deti universidade de aveiro departamento de eletrónica, telecomunicações e informática

TQS: Product specification report

Pedro Ramos [107348], João Luís [107403], Daniel Madureira [107603], Rodrigo Aguiar [108969]
v2024-04-19

1 Introduction

1.1 Overview of the project

The objective of this project is to propose, conceptualize and implement a multi-layer application, using a software architecture based on *enterprise frameworks*. Also apply a *Software Quality Assurance* (SQA) strategy, and integrate CI/CD on our project.

Therefore, **PhiHub** is going to be developed to enhance patient management at their upcoming hospital facilities. The system will be able to schedule new appointments for patients, and also check their “agenda” and access summaries of past consultations. Patients will also be able to follow the process of calling for appointments via screens.

The staff will be able to check-in and handle registrations from patients and handle administrative tasks.

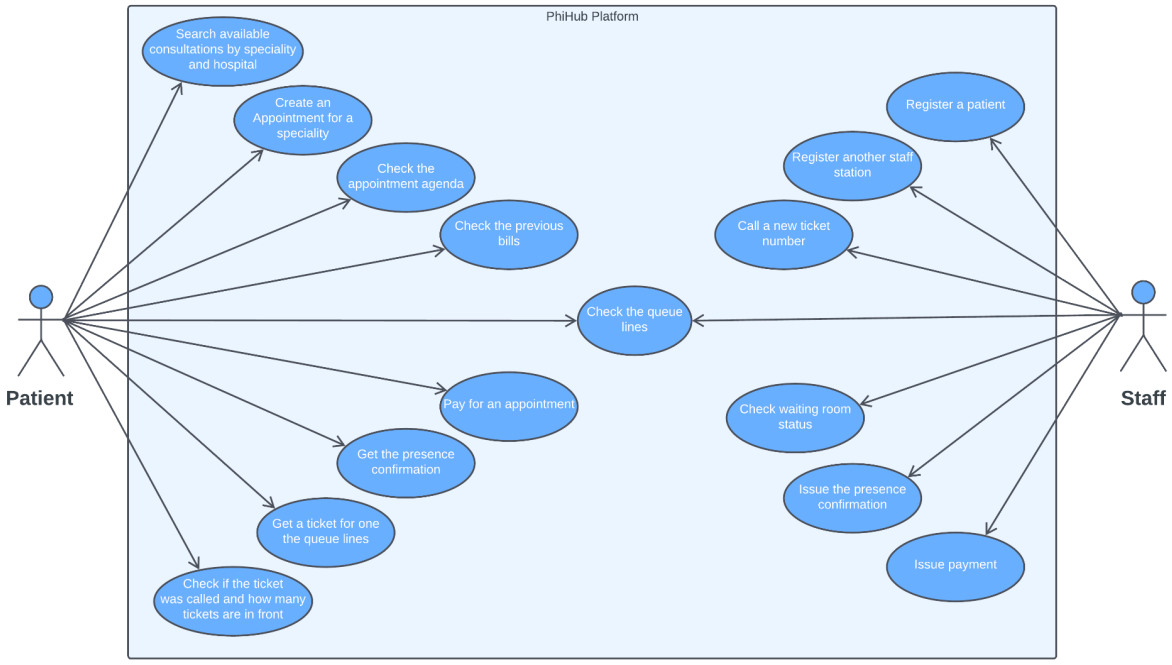
1.2 Limitations

<explain the known limitations/unimplemented (but planned) features>
For the system observability, we tried to implement an ELK stack, but not successfully.
The Payment feature is also limited, because we did not implement an actual integration with an 3rd party service to handle payments.

2 Product concept and requirements

2.1 Vision statement

- Our aim with PhiHub is to develop a system where patients, technical staff and doctors can associate appointments, bills, tickets and queues, as to simulate the entire workflow between registering an appointment, waiting for the appointment inside the hospital and paying for said appointment;
- This project tries to resolve the complex problem of organizing and maintaining a good flow of patients as to minimize wait time and mistakes from the staff;
- Our system will be similar to several others already in use by hospitals around Portugal, but while these systems usually only focus on one of the three main functionalities (setting up appointments, managing line tickets and schedule all the appointments for the staff), our system will provide all of these functionalities;
- In summary, we want to include the complete workflows of searching and setting up an appointment, waiting for your ticket to be called, paying for the appointment and issue the bills and presence confirmation.



