ITMB
COC252
F011321

EHR System Modernisation in the Republic of Moldova

by

Calin Corcimaru

Supervisor: Dr. Georgina Cosma

Department of Computer Science
Loughborough University

May 2025

Abstract

This report outlines the result of a project that addresses a key issue in Moldova's healthcare sector: fragmentation of patient data across multiple healthcare institutions. The lack of a nationally-wide integrated system has resulted in overreliance of paper-based and manual processes, leading to reduced patient care quality and incomplete medical histories. Following initial stakeholder consultations and a review of the relevant literature, the project proposed a patient-focused prototype that allowed users to collect and manage their patient data. The solution enables document uploading and record creation, automated lab result extraction with multimodal large language models and a controlled sharing mechanism with healthcare practitioners. The prototype was implemented using a Vue frontend and FastAPI backend as a web application that demonstrates the feasibility of a patient-focused solution in Moldova's healthcare sector. Stakeholder feedback was collected throughout the whole development process through a series of live demonstrations, allowing for iterative improvements to the application. Final stakeholder feedback confirms the application's potential to enhance care quality and introduce a patient-centric approach to healthcare in Moldova. Overall, this project aims to contribute to Moldova's digital healthcare transformation and hopes to serve as a foundation for future development and implementation of a real-world service.

Keywords: Personal Health Record, Republic of Moldova, Data Fragmentation, Multimodal Large Language Models, Patient-Centric Healthcare

Acknowledgements

I would like to express my gratitude to my project supervisor, Dr. Georgina Cosma, for her invaluable guidance and support throughout this project. Her feedback and encouragement have been crucial in motivating and encouraging me to continue exploring and innovating in my system design and development.

I am also deeply grateful to all the stakeholders who have generously contributed their time and ideas to this project. Their insights into the workings of Moldova's healthcare sector and the digital landscape of healthcare technology have been invaluable in shaping the direction of this project. Their feedback and suggestions have been instrumental in refining the system design and ensuring that it meets the needs of its users and improves the quality of care provided to patients. This ultimately ensured that this project efficiently addresses the real-world challenges faced by the healthcare sector in Moldova.

Next, I would like to extend my appreciation to the Loughborough University staff and faculty who provided all the necessary knowledge and support throughout my studies. This project represents a culmination of everything I have learned during my degree, and I am grateful for the opportunity to apply my skills and knowledge in a practical setting.

Finally, I would like to thank my family and friends, who have always been encouraging me during this project. Their belief in my abilities and support have been a source of motivation and inspiration.

I'd like to dedicate this project to my late grandfather, Ion Corcimaru, who was a well-known doctor in Moldova. His devotion to this field and commitment to saving patient lives have always been a source of inspiration for our family. Even though I'm not pursuing a career in medicine, I hope to honour his legacy by contributing to the improvement of healthcare in Moldova through this project. I hope that my work will help to create a better healthcare system for future generations, just as he did for his patients.

Contents

List of Figures

List of Tables

List of Abbreviations

API	Application Programming Interface
CNAM	Centrul Național de Asigurări în Medicină (National Health Insurance Company)
CNPDCP	Centrul Național pentru Protecția Datelor cu Caracter Personal (National Center for Personal Data Protection)
EHR	Electronic Health Record
EMR	Electronic Medical Record
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
JSON	JavaScript Object Notation
JWT	JSON Web Token
LLM	Large Language Model
MLLM	Multimodal Large Language Model
MFA	Multi-Factor Authentication
NLP	Natural Language Processing
ORM	Object-Relational Mapping
PHR	Personal Health Record
SDLC	Software Development Life Cycle
SFC	Single File Component
SIA AMS	Sistemul informațional automatizat „Asistența Medicală Spitalicească”
SPA	Single Page Application
SSL	Secure Sockets Layer
UML	Unified Modeling Language
USMF	Universitatea de Stat de Medicină și Farmacie (State University of Medicine and Pharmacy)
UUID	Universally Unique Identifier

Glossary

Application Programming Interface (API) A set of protocols and tools that allow different software applications to communicate with each other.

Authentication The process of verifying the identity of a user.

Authorisation The process of determining whether a user has permission to access a resource or perform an action.

Backend The server-side of an application that handles business logic, database operations and API requests.

Dependency injection A software engineering design pattern where a function receives the object it depends on from an external source rather than creating it itself.

Entity-Relationship Diagram (ERD) A visual representation of the database structure, showing the relationships between different entities.

Frontend The client-side of an application that users interact with by using a user interface.

Hallucination A phenomenon in which a large language model generates information that appears to be factual but is incorrect or completety made up.

Javascript Object Notation (JSON) An interchange format for structured data representation, often used in APIs to exchange data between a client and a server.

Object Relational Mapping (ORM) A programming technique that allows developers to interact with a database by mapping database tables to classes and rows to objects.

Republican hospital A tertiary healthcare institution in Moldova that serves as the highest level of specialised medical care in the country.

Single Page Application (SPA) A web application that loads a single page and dynamically updates the content as the user interacts with the app.

Chapter 1

Introduction

1.1 Background

The Republic of Moldova, a small country in Eastern Europe that borders Romania and Ukraine, with a population of 2.4 million people, has faced significant economic and political challenges since gaining independence in 1991 (National Bureau of Statistics of the Republic of Moldova, 2024; BBC, 2024). Despite its struggles, Moldova has made substantial progress in its digital transformation initatives aimed at modernising public services and improving citizens' quality of life (E-Governance Agency, n.d.[a]). A notable example is the Citizen's Government Portal (MCabinet), which provides digital access to all citizens via electronic signature to government-held information and select services (E-Governance Agency, n.d.[b]). Continuing in this direction, Moldova's Cabinet of Ministers has recently approved the 'Digital Transformation Strategy of the Republic of Moldova for 2023–2030', which aims to transform the country into a digital society by 2030, with the ultimate goal of having 'all public services available in a digitalized format' (United Nations Development Programme, 2023).

1.2 Problem Statement

In parallel to government service transformations, the healthcare sector has also seen some digitalisation efforts, evidenced by the introduction of an EHR system in 15 hospitals across the country in 2017, SIA AMS (Ciurcă, 2021). While the system has improved practitioners' access to patient information, the system hasn't been updated since its inception in 2017 and several challenges persist in 2024.

The system's user experience represents its main limitation, with an outdated interface that is difficult to navigate and fails to meet modern accessibility standards. Additionally, it is only accessible via Internet Explorer or a legacy version of Microsoft Edge, with no support for other browsers or devices (Ciurcă, 2021).

Another fundamental challenge is the siloing of EHR systems across the country. The lack of a nationally-wide integrated system, such as the NHS in the UK, has resulted in fragmented patient data spread across individual healthcare institutions, with no centralised access available (Ciurcă, 2021). In turn, this fragmentation has created an overreliance on paper-based or manual processes, which are time-consuming and prone to human error, hindering the quality of care provided to patients.

Moldova's precarious economic situation further complicates any efforts to modernise the current healthcare system. As such, Moldovan citizens are incresingly turning to private digital initiatives and local startups that offer innovative solutions, as evidenced by recent developments showcased at events like the Digital Health Forum 2024 (Startup Moldova Foundation, 2024).

1.3 The Client

The department of external relations at USMF will serve as this project's client. USMF is a public institution that offers a range of medical programs, including medicine, dentistry and pharmacy (USMF, 2023). Many of the faculty at USMF are also practicing doctors at hospitals and clinics across Moldova, and have first-hand experience with the current digital systems used in both public and private medical institutions.

Members of the department of external relations have expressed their interest to investigate potential technology-enabled solutions to improve the healthcare sector in Moldova. They have requested for a prototype of a digital system that will either replace or complement existing technological healthcare offerings in Moldova.

1.4 Project Objectives

This project aims to address some the challenges faced by the healthcare sector in Moldova, focusing on providing a working, high-fidelity prototype. This will be achieved by a combination of market research and stakeholder interviews, followed by a literature review of existing technologies that will power the prototype, and finally, the design and development of a working prototype. The project's ultimate goal is to create a viable solution that

will establish a foundation for potential future broader implementation within Moldova's healthcare sector.

As such, the objectives of the project are as follows:

1. Identify 2 to 4 stakeholders across different healthcare perspectives by the end of November 2024 to provide insights into current practices and systems used in Moldova.
2. Conduct at least one interview with each identified stakeholder by mid-December 2024 to gather information about current systems used, their challenges and requirements.
3. Carry out a literature review of at least 30 academic sources by the end of January 2025 to evaluate 3 appropriate frameworks and 2 project management methodologies that can be used to develop a modern EHR system.
4. Analyse at least 2 existing EHR systems by the end of January 2025 and compare their strengths and weaknesses in a table format.
5. Design and develop a functional EHR system prototype that implements the top 10 most important gathered requirements by the end of April 2025.
6. Present the completed prototype to at least 3 stakeholders by end of April 2025 to gather final feedback and validate its potential for further development.

Chapter 2

Research and Requirements

This chapter will outline the stakeholder analysis process, followed by requirements gathering. A diverse group of stakeholders was chosen to ensure multiple perspectives of the healthcare sector are represented and so that a comprehensive list of requirements is created. At the of this chapter, a list of the most important requirements will be presented.

2.1 Stakeholder Analysis

The initial stakeholders identified included:

- Doctors and other medical staff working in hospitals
- Department head of a republican hospital
- IT staff members in hospitals
- CNAM staff members
- Ministry of Health staff
- Patients

These stakeholders were mapped on an influence-interest grid (which will be discussed in section 3.2.2) to help better understand the level of influence and interest that each stakeholder has in the project and prioritise the engagement with them. The grid can be seen in figure 2.1.

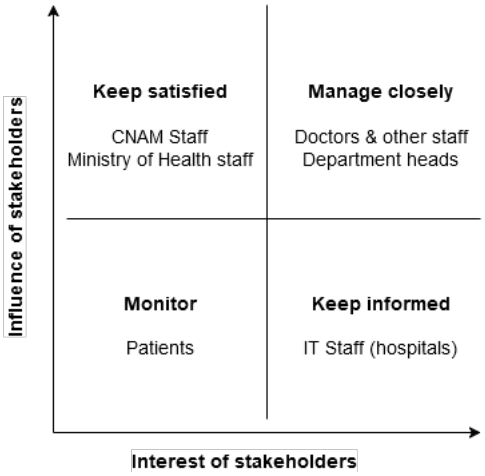


Figure 2.1: Chosen stakeholders in the stakeholder influence-interest grid

2.1.1 Current situation analysis

Following the analysis, exploratory interviews were conducted with available stakeholders, excluding CNAM and Ministry of Health staff members who were not available for interviews. These discussions revealed three main challenges in Moldova's healthcare sector:

1. The current EHR system is outdated and only accessible via Internet Explorer or legacy version of Microsoft Edge. A potential solution was proposed to develop a modernised version of the existing system, retaining the core functionality.
2. The lack of interoperability between medical institutions due to an absence of a nationally-wide integrated system. A potential solution was proposed to create a digital patient archive, where users can upload their own medical records (such as lab tests, previous medical history, etc) and share them with any healthcare practitioner.
3. The lack of digitalisation of systems that still rely on paper-based records, such as the national transplant registry. A potential solution was proposed to create a digital transplant registry, that can be accessed by any healthcare practitioner in Moldova.

After evaluating the feasibility of the proposed solutions, it was decided for the project to focus on issue #2, as the other two solutions were determined to be too complex for the constraints of an academic project. The decision was supported by the fact that the chosen solution would be independent from existing healthcare systems, allowing for a more straightforward and less complex implementation.

Consequently, the stakeholder list has been updated to reflect the changed focus of the

project:

- Doctors working in hospitals and clinics
- Ministry of Health staff
- CNAM staff
- Patients
- Other medical staff members
- CNPDCP staff

At the same time, the stakeholder influence-interest grid has been updated to reflect the changes in the project focus, which is illustrated in figure 2.2.

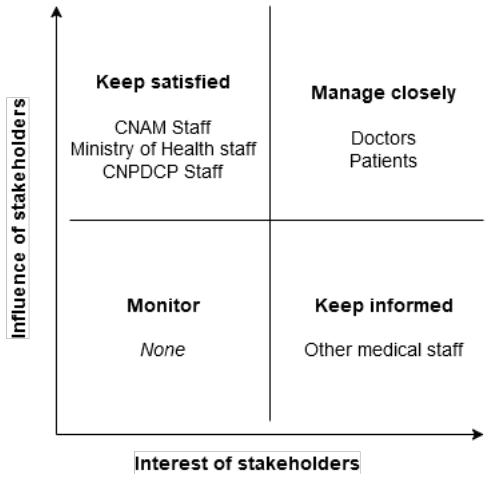


Figure 2.2: Updated stakeholders in the stakeholder influence-interest grid

2.2 Requirements

Additional interviews were conducted to focus on specific requirements for the chosen solution. Enough information was gathered from the other stakeholders to identify the core requirements for the project. Staff members at CNAM, the Ministry of Health and CNDCP were still not available for interviews, so instead legislation and regulations on their websites will be reviewed (Compania Națională de Asigurări în Medicină, n.d.; Centrul Naţional pentru Protecţia Datelor cu Caracter Personal al Republicii Moldova, n.d.; Ministerul Sănătății al Republicii Moldova, 2024).

All of the gathered requirements can be found in appendix A, but the most important ones have been summarised in the table below:

ID	Category	Requirement
1	Document upload	When a the lab category is selected during record creation, the system must allow the user to enable AI processing.
2	Document upload	If lab extraction was enabled, the system must display the extraction results to the user before sending to the database.
3	Document upload	The system must allow patients to upload their own medical records in a variety of formats (PDF, DOC, etc).
4	Personal cabinet	The system must display the patient's history in a chronological order in a table format.
5	Personal cabinet	The patient personal cabinet must provide an overview of the patient's history through 3 main sections: personal information, lab tests, and doctor consultations.
6	Personal cabinet	The system must allow patients to add their own personal information, such as name, date of birth, or address.
7	Personal cabinet	The system must allow patients to add their own allergies, vaccinations and medication.
8	Personal cabinet	When multiple vital and lab entries are made, the system should display a historical graph of the results.
9	Shareable link	The system must allow the patient to generate a shareable link to provide access to their medical records.
10	Doctor view	When shared with the doctor, the system must allow the doctor to only view the patient's history, not edit it.

Table 2.1: Summary of the Most Important Requirements

Chapter 3

Literature Review

This chapter will provide a review of the existing literature, which will be used to guide the planning and development efforts of the project.

As such, it will be covering the following areas:

- Software development methodologies
- Requirement and stakeholder management
- Tech stack (backend and frontend)
- Large Language Models (LLMs)
- PHR Systems

3.1 Software Development Methodologies

3.1.1 The software development life cycle

The SDLC is a process used to guide the development of software applications or systems (Ruparelia, 2010). It consists of multiple phases, each with its own set of activities and deliverables.‘ ’Yas, Alazzawi, and Rahmatullah (2023) outline the phases of the SDLC as following:

1. **Requirement gathering and analysis phase** — the requirements are gathered saved in a document. Based on the requirements gathered, a development plan is created.

2. **Design phase** — requirements are written in a more technical manner and system desings are created.
3. **Implementation phase** — Actual development of the software occurs in this phase. Additionally, some smaller unit tests may be done during this phase.
4. **Testing phase** — may involve multiple types of testing, such as unit testing, integration testing, and system testing.‘ ’Luo (2001) describes the different types of tests:
    - Unit testing — testing individual units or components of the software.
    - Integration testing — performed on two or more units combined together, focusing on the interfaces between these components.
    - System testing — focuses on the ‘end-to-end quality of the entire system’, testing it as a whole.
5. **Maintenance phase** — involves the deployment and maintenance of the software. Additionally, this phase may include end-user acceptance testing, to ensure that it meets their needs (Luo, 2001).

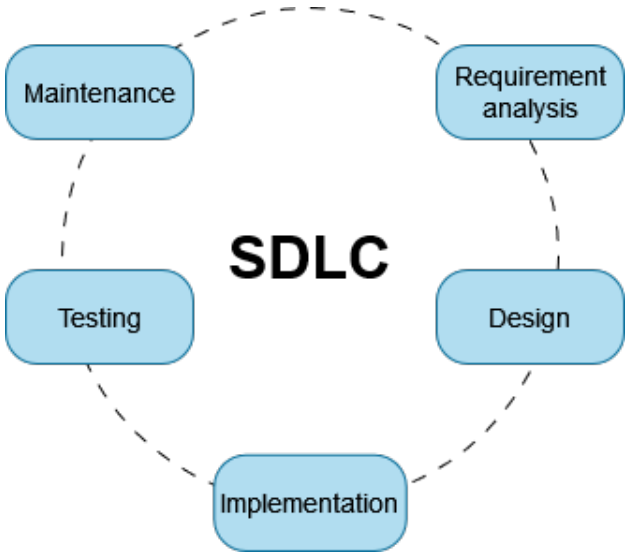


Figure 3.1: The Software Development Life Cycle

3.1.2 SDLC Models

The literature describes several SDLC models that have been used in the development of software applications.‘ ’Ruparelia (2010) and Yas, Alazzawi, and Rahmatullah (2023) highlight the most common ones: Waterfall model, V model, Spiral model, Interative model, and Agile model.

Waterfall Model

The Waterfall Model is probably the most well-known SDLC model. It is a linear model, where the development process is divided into distinct, sequential phases (which can be seen in figure 3.2).

The Waterfall Model's strengths lie in its simplicty of use, ease of understanding and a clear, structured approach (Alshamrani and Bahattab, 2015). An additional stregth that the authors note is its extensive documentation and planning, emphasising quality and adherence to regulations.

On the other hand, one of Waterfall's main weaknesses is its lack of flexibility in regards to change (Mirza and Datta, 2019). Thus, this model is not suitable for projects where the requirements are not well understood or are likely to change. Additionally, the project deliverable is not avaialable until the end of the project, any changed or feedback cannot be done during its development (Mirza and Datta, 2019).

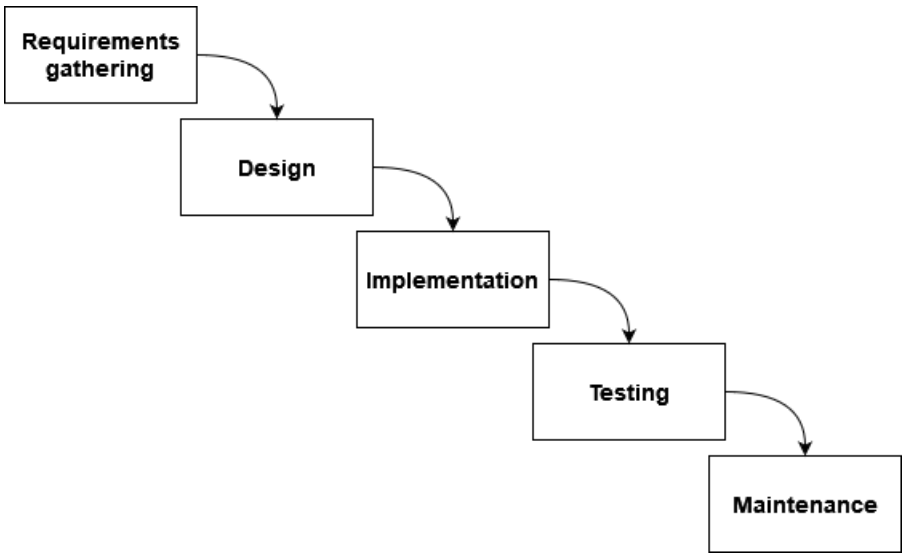


Figure 3.2: Waterfall model

Agile Model

Another well-known model is Agile. It has multiple frameworks, with Scrum and Kanban being the most popular ones. Scrum is a framework where the project is divided into sprints, each lasting between 2–4 weeks, that aim on delivering value to the customer through incremental software features (Alqudah and Razali, 2018; Sunner, 2016). Kanban focuses on visualising the project workflow by using a visual board with columns, cards and swimlanes. It uses column limits and a pull system to make the flow of work through the system more efficient (Sunner, 2016).

Agile has some drawbacks — its less formal and detail documentation and planning, especially in the early stages of the project, may not be suitable for large scale projects (Sunner, 2016; Yas, Alazzawi, and Rahmatullah, 2023). Similarly, lack of knowledge on how to use the frameworks may be a barrier for some teams (Mirza and Datta, 2019; Yas, Alazzawi, and Rahmatullah, 2023).

Nowadays, the combined use of Scrum and Kanban is becoming quite popular, with many teams employing both frameworks in their projects, allowing them to adopt the appropriate practices and adapt them accordingly based on their needs (Alqudah and Razali, 2018).

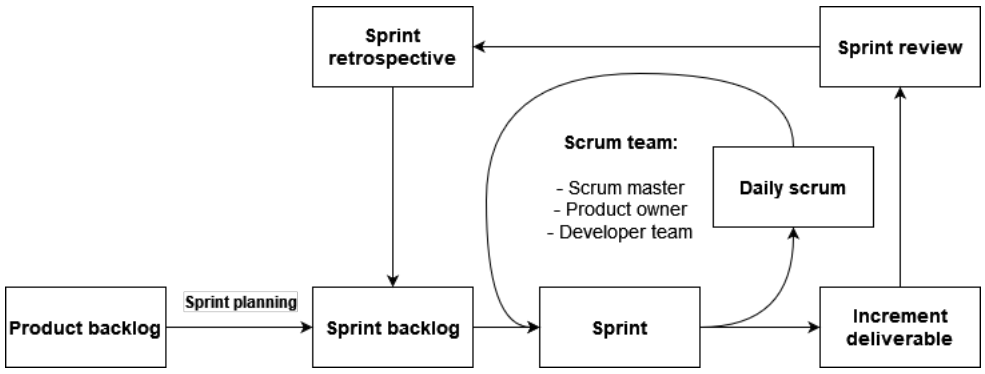


Figure 3.3: Scrum framework

3.1.3 A hybrid approach

A hybrid approach has also been emerging in software development projects. Various surveys report the most common combinations are Scrum, Iterative Development, Kanban, Waterfall and DevOps, with hybrid Waterfall and Scrum being the most popular one (Gemino, Reich, and Serrador, 2021; Prenner, Unger-Windeler, and Schneider, 2020). In this approach, the development part is done in an Agile way, with the rest of the project using Waterfall as a backbone (Prenner, Unger-Windeler, and Schneider, 2020).

Gemino, Reich, and Serrador (2021) note that projects using either Agile, traditional or hybrid approach show similar levels of success in terms of budget, time and quality. However, the authors have found that agile and hybrid approaches perform much better on customer satisfaction than the traditional ones.

3.2 Requirements Gathering

Requirements gathering is the first step in any software development process. As described by Young (2002), a requirement is a 'necessary attribute in a system...that identifies a capability, characteristic, or quality factor of a system in order for it to have value and utility to a user'. Multiple studies mention how proper requirement gathering plays a pivotal role in the project success, with many project failures attributed to poor requirements gathering (Hickey and A.M. Davis, 2003; S. Sharma and Pandey, 2013; Alan Davis et al., 2006a).

3.2.1 Requirement types

Requirements can be classified into 2 categories: functional and non-functional. Functional requirements describe the system's behavior, while non-functional requirements describe the system's quality attributes, such as performance, security, reliability, etc.' '(Laplante, 2019, p. 6).

When writing the requirements in a document, it is important to ensure clarity and conciseness to avoid ambiguity. Based on the recommendations of Laplante (2019, p. 112) and IEEE (2018), the following principles and practices can be followed:

- Write in a simple and consistent language.
- Avoid technical jargon, vague terms, and combining multiple requirements in a single statement.
- Ensure that requirements are necessary, appropriate, complete, feasible, and verifiable.
- Include attributes for each requirement, such as identification, owner, priority, risk, rationale, difficulty, and type (functional/non-functional).

Prioritising requirements is another crucial task, especially in projects with numerous requirements. Some methods mentioned by Khan et al. (2015) include using a low to high priority, assigning a numerical value within a specific range or MoSCoW, which classifies requirements into four categories:

- Must have — must be implemented in the software before being released

- Should have — important but not necessary for the software to be released
- Could have — desirable but not necessary for the software to be released
- Won't have — requirements that are not included in the current release

Requirements in Agile

In Agile projects, requirements are written in the form of User Stories, which are simple descriptions of a feature desired by the customer, using a specific format: 'As a [user], I want to [action] so that [benefit]' (Laplante, 2019, p. 191). Components of a User Story include a title, acceptance criteria, priority, story points and description. Epics are requirements that cannot be completed in a single sprint and can be broken down into user stories. Epics and User Stories are part of the Product and Sprint backlogs, which contain the requirements for the whole project and the current sprint, respectively.

3.2.2 Stakeholders

Stakeholders are the individuals who have some interest in the success of the system or project, thus it is important to identify all possible stakeholders in the early stages of the project to avoid missing important requirements or constraints (Laplante, 2019, p. 34).

Stakeholder analysis can help understand their position within the project. One way of doing it is by using a stakeholder matrix, such as the Influence/Interest grid (see figure 3.4), which classifies stakeholders based on their influence and interest in the project (Morphy, 2020; Reddi, 2023).

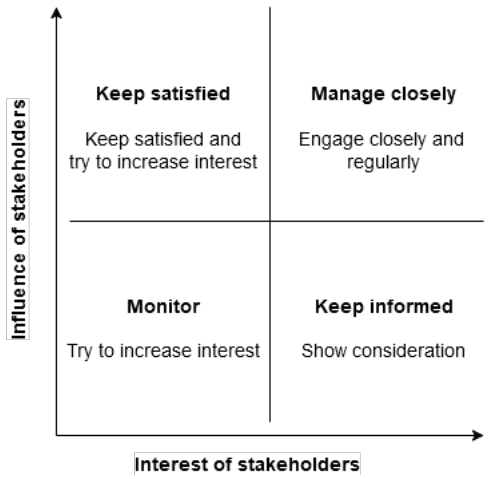


Figure 3.4: Stakeholder Influence/Interest matrix

3.2.3 Requirement gathering techniques

Multiple studies mention the most popular requirement gathering techniques are interviews, workshops, prototyping, modelling, brainstorming, storyboards and observing users (Hickey and A.M. Davis, 2003; Young, 2002; S. Sharma and Pandey, 2013; Tiwari, Rathore, and Gupta, 2012). In one of them, individuals with multiple years of experience in requirement gathering were interviewed, and the authors found that the most used techniques were collaborative meetings, interviews, ethnography and modelling (Hickey and A.M. Davis, 2003).

Multiple research papers recognise interviews as the most common technique for requirement gathering (Alan Davis et al., 2006b; Donati et al., 2017; Bano et al., 2019). Some studies have looked at best practices and common mistakes when conducting requirement gathering interviews. The recommended practices, based on Mohedas et al. (2022) and Loweth et al. (2021), and the common mistakes, from Donati et al. (2017) and Bano et al. (2019), are summarised in table 3.1.

Common Mistakes	**Recommended Practices**
Wrong opening: failing to understand the context before discussing the problem.	State goals at the beginning and allow customer input at the end.‘ ’
Not leveraging ambiguity to reveal knowledge gaps.	Avoid ambiguity by asking clarifying questions.‘ ’
Lack of planning: unstructured sequence of questions.	Plan interviews with a structured sequence of questions.‘ ’
Failing to build rapport with the customer.	Building rapport through small talk or personal questions in the beginning.‘ ’
Implicit goals: failing to ask or clarify stakeholder goals.	Verify alignment and current interpretation with the customer's vision.‘ ’
Question omission: not asking about business processes or doing follow-up questions.	Be flexible by probing into relevant topics.‘ ’
Weak communication: too much technical jargon usage or not listening to the customer.	Use projective techniques like scenarios to encourage deep thinking.‘ ’
Poor question formulation: vague, technical, irrelevant, or too long.	Break down questions or responses into smaller parts and use story telling.‘ ’
Wrong closing sequence: skipping interview summaries or feedback.	Leaving time at the end for the stakeholder to offer any feedback or thoughts.‘ ’

Table 3.1: Comparison of Common Mistakes and Recommended Practices

3.3 Application Tech Stack

3.3.1 Database types

There are several types of databases, such as: relational (SQL), NoSQL databases, graph databases or object-oriented databases (Oracle, 2020). The choice of database depends on the project or organisation requirements, such as the amount of data, the complexity of the data, the need for scalability, etc.

Relational databases

Relational databases store structured data in tables, linked through keys to create relationships between entries (Anderson and Nicholson, 2022). They use SQL (Structured Query Language) to create queries and schemas to help manage data efficiently. Anderson and Nicholson (2022) highlight that relational databases are used, thanks to their high data integrity, for industries like finance and healthcare. Relational databases are widely used, making it easier to find support and resources. However, the rigid schema limits adaptability to rapid data changes or usage of unstructured data. Examples include MySQL, PostgreSQL, Oracle, and Microsoft SQL Server.

NoSQL databases

NoSQL databases manage unstructured or semi-structured data without rigid schemas or relationships (Anderson and Nicholson, 2022). As the authors describe, NoSQL databases, such as key-value, document, column-family, and graph databases, excel in flexibility and scalability. NoSQL databases often prioritise performance over strict consistency, making them suitable for large datasets of unstructured data but less ideal for complex transactions. NoSQL systems lack SQL's mature standardisation and support. Examples include MongoDB, Cassandra, Couchbase, and Redis.

3.3.2 Backend frameworks

Choosing the right backend framework is crucial — it is responsible for handling the business logic of the application, such as processing requests, interacting with the database, and returning responses to the client. A good framework also comes with the added benefit of included features such as security and authentication, database support, a big community and documentation.

Based on recent a recent survey by Vailshery (2024), the most popular backend frameworks are Express, Flask, Spring Boot, Django and Laravel. This data is also supported by the

Stack Overflow Developer Survey 2024, which lists the most popular programming languages as JavaScript (Express), Python (Flask, Django), Java (Spring Boot) and PHP (Laravel) (Stack Overflow, 2024).

Due to their high popularity among developers, the above frameworks will be compared. Using the information from Broadcom (2024), OpenJS Foundation (n.d.), Django (2024), and FastAPI (2024), table 3.2 will compare the frameworks based on the following criteria: programming language used, learning curve, community support, security features, database features, and project size suitability.

Framework	**Django**	**FastAPI**	**Spring Boot**	**Express**
Language	Python	Python	Java	JavaScript
Learning curve	Medium	Low	High	Low
Community Support	High	High	High	High
Security features	High	High	High	Medium
Database features	Medium	Low	High	Medium
Project size suitability	Small to medium	Small to medium	Medium to large	Small to medium

Table 3.2: Comparison of backend frameworks

3.3.3 Frontend frameworks

Similar to the backend frameworks, choosing a suitable frontend framework is equally important — it is responsible for the user interface of the application, such as displaying data, handling user interactions, and making requests to the backend.

Based on the same survey by Vailshery (2024), the most popular frontend frameworks are React, Angular, Vue.js, and Svelte. This data is also supported by the Stack Overflow Developer Survey 2024, where those frameworks rank among the highest for desirability and admirability among developers (Stack Overflow, 2024).

Table 3.3 will use the information from Meta Platforms (n.d.), Google (2024b), You (2024), and Svelte (n.d.) to compare the frameworks based on the following criteria: learning curve, community and documentation, ecosystem and tooling support, performance, state management, and project size suitability.

Framework	React	Angular	Vue.js	Svelte
Learning curve	Low	High	Medium	Low
Community and documentation	High	High	High	Medium
Ecosystem and tooling support	High	High	Medium	Low
Performance	High	Medium	Medium	High
State management	High	High	Medium	Low
Project size suitability	Small to large	Medium to large	Small to medium	Small to medium

Table 3.3: Comparison of frontend frameworks

3.4 Large Language Models (LLMs)

LLMs are artificial intelligence systems that are used for NLP tasks such as text generation, translation, summarisation and question answering (Naveed et al., 2023; Nazi and Peng, 2024). Additionally, LLMs have been found to have emergent capabilities, like reasoning, planning, decision-making and in-contenxt learning (Naveed et al., 2023). These extraordinary capabilities are achieved through extensive training on large corpus of text data, high parameter count (in the billions) and usage of techniques such as fine-tuning or prompt engineering to improve their performance (Naveed et al., 2023; Nazi and Peng, 2024).

LLMs are built on the transformer architecture, which allows them to understand text by learning and remembering the relationships between words (Vaswani et al., 2017). These models are first pre-trained on large amounts of unlabeled data using, allowing them to excel in a wide variety of tasks (Zhou et al., 2023; Naveed et al., 2023). These pre-trained models, known as foundation models such as the GPT or Llama families, can then be fine-tuned for specific tasks, improving their performance and accuracy even further (OpenAI et al., 2024; Dubey et al., 2024; Naveed et al., 2023).

3.4.1 Multimodal LLMs

One advancement in the field of LLMs has been the addition of multimodal abilities, allowing them to process, understand and generate text and images, audio or videos (Yin et al., 2024; D. Zhang et al., 2024). These new multimodal LLMs utilise existing reasoning capabilities of LLMs, which are connected to an encoder that can processes images, audio or videos and a generator that helps with generating multimodal outputs (Yin et al., 2024). This integration of new modalities allows MLLMs to become versatile tools, expanding their possible use cases and bridging the gap between human and machine interaction (Nazi and Peng, 2024).

API model providers

Running and hosting LLMs locally can be a challenge, considering their big model sizes and high computational requirements. As such, many platforms offer APIs that allow users to access LLMs through the cloud. A list of some free API providers, the models offered and their rate limits has been compiled by Cheah and Markham (2024) and some are listed in table 3.4.

Provider	**Model name(s)**	**Free tier limits**
Groq	Llama 3.2 11B Vision	7,000 requests/day, 7,000 tokens/minute
	Llama 3.2 90B Vision	3,500 requests/day, 7,000 tokens/minute
OpenRouter	Llama 3.2 11B Vision Instruct Llama 3.2 90B Vision Instruct Gemini 2.0 Flash Experimental	20 requests/minute, 200 requests/day
Google AI Studio	Gemini 2.0 Flash	4,000,000 tokens/minute, 10 requests/minute
	Gemini 1.5 Flash	1,000,000 tokens/minute, 1,500 requests/day, 15 requests/minute
	Gemini 1.5 Pro	32,000 tokens/minute, 50 requests/day, 2 requests/minute
GitHub Models	OpenAI GPT-4o OpenAI GPT-4o mini	Rate limits dependent on Copilot subscription tier
Cloudflare Workers AI	Llama 3.2 11B Vision Instruct	10,000 tokens/day
glhf.chat	Any model on Hugging Face that fits on an A100 node (640GB VRAM)	480 requests/8 hours

Table 3.4: API providers for LLMs

3.4.2 LLMs in healthcare

One application of MLLMs is in healthcare, where the growing volume and complexity of data creates the need for more advanced tools to process and analyse it. LLMs and MLLMs have found use in various healthcare applications, either by using existing models or by developing new, specialised medical models such as Med-PaLm2, BioMistral or Med-Gemini (Labrak et al., 2024; Google, 2024a; Singhal et al., 2023). Some of these applications include:

- **Improving medical diagnosis:** By combining patient records, existing symptoms, and medical history, LLMs can use their reasoning capabilities and memory to assist in diagnosing or preventing health conditions (Nazi and Peng, 2024; Niu et al., 2024; Xiao et al., 2024).
- **Medical Imaging and Multimodal Capabilities:** In diagnostic imaging, multimodal models can assess both text and images (such as X-rays and MRIs) to offer comprehensive analysis. Clinicians can input medical images and contextual information, making MLLMs valuable assistants in the real-time diagnostic processes (Niu et al., 2024).
- **Virtual Health Assistants:** LLMs can also be deployed as virtual assistants, helping patients with personalised care and general health inquiries (Nazi and Peng, 2024; Niu et al., 2024). Patients in areas with limited healthcare access can benefit from these assistants, which also supports healthcare providers by lightening their workloads.
- **Administrative Support:** LLMs can assist in generating EHRs, allowing healthcare providers to focus more on patient interaction (Xiao et al., 2024). Additionally, they can also help translate complex medical terms into more simple language, assist in administrative tasks, and more.

3.4.3 Prompt engineering

The success of LLMs depends not only on the model itself — but also on how it's effectively used by the users, using techniques like prompt engineering, which involves the constant designing and refining of prompts to guide the output of LLMs (Meskó, 2023; Amatriain, 2024). Prompts represent instructions given to the model to guide its output, such as providing context, examples, or constraints to the model (Giray, 2023; Schulhoff et al., 2024; Amatriain, 2024).

There are multiple techniques for prompt engineering, ranging from simple to more advanced. The tables and subsections below outlines some of the most common techniques.

Zero-Shot Prompting

Zero-shot prompting are techniques where the LLM is given a prompt without any examples, allowing it to generate an output based on the prompt alone (Schulhoff et al., 2024).

