

# DEVELOPMENT OF A RADIOLOGY INFORMATION SYSTEM (RIS)

By Biel Simon Rojas

25<sup>th</sup> October, 2021

Director: Carles Camins Giménez

Ponent: Xavier Burgués Illa

Bachelor Degree in Informatics Engineering

Specialization: Software Engineering

UNIVERSITAT POLITÈCNICA DE CATALUNYA (UPC)

Facultat d'Informàtica de Barcelona (FIB)

UNIVERSITAT POLITÈCNICA DE CATALUNYA  
BARCELONATECH

Facultat d'Informàtica de Barcelona



## Table of Contents

List of Tables .....	5
List of Figures .....	5
1 Introduction .....	8
1.1 Goal of the project .....	8
1.2 MedicalOne .....	8
1.3 Justification and state of the art .....	9
1.4 Scope .....	10
1.4.1 Scope of the whole system .....	10
1.4.2 Domain overview .....	10
1.5 Scope of this project .....	12
1.5.1 Goal: What is this project about .....	12
1.5.2 In Scope .....	12
1.5.3 Out of scope .....	12
1.6 Stakeholders .....	14
1.6.1 Patient .....	14
1.6.2 Reception personnel .....	14
1.6.3 Radiology Technologists .....	14
1.6.4 Administrative and accounting managers (invoicing, reviewing processes) .....	14
1.6.5 Executive officers (statistics, business intelligence) .....	14
1.6.6 MedicalOne Staff and Management .....	14
2 Project Management .....	15
2.1 Project planning .....	15
2.1.1 Methodology being used .....	15
2.1.2 Description of tasks .....	15
2.1.3 Estimates and Gantt diagram .....	19
2.1.4 Risk management: Alternative plans and obstacles .....	22
2.1.5 Resources needed .....	23
2.2 Budget .....	24
2.3 Sustainability report .....	26
2.3.1 Economical Area .....	26
2.3.2 Environmental Impact .....	26
2.3.3 Social Area .....	26

3	Requirements and specification .....	27
3.1	Requirements.....	27
3.1.1	Functional requirements.....	27
3.1.2	Non functional requirements.....	28
3.2	Specification.....	28
3.2.1	Logical Domain Model of the complete system .....	28
3.2.2	Specific Logical Domain Model .....	30
4	Software Design .....	33
4.1	Overall architecture .....	33
4.2	Frontend design .....	35
4.2.1	Introduction .....	35
4.2.2	Frontend General Navigation Map .....	35
4.2.3	Features (Main flows) .....	36
4.3	Backend design .....	43
4.3.1	Introduction .....	43
4.3.2	Operations.....	43
5	Implementation .....	45
5.1	Frontend.....	45
5.1.1	Technology used .....	45
5.2	Backend.....	50
5.2.1	Technology used .....	50
5.3	Database .....	53
5.3.1	PostgreSQL .....	53
5.3.2	Schema of entities used by the features in the scope of this project .....	54
5.3.3	Fine tuning .....	55
5.4	Object Storage (S3) .....	56
5.5	Twelve-Factor App .....	57
6	Deployment, Workflow and CI / CD.....	59
6.1	Environments.....	59
6.2	Server architecture .....	60
6.2.1	Nginx (Reverse Proxy) .....	60
6.2.2	PM2 (Process manager) .....	61
6.3	Jenkins (CI / CD) .....	62

7	Legal Aspects.....	63
8	Conclusions .....	64
8.1	Accomplished objectives.....	64
8.2	Technical competences accomplishment in relation to the software engineering specialization. 64	
8.3	Planning and budget review .....	65
8.4	Future work.....	65
9	References .....	67

## List of Tables

Table 1. Tasks and workload estimation.....	19
Table 2. The resources needed for the project with their typology, purpose and tasks it is planned to be used in.....	23
Table 3. Cost per hour of the different roles present in the team. Source: Own compilation, data extracted from salaryexpert.com (3).....	24
Table 4. Economic cost of performing the project tasks based on scenarios 1 and 2.....	24
Table 5. Resources used in the project and their total used price .....	25

## List of Figures

Figure 1. Simple entity-relationship diagram representing the main domain entities. In the inception phase of the project, a lot more entities will likely be defined and will end up in the final product.....	11
Figure 2. Gantt diagram of the project .....	21
Figure 3 Diagram of the Domain Model extracted directly from the documentation of the project within the company .....	29
Figure 4 UML Class Diagram showing the entities and the relationships used to support the main workflow. All entities have id as an autogenerated surrogate key. ....	31
Figure 5 UML Class Diagram showing the additional entities and the relationships used to support the authentication and role-based authorization parts of the system: User, UserRole and Role. All entities have id as an autogenerated surrogate key. ....	32
Figure 6 UML Diagram showing the high-level architecture of the system with its main moving parts. It's worth noting that there are many terminals interacting with the same Cloud Server instance. ....	34
Figure 7 UML State diagram representing the main pages and navigation between them.....	35
Figure 8 Navigation diagram using miniatures of the Mockups to show the global concept. Numbers correspond to states in the UML State Diagram. ....	36
Figure 9 UML Activity Diagram of the Frontend Flow 1: Appointment booking flow using calendar.....	37
Figure 10 Patients Page. From this page the user will be able to browse, create, edit and delete patients. The user can also create new appointments for the patients or view existing ones.....	37
Figure 11 Schedule Page while in the flow of creating new appointments. Numbered blue circles represent, in order, the actions taken to create 3 new appointments for 3 different resources.....	38
Figure 12 Visits Page after creating 3 new appointments using the schedule (as shown in the previous figure).....	38
Figure 13 Visits Page after filling in the rest of the information for the visit lines. At this point the appointments are ready for the day the patient comes.....	39
Figure 14 UML Activity Diagram of the patient arrival flow .....	40
Figure 15 Schedule view when opened from the main menu. Allows visits to be clicked and opens the visits for patient page as shown in Figure 17.....	41
Figure 16 Visits page when opening it on a day in which the patient has appointments when the patient is on the reception desk showcasing the document uploading and classification UI concept. We can see that the selected document is associated to the entire Appointment 1 and only the first visit line of the Appointment 2 .....	41
Figure 17 Visits page when opened for a patient that is ready to enter a visit or has already completed the radiology exams.....	42

Figure 18 UML Sequence diagram illustrating the inner workings of the visitsForPatient operation, prepared to be called from the frontend. ....	43
Figure 19 UML Sequence diagram illustrating the process of attaching many documents. Note that the upload is made directly from the frontend to the directly to the S3 object storage server, reducing in this way the load the Business Logic Server has to take when uploading documents. ....	44
Figure 20 Typescript logo. Typescript adds static typing on top of Javascript and is the language of choice for our project. ....	45
Figure 21 React logo. React is a frontend framework that takes care of the view layer. ....	46
Figure 22 Webpack logo. Webpack is a module bundler for the web. ....	46
Figure 23 Data grid example of usage in the final product. This page corresponds to the Patients Page and shows all patients in the system in a Data Grid. All data presented is anonymized to comply with data protection regulation. ....	47
Figure 24 Example of custom usage of the Data Grid in the final implementation of Visits Page. ....	47
Figure 25 Example usage of the Calendar library found in the Schedule Page. This result was achieved by using FullCalendar as a base and adding some additional features on top such as custom event contents, mouse over tooltips and a time indicator by the side of the cursor. ....	48
Figure 26 Logo of BlueprintJS. Main source of components and inspiration for the design system. ....	49
Figure 27 MobX Logo. MobX is the frontend state management library used to solve the state management needs in this project. ....	49
Figure 28 Diagram showing how MobX extends the language to track dereferences (or property accesses) to build a reactive engine on top of JavaScript. More information in MobX Documentation. Source: MobX Documentation. ....	50
Figure 29 Image explaining how MobX reactivity works. Source: MobX Documentation. ....	50
Figure 30 Logo of NestJS, the framework of choice for this project. ....	50
Figure 31 Logo of TypeORM, the ORM of choice for this project. ....	52
Figure 32 Diagram showing the tables relevant to the part of the project addressed in this document. The entire database schema is much larger and does not fit in a single page. ....	55
Figure 33 MinIO logo. MinIO is a self-hosted S3 storage solution (13). ....	56
Figure 34 Diagram illustrating the three environments, the branches they use as source for their deployments and the databases they are connected to. ....	59
Figure 35 Diagram illustrating how ingress traffic gets delivered to the different exposed services based on the requested subdomain. Note that not all services are directly accessed from the outside. In the diagram @ represents the domain through which the server is exposed to the internet. ....	61
Figure 36 PM2 logo. PM2 is a process manager for NodeJS adequate for production environments. It performs automatic restarts, healthchecks, monitoring, log aggregation and more. ....	62
Figure 37 Jenkins logo. Jenkins is a platform for easily automating tasks. In our case CI/CD deployments and database backups. ....	62
Figure 38 Activity Record page. This page shows the record of actions performed by all users sequentially. It also provides a filter UI to allow administrative staff to perform quick investigations. ....	63

## Abstract

### English

A Radiology Information System or RIS is a unified software system which that aims to manage all the data and support all the required business processes and workflows in a radiology center. This document showcases the process of designing, developing, and implanting a newly created solution. The whole process starting from the prior analysis up to the final result is explained in this document. It is important to note that due to extension and time constraints, a smaller representative scope has been carefully selected in a way that still illustrates the larger project as best as possible.

### Catalan

Un sistema d'informació de radiologia o RIS és un sistema de programari unificat que té com a objectiu gestionar totes les dades i donar suport a tots els processos de negoci i fluxos de treball necessaris en un centre de radiologia. Aquest document mostra el procés de dissenyar, desenvolupar, i implantar una solució de nova creació. En aquest document s'explica tot el procés des de l'anàlisi prèvia fins al resultat final. És important tenir en compte que, a causa de les restriccions d'extensió i de temps, s'ha seleccionat acuradament un abast representatiu més reduït de manera que encara il·lustra el millor possible el projecte.

### Spanish

Un sistema de información de radiología o RIS es un sistema de software unificado que tiene como objetivo gestionar todos los datos y dar soporte a todos los procesos de negocio y flujos de trabajo necesarios en un centro de radiología. Este documento muestra el proceso de diseñar, desarrollar e implantar una solución de nueva creación. En este documento se explica todo el proceso desde el análisis previo hasta la implementación e implantación final. Es importante tener en cuenta que, debido a las restricciones de extensión y de tiempo, se ha seleccionado cuidadosamente un alcance representativo más reducido que aún ilustre lo mejor posible el proyecto.

# 1 Introduction

## 1.1 Goal of the project

The goal of this project is to create a Radiology Information System (RIS) for a client of the MedicalOne software company.

A RIS is a subcategory inside the broader Healthcare Information System or HIS. To summarize, the scope of the system we plan to build includes managing all the data and support all the current workflows in the organization, except for some areas where already existing specialized tools fit better (1).

In the case of this client, a complete solution that was able to handle all aspects of the business was requested. This included resource scheduling, patient electronic health record, invoicing, and medical image handling among many others. The system has to be extensible so that in the future more processes can be automated by adding additional features to the system.

Since the process of building the entire planned system is too large to be covered by this End of Degree project (or TFG), only some aspects of it are going to be covered. The parts that are covered will be contextualized as best as possible in the “Scope of this project” section in a way that the reader can clearly understand where they fit in the global picture. During the entire writing of the document, there will be special attention placed into keeping this global picture in mind so that it is clear at all times why we are doing what we are doing.

This document covers the entire process behind the project. It not only presents the final result, but also walks through each step in the process of making it and explains the why and how behind it.

## 1.2 MedicalOne

MedicalOne is a software company that works with academic and scientific institutions and industry partners to develop new products and access new markets. It was founded in 1989 in Parc Científic de Barcelona (PCB) (2).

Its current offerings include software solutions for Cardiology, Parenteral Nutrition and Radiology - the latter being the area concerning this project.

MedicalOne is currently serving more than 700 corporations, universities, and organizations around the world. A list of the public clients and software solutions can be found on its website<sup>1</sup>.

MedicalOne is currently migrating a lot of software solutions that were developed using older technologies over to web technologies. This project, being one of the first that is developed using web technologies, intends to serve as a basis to this re-engineering effort that should be able to modernize most of the products and offerings the company currently has in older technologies.

---

<sup>1</sup> MedicalOne Website: <https://www.medicalone.com/clients/>



### 1.3 Justification and state of the art

They have an existing information system with existing data that has to be migrated. The current system does not fulfill their needs and they are looking for an upgrade. In particular, the following issues were present in the existing system:

1. **Technical debt:** Due to architectural issues and technical debt, the current codebase cannot be changed or altered in any meaningful way to add new features or fix existing problems
2. **Not fitting their use case:** Is based on a generic solution that does not exactly fit their use case and workflow. This is causing inefficiencies in the daily workflows of the
3. **Security:** Is based on older versions of the PHP stack without a clear upgrade path. Including language and framework versions which is a problem security-wise since those contain known vulnerabilities and bad security practices. The current solution for this problem has been to limit the access to the system to certain fixed IP addresses. But this hinders the usability of the system since it cannot be used from anywhere in the internet.

These points defend the case for the replacement of the system. However the organization could choose one of the existing solutions in the market so satisfy their needs. This is a path that was already explored by the client when the company started with this project. There were some factors that made following this path infeasible:

1. **Connections and integrations with physical equipment:** A hard requirement for the system was that it had to be able to integrate with the existing radiology equipment. This was a strategic decision taken by the management of the medical center in order to optimize processes and increase efficiency and competitiveness. This eliminated a large chunk of existing solutions in the market.
2. **Niche product:** Large medical software providers often omit this market segment in their solutions because it is small when compared to others. It is in this type of products that MedicalOne specializes in.

Taking all this points into consideration, a decision was made to start from scratch and implement a new system that would fulfill the requirements. This would come with the added benefit that the solution is going to be tailor-made for their business and is most likely to succeed in their attempt to increase efficiency and competitiveness in their sector.

## 1.4 Scope

This project consists of only a part of the whole MedicalOne RIS system. This system is planned to be a large system that solves almost every aspect of the daily tasks and data management needs at a given radiology center that operates similarly to the one that ordered the making of this software product.

This client is currently working with an existing software application that they are looking to replace due to some issues they have with it. Those are explained in the “Justification” section along with an explanation of why the decision to start this project was made.

### 1.4.1 Scope of the whole system

The requirements for the entire MedicalOne RIS system are the following. These requirements include all the features present in the old system they are using currently plus new ones that were requested for this new one:

1. **Scheduling:** The ability to manage a schedule for each resource (in this case mostly radiology equipment)
2. **Patients:** The ability to keep track of patients in the system
3. **Appointments:** The ability to keep track of the appointments and actual radiology exams or visits of each patient
4. **Visits:** Be able to support personnel in the workflow of a visit they currently have, improving it where possible and adapting to it otherwise.
5. **Physical equipment:** Connect to the medical equipment and send jobs to the Worklist via PACS
6. **Invoicing:** Issue and manage invoices to the possible clients of the medical center which are patients, insurance companies and the social security.
7. **Tracking referrers:** Another important feature of the system is the ability to track medicals and centers which refer patients to the radiology center and the commercial relationship with them.
8. **Business Intelligence:** The system should have the capability to analyze the data and provide statistics and insights based on those.

### 1.4.2 Domain overview

In order to quickly understand the domain we are working with, we can take a look on the primary entities. Those are **Patient**, **Visit** and **VisitLine**. A patient can have many visits associated to it with every visit having one or more products or services being applied to the patient – a concept represented by the VisitLine entity. It is on each one of this lines that a **Product**, otherwise known as radiology exploration is attached to.

The next most relevant concept is the **Resource**. Every Visit takes place on and needs to allocate time for it in a Resource. In this case, Resources are commonly going to be pieces of radiology equipment such as an MRI scanner. Those are very expensive and need to be used as efficiently as possible.

In the figure, entities belonging to different packages of the system are circled in gray and annotated with the package name. This delimitation is purely for organization and clarity purposes. In the system all those entities are going to be able to interact with each other without any kind of barrier between them.

We have discussed the primary entities and relationships in the system. We are assuming that the role of the rest can be inferred by their self-descriptive names as much as possible. In future sections we are going to go into more detail with the entities that end up being relevant to this project.

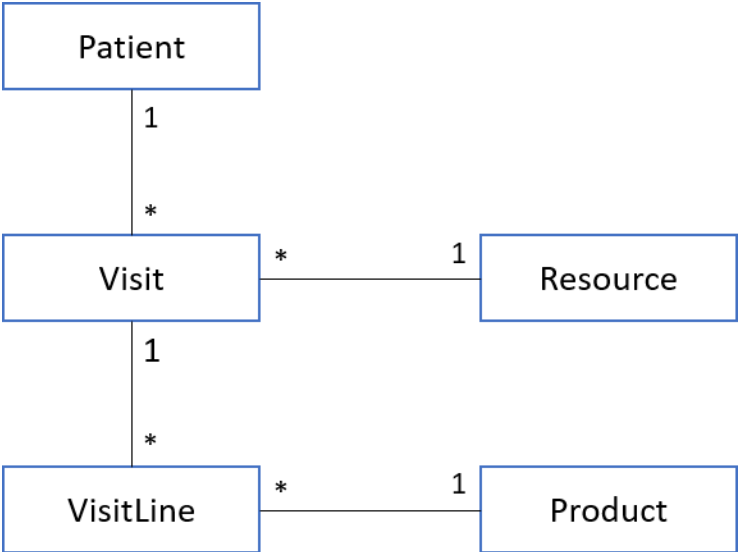


Figure 1. Simple entity-relationship diagram representing the main domain entities. In the inception phase of the project, a lot more entities will likely be defined and will end up in the final product.

## 1.5 Scope of this project

### 1.5.1 Goal: What is this project about

In this project we are going to build the resource planning part of the system. The goal is to provide the ability to use the resources available as efficiently as possible. To do that, the software system must be able to give support to a workflow or mode of operation within the medical center that maximizes the utilization of those resources. It should also be able to perform additional tasks such as automatically calculating the durations of an appointment bases on the services planed in it, keeping track of related information such as the medical who requested the test or the date in which the result is expected to be delivered.

