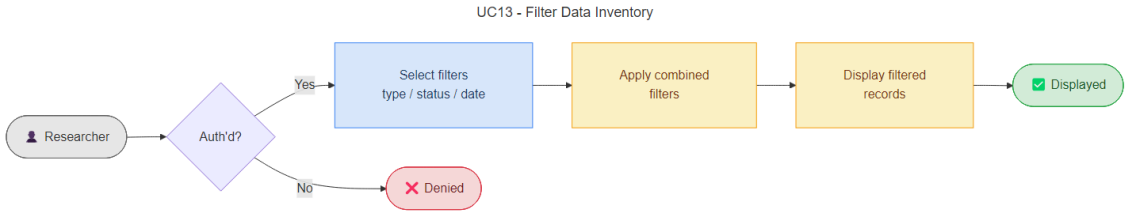
UC11



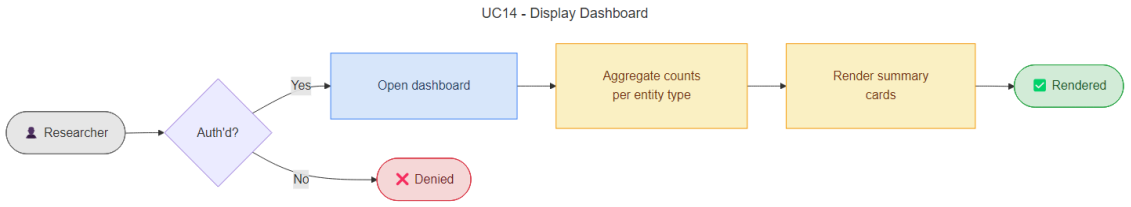
UC12



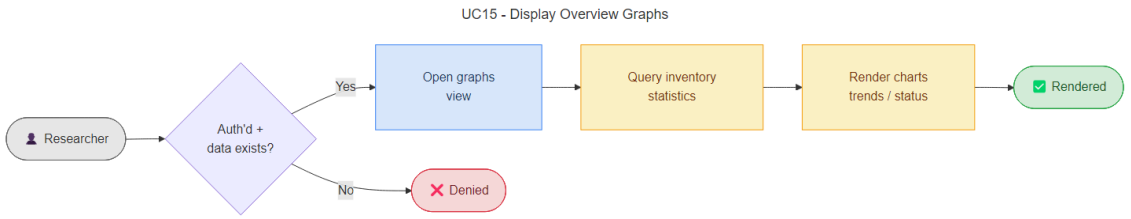
UC13



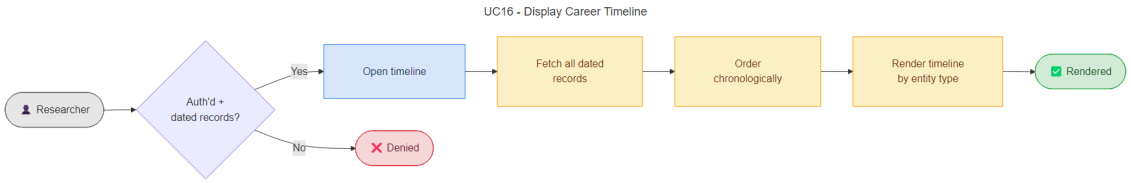
UC14



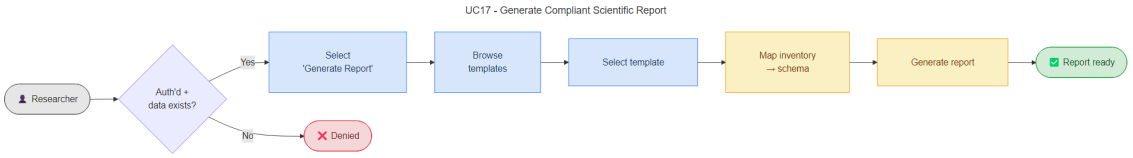
UC15



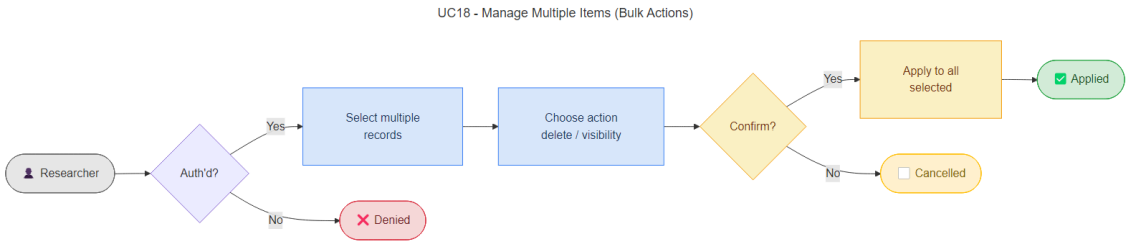
UC16



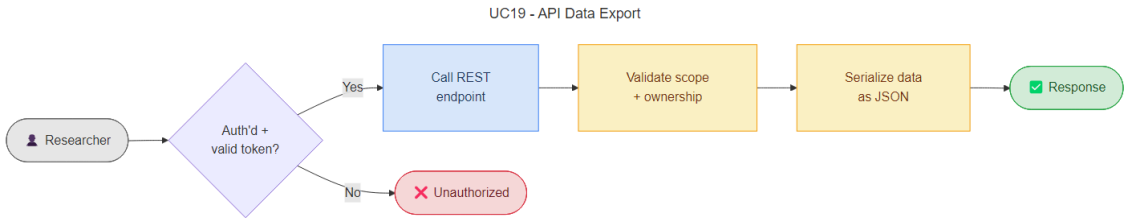
UC17



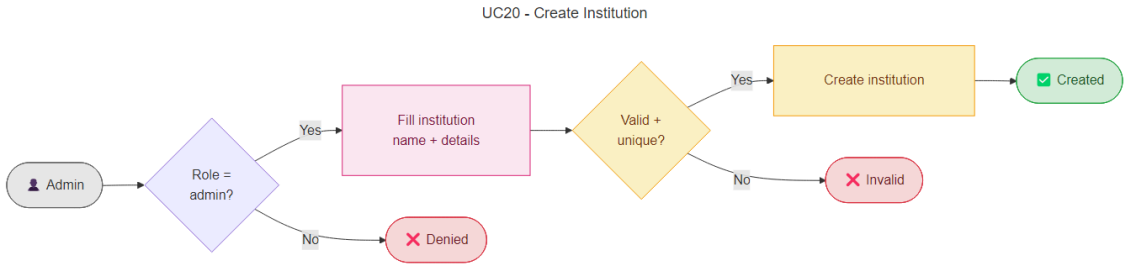
UC18



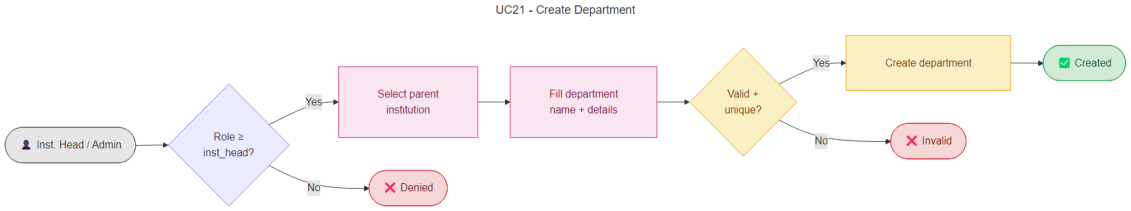
UC19



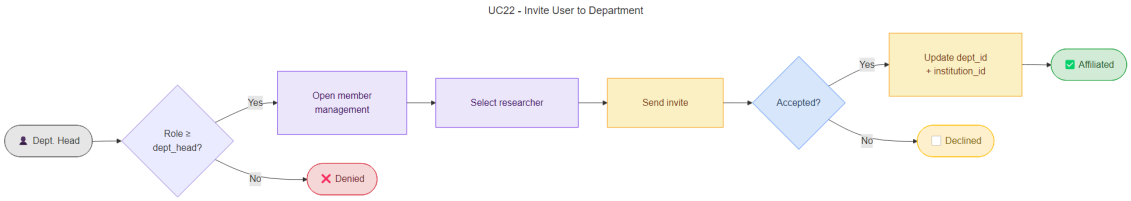
UC20



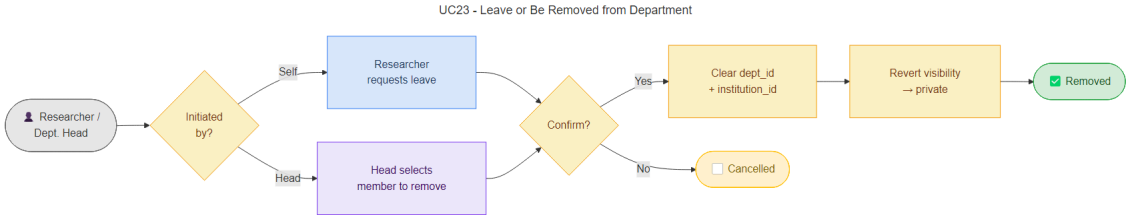
UC21



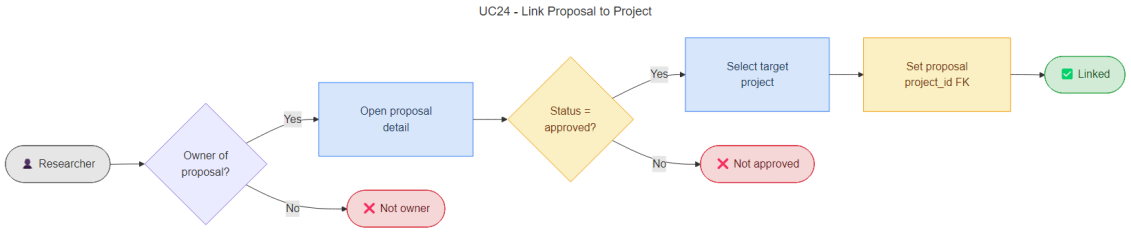
UC22



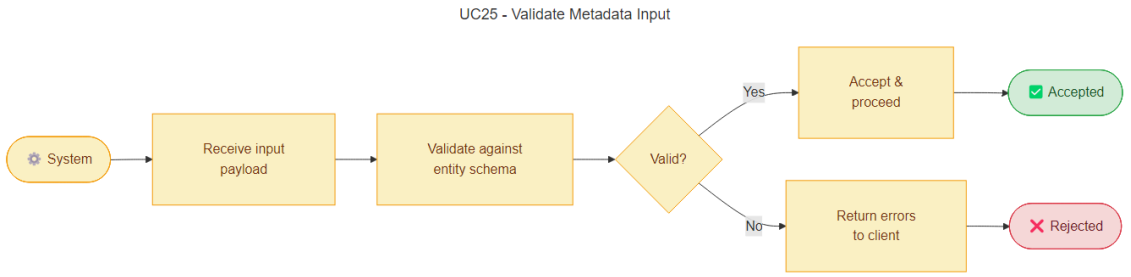
UC23



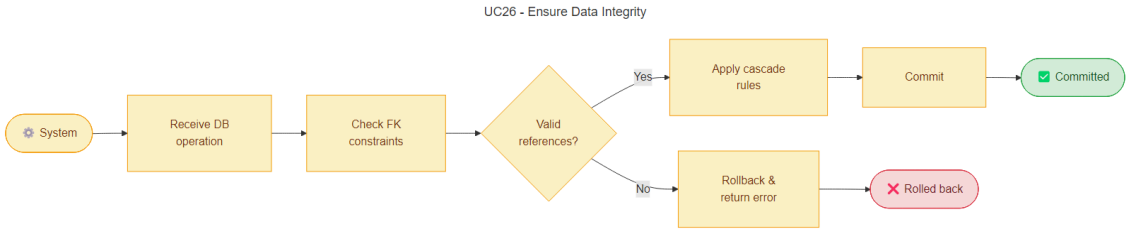
UC24



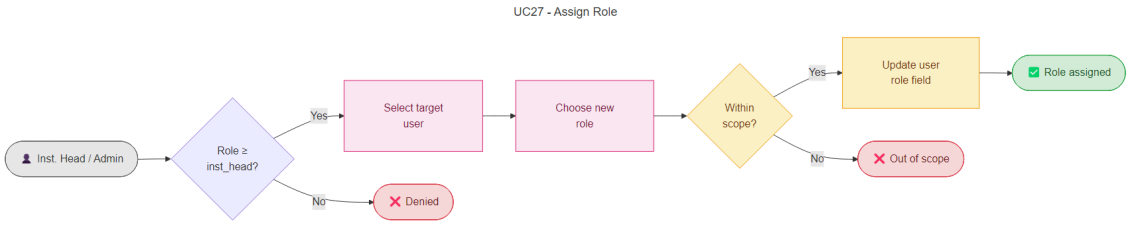
UC25