Technique	Description	Source
Role prompting	Assigning a specific role to the LLM in the prompt. The authors note that generally it provides mixed results but may be useful in certain settings.	Zheng, Pei, and Jurgens (2023)
Style prompting	Specifying the desired style or tone in the prompt.	Lu et al. (2023)
Emotion prompting	Incorporating phrases of psychological relevance to humans in the prompt.	Cheng Li et al. (2023)
Re-reading	Adding the phrase 'Read the question again' to the prompt in addition to repeating the question.	Xu et al. (2023)
Self-Ask	Prompting the LLM to decide if it needs to ask any follow-up questions for a given prompt.	Press et al. (2022)

Table 3.5: Zero-Shot Prompt Techniques

Few-Shot Prompting

Few-shot prompting are techniques where the LLM learns how to complete a task based on a few examples given in the input prompt (Schulhoff et al., 2024).

Technique	Description	Source
Self-Generated In-Context Learning	Using the LLM to automatically generate examples when training/example data is not avaialable.	Kim et al. (2022)
Prompt Mining	Scanning the training data to discover common formats that can be used as prompt templates.	Jiang et al. (2020)

Table 3.6: Few-Shot Prompt Techniques

Multimodal and Multilingual Prompting

Multimodal and multilingual prompting are techniques which aim to improve an LLM's performance by leveraging multiple modalities or languages in the prompt (Schulhoff et al.,

2024).

Technique	Description	Source
Translate-first prompting	Translating the input prompt into English to leverage LLMs strengths in dealing with English inputs, compared to non-English inputs.	Etxaniz et al. (2023)
English prompting	Writing the prompt in English may usually be more effective than using the task language for multilingual tasks. The authors argue it may because of the predominance of the English language in the pretraining data.	Ahuja et al. (2023)
JSON/XML output formatting	Asking the LLM to format the response in a JSON or XML format and providing the expected schema has been found to improve the accuracy of LLM outputs	Hada et al. (2023)
Image-as-Text	Generating or writing a textual description of an image that can then be included in a text-based prompt.	Hakimov and Schlangen (2023)

Table 3.7: Multimodal and Multilingual Techniques

3.4.4 Challenges and concerns of using LLMs

While LLMs bring many benefits when applied to the healthcare domain, it is important to note that their use does come with several challenges:

- **Data Privacy and Compliance:** Patient data is highly sensitive, thus ensuring compliance with standards is essential, requiring data anonymisation and secure handling practices to ensure patient data safety.(Nazi and Peng, 2024; Harrer, 2023; Xiao et al., 2024).
- **Transparency and Explainability:** LLMs are often described as 'black boxes', making it difficult to explain their decision-making processes. In healthcare, transparency is crucial — lack of it poses risks and raises ethical concerns about relying on such systems in high-stakes scenarios (Nazi and Peng, 2024; Harrer, 2023; Xiao et al., 2024)..
- **Bias and Fairness:** LLMs trained on vast datasets can inherit biases in the data, leading to skewed or unfair outcomes (Harrer, 2023).

- **Hallucinations:** LLMs sometimes generate false or fabricated outputs, also known as 'hallucinations'. In healthcare, this poses significant risks, as incorrect or misleading information could jeopardize patient safety and trust in the technology (Xiao et al., 2024; Nazi and Peng, 2024).
- **Accountability:** Responsibility must be clearly communicated and understood by all parties involved in the development and use of the model (Harrer, 2023). The author recommends the usage of clear guidelines, policies and code of conducts to ensure that all parties are aware of their obligations.
- **High Costs and Infrastructure Needs:** Training and operating LLMs requires extensive computational resources, which can be a limiting factor for healthcare institutions (Xiao et al., 2024).

3.5 PHR Systems

PHR is an electronic resource used by patients to manage their own health information (Hosseini et al., 2023; Lee et al., 2021). PHRs are different from EHRs and EMRs which are inter-organisational or internal systems to organise patient health records (Heart, Ben-Assuli, and Shabtai, 2017; Lee et al., 2021). Three different types of PHRs are described by Hosseini et al. (2023): stand-alone, which require manual entry to update the records; instituion-specific, which are connected to a specific healthcare institution; and integrated, which can connect to multiple healthcare systems to aggregate data from multiple sources.

Usage of PHRs can bring many benefits to patients, such as: empowering patients to manage their health, improving patient outcomes, decreasing the cost of healthcare and improving the taking of medication (Hosseini et al., 2023).

PHRs contain highly sensitive health information, so it is important to ensure that the data is secure and private. Based on a survey of health information management and medical informatics experts, Hosseini et al. (2023) identified 7 dimensions that need to be addressed when developing a PHR system:

1. Confidentiality
2. Availability
3. Integrity
4. Authentication
5. Authorisation

6. Non-repudiation
7. Access rights

The authors recommend mechanisms to ensure adherence to the above-mentioned dimensions, such as encrypting the data in the database, using backups or defining user access to data and access rights.

3.5.1 Existing PHR applications

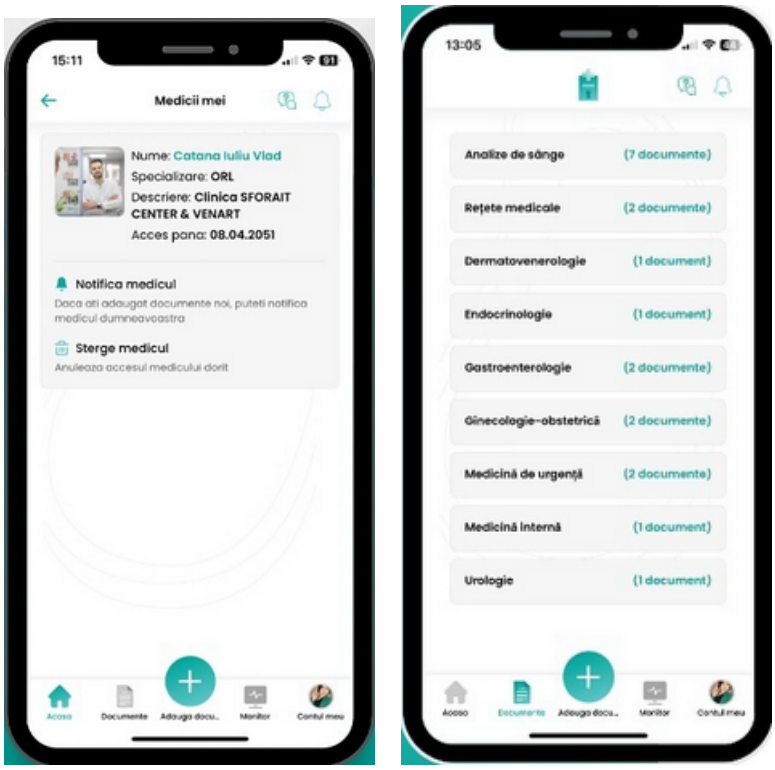
PHR systems have been implemented nation-wide in many developed countries, such as the NHS App in the UK (Lee et al., 2021). Additionally, there are many private solutions that offer similar features to the one proposed in this project. The next sections will provide a brief overview of 3 existing systems: Medvalet, Andaman7 and Fasten Health.

Medvalet

A mobile app developed in Romania that allows patients to upload their medical history as PDFs or scanned documents (Medvalet, 2024). See Table 3.8 for a summary of its features and limitations and figure 3.5 for a screenshot of the app.

Key Features/Benefits	**Limitations/Drawbacks**
• Upload medical history as PDFs or scanned documents. • Categorise documents by type (e.g., prescriptions, lab results). • Graphically track vitals like blood pressure and weight over time. • Patients can input personal details such as name, age, and weight. • Doctors can access patient history directly via the app.	• Requires doctors to create accounts, which may deter use. • Doctors can access patient history without explicit consent, raising privacy concerns. • App acts as document storage, which can be cumbersome to access for lengthy histories. • Lacks data extraction or summarisation features from uploaded documents. • Only available as a mobile app, limiting accessibility for desktop-only users.

Table 3.8: Medvalet Features and Limitations



(a) My doctors screen (b) Uploaded documents screen

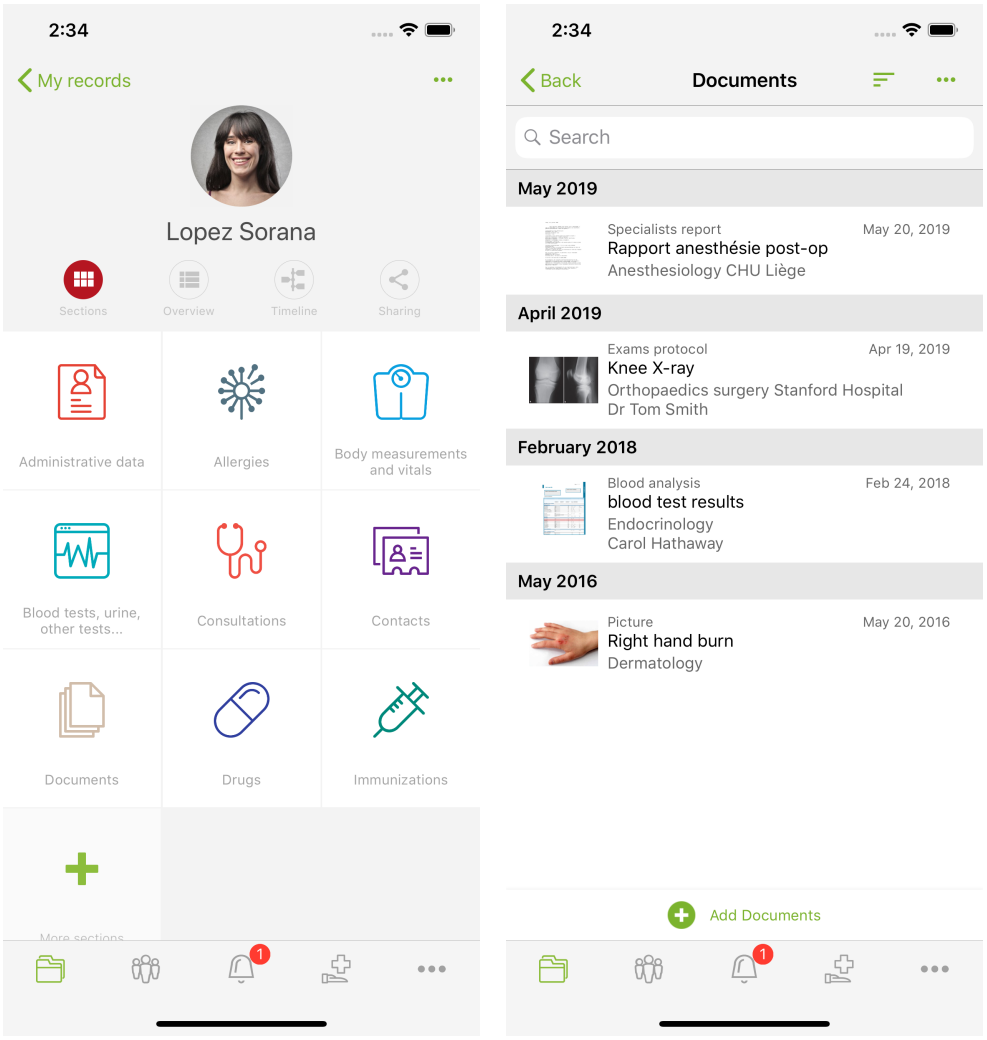
Figure 3.5: Medvalet screenshots

Andaman7

A mobile app developed by a Belgian-American eHealth company with the goal to improve doctor-patient communication, compliant with GDPR and HIPAA (A7 Software, 2022). See Table 3.9 for a summary of its features and limitations and figure 3.6 for a screenshot of the app.

Key Features/Benefits	Limitations/Drawbacks
• Offers sections for personal information, medical history, allergies, vaccinations, medications, etc. • Automatically collects health data from over 300 hospitals and clinics in the US and Europe. • Supports input from diverse sources like hospitals, labs, smart devices or even manual input. • Stores data locally on patients' devices, ensuring privacy. • Data sharing with QR codes and revokable access. • AI tools for summarisation, translation, and simplifying medical jargon.	• Requires patients and doctors to both create accounts. • Does not extract data or values from uploaded documents like lab results. • Limited to mobile platforms, which may limit usability for desktop-only users.

Table 3.9: Andaman7 Features and Limitations



(a) PHR Sections screen

(b) Documents section screen

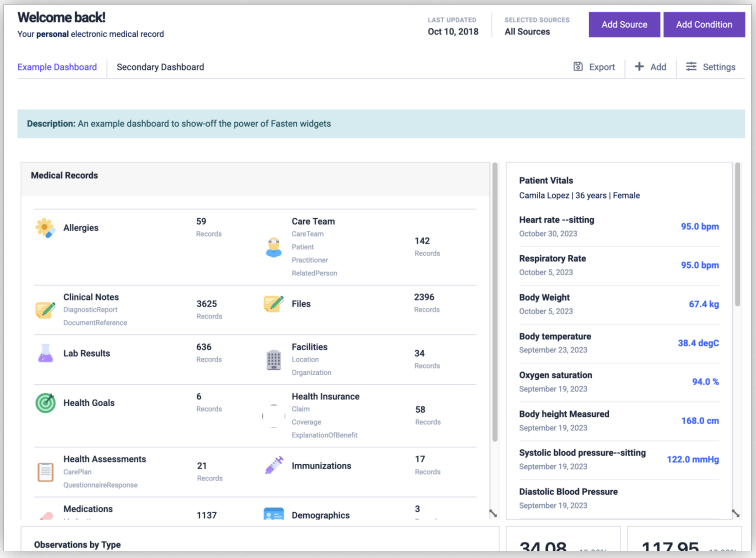
Figure 3.6: Andaman7 screenshots

Fasten Health

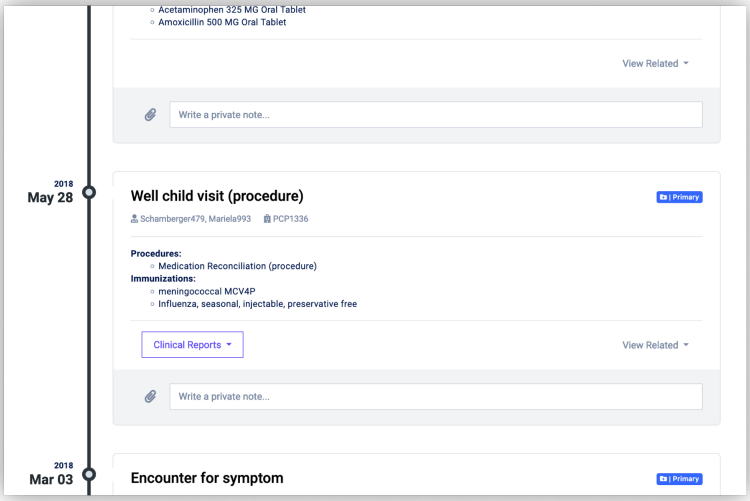
An open-source, self-hosted electronic medical record aggregator with optional paid desktop versions for Windows and Mac (Fasten Health, 2022). See Table 3.10 for a summary of its features and limitations and figure 3.7 for a screenshot of the app.

Key Features/Benefits	**Limitations/Drawbacks**
• Automatically aggregates records from multiple providers, like hospitals and labs. • Supports self-hosting for complete control over data, stored locally. • Compatible with protocols such as DICOM, FHIR, and OAuth2. • Allows manual entry for allergies, vaccinations, and medications. • Offers multiple dashboards with graphs to visualize health data. • Supports multi-user functionality for families.	• Paid desktop versions may deter users. • Manual data entry limited to new or existing encounters, complicating usage. • Does not support OCR or automatic data extraction from documents. • Lacks data-sharing capabilities with doctors. • Requires technical expertise for self-hosting. • Restricted to healthcare providers in the United States.

Table 3.10: Fasten Health Features and Limitations



(a) Dashboard screen



(b) Visit history screen

Figure 3.7: Fasten Health screenshots

Chapter 4

Solution Planning and Design

This chapter will outline the solution planning and system design process, which will include discussions on the chosen project management methodology, the system architecture, the tech stack that will be used during development and the user interface design illustrated by using wireframes.

4.1 UML Diagrams

UML is a standardised modeling language that consists of a set of diagrams used for documenting software systems (Paradigm, 2024). It enables better communication potential of designs and architectural decisions. Common UML diagram types include:

- **Use Case Diagram** — Illustrates the system's intended functionality in terms of actors, use cases, and their relationships. It can be accompanied by a use case specifications document, which provides a description of each use case.
- **Sequence Diagram** — Demonstrates how objects interact in a particular, timed sequence scenario, focusing on the messages passed between objects.
- **Activity Diagram** — Represents the workflow of a target use case or business process through a series of activities, emphasizing steps, choices, iterations, and concurrency.

These diagrams were used to present the stakeholders with a visual representation of the system's design and functionalities. They can be found below, under their respective subsections.

4.1.1 Use Case Diagram

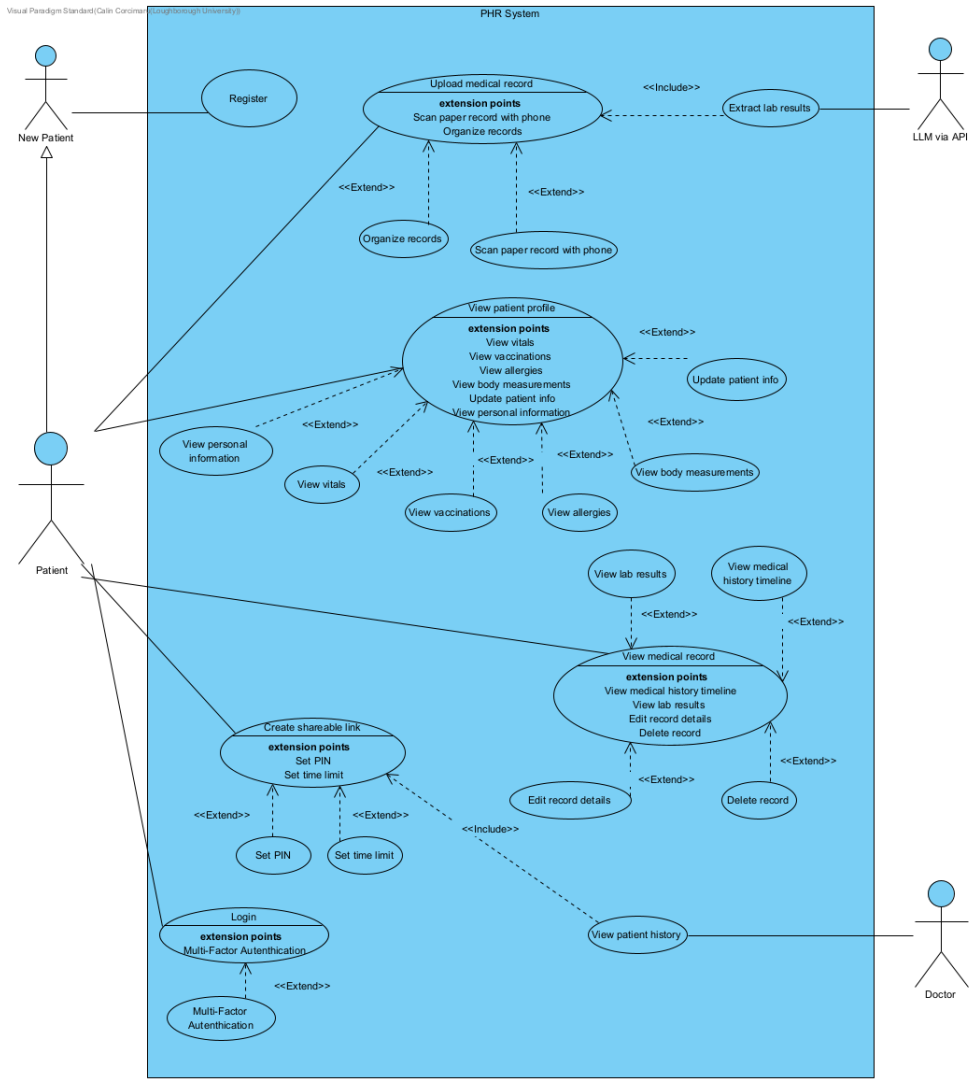


Figure 4.1: UML Use Case Diagram

4.1.2 Use Case Specifications

Login

Description	The Login use case allows the app user to log in to their existing account via their credentials, with an optional use of MFA.
Actors	Patient
Preconditions	Must have an existing account
Steps	1. User enters user and password 2. User clicks on the login button 3. App validates user credentials 4. Patient enters MFA code if enabled 5. App logs user in

Register

Description	The Register use case allows the app user to create a new account.
Actors	New Patient
Preconditions	No existing account with the email used to register
Steps	1. User enters user and password 2. User clicks on the register button 3. App validates user credentials 4. App logs user in 5. App sends verification email 6. User verifies email

Upload Record

Description	The Upload Record use case allows the patient to upload their medical records to the app.
Actors	Patient, LLM
Preconditions	Must be logged in
Steps	1. User selects file upload (or camera scan) 2. User selects the record to upload 3. User clicks on the upload button 4. App validates the record 5. User selects the appropriate record type 6. If record type is lab result, app sends the record to LLM via API for processing into JSON format 7. App adds the record to the database 8. App shows confirmation to user

Share Records

Description	The Share Records use case allows the patient to share their medical records with doctors.
Actors	Patient, Doctor
Preconditions	Must be logged in and have records uploaded
Steps	1. User selects option to create a share link 2. OPTIONAL: User selects the records to share 3. OPTIONAL: User adds a PIN to the share link 4. User selects time limit for the share link 5. App generates the share link 6. App sends the share link to the doctor via email 7. App shows confirmation to user 8. Doctor clicks on the share link 9. Doctor enters the PIN (if required) 10. App validates the PIN 11. App shows the records to the doctor

View Records

Description	The View Records use case allows the patient to view and edit their uploaded medical records.
Actors	Patient
Preconditions	Must be logged in and have records uploaded
Steps	1. App provides a list of records or medical history 2. User selects record to view from list or medical history 3. App retrieves the record from the database 4. App shows the record to the user 5. OPTIONAL: User can view the record in a graphical format if lab result 6. OPTIONAL: User edits the record 7. OPTIONAL: User deletes the record

View Patient Profile

Description	The View Patient Profile use case allows the patient to view and edit their profile information, including health information such as allergies, medications, vaccinations and recorded health data like blood pressure, glucose levels, etc.
Actors	Patient
Preconditions	Must be logged in
Steps	1. User selects the profile section 2. App retrieves the profile information from the database 3. App shows the profile information to the user — vaccinations, allergies, medications, health data 4. OPTIONAL: User edits the profile information 5. OPTIONAL: User adds new health data 6. OPTIONAL: User deletes health data 7. OPTIONAL: User can view health data in a graphical format

4.1.3 Sequence Diagrams

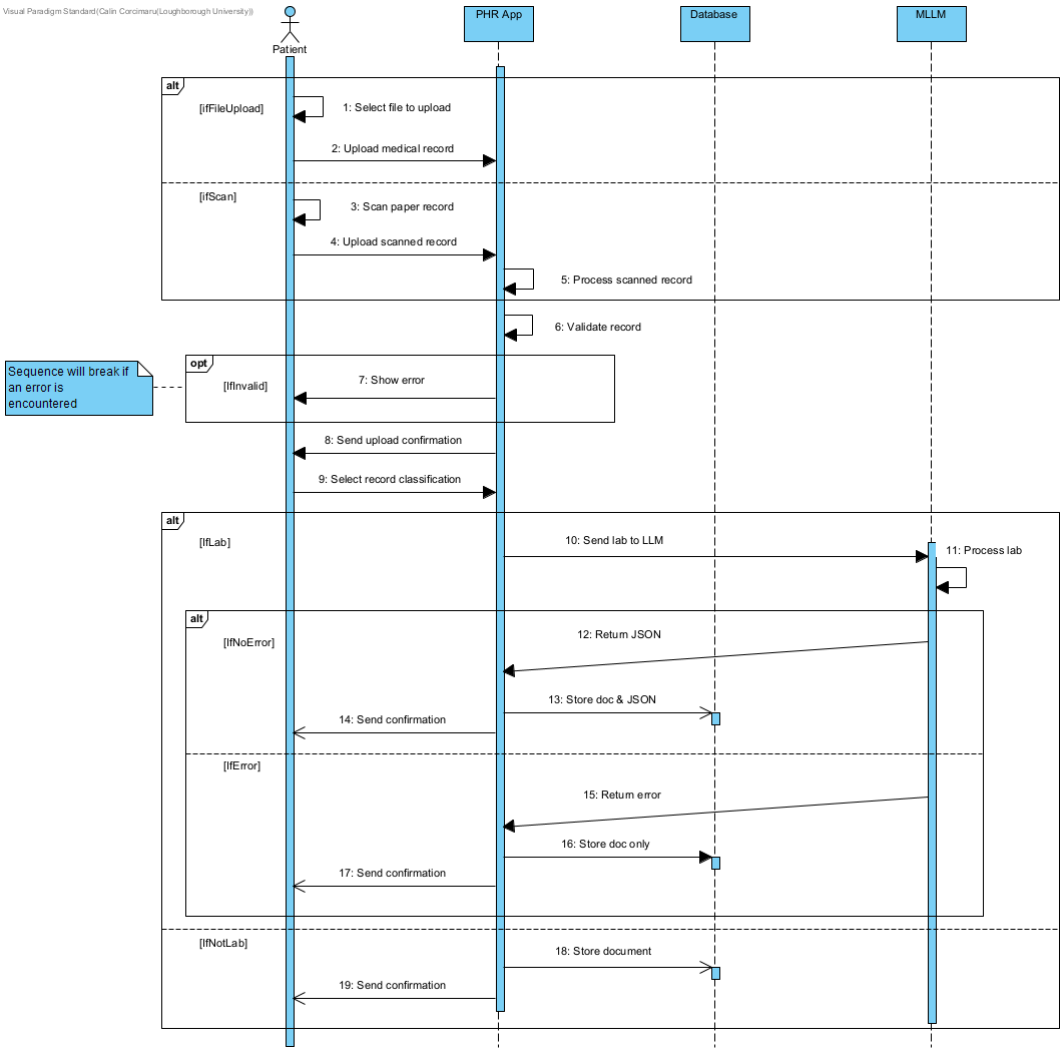


Figure 4.2: UML Sequence Diagram — Upload Record Use Case

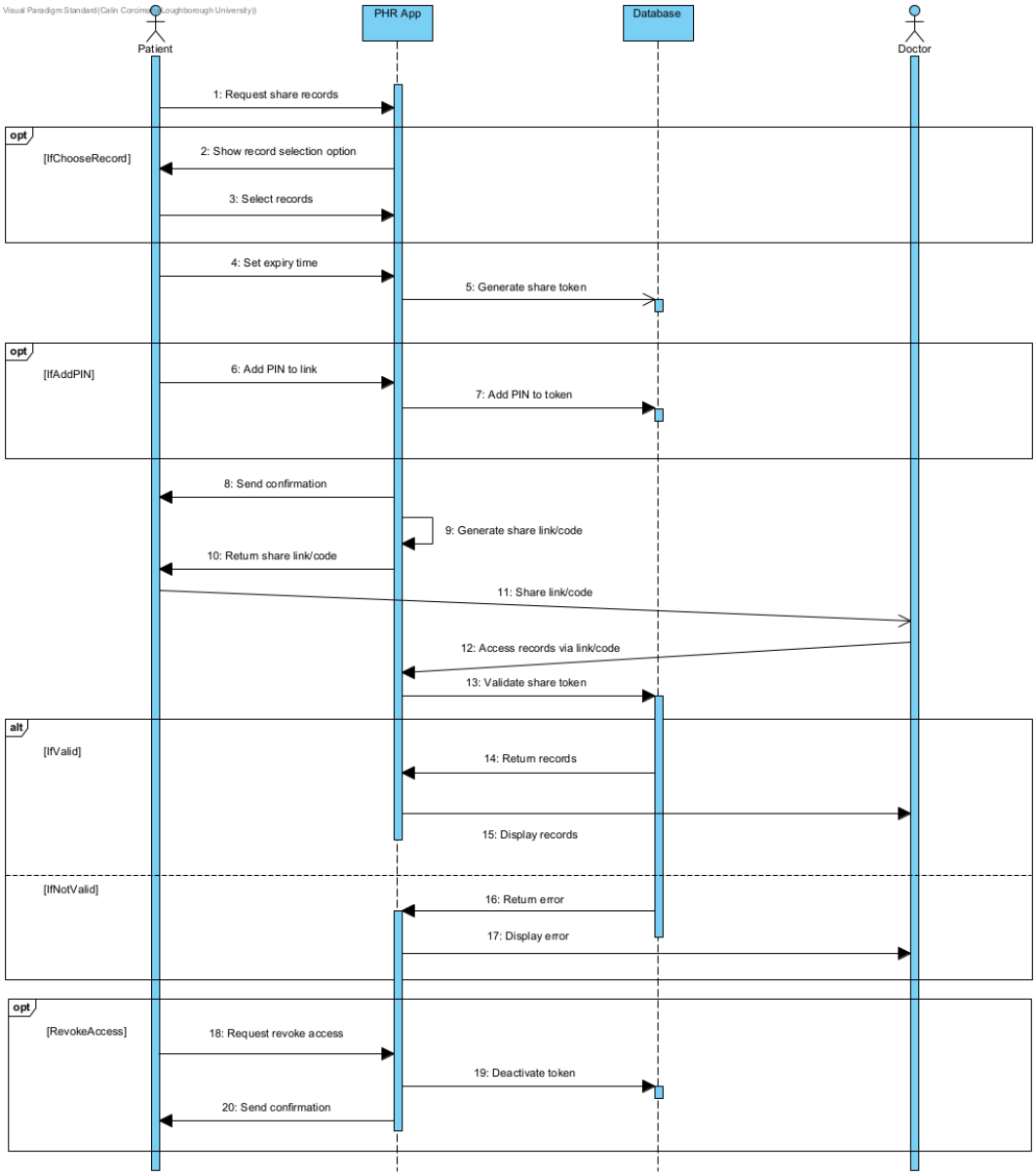


Figure 4.3: UML Sequence Diagram — Share Records Use Case

4.1.4 Activity Diagrams

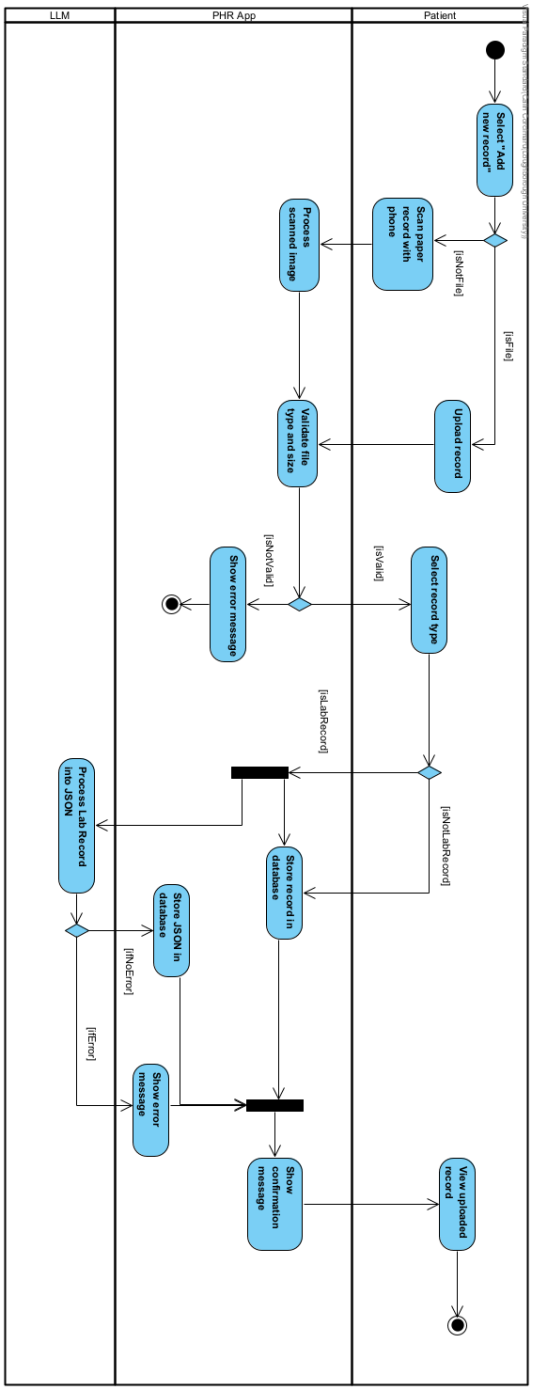


Figure 4.4: UML Activity Diagram - Upload Record Use Case

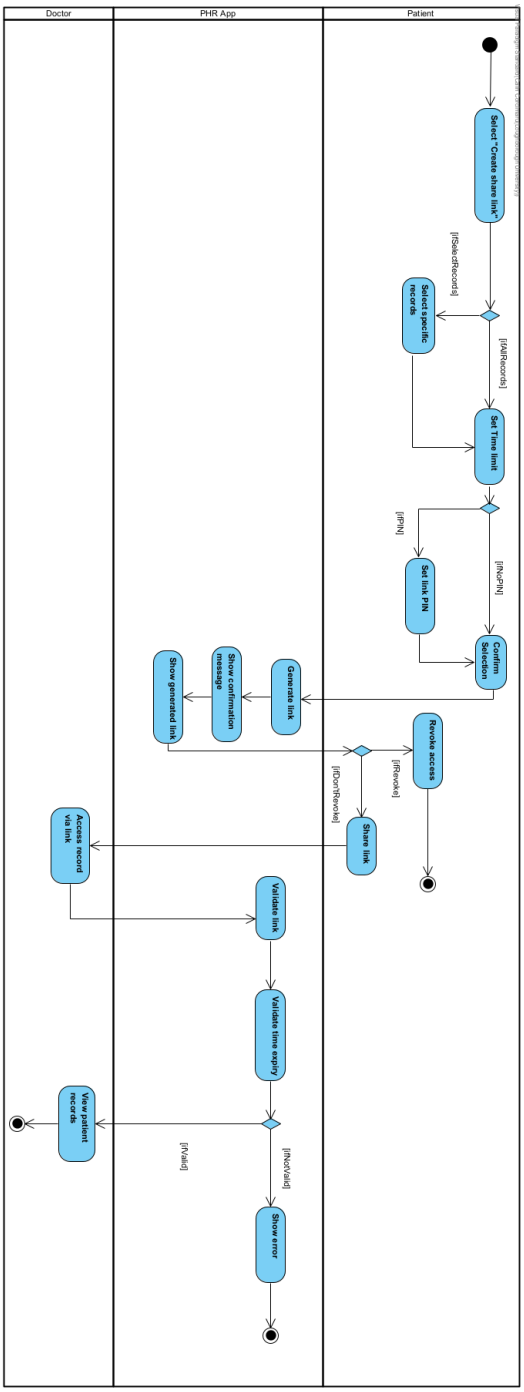


Figure 4.5: UML Activity Diagram - Share Records Use Case

4.2 Database Design

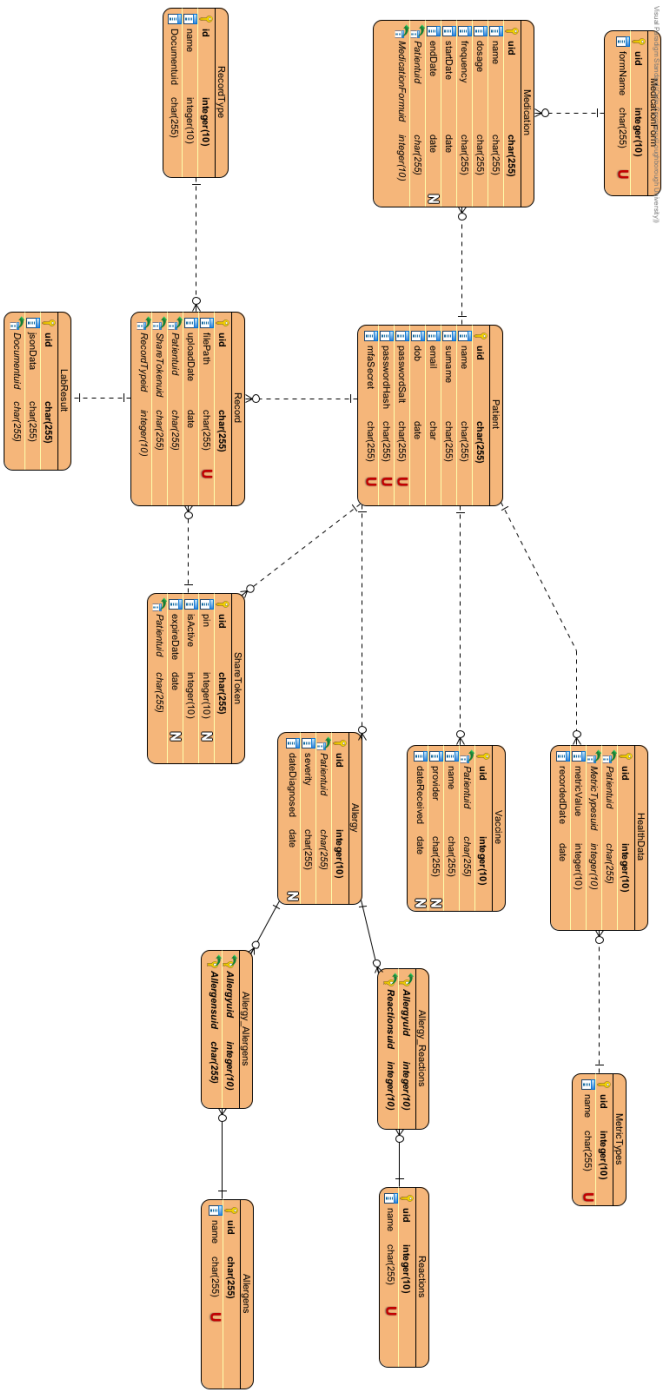
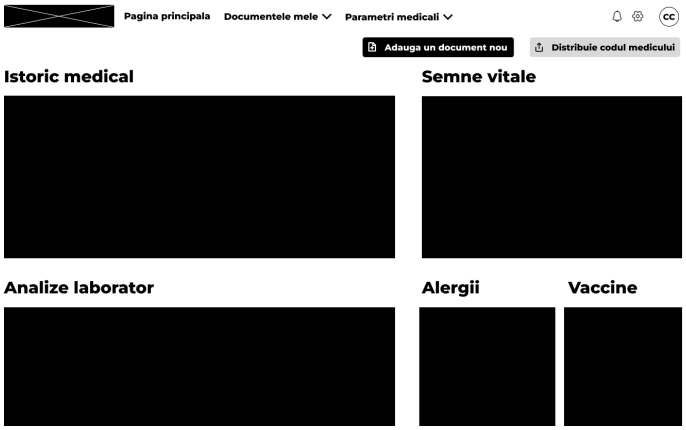


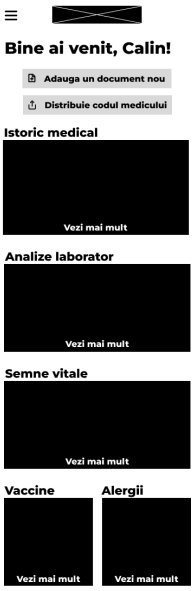
Figure 4.6: Entity Relationship Diagram

4.3 Wireframes

Wireframes were used to provide a high-level overview of the application's design and present some of its key functionalities. These have been created using Figma and presented to key stakeholders during virtual meetings. Based on the feedback received, the wireframes have been adjusted and some examples can be seen in figures 4.7, 4.8 and 4.9. The full set of wireframes can be found in appendix B.