### 1.5.2 In Scope

The scope of this project explicitly includes the following items:

1. **Patient Management:** The system has to be able to keep track of patients. This includes their basic details such as Name, Surnames and contact details along with other information required by the processes that are carried out by the medical facilities using the software.
2. **Appointment Management and Scheduling:** The system has to be able to register and manage appointments for the different radiology exams and procedures in the available resources. The system must look after the best resource utilization possible to increase profitability and decrease patient appointment wait times.
3. Resource Management
4. **Document management:** The system has to be able to keep track of, store and process the documents required for the different radiology exams and procedures. This documents can be Informed Consent, Radiology Imaging Requests, Insurance Company Authorizations, ... among others.
5. **Architected to expand:** All the above elements must be considered in the context of a larger system that will cover other many aspects of the daily operation of the radiology system. Also, the system is expected to grow significantly in features and areas of concern. Therefore, during the design and architecture phase of the system it will be essential to strive for as much changeability as possible and the use of architectures that provide extension points where we can foresee likely future expansions.
6. **Architected to scale:** While there are other priorities that come before raw performance in this project, we must ensure that the project will be able to scale the required amount and that there will not be problems down the line

### 1.5.3 Out of scope

The scope of this project does not include the following items, although some are included in the Radiology System.

1. **Client management:** The management of clients which in this case can be patients, insurance companies or the social security is not included in this project.

2. **Invoicing:** Keeping track and issuing invoices to the different kinds of clients the medical center may have. This is part of the system but is not included in this project.
3. **Medical image processing and storing:** Medical images are handled outside the scope of this project.
4. **Business intelligence:** Statistics reports and insights are outside the scope of this project.
5. **External standardized API:** The building of an API that enables integration with other systems is not included in the scope of this project.

## 1.6 Stakeholders

Main stakeholders of the project:

1. Patients
2. Reception personnel
3. Radiology Technologists
4. Administrative and accounting managers (invoicing, reviewing processes)
5. Executive officers (statistics, business intelligence)

### 1.6.1 Patient

As the main stakeholder of this project, the Patient should be counted as stakeholder because he will be impacted by the software working properly, or not, when they go through the process of getting any kind of radiology test performed on them.

### 1.6.2 Reception personnel

Reception personnel are a big stakeholder of the system. They are the ones that spend most of the time working using the software and any amount of improvement in process efficiency or usability can lead to huge amounts of time savings. The fact that their jobs can be easier can also lead them to become less stressed and happier in their jobs. However, we have to be careful and take their input during the development process of the software in order to ensure the success of the project and avoid any pushbacks from them once the project is deployed and in use.

### 1.6.3 Radiology Technologists

Similarly, Radiology technologists also spend a huge chunk of their time using the RIS, which they use to look up necessary information in order to properly execute the requested tests and submit the results along with their observations. Improvements in this area will most likely lead to increased efficiency and quality of their work.

### 1.6.4 Administrative and accounting managers (invoicing, reviewing processes)

Administrative staff are the users of advanced features of the system. They are the ones that know how the processes work and are very valuable during the development processes. Once the system is finished and implemented it is important to provide them with enough training so that the system is used properly and to make them able to use any advanced features that may exist.

### 1.6.5 Executive officers (statistics, business intelligence)

Although administrative staff does not use the software with as much frequency as the rest of the staff, when they do it is often to make important decisions and supervise important processes. Therefore, a smooth experience should be ensured for them.

### 1.6.6 MedicalOne Staff and Management

Finally, the interest of the components of the MedicalOne company must not be understated. They are the ones who are going to do the development and later maintenance and future expansion. It is important to ensure their voice is heard during the entire lifetime of the project.

## 2 Project Management

### 2.1 Project planning

#### 2.1.1 Methodology being used

The methodology we are going to use is the one which is usually used in projects of this kind at MedicalOne. To describe it briefly, it is a variant of agile or iterative development with monthly deliveries to the client. The product so-far is going usually tested in a real setting as well as assessed in meetings with representatives of all stakeholders.

In most cases, the first iteration corresponds to an Inception iteration, where the product is defined completely in terms of requirements and scope. UI Mockups and detailed ER (Entity Relationship) diagrams are used in order to clearly define the product. The second iteration then corresponds to the building of all the architectural components using the real tools and code. The architecture should take into account all the planned features and stay nearly the same from the end of this iteration until the completion of the project. The next iterations consist of adding all the planned features little-by-little in manageable pieces, producing deliverables which are presented to the stakeholders so they can evaluate and provide feedback in time.

#### 2.1.2 Description of tasks

In this section, the tasks that the project is broken into are presented along with their respective durations.

##### 2.1.2.1 T1 Project Planning

In this first task, the goal is to plan the project with as much detail as possible and elaborate a document containing a concrete and actionable plan that allows the project to be completed successfully and on time.

Total duration: 75 hours

##### 2.1.2.1.1 T1.1 Scope

In this task, an evaluation of the domain is made and written into a document. With this analysis we can envision what the product is going to be. This allows us to later break this vision down into tasks that can be executed to get the final project.

Duration: 20 hours

##### 2.1.2.1.2 T1.2 Planning

When the scope of our project is clear, the resulting tasks are going to be planned, taking into account the duration, dependencies and complexity of the tasks.

Duration: 20 hours

##### 2.1.2.1.3 T1.3 Estimate

Estimation of the budget taking into account all of the tasks and resources needed to do them.

Duration: 25 hours

##### 2.1.2.1.4 T1.4 Improvements

Duration: 7 hours

#### 2.1.2.1.5 T1.5 Final document writing

This task consists of writing the final document.

Duration: 3 hours

#### 2.1.2.2 T2 Inception

The main objective of this task is to determine the functional and non-functional requirements of the system. This includes all the features needed to satisfy the client's needs as well as other requirements that apply to all software projects developed for this industry.

Total duration: 105 hours

#### 2.1.2.2.1 T2.1 System requirements and use case analysis

In this phase consist of analyzing the need of the client and gathering all the requirements that the future software solution should fulfill.

Duration: 35 hours

#### 2.1.2.2.2 T2.2 Logical domain model

This phase consists of extracting the logical domain model that the system is going to work with. It is important to go as much detail as possible in order to find flaws in the design as early as possible in the development process. This model should already be easily translatable into a database model.

Duration: 30 hours

#### 2.1.2.2.3 T2.3 Detailed product sketch

This task consists of generating visual artifacts that describe the entire product. Both from visual and user interface point of view and from behavioral and workflow point of view. At this point, the product should be clearly defined. The scope and expected result of this project should be well known and agreed upon by both the development team and the client.

Duration: 40 hours

#### 2.1.2.3 T3 First Development Iteration

Total duration: 80 hours

#### 2.1.2.3.1 T3.1 Detailed UI Designs

The first step in this iteration is to create detailed sketches or mockup of all the user interface screens and elements that are going to be present in the system. These sketches should contain enough detail that no further decisions regarding placement or visual appearance of UI elements must be made during the rest of the development process. This allows the client to get a feel for the product before any code has been written and any miss understandings can be solved in this early stage of the process.

Duration: 40 hours

#### 2.1.2.3.2 T3.2 Database model

The second step of this iteration is to create a detailed database model. At this point, we already have analyzed the logical domain and know how all the features we expect to build look visually. Basing on



that, we can design a data model containing all the tables and fields with their correct types that are going to be present in the final system.

Duration: 10 hours

#### [2.1.2.3.3 T3.3 Backend code architecture](#)

In this third step of the second iteration we will design the architecture of our back-end. At this point, we are going to make all the big decisions and trade-offs that will impact the project during its entire lifetime.

Duration: 15 hours

#### [2.1.2.3.4 T3.4 Frontend code architecture](#)

In this fourth step we will design the architecture of our frontend. We already know which technologies we are going to use based on similar projects done at the same company. But we will still need to make the big decisions that are needed to create the skeleton and code layout that will be the base of our frontend codebase.

Duration: 15 hours

### [2.1.2.4 T4 Second Development Iteration](#)

Total duration: 35 hours

#### [2.1.2.4.1 T4.1 Basic patient handling](#)

The goal of this task is to develop the feature of creating, updating and viewing registered patients in the system in grid-like UI. It includes both the frontend and backend sides of the development.

Duration: 10 hours

#### [2.1.2.4.2 T4.2 Patients advanced search functionality](#)

In this task we will iterate upon the basic patient handling functionality and include advanced feature that will allow to quickly find any patient in the system that meets certain criteria.

Duration: 15 hours

#### [2.1.2.4.3 T4.3 Additional patient information](#)

In this task we will add the functionality related to Electronic Health Record (EHR) to the existing patient system. The exact information will be already agreed upon by the client at this point.

Duration: 10 hours

### [2.1.2.5 T5 Third Development Iteration](#)

In this third development iteration we will work on aspects related to the appointments and visits of a patient.

Total duration: 73 hours

#### [2.1.2.5.1 T5.1 Basic patient appointments screen](#)

The goal of this task is to create the skeleton of the screen that is going to show all the information about the patient. This includes the basic patient information as well as any appointments and previous visits it may have.

Duration: 15 hours

#### 2.1.2.5.2 T5.2 Appointments of a patient visualizer

In this task we build upon the basic skeleton generated in the previous task and create a viewer that allows the management of all the appointments and visits. We expect this to be the most complex part of the UI in the entire system.

Duration: 40 hours

#### 2.1.2.5.3 T5.3 Appointment process control interface

In this task we plan to add a UI element that allows to keep track of in which stage each individual visit is and for which stages it has already past and while registering the timestamps of all those events.

Duration: 8 hours

#### 2.1.2.5.4 T5.4 Additional UI elements required that may appear in the design

IN this stage we will add additional elements that correspond to minor features designed to support staff at performing daily tasks that also appear in the mockups that we may have at this point.

Duration: 10 hours

#### 2.1.2.6 T6 Fourth Development Iteration

Total duration: 55 hours

##### 2.1.2.6.1 T6.1 Calendar UI

In this task we will develop a feature that is able to present the existing appointments in the system in a calendar UI. We are going to start from an existing calendar system and adapt it to match our requirements.

Duration: 30 hours

##### 2.1.2.6.2 T6.2 Advanced calendar features

This task is a continuation of the previous one and involves building advanced features for the calendar UI. Those include supporting advanced operations like drag and drop rescheduling, resizing to change the duration and dragging and dropping between different compatible resources.

Duration: 25 hours

#### 2.1.2.7 T7 Fifth Iteration

In this fifth and last development iteration we are going to finish the implementation of the system by adding the capability of managing files. In this context, a file refers to all the entities which are not central to the system. Examples of those would be the Resource or the Product entity. This will allow users of the system to maintain and update those over time.

Total duration: 45 hours

##### 2.1.2.7.1 T7.1 Files dashboard

This task consist of building a screen that will allow to access all the editors for the different file entities.

Duration: 5 hours

### 2.1.2.7.2 T7.2 Individual file editing screens

This task involves creating custom editing screens, as defined in the mockups, for each individual file entity. Those screens are then going to be accessed via the screen we created in the previous task.

Duration: 40 hours

### 2.1.2.8 T8 Thesis defense preparation

Total duration: 40 hours

## 2.1.3 Estimates and Gantt diagram

### 2.1.3.1 Tasks and workload estimation

Table 1. Tasks and workload estimation

	<b>Code</b>	<b>Hours</b>	<b>Dependencies</b>
<b>T1 Project Planning</b>	T1		
<b>T1.1 Scope</b>	T1.1	20	
<b>T1.2 Planning</b>	T1.2	20	T1.1
<b>T1.3 Estimate</b>	T1.3	25	T1.2
<b>T1.4 Improvements</b>	T1.4	7	T1.3
<b>T1.5 Final document writing</b>	T1.5	3	T1.4
<b>T2 Inception</b>	T2		T1
<b>T2.1 System requirements and use case analysis</b>	T2.1	35	
<b>T2.2 Logical domain model</b>	T2.2	30	T2.1
<b>T2.3 Detailed product sketch</b>	T2.3	40	T2.1
<b>T3 First Iteration</b>	T3		T2
<b>T3.1 Detailed UI Designs</b>	T3.1	40	
<b>T3.2 Database model</b>	T3.2	10	T3.1
<b>T3.3 Backend code architecture</b>	T3.3	15	T3.1
<b>T3.4 Frontend code architecture</b>	T3.4	15	T3.1
<b>T4 Second Iteration</b>	T4		T3
<b>T4.1 Basic patient handling</b>	T4.1	10	
<b>T4.2 Patients advanced search functionality</b>	T4.2	15	T4.1
<b>T4.3 Additional patient information</b>	T4.3	10	T4.1
<b>T5 Third Iteration</b>	T5		T4
<b>T5.1 Basic patient details screen</b>	T5.1	15	
<b>T5.2 Appointments of a patient visualliser</b>	T5.2	40	T5.1
<b>T5.3 Appointment process control interface</b>	T5.3	8	T5.2
<b>T5.4 Additional UI elemtents required that may appear in the design</b>	T5.4	10	T5.1

<b>T6 Fouth iteration</b>	T6	T5	
<b>T6.1 Calendar UI</b>	T6.1	30	
<b>T6.2 Advanced calendar features</b>	T6.2	25	T6.1
<hr/>			
<b>T7 Fifth Iteration</b>	T7	T6	
<b>T7.1 Files dashboard</b>	T7.1	5	
<b>T7.2 Individual file editing screens</b>	T7.2	40	T7.1
<hr/>			
<b>T8 Thesis defense preparation</b>	T8	40	T7
<hr/>			
<b>Total:</b>		508	

### 2.1.3.2 Gantt diagram

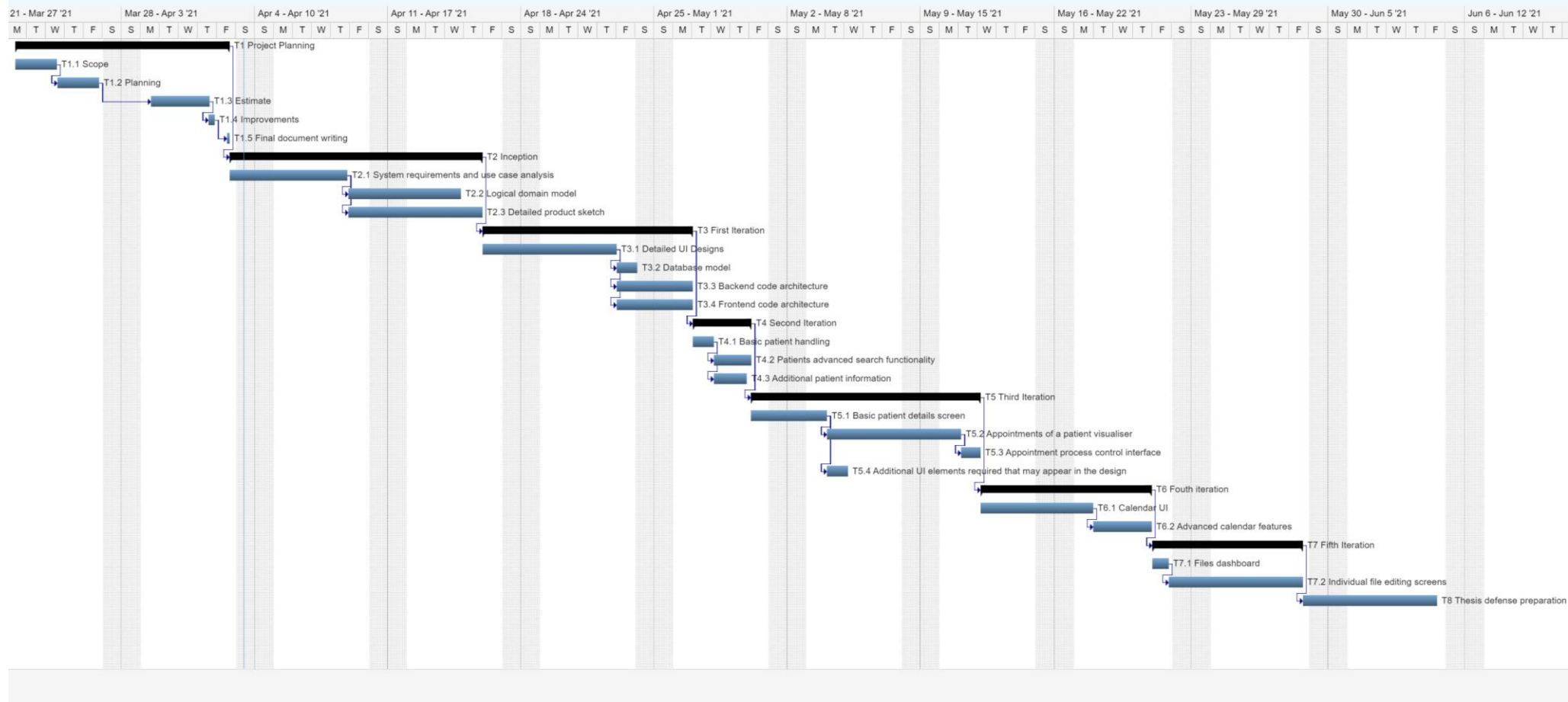


Figure 2. Gantt diagram of the project

#### 2.1.4 Risk management: Alternative plans and obstacles

In this section we go over potential risks we have considered and explain the mitigation plans we have for them.

##### 2.1.4.1 *R1 Deviation in the original planning*

While the aim has been to make the scope and planning of the project as detailed as possible, this is a big project and we have to take into account the risk of it or some parts of it deviating from the original planning in time.

Probability: Medium

##### 2.1.4.2 *R2 Wrong product gets delivered in the end*

There could be the possibility that the product that gets finally delivered to the client does not exactly satisfy its needs. In the planning of the project, we make sure to have the complete specification of the final product as early as possible in the development process. Concretely, we should have it completely defined and validated by iteration 2. In case some deviation from expectations exists, we would correct it in the same iteration.

Probability: Low

##### 2.1.4.3 *R3 Bugs and quality issues*

There exists the possibility of facing bugs and hard to resolve quality issues. This is a risk always present in software projects. However we are avoiding it as much as possible by using proven technology that has been already used in other projects at the same company for building similar products.

Probability: Low

##### 2.1.4.4 *R4 Pushback from certain stakeholders*

There also exists the possibility that some stakeholders may oppose the implementation of the system due to personal or departmental interests. We plan to avoid this by issuing a clear contract and ensure all the stakeholders are on the same page, with special emphasis on the decision makers.