2.2 Personas and scenarios

Personas:

1. Emily Smith

Is 32 years old and suffers from a chronic condition. Therefore, Emily has to go to the hospital frequently and is looking for a way to schedule/manage their appointments via digital platforms and to check her appointment history.

Emily lives in a rural region so is often forced to drive long distances to go to the hospital, many times having to face long waiting queues,often reaching more than 12h, sometimes not even getting an appointment.

Motivation:

Emily's bad experiences with hospital check-ins and appointments are making her look for an easier and more in-hand way of scheduling her appointments while at home, through her smartphone or computer.

Emily is also looking for a way to check her registered appointments so she doesn't forget about them.

2. Tom Desk

Tom is 43 years old and works at Hospital reception. He feels constantly frustrated about his work because sometimes the reception area is full of patients and he can't do the check-in process as quickly as he'd like. He would appreciate a system that could resolve this problem and also guarantee that patients are being called by order.

Tom is also constantly getting phone calls from customers trying to make appointments, often leading to being stressed during the work days.

Motivation:

Tom is looking for a digital system that can facilitate his job, making sure that patients know when it's their turn to do the check-in process and automating the process of calling patients to appointments.

Tom would like a digital system where patients can register their appointments, without having to contact the reception directly, that is easy to manage.

3. Lisa Moose

Lisa is 51 years old and works as an administrator of a Hospital. She finds it hard to know if the current working protocols are efficient, both for patients and the hospital workers, often having to read through various files from the hospital areas.

Motivation:

Lisa would like a centralized system where she could review the hospital statistics, such as consultations or appointments done by each medic, the number of total consultations done in a day or how many patients visited the hospital that day.

Main Scenarios:

Scenario 1:

Emily wakes up feeling an unusual discomfort which she suspects could be related to her chronic condition. She uses the PhiHub to search for an available consultation with a specialist in her area. After finding a suitable time slot, she books the appointment and receives confirmation. On the day of her appointment, she uses the digital signage portal to self-check-in upon arrival at the hospital.

Scenario 2:

Tom starts his shift at the reception and logs into the PhiHub. He reviews the list of patients scheduled for the day and prepares the necessary paperwork. As patients arrive, he checks them in using the system, updating their status so that the screen at the reception area displays current waiting times and queue numbers. Tom uses the system to manage priority cases, ensuring they are attended to promptly. He also handles payment transactions and issues presence confirmations after consultations.

Scenario 3:

During the flu season, it is very common for Tom to encounter a mix of regular appointments and walk-ins. This way, Tom uses PhiHub to manage the flux of patients, prioritizing appointments, but also fitting walk-ins by viewing no-shows and last minute cancellations.

Scenario 4:

Lisa reviews the various logs from the hospital areas on PhiHub and uses it to train the hospital staff and further implement new work protocols so the hospital is able to serve more patients while ensuring the staff isn't overworked.

2.3 Project epics and priorities

Epic 1: Schedule/manage appointments

User Story #1: As Emily, I want to be able to schedule, cancel and check appointments remotely without the need for actually going to the hospital;

Epic 2: Manage patients check-ins

User Story #5: As Tom, I want to be able to register new patients quickly as to optimize the time used in checking in first time patients;

User Story #6: As Tom, I want regular patients to be able to check-in by themselves so that I can help new patients.

Epic 3: Manage administrative personnel

User Story #10: As Lisa, I want to be able to easily check the day-to-day statistics of patients that pass through the hospital, as well as the average amount of time that a patient spent waiting to be attended to.

User Story #11: As Lisa, I want to be able to quickly check how many patients per hour are being attended to, as to optimize the working hours of the staff.

Epic 4: Real-Time monitoring of waiting lines and prioritization of patients

User Story #7: As Tom, I want to check which lines are the longest and how long the estimated queue time is so that I can prioritize certain queues for a better patient experience and time management.

User Story #8: As Tom, I want the system to automatically call in a patient from the queue line with the most priority or the most wait time, as to optimize patient satisfaction;

Epic 5: Self-check-in feature via mobile when arriving at the hospital

User Story #2: As Emily, I want to be able to check-in for my consults after arriving in the hospital from my mobile phone, to circumvent having to wait in a queue for the check-in;

Epic 6: Check agenda, processed bills and presence confirmations

User Story #3: As Emily, I want to be able to check all my previous bills so I can know exactly how much money I am spending with the visits;

User Story #4: As Emily, I want to be able to check my current agenda so I will not miss my next appointments;

User Story #12: As Lisa, I want to be able to check how many bills are passed and how much money is generated by the hospital.

