deti universidade de aveiro
departamento de eletrónica, telecomunicações e informática

TQS: Product specification report

Bernardo Pinto 105926, Filipe Obrist 107471, Liliana Ribeiro 108713, Lia Cardoso 107548
v2024-06-03

1 Introduction

1.1 Overview of the project

This project was developed as part of the Software Testing and Quality Assurance (TQS) course. The primary objective of this assignment was to design and implement a high-quality software solution, applying rigorous testing and quality assurance practices learned during the course. The project required a comprehensive approach, emphasizing code quality, functional correctness, and maintainability through the effective use of testing strategies and static code analysis.

The application, HealthConnect, is a comprehensive system designed to simplify appointment scheduling for both patients and staff members. In HealthConnect, patients can easily schedule appointments, while staff members can efficiently manage patient information, oversee the appointment queue, and advance the tokens.

1.2 Limitations

While HealthConnect is designed to be a robust solution, some limitations were identified during the project:

Payment Integration: Currently, the system lacks direct payment gateway integration. The feature is planned for future releases.
User Role Management: Although basic role-based access control is implemented, more granular role management is yet to be developed, like doctors or admins.
Multi-language Support: The application currently supports only English.

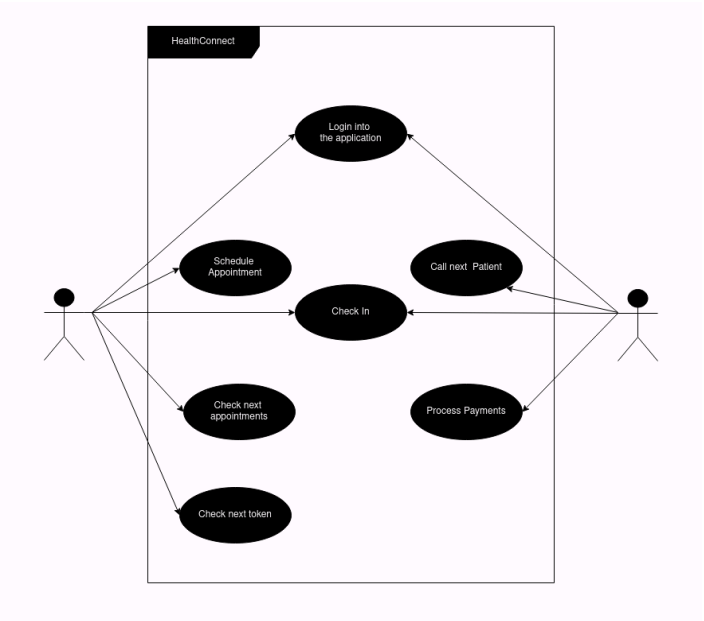
2 Product concept and requirements

2.1 Vision statement

The application, HealthConnect, is designed to be the go-to solution for healthcare facilities seeking efficient and automated appointment scheduling. The system focuses on solving the common challenges faced by both patients and staff members in booking, managing, and fulfilling medical appointments. By offering an intuitive platform for tracking appointments and managing schedules, HealthConnect reduces manual intervention, improves accuracy, and enhances patient satisfaction.

Functional Overview:

- Appointment Management: Login into account; Create and track patient appointments; Check queue status.
- Staff: Login into account; See Patient information and proceed with Patient check-in; Advance tokens in the queue; Process Payments.



2.2 Personas and scenarios

Personas

Patient:

João Medeiros, 55 years old, Software Engineer, has back problems (scoliosis), and is currently undergoing evaluation appointments for possible surgery.

To book an appointment, he currently needs to find out which health unit the specialist he wants to visit is working at, then travel to that location and schedule an appointment at the reception.

João would like to find a simpler way to complete this process, such as being able to do everything from home instead of having to travel.

As a software engineer, he has a relatively high level of technological literacy.

Staff Member:

Maria Amélia, 32 years old, Receptionist, is responsible for managing patient check-ins and administrative tasks at the hospital reception.

She needs a system that simplifies the check-in process and helps her efficiently handle patient appointments. Currently, Maria faces several obstacles and limitations, such as crowded waiting areas, manual paperwork, and difficulty in coordinating patient flow.

Maria wants to efficiently register patients, manage waiting lines, and process payments seamlessly.

Scenarios

Main Scenario #1 - Patient - Book an Appointment:

João needs to book an appointment with a specialist. He accesses the Φ-Patient portal, selects the required medical specialty, chooses a convenient time, and schedules the appointment.