Probability: Low

### 2.1.5 Resources needed

Table 2. The resources needed for the project with their typology, purpose and tasks it is planned to be used in

<b>Code</b>	<b>Resource</b>	<b>Typology</b>	<b>Purpose</b>	<b>Tasks</b>
<b>R1</b>	Dell XPS 17 Laptop	Technological, hardware resource	Used as workstation PC	T1, T2, T3, T4, T5, T6, T7, T8
<b>R2</b>	WebStorm IDE	Technological, software resource	Used as main code editing tool	T2, T3, T4, T5, T6, T7
<b>R3</b>	GitHub	Technological, software resource	Code sharing and VCS (Version control system) tool	T2, T3, T4, T5, T6, T7
<b>R4</b>	PostgreSQL	Technological, software resource	RDBMS (Database engine) used in the project.	T2, T3, T4, T5, T6, T7
<b>R5</b>	NodeJS	Technological, software resource	Backend runtime and development platform	T2, T3, T4, T5, T6, T7
<b>R6</b>	ReactJS	Technological, software resource	Frontend UI library	T2, T3, T4, T5, T6, T7

## 2.2 Budget

In this section an estimate of the cost of the project is presented. In order to quantify the cost of the project, two different alternative scenarios have been considered.

1. **Scenario 1:** The project is developed by a student that earns 9/€ hour which is the standard rate that applies to internships under the UPC regulation.
2. **Scenario 2:** The project is carried out by an experienced team that includes people from all the required roles earning standard industry rates.

In both cases, in order to account for additional communication and other overheads, we added a 30% deviation on top of the results we got after applying hourly rates for each role.

Table 3. Cost per hour of the different roles present in the team. Source: Own compilation, data extracted from salaryexpert.com (3)

Role	Scenario 1 (€/h)	Scenario 2 (€/h)
Project Manager	9	97.5
Senior System Analyst	9	71.5
Software Architect	9	78
Programmer	9	35
UI Designer	9	30

Table 4. Economic cost of performing the project tasks based on scenarios 1 and 2

Task	Role	Hours	Scenario 1	Scenario 2
<b>T1 Project Planning</b>				
T1.1 Scope	PM	20	180,00 €	1.950,00 €
T1.2 Planning	PM	20	180,00 €	1.950,00 €
T1.3 Estimate	PM	25	225,00 €	2.437,50 €
T1.4 Improvements	PM	7	63,00 €	682,50 €
T1.5 Final document writing	PM	3	27,00 €	292,50 €
<b>T2 Inception</b>				
T2.1 System requirements and use case analysis	SSA	35	315,00 €	2.502,50 €
T2.2 Logical domain model	SSA	30	270,00 €	2.145,00 €
T2.3 Detailed product sketch	SSA	40	360,00 €	2.860,00 €
<b>T3 First Iteration</b>				
T3.1 Detailed UI Designs	D	40	360,00 €	1.200,00 €
T3.2 Database model	SA	10	90,00 €	780,00 €
T3.3 Backend code architecture	SA	15	135,00 €	1.170,00 €
T3.4 Frontend code architecture	SA	15	135,00 €	1.170,00 €
<b>T4 Second Iteration</b>				



<b>T4.1 Basic patient handling</b>	P	10	90,00 €	350,00 €
<b>T4.2 Patients advanced search functionality</b>	P	15	135,00 €	525,00 €
<b>T4.3 Additional patient information</b>	P	10	90,00 €	350,00 €
<b>T5 Third Iteration</b>				
<b>T5.1 Basic patient details screen</b>	P	15	135,00 €	525,00 €
<b>T5.2 Appointments of a patient visualizer</b>	P	40	360,00 €	1.400,00 €
<b>T5.3 Appointment process control interface</b>	P	8	72,00 €	280,00 €
<b>T5.4 Additional UI elements required that may appear in the design</b>	P	10	90,00 €	350,00 €
<b>T6 Fourth iteration</b>				
<b>T6.1 Calendar UI</b>	P	30	270,00 €	1.050,00 €
<b>T6.2 Advanced calendar features</b>	P	25	225,00 €	875,00 €
<b>T7 Fifth Iteration</b>				
<b>T7.1 Files dashboard</b>	P	5	45,00 €	175,00 €
<b>T7.2 Individual file editing screens</b>	P	40	360,00 €	1.400,00 €
<b>T8 Thesis defense preparation</b>	-	40	0,00 €	0,00 €
<b>Total</b>		508	4.212,00 €	26.420,00 €
<b>Total (30% deviation)</b>		609,6	<b>5.475,60 €</b>	<b>34.346,00 €</b>

Economic cost associated with the used resources:

Table 5. Resources used in the project and their total used price

Code	Resource	Price	Used %	Used Price
<b>R1</b>	Dell XPS 17 Laptop	2849 €, lifespan 8 years	5,21%	148,69 €
<b>R2</b>	WebStorm IDE	400 €/year	41 %	164,00 €
<b>R3</b>	GitHub	Free	-	-
<b>R4</b>	PostgreSQL	Free	-	-
<b>R5</b>	NodeJS	Free	-	-
<b>R6</b>	ReactJS	Free	-	-
			<b>Total:</b>	<b>312,69 €</b>

Total estimated cost of the project:

1. **Scenario 1:** 5.475,60 + 312,69 = **5.788,29 €**
2. **Scenario 2:** 34.346,00 + 312,69 = **34.658,69 €**

## 2.3 Sustainability report

### 2.3.1 Economical Area

#### 2.3.1.1 PPP

In terms of building the project, we know at this point that the project is viable economically since there is a client that is willing to pay for it. That is at least if everything goes to plan. If some big deviation in the budget occurred for some reason, the project could then become economically inviable. To mitigate that we made an analysis of the different risks that could affect the budget and planned mitigation plans for all of them.

#### 2.3.1.2 Useful Lifespan

The project should be able to run viably for an indefinite amount of time once completed. However, the costs for running it are going to be non-negligible. Costs for Backups/Recovery, Security, Support, Version Updates, Bug Fixes, Feature Additions, Ongoing Training Needs, Scaling, Staffing and Labor and more have to be considered. Taking all of these into account we should be able to calculate an estimate TCO (Total Cost of Ownership) and ensure that we are building a product that we can afford to maintain. (4)

### 2.3.2 Environmental Impact

In the case of this project, the environmental impact in terms of energy consumed is expected to be reasonable. The sources of energy consumption and thus CO2 generation to consider would be:

1. During the **development** process: Workstations and office equipment
2. During the **lifespan** of the project: Cloud servers and terminals to access

Most of the energy consumed over the lifespan of the project is going to come from the servers it's being deployed on, and its efficiency depends in part on the technology used by the cloud provider or infrastructure layer that runs on it (5). While datacenters consume a significant amount of energy, the energy consumed has stayed mostly flat in the last years. This is due to a simultaneous advancements on the energy efficiency front (6).

### 2.3.3 Social Area

Since this project aims to increase efficiency in a medical center, it can indirectly make the services offered there more accessible to the general public. On the other hand, since with this product we plan to automate tasks that were not already automated before, it could lead to job losses for some of the workers in the medical center. While this has to be considered, and is a real social threat if not managed properly, we believe that the benefits of increasing the efficiency of the service outweigh this social drawbacks and can increase the overall quality of life (7).

Whether there is an actual need for this product or not is discussed earlier in this documents, where various reasons are given to justify its existence.

In terms of personal growth, it is the first time I participate in a project of this dimensions and I am learning a lot not just about software development but also about communication skills, interpersonal relationships and time management skills.

## 3 Requirements and specification

### 3.1 Requirements

Although the larger system is intended to be very big and broad in scope, not everything which is in the scope of the RIS System is in the scope of this project. The focus of this project is only on the creation of the resource scheduling part of the system.

#### 3.1.1 Functional requirements

1. Patient CRUD
  - a. Be able to create, modify and delete registered patients in the system
2. Appointment scheduling in Calendar UI
  - a. Be able to schedule new appointments
  - b. Being able to show all the appointments and visits in a calendar UI so the staff can visually manage the schedules.
  - c. Be able to move appointments between similar resources
  - d. Perform time calculations: Taking into account the services offered to the patient, calculate how long the visits will take.
  - e. Be able to define resource downtime and other events which can be recurring
3. Appointment and visit details management
  - a. Information: Registering required data for the procedures such as the resource, the tests that will be performed, the doctor who requested the test, the delivery date and additional comments.
  - b. Stages: Keep track of the stage of an appointment or visit within the workflow. Staff will advance the stages as the patient progresses in the workflow
  - c. Services: Keep track of services provided to a patient within a visit.
  - d. Billing information: Prepare Visit data to be passed to invoicing later
4. Document handling
  - a. Document storage: The system should be able to store documents and make them available for download in a later time.
  - b. Document attachment: Those documents should be able to be attached to Patients, Visits and VisitLines indistinctively. It should then be possible to query this information in a relational manner.
5. Manage files (entities that are not objects of the workflow, they only participate)
  - a. Products: Their names, identification codes and times when performed in a certain resource

- b. Resources: The resources available, with each one having a schedule.
- c. Referring medicals: The list of known medicals that refer patients to the radiology center.

### 3.1.2 Non functional requirements

1. Security: Since we are working with medical information which is classified as personal, the security of the system is a factor to keep in mind on all of the decisions made during the development.
2. Reliability: This system is going to be at the heart of the organization. In case of it going down, all workers would likely have no other option than to stop working leading to a huge financial loss. We have to take all the possible preventive measures against this scenario.
3. Performance: This system is going to be used intensively during all the processes. It would not be acceptable to slow the workers on the medical center as a result of the system itself being slow.
4. Scalability: We have to architect and dimension the system to be able to handle the expected load if not more to avoid downtime as a result of not being able to keep up with simultaneous requests.
5. Accessibility: Ease of use and accessibility is of major importance in a system that will be used by users of all ages and technical backgrounds.
6. Visually pleasing: In order to drive adoption of this new platforms, it has to be satisfying and pleasing to use by the end users that are going to use it every day. Making it visually appealing is a good way to support achieving that provided that all other non-functional requirements are being fulfilled.

## 3.2 Specification

### 3.2.1 Logical Domain Model of the complete system

The complete system covers many aspects of the enterprise information system, including some that are not in the scope of this project. However, we are going to present the entire domain model to give context for the reduced model that we are going to work within the scope of this project – which we are going to present in the next section.

In Figure 3, which is a diagram extracted directly from the documentation of the project within the company, we can see a hand-drawn entity-relationship diagram where only the names of the entities are shown. There are also other visual elements that aim to help in making the domain easier to understand to someone that encounters it for the first time.

Concepts related to each other are grouped in packages, which have a name and are outlined in gray.

There is also information encoded in the color of the boxes that represent each concept that hint at possible design decisions in later in the software design process.

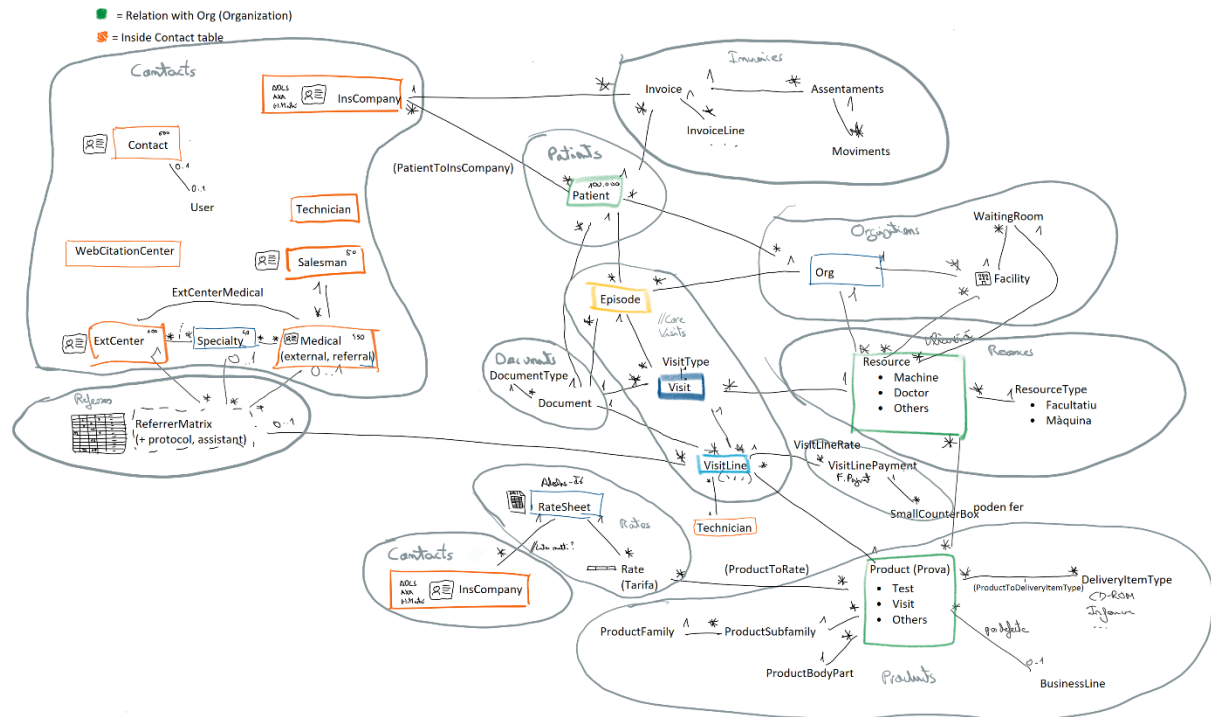


Figure 3 Diagram of the Domain Model extracted directly from the documentation of the project within the company

### 3.2.2 Specific Logical Domain Model

In this section we present the part of the domain model that is directly involved in supporting the features in the scope of this project. The concepts presented here must still be considered in the context of the global system.

#### 3.2.2.1 Main workflow entities

In Figure 4 a diagram showing the entities that support the main workflow is presented. Those are the entities we are going to work with for the rest of the project.

All entities shown in the diagram have id as an autogenerated surrogate key. The reason behind this decision as opposed to using natural keys where applicable was a decision made by the team based on experience on previous similar applications.

Overview of those entities, their purpose and attributes:

- **Patient:** Holds the information required by all processes about a patient.
- **Visit:** Represents the concept of a patient reserving and using a Resource on a certain date to get one or more procedures (Products) done. Holds information in the form of date attributes that allow to track the stages in the workflow and keep a record of events to use later in the data analysis part of the system.
- **VisitLine:** Represents one radiology procedure done on a Patient within a certain Visit (Which represents an appointment to a Resource).
- **Product:** Items or services provided by the organization. Usually used to represent radiology procedures.
- **Resource:** Represents everything that has a schedule of its own. Mostly used to represent radiology equipment, but it is expected to be used for doctor appointments also.
- **Document:** Entity in the relational database that represents documents stored in an external file server.

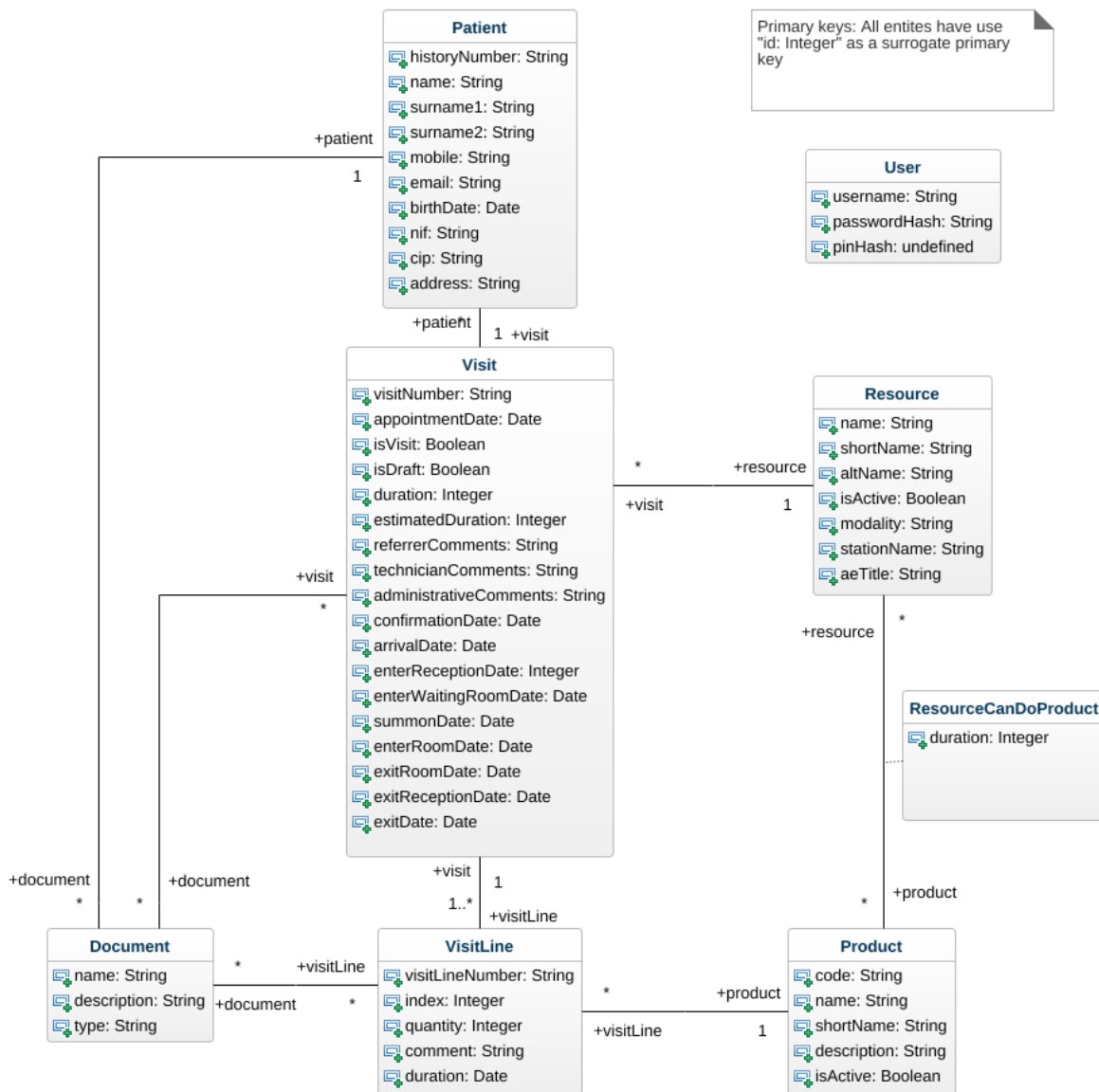


Figure 4 UML Class Diagram showing the entities and the relationships used to support the main workflow. All entities have id as an autogenerated surrogate key.