Epic 7: Call and manage tickets

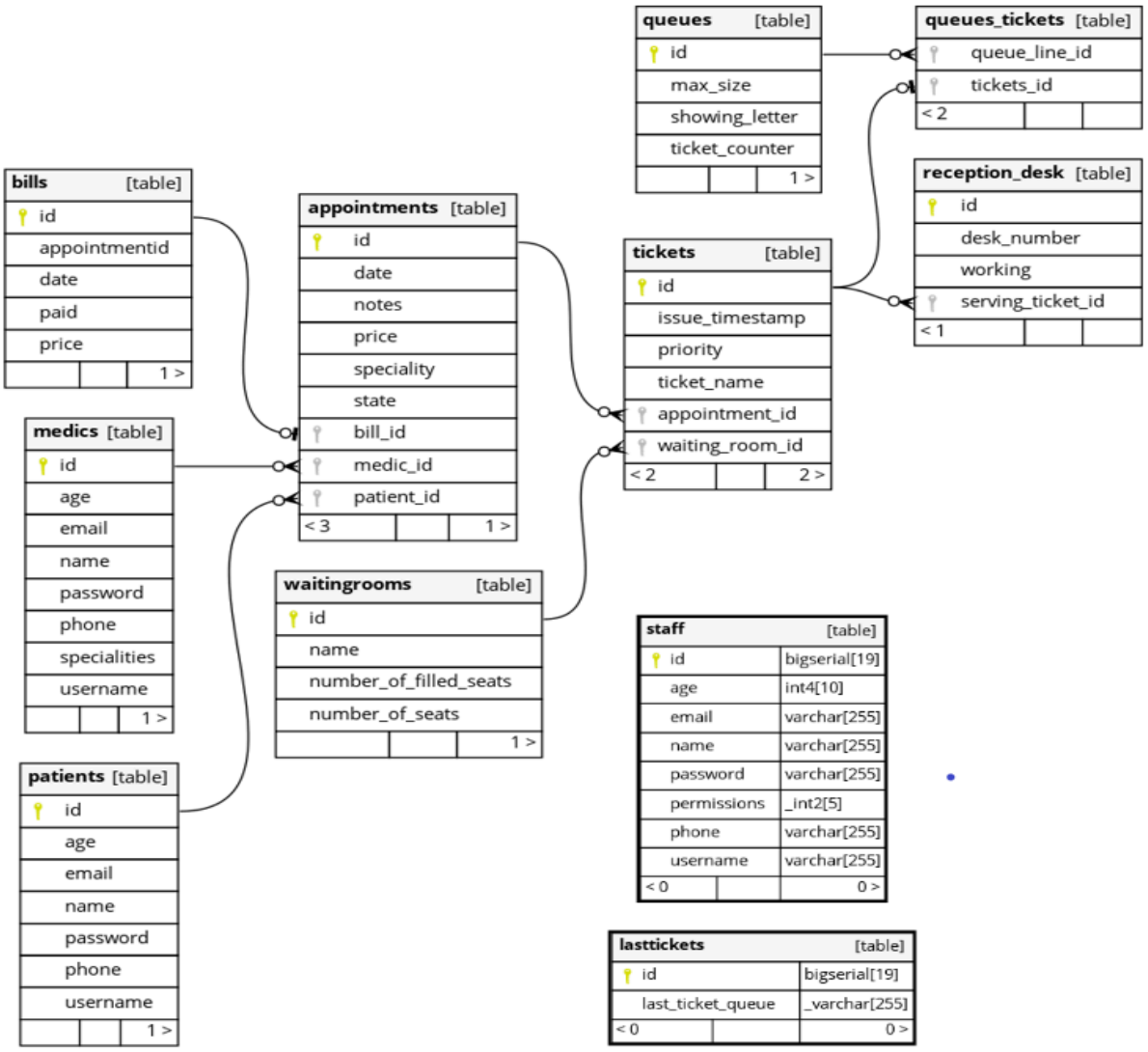
User Story #9: As Tom, I want the system to quickly show the patients which ticket is being called, as to waste as little time as possible in the calling process;

3 Domain model

In designing the database schema, we strived mainly for simplicity, while making sure that we could still fulfill all the needed requirements without a high number of queries for each of the use cases.

The domain model may be seen on the image below:

To start, there are the 3 different types of users that will use the application, **Patient, Medic, Staff**, which have similar information besides some specific attributes (permissions, specialities , …) who are unique to each of them.



Appointments can be made in the application, which involve a medic, a patient, and a **Bill** to pay.