Main Scenario #2 - Staff Member - Patient Check-in:

Maria, a hospital receptionist, is at the registration desk during the morning rush hour. She uses Φ-Desk to process patient check-ins arriving for their appointments. With just a few clicks, she verifies patient details, confirms the appointment, and processes the payment, ensuring a quick and efficient process.

Patient - Check-in:

João arrives at the hospital for his appointment. He opts to check-in using the mobile app, avoiding the reception to save time.

Patient - Book an Appointment:

João is at home, and his back pain reminds him that he needs to book an appointment with a specialist. To do so, he must select the required medical specialty, choose a convenient time, and schedule the appointment.

Patient - View Schedule:

João is busy with his projects and forgot when his appointment is. To refresh his memory, João checks his schedule.

Patient - After Check-in:

After checking in, João looks at the screen to compare his ticket number with the most recently called number. Seeing there are still many numbers in between, he looks for a place to sit.

Staff Member - Confirm Patient Information:

Maria wants to confirm the information of a patient who has booked an appointment through the system to ensure there are no errors.

Staff Member - Call Next Patient:

Maria wants to call the next patient in the waiting line for check-in, advancing to the next ticket number in the system.

Staff Member & Patient - Payment:

João wants to pay for his appointment, either by card or cash. Maria processes the transaction through the system, recording the amount in the system if paid in cash, or marking the transaction as paid in the system if done through a card machine.

Send Reminder Emails:

At the end of the day, the application sends reminder emails to patients who have appointments the day before.

Display Panel - Patient Observes Ticket Number:

A patient enters the hospital, takes a ticket, and looks at the display panel to see which ticket number is currently being called.

Display Panel - Staff Member Advances Ticket:

A staff member chooses the "Advance Ticket" option, and the panel displays the next ticket number.

2.3 Project epics and priorities

Epics

Epic 1: Patient Appointment Management

Book Appointment: Patients can book an appointment with a specialist online.
View Schedule: Patients can view their appointment schedule.

Epic 2: Staff Management and Patient Check-in

Manage Patient Information: Staff members can view and edit appointment information.
Patient Check-in: Staff members can check-in patients efficiently.
Advance Ticket Number: Staff members can advance to the next ticket number.

Epic 3: Billing and Payments

Process Payments: Staff members can process payments via cash or card, outside the app.

Epic 4: Notification System

Send Reminder Emails: The app automatically send reminder emails to patients.

Epic 5: Display Panel System

Display Queue Status: Patients can see the current ticket number being served.

Priorities

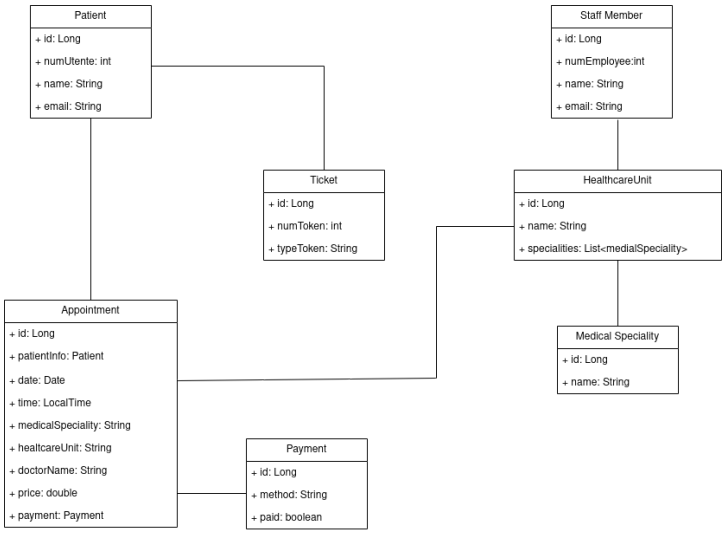
High Priority (Core Requirements):

1. Book Appointment
2. Manage Patient Information
3. Patient Check-in
4. Process Payments
5. Advance Ticket Number

Medium Priority:

1. View Schedule
2. Send Reminder Emails
3. Display Queue Status

3 Domain model



4 Architecture notebook

4.1 Key requirements and constrains

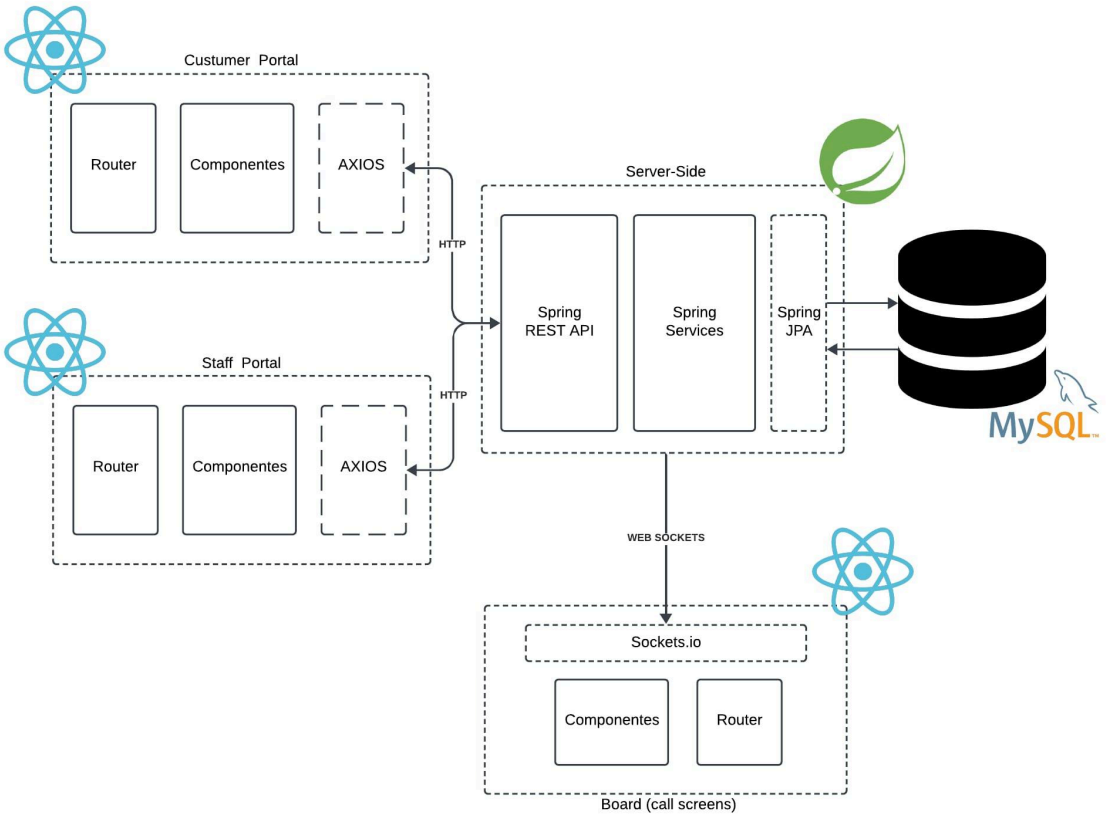
There are some key requirements and system constraints that have a significant bearing on the architecture. They are:

- The HealthConnect-Patient portal must ensure seamless integration with existing legacy systems, including those managing patient records and appointment scheduling. Ensuring compatibility with the data formats and DBMS of these legacy systems is paramount.
- It must support the interface with legacy billing systems, adhering to predefined billing interface specifications, facilitating patient payments smoothly.
- All patient and staff functionalities must be available through web-based access. Future enhancements will consider support for mobile devices and large display screens, ensuring accessibility across various user interfaces and enhancing user experience.
- The HealthConnect-Patient portal must ensure complete protection of data from unauthorized access. Furthermore, all sensitive data should be encrypted, and the system should comply with relevant data protection regulations, such as GDPR and HIPAA, ensuring data integrity and privacy.
- The system must be designed to handle multiple concurrent users efficiently, particularly during peak usage periods such as morning check-ins. Real-time updates for appointment schedules and waiting line management ensure that patients and staff have access to the most current information, enhancing operational efficiency.
- The system should be implemented as a client-server architecture. The client applications, encompassing the customer and staff portals, must interact seamlessly with the server-side components, including the Spring REST API, Spring Services, and Spring JPA. This design leverages a scalable infrastructure, ensuring the backend can accommodate varying loads and maintain high availability.
- Efficient and secure data management must be achieved through the use of Spring JPA and MySQL, which support complex queries and transactions essential for appointment scheduling and patient management.
- Real-time communication must be facilitated through WebSockets, providing timely updates to both patients and staff, such as current ticket numbers displayed on panels.

- The modular design of the architecture, with clearly defined components and services, ensures that the system is maintainable and can be easily extended with new features.