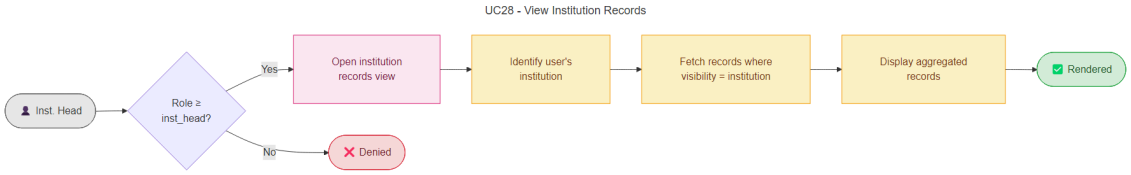
UC26



UC27


UC27 - Assign Role
Inst. Head / Admin
Role ≥ inst_head?
Yes
No
Denied
Select target user
Choose new role
Within scope?
Yes
No
Out of scope
Update user role field
Role assigned

UC28


UC28 - View Institution Records
Inst. Head
Role ≥ inst_head?
Yes
No
Denied
Open institution records view
Identify user's institution
Fetch records where visibility = institution
Display aggregated records
Rendered

Annex B

Mandatory declaration form on the use of AI during the development of this project. Starts on the following page.

Observation: The form was provided and answered in portuguese, and the resulting PDF file was appended to the project report.

Formulário de declaração de uso de ferramentas de Inteligência Artificial a anexar a relatório