### 3.2.2.2 Authentication and user handling entities

In Figure 5 we can see how three additional entities are related to the ones explained in the section above.

- **User**: Represents a human user of the system. Stores the information required to perform authentication using different mechanisms such as username and password and pin number.
- **Role**: Represents a role in the system that can be assigned to a User. Instances of this class are predefined and do not change in runtime.

- **UserRole**: Represents a role taken by a user. Stores the fact that certain user has a role and includes additional information such as whether it is read only or the level of permission. Those extra attributes are used to simplify the role management as well as allowing for future expansion of the role system if required in the future.

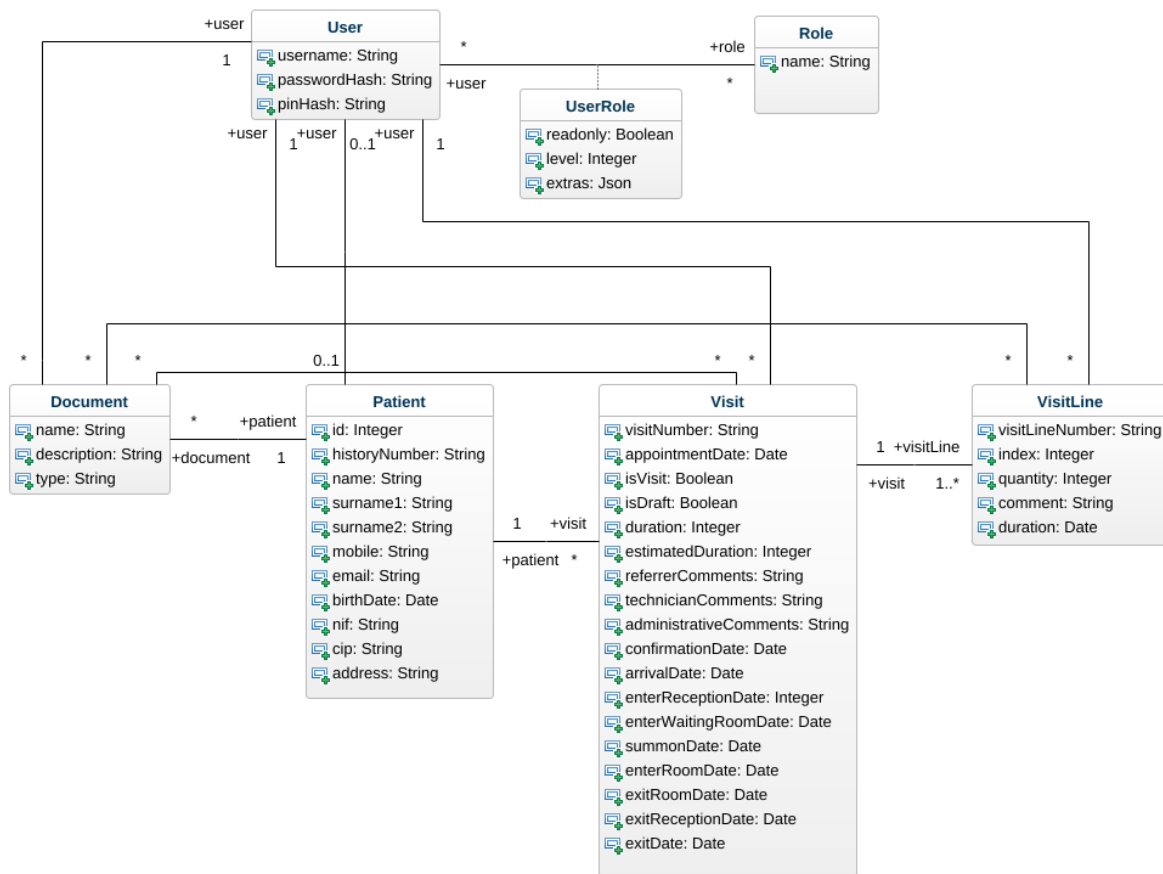


Figure 5 UML Class Diagram showing the additional entities and the relationships used to support the authentication and role-based authorization parts of the system: **User**, **UserRole** and **Role**. All entities have `id` as an autogenerated surrogate key.



## 4 Software Design

### 4.1 Overall architecture

The main aspect of the application design is a client - server architecture. We expect the client to be implemented using web technologies and distributed via a web application server. In Figure 6 the main components composing the system are shown:

- **Cloud Server:** The cloud server instance where the server-side software will run.
  - **Web App Server:** Delivers the web application to the terminals where it's going to be used. Using this approach has many advantages over the traditional desktop software distribution model. Clients are always up to date and updates are preformed at the same time for everyone.
  - **Business Logic Server (or Backend):** Responsible for providing the data and server-side functionality for the application. Its main job is to provide and manage the information the application works with. It is also a secure and trusted environment that is used to perform tasks where this safety is essential such as Authentication and Authorization.
  - **Database:** An RDBMS that stores relational data used by the system in a structured way so the access to it is fast and efficient.
  - **S3 Object Storage:** Stores any kind of documents and files that may be used by the system that would be too big to store and manage with the rest of the data efficiently.
  - **Other services:** In the future, other services consisting of existing software or of own implementation can be added if appropriate.
- **Client Terminal:** The devices where the user client-side software runs.
  - **Web App (or Frontend):** Represents the code that runs on each terminal using the system. It has logic of its own where applicable and delegates processing to the Business Logic Server when appropriate.
  - **Platform controller:** Additional piece of software that is used to implement features that need access to features that are not provided by the browser. Examples include interfacing with radiology equipment or direct printing using platform drivers.

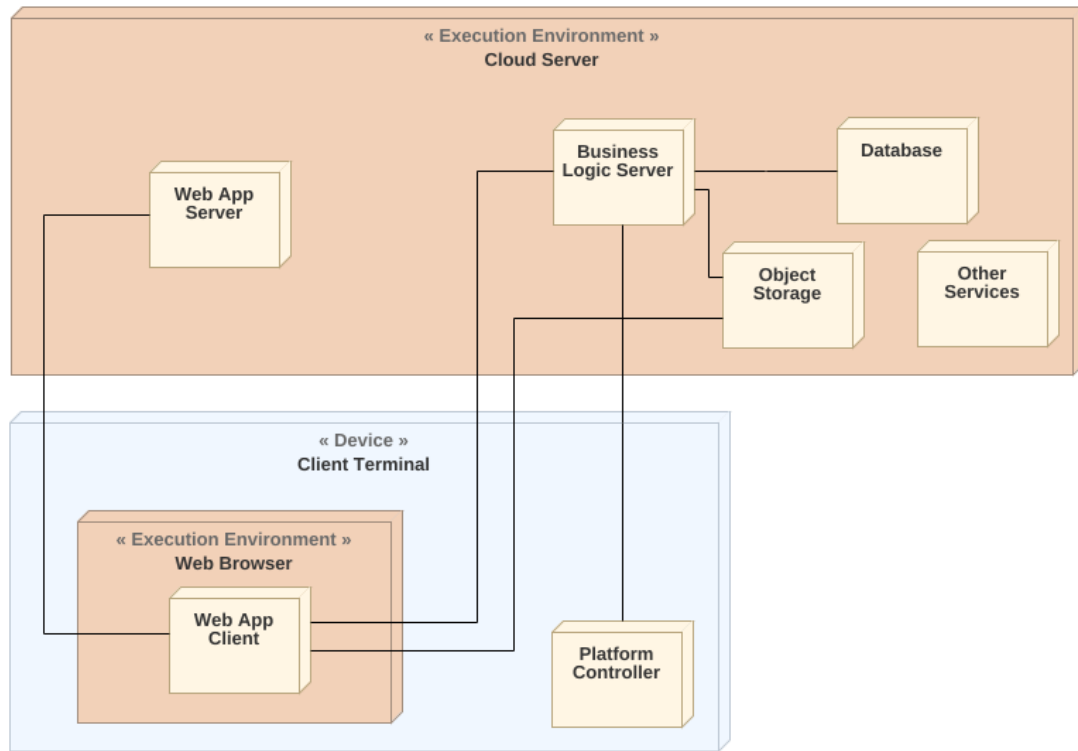


Figure 6 UML Diagram showing the high-level architecture of the system with its main moving parts. It's worth noting that there are many terminals interacting with the same Cloud Server instance.

## 4.2 Frontend design

### 4.2.1 Introduction

We are placing the frontend section first because we followed a frontend-first approach for the development of the system. This means that once the logical data model was clear, we proceeded to decide on the expected behavior of every feature from the point of view of what's going to be shown on the screen. That implied that the decisions and specific requirements of the backend were strongly tied to the frontend.

The main pattern followed to implement the frontend is what's called a Single-Page Application or SPA. In an SPA, the application is loaded from the web server only once at the beginning. As the user interacts with the application and new data is required, it is requested to the respective Business Logic Server as needed. This contrasts with traditional web applications where a new page is fetched on every state transition.

In this section we present the navigation flow of this frontend and the design for the features that handle the two main flows supported by the application. We also present mockups of the application for every state in the state diagram.

### 4.2.2 Frontend General Navigation Map

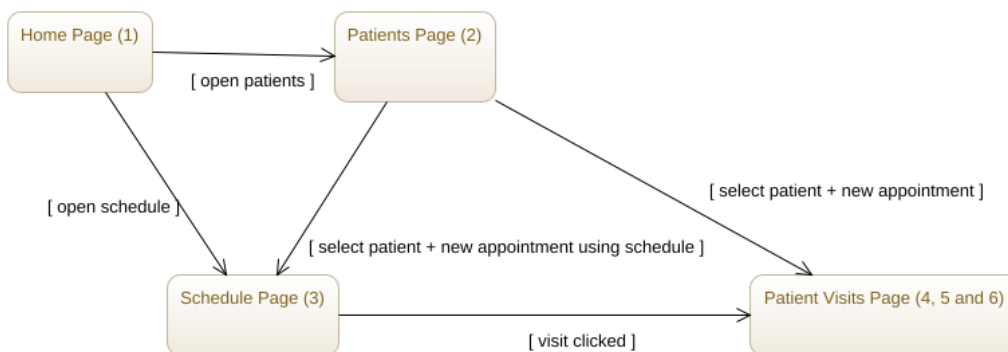


Figure 7 UML State diagram representing the main pages and navigation between them.

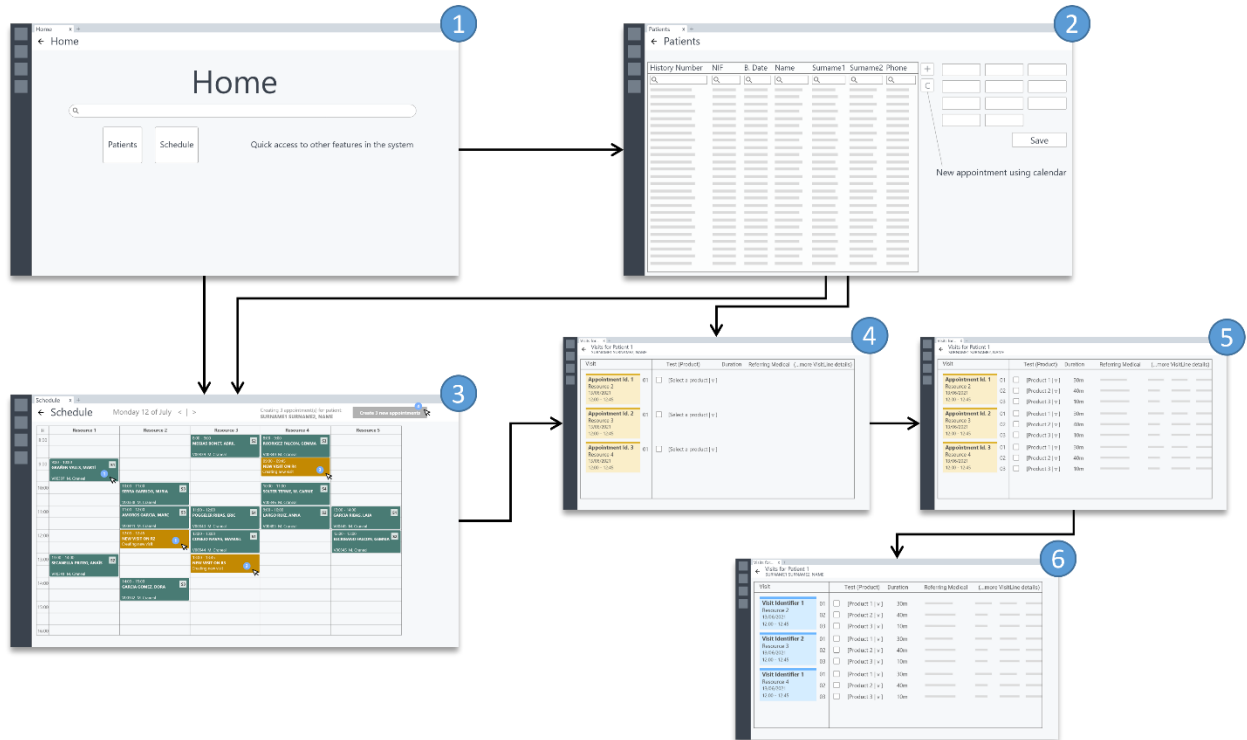


Figure 8 Navigation diagram using miniatures of the Mockups to show the global concept. Numbers correspond to states in the UML State Diagram.

#### 4.2.3 Features (Main flows)

We will describe in depth the features that support the main flows of the app.

##### 4.2.3.1 Frontend Flow 1: Book one or more appointments using calendar

One of the main workflows that the application must support is the booking of new appointments by receptionists. The requesting patients could either be calling in from the outside or be in front of the reception desk.

The flow, as shown in the UML state diagram in Figure 9, starts by searching the patient. The application must allow to find patients by any field to facilitate finding the patient in edge cases and minimize patient duplication. In case the patient is not found, it is created. Once the patient is ready and selected, the receptionist proceeds to indicate that he or she wants to book an appointment for this patient using the schedule. It is in this schedule where the receptionist selects free slots in the requested resources. Then, the receptionist can proceed to selecting the products requested for each resource. Followingly, additional information for each visit line can be added, completing in this way the process of booking the requested appointments for the patient.

Mockups for the different stages of the processes along with their explanation can be found in Figure 10, Figure 11, Figure 12 and Figure 13.



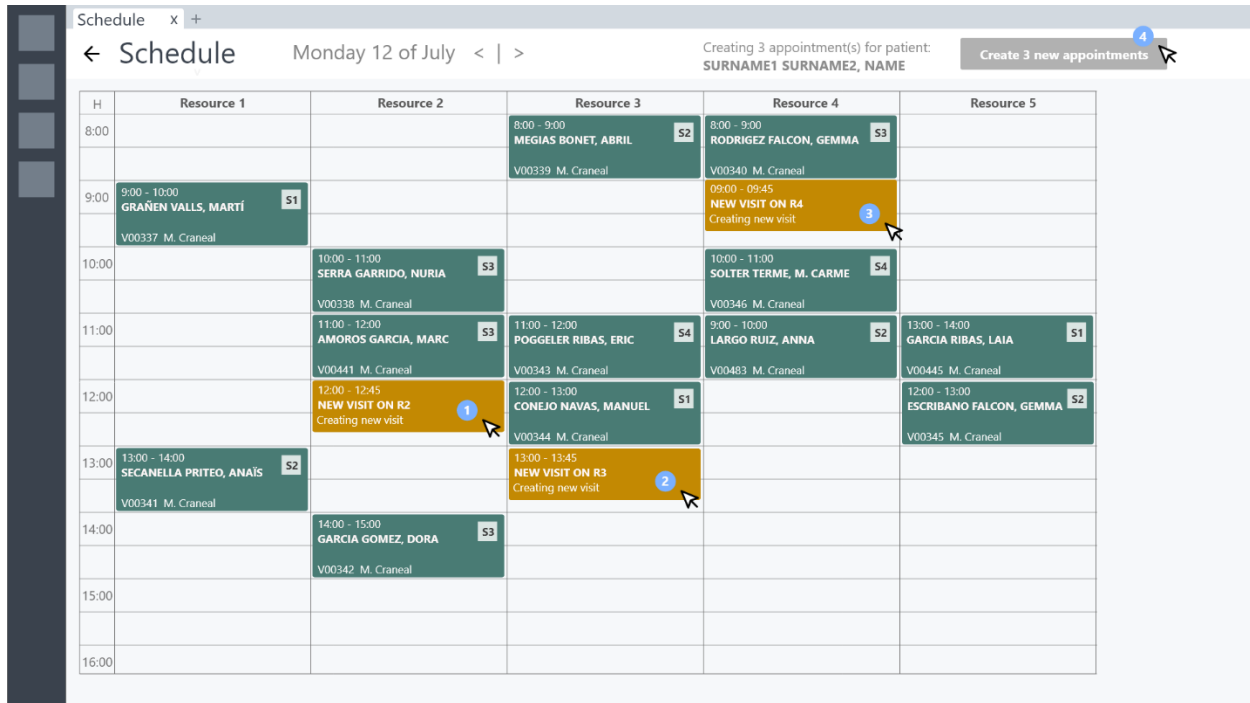


Figure 11 Schedule Page while in the flow of creating new appointments. Numbered blue circles represent, in order, the actions taken to create 3 new appointments for 3 different resources.