4.2 Architecture view

The architecture of our software solution is designed to be modular and scalable, we used modern web technologies and frameworks to provide a seamless experience for users.
The main components of the system are the backend server, the three frontend portals: staff, customer and board, and a database. The diagram below illustrates the high-level architecture:



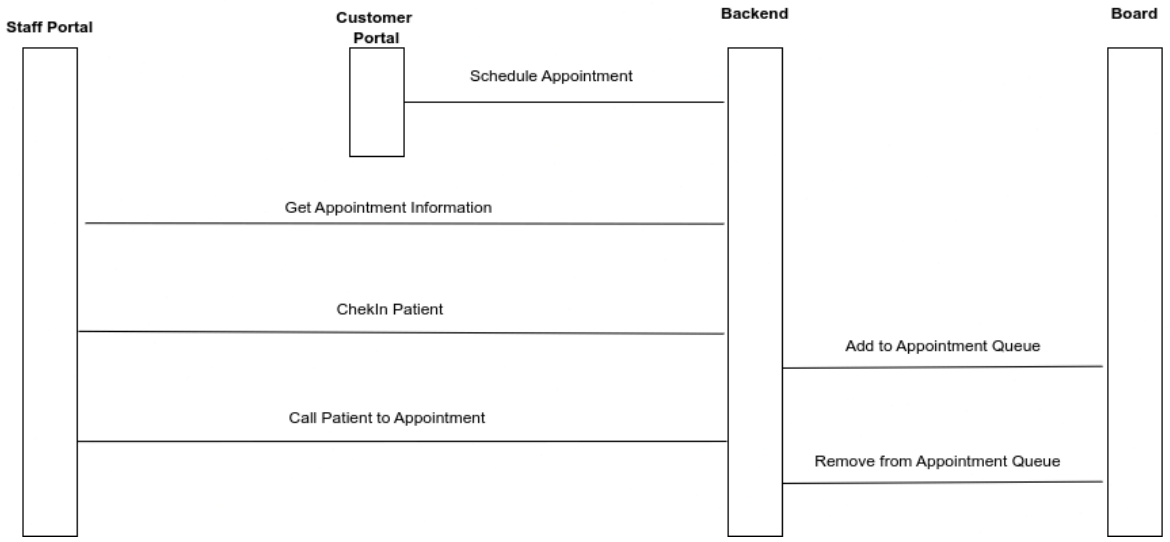
The Customer and Staff Portals are built with React and interact with the backend via HTTP using Axios for API calls. These portals handle the customer and staff operations such as booking appointments or managing appointments, respectively.

The Board (Calls Screen) uses React and Socket.io to display real-time updates on appointment status. It communicates with the backend through WebSockets for instant updates.

The Backend Service was constructed using Spring REST API; it serves as the main entry point for all HTTP requests from the frontend portals. It handles business logic and communicates with the database. Additionally we have the Spring Services that encapsulate the core business logic and interact with the JPA layer to perform CRUD operations. The Spring JPA Manages database interactions, ensuring data persistence and retrieval.

For the DataBase we used MySQL that stores all application data, including user information and appointments. It ensures data integrity and supports complex queries.

In the diagram below is a high representation of the interactions along time of our application.



In our application, we have integrated an email service to enhance user experience. This service sends automated reminder emails to patients the day before their appointment. The email service is integrated with our backend, triggering notifications based on scheduled appointment data, ensuring patients are always reminded of their upcoming consultations.

4.3 Deployment

For the deployment, we used Docker and a container-based approach, as identified in the architecture figure. Additionally, we utilized the VM provided by the teacher.

5 API for developers

Our API offers several services and resources that developers use. Below is a overview of these services:

Appointment Controller:

PUT /appointments/{appointmentId}/setPatient/{patientId}: Assigns a patient to an appointment.
PUT /appointment/setPaymentDone: Marks the payment for an appointment as done.
PUT /appointment/setCheckInDone: Marks the check-in for an appointment as done.
GET /appointments/withoutPatient: Retrieves appointments that do not have an assigned patient.
GET /appointments/patient/{patientId}: Retrieves appointments for a specific patient.
GET /appointments/available-times/{date}: Retrieves available appointment times for a specific date.
GET /appointment: Retrieves all appointments.

Ticket Controller:

POST /newTicket: Creates a new ticket.
POST /newAppointmentTicket: Creates a new appointment ticket.
POST /clearCheckIn: Clears the check-in status.
POST /clearAppointments: Clears all appointments.
GET /nextCheckInTicket: Retrieves the next check-in ticket.
GET /nextAppointmentTicket: Retrieves the next appointment ticket.
GET /called: Retrieves all called tickets.

GET /CheckInNotCalled: Retrieves all check-ins that have not been called.
GET /AppointmentsNotCalled: Retrieves all appointments that have not been called.

The detailed documentation of these methods, including request and response structures, parameters, and usage examples, is provided through Swagger.

The API documentation can be seen in /swagger-ui/index.html