Todos os relatórios deverão incluir anexo com cópia, devidamente preenchida, do formulário abaixo.
Assinalar as opções aplicáveis e completar os campos solicitados.

1. Utilização de IA

[ ] Não foram utilizadas ferramentas de IA na realização deste trabalho.
[x] Foram utilizadas ferramentas de IA na realização deste trabalho.

---

2. Ferramentas utilizadas

Assinalar todas as que se aplicam.

Assistência geral à escrita, análise ou ideação

[ ] ChatGPT
[ ] Microsoft Copilot
[ ] Gemini
[x] Claude
[ ] Perplexity
[ ] Outras. Quais? _____

Assistência à programação / desenvolvimento

[ ] GitHub Copilot
[x] Claude
[ ] OpenAI Codex
[ ] Cursor
[ ] Tabnine
[ ] Amazon CodeWhisperer / Amazon Q
[ ] Outras. Quais? _____

Geração de imagem / design / multimédia

[ ] DALL·E
[ ] Midjourney
[ ] Stable Diffusion
[ ] Canva AI / Magic Design
[ ] Outras. Quais? _____

Outros usos

[ ] Contexto: Ferramentas? _____

3. Fases do trabalho em que foi utilizada IA

[ ] Planeamento do trabalho
[x] Pesquisa exploratória / levantamento inicial de informação
[ ] Documentação técnica
[ ] Redação do relatório
[ ] Desenho / modelação / arquitetura
[x] Design / prototipagem / interface
[x] Geração de código
[ ] Revisão / refatoração / debugging de código
[ ] Criação de testes / casos de teste
[ ] Análise de resultados
[ ] Preparação de apresentação ou materiais auxiliares
[ ] Outros. Quais? _____

4. Tipo de utilização

Descrever sucintamente como a IA foi utilizada.