Tickets are issued to patients on check-in so that reception desk operators can call them to the **Reception Desk .** Each ticket is also assigned to a **Waiting Room,** where the patient may wait till he/she gets called to the reception

Each ticket is assigned to a **Queue,** which represents the typical queues that are seen on hospital digital signages (A, B, C, D), which store all the tickets that belong to itself.

In order to save the last called tickets to reception, there is also a **Last Tickets** table which stores the previously called tickets.

4 Architecture notebook

4.1 Key requirements and constraints

Being a new application done from scratch, there are no constraints in terms of architecture, which means that we can design it with modern high-level tools, frameworks and programming languages.

In order to simplify the development experience and also ensure that the system may run on any deployment machine, **Docker Containers** are used, which allow us to spawn containers, fit to run the application code, on any given machine.

As the project involves different entities, which have different roles in the application, authorization and authentication must be applied for all actions, in order to guarantee that no abnormalities happen.

The system must also guarantee a good level of performance under situations of load, given its area of use (medical field).

4.2 Architecture view

For the general architectural model, the application follows a simple backend-frontend structure, without any additional middleware and modules.

For handling the general parsing of requests and paths, a NGINX service was implemented. This component has the capability of receiving the requests, parsing their metadata and finally redirecting the request to the correct module of the application.

The backend of the application is a SpringBoot application capable of handling the processing of the requests created by the frontend and managing the database system.

The use of SpringBoot allows easy integrations with test frameworks and a quick development time, as well as a simple and organized file structure.

SpringBoot also allows for the use of external libraries such as loggers, code inspector tools and in-memory databases for caching.

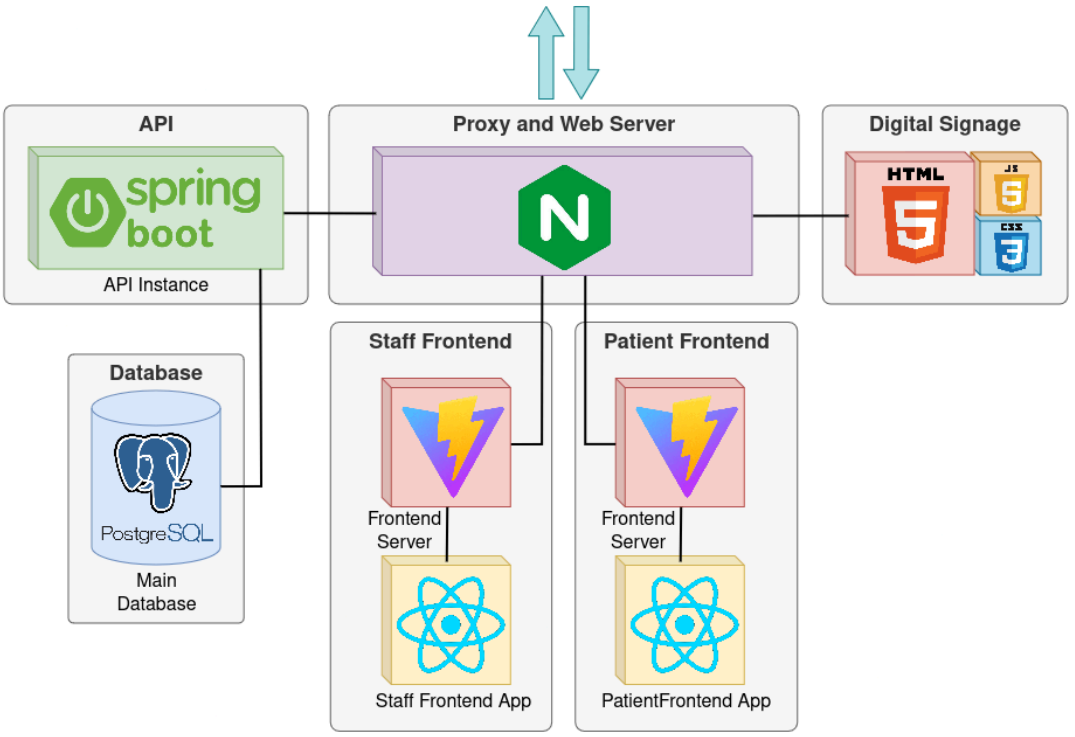
For the main database, PostgreSQL was chosen for its simple integration with SpringBoot's JPA data manipulation tool and for its Open Source nature, which allows us to get better community support in case of any problems with the implementation.