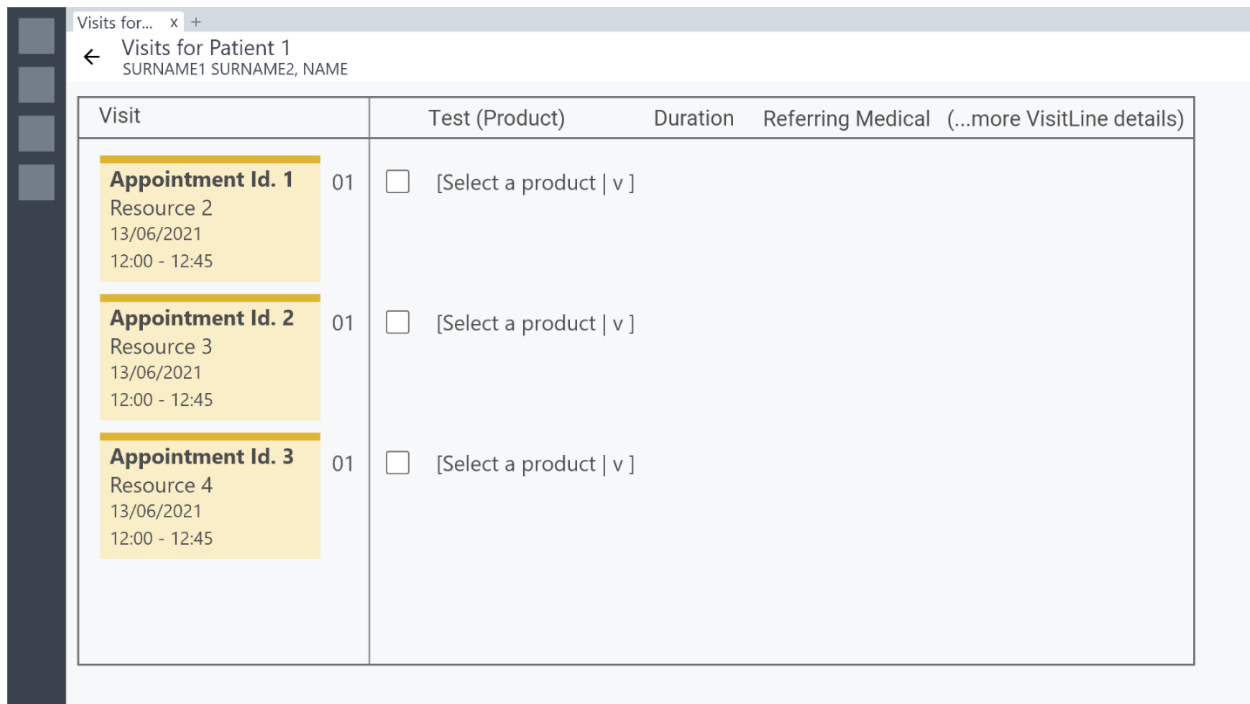


Figure 12 Visits Page after creating 3 new appointments using the schedule (as shown in the previous figure).

Visit	Test (Product)	Duration	Referring Medical	(...more VisitLine details)
<b>Appointment Id. 1</b>				
01 <input type="checkbox"/>	[Product 1   v]	30m	_____	_____
Resource 2				
02 <input type="checkbox"/>	[Product 2   v]	40m	_____	_____
13/06/2021				
03 <input type="checkbox"/>	[Product 3   v]	10m	_____	_____
12:00 - 12:45				
<b>Appointment Id. 2</b>				
01 <input type="checkbox"/>	[Product 1   v]	30m	_____	_____
Resource 3				
02 <input type="checkbox"/>	[Product 2   v]	40m	_____	_____
13/06/2021				
03 <input type="checkbox"/>	[Product 3   v]	10m	_____	_____
12:00 - 12:45				
<b>Appointment Id. 3</b>				
01 <input type="checkbox"/>	[Product 1   v]	30m	_____	_____
Resource 4				
02 <input type="checkbox"/>	[Product 2   v]	40m	_____	_____
13/06/2021				
03 <input type="checkbox"/>	[Product 3   v]	10m	_____	_____
12:00 - 12:45				

Figure 13 Visits Page after filling in the rest of the information for the visit lines. At this point the appointments are ready for the day the patient comes.

#### 4.2.3.2 Frontend Flow 2: Patient arrives for the appointment

The next most important workflow is the handling of the patient arrival at the appointment. The patient arrives on a certain date and time with the intention of taking some radiology imaging procedures because they have requested and successfully booked and appointment previously.

The flow, as shown in the UML state diagram in Figure 14, starts by the patient entering the radiology center. It is then attended by a receptionist which has the software running on their computer. At this point, the goal of the receptionist is to find the appointment for the patient. It can do so either by using the patients page and searching the patient by its name or other identifier, or by using the schedule page in the today view where it is easy to find the appointment visually after the patient provides its name. Once found, the receptionist can see the appointments the patient has for the day.

The receptionist must the go over each appointment and request, scan and upload all the required documentation and complete any missing information for said appointment. The UI concept for this stage can be seen in Figure 16. When an appointment is validated and has its information completed, it gets converted into a Visit. Visits are represented by the blue color, as opposed to appointments which are represented by the orange color. The identifier remains the same when Appointments get converted into Visits.

Finally, when all the documentation has been uploaded to the system the patient is ready to take the procedures and he is told to wait for his turn in the waiting room. Other procedures that are not in the scope of this project continue the flow from the waiting room up to completion.

Mockups for the different stages of the processes along with their explanation can be found in Figure 15, Figure 16 and Figure 17.

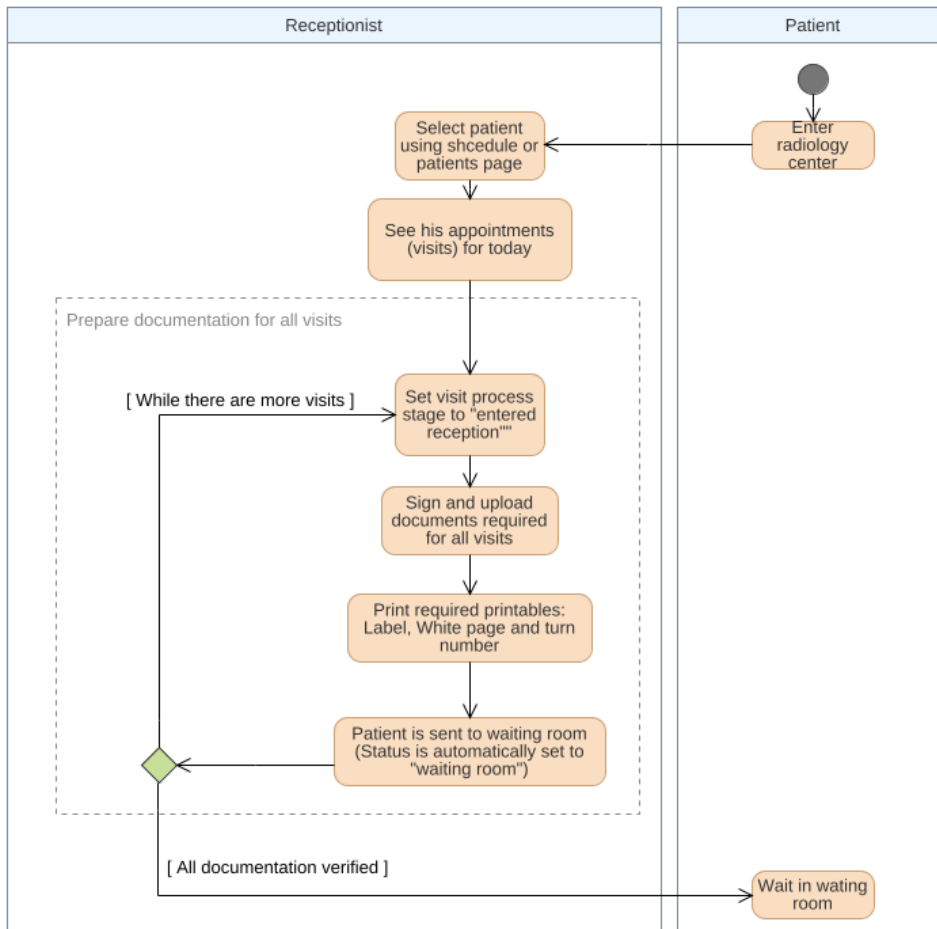


Figure 14 UML Activity Diagram of the patient arrival flow



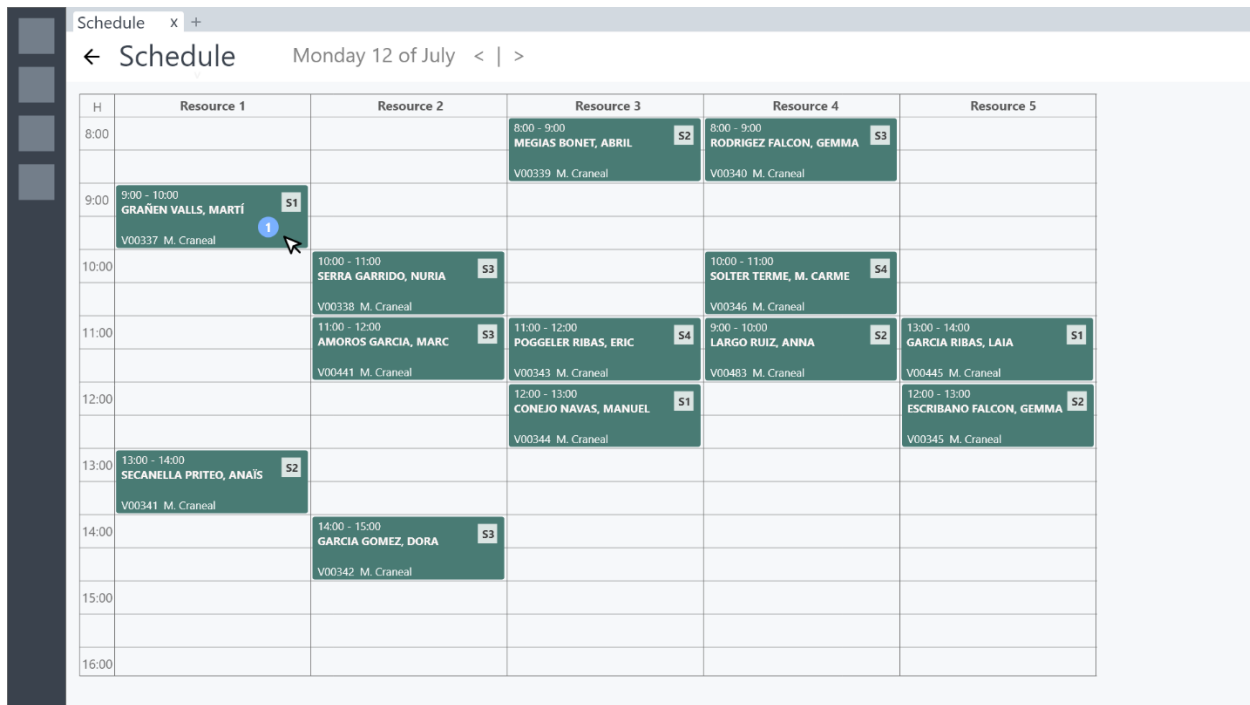


Figure 15 Schedule view when opened from the main menu. Allows visits to be clicked and opens the visits for patient page as shown in Figure 17

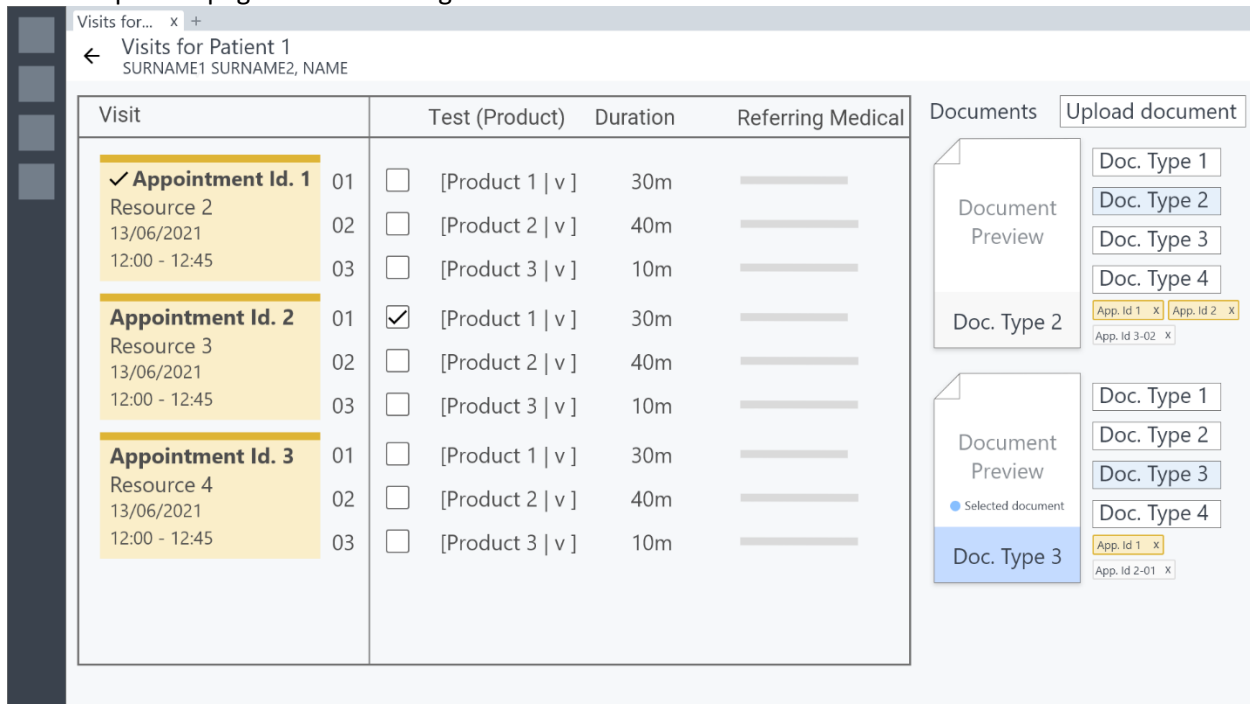


Figure 16 Visits page when opening it on a day in which the patient has appointments when the patient is on the reception desk showcasing the document uploading and classification UI concept. We can see that the selected document is associated to the entire Appointment 1 and only the first visit line of the Appointment 2

Visits for... x +

← Visits for Patient 1  
SURNAME1 SURNAME2, NAME

Visit		Test (Product)	Duration	Referring Medical	(...more VisitLine details)			
<b>Visit Identifier 1</b> Resource 2 13/06/2021 12:00 - 12:45	01	<input type="checkbox"/> [Product 1   v ]	30m	_____	_____	_____	_____	
	02	<input type="checkbox"/> [Product 2   v ]	40m	_____	_____	_____	_____	
	03	<input type="checkbox"/> [Product 3   v ]	10m	_____	_____	_____	_____	
<b>Visit Identifier 2</b> Resource 3 13/06/2021 12:00 - 12:45	01	<input type="checkbox"/> [Product 1   v ]	30m	_____	_____	_____	_____	
	02	<input type="checkbox"/> [Product 2   v ]	40m	_____	_____	_____	_____	
	03	<input type="checkbox"/> [Product 3   v ]	10m	_____	_____	_____	_____	
<b>Visit Identifier 3</b> Resource 4 13/06/2021 12:00 - 12:45	01	<input type="checkbox"/> [Product 1   v ]	30m	_____	_____	_____	_____	
	02	<input type="checkbox"/> [Product 2   v ]	40m	_____	_____	_____	_____	
	03	<input type="checkbox"/> [Product 3   v ]	10m	_____	_____	_____	_____	

Figure 17 Visits page when opened for a patient that is ready to enter a visit or has already completed the radiology exams.

## 4.3 Backend design

### 4.3.1 Introduction

The backend is the part of the application that runs on the server. It receives requests from the clients and contains the necessary logic to send the appropriate response with data back to the client. The backend also contains database, which will persistently store all the data for the application, and other supporting services. This section focuses on the code and software configuration on the server-side that makes this possible.

This section presents an overview of the system and describes some of the operations implemented within the main part of the backend which is the Business Logic Server. The operations are defined using UML diagrams in order to illustrate the design principles used in all the similar operations in the project. Note that there are a lot more operations defined in the real system than those described in this document.

### 4.3.2 Operations

#### 4.3.2.1 Operation 1: Get visits for patient

The first operation we describe is the one of getting all visits for a patient. We expect this operation to be called, at least, from the part of frontend that implements the design concept found in Figure 17, which we called Visits Page.

This operation allows the frontend to have all the Visits and VisitLines for a certain Patient which match a certain filter. This filter is illustrated in the diagram as a historySince: Date parameter, but in reality more parameters are allowed to enable more complex filters. It is also important to note that the response comes in denormalized form, and it is the job of the frontend to rearrange it in the form it best suits its needs. This increases reusability of this operation to be used from different parts of the frontend or even for any access future services may have via a public API.

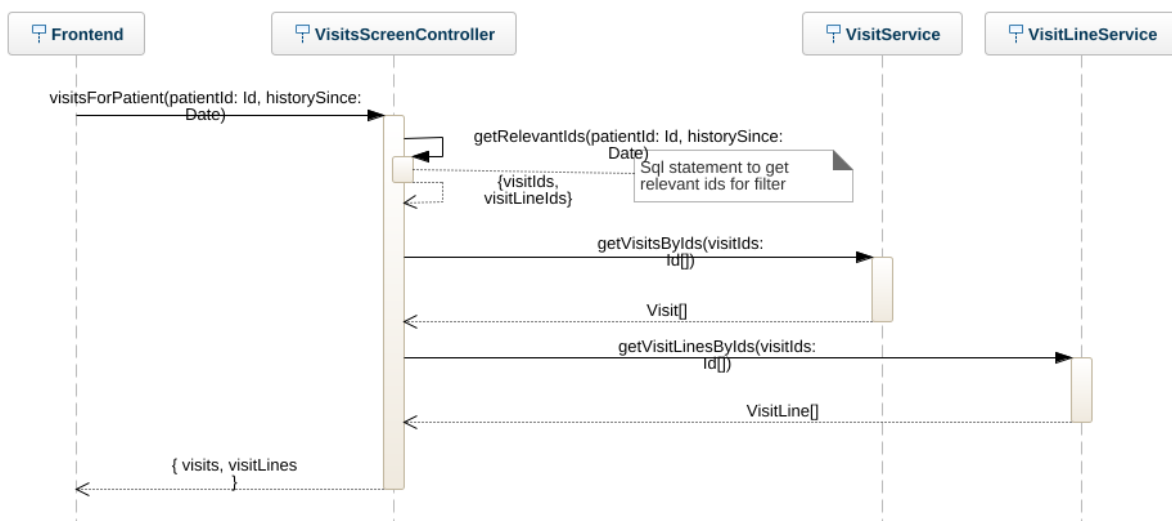


Figure 18 UML Sequence diagram illustrating the inner workings of the `visitsForPatient` operation, prepared to be called from the frontend.