Exemplos: brainstorming, estruturação de secções, revisão linguística, sugestão de arquitetura, geração de exemplos, explicação de conceitos, geração parcial de código, correção de erros, criação de casos de teste, apoio ao design.

IA foi utilizada maioritariamente na geração de código 'repetido' (Models ou Schemas de base de dados similares), para verificar a consistência ao longo das várias iterações de desenvolvimento de código e relatório, geração ou alteração de elementos UI.

5. Partes do trabalho afetadas

Indicar as secções, componentes, módulos, ficheiros, entregáveis ou atividades que foram influenciados pelo uso de IA.

Módulo de frontend: maioritariamente para estruturação e estilização de UI, mas também auxílio na resolução de problemas relacionados à configuração e gestão de packages.

Módulo de Backend: Organização/manutenção de código, geração de ficheiros similares (models/schemas) e criação de dados *mock* para demonstração (ficheiro *backend/app/routers/demo.py*)

6. Exemplos de *prompt*

Inserir exemplos de *prompt*, diferenciando por âmbito (enquadrado na questão 2) e fase (enquadrado na questão 4)

"Here is the data model for my application and the implementation of 'Publication' model on my FastAPI/PostgreSQL application. According to the available data model and following the same principles, implement the remaining record-type models"

"I updated the API route for the visibility of records for the 'researcher' role. Are there any changes necessary that I forgot on the schemas or models?"

"Using the available data model and and schemas, generate mock data for demonstration purposes. The must be 2 Institutions, each with at least 4 users and one of each role."

---

7. Validação, revisão e intervenção dos autores

Descrever que verificação, revisão, correção, adaptação ou reescrita foi realizada pelos autores.

Nota: se a IA tiver sido usada em código, testes, scripts, modelos, consultas, configurações ou outros artefactos técnicos, deve ser indicado de que forma os autores validaram o funcionamento e confirmaram a sua compreensão.

Todo o código gerado foi extensamente análisado e na vasta maioria das vezes sofreu grandes alterações, tanto manuais quanto 'revisões' ("This approach causes X and should instead follow the Y pattern, for example: (...)").

---

8. Grau de utilização

[ ] Residual
[ ] Moderado
[ ] Extensivo

[ ] Utilização homogénea
[X] Grau de uso diferenciado por fase ou componente de trabalho

Descrever sucintamente os diferentes usos.

O uso foi lcoalizado e cauteloso no backend, sendo utilizado de modo mais extensivo na geração de UI.

9. Trabalhos em parceria

Protecção de dados confidenciais e recursos proprietários de parceiros

[ ] O trabalho foi realizado em parceria com entidade externa ao DEISI

No caso da resposta anterior ser verdadeira, responder às seguintes questões:

[ ] O parceiro tem regras para restringir submissão de dados
[ ] As submissões validam aplicação de regras de tratamento de dados
[ ] Foram implementados mecanismos para restringir a partilha de recursos proprietários

---

10. Declaração de responsabilidade

Ao assinarem a presente declaração, os autores declaram que:

- a informação acima é verdadeira e reflete o uso efetivo de ferramentas de IA na realização do trabalho;
- compreendem que a IA não substitui autoria nem responsabilidade académica;
- verificaram a validaram e veracidade das referências bibliográficas incluídas no relatório
- assumem integralmente a responsabilidade técnica, científica, ética e académica por todo o conteúdo submetido, incluindo texto, código, modelos, testes, imagens, diagramas e restantes artefactos entregues.

---

11. Identificação dos autores

Nome(s): Guilherme Frantz

Número(s): a22204098

Data: 11 / 04 / 2026

Assinatura(s):