PostgreSQL also provides a comparatively small processing overhead and an easy implementation using docker containers, which cuts down on development time and on the complexity of the database configuration.

Both the staff and patient frontend applications were developed in React, using ViteJS for serving the final compiled web pages and assets.

React was chosen since it is a relatively modern and widely used framework with a wide range of libraries and online support. ViteJS is usually paired with React to compile and serve the pages, so a correct integration between both is very easy to achieve.

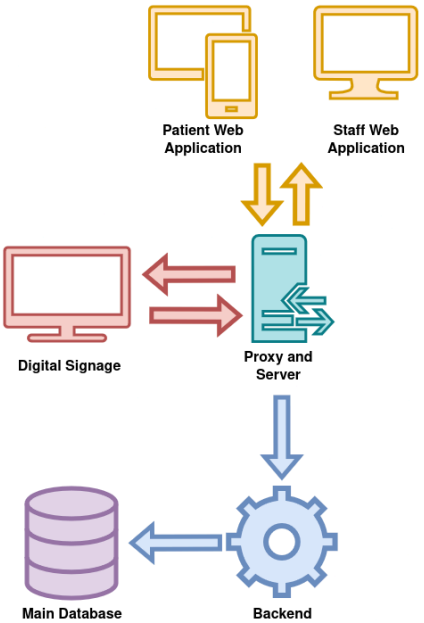
Finally, the Digital Signage was developed using a conventional HTML + CSS + JavaScript layout, to provide a quick development and a low overhead implementation, since it does not require any complex compilation or a long list of dependencies. This allows for the service itself to be extremely small and efficient, which is a requirement for a signage that could be deployed on machines with limited processing or storage capacity.



As for the interactions between modules, the application mostly utilizes HTTP requests where possible. This allows easy parsing of the data and messages in case of erroneous requests.

The Patient, Staff and Signage Applications send their HTTP requests to the NGINX service which is then transmitted to the backend.

The backend then processes and persists the data generated and responds back to the NGINX, which then sends the response back to the original sender.



4.3 Deployment architecture

For the deployment structure, a docker-compose environment was created in order to simplify and streamline the actual deployment process.

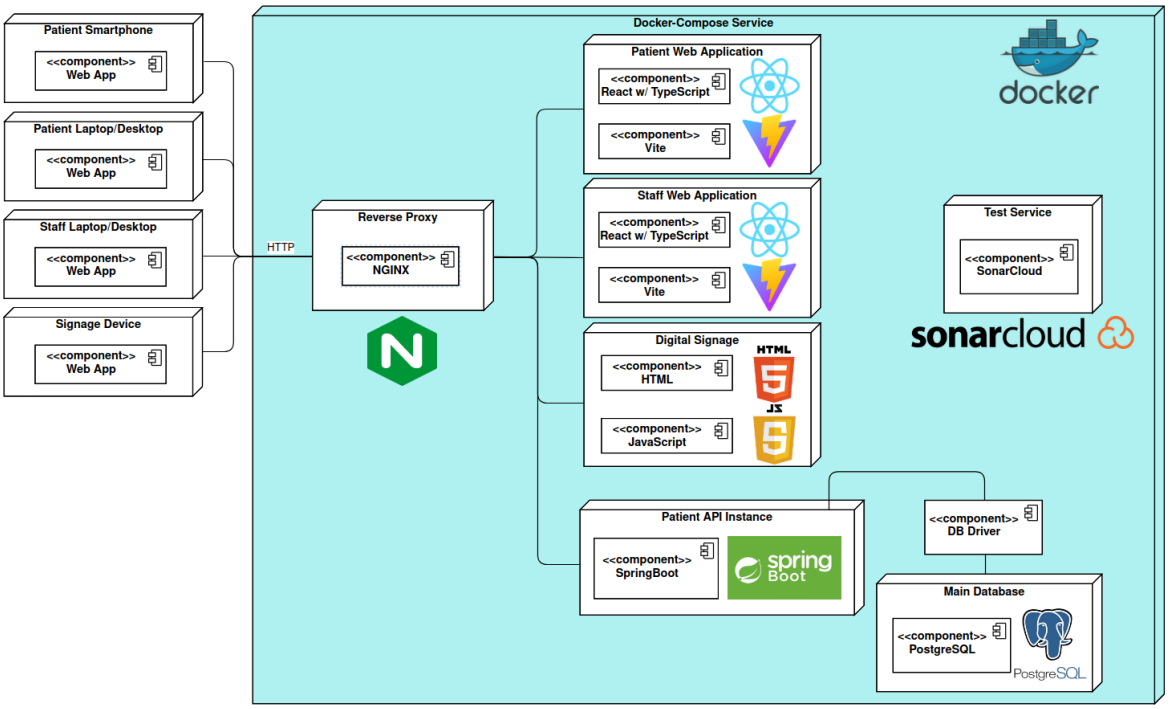
This allows the whole application to quickly change between production and development environments without the need to alter the code itself.