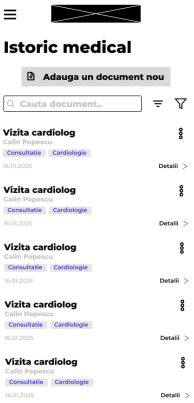
(a) Desktop version



(b) Mobile version

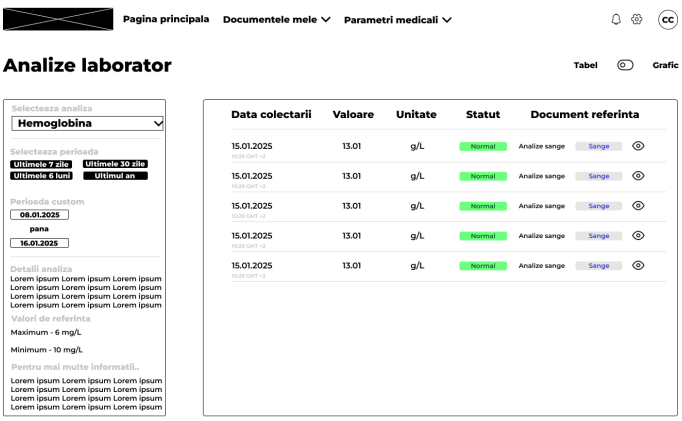
Figure 4.7: Desktop and Mobile version of the Dashboard screen

(a) Desktop version



(b) Mobile version

Figure 4.8: Desktop and Mobile version of the Medical History screen



(a) Desktop version

(b) Mobile version

Figure 4.9: Desktop and Mobile version of the Lab Test Results screen

4.4 Project Tech Stack

Based on the research conducted in sections 3.3.1, 3.3.2 and 3.3.3, the following tech stack was chosen for the project, which can be seen in diagram 4.10.

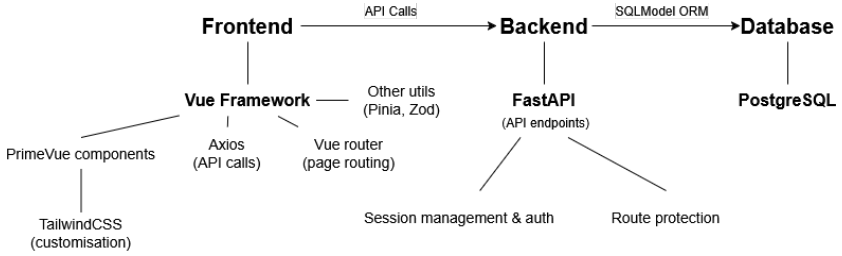


Figure 4.10: Proposed System Architecture

4.4.1 Frontend

Vue was chosen for the frontend, as it is a JavaScript framework used for building SPAs known for its ease of use and flexibility (You, 2024). It is powered by components, element reactivity and SFCs that enable HTML, CSS and JS to be used in a single `.vue` file. Vue also boasts a large and active community, with many libraries and guides available for developers. Previous experience with Vue made it a suitable choice for this project. Diagram 4.11 showcases how different elements of the frontend interact. The next subsections will briefly discuss each.

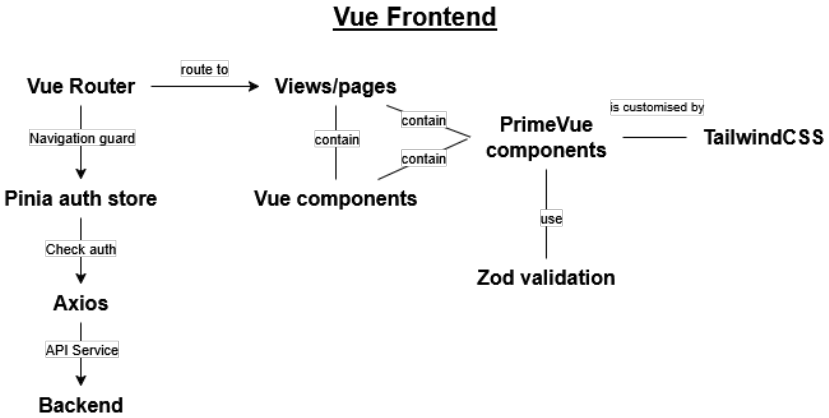


Figure 4.11: Interaction of Frontend Components

Vue Router

Vue Router is the official client-side router for Vue.js, which allows the user to navigate between application pages without a full reload (Vue Router, n.d.). It makes full use of the SPA capabilities of Vue, providing a seamless user experience. Vue Router can also be used to create navigation guards to protect certain routes from unauthorized access, which is crucial for the security of the application.

Pinia

Pinia is a store library for Vue made by the same team (Pinia, n.d.). Its main functionality is to provide a way to share states across Vue components and pages. In this project, Pinia will be used to store the user's authentication status and other global states that need to be shared within the application.

Axios

Axios is a promise-based HTTP client for the browser and Node.js (Axios, n.d.). It is used to make HTTP requests to the backend. Similarly, it allows for automatic interaction with cookies in requests and responses, which is important for API communication.

Vue views and components

Vue components and views are the foundation of the frontend. Views represent the different pages of the application, which can be navigated to using the Vue Router. Components allow to break down the UI into independent and reusable elements (Vue.js, n.d.). Each component can be stored in a separate `.vue` file and imported when necessary.

PrimeVue and TailwindCSS

PrimeVue is a component library for Vue, which provides a set of pre-built components that can be used to easily create the frontend of the application (PrimeTek, n.d.). TailwindCSS is a CSS framework that provides a multitude of utility classes that can be applied to style application elements (Tailwind Labs Inc., 2025). When combined, PrimeVue provides the basic building blocks such as buttons, forms, and tables, while TailwindCSS styles these components.

4.4.2 Backend

FastAPI was chosen as the backend framework. It is a modern framework used in building APIs with Python, chosen due to its ease of use, simplicity and heaps of documentation

available (FastAPI, 2024). Previous experience with Python made this framework an accessible choice for this project. A separate backend would also allow for future flexibility — its independence from the frontend enables it to be re-used for other platforms, such as mobile. Diagram 4.12 shows how different components of the backend will interact with each other.

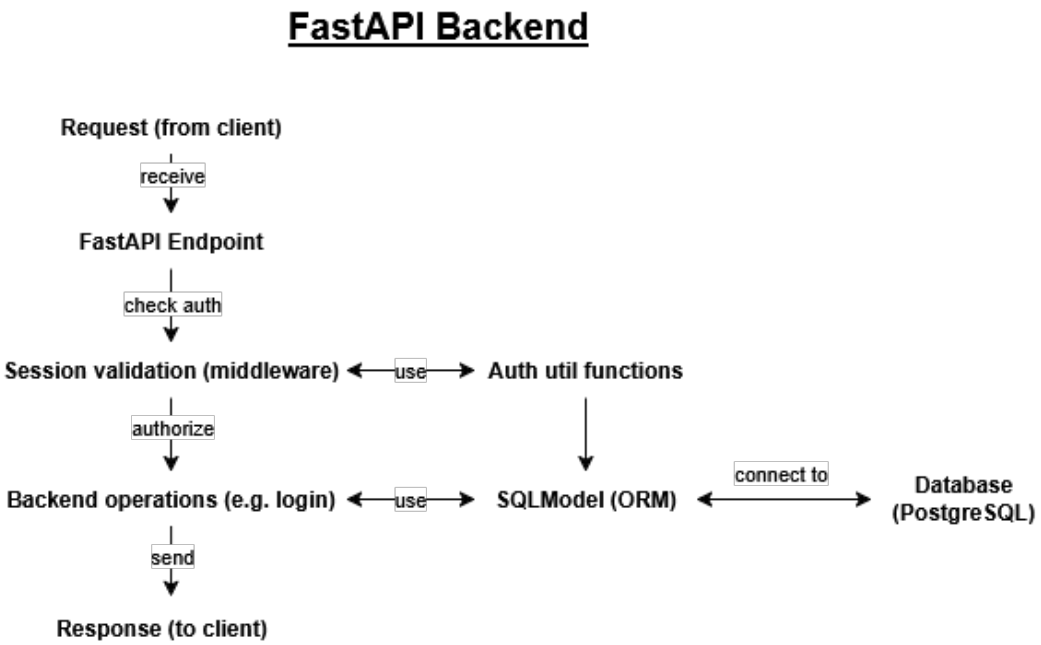


Figure 4.12: Interaction of Backend Components

4.4.3 Database

PostgreSQL was the database of choice, which is a relational database system that boasts a good reputation for its reliability, wide feature set and performance (PostgreSQL Global Development Group, 2025). It supports a wide range of data types, including JSON and UUID, which are important for this project. Similarly, PostgreSQL offers robust security features like an access-control system or column and row-level security.

SQLModel, a Python library built on top of SQLAlchemy and Pydantic, will be used to interact with the database. SQLModel provides a way to define database tables using classes, which are then used to interact with the database (SQLModel, n.d.).

4.5 Project Management

4.5.1 Methodology and Tools

Based on the research in section 3.1, a hybrid project management approach has been adopted, with Waterfall as the main methodology covering the initial phases of the project that transitions to Agile during the development phase.

1. **The nature of the project:** The limited timeframe and resources required using a flexible approach to ensure a viable prototype is delivered.
2. **Documentation requirements:** The academic nature of this project calls for extensive documentation of requirements, design choices and implementation decisions.
3. **Stakeholder engagement:** The development will require continuous stakeholder feedback to ensure the solution meets stakeholder needs.

Notion will be used as the project management tool. It allows for easy task management and documentation, providing a wide range of templates for Gantt charts, user stories, backlogs, sprints and Kanban boards (Notion Labs, 2025).

4.5.2 Sprint planning

Following the completion of the initial project phases, the development phase will be divided into 6 sprints. Each sprint will last 2 weeks, with a total of 12 weeks allocated. The sprints will be planned as follows:

- **Sprint 1** — Will configure the development environment, frameworks, database and implement core functionalities like user authentication and registration.
- **Sprint 2** — Will implement the vaccines, allergies, medications and health data sections alongside the a basic layout of the dashboard.
- **Sprint 3** — Will complete the main dashboard and implement the document upload feature alongside the medical history section.
- **Sprint 4** — Will integrate the MLLM and implement the automated lab test extraction feature.
- **Sprint 5** — Will develop the record sharing feature via a shareable link, as well as the doctor view of the records.
- **Sprint 6** — Will address any identified remaining gaps, such as missing features, as well as final testing processes.

Chapter 5

Development

This chapter discussses the development process of the project, structured on a sprint-by-sprint basis. Each sprint section will outline the key features implemented, challenges faced and the requirements completed in each sprint.

5.1 Sprint #1

The first sprint focused on establishing the foundation for this system. Its main objectives involved setting up the database, creating the authentication API endpoints, basic authentication flow and the frontend of the application.

5.1.1 SQLModel Models & Database creation

```
    class DateFormattingModel(SQLModel):
        dob: date | None = None
        @field_serialiser('dob')
        def serialise_dob(self, value: date) -> str:
            return value.strftime("%d-%m-%Y")
    class User(DateFormattingModel, table=True):
        id: uuid.UUID = Field(default_factory=uuid.uuid4, primary_key=
    True)
        name: str
        email: EmailStr = Field(index=True, unique=True)
        hashed_password: str
```

Listing 5.1: SQLModel User Model

SQLModel played a pivotal role in creating the database tables. The code above is an example of a model that was used to create the User table in the database. It was also used to create specific models for the endpoints to validate the incoming or outgoing data. The following snippet is an example of a model that was used to validate the data that was being sent to the register endpoint.

```
    class UserAuth(SQLModel):
        email: EmailStr
        password: str
        name: str | None = None # Will only be used for registration
```

Listing 5.2: SQLModel Authenticatio Model

5.1.2 Authentication & APIs

As the application would handle sensitive health data, its security was determined as an important concern for both the system and its users. An evaluation of modern authentication methods was conducted, identifying three possible approaches:

- JWT Tokens
- Session-based authentication
- Using 3rd party authentication services like Auth0

JWT (JSON Web Tokens) authentication involves generating tokens containing user information for client-side storage (Akanksha and Chaturvedi, 2022). Its main advantages are it statelesness and minimal server-side management. However, tokens can be vulnerable to XSS and CSRF attacks and cannot be invalidated once created (Akanksha and Chaturvedi, 2022; Papathanasaki, Maglaras, and Ayres, 2022). The authors recommend using methods like pairs of access and refresh tokens to mitigate these risks, or to store the tokens inside cookies.

An alternative, session-based authentication, stores session information server-side, using cookies to maintain the sessions (Akanksha and Chaturvedi, 2022; Papathanasaki, Maglaras, and Ayres, 2022). While facing similar security risks as JWT, they can be secured through flags like HttpOnly and SameSite (Mozilla, 2025). Sessions can be invalidated server-side, providing better security control.

Third-party authentication options include OAuth2 through providers like Google (Papathanasaki, Maglaras, and Ayres, 2022; Okta, Inc., 2025b) or services like Auth0 (Okta, Inc., 2025a). While these offer quick implementation, they may introduce dependencies or security concerns due to third-party data storage.

Consideration was also given to Mpass, which is a Single Sign On (SSO) authentication system that offers a secure and easy way to access electronic government services in Moldova (Agenția de guvernare electronică, 2025). While it may not be possible to currently integrate MPass into the system, it will be documented as a possible future integration, which will be discussed in the 7.2 section.

Following the evaluation of authentication methods, session was selected based on its ease of implementation and balanced level of security. The ability to control the session on the server side was also an important factor, as it allows the system to safely log out users or invalidate sessions due to inactivity or suspicious activity.

The implementation used a Session database table containing a UUID, which was passed to the user via cookies. Upon succesful authentication, the backend would configure the cookie and use it with every subsequent backend request. To ensure safety from common attacks such as Cross-Site Scripting (XSS) and Cross-Site Request Forgery (CSRF), best practices from Cheat Sheets Series Team, 2025; Mozilla, 2025 were used to configure the cookies:

```
    # Set the session cookie in the response and send it to the client
    response.set_cookie(
        "session_id",
        str(session_id), # Using str() to convert the UUID to a string
        httponly=True,
        max_age=3600, # 1 hour
        samesite="strict",
        secure=True)
```

Listing 5.3: Session Cookie Flags

The system's API endpoints were protected through a session validation middleware implemented via FastAPI's dependency injection system, which would check if the user was authenticated before processing the request. An example of this can be seen below:

```
@app.post("/logout")
# Logout endpoint example with validation dependency
async def logout(response: Response, request: Request, user_id: uuid.
    UUID = Depends(validate_session), session: Session = Depends(
    get_session))

# Validation function to check if the user is authenticated
async def validate_session(request: Request, session: Session =
    Depends(get_session)):
  session_id = request.cookies.get("session_id")
```

```
  if not session_id:
    raise HTTPException(
      status_code=status.HTTP_401_UNAUTHORIZED,
      detail="Session cookie not found"
    )

  existingAuthSession = session.exec(
    select(AuthSession)
    .where(AuthSession.id == uuid.UUID(session_id))
    .where(AuthSession.expires_at > datetime.now())
  ).first()

  if not existingAuthSession:
    raise HTTPException(
      status_code=status.HTTP_401_UNAUTHORIZED,
      detail="Session not found"
    )

  return existingAuthSession.user_id
```

Listing 5.4: FastAPI Dependency for Session Validation

The system also used a hashing algorithm to hash the passwords before storing them in the database. Three hashing algorithms were identified as the best choices: bcrypt, scrypt and Argon2.

Bcrypt is a popular hashing algorithm widely used in the industry (Naiakshina et al., 2020; Ntantogian, Malliaros, and Xenakis, 2019). It offers an optimal balance between security and speed, making it a popular choice for real-world implementation (A. Sharma et al., 2024). Bcrypt was ranked on the mid-to-high end of the security scale, with algorithms such as scrypt and Argon2id scoring higher in the dimension of security (A. Sharma et al., 2024; Naiakshina et al., 2020). However, while scrypt and Argon2id are more secure, they are also more computationally expensive, and may require more configuration to be used effectively (A. Sharma et al., 2024).

In the end, the decision was made to use bcrypt due to its simplicity of use, wide adoption and its balance between security and perfomance. The passlib library was used to hash the passwords before storing them in the database. An example of this can be seen below:

```
    def verify_hash(plaintext_password: str, hashed_password: str):
        return bcrypt.verify(plaintext_password, hashed_password)

```

```
    def create_hash(plaintext_password: str):
        return bcrypt.hash(plaintext_password)
```

Listing 5.5: Hashing passwords with bcrypt

5.1.3 Frontend Setup

The frontend development done in this sprint focused on creating the login and register pages. PrimeVue components were used to quickly and easily create the form elements, maintaining visual consistency and a good user experience. Figure 5.1 shows an example of the DatePicker component implemented in the Register page.

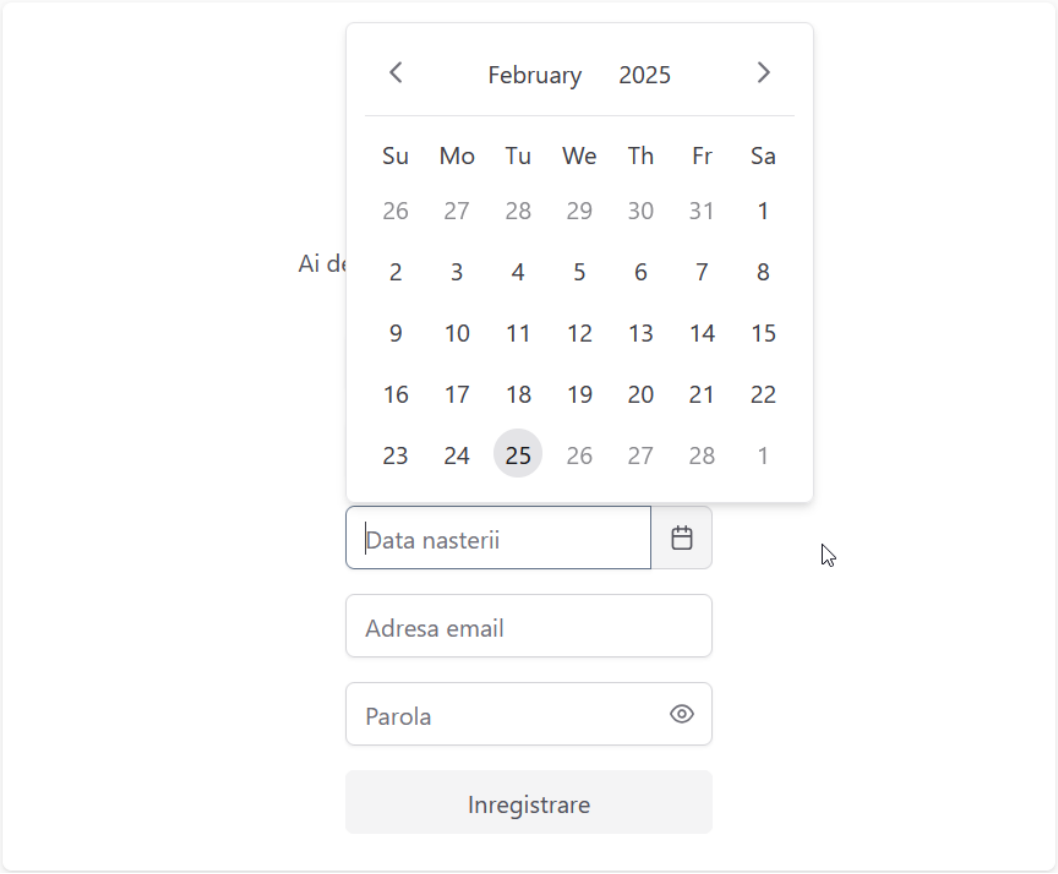


Figure 5.1: PrimeVue DatePicker component

To ensure control of route access, a simple navigation guard was created using Vue Router. Its role is to protect the routes flagged as requiring authenticatication, performing validation before navigation. An example of this can be seen below:

```
router.beforeEach(async (to) => {
  const authStore = useAuthStore()
  if (to.meta.requiresAuth) {
    await authStore.checkAuth() // Check if user is authenticated
    if (!authStore.isAuthenticated) {
      return { name: 'Login' } // Redirect to login if not
    authenticated
    }
  } // More code to handle other cases
})
export default router
```

Listing 5.6: Navigation Guard

Pinia store was used to manage the global authentication state across the application. It enabled access of authentication states across multiple application components, and was used by the Navigation Guard to validate a user's authentication status. The authentication store can be seen below:

```
export const useAuthStore = defineStore('auth', () => {
  const isAuthenticated = ref(false)
  const user = ref(null)

  async function checkAuth() { // check if user is authenticated
    try {
      const response = await api.get('/me') // return user ID if user
    is authenticated
      isAuthenticated.value = true // user is authenticated
      user.value = response.data.user
    }
    // More code to handle errors and no authentication
  }
  return { isAuthenticated, user, checkAuth }})
```

Listing 5.7: Pinia Store for Authentication

5.1.4 Challenges

Lack of experience

A primary challenge encountered in this sprint was the lack of experience with some of the frameworks and libraries. As such, the development progress moved slower than expected

due to the need to consult documentation or researching guides online. In the end, this learning process enabled to establish a solid foundation that would benefit the next sprints' development.

Implementation decisions

Another challenge was evaluating implementation options for different elements of the application, such as authentication. Additional time was spent researching the best practices for each feature analysing their advantages, disadvantages and ways to implement them in the current system.

5.1.5 Requirements completed

This initial sprint enabled the completion of some of the foundational requirements for this application:

- The system must provide a secure login mechanism for patients by using a combination of login and password.
- The database must store the user credentials in a secure manner.
- The system must be accessible on all modern desktop and mobile-based browsers.
- The system must allow patients to add their own personal information, such as name or date of birth.

5.2 Sprint #2

The second sprint focused on implementing system functionality, specifically managing vaccines, allergies and medications. This approach enabled further familiarisation with the frameworks and libraries used in the project, while also establishing design and implementation patterns used in further sprints.

5.2.1 Vaccines, allergies and medication management

Database models were first created for vaccines, allergies and medcations by using SQLModel, followed by corrensponding API endpoints to add, update, delete and view the data. The frontend development used Vue components to create reusable elements, that would be used to display the user added records. These components used other Vue features such as props and emits to pass data and events between elements. An example of the Vaccine Card can be seen below:

```
<Card class="w-full" :pt="cardStyles">
  <template #title>
    <span class="font-bold text-2xl">{{ name }}</span>
  </template>
  <template #subtitle>
    <div class="flex items-center justify-between">
      <div>
        <span class="font-bold">{{ provider }}</span>
        <span>{{ date_received }}</span>
      </div>
      <div>
      <Button icon="pi pi-eye" class="p-button-rounded p-button-text"
        @click="emit('showFile', props.id)" v-if="hasCertificate"/>
      <Button icon="pi pi-ellipsis-h" class="p-button-rounded p-button
    -text"
        @click="toggle"/>
      <Menu ref="menu" :model="items" :popup="true" />
      </div>
    </div>
  </template>
</Card>
```

Listing 5.8: Vue Vaccine Card Component Example

The VaccineCard component was then easily integrated in its respective page:

```
<VaccineCard
  v-for="vaccine in vaccines"
  :key="vaccine.id"
  v-bind="vaccine"
  :has-certificate="!!vaccine.certificate"
  @delete="deleteVaccine"
  @open-edit="openEditDialog"
  @show-file="showCertificate"
/>
```

Listing 5.9: Using VaccineCard Component

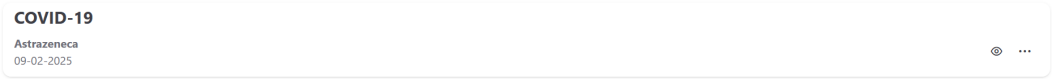


Figure 5.2: Vue Vaccine Card component

Vue's reactivity system was used to ensure the data was automatically updated whenever any change was made. Examples of the reactivity system in use can be seen below:

```
// Will add a vaccine to the vaccines array, which will automatically
// create a new VaccineCard element
const addVaccine = (vaccine) => {
  vaccines.value.push(vaccine)
}

// Will remove a vaccine from the vaccines array, which will
// automatically remove the VaccineCard element
const deleteVaccine = (id) => {
  vaccines.value = vaccines.value.filter((vaccine) => vaccine.id !==
    id)
}
```

Listing 5.10: Vue Reactivity System

5.2.2 File Uploads feature

A file upload feature was also implemented in this sprint. It was initially intended for vaccination certificates, but designed to be easily extensible for other future records. This required for a new table to be created in the database, FileUpload, which would contain file metadata. Similarly, a link table for allergies and severities was created to enforce consistency in the names used and allow for easy filtering and sorting on the frontend. The updated ERD diagram, with the new tables, can be seen in figure 5.3:

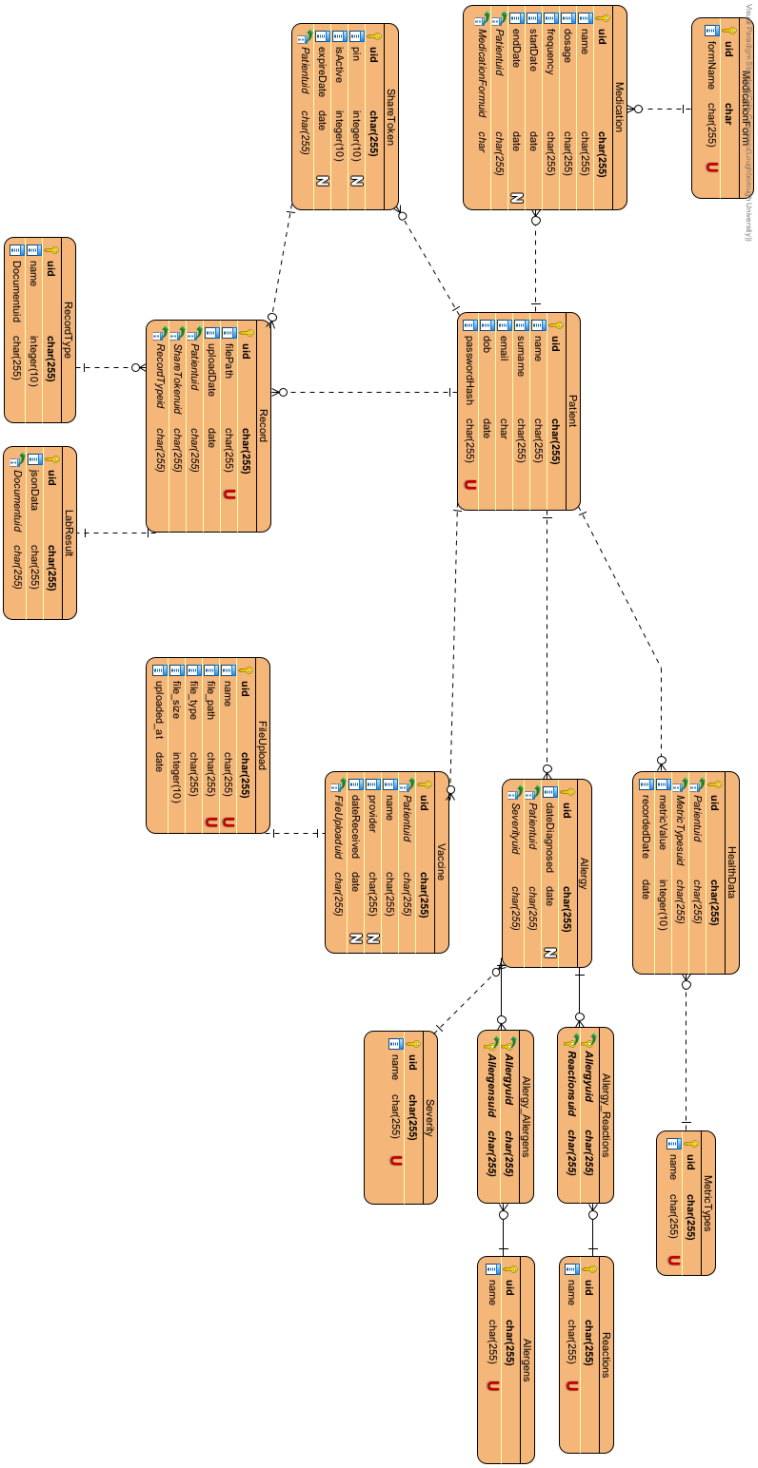


Figure 5.3: Updated Entity Relationship Diagram Sprint # 2

File security was implemented by using AES as the encryption algorithm. It was selected based on a comparative research of existing encryption methods, with AES consistently offering the best balance between security and performance among other symmetric and asymmetric encryption algorithms (Nurgaliyev and H. Wang, 2021; Marqas, Almufti, and Ihsan, 2020; Alenezi, Alabdulrazzaq, and Mohammad, 2020). A symmetric algorithm was used for its implementation simplicity and to prevent users losing access to their files by losing their private key. The key used to encrypt the files was stored in the environment variables. In the interest of time, no rotation system for the key was implemented, however this would be a good feature to add in the future.

The file upload process can be seen in figure 5.5 below. It encompasses the following steps:

1. File upload to server via multipart form data
2. Server-side validation of file type and size
3. File encryption and storage in the file system and database
4. For retrieval, the file would be streamed via a new endpoint

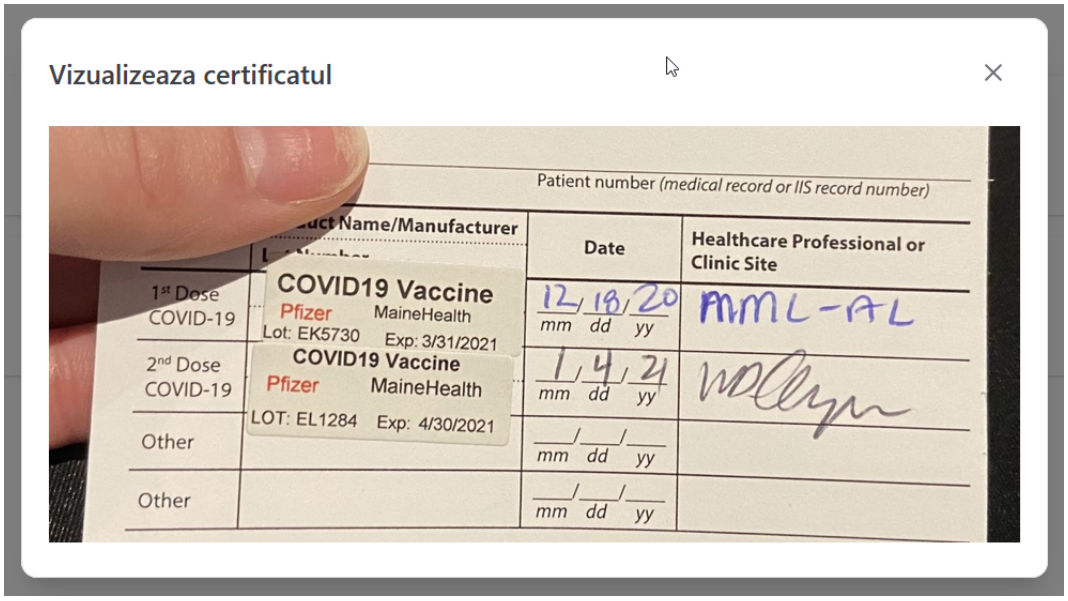
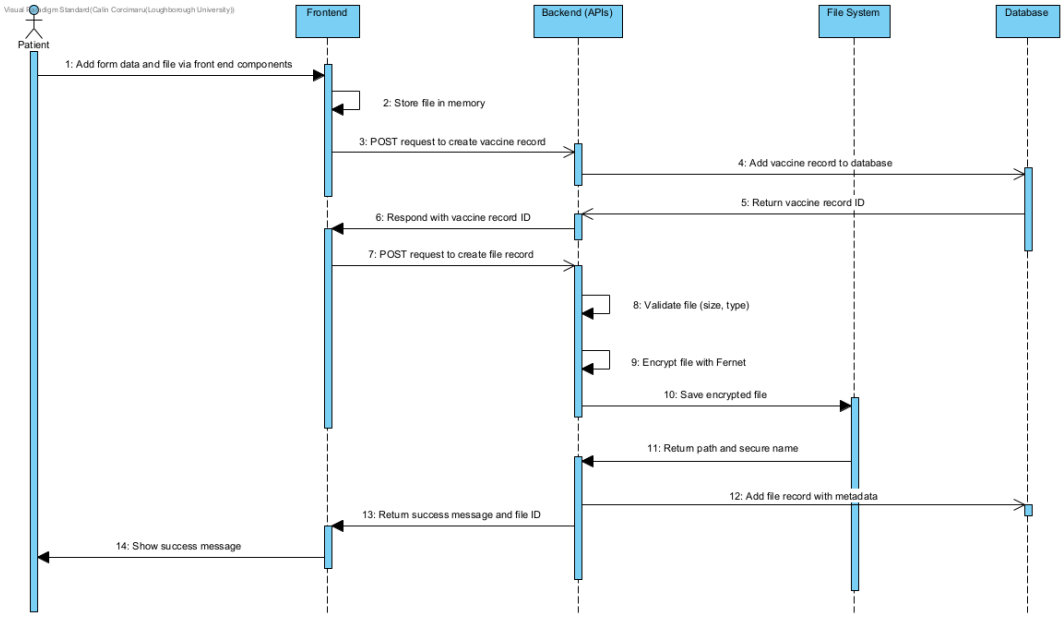
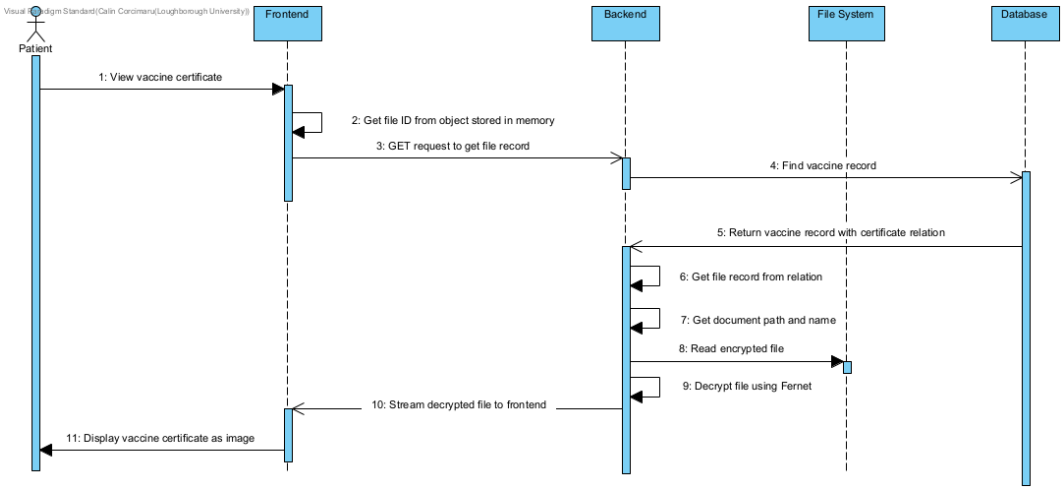


Figure 5.4: Uploaded Vaccine Certificate



(a) File Upload Sequence Diagram



(b) File View Sequence Diagram

Figure 5.5: Sequence Diagrams for File Upload and File View

```python
# Get a file by ID
@app.get("/files/{record_type}/{record_id}")
async def get_file(record_type: str, record_id: uuid.UUID, user_id:
    User = Depends(validate_session), session: Session = Depends(
    get_session)):
  # Database query omitted for brevity
  async def get_data_from_file():
    with open(file_record.file_path, "rb") as f:
      encrypted_content = f.read()
    decrypted_content = decrypt_file(encrypted_content)

    yield decrypted_content

  return StreamingResponse(
    content=get_data_from_file(),
    media_type=file_record.file_type,
    status_code=status.HTTP_200_OK,
    headers={"Content-Disposition": f"inline; filename={file_record.
    name}"}
  )
```

Listing 5.11: File Download Endpoint

The frontend was created with both mobile and desktop browsers in mind. This was achieved through using a combination of PrimeVue components TailwindCSS styling. Examples of the allergies page on both desktop and mobile can be seen in figure 5.6.