#### 4.3.2.2 Operation 2: Attach many documents

The next operation we describe is the process of uploading a document from the frontend. A diagram for this operation can be found in Figure 19. We expect this operation to be called from the Visits Page detailed previously in frontend design in Figure 16. It is important to note that the S3 Object Storage is not a component we design or implement but one that is already implemented for us and highly specialized in the task of handling large amounts of documents. There are many vendors and alternatives for this piece of the system which we are going to discuss in the implementation section.

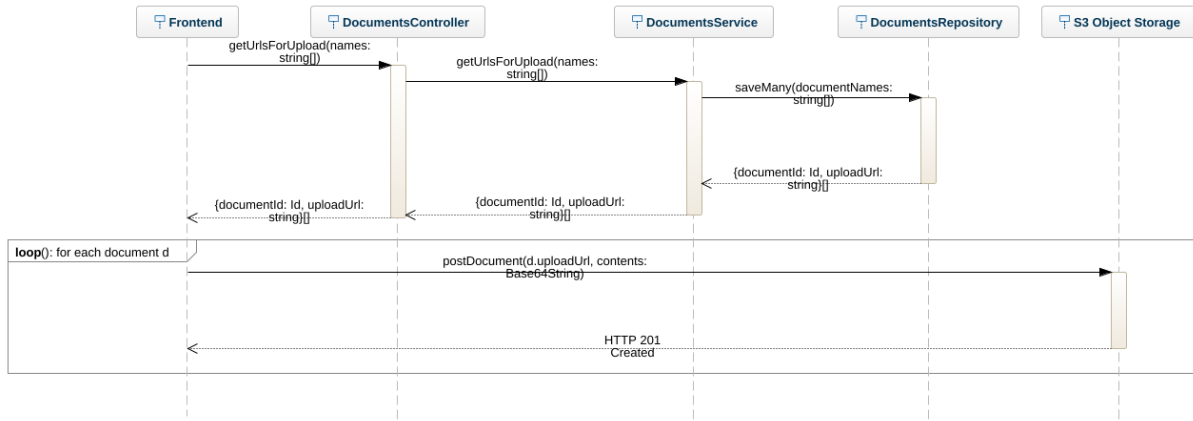


Figure 19 UML Sequence diagram illustrating the process of attaching many documents. Note that the upload is made directly from the frontend to the directly to the S3 object storage server, reducing in this way the load the Business Logic Server has to take when uploading documents.

## 5 Implementation

In this section we are going to present how the software design was finally implemented. Which tools were used and how they were used to get to the final solution.

Most of the pieces of code of own creation have been developed using TypeScript (8). TypeScript is a language that compiles (traspiles) to and is a superset of JavaScript. It essentially adds static typing to the JavaScript language. We can use it in the browser because it is the native language of the web and on the server by using the NodeJS runtime.



*Figure 20 Typescript logo. Typescript adds static typing on top of Javascript and is the language of choice for our project.*

### 5.1 Frontend

#### 5.1.1 Technology used

##### 5.1.1.1 *Web technologies*

There are many possible approaches using different technologies for implementing an enterprise software solution such as this one. However, we chose the web platform for various reasons.

Web technologies are increasingly becoming the industry standard for enterprise applications. They allow for easy distribution, are always up to date and are natively multiplatform. Using almost any other technology would have required to develop multiple versions of the same frontend with code reuse between them being minimal or non-existent.

Using Web Technologies we get a mobile version of the system by default and only requires special attention to make the required use cases more usable in small and touch-based devices.

##### 5.1.1.2 *SPA using React*

There are many possible ways of using web technologies to create an enterprise application. Given our requirements the best way to implement is using a Single Page application or SPA. There are also many ways of implementing a SPA. While it could be done using plain vanilla web technologies (HTML, JS and CSS) it is very common to use a framework to avoid reinventing the wheel to solve problems and implement common features that are necessary but are not provided by the web platform.



Figure 21 React logo. React is a frontend framework that takes care of the view layer.

#### 5.1.1.3 *Module bundler: Webpack*

To be able to efficiently distribute the application, it is a very common practice to use a module bundler. The module bundler takes on the task of combining all the code and files and placing it in one bundle that the browser can download all at once, vastly decreasing the number of requests needed to load the application and increasing efficiency. Webpack also minifies and uglifies the code, which further increases efficiency and makes it harder for someone to reverse-engineer the code delivered to the client-side. The full list of features, many of which are being used can be found in the Webpack documentation (9).



Figure 22 Webpack logo. Webpack is a module bundler for the web .

#### 5.1.1.4 *Components*

##### 5.1.1.4.1 *Data grid*

A critical component in every enterprise application is the Data Grid. We are using a solution that is already being used successfully in other projects within the company: AgGrid. AgGrid is a framework agnostic library that aims to standardize and facilitate the inclusion of customized feature-rich data grids in enterprise applications. In Figure 23 and Figure 24, example usages of the data grid can be seen in the finished application, implementing the various features described before in the Software Design section.

GU Pacients - Demo TOTAL 122759 pacients

Cerca per qualsevol camp... Neteja Filtres

N.H.	Data naix.*	NIF	H./...	Cognom 1*	Cognom 2	Nom*	Mobil*	Tel.*	Email	Idi...
10001	11/03/1931	3942085XX	D	ARQUED	BARON	VIRA	657940XX	628482XX		
10002	01/09/1979	3942085XX	D	CUADRADO	OROZCO	QIANGYING	687380XX	628482XX		
10003	04/08/1985	3942085XX	D	HERRERA	PEREZ	MARTI	636033XX	628482XX		
10004	11/12/1958	3942085XX	H	ROVIRA	SERRANO	LI	630937XX	628482XX		
10005	21/11/1978	3942085XX	H	CARDENETE	RECIO	JORDI	695452XX	628482XX		
10006	01/01/1970	3942085XX	H	MONTANES	SILVA	M <sup>h</sup> JOSEFA	659242XX	628482XX		
10007	20/09/1971	3942085XX	D	BARBERAN	ESCOLA	SARA ISABEL	933915XX	628482XX		
10008	14/07/1937	3893564XX	D	VILLA	COSTA	JUAN PEDRO	932196XX	628482XX		
10009	28/07/1935	3942085XX	D	VILAR	OULEGO	JAVIER	615181XX	628482XX		
10010	13/11/1976	3942085XX	H	ALBA	FERNANDEZ	MARIA JAZ...	627771XX	628482XX		
10011	08/08/1947	3942085XX	H	TORRELO	MORA	FRANCISCO ...	629621XX	628482XX		
10012	20/12/1968	5383052XX	D	ZEKRI	EL HAMIANI	ALEX	617857XX	628482XX		
10013	07/08/1991	3942085XX	D	BERMUDEZ	GONZALEZ	JOAQUIN	683150XX	628482XX		
10014	27/03/1944	3942085XX	D	VIDAL	MAYO	MARIA	934303XX	934303XX		
10015	10/12/1944	2836610XX	D	GOMEZ	ARCOS	ZULEMA	933850XX	628482XX	DIBUMEMO@GMAIL...	
10016	17/05/1935	3942085XX	D	PORRAS	CANDELA	DESIREE	660049XX	628482XX		
10017	01/01/1970	3942085XX	-	PAN	MIKHAIL	MIKHAIL	685872XX	628482XX		
10018	20/01/1961	3942085XX	H	NICOLAU	GONZALEZ	OSCAR	656601XX	628482XX	DIBUMEMO@GMAIL...	
10019	18/07/1959	1841515XX	D	IBAÑEZ	HERNANDEZ	IVAN	616021XX	628482XX	manuel19599@gmail...	
10020	18/06/1964	3942085XX	H	ARRANZ	CORTES	JAIMES NELS...	629107XX	628482XX		
10021	19/11/2003	3942085XX	H	ALBERICH	GOMEZ	YULIA	692994XX	628482XX		
10022	01/01/1970	3942085XX	-	VAZQUEZ	ALEJANDRO	ALEJANDRO	648878XX	628482XX		
10023	23/09/1956	3768003XX	D	ACIN	GUTIERREZ	PEDRO	647295XX	628482XX		
10024	31/12/1799	3942085XX	D	TORREMOR...	FERNANDO	FERNANDO	605541XX	628482XX		
10025	28/06/1945	4620157XX	D	VIDAL	ARACIL	JUAN	699957XX	628482XX		
10026	03/09/1967	3942085XX	D	VENTURA	HERRERA	LUIS ENRIQUE	667486XX	628482XX		

Selecioni o crei un pacient

Figure 23 Data grid example of usage in the final product. This page corresponds to the Patients Page and shows all patients in the system in a Data Grid. All data presented is anonymized to comply with data protection regulation.

GU Cites i visites - SONIA GARCIA ...

Cites i visites SONIA GARCIA IGLESIAS Avui i futures (1) Última setmana (1) Últim any (2) Totes (2) Temporals (0) Cites (1) Visites (1)

**C509540 DXA** Fixa visita Cal. Maq. admin

DXA DENSITOMETRIA 16/09/2021 08:00 08:15 15 min (Ult. Edició: admin)

Comentari cita Comentari fibra Comentari tècnic

Cita 14/10 13:23 E. Recep En sala d'espera Cridat Dins visita Fora visita

Cita + Cita Linya + Linya Linya Facturar 1/1 Veure factures OBLIGATORI: Documents

Visita	Prova	min	Centre referent	Referent	E. Prevista	D. Entregat	N. Fact...	M. Pagament	Client	F. Pagament	Barem / Tarifa
V506367	01 RM COLUMNNA LUMBAR O LU...	30'		Dr BARUTEL	12/08/2021	12/08/2021	3824 ✓	(Privat)	GARCIA IGLESIAS...	--	35344 RM COLUMNNA LUMBAR (1) (25)
	02 RM COLUMNNA CERVICAL	30'		Dr BARUTEL	12/08/2021	12/08/2021	3824 ✓	(Privat)	GARCIA IGLESIAS...	--	35341 RM COLUMNNA CERVICAL (1) (25)
C509540	01 DENSITOMETRIA	15'			20/09/2021			(Privat)	GARCIA IGLESIAS...	--	(0) (25)

509540 no té cap document Cancel·lar cita No vingut Recordatori Etiqueta F. Blanc Torn Justificant Tots els barems

Figure 24 Example of custom usage of the Data Grid in the final implementation of Visits Page.

### 5.1.1.4.2 Calendar

In order to implement the calendar, the consensus was to use an external library instead of doing an implementation of our own. After researching and comparing the available alternatives, FullCalendar was chosen. However, FullCalendar does not have all the needed customization options by default and some work on top of it was necessary before it was ready to suit our needs. In Figure 25 an example usage of the calendar in the Schedule Page of the finished application can be seen.

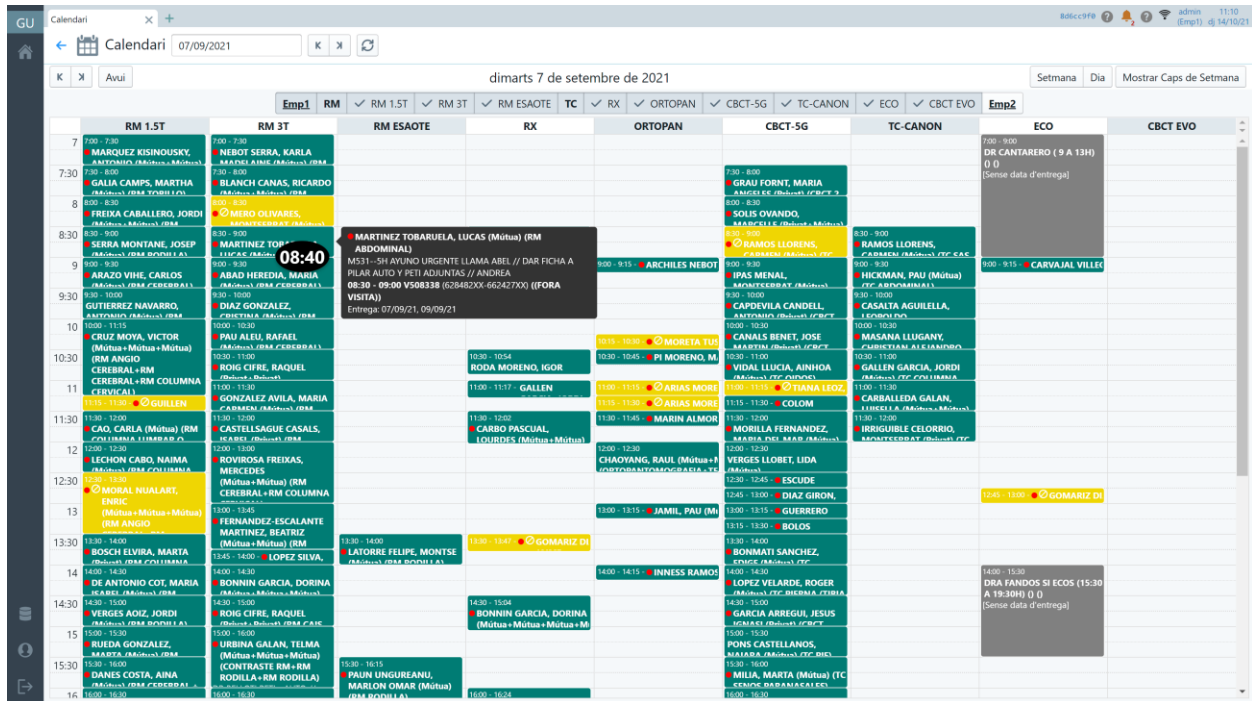


Figure 25 Example usage of the Calendar library found in the Schedule Page. This result was achieved by using FullCalendar as a base and adding some additional features on top such as custom event contents, mouse over tooltips and a time indicator by the side of the cursor.

### 5.1.1.4.3 Form components

For this project, we used the internal library of form components that provide the basic features to let the user edit and perform basic operations on data. Those components are used ubiquitously in the project.

Components are sourced from various sources, but special credit must be given to the awesome open-source component library made available by Palantir: BlueprintJS. This library also served for inspiration for defining the design system used throughout the application. Examples of those components can be found in its Documentation (10) (11).





Figure 26 Logo of BlueprintJS. Main source of components and inspiration for the design system.

#### 5.1.1.5 *State management: MobX*



Figure 27 MobX Logo. MobX is the frontend state management library used to solve the state management needs in this project.

State management is a crucial part of implementing a Single Page Application. There are lots of state management solutions available and there is no clear consensus on which one is the best. After considering other alternatives and testing it in smaller experimental projects, we decided to choose MobX as our state management library. We also evaluated Redux, the competing solution, in-depth but after carrying out these tests we came to the conclusion that MobX would allow us to move faster and achieve a greater development speed with little to no disadvantages.

MobX essentially enables reactive programming in Javascript and Typescript. It does so by allowing the developer to define state, actions and derivations (11). It then uses its internal engine to trigger recalculations for values when and only when needed. The diagram in Figure 28 shows how MobX uses property accesses to track dependencies between state and actions and derivations (12). Based on this concept, MobX provides a sophisticated but easy to use reactive state management system. MobX is used throughout the frontend to solve all the reactivity and state management needs.

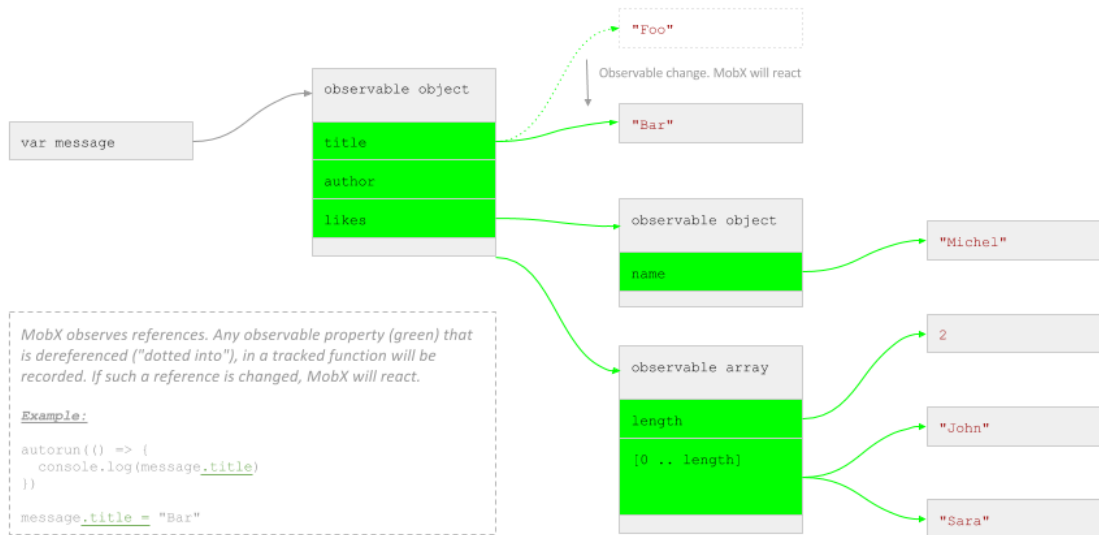


Figure 28 Diagram showing how MobX extends the language to track dereferences (or property accesses) to build a reactive engine on top of JavaScript. More information in MobX Documentation. Source: MobX Documentation.

Figure 29 Image explaining how MobX reactivity works. Source: MobX Documentation.

## 5.2 Backend

### 5.2.1 Technology used

#### 5.2.1.1 NestJS



Figure 30 Logo of NestJS, the framework of choice for this project

NestJS is our backend framework of choice built on top of NodeJS. The framework is heavily inspired by the Angular framework by Google but translates those concepts to the backend. The main job of NestJS is to give a general and predictable structure to the project. This allows for easier onboarding of new team members because they may already be familiar with the structure presented by NestJS and even if they are not, they can be referred to NestJS documentation. It is recommended to take a look at NestJS documentation (13) to better understand the attached source code of the project.