To accomplish this, two docker environments were created, one for production and one for development. Both are similar but contain different variables and commands.

In the diagram below, we can see that the NGINX reverse proxy is responsible for taking in the requests and responses from the various modules and distributing them to the correct service.

The Patient, Staff and Digital Signage front ends are completely separated and autonomous.

The backend is connected to the main PostgreSQL database and is responsible for all management of persisted data.



5 API for developers

Given that three different users, Patient , Medic and Staff will be utilizing the same Backend, the REST API was developed with the separation of roles in mind.
Given so, endpoint names follow the following format {role}/{object_they_act_upon}, such as /staff/appointments, which constitutes an endpoint for staff members to manipulate appointments.

API documentation was developed utilizing the **Swagger Framework,** through the use of the **SpringDoc** dependency, which automatically generates the documentation UI given a few describing annotations.

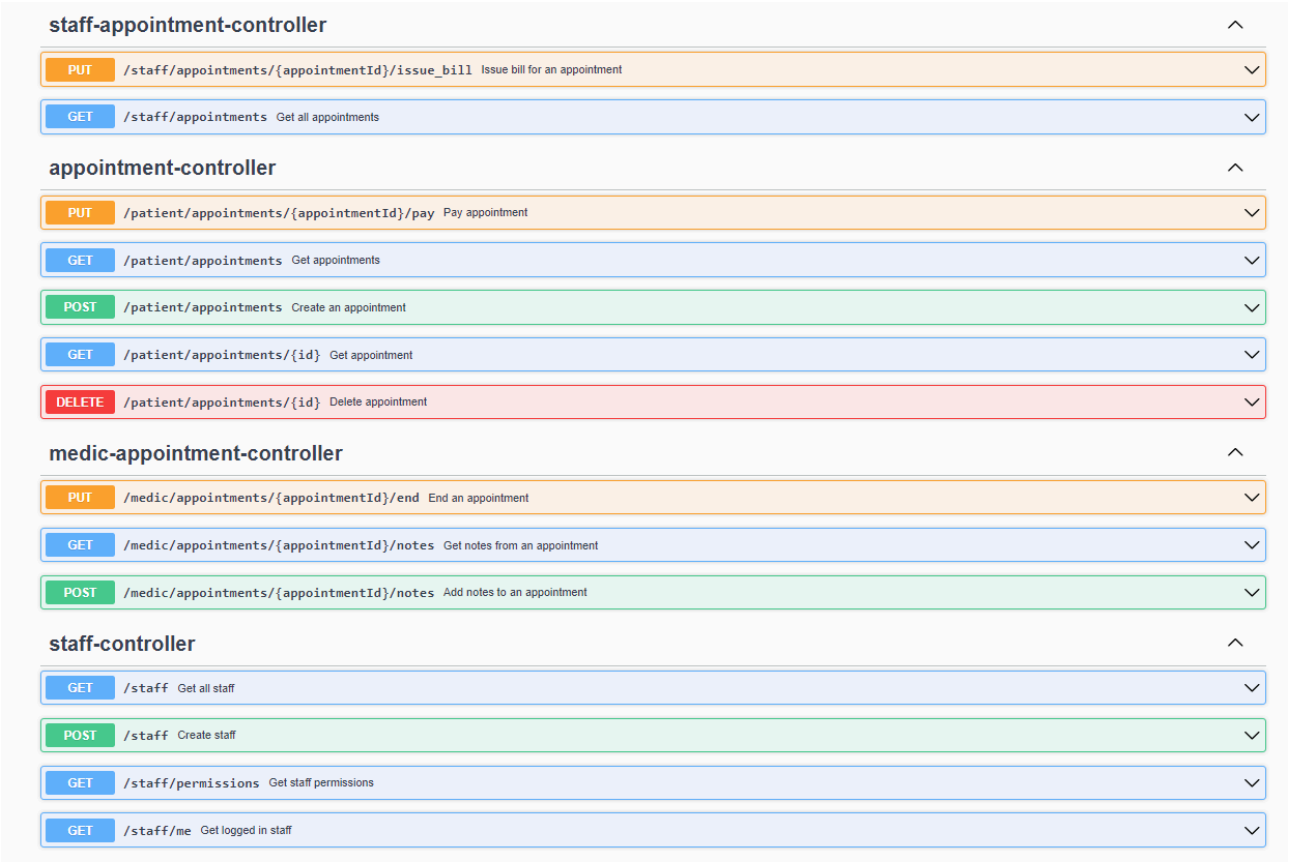
A example of a documented endpoint may be seen below:

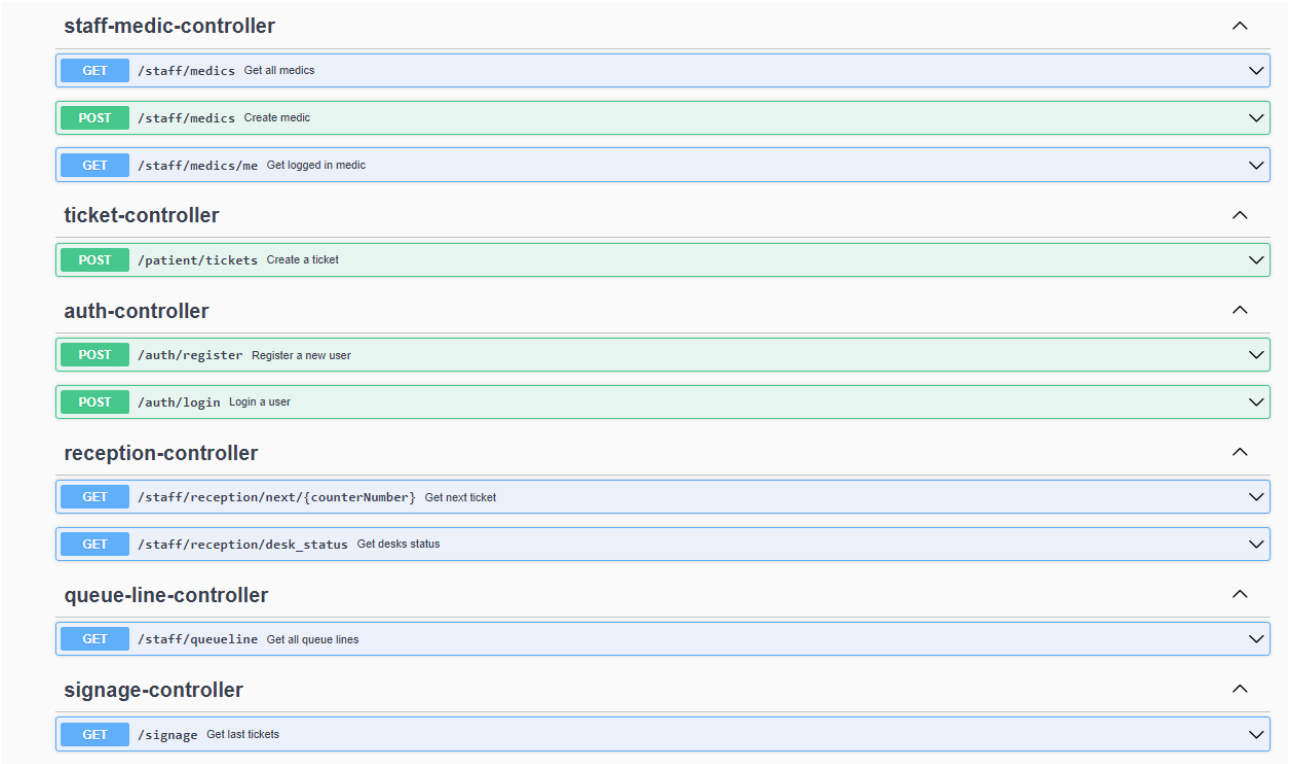
```
@Operation(summary = "Get appointments", description = "Get appointments of the logged in user")
@ApiResponses(value = {
        @ApiResponse(responseCode = "200", description = "Appointments retrieved"),
        @ApiResponse(responseCode = "401", description = "Unauthorized")
})
@GetMapping
public ResponseEntity<List<Appointment>> getAppointments() {
    var user = userService.getUserFromContext();
    List<Appointment> appointments = appointmentService.getAppointmentsByPatient(user);
    return ResponseEntity.ok(appointments);
}
```