5.2.3 Challenges

This sprint encountered its share of challenges. However, tackling them showed that undertaking a more agile approach to the project was beneficial, as the project allowed to embrace change and to be flexible during the development process.

New functionality integration

Introducing new functionality in the system, while ensuring compatibility with both the frontend and backend, was the main challenge this sprint. Additional complexity was added by changing existing code or database models to accommodate the new features, which required careful planning and testing to ensure that the new features did not break any existing functionality.

File Uploads

Lack of prior experience working with server and client side file handling features was another challenge encountered this sprint. It required additional research, development and testing time to ensure proper implementation and compatibility with the existing system.

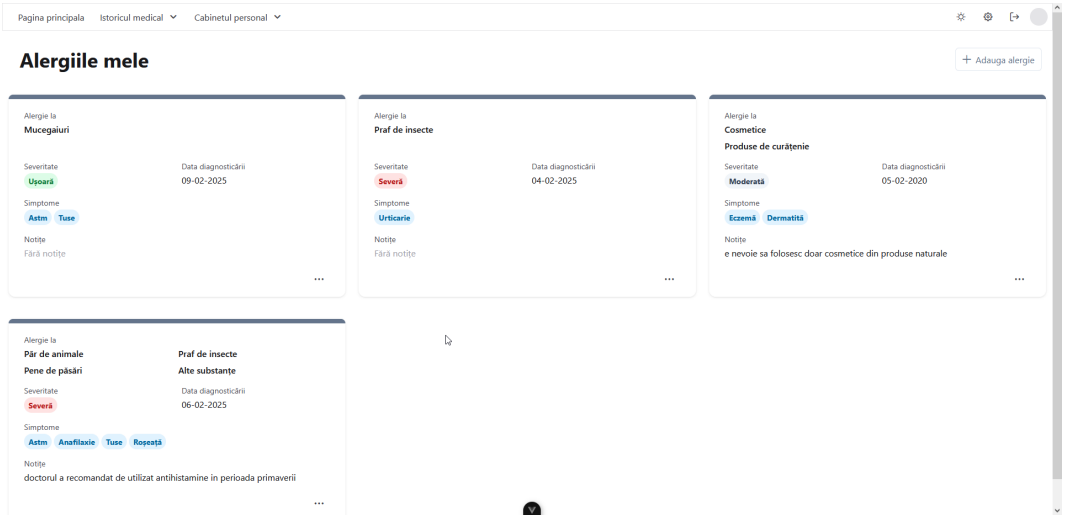
Poor planning & time management

The additional complexity brought by the integration of this sprint's feature required more time than initially estimated. As such, poor planning and time management resulted in moving some of the sprint's planned features to the next one, ensuring a rushed development would be avoided.

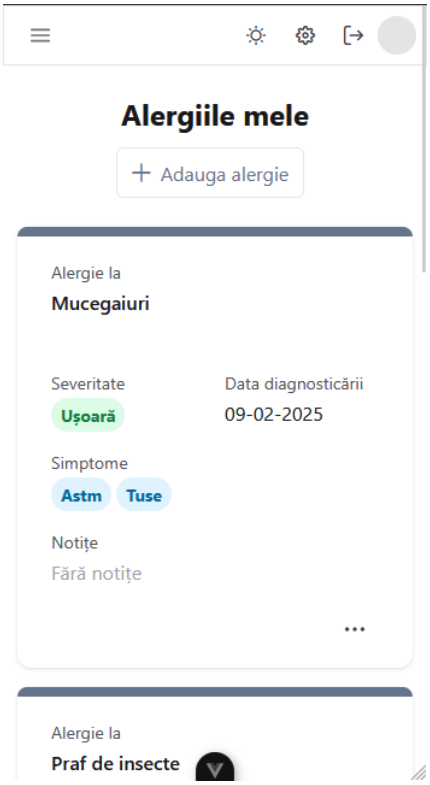
5.2.4 Requirements completed

The main requirements completed in this sprint were:

- The system must store the data in a secure manner, ensuring only the patient and doctor can access the data.
- The system must allow patients to upload records in multiple file formats.
- The system should allow users to assign uploaded files for all record types.
- The system must allow the patient to add their own allergies.
- The system must allow the patient to add their own vaccinations.
- The system must allow patients to enter their current medication details.
- The system must allow patients to add new medication to their list.



(a) Desktop version



(b) Mobile version

Figure 5.6: Desktop and Mobile version of the Allergies page

5.3 Sprint #3

The third sprint focused on implementing the main dashboard and the vital signs functionality of the system, which would allow patients to add information such as weight or blood pressure in both a tabular and graphical format. Stakeholder feedback from previous sprints also had to be addressed, which included reworking some frontend elements to ensure consistency across application pages.

The features were chosen to be completed this sprint as their implementation would help build proficiency with the frameworks and libraries before tackling more complex functionality in subsequent sprints.

5.3.1 User interface rework

During a presentation of previous sprint results, the stakeholders identified inconsistencies in how information was presented across pages in the application, caused by the different information each record contained. To ensure a consistent view of medical records, PrimeVue's Data View component was implemented. It provided both a grid and list visualisation option, while allowing for customisation of how individual records were displayed. The information displayed was also simplified, to ensure important details were conveyed without distractions.

An example of the updated allergies page can be seen in figure 5.7. The page now offers both a grid and list view, which can be toggled by the user, creating a more consistent view across different medical records.

5.3.2 Filtering and sorting

Stakeholder feedback also identified the need to filter and sort the information displayed in the application. This was implemented by using Vue's reactivity system, ref and computed functions. It allowed to create complex relationships, which would update variables based on given input or data change. An example of the sorting and filtering functionality can be seen below:

```
1  const filteredMedicine = computed(() => {
2    return props.medications.filter((medication) => {
3      return medication.name.toLowerCase().includes(searchQuery.value.
     toLowerCase())
4    })
5  })
```

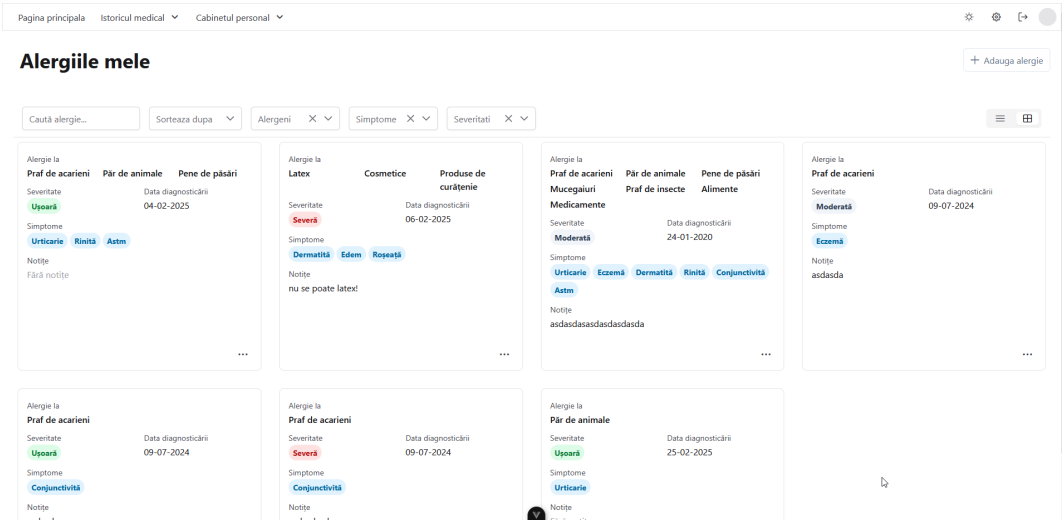
Listing 5.12: Medication Sorting and Filtering

```
  const filteredAllergies = computed(() => {

  if (selectedAllergens.value.length > 0) {
    return props.allergies.filter((allergy) =>
      allergy.allergens.some((allergen) => selectedAllergens.value.
    includes(allergen))
    )
  }

  if (selectedReactions.value.length > 0) {
    return props.allergies.filter((allergy) =>
      allergy.reactions.some((reaction) => selectedReactions.value.
    includes(reaction))
    )
  }

  if (selectedSeverities.value.length > 0) {
    return props.allergies.filter((allergy) =>
      selectedSeverities.value.includes(allergy.severity)
    )
  }

  return props.allergies.filter((allergy) =>
    allergy.allergens.some((allergen) =>
      allergen.toLowerCase().includes(searchQuery.value.toLowerCase())
    )
  )
})
```

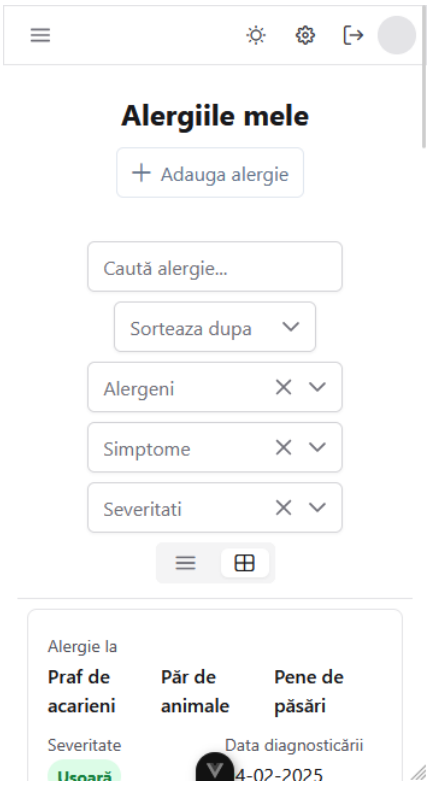
Listing 5.13: Allergies Sorting and Filtering

5.3.3 Vital signs monitoring

The vitals signs feature was implemented to add and track basic health metrics such as weight, blood pressure or pulse over time. This addition allowed for the data to be displayed in both a tabular and graphical format to enable track progress over time. To display the data in a tabular and graphical view, the system uses PrimeVue's Data Table component and Chart.js library respectively. To further enhance the feature, current value indicators and trend markers were added to the component to provide a better user experience. Examples of how the Vitals page looks can be seen in figure 5.8 and 5.9.

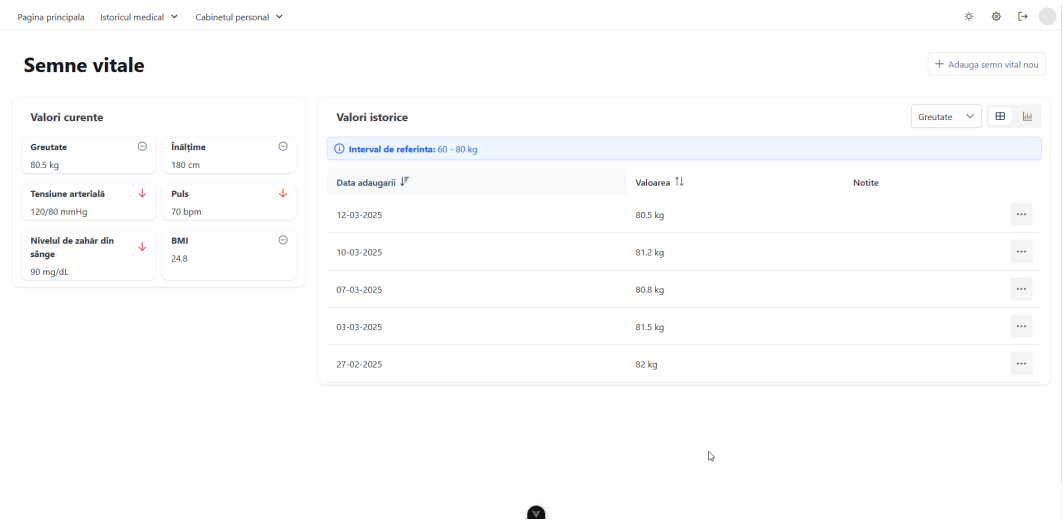


(a) Desktop version

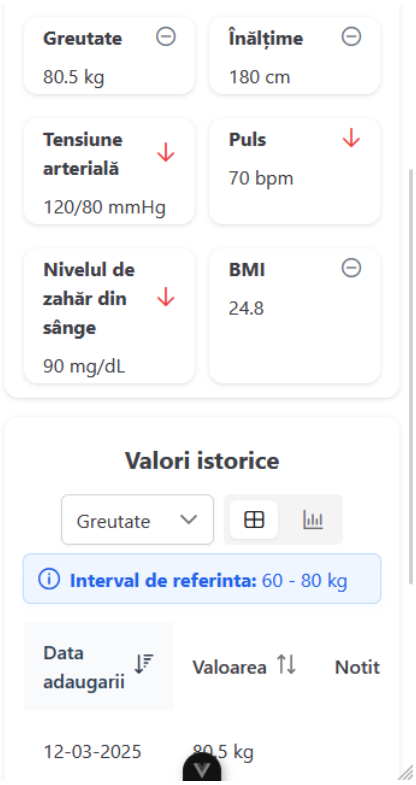


(b) Mobile version

Figure 5.7: Desktop and Mobile version of the Allergies page 2nd version

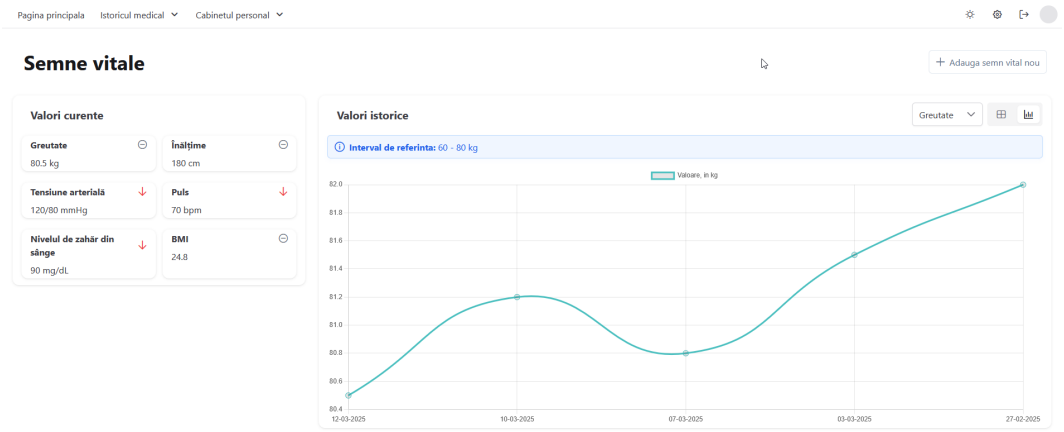


(a) Desktop version

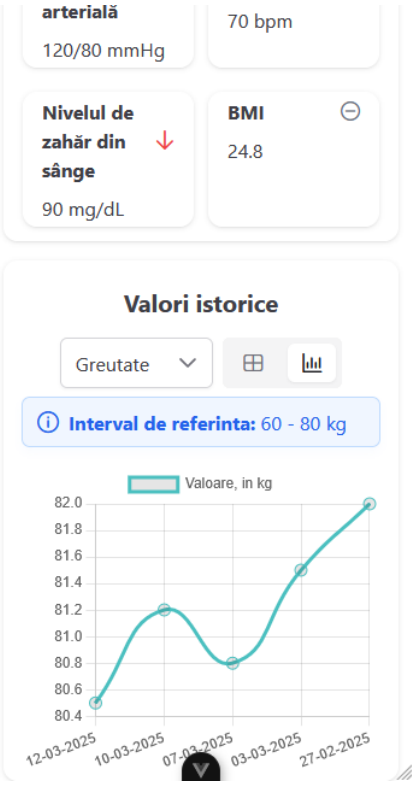


(b) Mobile version

Figure 5.8: Desktop and Mobile version of the Vitals page



(a) Desktop version



(b) Mobile version

Figure 5.9: Desktop and Mobile version of the Vitals page, with Graph

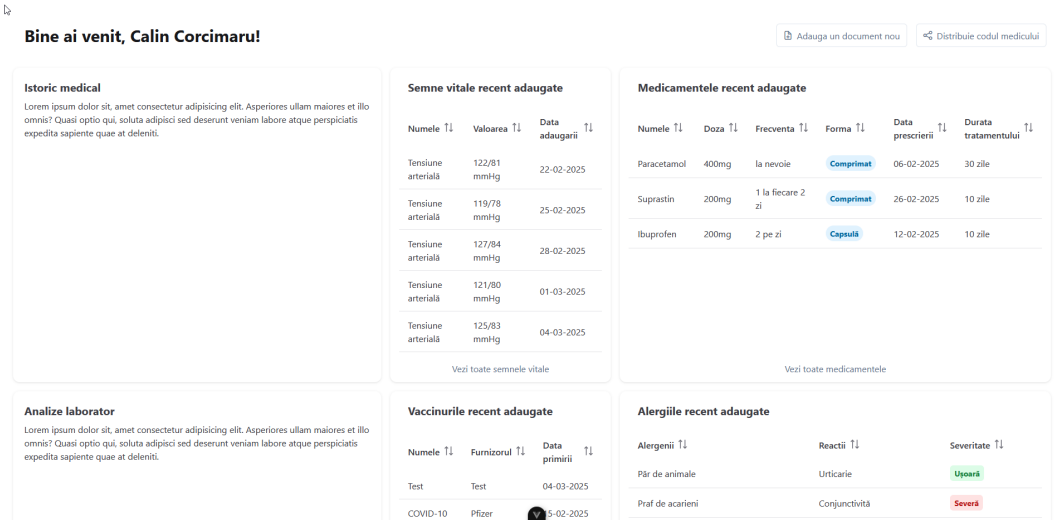
5.3.4 Dashboard implementation

The dashboard feature was implemented to provide and overview of recent records added to the system. It was designed to present up to 5 recent records for each section, encouraging users to visit the respective page to view more records and details. The display format for each record was kept consistent with their individual pages.

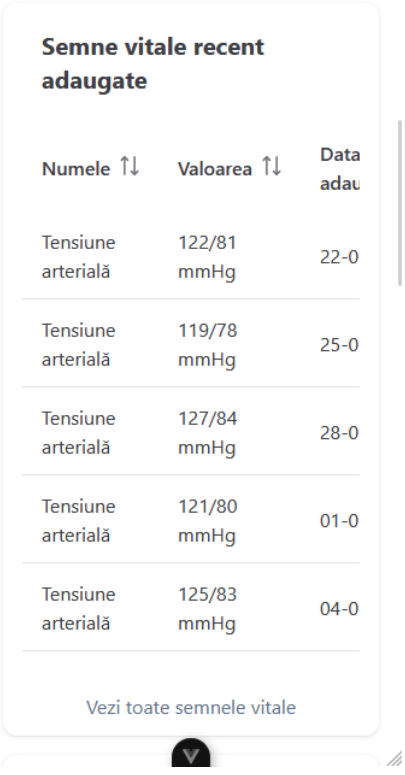
Data Table from PrimeVue was used to create the individual tables for the dashboard sections. Each section was an individual and modular Vue component, which was then imported into the dashboard main file. An example of a section code can be seen below:

```
<template>
  <Card :pt="cardStyles" class="h-full">
    <template #title><h2 class="text-xl font-bold p-4">Semne vitale recent adaugate</h2></template>
    <template #content>
      <DataTable :value="props.vitals" removableSort>
        <Column field="name" header="Numele" sortable />
        <Column field="value" header="Valoarea" sortable>
          <template #body="{data}">
            <span v-if="data.value">{{ data.value }} {{ data.unit }}</span>
            <span v-else>{{ data.value_systolic }}/{{ data.value_diastolic }} {{ data.unit }}</span>
          </template>
        </Column>
        <Column field="date_recorded" header="Data adaugarii" sortable>
          <template #body="{data}">{{ data.original_date_recorded }}</template>
        </Column>
      </DataTable>
    </template>
    <template #footer><RouterLink to="/vitale" class="p-button p-button-text">Vezi toate semnele vitale</RouterLink></template>
  </Card>
</template>
```

Listing 5.14: Vital Signs Dashboard Section Component



(a) Desktop version



(b) Mobile version

Figure 5.10: Desktop and Mobile version of the Dashboard page

5.3.5 Backend changes

Several database changes were implemented to support the new frontend functionalities. The updated backend diagram can be seen in figure 5.11. The main changes included:

- Addition of a new database table for vital signs, with a separate data type to standardise units and references ranges across values.
- Addition of a new date_added fields to all database tables to differentiate between the clinical date it represents and the date it was added to the system, enabling the ability to sort and query the recently added records.

To accommodate the newly added table, changes were made to the vital sign API endpoints. Additional endpoints were also created for the dashboard sections to display their most recent results by using the new date_added field. The dashboard endpoint can be seen below:

```
@app.get("/dashboard", response_model=UserDashboard)
async def get_dashboard(user_id: uuid.UUID = Depends(validate_session)
    , session: Session = Depends(get_session)):
    user = session.get(User, user_id)

    if user_id != user.id:
        raise HTTPException(status_code=403, detail="You do not have
    permission to access this endpoint!")

    newest_vaccines = session.exec(select(Vaccine).where(Vaccine.
    user_id == user_id).order_by(col(Vaccine.date_added).desc()).limit
    (5)).all()

    # Similar code for allergies, medications and vitals...
    # Response transformation code omitted for brevity

    user_dashboard = UserDashboard(
        id = user.id,
        name = user.name,
        vaccines = vaccines_response,
        allergies = allergies_response,
        vitals = healthdata_response,
        medications = medications_response
    )
    return user_dashboard
```

Listing 5.15: Dashboard Endpoint

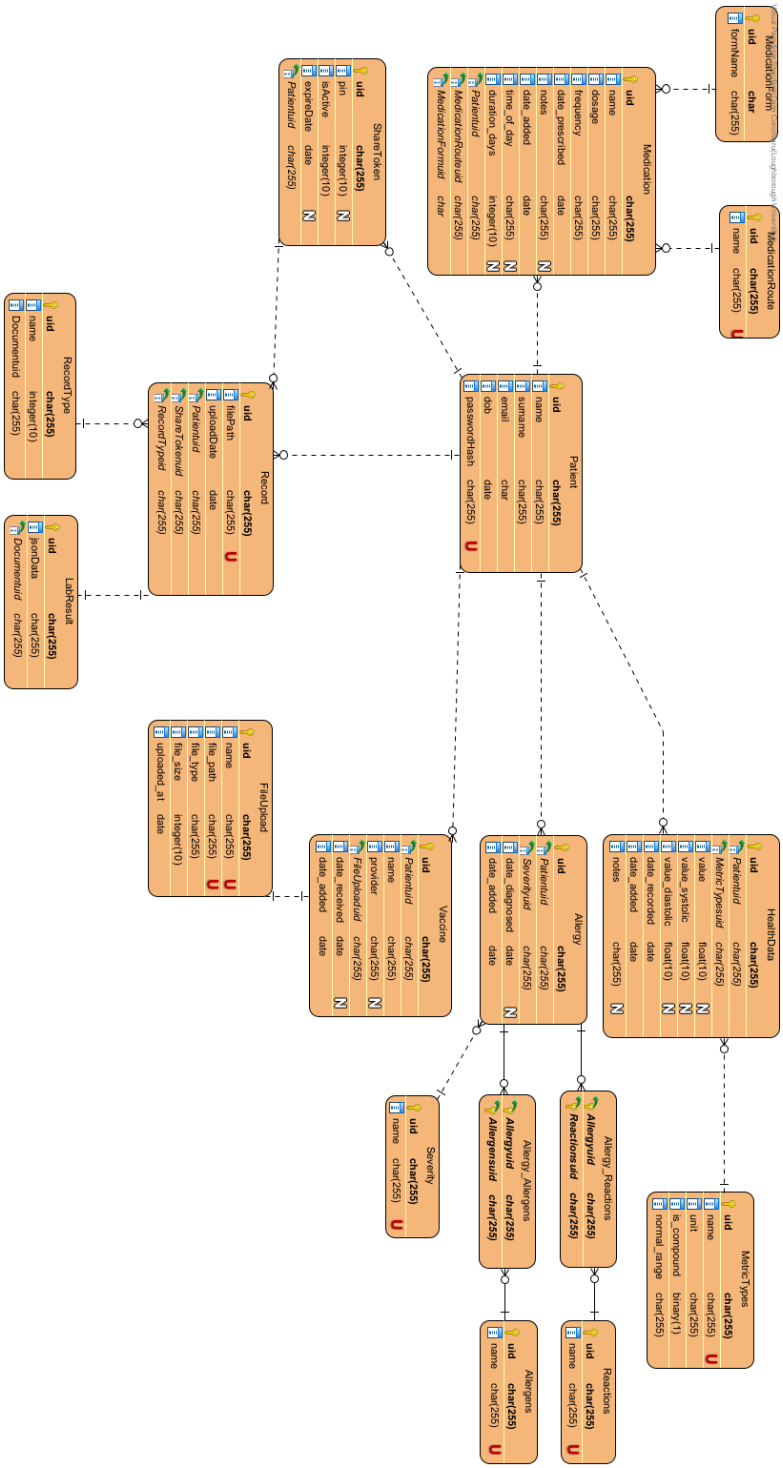


Figure 5.11: Updated Entity Relationship Diagram for Sprint #3

5.3.6 Challenges

Scope Creep

Stakeholder feedback introduced at the start of the sprint led to additional work being done beyond the initial workload planned. Both backend and frontend changes were required to ensure adherence to the new requirements, sapping valuable time and resources from the main sprint objectives. This issue emphasises both the importance of frequent and clear communication with stakeholders, but also the need to establish a clear scope and priority for each sprint.

Infrequent communication with stakeholders

Infrequent stakeholder interaction was another issue encountered in this and previous sprints. The unavailability of some stakeholders due to busy schedules as doctors led to some decisions made that later required revision, adding additional workload. As such, more consistent communication with stakeholders was identified as a key improvement necessary for the success of subsequent sprints.

Testing limitations

The lack of experience with automated testing frameworks for the framework resulted in limited testing capability and coverage. Even though testing was still done, albeit manually, not all issues would be immediately caught. This results in intermittent bugs appearing during the development process, which sidetracked the development efforts from the sprint objectives. Additionally, due to time constraints, testing was usually left out at the end or not done at all, leading to even more issues arising unexpectedly in the future.

5.3.7 Requirements completed

- The system must have a dashboard view which displays an overview of the most recent information added to the system (latest lab tests, doctor consultations, vaccinations etc).
- The system should display the health records in both a list or grid view.
- The system should allow the patient to enter vitals information, such as height, weight, blood pressure, etc.
- When multiple vital entries are made, the system could display a historical graph of the patient's vitals.

5.4 Sprint #4

The fourth sprint of the project focused on implementing the health records functionality, enabling users to add records such as lab tests, doctor consultations and imaging results. It addresses one of the core requirements of this application, which is to view and manage your medical history in one place. Additional feedback was received from stakeholders at the end of sprint #3, which included:

- Integration of reference ranges for vitals sign graphs
- Implementation of date filtering for vital signs data
- Adjustment of recent vital signs comparison logic to compare with normal ranges instead
- Refinement of dashboard display for allergies and vital signs sections: allergies with moderate and above severities and the most recent value for each vital sign type

5.4.1 Health Records

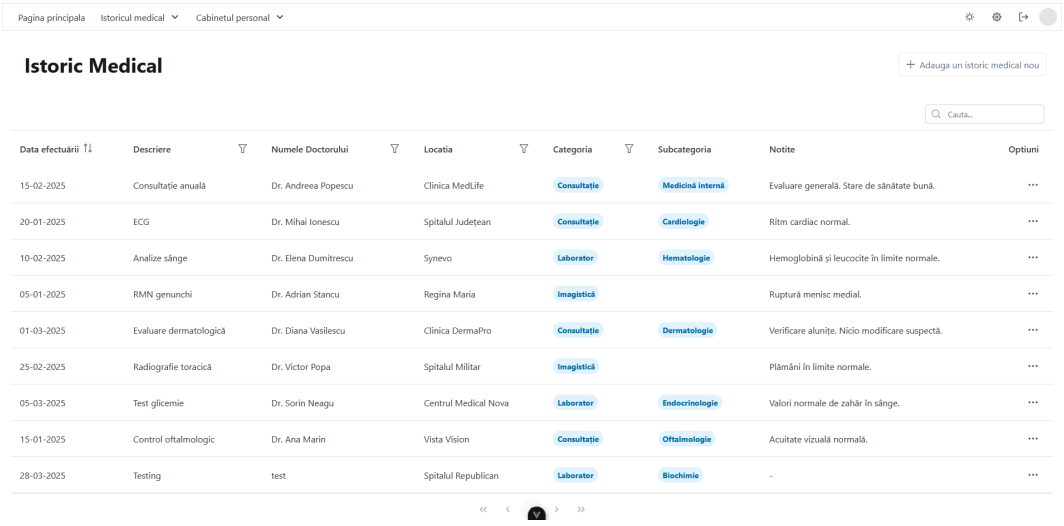
The medical history feature was implemented using PrimeVue's DataTable component to display the data in a tabular format. This allowed for a more structured view of the data, but also enabled the ability to sort and filter by column data thanks to DataTable's built-in functionalities. As such, it was possible to ensure an efficient organisation of data, with consistency across record types. An example of the health records page can be seen in figure 5.12.

Advanced filtering options were enabled through DataTable's filtering API, allowing for both global and column-specific filtering options. An example of a filter menu can be seen in figure 5.14.

Finally, to provide an even better user experience, the document management functionality was enhanced with the ability to display PDF files. This allowed patients to attach files to specific medical records for future reference. On desktop, the files were displayed using the browser's embedded PDF viewer, while on mobile the application would provide a download option to avoid compatibility issues. The code for the PDF viewer can be seen below, with a mobile and desktop example shown in figure 5.13.

```
  <Dialog
    v-model:visible="visible"
    modal
    header="Vizualizeaza fisierul"
    class="w-full md:w-3/4 md:h-3/4"
    @hide="emit('close')">
    <div v-if="metadata?.file_type == 'application/pdf'">
      <object
        :data="`http://localhost:8000/files/medicalhistory/${props.
    historyId}`"
        type="application/pdf"
        class="w-full h-[70vh] hidden md:block"/>
      <div class="block md:hidden text-center mt-2">
        <a
          :href="`http://localhost:8000/files/medicalhistory/${props.
    historyId}`"
          target="_blank"
          class="bg-blue-500 text-white px-4 py-2 rounded-md inline-
    block mb-4">
          Deschide certificatul PDF
        </a>
      </div>
    </div>
    <Image
      v-else
      :src="`http://localhost:8000/files/medicalhistory/${props.
    historyId}`"
      alt="Fisier consultatie"
      class="w-full h-screen"
      preview/>
  </Dialog>
</template>
```

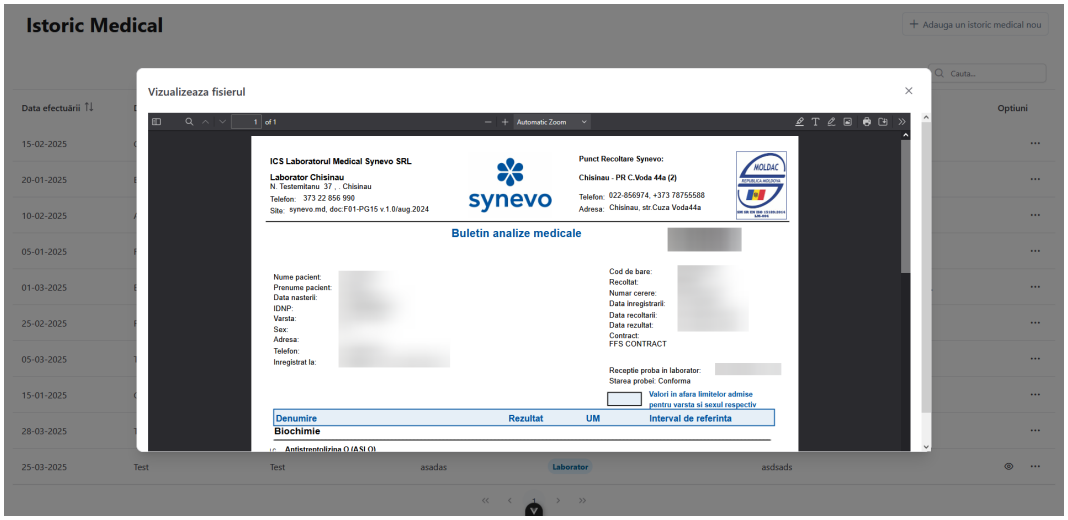
Listing 5.16: PDF viewer Function for Health



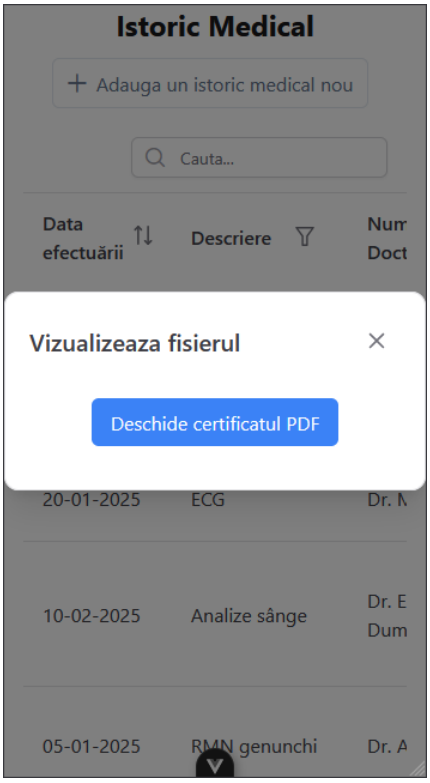
(a) Desktop version

(b) Mobile version

Figure 5.12: Desktop and Mobile version of the History page



(a) Desktop version



(b) Mobile version

Figure 5.13: Desktop and Mobile version of the PDF viewer

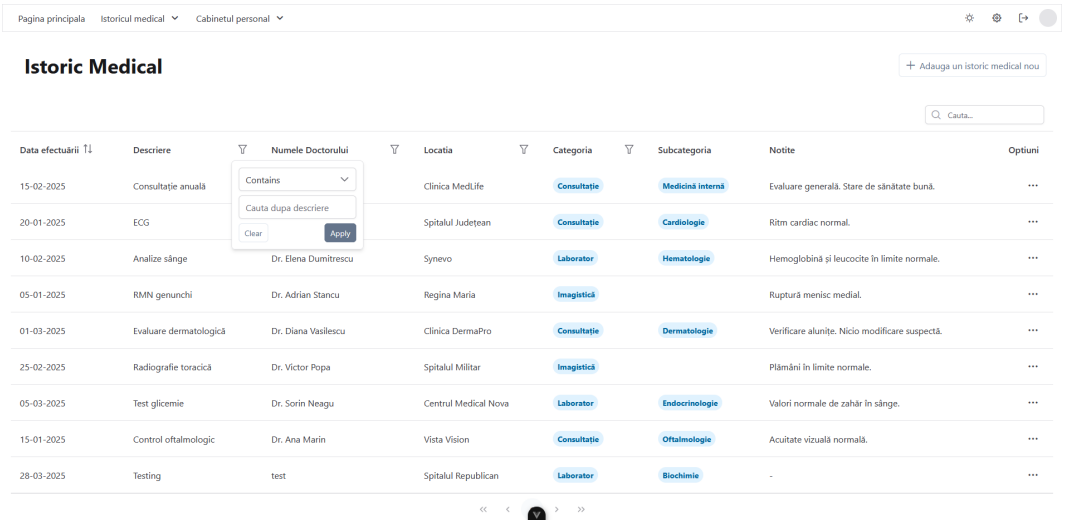


Figure 5.14: Filter Menu for Health Records

5.4.2 Backend changes

To accommodate the medical history feature, the database was extended with new tables to support record organisation, which include a core table for record metadata, a table for primary classification (into consultation, laboratory or imagining) and secondary tables for consultation specialties and test types. The new ERD diagram can be seen in figure 5.15.