#### 5.2.1.1.1.1 Example of a NestJS Controller

This section includes an implementation of a controller. Controllers are the entry points to the system and represent its exposed external API. All the methods annotated with `@Method('<url>')` get mapped to HTTP REST endpoints using the specified URLs and HTTP methods. An implementation for a Controller looks as follows, using the DocumentController as an example:

```
@Controller('document')
export class DocumentController {
  constructor(public service: DocumentService) {
  }
  @Patch(':documentId')
  async saveDocument(@Param('documentId') documentId, @Body() body) {
    return await this.service.saveDocument(parseInt(documentId), body)
  }
  @Get(':documentId/dl')
  async dlDocument(@Param('documentId') documentId) {
    return this.service.getDocumentForDl(parseInt(documentId))
  }
  @Get(':documentId/ul')
  async ulDocument(@Param('documentId') documentId) {
    return this.service.getDocumentForUl(parseInt(documentId))
  }
}
```

#### 5.2.1.1.1.2 Example of a NestJS Service

This section includes an implementation of a service. Services expose all the required functionality which may then be used either by other services or by controllers

Using the DocumentService as an example. The service, in this case, exposes all the functionality related to the handling of documents.

```
@Injectable()
export class DocumentService extends TypeOrmCrudService<Document> {
  constructor(
    @InjectRepository(Document) repo,
    @InjectRepository(Document)
    private readonly documentRepo: Repository<Document>,
    @Inject(MINIO_CONNECTION) private readonly minioClient: MinioClient,
  ) {
    super(repo);
  }

  async getUploadUrlForDoc(documentName) {
    return await new Promise((resolve, reject) => {
      this.minioClient.presignedPutObject(process.env.DOCUMENTS_BUCKET, documentName, (err, url) => {
        if (err) reject(err);
        resolve(url);
      });
    });
  }

  async getDownloadUrlForDoc(documentName) {
    return await new Promise((resolve, reject) => {
      this.minioClient.presignedGetObject(process.env.DOCUMENTS_BUCKET, documentName, (err, url) => {
        if (err) reject(err);
        resolve(url);
      });
    });
  }

  async getUrlForDocs(documentObjects: { name: string, [any: string]: any }[], upload: boolean) {
    for (let i = 0; i < documentObjects.length; i++) {
      let docObj = documentObjects[i];
      if (upload) {
        let url = await this.getUploadUrlForDoc(docObj.name);
        docObj.uploadUrl = url;
      } else {
        let url = await this.getDownloadUrlForDoc(docObj.name);
        docObj.downloadUrl = url;
      }
    }
  }
}
```

```

    }
  }
  return documentObjects;
}

async getUrlForDoc(documentObject, upload: boolean) {
  return _.first(await this.getUrlForDocs([documentObject], upload));
}

async createNewDocuments(body) {
  if (!_.isArray(body)) body = [body];
  let getName = (docName) => {
    let split = docName.split('.');
    return `${_.first(split).substring(0, 16)}-${nanoid(8)}.${_.last(split)}`;
  };
  body.forEach(d => d.name = getName(d.name));
  body.forEach(d => d.createdAt = new Date());
  let res = await this.documentRepo.save(body);
  let resWithUrl = await this.getUrlForDocs(res, true);
  return resWithUrl;
}

async saveDocument(documentId, body) {
  let document = await this.documentRepo.findOne(documentId, { relations: ['visits', 'visitLines'] });
  let addOnlyMode = false;
  if(addOnlyMode){
    if (body.visits)
      body.visits = _.uniqBy([...body.visits, ...document.visits], v => v.id);
    if (body.visitLines)
      body.visitLines = _.uniqBy([...body.visitLines, ...document.visitLines], v => v.id);
  }
  await this.documentRepo.save({
    ...body,
  });
  return { saved: true }
}

async getDocumentForDl(documentId) {
  let document = await this.documentRepo.findOne(documentId);
  if (document == null) throw new NotFoundException('Document not found');
  return await this.getUrlForDoc(document, false);
}

async getDocumentForUl(documentId) {
  let document = await this.documentRepo.findOne(documentId);
  if (document == null) throw new NotFoundException('Document for upload not found');
  return await this.getUrlForDoc(document, true);
}
}

```

### 5.2.1.2 TypeORM

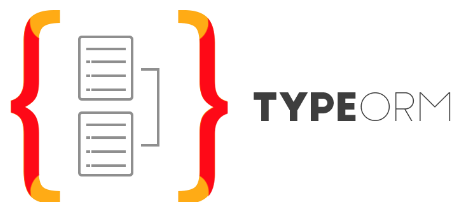


Figure 31 Logo of TypeORM, the ORM of choice for this project

Our ORM of choice is TypeORM. Although it has some caveats, this ORM is already used successfully in other projects developed in the same company. We use it to perform basic – but very common and repetitive – interactions with the database. Those usually fall under the category of CRUD (Create, Read, Update and Delete) operations, but there are some useful advanced features of the ORM being used. All the TypeORM features are detailed in its Documentation (13).

Another advantage of using the ORM is that it automatically escapes parameters for you, thus preventing SQL Injection attacks and other user input-related software bugs that would be present if the queries were sent to the database without escaping parameters by accident.

#### 5.2.1.3 *Manual SQL Queries (for Postgres)*

In many cases, the features, efficiency and quality of the generated queries provided by the orm, in our case TypeORM was not enough. In that case, we bypassed the ORM and used manually crafted queries. This offered the best of both worlds. We were able to increase the development speed, maintainability and changeability of the project by using an ORM. But we were also able to use the advanced capabilities of the database for querying data in complex ways. This is an important design decision because it allows us to offload the processing of data to the database which is highly optimized for that task. In our case the chosen database is Postgres, which apart from being heavily multithreaded, it is written in C making it faster by nature than our NodeJS business logic server. NodeJS is slightly more focused in development speed than performance so it's a good idea in general to offload complex tasks to the database to increase performance where needed to fulfill non-functional requirements.

## 5.3 Database

### 5.3.1 PostgreSQL

The chosen RDBMS is PostgreSQL . While alternatives were evaluated, PostgreSQL was found to fit our use case and is already being used in other solutions in the same company. The fact that recent versions brought many improvements also helped to make this decision.

The manual queries used use standard SQL in general, but there are no special restrictions and coupling to the PostgreSQL dialect is allowed. This is the case because switching databases is not expected given the status of the PostgreSQL project and that there is no cost for using the software. If in the future there was the need for some custom feature that can only be implemented in the database, it could be developed using a PostgreSQL extension, which minimizes the risk of blocking the project for a database related cause.

In the next section we present the schema used for the database including all tables for all the features regarding this project. The entire database schema diagram is attached, but not shown inline because of size constraints.

### 5.3.2 Schema of entities used by the features in the scope of this project

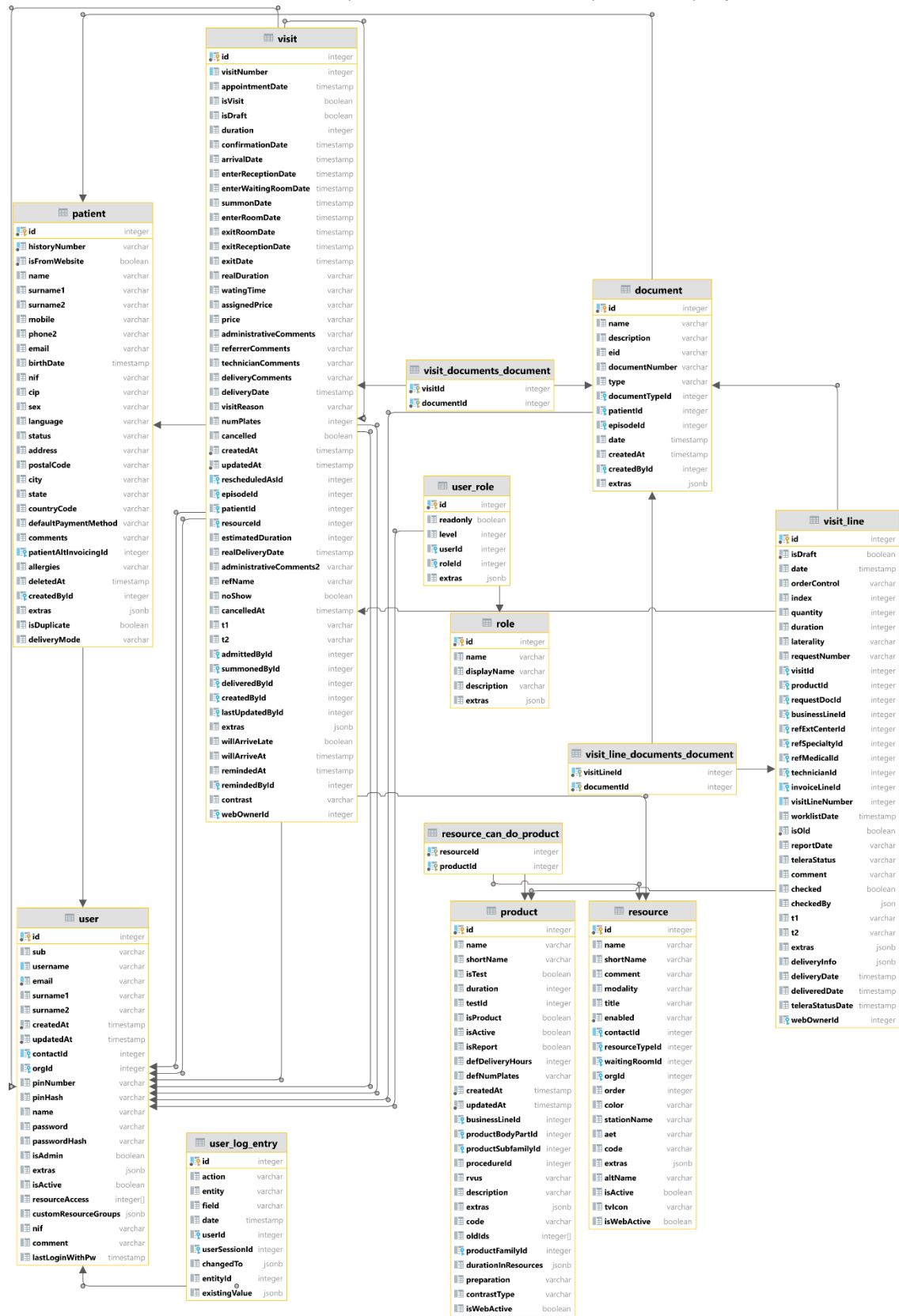


Figure 32 Diagram showing the tables relevant to the part of the project addressed in this document. The entire database schema is much larger and does not fit in a single page.

### 5.3.3 Fine tuning

In our case we were able to increase the performance of the database by paying attention to its configuration.

We found that due to the scale and budget of the project it was appropriate to use a higher end dedicated cloud server that used an SSD and provided 32 Gigabytes of main memory.

We then modified the configuration of PostgreSQL to take advantage of this hardware capabilities. We used the PgTune calculator found online (14) along with other online information and trial and error using relevant benchmark queries to find the best values for our configuration.

Our application falls into the category of an OLTP (Online Transaction Processing) kind of application. This means that there may be some long transactions or complex queries but also lots of simple ones. It is important to note that the business intelligence needs are not taken care of using this database.

Main changes made to the configuration:

- Increase the maximum memory usage: The default memory usage limit much smaller than the amount we had available on the server. We changed this value to 26 GB, leaving the rest of the 32 GB for the Business Logic Server instances (NodeJS processes) and other services.
- Decrease the heuristic cost of going to disk: Since the server used an SSD and the default configuration is prepared for slower HDDs, we researched what value was appropriate for this configuration option and used it.

In the future if the system needs to scale and scaling vertically is not an option, we could easily use another server only for the database. However, having the database in the same machine as the web servers offers an advantage that must not be overlooked: Latency. It is important to note that when accessing the database in the same machine the latency is close to zero. While in a datacenter latencies are very small between servers, it can matter when lots of queries are required to perform an operation, as described in this article: “How Millisecond Delays May Kill Database Performance” (14).

## 5.4 Object Storage (S3)

S3 is a term popularized by Amazon Web Services. It was originally a revolutionary product that offered storage as a service through a web interface. Since then it has evolved to a standard for handling large blobs of data in the web. It often takes the place and handles the features FTP servers would have handled in the past. Using an S3 service instead of handling uploads using application code offers the benefit that document uploads and downloads can use a secure side-channel which is more efficient and does not block other more trivial requests being made to the server. This especially important when using NodeJS because every instance is single-threaded so long-running requests can put many others on hold, reducing the performance for all users and threatening to fail at fulfilling non-functional requirements for the system.

There are many vendors that offer S3 storage and also many self-hosted options. In our case a decision to start by self-hosting using MinIO was made. It is important to note that at any moment MinIO can be replaced by another S3 compatible option such as AWS S3. It can be replaced individually on select production environments or for all of them since there is no specific coupling to MinIO.



Figure 33 MinIO logo. MinIO is a self-hosted S3 storage solution (15).



## 5.5 Twelve-Factor App

We tried to incorporate the philosophy behind the Twelve-Factor App. The twelve-factor app is a methodology for building software-as-a-service apps. It concerns both the architecture of the software and the processes used to build and deploy it. When an app fulfills the twelve factors, it is a trivial task to convert it to a cloud-native app, which we intend to do in the future.

The twelve factors (quoted from the official website) (16):

### *I. Codebase*

*One codebase tracked in revision control, many deploys*

### *II. Dependencies*

*Explicitly declare and isolate dependencies*

### *III. Config*

*Store config in the environment*

### *IV. Backing services*

*Treat backing services as attached resources*

### *V. Build, release, run*

*Strictly separate build and run stages*

### *VI. Processes*

*Execute the app as one or more stateless processes*

### *VII. Port binding*

*Export services via port binding*

### *VIII. Concurrency*

*Scale out via the process model*

### *IX. Disposability*

*Maximize robustness with fast startup and graceful shutdown*

### *X. Dev/prod parity*

*Keep development, staging, and production as similar as possible*

### *XI. Logs*

*Treat logs as event streams*

## *XII. Admin processes*

*Run admin/management tasks as one-off processes*

## 6 Deployment, Workflow and CI / CD

### 6.1 Environments

The project is using Git as its version control system. The branching model is very simple: There are only two main branches:

- **Production branch (master):** Contains the latest production-ready version. When the development branch is considered stable it gets merged into master. Alternatively, in some emergency cases where a fix is needed as fast as possible, it is possible to work on a hotfix directly to the master branch.
- **Development branch (develop):** Contains the latest changes in all the issues the team is working on.

The convention is to try and branch as little as possible and keep the latest version always in the development branch. This is a convention extracted from DevOps culture, where the norm is to face integration challenges as early as possible as explained in Trunk-Based Development (17).

These branches are automatically deployed to three different environments:

- **Development (dev):** Used internally by the team to see and test the latest version
- **Pre-production or staging (next):** Used internally to ensure the stability of the application before deploying the exact same version to production.
- **Production (prod):** Deployments to production are either manual or require all end to end tests to pass when there is sufficient coverage in the codebase. Merging to master and deploying to production should be done as frequently as possible while maintaining the quality standards for the delivered software.

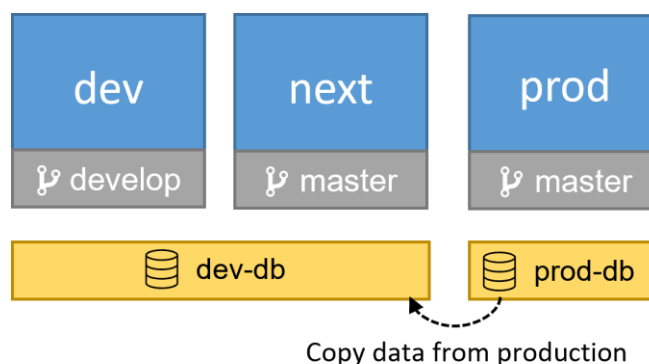


Figure 34 Diagram illustrating the three environments, the branches they use as source for their deployments and the databases they are connected to.

## 6.2 Server architecture

### 6.2.1 Nginx (Reverse Proxy)

Nginx describes itself as web server capable of acting as an HTTP and reverse proxy server, a generic TCP/UDP proxy server, and a mail proxy server. According to the Netcraft October 2021 Web Server Survey, it is being used in production as a reverse proxy for 22.5% of the most visited sites in the internet (17).

The primary responsibility that Nginx has in our system is to route incoming requests to the services they intend to access by using information included in the http header. In our case we are using the subdomain to make this distinction.

Another Nginx feature being used is load-balancing. Nginx can perform load-balancing across many identical instances of the same application. Since NodeJS is single-threaded by design, the only way to use more than one server cpu core is to run multiple identical instances as separate processes and load-balance between them. Nginx load-balancing has many modes of operation to fit different use cases.

Nginx is capable of operating on Layer 7 (Application level) and on Layer 4 (TCP/IP level) of the OSI networking model. In our case we must set it to operate on Layer 7 because we need to perform TLS Termination. When using TLS termination, Nginx establishes TCP connections with the requesters and use our TLS certificate do decrypt and encrypt ingress and egress traffic, that is traffic that originates at or is going to external clients which are making the requests. It then fulfills those requests by simultaneously connecting to the relevant backend service using internal unencrypted http requests. This allows us to delegate the responsibility of handling TLS encryption to Nginx, which is more efficient and allows us to develop services as if they were going to be accessed using plain HTTP.

It is also possible to choose the algorithm that Nginx uses to load balance. There is the option of switching between processes in a Round Robin fashion (requests split evenly between instances) and also there is the option of using more advanced algorithms such as least connections. In our case, we stayed with the default which is the Round Robin algorithm because it fits our use case. If in the future we detect any issue or performance bottleneck we could reevaluate this decision.

It is important to note that for us to be able to implement a load-balancing strategy at the instance level like this one we must ensure that our instances are stateless. In particular, it must be possible to send any request coming from any user to any instance and the response has to be exactly the same. The instances are only allowed to store some state if doing so doesn't break this premise.

In Figure 35 we can see a diagram illustrating how ingress traffic is able to access the desired services. Note that many services are being exposed even though there is only one port open to the internet. The green blocks represent Nginx rules for forwarding to the different backends with "@" representing the base domain used for the whole deployment. End users are going to access the system using the FQDNs (Fully Qualified Domain Names) shown in the green blocks. To give an example, if the base domain is "example.com", the development environment would be accessible at "https://dev.example.com" and the pre-production environment would be accessible at "https://next.example.com". Other supporting systems such as s3 or Jenkins are also accessed in a similar way as shown in the diagram.