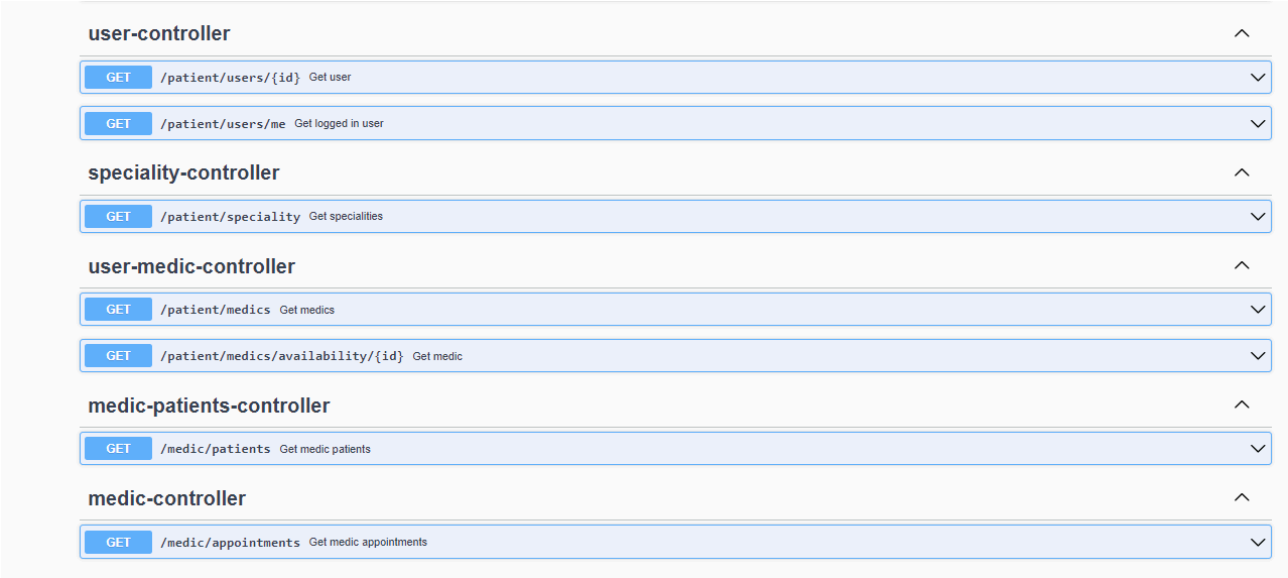
The whole API documentation may be seen:

- Running the application locally, at localhost:8080/docs
- In the production environment, at deti-tqs-18.ua.pt:8080/docs

Given the scope of the project, this user interface includes **all** endpoints with given documentation, so that future maintainers of the system can build upon the existing resources.


staff-appointment-controller
PUT /staff/appointments/{appointmentId}/issue_bill Issue bill for an appointment
GET /staff/appointments Get all appointments
appointment-controller
PUT /patient/appointments/{appointmentId}/pay Pay appointment
GET /patient/appointments Get appointments
POST /patient/appointments Create an appointment
GET /patient/appointments/{id} Get appointment
DELETE /patient/appointments/{id} Delete appointment
medic-appointment-controller
PUT /medic/appointments/{appointmentId}/end End an appointment
GET /medic/appointments/{appointmentId}/notes Get notes from an appointment
POST /medic/appointments/{appointmentId}/notes Add notes to an appointment
staff-controller
GET /staff Get all staff
POST /staff Create staff
GET /staff/permissions Get staff permissions
GET /staff/me Get logged in staff


staff-medic-controller
GET /staff/medics Get all medics
POST /staff/medics Create medic
GET /staff/medics/me Get logged in medic
ticket-controller
POST /patient/tickets Create a ticket
auth-controller
POST /auth/register Register a new user
POST /auth/login Login a user
reception-controller
GET /staff/reception/next/{counterNumber} Get next ticket
GET /staff/reception/desk_status Get desks status
queue-line-controller
GET /staff/queueline Get all queue lines
signage-controller
GET /signage Get last tickets



6 References and resources

<document the key components (e.g.: libraries, web services) or key references (e.g.: blog post) used that were really helpful and certainly would help other students pursuing a similar work>

https://tanstack.com/query/latest/docs/framework/react/overview

https://tailwindcss.com

https://vitejs.dev

https://schemaspy-org.translate.goog/?_x_tr_sl=en&_x_tr_tl=pt&_x_tr_hl=pt-PT&_x_tr_pto=sc

https://react.dev

https://spring.io/projects/spring-boot

https://www.baeldung.com/security-spring

https://www.toptal.com/spring/spring-security-tutorial