Additionally, API endpoints were created to allow the user to add, edit and delete health records. To promote reusability within the system, the file management endpoints created in sprint #2 were reused for the medical history feature.

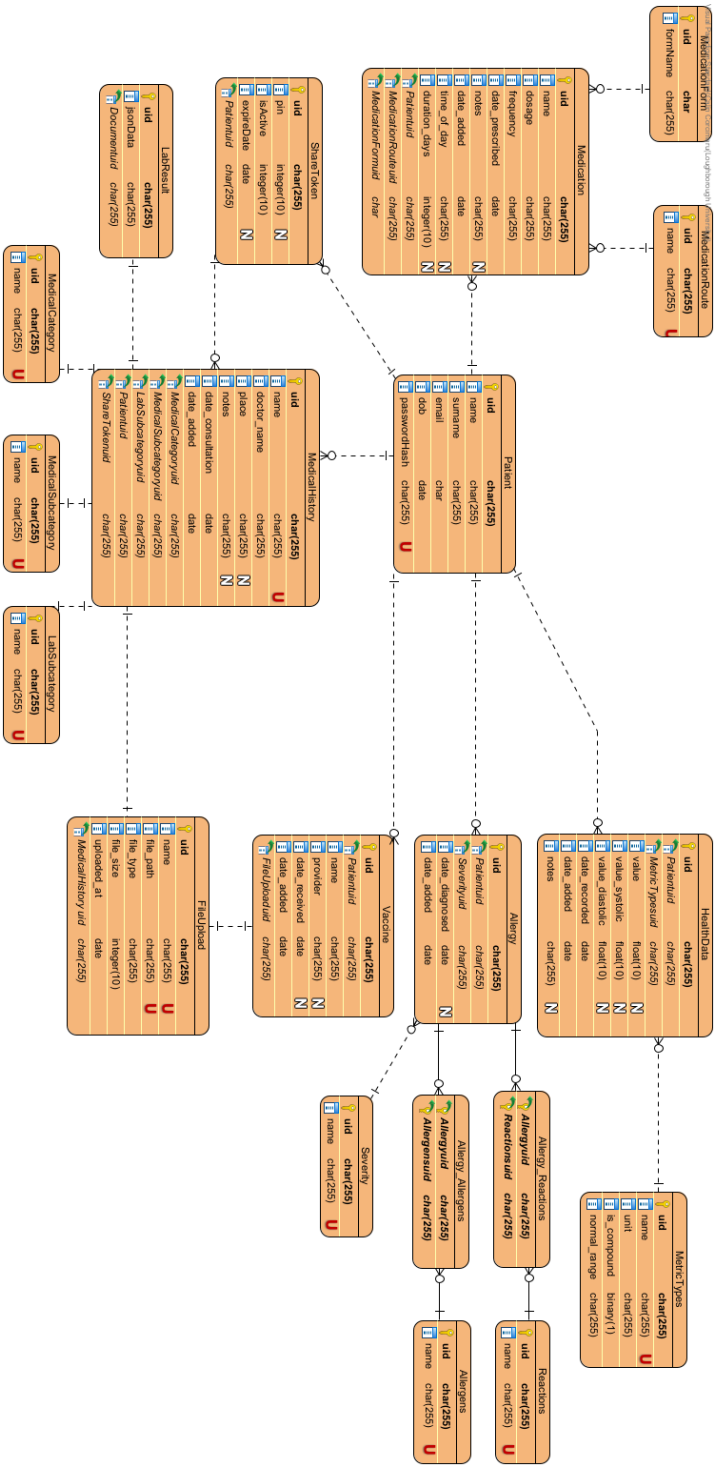


Figure 5.15: Updated Entity Relationship Diagram Sprint #4

5.4.3 Work on stakeholder feedback

Several changes to the user interface were implemented based on stakeholder feedback. The biggest change was the addition of 'normal range' boundaries to the graph views for some vital sign types, providing immediate context and feedback for vital sign results. Additionally, a date filter was introduced on the page, allowing users to filter the data based on selected date ranges. Finally, the computation logic for recent vital sign trends was changed to utilise their respective normal ranges to dictate their trend, instead of the previous value. Overall, these changes were made to improve the user experience and provide more context for the data displayed. Their implementation can be seen below in figure 5.16.

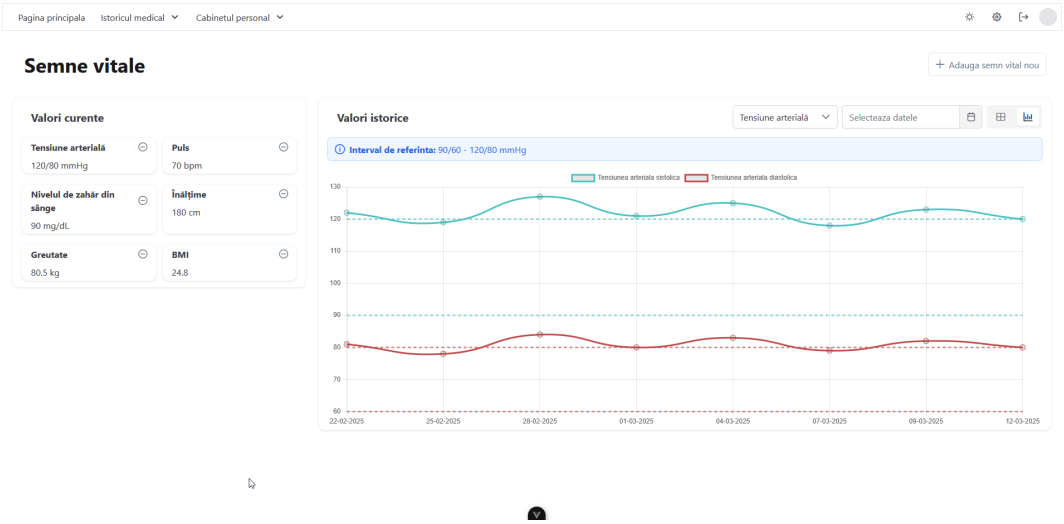


Figure 5.16: Updated Vitals View based on stakeholder feedback

The main dashboard view was also optimised to display more accurate and relevant information in some of the existing sections. For vitals signs, only the most recent value for each type was now shown in the dashboard, alongside their trend indicator. For the allergies section, it was changed to only display records with a severity of 'moderate' and above to prioritise showing more severe (and more clinically important) cases of a patient's allergies. The new dashboard can be seen in figure 5.17.

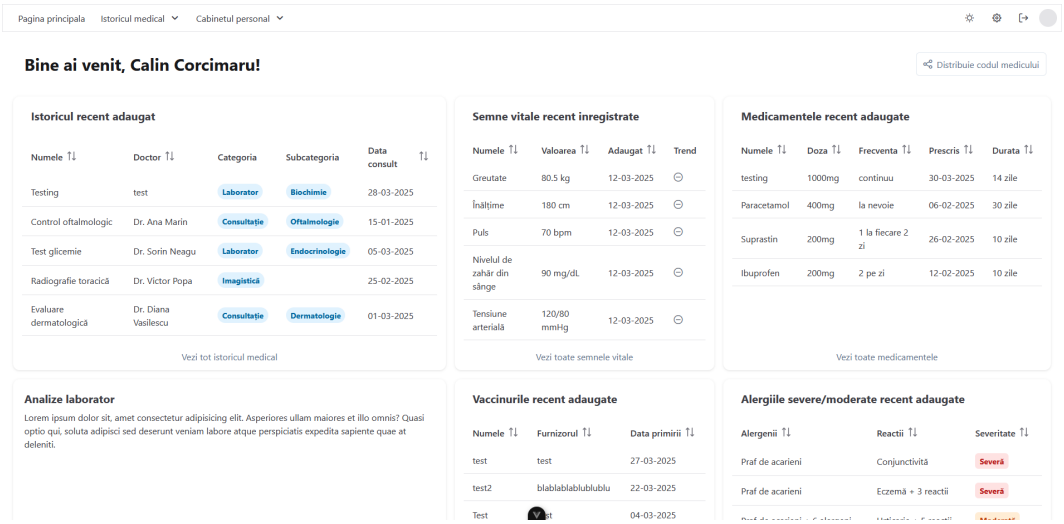


Figure 5.17: Updated Dashboard View based on stakeholder feedback

5.4.4 Challenges

The primary challenge encountered during this sprint was, interestingly enough, unrelated to any technical implementation issues. Instead, an unexpected period of illness resulted in approximately one week lost of development time, leading to a reduced feature output for this sprint. As such, some of the unfinished worked had to be transferred to the next sprint's backlog.

5.4.5 Requirements completed

- The system must allow the patient to specify and categorise the type of document they are uploading (lab test, doctor consultation, etc).
- The system must allow patients to upload their own medical records in a variety of formats (PDF, DOC, etc).
- The system must display the patient's history in a chronological order in the form of a timeline.
- When viewing doctor consultations, the system should divide them into categories based on the domain of the doctor (cardiology, neurology, etc).
- The system must allow the patient to add a description of the document they are uploading.
- The system should allow the patient to add a date for the document they are uploading.

- The system should allow the patient to add a location for the document they are uploading.
- The system should allow the patient to add the doctor name for the document they are uploading.
- The system should allow the patient to sort and filter the documents based on the type of document, date, location, and doctor name.

5.5 Sprint #5

The fifth sprint focused on implementing an automated lab test data extraction system that utilised a MLLM to process uploaded lab result documents. This feature represented an incredible innovation within this application, as it enabled automatic extraction of test results from documents, with minimal manual intervention. Being a continuation of the work done in the previous sprint, some of the initial work required for the implementation of this feature was already completed. This allowed for easier and faster development of the lab extraction functionality, which was completed as per the initially agreed requirements and wireframes, with minor changes to how the lab results were displayed on the frontend.

The sequence diagram for the lab tests extraction process can be seen in figure 5.18. It displays how a user would interact with the system when uploading a lab result document, how the system processes the filem sends it to the MLLM API, and how the response is then received and displayed to the user.

5.5.1 Lab Tests Extraction

The implementation of initial lab extraction steps leveraged previously added components in sprint #4, to streamline the user experience and maintain a single entry point for document uploads with optional AI processing. As such, the 'Add Health Record' component and the 'Laboratory' health record type were reused to expand the document upload flow. Upon selecting the 'Laboratory' category, the user would get the option to toggle document AI processing, which will send the file to the MLLM API. This design choice closely matched the initial wireframe design for the 'New Document' dialog, which can be seen in figure B.6. As such, the updated 'Add Health Record' dialog can be seen in figure 5.19.

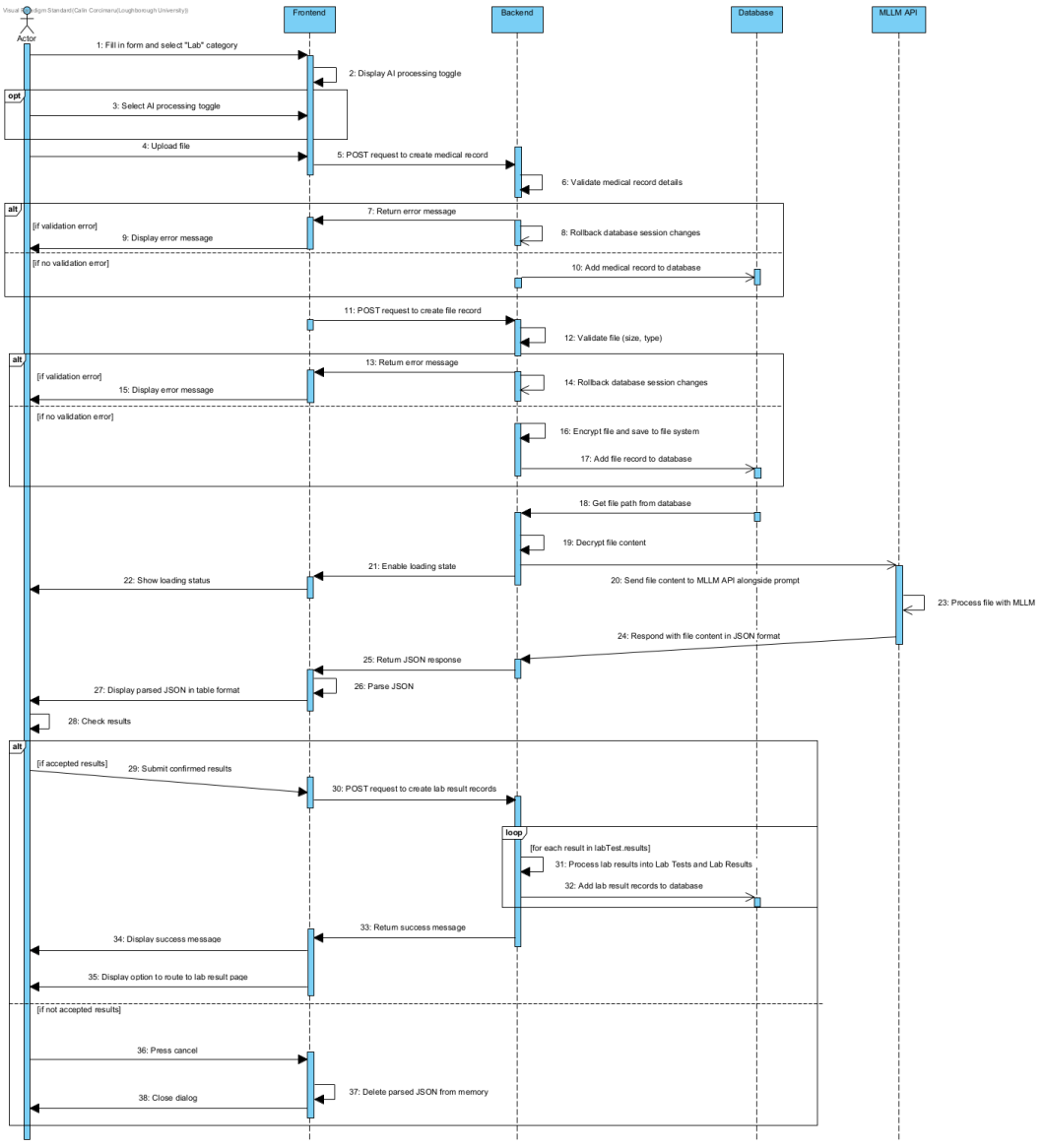
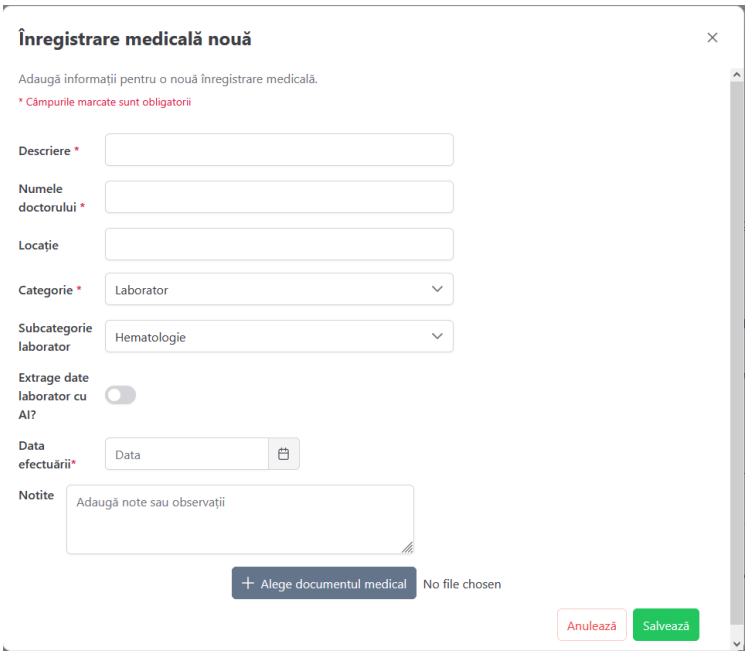


Figure 5.18: UML Sequence Diagram — Extracting Lab Results



(a) Desktop version

(b) Mobile version

Figure 5.19: Desktop and Mobile version of the new Add Document dialog

Google's Gemini 2.0 Flash was selected as the application's MLLM, based on research of available free tiers with API access from table 3.4. However, it was noted that a local MLLM could've also been used for the project, a point which will be discussed in more detail in section 7.2. The MLLM implementation mainly involved using Prompt Engineering techniques to provide explicit instructions and format the output consistently. Prompting techniques from section 3.4.3 were used to write the following prompt, which is passed alongside the file to the API:

```
def extract_with_llm(file_content: bytes, file_type: str):
  prompt = """You have been given a document that contains lab results
     that is written in the Romanian language. Your job is to extract
    all lab results from this document in JSON format with the
    following fields: test_name, test_code, value, unit,
    reference_range, method. Sometimes the code of the test will be in
     the name itself, and it is your job to determine if the code is
    there, for example in brackets or separated by a comma, and
    separate the name and the code. Sometimes the lab result will not
    have a method specified, and in that case you return an empty
    string. The document is in Romanian, however the JSON keys should
    be in English.

  EXTREMELY IMPORTANT FORMATTING INSTRUCTIONS:
  1. Return ONLY the raw JSON array
  2. DO NOT use code blocks, backticks, or markdown formatting
  3. DO NOT include ```json or ``` anywhere in your response
  4. DO NOT include any explanations or text before or after the JSON
  5. Your response must start with the '[' character and end with the
    ']' character
  6. The output should be valid JSON that can be parsed directly
  7. Use period (.) as the decimal separator, not comma (,)

  Example of how your output should look, starting from the very first
     character:
  [{"test_name":"Hemoglobina","test_code":"HGB","value":"14.3","unit":
    "mg/dL","reference_range":"13.2-17.3", "method":"Chemiluminiscenta
    "}]
  if there is no method specified, return an empty string:
  [{"test_name":"Hemoglobina","test_code":"HGB","value":"14.3","unit":
    "mg/dL","reference_range":"13.2-17.3", "method":" "}]"""
```

Listing 5.17: Prompt for Lab Results Extraction

Upon receiving the results in a JSON format, the system would parse the response and display it in a table format to the user. Next, a validation step was added to ensure result accuracy before submitting the data to the backend, by utilising PrimeVue's DataTable row editing feature to enable the user to correct the results if necessary. This step was deemed necessary to ensure any possible MLLM hallucination could be corrected by the user. An example of how the extracted lab results are displayed can be seen in figure 5.20.

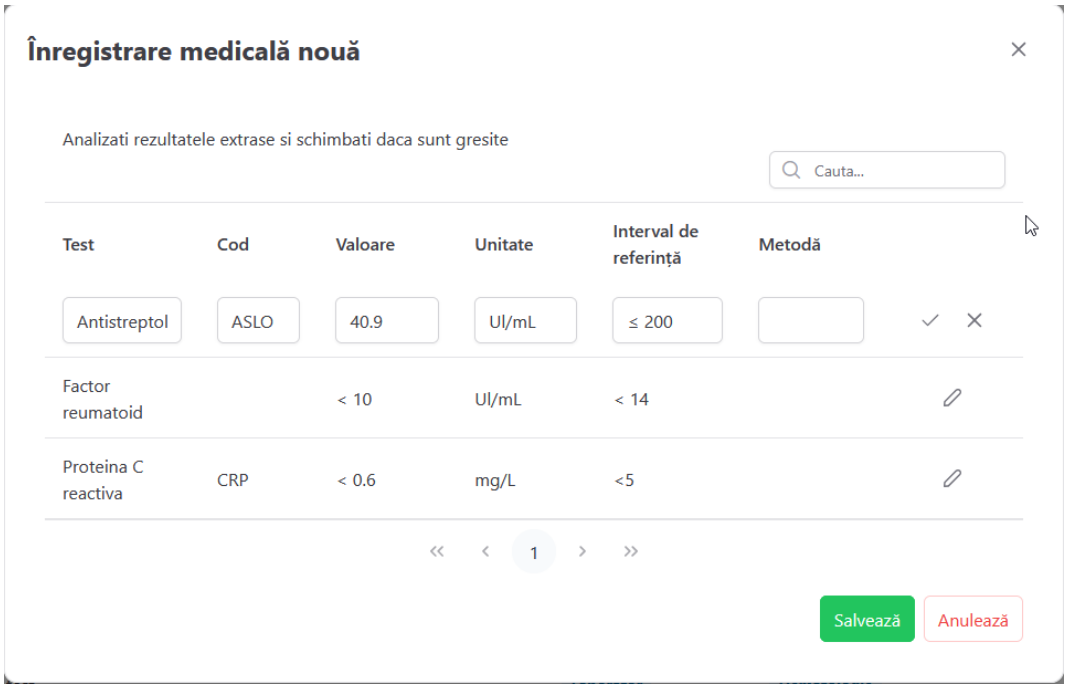


Figure 5.20: Desktop dialog validation of extracted lab results

The extracted lab results also leverage DataTable's dynamic column generation feature, ensuring the ability to accommodate any set of columns passed by the result data. This is important, as it may not be possible to predict the number of columns beforehand, but it also greatly simplifies the code needed to create the table. A code snippet of the DataTable can be seen below:

```
<DataTable
  :value="extractionResult"
  v-model:editingRows="editingRows"
  class="w-full"
  :rows="10"
  editMode="row"
  paginator
```

```
    dataKey="test_name"
    @row-edit-save="onRowEditSave"
    v-model:filters="filters"
    :globalFilterFields="['test_name', 'test_code']">
    <template #header>
      <h2 class="m-0">Analizati rezultatele extrase si schimbati daca
    sunt gresite</h2>
      <div class="flex justify-end">
        <IconField>
          <InputIcon>
            <i class="pi pi-search" />
          </InputIcon>
          <InputText
            size="small"
            v-model="filters['global'].value"
            placeholder="Cauta..."/>
        </IconField>
      </div>
    </template>
    <Column
      v-for="col of columns"
      :key="col.field"
      :field="col.field"
      :header="col.header">
      <template #editor="{ data, field }">
        <InputText v-model="data[field]" class="w-full" fluid />
      </template>
    </Column>
    <Column
      :rowEditor="true"
      style="width: 10%; min-width: 8rem"
      bodyStyle="text-align:center"/>
  </DataTable>
```

Listing 5.18: Dynamic DataTable for Lab Results

Backend changes

To accommodate the new lab extraction feature, the database was extended with two new tables: LabTest, which contained the core test metadata (name and code) and LabResult, which contained individual test results with values, units and reference ranges. A dual

table approach was chosen to prevent data duplication and facilitate historical tracking by allowing to associate multiple results to one test type. The updated database can be seen in figure 5.21. It includes a direct patient-result relationship to optimise queries for cases when retrieving all lab results for a patient is necessary, as is the case for the dashboard.

A code snippet of the endpoint that handles the processing of lab results can be seen below:

```
  # Create the lab test records in the database
@router.post('/me/labtests/')
async def create_lab_tests(extraction_result: LabsCreate, user_id:
    uuid.UUID = Depends(validate_session), session: Session = Depends(
    get_session)):
    medhistory = session.get(MedicalHistory, extraction_result.
    medicalhistory_id)
    user = session.get(User, user_id)

    if not medhistory:
        raise HTTPException(status_code=status.HTTP_404_NOT_FOUND,
    detail="Medical history not found")

    if medhistory.user_id != user_id:
        raise HTTPException(status_code=status.HTTP_403_FORBIDDEN,
    detail="Not authorised to access this medical history")

    for lab_item in extraction_result.lab_tests:
        lab_test = session.exec(select(LabTest).where(LabTest.name ==
    lab_item.name)).first()

        if not lab_test:
            lab_test = LabTest(
                name = lab_item.name,
                code = lab_item.code,)
            session.add(lab_test)
            session.flush()

        lab_result = LabResult(
            value = lab_item.value,
            is_numeric = check_is_numeric(lab_item.value),
            unit = lab_item.unit,
            reference_range = lab_item.reference_range,
            date_collection = extraction_result.date_collection,
```

```
            test = lab_test,
            medicalhistory = medhistory,
            user = user,
            method = lab_item.method)

        session.add(lab_result)
        session.flush()

    session.commit()

    return {"status": status.HTTP_201_CREATED, "message": "Lab tests
    created successfully"}
```

Listing 5.19: Lab Results Endpoint

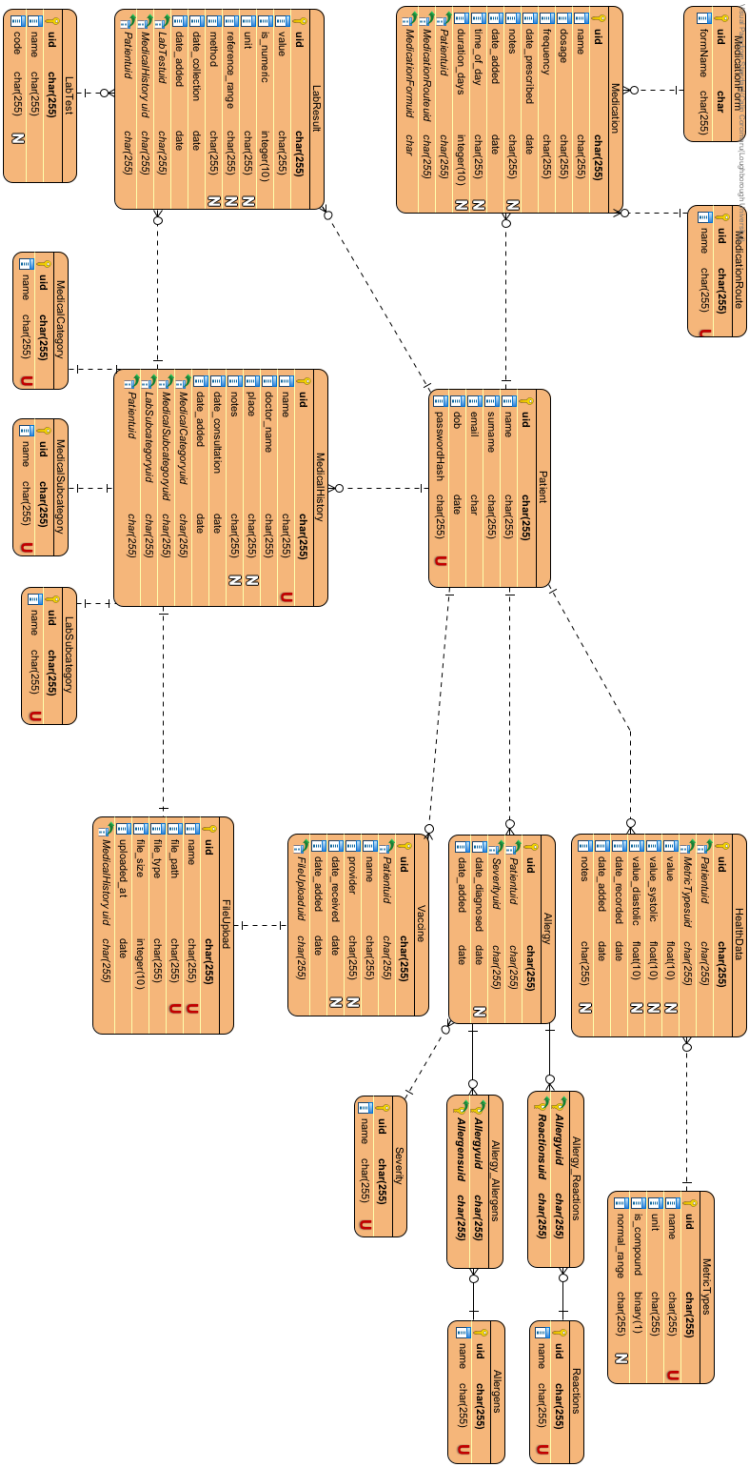


Figure 5.21: Updated Entity Relationship Diagram Sprint #5

5.5.2 Results visualisation

The initial idea to display the extracted lab results was to reuse the existing design from the vitals page, developed in sprint #3, which would've also followed the initially agreed wireframes and design. However upon discussing with stakeholders, a new design was proposed which used a hierarchical format, with expandable rows for each test type. Alongside each test type a small graph would be displayed to show the evolution of test results over time, with a visual indicator on the side which would inform patients if the recent result was outside the reference range. An example of the new lab results page can be seen in figures 5.22 and 5.23.

This new design offered several advantages:

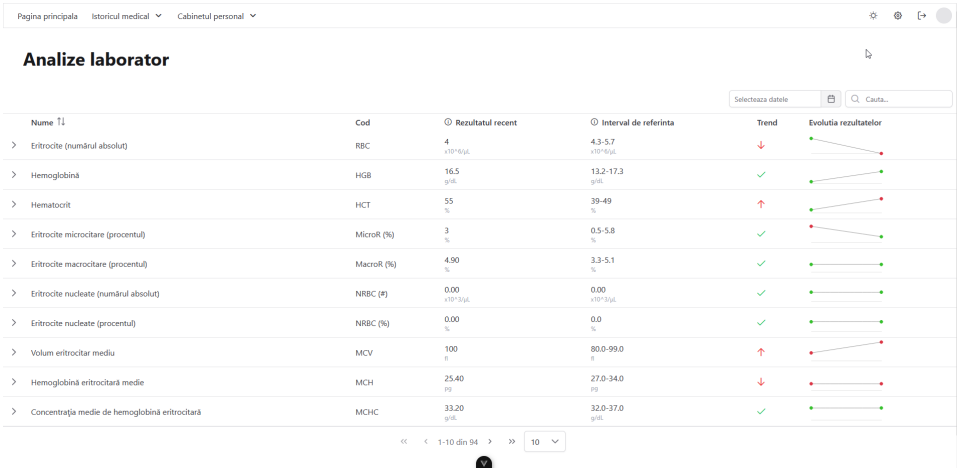
1. A compact presentation of large amounts of test results
2. Efficient filtering for existing test types
3. Integrated trend visualisation through sparklines and reference range indicator
4. A more responsive design for mobile users

5.5.3 Challenges encountered

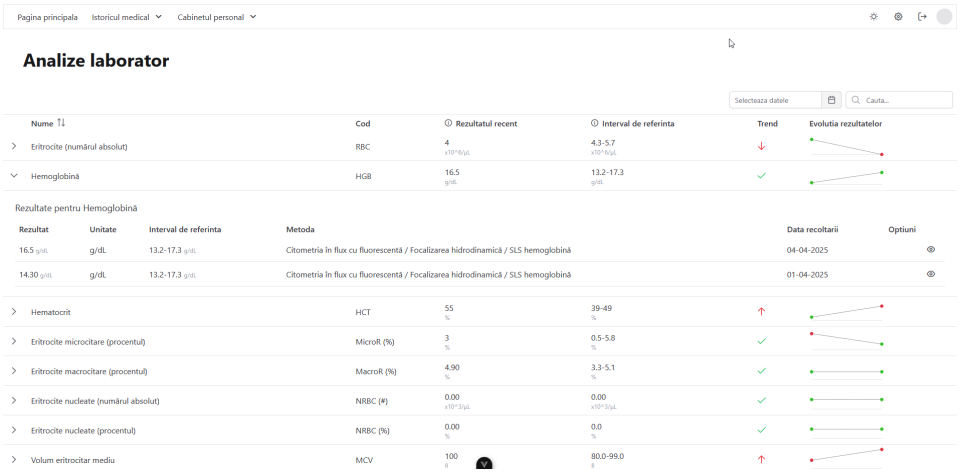
The primary challenge encountered this sprint was adapting to the mid-sprint lab result visualisation design change that was requested by the stakeholders. While it introduced additional design and research work, the agile approach taken and PrimeVue's extensive component library facilitated an effective implementation. In the end, the design actually streamlined the development as it leveraged the DataTable's capabilities for complex data presentation. Another challenge was the project's timeline constraints, which resulted in core functionality being prioritised over implementing more experimental features.

5.5.4 Requirements completed

- The system must provide an overview of the patient history through 3 main sections: personal information, lab tests, and doctor consultations.
- The system must allow the user to view blood tests in a graphical format.
- The system must allow the user to view blood tests in a numerical, tabular format.
- For blood test results, the system should display the normal range values for each test.
- The system could allow the user to view the document from which the lab results were extracted.

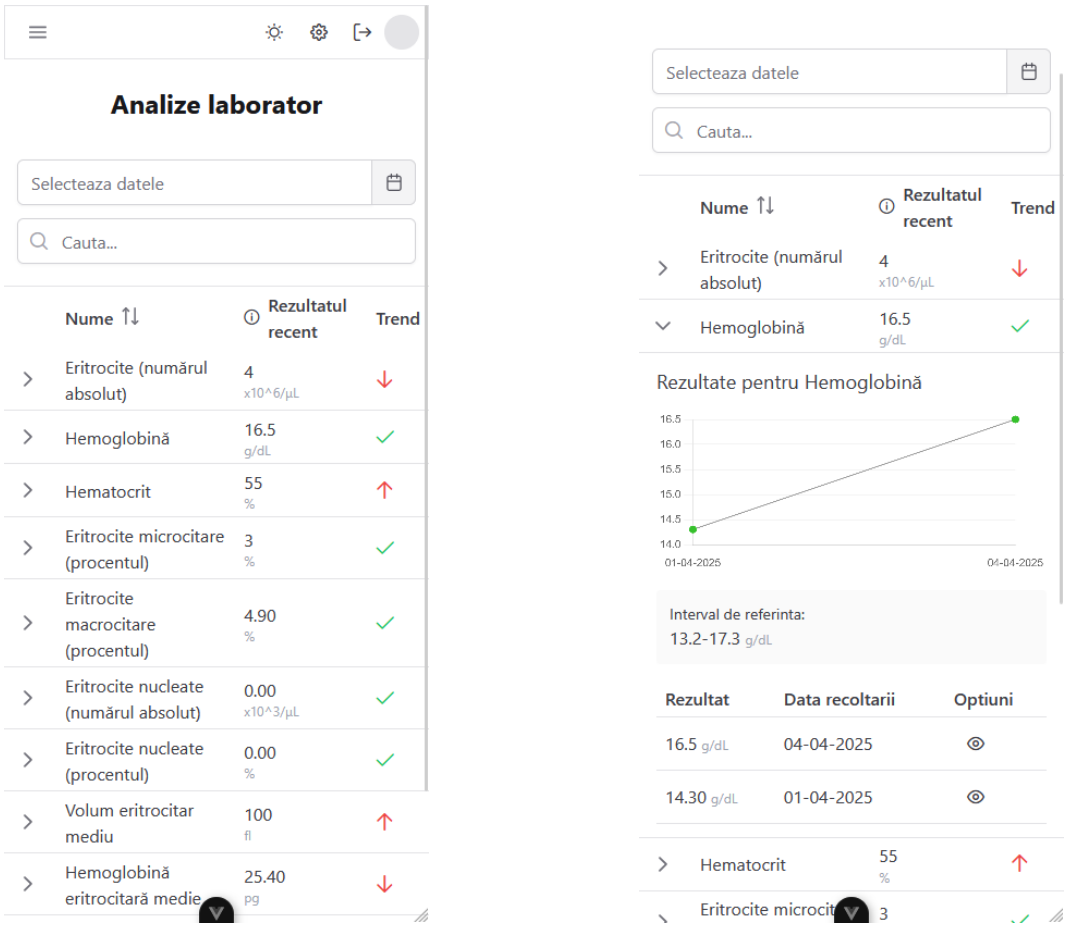


(a) Desktop version



(b) Desktop version #2

Figure 5.22: Desktop version of the new Lab View page



(a) Mobile version

(b) Mobile version #2

Figure 5.23: Mobile version of the new Lab View page

5.6 Sprint #6

The final sprint focused on implementing the record sharing functionality, a critical feature that would enable patients to securely share their medical records with healthcare practitioners. This feature addressed one of the core objective of this project, which was to eliminate information siloing across Moldova's multiple healthcare institutions. Instead, this feature would empower the user, creating a patient-controlled approach that complements existing institutional systems. It also represented a consolidation of previously implemented features, leveraging existing components and endpoints to display records to healthcare practitioners.