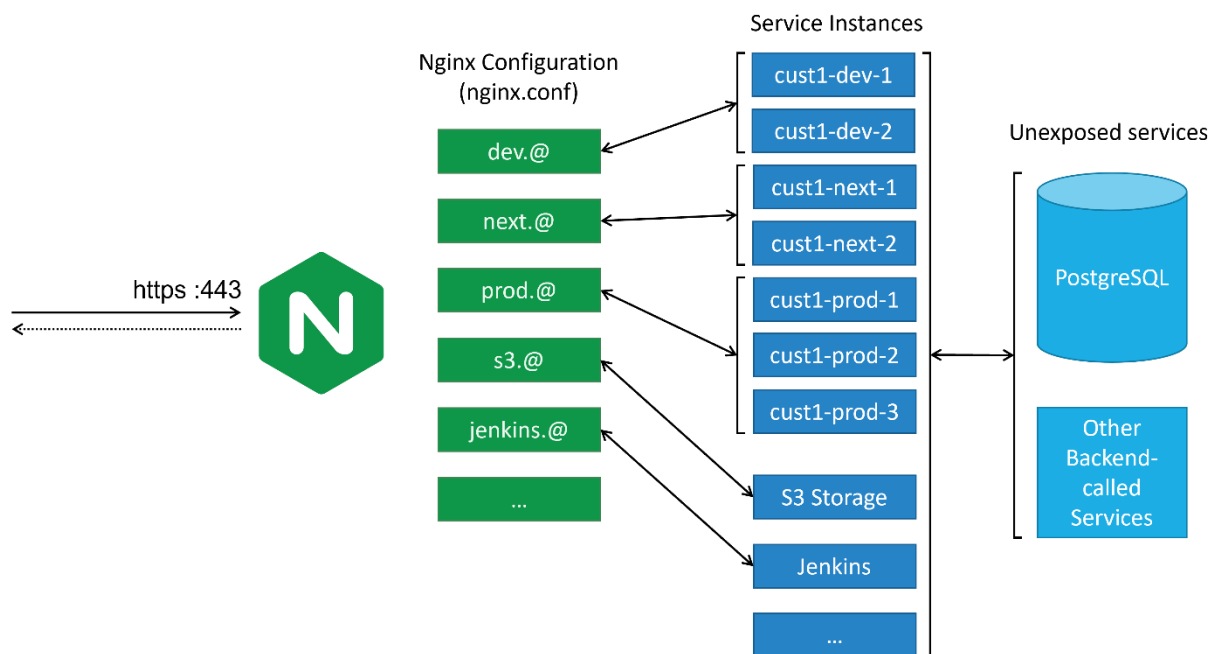


Figure 35 Diagram illustrating how ingress traffic<sup>2</sup> gets delivered to the different exposed services based on the requested subdomain. Note that not all services are directly accessed from the outside. In the diagram @ represents the domain through which the server is exposed to the internet.

### 6.2.2 PM2 (Process manager)

Another critical piece of infrastructure is PM2. PM2 describes itself as a NodeJS process manager (18). It takes care of running the code in the cloud machine. It provides features that make it adequate for running workloads like our backend code in production which need to be highly available. PM2 provides the following features:

- **Automatic restarts on error:** It restarts processes immediately when they raise unexpected exceptions or otherwise crash.
- **Memory usage limits and alerts:** It monitors the memory usage and it can be set up in a way that a process is automatically restarted if memory usage goes above a certain threshold. This allows for the mitigation of the impact of memory leaks. PM2 then keeps track of the causes for restarts so an administrator can come later and diagnose the issue with minimal impact to the availability of the system.

<sup>2</sup> Ingress traffic consists of all incoming requests from the outside that expect to land on a specific service within the server. The information indicating which service to access is usually included in the URL when using the http protocol.

- **Health checks:** It detects, disconnects and replaces a process if it detects it is unable to successfully respond to requests
- **Basic monitoring:** The ability to monitor resource usage and network activity for each of the running processes
- **Log aggregation:** The ability to read and analyze log and error output from all or some of the running processes at once. This allows for quick diagnose of issues in production.
- **Infrastructure as code:** The processes that should be running can be defined using a JavaScript configuration file. This allows to define the processes that should be running based on arbitrary logic and allows us to automatically deploy the different defined environments for every instance. This is especially useful since when multiple users have their own instances the number of processes that would need to be manually configured would be unmanageable.



Figure 36 PM2 logo. PM2 is a process manager for NodeJS adequate for production environments. It performs automatic restarts, healthchecks, monitoring, log aggregation and more.

### 6.3 Jenkins (CI / CD)

Jenkins is used in order to perform automated deployments with git integration and run scheduled jobs (19). There are lots of alternatives. In this case the decision was made to go with the default choice for projects in the company because it was a safe choice and we did not have any special requirements regarding CI/CD pipeline features. We also used it to run other scheduled jobs such as data backups.



Figure 37 Jenkins logo. Jenkins is a platform for easily automating tasks. In our case CI/CD deployments and database backups.

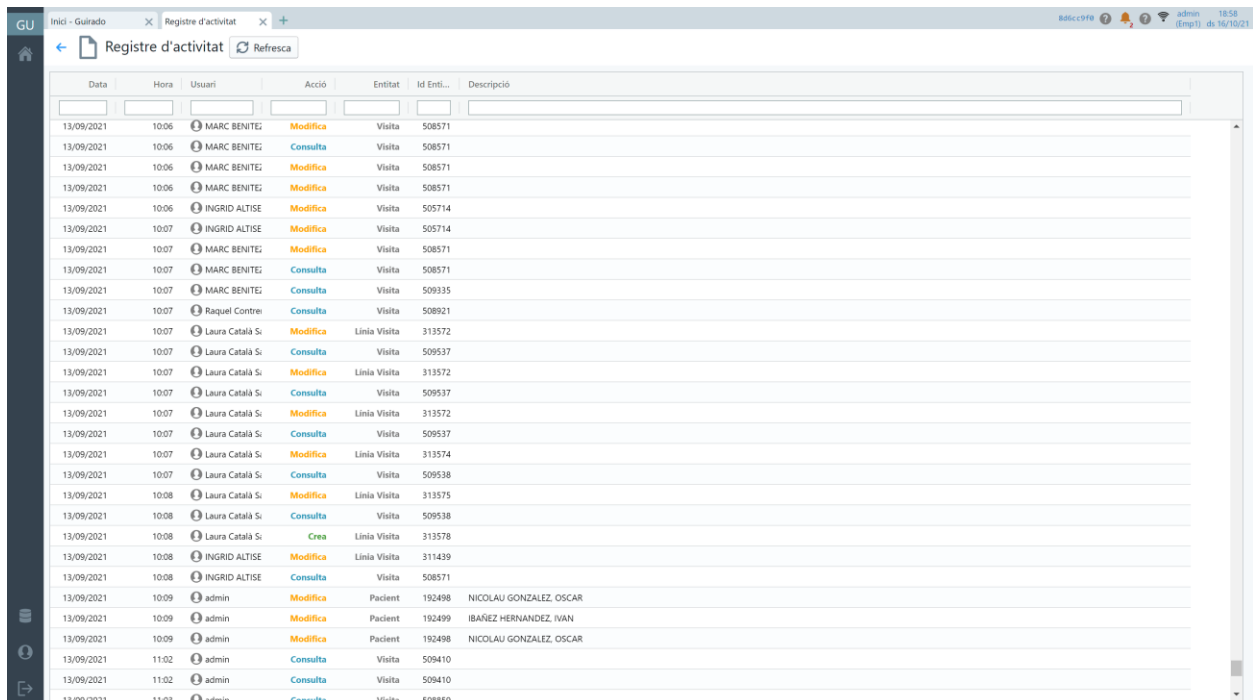
## 7 Legal Aspects

MedicalOne is targeting customers within the European Union for the first versions of the product. This means it is necessary to comply with applicable data protection laws and regulations. The main applicable legislation to take into account is the GDPR (General Data Protection Regulation).

The GDPR states that any personal data, that is any data that is able to be traced back to the natural person that generated it, must be treated according to this regulation.

The user must be informed of which data is going to be collected

Additionally, especially in the medical field it is necessary for the system to keep a record of user actions such as reading, creating, updating, or deleting personal data. To fulfill this requirement we keep track of all operations that involve personal data and present them in a convenient UI to system administrators as shown in Figure 38. The data is being stored in the 'user\_log\_action' table that has been included in the diagram in Figure 24.



Data	Hora	Usuari	Acció	Entitat	Id Enti...	Descripció
13/09/2021	10:06	MARC BENITEZ	Modifica	Visita	508571	
13/09/2021	10:06	MARC BENITEZ	Consulta	Visita	508571	
13/09/2021	10:06	MARC BENITEZ	Modifica	Visita	508571	
13/09/2021	10:06	MARC BENITEZ	Modifica	Visita	508571	
13/09/2021	10:06	INGRID ALTISE	Modifica	Visita	505714	
13/09/2021	10:07	INGRID ALTISE	Modifica	Visita	505714	
13/09/2021	10:07	MARC BENITEZ	Modifica	Visita	508571	
13/09/2021	10:07	MARC BENITEZ	Consulta	Visita	508571	
13/09/2021	10:07	MARC BENITEZ	Consulta	Visita	509335	
13/09/2021	10:07	Raquel Contre	Consulta	Visita	508921	
13/09/2021	10:07	Laura Català S	Modifica	Linia Visita	313572	
13/09/2021	10:07	Laura Català S	Consulta	Visita	509537	
13/09/2021	10:07	Laura Català S	Modifica	Linia Visita	313572	
13/09/2021	10:07	Laura Català S	Consulta	Visita	509537	
13/09/2021	10:07	Laura Català S	Modifica	Linia Visita	313572	
13/09/2021	10:07	Laura Català S	Consulta	Visita	509538	
13/09/2021	10:07	Laura Català S	Modifica	Linia Visita	313574	
13/09/2021	10:07	Laura Català S	Consulta	Visita	509538	
13/09/2021	10:08	Laura Català S	Modifica	Linia Visita	313575	
13/09/2021	10:08	Laura Català S	Consulta	Visita	509538	
13/09/2021	10:08	Laura Català S	Crea	Linia Visita	313578	
13/09/2021	10:08	INGRID ALTISE	Modifica	Linia Visita	311439	
13/09/2021	10:08	INGRID ALTISE	Consulta	Visita	508571	
13/09/2021	10:09	admin	Modifica	Pacient	192498	NICOLAU GONZALEZ, OSCAR
13/09/2021	10:09	admin	Modifica	Pacient	192499	IBAÑEZ HERNANDEZ, IVAN
13/09/2021	10:09	admin	Modifica	Pacient	192498	NICOLAU GONZALEZ, OSCAR
13/09/2021	11:02	admin	Consulta	Visita	509410	
13/09/2021	11:02	admin	Consulta	Visita	509410	
13/09/2021	11:02	admin	Consulta	Visita	508860	

Figure 38 Activity Record page. This page shows the record of actions performed by all users sequentially. It also provides a filter UI to allow administrative staff to perform quick investigations.

## 8 Conclusions

### 8.1 Accomplished objectives

At the time of writing, the software is installed and being used on three separate Radiology Centers. It is also important to note that the development and installation of the software have been done in the expected timeframe within margin of error. This implies that the main objective has been accomplished.

In detail, the following objectives have been accomplished successfully:

1. **Develop and deploy the software solution:** As said, this was the main objective and its worth not within the expected timeframe
2. **Cover the planned scope:** The resulting software at the end of the project allows the radiology centers to operate completely and with improvements with respect to the last software solution being used already on the first version.
3. **Fulfill non-functional requirements:** We faced some challenges meeting performance requirements in the case of very low-end thin client devices but were able to overcome them by devoting extra time to researching and applying optimizations to the frontend code.

### 8.2 Technical competences accomplishment in relation to the software engineering specialization.

- **CES1.1: To develop, maintain and evaluate complex and/or critical software systems and services. [In Depth]**
  - This competence is accomplished exhaustively because the product being developed covers a large enough scope and stands in the middle of critical processes being carried out daily in the radiology center.
- **CES1.2 To solve integration problems in function of the strategies, standards and available technologies. [In Depth]**
  - A very big part of this project consists of exploring different possible solutions and technologies, evaluating them and selecting the combination of them that together are going to be part of the final product. Naturally this includes integrating those tools and technologies to conform a working final product that fulfills all functional and non-functional requirements.
- **CES1.3 To identify, evaluate and manage potential risks related to software building which could arise. [Medium]**
  - Throughout the project we have been analyzing the possible risks that could threaten the success of the project. In fact, there have been some changes made with respect to the initial planning in order to avoid some of those risks
- **CES1.4 To develop, maintain and evaluate distributed services and applications with network support. [Somewhat]**



- Since we are building a web application, we are in fact building a distributed system which consists of the cloud server and each one of the clients that use it.
- **CES1.5 To specify, design, implement and evaluate databases. [In depth]**
  - We spent a lot of effort crafting first the domain model and later the database schema. We also spent some time evaluating which RDBMS to use. In the final stage of the project we fine-tuned the database to perform best on the cloud instance we used.
- **CES1.8 To develop, maintain and evaluate control and real-time systems. [A little]**
  - The system is built to reflect changes to all pages in real time to allow collaboration and prevent data desynchronization.
- **CES2.1 To define and manage the requirements of a software system. [In Depth]**
  - Defining, managing and finally fulfilling software requirements has been a very important part of this project.
- **CES2.2 To design adequate solutions in one or more application domains, using software engineering methods which integrate ethical, social, legal and economical aspects. [A little]**
  - Ethical, social and legal aspects were considered and included in this document under the sustainability report section. Legal aspects were also considered.

### 8.3 Planning and budget review

After reviewing the initial planning, we found that all the planned tasks have been completed successfully. Since the project required a lot of exploration and experimentation, some tasks were not performed in the same order they were planned. Also, some tasks were adapted as new insights came from the client and others were discarded or replaced by better approaches that were not considered in the planning phase.

Overall, all the required features plus some extra ones were implemented in the expected time frame of the project.

It is important to note that the initial planning has been useful in providing certainty to many of the major stakeholders in the project. In particular, when taking into account the planning for the whole system, it allowed us to give the client an accurate estimate for the expected duration of the project. It also allowed us to calculate the estimated cost and evaluate the viability of the project.

Since there were no major deviations in the timeline, there were no major deviations in the budget either.

### 8.4 Future work

- **Improve the server architecture and increase automation:** This would be important in the case of increasing the number of clients the company has using this solution. In order to be able to manage deployments and backups in a multi-production environment it is essential to have as much automation in place as possible. One option for doing that, although it should be carefully evaluated is to port the implemented system to cloud native technologies and deploy it using containers and an orchestrator such as Kubernetes or Hashicorp Nomad.

- **Implement more features:** That would increase the scope of the system and allow to replace some existing legacy systems that are currently being used.
- **Add Integrations with external systems:** The system is definitely not operating in an isolated environment but rather it is surrounded by other systems in all areas of business. Often, integrating with those adds automation and increases the efficiency of processes, requiring less manual handling of data.
- **Add a public external API:** This would allow other companies to make integrations for their software solutions without MedicalOne as a company having to intervene in the process of making those. This allows for a scalable way of increasing the variety of use cases the software can be used in.
- **Marketing of the newly born solution:** After finishing the first usable version of the product, we must not forget that in order to be used, it needs to be known. Now is the time to start making marketing and awareness campaigns so potential costumers get to know that it is an option at the time of making a choice for a new solution.
- **Internationalization:** Currently the application is only available in one language. As clients from other regions or that prefer to use another language come it will be essential to introduce multi language support as well as the ability to format numbers according to other locales and offer support for other currencies and units among other features required to allow users from other countries to adopt the software. While it has still not been implemented, an internationalization strategy is already in place in the project.

## 9 References

1. O'Connor, S. What Is a Radiology Information System? *Adsc.com*. [Online] 2021. [Cited: April 8, 2021.] <https://www.adsc.com/blog/what-is-a-radiology-information-system>.
2. MedicalOne Website Home Page. *MedicalOne Website*. [Online] [Cited: April 06, 2021.] <https://www.medicalone.com/es/inicio/>.
3. *Salary Expert*. [Online] [Cited: April 8, 2021.] <https://www.salaryexpert.com/>.
4. How Much Does Custom Software Cost in the Long Run? *Worthwhile.com*. [Online] 2021. [Cited: May 19, 2021.] <https://worthwhile.com/insights/2017/09/11/software-long-term-costs/>.
5. Musthaler, L. Energy-aware software design can reduce energy consumption by 30% to 90%. *Network World*. [Online] 2021. [Cited: May 14, 2021.] <https://www.networkworld.com/article/2861005/energy-aware-software-design-can-reduce-energy-consumption-by-30-to-90.html>.
6. IEEE Communications Society. Energy Efficiency in Data Centers. *Comsoc.org*. [Online] 2021. [Cited: May 14, 2021.] <https://www.comsoc.org/publications/tcn/2019-nov/energy-efficiency-data-centers>.
7. Peurakoski, T. Automation and the Quality of Life. *SogetiLabs*. [Online] 2021. <https://labs.sogeti.com/automation-and-the-quality-of-life/>.
8. TypeScript. [Online] 2021. [Cited: October 15, 2021.] <https://www.typescriptlang.org/>.
9. Webpack. [Online] 2021. [Cited: October 16, 2021.] <https://webpack.js.org/>.
10. BlueprintJS Documentation. *BlueprintJS*. [Online] <https://blueprintjs.com/docs/>.
11. The gist of MobX. *MobX*. [Online] <https://mobx.js.org/the-gist-of-mobx.html>.
12. Understanding reactivity. *MobX Documentation*. [Online] 2021. [Cited: September 03, 2021.] <https://mobx.js.org/understanding-reactivity.html>.
13. TypeOrm Documentation. *TypeOrm*. [Online] 2021. [Cited: September 3, 2021.] <https://typeorm.io/>.
14. How Millisecond Delays May Kill Database Performance. [Online] 2021. [Cited: September 6, 2021.] <https://blog.packet-foo.com/2014/09/how-millisecond-delays-may-kill-database-performance/>.
15. MinIO. *MinIO*. [Online] 2021. [Cited: September 07, 2021.] <https://min.io/>.
16. The Twelve Factor App. [Online] 2021. [Cited: April 9, 2021.] <https://12factor.net>.
17. October 2021 Web Server Survey. *Netcraft*. [Online] 2021. [Cited: September 28, 2021.] <https://news.netcraft.com/archives/2021/10/15/october-2021-web-server-survey.html>.
18. PM2 Documentation. [Online] 2021. [Cited: October 5, 2021.] <https://pm2.io>.
19. Jenkins Documentation. [Online] 2021. [Cited: September 23, 2021.] <http://jenkins.io>.