5.6.1 Share links

During discussion with stakeholders, links were quickly identified as the preferred method of sharing medical information. Links were chosen due to their accessibility, as most doctors in Moldova have access to a computer at their desk. Similarly, links can be very versatile in how they are communicated: via email, dictated or even generated as a QR code.

To ensure a safe sharing process, the mechanism was designed with three security layers in mind. The first layer would be a random 8-character alphanumeric code, implemented to facilitate easy sharing while maintaining reasonable security. An example of a share link would be *https://example.com/share/7zSaQbBd*, with the code for the link generation and token model available below:

```
  # Helper function to generate random strings for share codes
def generate_random_string(length: int) -> str:
    characters = string.ascii_letters + string.digits
    return ''.join(secrets.choice(characters) for _ in range(length))

# Share Token model for database
class ShareToken(SQLModel, table=True):
    id: uuid.UUID =Field(default_factory=uuid.uuid4, primary_key=True)
    share_code: str = Field(index=True, unique=True, default_factory=
    lambda: generate_random_string(8))
    expiration_time: datetime
    created_at: datetime = Field(default_factory=datetime.now)
    hashed_pin: str
    shared_items: dict = Field(default_factory=dict, sa_column=Column(
    JSON))
    user_id: uuid.UUID = Field(foreign_key="user.id")
    user: "User" = Relationship(back_populates="share_tokens")
```

Listing 5.20: Share Link Generation

The second layer was a configurable expiration period, which by default was 1 hour but could be set to a maximum of 24 hours. It was implemented to limit any potential exposure of the shared information, aligning with the single-use nature of the links during consultations. Finally, the third layer was a user-defined 6-character PIN, set during the initial steps of the share process. The PIN would be hashed and stored in the database for increased security. When accessed, the system first validates the link's expiration time, then prompts for the correct PIN before displaying any medical data. The verification endpoint is shown below:

```
@router.post("/share/{share_code}/verify", status_code=status.
  HTTP_200_OK, response_model=ShareItemsResponse)
@limiter.limit("5/minute")
async def verify_share_token(
  request: Request, share_code: str, pin: Annotated[str, Body(embed=
  True)],
  session: Session = Depends(get_session)):
  share_token = session.exec(select(ShareToken).where(ShareToken.
  share_code == share_code)).first()

  if not share_token:
    raise HTTPException(status_code=status.HTTP_404_NOT_FOUND,
  detail="Share token not found")

  if not verify_hash(pin, share_token.hashed_pin):
    raise HTTPException(status_code=status.HTTP_403_FORBIDDEN,
  detail="Invalid PIN")

  user = session.exec(select(User).where(User.id == share_token.
  user_id)).first()
  user_data = {"name": user.name, "dob": user.dob.strftime("%d-%m-%Y
  ") if user.dob else None}
  items_data = get_item_data(share_token.shared_items, session)

  return ShareItemsResponse(
    expiration_time=share_token.expiration_time,
    patient=user_data,
    items=items_data)
```

Listing 5.21: Share Link Endpoint

The whole record sharing flow is illustrated in sequence diagrams 5.24 and 5.25.

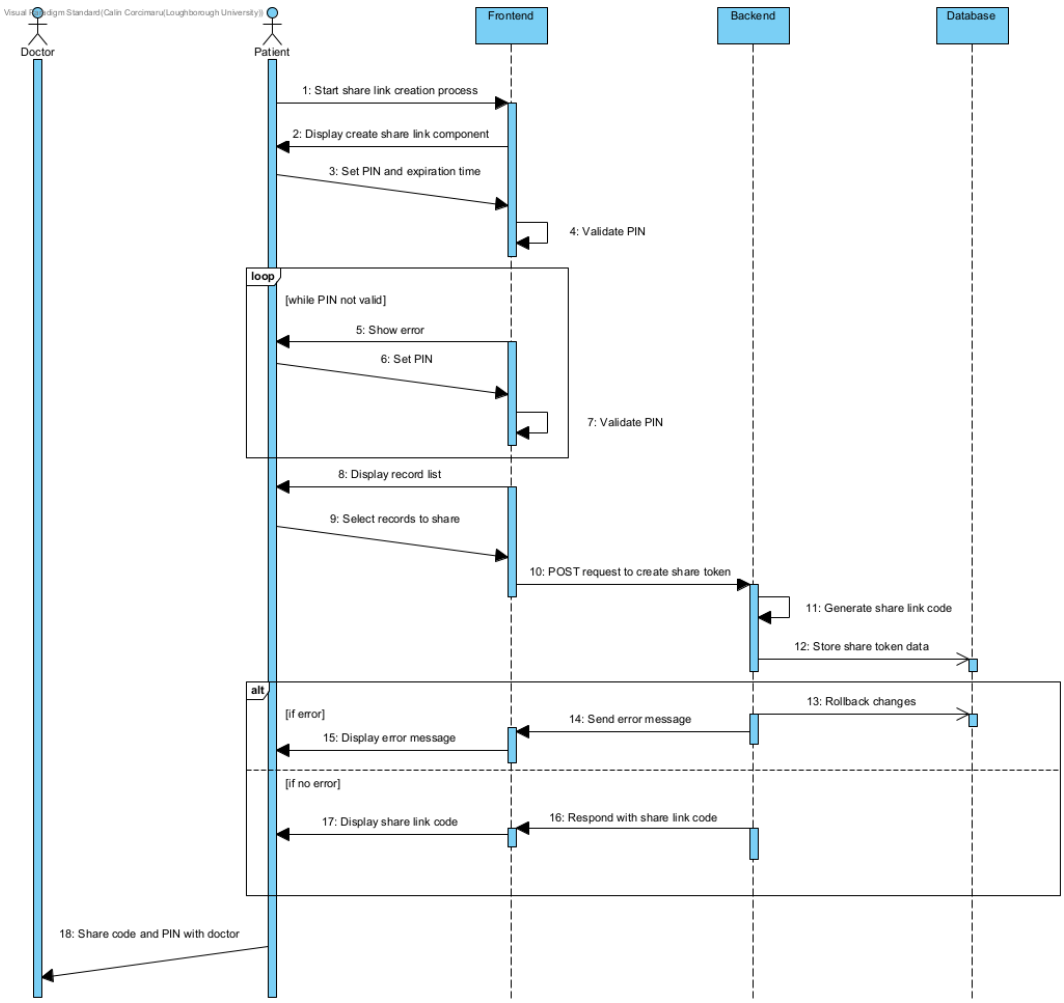


Figure 5.24: UML Sequence Diagram — Creating share link

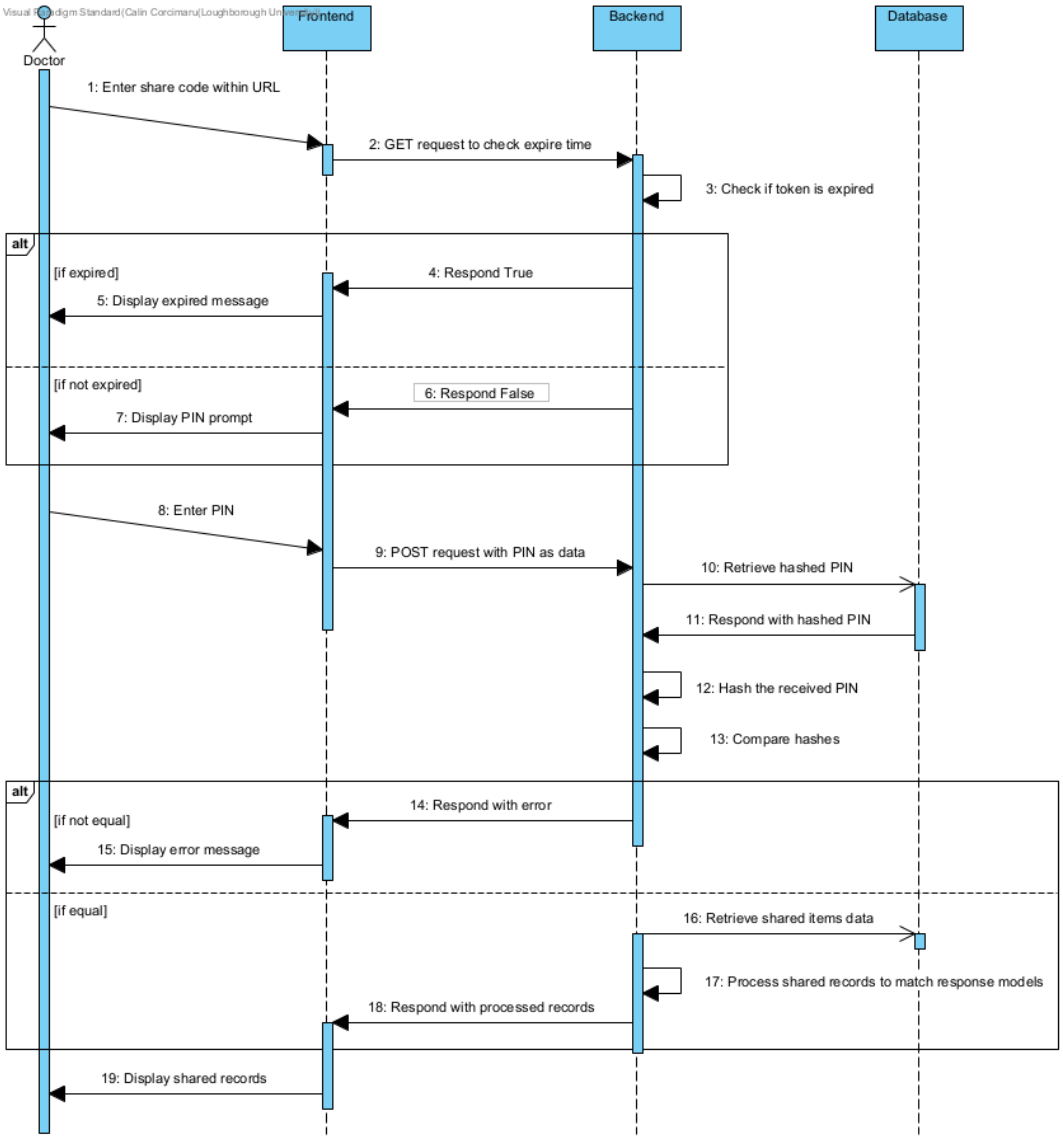


Figure 5.25: UML Sequence Diagram — Accessing shared link

5.6.2 User interface implementation

Based on initial wireframes designs, the link creation component was located in the dashboard. To ensure a streamlined user experience, the share creation process was structured as an easy to follow and guided three-step process. The first step involved configuring the expiration time and setting the PIN, which can be seen in figure 5.26. The second step allowed users to choose the specific records to share with healthcare practitioners, displayed in figure 5.27. The final step confirmed the share link creation and displayed the generated link, as shown in figure 5.28.

Figure 5.26: 1st step of share link creation

The frontend counterpart to the security measures was implemented next. As described in the previous section, the system would check the validity of the share link and if valid, would prompt the user to enter the PIN code. Both valid and invalid link outcomes can be seen in figures 5.30 and 5.29.

The actual results page was developed last. One challenge quickly encountered was how to display the shared information in a single page. The amount of information displayed would not work with previously used methods such as PrimeVue's DataView or DataTable. Instead, TabView was used to support multiple record types onto one page.

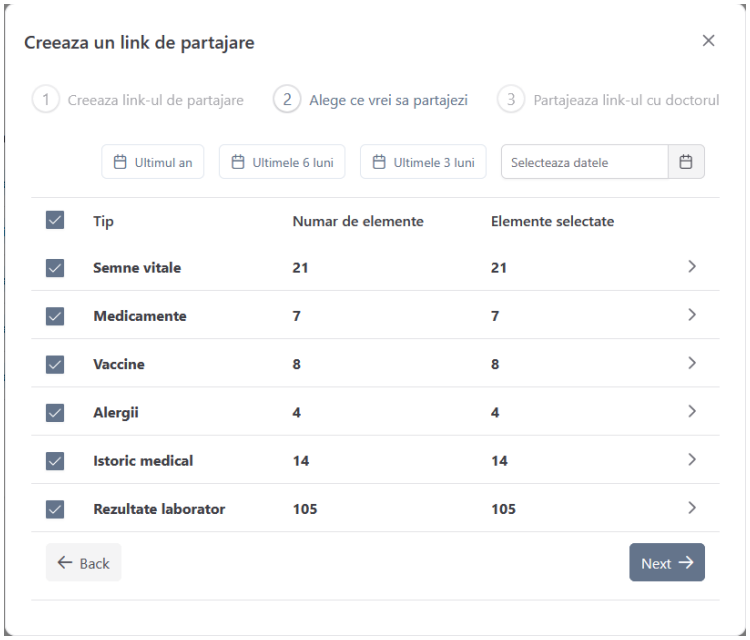


Figure 5.27: 2nd step of share link creation

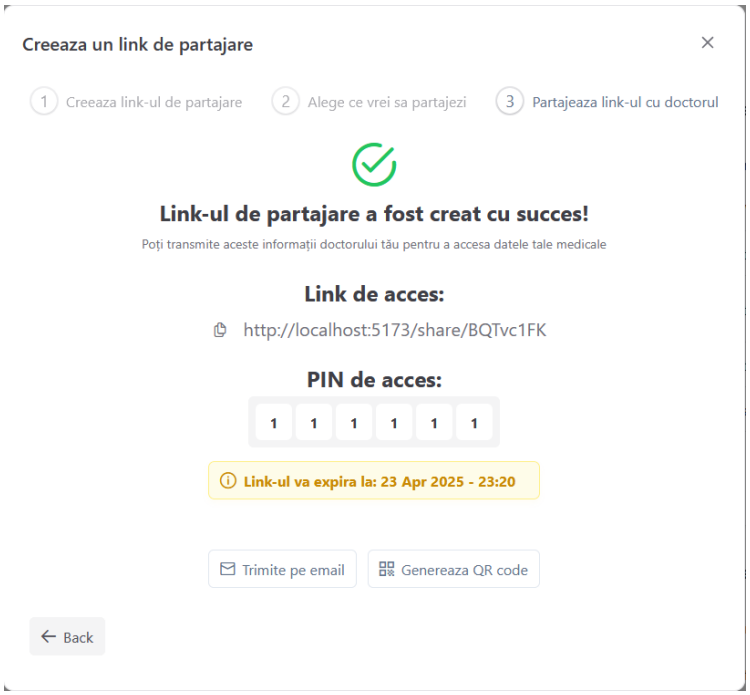


Figure 5.28: 3rd step of share link creation

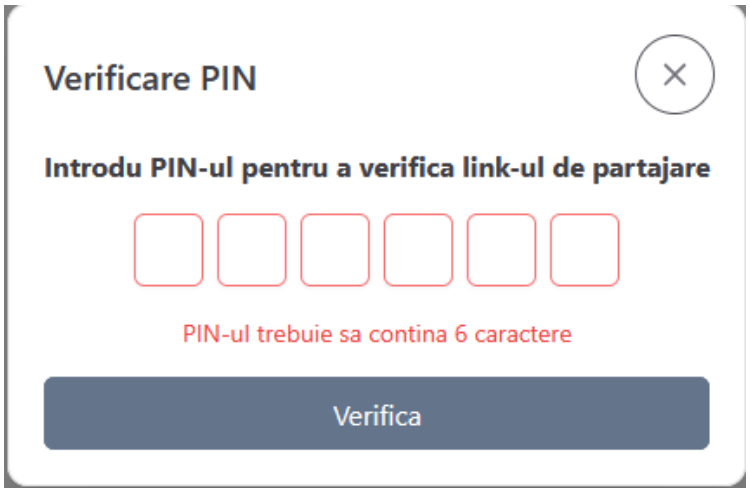


Figure 5.29: PIN prompt dialog

Figure 5.30: Expired link dialog

A combination of TabView and DataTable was used to organise the different record categories — the former would allow the user to switch different tabs representing different records (Vaccines, Allergies, etc), while the latter would contain and display the actual record data within each tab. This ensured that the user could navigate between different records without requiring a page load or additional API calls, which also greatly simplified the development process. The tabs and table columns were dynamically created based on the data received from the backend to accommodate any combination of record data.

To further expand the user experience on the shared page, basic user information and a countdown timer was added at the top of the page. An example of share page can be seen in figure 5.31.

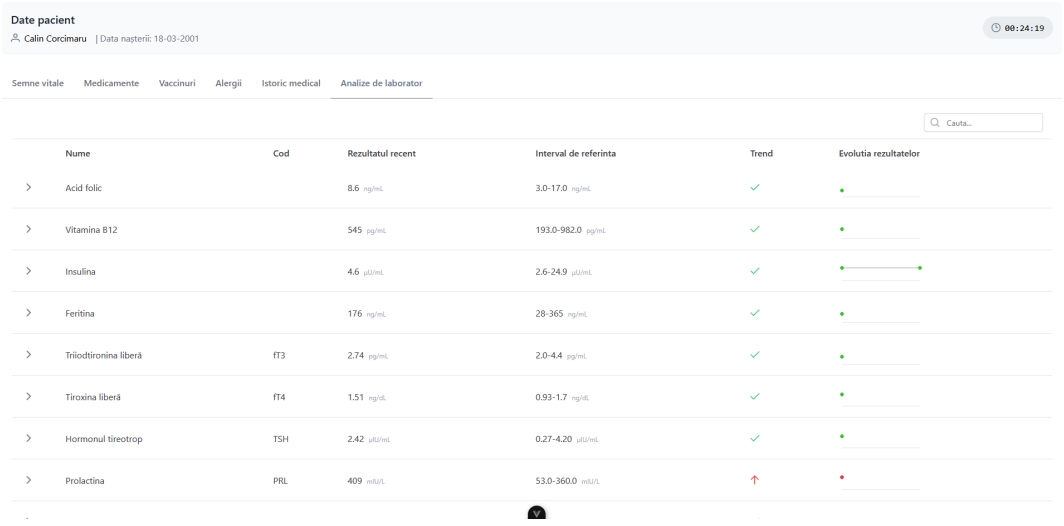


Figure 5.31: Share page — Desktop version

5.6.3 Backend

A new table was created in the database to store the share tokens, which contained metadata such as the hashed PIN, share code, expiration time and the shared items. Two different methods of storing the shared items were considered: a new table containing the record type and its respective UUID, or using a JSON field to store the shared records in the token table. The first method was quickly dismissed due to complex processing and database queries needed to be executed to retrieve and process the items into their response models. Instead, the second method was chosen, as it offered several advantages:

1. Reduced processing overhead and number of database queries

2. Storage of records in a ready-to-use format (response model)
3. Simplified record retrieval process
4. Future compatibility with new record types

The updated ERD can be seen in figure 5.32.

5.6.4 Challenges encountered

Two notable challenges influenced this sprint's development process. Firstly, the approaching project deadline resulted in a shift in focus towards core functionality and away from additional features or refinements. This particularly affected the user interface of the share page, which was simplified to accommodate the time constraints.

Multilingual conflicts was another challenge encountered. The combination of a Romanian frontend with an English-based database and codebase created name inconsistencies, which were especially noticeable in dynamically generated content such as table headers. This discrepancy stemmed from developing the application within a English-based academic context, while targeting a Romanian-speaking audience. While acceptable for a prototype implementation, a production system would require comprehensive internationalisation support to ensure a consistent user experience.

5.6.5 Requirements completed

- The system must allow the patient to generate a shareable link to provide access to their medical records.
- When creating the shareable link, the system must allow the patient to set an expiration date for the link.
- When creating the shareable link, the system should allow the patient to set an access password for the link.
- When creating the shareable link, the system should allow the patient to select which records to share with the doctor.

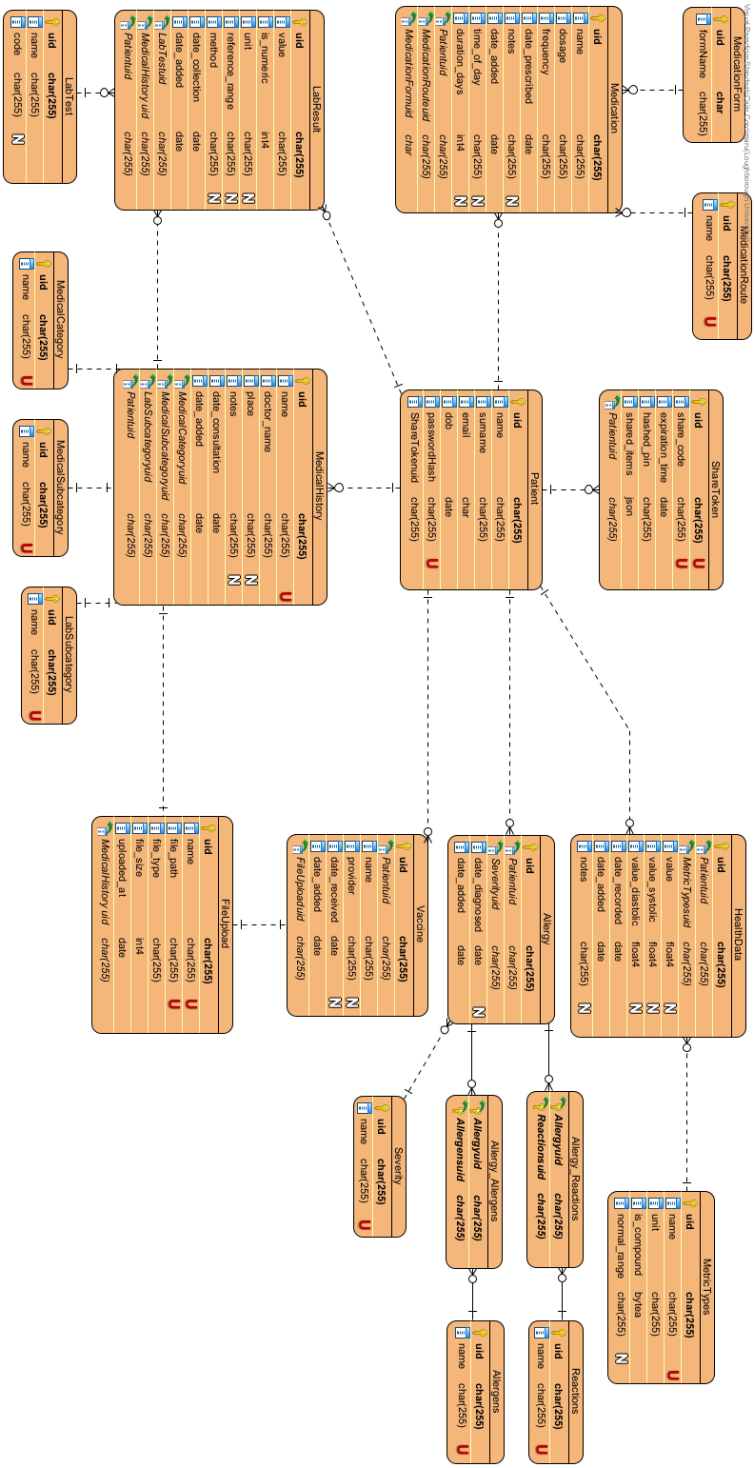


Figure 5.32: Updated Entity Relationship Diagram Sprint #6

Chapter 6

Evaluation

This chapter will explore the testing process of the application's frontend and backend, followed by the evaluation of the MLLM and ending with brief analysis of stakeholder feedback.

6.1 Frontend Testing

Frontend testing was conducted manually due the time constraints, component complexity and lack of experience with automated testing frameworks. To ensure consistency in the testing procedure, structured test cases was created for most components. These test cases used the Gherkin syntax, which follows the Given-When-Then format (The Cucumber Open Source Project, 2025).

An example of test cases for the share link record selection functionality is shown below. The testing outcomes can be viewed in figures 6.1, 6.2 and 6.3, which demonstrate the implementation of the selection behaviour.

To ensure an even coverage of all implemented components, a site map was created to document all pages, components and their relationships. This figure provided a high-level system overview and aided the testing progress tracking. The site map can be seen in figure 6.4.

```
Feature: Record selection for share link

Scenario: User selects all record types to be shared
    Given the user is on the share link page
    When the user selects the top checkbox to select all record types
    Then all record types should be selected for sharing
    And the all children checkboxes should be selected as well

Scenario: User selects specific record types to be shared
    Given the user is on the share link page
    When the user selects a specific record type to be shared
    Then only the selected record types should be selected for sharing
    And the respective children checkboxes should all be selected as well

Scenario: User selects specific children records to be shared
    Given the user is on the share link page
    When the user selects specific children records to be shared
    Then only the selected children records should be selected for sharing
    And the respective parent checkbox should not be selected
    And the top child checkbox should not be selected
```

Listing 6.1: Test cases for selecting records to be shared in a share link

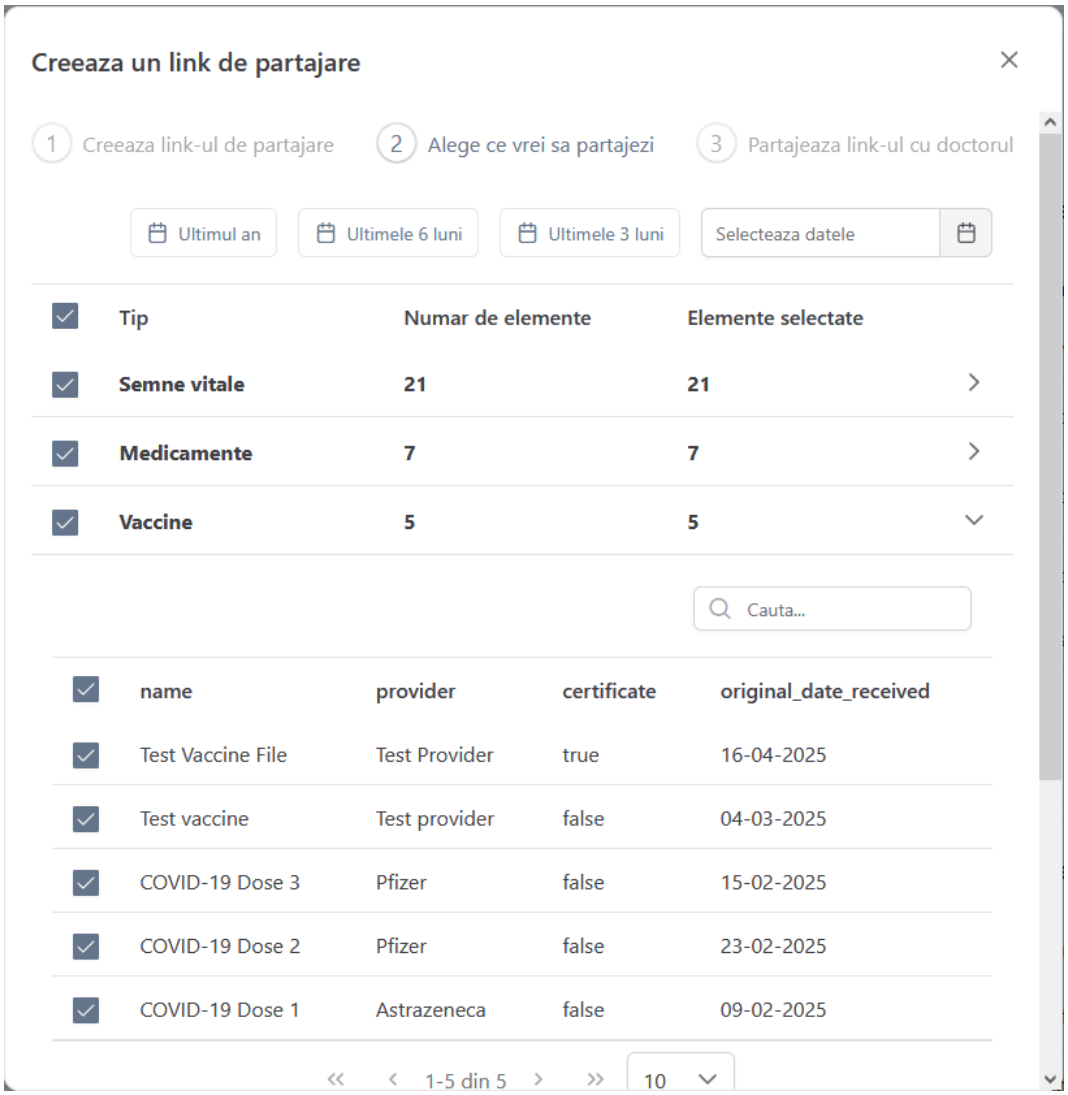


Figure 6.1: Test Scenario #1

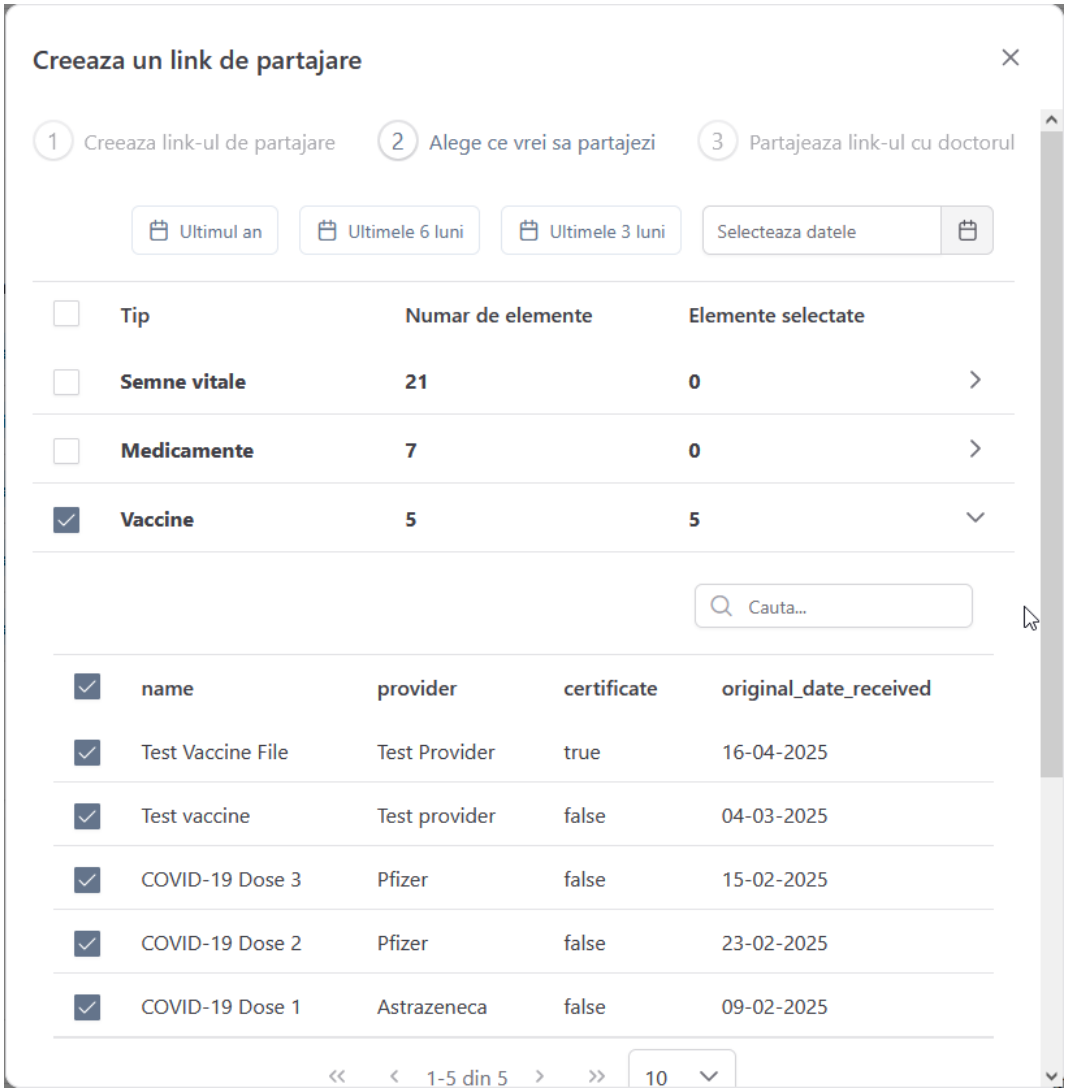


Figure 6.2: Test Scenario #2

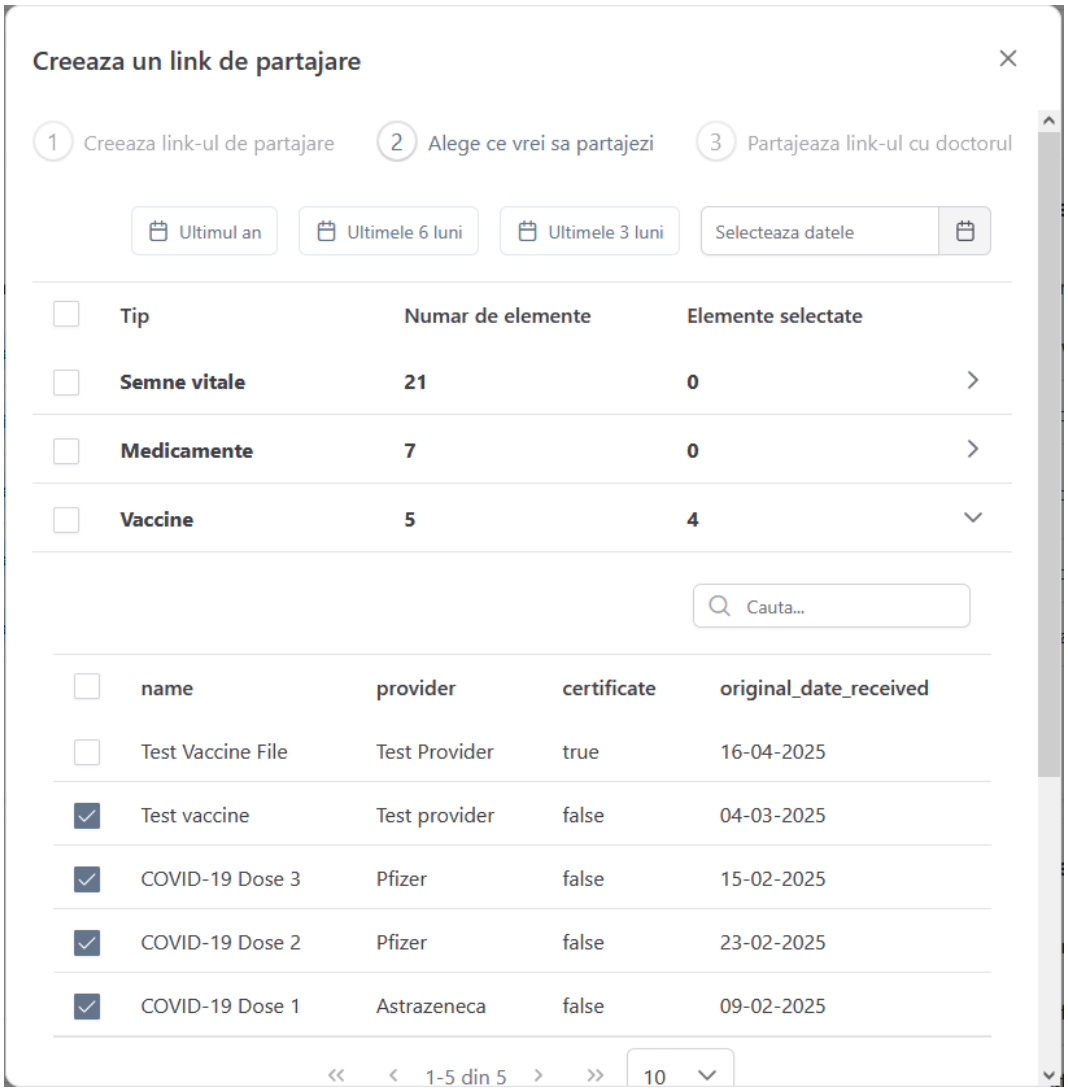


Figure 6.3: Test Scenario #3

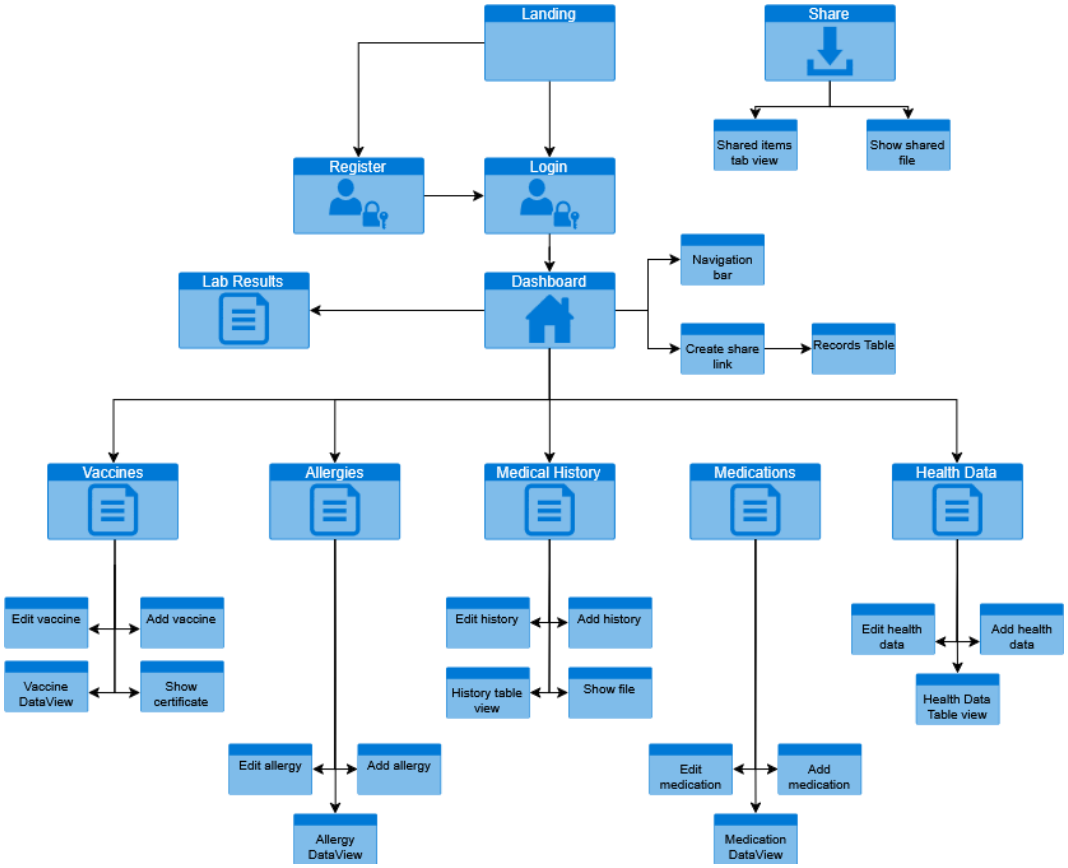


Figure 6.4: PHR System Site Map

6.2 Backend Testing

Backend testing was conducted through two approaches: indirect testing, done in parellel with the frontend, and direct testing by using Postman for the API endpoints. The backend's reduced complexity, compared to the frontend, allowed for a more straightforward testing process. The backend functionality was often tested in conjunction with the frontend, as the frontend components relied on the backend to function correctly. This meant that many of the test cases were already covered during the frontend testing process.

To ensure correct endpoint behaviour and functionality, the endpoints were additionally tested using Postman, without relying on the frontend components. The following example illustrates the testing of the share link validation endpoint:

```
    @router.get("/share/{share_code}", status_code=status.HTTP_200_OK)
@limiter.limit("5/minute")
async def check_share_token(
    request: Request,
    share_code: str,
    session: Session = Depends(get_session)):
    share_token = session.exec(select(ShareToken).where(ShareToken.
    share_code == share_code)).first()

    if not share_token:
        raise HTTPException(status_code=status.HTTP_404_NOT_FOUND,
    detail="Share token not found")

    if share_token.expiration_time < datetime.now():
        raise HTTPException(status_code=status.HTTP_410_GONE, detail="
    Share token has expired")

    return {"valid": True}
```

Listing 6.2: Example endpoint for checking a share link for expiration time

The endpoint above validated three scenarios: invalid, expired and valid share tokens. Each scenario would be tested individually for the correct status code and response message using Postman, as can seen in figures 6.5, 6.6 and 6.7.

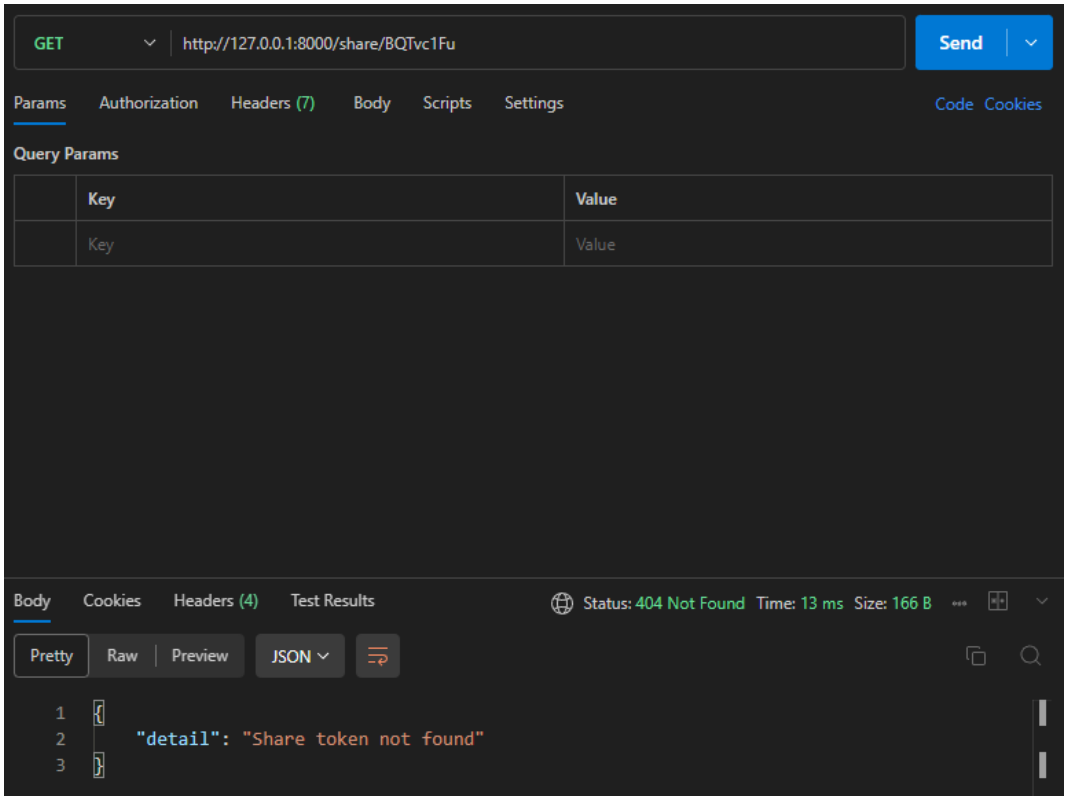


Figure 6.5: Postman test — share token not found

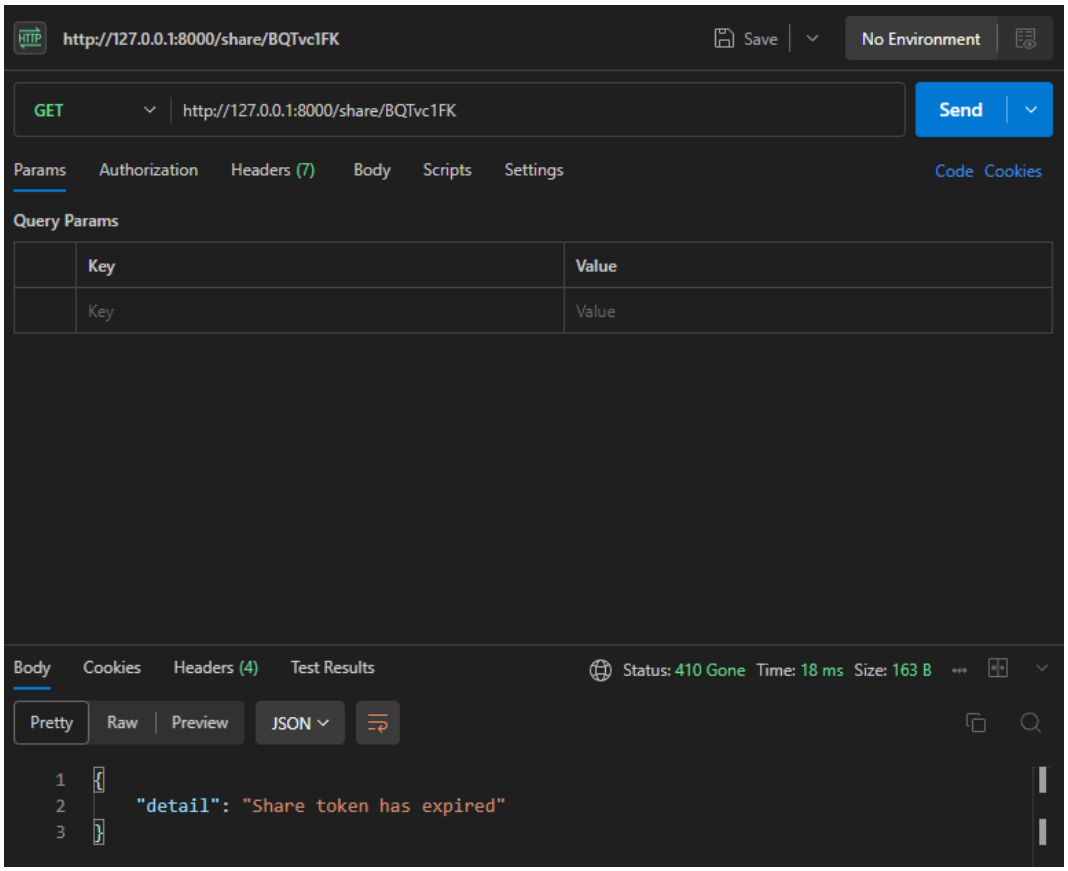


Figure 6.6: Postman test — share token expired

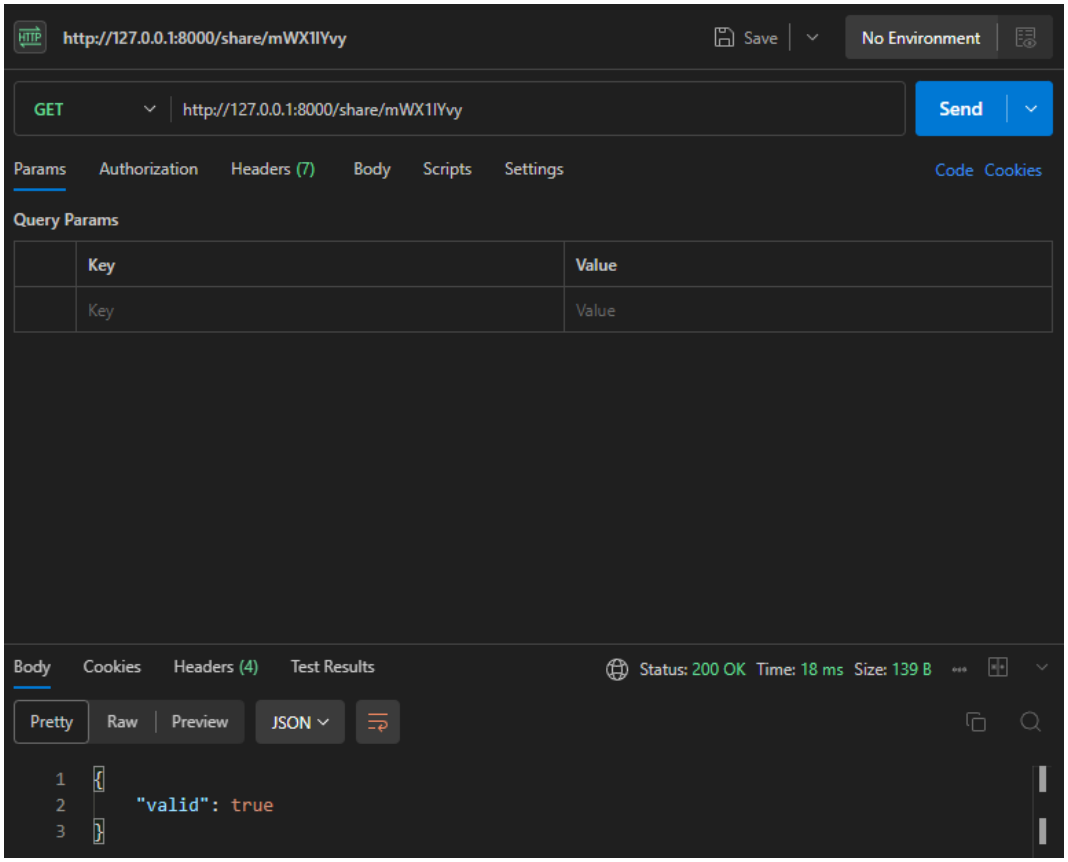


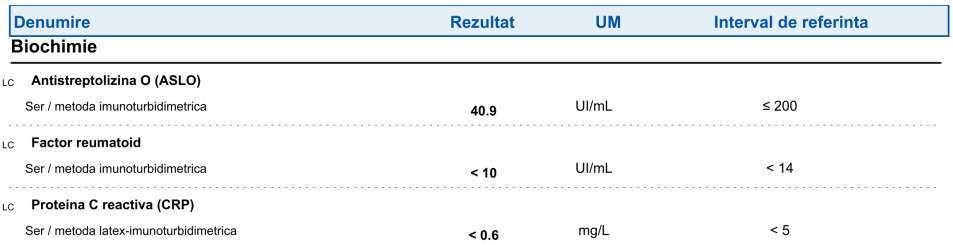
Figure 6.7: Postman test — share token valid

6.3 LLM Evaluation

Gemini 2.0 Flash was selected as the MLLM that powers the lab results extraction feature based on its generous free tier allocation (15 requests/minute with 1,000,000 tokens/minute), robust multimodal capabilities and competitive performance. As of writing this section, this model ranked among the top 10 models on the Chatbot Arena leaderboard for both language and vision tasks created by Chiang et al., 2024, showcasing its strong performance relative to similar models. As described by its authors, the leaderboard is an 'open platform for crowdsourced AI benchmarking', which generates live leaderboards based on user feedback from comparing responses from different models.

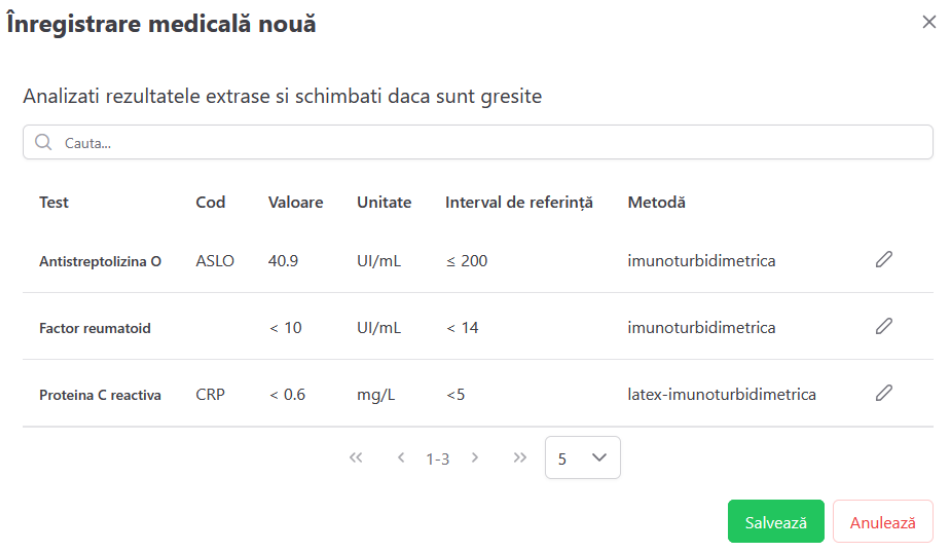
Further model evaluation on its lab result extraction capabilities included testing the model's performance on a set of documents from different healthcare institutions in Moldova. Three distinct documents were used for the evaluation — 2 from private lab networks and 1 from a private hospital. These institutions were chosen for their popularity and wide network of locations across the country (ALFA diagnostica, n.d.; Synevo Moldova, n.d.; Medpark, n.d.). The anonymised documents, alongside their extracted results, can be seen in figures 6.8, 6.9 and 6.10.

In conclusion, the evaluation demonstrated Gemini 2.0 Flash's strong quality and consistent performance across different documents, succesfully extracting test names, codes, values, units, reference ranges and methods. By using a single prompt for all documents, the model was able to deliver consistent results, regardless of the document's format or content, validating its selection for the task at hand.



Denumire	Rezultat	UM	Interval de referinta
Biochimie			
LC **Antistreptolizina O (ASLO)**			
Ser / metoda imunoturbidimetrica	**40.9**	UI/mL	≤ 200
LC **Factor reumatoid**			
Ser / metoda imunoturbidimetrica	**< 10**	UI/mL	< 14
LC **Proteina C reactiva (CRP)**			
Ser / metoda latex-imunoturbidimetrica	**< 0.6**	mg/L	< 5

(a) Lab document



(b) Extraction result

Figure 6.8: Synevo extraction results

Hemograma cu formula leucocitară

Sânge integral. Metoda: Citometria în flux cu fluorescentă / Focalizarea hidrodinamică / SLS hemoglobină

Parametru		Rezultat		Valori de referință	Unități de măsură
Eritrocite (numărul absolut)	RBC	5,62		4,3 — 5,7	x10^6/µL
Hemoglobină	HGB	14,30		13,2 — 17,3	g/dL
Hematocrit	HCT	43,10		39 — 49	%
Eritrocite microcitare (procentul)	MicroR (%)	**8,60**	▲	0,5 — 5,8	%
Eritrocite macrocitare (procentul)	MacroR (%)	4,90		3,3 — 5,1	%
Eritrocite nucleate (numărul absolut)	NRBC (#)	0,00		0,00	x10^3/µL
Eritrocite nucleate (procentul)	NRBC (%)	0,00		0,0	%
Volum eritrocitar mediu	MCV	**76,70**	▼	80,0 — 99,0	fl

(a) Lab document

Înregistrare medicală nouă

Analizati rezultatele extrase si schimbati daca sunt gresite

Cauta...

Test	Cod	Valoare	Unitate	Interval de referință	Metodă
Eritrocite (numărul absolut)	RBC	5.62	x10^6/µL	4.3-5.7	
Hemoglobină	HGB	14.30	g/dL	13.2-17.3	
Hematocrit	HCT	43.10	%	39-49	
Eritrocite microcitare (procentul)	MicroR (%)	8.60	%	0.5-5.8	
Eritrocite macrocitare (procentul)	MacroR (%)	4.90	%	3.3-5.1	
Eritrocite nucleate (numărul absolut)	NRBC (#)	0.00	x10^3/µL	0.00	
Eritrocite nucleate (procentul)	NRBC (%)	0.00	%	0.0	
Volum eritrocitar mediu	MCV	76.70	fl	80.0-99.0	

(b) Extraction result

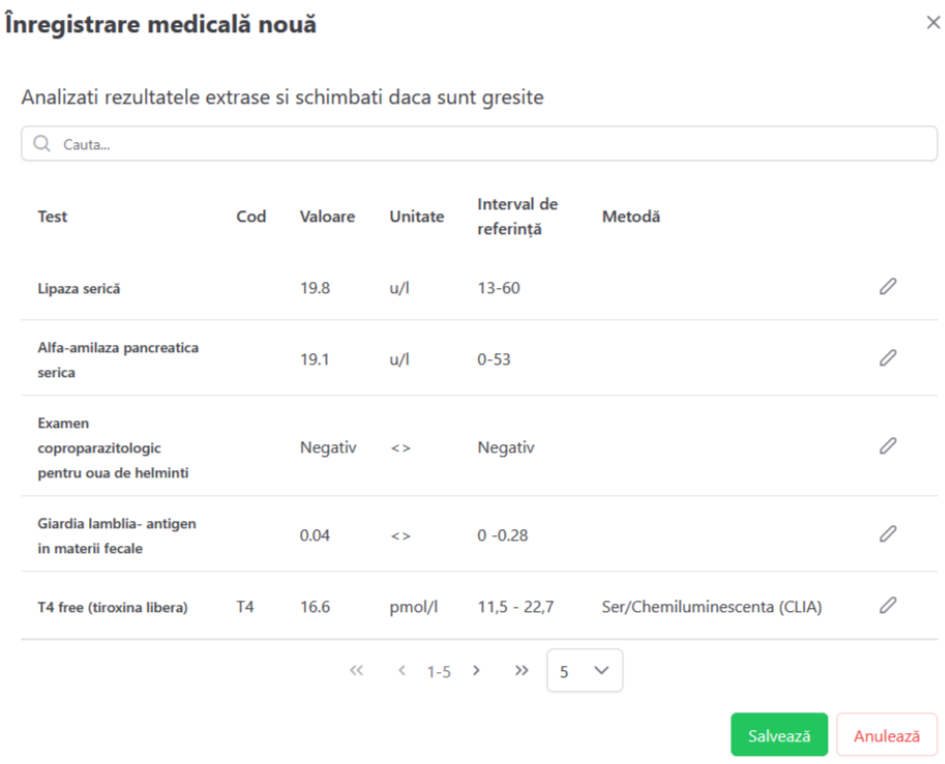
Figure 6.9: Alfalab extraction results

BULETIN DE ANALIZE MEDICALE
LABORATOR MEDPARK

Data	Denumire		Rezultat	Interval de referință	UM
[redacted]	**BIOCHIMIE**				
	BIOCHIMIE GENERALA din sange si urina				
	Lipaza serică		19.8	13 - 60	u/l
	Alfa-amilaza pancreatica serica		19.1	0 - 53	u/l
	MICROBIOLOGIE				
	Examen coproparazitologic pentru oua de helminti		Negativ	Negativ	<>
	Giardia lamblia- antigen in materii fecale		0.04	0 - 0,28	<>
	IMUNOLOGIE				
	MARKERI ENDOCRINI				
	T4 free (tiroxina libera) (Ser/Chemiluminescenta (CLIA))		16.6	11,5 - 22,7	pmol/l
	MARKERI BOLI AUTOIMUNE				
	Anti- TPO (anticorpi anti-peroxidaza) (Ser/Chemiluminescenta (CLIA))		17.3	0 - 34	IU/mL

lab

(a) Lab document



(b) Extraction result

Figure 6.10: Medpark extraction results

6.4 Stakeholder feedback

The system evaluation process concluded with final stakeholder feedback, which was collected through live demo sessions via virtual meetings. Stakeholders were encouraged to provide their feedback in a free format, without any structured questionnaire or formal feedback collection process, allowing for a natural and open discussion. Anonymised excerpts can be seen below, with full feedback available in appendix C.

Feedback 1, from a doctor with interest in digital healthcare applications:

> 'I appreciated the concept of the project and the way all essential components of a Personal Health Record were integrated: general, clinical, and paraclinical data, consultations, physician access, and data security. The platform is strongly patient-centered, enhancing access to personal medical data and significantly contributing to patient empowerment ... From my perspective as a medical doctor specialized in public health and an expert in digital transformation of healthcare, I see great potential in this project and I am committed to offering the necessary support for its further development'

Feedback 2, from a patient and healthcare user:

> 'I was really impressioned with the application on the digital health records book you have developed and presented to me. As a BA you have managed to capture the most important needs and then translated them into useful features included in the solution developed ... By developing this solution you have given a new opportunity to Moldova to make one big step forward to the digitisation, and hope it will be largely used by moldovans as the country offers them also other related nice opportunities like high speed internet access and good coverage by 5G network'

Chapter 7

Conclusion and Future Work

The final chapter of this report will reflect potential system improvements, explore further development opportunities, analyse key lessons learned and assess the objectives set at the beginning of the project.

7.1 Areas of Improvement

Despite overwhelming stakeholder feedback, several areas were identified to benefit from further improvement before any real-world deployment. These areas include security, compliance, auditing and user experience. The following subsections will briefly discuss these areas and how they can be improved in the future.

7.1.1 Security and data encryption

In its current state, the application incorporates only basic security measures, which would be considered insufficient for a live product. As such, the system would benefit from the following security improvements:

1. **Data at rest encryption:** Currently, all patient information is stored in a plaintext format in the PostgreSQL database. A real-world implementation would require, at a minimum, field-level encryption of sensitive data. If required, additional options such as disk or database-level encryption could be considered.
2. **Authentication system enhancements:** The current session-based authentication method would benefit from replacement with a more robust solution, such as a combination of JWT tokens. This could improve security, ensure mobile application com-

patility and even enable integration with third-party authentication providers like Auth0 or Mpass, which can offer advanced security features like two-factor authentication.

3. **Data in transit encryption:** HTTPS implementation with proper SSL certificate configuration will be an essential step to take before deploying the system to ensure secure communication between the client and server.

7.1.2 Logging and monitoring

The current system implementation lacks any monitoring capabilities. Thus, the integration of a logging system will be a crucial feature for the following reasons:

1. **Error handling:** Logs can be an invaluable resource when it comes to debugging and error handling. This in turn may aid in development and testing efforts, ensuring increased quality of the system.
2. **Auditing:** Audit trails are essential for tracking access and changes to sensitive data, such as medical records. This is particularly critical when patients share and potentially edit their data before doctor consultations, ensuring accountability and data integrity.
3. **Compliance:** Proper logging and monitoring can help ensure compliance with data protection regulations by providing a clear record of data access and modifications.
4. **Anomaly detection:** Monitoring allows the detection of anomalies in system or user behaviour, which could imply potential security incidents or breaches.

7.1.3 Legal and regulatory compliance

Processing of medical data will require the application to comply with a series of legal and compliance requirements, depending on the user base and system location. At a minimum, a production system will need to ensure the following:

1. Compliance with Moldova's data protection and healthcare regulations
2. Alignment with GDPR regulation
3. Creation of a privacy and data protection policy, which will outline how the system collects, uses, stores and processes data
4. Execution of regular audits and checks to ensure compliance with the above policies and regulations

Additionally, the use of a cloud-based MLLM may introduce additional legal and compliance considerations regarding data processing and storage practices. This may in turn require a transition to locally-hosted models to reduce the risk of data misuse, which may add complexity and cost to the system.

7.1.4 User experience improvements

The current implementation of the user interface prioritised functionality over design and user experience. As such, the system would benefit from a retrofit of the user interface to improve usability and accessibility. This could include:

1. **Native mobile applicationL** In its current state, the system is only available as a web application. A native mobile application would greatly improve accessibility and expand the potential user base, considering Moldova's extensive network coverage (Anrceti, 2024).
2. **Accessibility:** Implementation of accessibility features to ensure the system can be used by all users, regardless of their abilities. This could include support for screen readers, keyboard navigation and other assistive technologies.
3. **User interface redesign**: While functional, the current user interface lacks a modern design. A redesign could improve the overall user experience, making it more intuitive and user-friendly.

7.2 Potential feature expantions

Beyond refinement of existing features, the application could significantly benefit from extending its functionality. This section will explore potential additions to the system that could enhance its capabilities and improve the overall user experience. These features were either proposed by stakeholders or considered during the development process but not implemented.

7.2.1 Mpass integration

Integration with Moldova's governmental electronic authentication and authorization system, Mpass, could provide multiple benefits:

1. Simplified user registration and authentication process by reusing existing government-based digital identities
2. Access to patient demographic information from government databases

3. Enhanced security through existing identity verification processes, such as electronic signatures
4. Potential for future integration with other government services

7.2.2 Expanding AI features

The current MLLM implementation is limited to only extracting lab results information from uploaded files. Potential additions that would improve user experience with minimal AI intervention could include:

1. Automated processing of uploaded medical files, requiring minimal user input to create a medical history record
2. Intelligent categorisation of documents based on their content, allowing for smarter organisation of medical records
3. Intelligent comments and notes generation based on extracted information, providing patients with additional context and insights into their medical history

7.2.3 Integration with lab providers

The lab results extraction feature currently involves manual uploading of files by the user. A more direct integration with lab providers could streamline this process, allowing for faster and more accurate processing of results. This could be achieved through creating separate, secure email channels or even direct API integration with lab systems. Either option would bring significant benefits, such as:

1. Automated results retrieval and delivery to the systems
2. Elimination of manual entry and file uploads, reducing the risk of human error
3. Reduced latency between test completion and results availability

7.2.4 Enhanced doctor-patient communication

Late stakeholder feedback identified a potential opportunity to implement a direct communication channel between doctors and patients. Expanding the application with this feature could enable doctors to create new records or add notes to existing records, which would be visible to patients. This would bring the following benefits:

1. Reduced risk of miscommunication between doctors and patients during or after consultations

2. Improved patient engagement and understanding of their medical history and treatment plans
3. Future potential for telemedicine features, through messages or video consultations

7.2.5 Integration with 3rd party providers

Finally, the system could benefit from integration with 3rd party providers that specialise in health-related services. This could include:

1. **Health tracker apps:** Integration with health tracker applications, such as Google Fit or Apple Health, could allow for automatic syncing of relevant data, providing a more comprehensive view of the patient's medical history.
2. **Medical imaging:** Special formats, such as DICOM, are commonly used for medical imaging. As the system currently does not support these formats, integration with specific providers could allow to display X-rays, MRIs and other digital results directly in the application.

7.3 Lessons Learned and Reflections

This section will focus on the valuable insights provided by the project experience that could be applicable to similar initiatives in the future. The lessons learned include key technical and non-technical takeaways from the project, as well as reflections on the overall experience of working on a final year project.

7.3.1 Task prioritisation and time management

A critical lesson learned was the importance of prioritising tasks and managing time effectively, especially on projects with multiple components. The issue occured when the project requirements were not properly prioritised, leading to establishing of a wrong order of implementation. As a result, development began with less critical features, such as vaccines, allergies and medication, rather than the core features like lab extraction or record sharing. This misalignment of priorities resulted in wasted time and resources on less impactful elements, creating time pressure towards the end of the project and limiting the ability to expand the application in areas that required more attention, such as security.

To address this issue in future projects, the following recommendations will be considered:

1. Regular priority re-evaluation with team members and stakeholders to ensure a shared understanding of the the future direction of the project

2. Utilising prioritisation methods such as backlog refinement from Scrum together with priority categorisation approaches like MoSCoW
3. Aligning initial priorities with stakeholders at the beginning of a project

7.3.2 Importance of the team

Working independently has highlighted the value of teamwork and collaboration in software development. It quickly became apparent that the project would have benefitted from a team of individuals with diverse skills and expertise, such as:

- A dedicated business analyst to manage requirements, stakeholder communication and project documentation
- A design specialist to optimise the user experience and user interface
- A quality assurance engineer to ensure the system's functionality and reliability
- A project manager to oversee the project's progress and ensure alignment with timelines and goals

This experience has reinforced the importance of collaboration, communication and cross-functional teamwork in software development. While the project was successfully completed, it would have greatly benefitted from the input and expertise of a team of individuals with diverse skills and backgrounds.

7.3.3 Agile & Documentation

Finally, the project very well illustrated the evolution of documentation in software development, especially in an Agile approach. Most documents created during the solution planning phase experienced significant changes throughout the project due to emerging stakeholder feedback and evolving requirements. This was particularly evident in the UML diagrams, which underwent multiple revisions to reflect the most up-to-date state of the application.

As such, this project has demonstrated that effective documentation in software development project should be:

- Adaptable and regularly updated to reflect the current state of the application
- Collaborative, involving input from all team members and stakeholders
- Accessible, ensuring that all team members can easily access and understand the documentation

7.4 Objective assessment

In the introduction, a set of objectives was established to guide the project. The list can be found in section 1.4. This section will assess the extent to which these objectives were met and the overall success of the project.

1. **Stakeholder identification:** The objective to identify 2 to 4 stakeholders across different healthcare perspectives was successfully achieved by the end of November 2024. Three stakeholders were identified, including 2 healthcare practitioners and 1 patient. Their feedback and previous experience with Moldova's healthcare system was used to inform the design and development of the system.
2. **Stakeholder interviews:** The target of conducting at least one interview per stakeholder was exceeded, as the project maintained ongoing communication with all three throughout the the development process. Initial interviews were conducted by December 2024, with follow-up meetings occuring at the end of development sprints to gather feedback.
3. **Literature review:** The objective to carry out a literature review of at least 30 academic sources was successfully achieved by the end of January 2025. The review provided valuable insights into 4 different frameworks and 3 project management methodologies that informed the design and development of the system. Additionally, 3 different PHR systems were explored, which provided a better understanding of industry trends and best practices.
4. **Prototype development:** A working web application prototype was successfully developed by end of April 2025. The system implemented all 10 prioritised requirements, including core features such as lab results extraction, medical record sharing and record management.
5. **Stakeholder validation:** The final objective to present the completed prototype to at least 3 stakeholders was successfully achieved by the end of April 2025. It was presented to stakeholders, who all provided positive feedback and expressed their interest in further collaboration and development through a start-up initiative.

7.5 Final Thoughts

In conclusion, this project provided invaluable practical experience in software development, project and requirements management. It offered a great exposure to the complete SDLC and the challenges that arise at each stage. Additionally, working towards improving the healthcare system in Moldova was a rewarding experience, as it allowed for the application of technical skills to address real-world problems.

The stakeholder feedback has been praticularly encouraging, with most of them expressing their interest in the continuation and expansion of the project idea. Their offer of continued collaboration presents a future opportunity to transform this project into a real-world solution that could meaningfully improve existing healthcare systems in Moldova.

On a personal level, this experience has been particularly relevant to my future career, providing me with a solid foundation in both technical and non-technical aspects of software development. It has allowed me to developed a transferable skillset, which can be directly applicable to upcoming projects and challenges. As an ITMB student, I feel more prepared than ever to bridge the gap between technology and business, manage complex implementation processes and contribute to the success of any team I may be a part of.

References

A7 Software (2022). *Who are we ?* URL: https://www.andaman7.com/en/who-are-we. (Accessed: 11 Nov 2024).

Agenția de guvernare electronică (2025). *Întrebări frecvente*. URL: https://mpass.gov.md/info/faq. Accessed: 25 Feb 2025.

Ahuja, Kabir et al. (2023). "MEGA: Multilingual Evaluation of Generative AI". In: *ArXiv* abs/2303.12528. URL: https://api.semanticscholar.org/CorpusID:257663467.

Akanksha and Akshay Chaturvedi (2022). "Comparison of Different Authentication Techniques and Steps to Implement Robust JWT Authentication". In: *2022 7th International Conference on Communication and Electronics Systems (ICCES)*, pp. 772–779. DOI: 10.1109/ICCES54183.2022.9835796.

Alenezi, Mohammed N, Haneen Alabdulrazzaq, and Nada Q Mohammad (2020). "Symmetric encryption algorithms: Review and evaluation study". In: *International Journal of Communication Networks and Information Security* 12.2, pp. 256–272.

ALFA diagnostica (n.d.). *Despre noi*. URL: https://alfalab.md/ro/pagins/alfa-diagnostica-6015. Accessed: 25 Apr 2025.

Alqudah, Mashal and Rozilawati Razali (2018). "An empirical study of Scrumban formation based on the selection of scrum and Kanban practices". In: *Int. J. Adv. Sci. Eng. Inf. Technol* 8.6, pp. 2315–2322.

Alshamrani, Adel and Abdullah Bahattab (2015). "A comparison between three SDLC models waterfall model, spiral model, and Incremental/Iterative model". In: *International Journal of Computer Science Issues (IJCSI)* 12.1, p. 106.

Amatriain, Xavier (2024). *Prompt Design and Engineering: Introduction and Advanced Methods*. arXiv: 2401.14423 [cs.SE]. URL: https://arxiv.org/abs/2401.14423.

Anderson, Benjamin and Brad Nicholson (2022). *SQL vs. NoSQL Databases: What's the difference?* URL: https://www.ibm.com/think/topics/sql-vs-nosql. (Accessed: 28 Oct 2024).

Anrceti (2024). *Date statistice*. URL: https://harta.anrceti.md/#/statistics. Accessed: 02 May 2025.

Axios (n.d.). *Getting Started*. URL: https://axios-http.com/docs/intro. Accessed: 09 Feb 2025.

Bano, Muneera et al. (2019). "Teaching requirements elicitation interviews: an empirical study of learning from mistakes". In: *Requirements Engineering* 24, pp. 259–289.

BBC (2024). *Moldova country profile*. URL: https://www.bbc.co.uk/news/world-europe-17601580. (Accessed: 22 Oct 2024).

Broadcom (2024). *Why Spring?* URL: https://spring.io/why-spring. (Accessed: 14 Nov 2024).

Centrul Naţional pentru Protecţia Datelor cu Caracter Personal al Republicii Moldova (n.d.). *Legi*. URL: https://datepersonale.md/legislation/national-legislation/legi/. Accessed: 17 Dec 2024.

Cheah, Jun Siang and Kevin Markham (2024). *Free LLM API resources*. URL: https://github.com/cheahjs/free-llm-api-resources. Accessed: 18 Dec 2024.

Cheat Sheets Series Team (2025). *Session Management Cheat Sheet*. URL: https://cheatsheetseries.owasp.org/cheatsheets/Session_Management_Cheat_Sheet.html. Accessed: 24 Feb 2025.

Chiang, Wei-Lin et al. (2024). *Chatbot Arena: An Open Platform for Evaluating LLMs by Human Preference*. arXiv: 2403.04132 [cs.AI].

Ciurcă, Aliona (2021). *Sistemul informațional automatizat din spitale: cum funcționează în R. Moldova și ce cred medicii că ar trebui ajustat*. URL: https://www.zdg.md/stiri/stiri-sociale/sistemul-informational-automatizat-din-spitale-cum-functioneaza-in-r-moldova-si-ce-cred-medicii-ca-ar-trebui-ajustat/. (Accessed: 22 Oct 2024).

Compania Națională de Asigurări în Medicină (n.d.). *Legi*. URL: http://cnam.md/legislatie/legi/. Accessed: 17 Dec 2024.

Davis, Alan et al. (2006a). "Effectiveness of Requirements Elicitation Techniques: Empirical Results Derived from a Systematic Review". In: *14th IEEE International Requirements Engineering Conference (RE'06)*, pp. 179–188. DOI: 10.1109/RE.2006.17.

— (2006b). "Effectiveness of Requirements Elicitation Techniques: Empirical Results Derived from a Systematic Review". In: *14th IEEE International Requirements Engineering Conference (RE'06)*, pp. 179–188. DOI: 10.1109/RE.2006.17.

Django (2024). *Why Django?* URL: https://www.djangoproject.com/start/overview/. (Accessed: 14 Nov 2024).

Donati, Beatrice et al. (2017). "Common mistakes of student analysts in requirements elicitation interviews". In: *Requirements Engineering: Foundation for Software Quality: 23rd*

International Working Conference, REFSQ 2017, Essen, Germany, February 27–March 2, 2017, Proceedings 23. Springer, pp. 148–164.

Dubey, Abhimanyu et al. (2024). *The Llama 3 Herd of Models*. arXiv: 2407.21783 [cs.AI]. URL: https://arxiv.org/abs/2407.21783.

E-Governance Agency (n.d.[a]). *About EGA*. URL: https://egov.md/en/about-ega. (Accessed: 22 Oct 2024).

— (n.d.[b]). *Citizen's Government Portal*. URL: https://egov.md/en/content/citizens-government-portal. (Accessed: 22 Oct 2024).

Etxaniz, Julen et al. (2023). "Do Multilingual Language Models Think Better in English?" In: *North American Chapter of the Association for Computational Linguistics*. URL: https://api.semanticscholar.org/CorpusID:260378999.

FastAPI (2024). *FastAPI framework, high performance, easy to learn, fast to code, ready for production*. URL: https://fastapi.tiangolo.com/. (Accessed: 31 Jan 2025).

Fasten Health (2022). *What is Fasten?* URL: https://docs.fastenhealth.com/. (Accessed: 12 Nov 2024).

Gemino, Andrew, Blaize Horner Reich, and Pedro M. Serrador (2021). "Agile, Traditional, and Hybrid Approaches to Project Success: Is Hybrid a Poor Second Choice?" In: *Project Management Journal* 52.2, pp. 161–175. DOI: 10.1177/8756972820973082. eprint: https://doi.org/10.1177/8756972820973082. URL: https://doi.org/10.1177/8756972820973082.

Giray, Louie (2023). "Prompt engineering with ChatGPT: a guide for academic writers". In: *Annals of biomedical engineering* 51.12, pp. 2629–2633.

Google (2024a). *Advancing medical AI with Med-Gemini*. URL: https://research.google/blog/advancing-medical-ai-with-med-gemini/. Accessed: 17 Nov 2024.

— (2024b). *What is Angular?* URL: https://angular.dev/overview. (Accessed: 14 Nov 2024).

Hada, Rishav et al. (2023). "Are large language model-based evaluators the solution to scaling up multilingual evaluation?" In: *arXiv preprint arXiv:2309.07462*.

Hakimov, Sherzod and David Schlangen (July 2023). "Images in Language Space: Exploring the Suitability of Large Language Models for Vision & Language Tasks". In: *Findings of the Association for Computational Linguistics: ACL 2023*. Ed. by Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki. Toronto, Canada: Association for Computational Linguistics, pp. 14196–14210. DOI: 10.18653/v1/2023.findings-acl.894. URL: https://aclanthology.org/2023.findings-acl.894.

Harrer, Stefan (2023). "Attention is not all you need: the complicated case of ethically using large language models in healthcare and medicine". In: *eBioMedicine* 90.104512. DOI: https://doi.org/10.1016/j.ebiom.2023.104512.

Heart, Tsipi, Ofir Ben-Assuli, and Itamar Shabtai (2017). "A review of PHR, EMR and EHR integration: A more personalized healthcare and public health policy". In: *Health Policy and Technology* 6.1, pp. 20–25. ISSN: 2211-8837. DOI: https://doi.org/10.1016/j.hlpt.2016.08.002. URL: https://www.sciencedirect.com/science/article/pii/S2211883716300624.

Hickey, A.M. and A.M. Davis (2003). "Elicitation technique selection: how do experts do it?" In: *Proceedings. 11th IEEE International Requirements Engineering Conference, 2003.* Pp. 169–178. DOI: 10.1109/ICRE.2003.1232748.

Hosseini, Azamossadat et al. (2023). "Integrated personal health record (PHR) security: requirements and mechanisms". In: *BMC Medical Informatics and Decision Making* 23.1, p. 116.

IEEE (2018). "ISO/IEC/IEEE International Standard - Systems and software engineering – Life cycle processes – Requirements engineering". In: *ISO/IEC/IEEE 29148:2018(E)*, pp. 1–104. DOI: 10.1109/IEEESTD.2018.8559686.

Jiang, Zhengbao et al. (July 2020). "How Can We Know What Language Models Know?" In: *Transactions of the Association for Computational Linguistics* 8, pp. 423–438. ISSN: 2307-387X. DOI: 10.1162/tacl_a_00324. eprint: https://direct.mit.edu/tacl/article-pdf/doi/10.1162/tacl_a_00324/1923867/tacl_a_00324.pdf. URL: https://doi.org/10.1162/tacl%5C_a%5C_00324.

Khan, Javed Ali et al. (2015). "Comparison of requirement prioritization techniques to find best prioritization technique". In: *International Journal of Modern Education and Computer Science* 7.11, p. 53.

Kim, Hyuhng Joon et al. (2022). "Self-Generated In-Context Learning: Leveraging Auto-regressive Language Models as a Demonstration Generator". In: *ArXiv* abs/2206.08082. URL: https://api.semanticscholar.org/CorpusID:249712501.

Labrak, Yanis et al. (2024). *BioMistral: A Collection of Open-Source Pretrained Large Language Models for Medical Domains.* arXiv: 2402.10373 [cs.CL]. URL: https://arxiv.org/abs/2402.10373.

Laplante, Phillip A (2019). *Requirements engineering for software and systems.* 3rd. Auerbach Publications.

Lee, Jisan et al. (2021). "Review of national-level personal health records in advanced countries". In: *Healthcare Informatics Research* 27.2, pp. 102–109.

Li, Cheng et al. (2023). "Large Language Models Understand and Can be Enhanced by Emotional Stimuli". In: URL: https://api.semanticscholar.org/CorpusID:260126019.

Loweth, Robert P. et al. (Mar. 2021). "A Comparative Analysis of Information Gathering Meetings Conducted by Novice Design Teams Across Multiple Design Project Stages". In: *Journal of Mechanical Design* 143.9, p. 092301. ISSN: 1050-0472. DOI: 10.1115/1.

4049970. eprint: https://asmedigitalcollection.asme.org/mechanicaldesign/article-pdf/143/9/092301/6667332/md_143_9_092301.pdf. URL: https://doi.org/10.1115/1.4049970.

Lu, Albert et al. (2023). "Bounding the Capabilities of Large Language Models in Open Text Generation with Prompt Constraints". In: *ArXiv* abs/2302.09185. URL: https://api.semanticscholar.org/CorpusID:257039031.

Luo, Lu (2001). "Software testing techniques". In: *Institute for software research international Carnegie mellon university Pittsburgh, PA* 15232.1-19, p. 19.

Marqas, Ridwan B, Saman M Almufti, and Rasheed Rebar Ihsan (2020). "Comparing Symmetric and Asymmetric cryptography in message encryption and decryption by using AES and RSA algorithms". In: *Xi'an Jianzhu Keji Daxue Xuebao/Journal of Xi'an University of Architecture & Technology* 12.3, pp. 3110–3116.

Medpark (n.d.). *Despre noi*. URL: https://www.medpark.md/cine-suntem/. Accessed: 25 Apr 2025.

Medvalet (2024). *Despre noi*. URL: https://medvalet.ro/. (Accessed: 11 Nov 2024).

Meskó, Bertalan (2023). "Prompt engineering as an important emerging skill for medical professionals: tutorial". In: *Journal of medical Internet research* 25, e50638.

Meta Platforms (n.d.). *React. The library for web and native user interfaces*. URL: https://react.dev/. (Accessed: 14 Nov 2024).

Ministerul Sănătății al Republicii Moldova (2024). *Legislație Națională*. URL: https://ms.gov.md/legislatie/sanatate/legislatie-nationala/. Accessed: 17 Dec 2024.

Mirza, Mahrukh Sameen and Soma Datta (2019). "Strengths and Weakness of Traditional and Agile Processes-A Systematic Review." In: *J. Softw.* 14.5, pp. 209–219.

Mohedas, Ibrahim et al. (2022). "The use of recommended interviewing practices by novice engineering designers to elicit information during requirements development". In: *Design Science* 8, e16. DOI: 10.1017/dsj.2022.4.

Morphy, Tamzin (2020). *Stakeholder Analysis | Definition and best method*. URL: https://www.stakeholdermap.com/stakeholder-analysis.html. (Accessed: 31 Oct 2024).

Mozilla (2025). *Using HTTP cookies*. URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies. Accessed: 24 Feb 2025.

Naiakshina, Alena et al. (2020). "On Conducting Security Developer Studies with CS Students: Examining a Password-Storage Study with CS Students, Freelancers, and Company Developers". In: *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*. CHI '20. Honolulu, HI, USA: Association for Computing Machinery, pp. 1–13. ISBN: 9781450367080. DOI: 10.1145/3313831.3376791. URL: https://doi.org/10.1145/3313831.3376791.

National Bureau of Statistics of the Republic of Moldova (2024). *Population*. URL: https://statistica.gov.md/en/statistic_indicator_details/25. (Accessed: 22 Oct 2024).

Naveed, Humza et al. (2023). "A comprehensive overview of large language models". In: *arXiv preprint arXiv:2307.06435*.

Nazi, Zabir Al and Wei Peng (2024). "Large Language Models in Healthcare and Medical Domain: A Review". In: *Informatics* 11.3. ISSN: 2227-9709. DOI: 10.3390/informatics11030057. URL: https://www.mdpi.com/2227-9709/11/3/57.

Niu, Qian et al. (2024). *From Text to Multimodality: Exploring the Evolution and Impact of Large Language Models in Medical Practice*. arXiv: 2410.01812 [cs.CY]. URL: https://arxiv.org/abs/2410.01812.

Notion Labs (2025). *Want Agile results? Use Notion for Agile management*. URL: https://www.notion.com/use-case/agile-management. (Accessed: 11 Feb 2025).

Ntantogian, Christoforos, Stefanos Malliaros, and Christos Xenakis (2019). "Evaluation of password hashing schemes in open source web platforms". In: *Computers & Security* 84, pp. 206–224. ISSN: 0167-4048. DOI: https://doi.org/10.1016/j.cose.2019.03.011. URL: https://www.sciencedirect.com/science/article/pii/S0167404818308332.

Nurgaliyev, Alibek and Hua Wang (2021). "Comparative study of symmetric cryptographic algorithms". In: *2021 International Conference on Networking and Network Applications (NaNA)*, pp. 107–112. DOI: 10.1109/NaNA53684.2021.00026.

Okta, Inc. (2025a). *Auth0 Overview*. URL: https://auth0.com/docs/get-started/auth0-overview. Accessed: 25 Feb 2025.

— (2025b). *What is OAuth 2.0?* URL: https://auth0.com/intro-to-iam/what-is-oauth-2. Accessed: 25 Feb 2025.

OpenAI et al. (2024). *GPT-4 Technical Report*. arXiv: 2303.08774 [cs.CL]. URL: https://arxiv.org/abs/2303.08774.

OpenJS Foundation (n.d.). *Fast, unopinionated, minimalist web framework for Node.js*. URL: https://expressjs.com/. (Accessed: 14 Nov 2024).

Oracle (2020). *What is a Database?* URL: https://www.oracle.com/uk/database/what-is-database/. (Accessed: 28 Oct 2024).

Papathanasaki, Maria, Leandros Maglaras, and Nick Ayres (2022). "Modern Authentication Methods: A Comprehensive Survey". In: *AI, Computer Science and Robotics Technology*. DOI: 10.5772/acrt.08. URL: https://doi.org/10.5772/acrt.08.

Paradigm, Visual (2024). *What is Unified Modeling Language (UML)?* URL: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-uml/#sequence-diagram. (Accessed: 07 Nov 2024).

Pinia (n.d.). *Introduction*. URL: https://pinia.vuejs.org/introduction.html. Accessed: 09 Feb 2025.

PostgreSQL Global Development Group (2025). *About*. URL: https://www.postgresql.org/about/. Accessed: 09 Feb 2025.

Prenner, Nils, Carolin Unger-Windeler, and Kurt Schneider (2020). "How are hybrid development approaches organized? A systematic literature review". In: *Proceedings of the International Conference on Software and System Processes*, pp. 145–154.

Press, Ofir et al. (2022). "Measuring and Narrowing the Compositionality Gap in Language Models". In: *ArXiv* abs/2210.03350. URL: https://api.semanticscholar.org/CorpusID:252762102.

PrimeTek (n.d.). *Introduction*. URL: https://primevue.org/introduction/. Accessed: 09 Feb 2025.

Reddi, Latha Thamma (2023). *Stakeholder Analysis using the Power Interest Grid*. URL: https://www.projectmanagement.com/wikis/368897/Stakeholder-Analysis--using-the-Power-Interest-Grid. (Accessed: 16 Dec 2024).

Ruparelia, Nayan B. (May 2010). "Software development lifecycle models". In: *SIGSOFT Softw. Eng. Notes* 35.3, pp. 8–13. ISSN: 0163-5948. DOI: 10.1145/1764810.1764814. URL: https://doi.org/10.1145/1764810.1764814.

Schulhoff, Sander et al. (2024). *The Prompt Report: A Systematic Survey of Prompting Techniques*. arXiv: 2406.06608 [cs.CL]. URL: https://arxiv.org/abs/2406.06608.

Sharma, Akshat et al. (2024). "A Secure Mechanism for Password Hash Value Generator with the Security Analysis of Various Hashing Algorithms". In: *2024 4th International Conference on Computer, Communication, Control & Information Technology (C3IT)*, pp. 1–6. DOI: 10.1109/C3IT60531.2024.10829444.

Sharma, Shreta and SK Pandey (2013). "Revisiting requirements elicitation techniques". In: *International Journal of Computer Applications* 75.12.

Singhal, Karan et al. (2023). *Towards Expert-Level Medical Question Answering with Large Language Models*. arXiv: 2305.09617 [cs.CL]. URL: https://arxiv.org/abs/2305.09617.

SQLModel (n.d.). *SQLModel, SQL databases in Python, designed for simplicity, compatibility, and robustness*. URL: https://sqlmodel.tiangolo.com/. Accessed: 09 Feb 2025.

Stack Overflow (2024). *Technology*. URL: https://survey.stackoverflow.co/2024/technology#most-popular-technologies-language. (Accessed: 14 Nov 2024).

Startup Moldova Foundation (2024). *Digital Health Forum 2024: Catalyzing Healthcare Innovation in Moldova*. URL: https://www.startupmoldova.digital/digital-health-forum-2024-catalyzing-healthcare-innovation-in-moldova. Accessed: 03 May 2025.

Sunner, Daminderjit (2016). "Agile: Adapting to need of the hour: Understanding Agile methodology and Agile techniques". In: pp. 130–135. DOI: 10.1109/ICATCCT.2016.7911978.

Svelte (n.d.). *Overview*. URL: https://svelte.dev/docs/svelte/overview. (Accessed: 14 Nov 2024).

Synevo Moldova (n.d.). *Despre noi*. URL: https://synevo.md/despre-noi/. Accessed: 25 Apr 2025.

Tailwind Labs Inc. (2025). *Styling with utility classes*. URL: https://tailwindcss.com/docs/styling-with-utility-classes. Accessed: 09 Feb 2025.

The Cucumber Open Source Project (2025). *Reference*. URL: https://cucumber.io/docs/gherkin/reference. Accessed: 24 Apr 2025.

Tiwari, Saurabh, Santosh Singh Rathore, and Atul Gupta (2012). "Selecting requirement elicitation techniques for software projects". In: *2012 CSI Sixth International Conference on Software Engineering (CONSEG)*, pp. 1–10. DOI: 10.1109/CONSEG.2012.6349486.

United Nations Development Programme (2023). *A 100% digital state: The strategy for digital transformation of the Republic of Moldova for 2023-2030, approved by the Executive*. URL: https://www.undp.org/moldova/press-releases/100-digital-state-strategy-digital-transformation-republic-moldova-2023-2030-approved-executive. (Accessed: 22 Oct 2024).

USMF (2023). *Brief history*. URL: https://usmf.md/en/brief-history-usmf. (Accessed: 22 Oct 2024).

Vailshery, Lionel Sujay (2024). *Most used web frameworks among developers worldwide, as of 2024*. URL: https://www.statista.com/statistics/1124699/worldwide-developer-survey-most-used-frameworks-web/. (Accessed: 14 Nov 2024).

Vaswani, Ashish et al. (2017). "Attention is all you need". In: *Advances in neural information processing systems*.

Vue Router (n.d.). *Getting Started*. URL: https://router.vuejs.org/guide/. Accessed: 09 Feb 2025.

Vue.js (n.d.). *Components Basics*. URL: https://vuejs.org/guide/essentials/component-basics.html. Accessed: 09 Feb 2025.

Xiao, Hanguang et al. (2024). *A Comprehensive Survey of Large Language Models and Multimodal Large Language Models in Medicine*. arXiv: 2405.08603 [cs.CL]. URL: https://arxiv.org/abs/2405.08603.

Xu, Xiaohan et al. (2023). "Re-Reading Improves Reasoning in Large Language Models". In: *Conference on Empirical Methods in Natural Language Processing*. URL: https://api.semanticscholar.org/CorpusID:261696483.

Yas, Qahtan, Abdulbasit Alazzawi, and Bahbibi Rahmatullah (Nov. 2023). "A Comprehensive Review of Software Development Life Cycle methodologies: Pros, Cons, and Future Directions". In: *Iraqi Journal For Computer Science and Mathematics* 4.4, pp. 173–190. DOI: 10.52866/ijcsm.2023.04.04.014. URL: https://journal.esj.edu.iq/index.php/IJCM/article/view/664.

Yin, Shukang et al. (2024). "A survey on multimodal large language models". In: *National Science Review*, nwae403.

You, Evan (2024). *What is Vue?* URL: https://vuejs.org/guide/introduction.html. (Accessed: 14 Nov 2024).

Young, Ralph R (2002). "Recommended requirements gathering practices". In: *CrossTalk* 15.4, pp. 9–12.

Zhang, Duzhen et al. (2024). "Mm-llms: Recent advances in multimodal large language models". In: *arXiv preprint arXiv:2401.13601*.

Zheng, Mingqian, Jiaxin Pei, and David Jurgens (2023). "When"A Helpful Assistant"Is Not Really Helpful: Personas in System Prompts Do Not Improve Performances of Large Language Models". In: URL: https://api.semanticscholar.org/CorpusID:265220809.

Zhou, Ce et al. (2023). "A comprehensive survey on pretrained foundation models: A history from bert to chatgpt". In: *arXiv preprint arXiv:2302.09419*.

Appendix A

Full Requirements List

ID	Requirement	Priority
2.1	The system must provide a secure login mechanism for patients by using a combination of login and password.	Must Have
2.2	The system should provide an option for password recovery.	Should Have

Table A.1: Login Requirements

ID	Requirement	Priority
4.1	The system must allow the patient to generate a shareable link to provide access to their medical records.	Must Have
4.2	When creating the shareable link, the system must allow the patient to set an expiration date for the link.	Must Have
4.3	When creating the shareable link, the system should allow the patient to set an access PIN for the link.	Should Have
4.4	When creating the shareable link, the system should allow the patient to select which records to share with the doctor.	Should Have

Table A.2: Patient Shareable Link Requirements

ID	Requirement	Priority
1.1	The system must be accessible on all modern desktop and mobile-based browsers.	Must Have
1.2	The system must protect sensitive data through encryption at rest and in transit.	Must Have
1.3	The system must hash and salt all passwords before storing them in the database.	Must Have
1.4	The system must encrypt uploaded documents before storing them in the file system.	Must Have
1.5	The system must allow access only to the patient's associated records.	Must Have
1.6	When records are shared, the system must allow a doctor's access only to the selected shared records.	Must Have
1.7	The system must provide clear error messages to users when operations fail or errors occur during data entry.	Must Have
1.8	The system must be able to handle file uploads of at least 10MB and support multiple file formats (PDF, DOC, etc).	Must Have
1.9	The web application should have a responsive design that adapts to different screen sizes.	Should Have
1.10	The application should have a consistent design across all pages	Should Have
1.11	The system interface should be available in the Romanian language	Should Have
1.12	When shared with the doctor via a link, the share link must load within 3 to 5 seconds.	Should Have
1.13	The system should validate the share link using its expiration time and PIN before allowing access.	Could Have

Table A.3: Non-functional Requirements

ID	Requirement	Priority
3.1	The system must allow patients to upload their own medical records in a variety of formats (PDF, DOC, etc).	Must Have
3.2	The system should allow to upload files/documents for other types of records (vaccine certificates, etc).	Should Have
3.2	The system must allow the patient to specify and categorise the type of document they are uploading (lab test, doctor consultation, etc).	Must Have
3.3	When a the lab category is selected during record creation, the system must allow the user to enable AI processing.	Must Have
3.4	If lab extraction was enabled, the system must display the extraction results to the user before sending to the database.	Must Have
3.5	The system must allow the patient to add a description of the document they are uploading.	Must Have
3.6	The system should allow the patient to add a date for the document they are uploading.	Should Have
3.7	The system should allow the patient to add a location for the document they are uploading.	Should Have
3.8	The system should allow the patient to add the doctor name for the document they are uploading.	Should Have
3.9	The system should allow the patient to sort and filter the documents based on the type of document, date, location, and doctor name.	Should Have
3.10	If used on mobile, the system should allow the patient to take a picture of the document and upload it.	Could Have

Table A.4: Document Upload Requirements

ID	Requirement	Priority
5.1	The patient personal cabinet must provide an overview of the patient's history through 3 main sections: personal information, lab tests, and doctor consultations.	Must Have
5.2	The system must display the patient's history in a chronological order in a table format.	Must Have
5.3	The system must have a dashboard view which displays an overview of the most recent information added to the system (latest lab tests, doctor consultations, vaccinations etc).	Must Have
5.4	The system must allow patients to add their own personal information, such as name or date of birth.	Must Have
5.5	The system must allow the patient to add their own allergies.	Must Have
5.6	The system must allow the patient to add their own vaccinations.	Must Have
5.7	When viewing doctor consultations, the system should divide them into categories based on the domain of the doctor (cardiology, neurology, etc).	Should Have
5.8	The system should allow the patient to enter vitals information, such as height, weight, blood pressure, etc.	Should Have
5.9	The system should display the information in both a list or grid view.	Should Have
5.10	When multiple vital and lab entries are made, the system should display a historical graph of the patient's vitals.	Should Have
5.11	The system should allow the patient to switch between viewing the lab tests in the document format or in a tabular, numerical format.	Should Have

Table A.5: Patient Personal Cabinet Requirements

ID	Requirement	Priority
6.1	The system must provide an overview of the patient history through 3 main sections: personal information, lab tests, and doctor consultations.	Must Have
6.2	When shared with the doctor, the system must allow the doctor to only view the patient's history, not edit it.	Must Have
6.3	The system must allow the doctor to view blood tests in a graphical format.	Must Have
6.4	The system must allow the doctor to view blood tests in a numerical, tabular format.	Must Have
6.5	The system must allow the doctor to view the patient's history in a chronological order.	Must Have
6.6	The system must display the doctor consultation and every lab test, except for blood tests, in a free text or document format.	Must Have
6.7	When viewing blood test results, the system should show the source document of the blood test value.	Should Have
6.8	For blood test results, the system should display the normal range values for each test.	Should Have

Table A.6: Shared Patient Information Requirements (Doctor View)

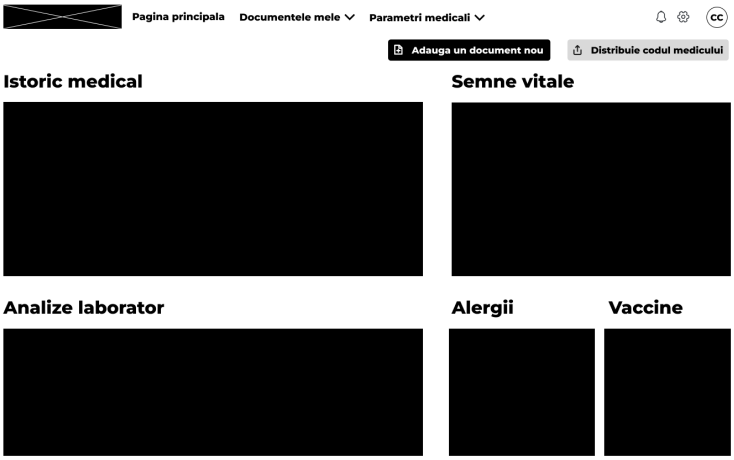
ID	Requirement	Priority
7.1	The system must allow patients to enter their current medication including details such as the name of the drug, dosage, frequency and start/end date.	Must Have
7.2	The system must allow patients to add new medication to their list.	Must Have
7.3	The system could allow patients to set medication reminders.	Should Have
7.4	After entering the medication, the system could allow the patient to track the medication intake.	Could Have

Table A.7: Patient Medication Requirements

Appendix B

Full Wireframes List

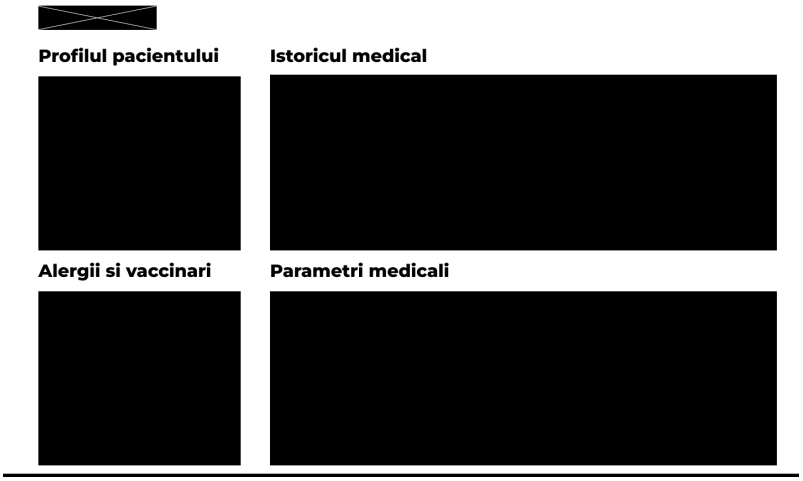
Figure B.1: Mobile version of the Vaccines screen



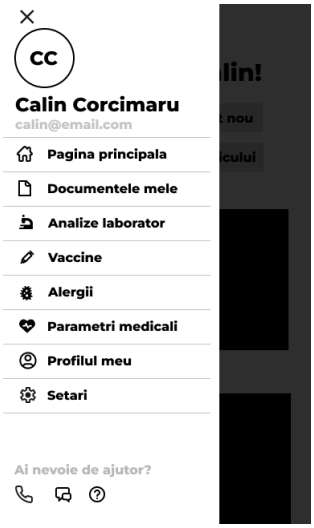
(a) Desktop version

(b) Mobile version

Figure B.2: Desktop and Mobile version of the Dashboard screen

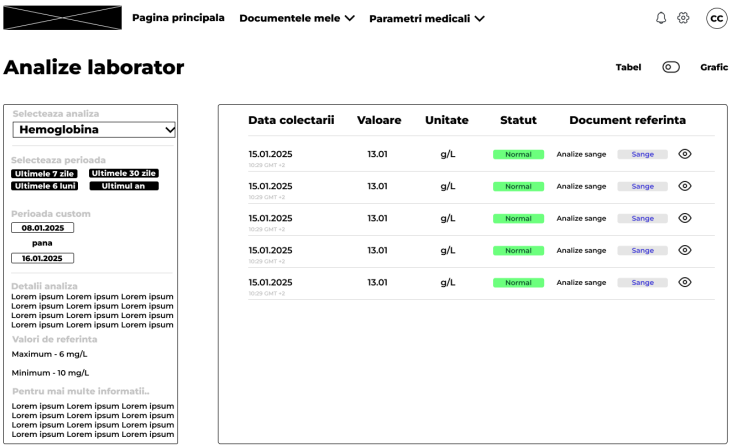


(a) Desktop version



(b) Mobile version

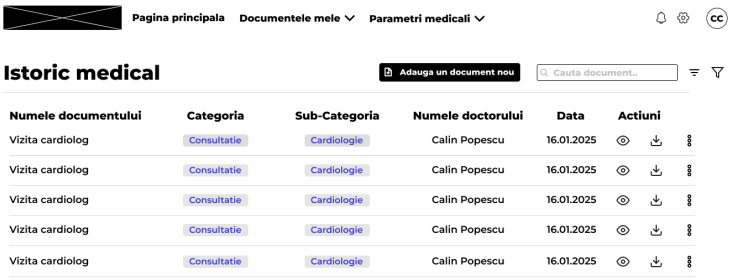
Figure B.3: Desktop and Mobile version of the Doctor View screen



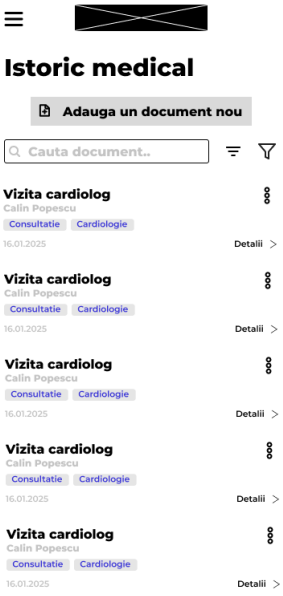
(a) Desktop version

(b) Mobile version

Figure B.4: Desktop and Mobile version of the Lab Test screen

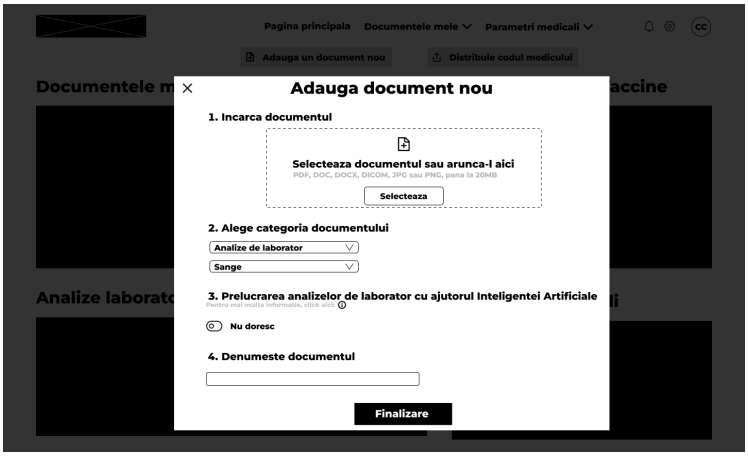


(a) Desktop version



(b) Mobile version

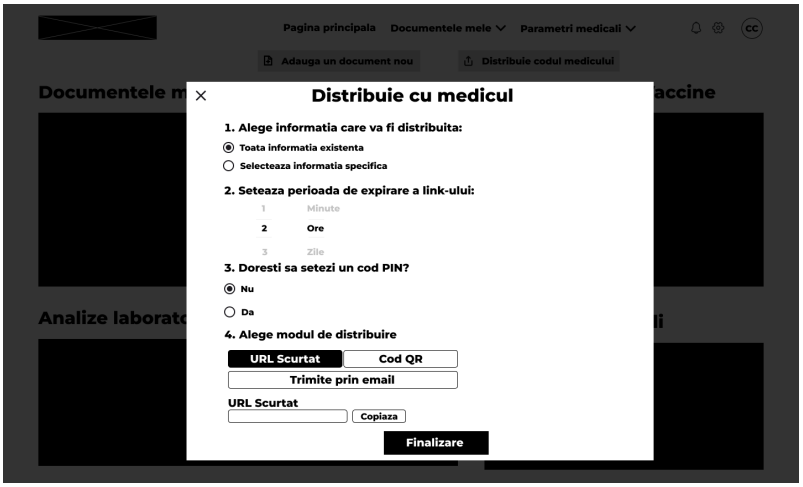
Figure B.5: Desktop and Mobile version of the Medical History screen



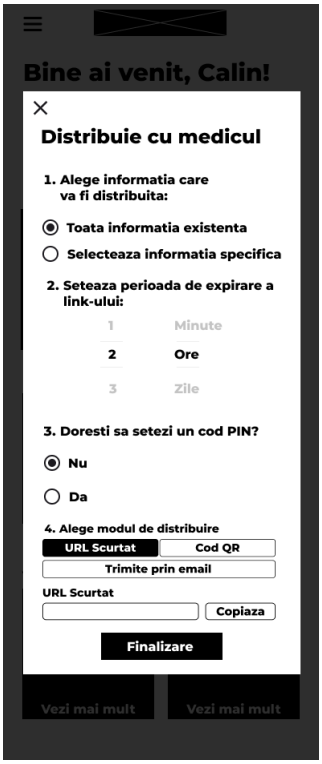
(a) Desktop version

(b) Mobile version

Figure B.6: Desktop and Mobile version of the New Document screen

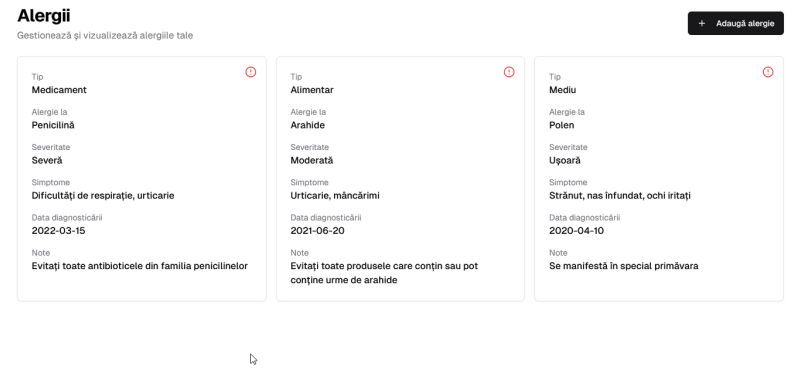


(a) Desktop version



(b) Mobile version

Figure B.7: Desktop and Mobile version of the Share Doctor screen



(a) Desktop version

(b) Mobile version

Figure B.8: Desktop and Mobile version of the Allergies screen

Appendix C

Final Stakeholder Feedback

Full feedback for stakeholder 1:

> 'I appreciated the concept of the project and the way all essential components of a Personal Health Record were integrated: general, clinical, and paraclinical data, consultations, physician access, and data security. The platform is strongly patient-centered, enhancing access to personal medical data and significantly contributing to patient empowerment.
>
> I would emphasize the importance of adhering to interoperability standards, both for future integration with hospital software systems and for enabling the upload of data from mobile devices.
>
> From my perspective as a medical doctor specialized in public health and an expert in digital transformation of healthcare, I see great potential in this project and I am committed to offering the necessary support for its further development'

Full feedback for stakeholder 2:

> 'I was really impressioned with the application on the digital health records book you have developed and presented to me. As a BA you have managed to capture the most important needs and then translated them into useful features included in the solution developed.
>
> Even though it is one the first releases, it looks already complete and covers all the journey of a citizen fascinated with the use of the technology for managing the comprehensive records related to its health status in a very efficient and effective way. The application is user friendly, it looks so intuitive as the user

can easily find any data or add something new to the app. It offers a lot of flexibility in terms of types of data that can be added and monitored. The dashboards and reports generated in the app are easy to read and understand, as well as provide insightful information on the evolution of the status. I also would like to mention the professional approach as you have provided for notifications related to personal data protection, as the applications seem to use very sensitive information.

By developing this solution you have given a new opportunity to Moldova to make one big step forward to the digitisation, and hope it will be largely used by moldovans as the country offers them also other related nice opportunities like high speed internet access and good coverage by 5G network'