TQS: Product specification report

Gonçalo Lopes [107572], Rodrigo Graça [107634], Tiago Cruz [108615]

1 Introduction

1.1 Overview of the project

This project, designed as part of the Total Quality Software (TQS) course, aims to develop a digital solution for the New Private Health Initiative (MedConnect), a hypothetical healthcare system. The primary objective is to design and implement an integrated IT solution that streamlines patient appointments and admissions while excluding clinical records for now. This IT solution enhances patient and staff experience by providing efficient, robust, and user-friendly digital services. It is developed to handle pre-appointment scheduling, reception services, and post-encounter processes like billing and presence confirmation, focusing on offering competitive services by improving accessibility and operational efficiency in healthcare settings.

1.2 Limitations

While MedConnect aims to be comprehensive, certain limitations are inherent in the current scope of the project:

Clinical Records Integration: At this stage, integration with existing clinical records systems is not included. This functionality is essential for a complete healthcare management system and is planned for future phases.
Mobile Application: The mobile version of the Patient portal, which would allow for greater accessibility and convenience, is also planned but not yet implemented.
Advanced Analytics: Use of data analytics for optimizing hospital operations and patient flow is considered but currently outside the project scope.

2 Product concept and requirements

2.1 Vision statement

MedConnect is designed to revolutionize the way patients interact with healthcare facilities by providing a streamlined, intuitive digital platform for scheduling and managing healthcare appointments. This system addresses the high-level business problem of inefficient healthcare management, where patients struggle with long wait times, complex booking procedures, and inefficient administrative processes. By consolidating patient management, administrative tasks, and digital display systems into a unified platform, MedConnect ensures a smoother operational flow and enhanced patient care experience

2.2 Personas and scenarios

Personas

- **Persona 1: Emma (Patient)**
  - **Demographics:** Age 32, employed, tech-savvy.
  - **Behaviors:** Prefers online scheduling, uses mobile apps for health management.
  - **Goals:** Quickly find and book appointments, easy check-in, minimal waiting times.
- **Persona 2: Dr. Harris (Healthcare Provider)**
  - **Demographics:** Age 45, general practitioner.
  - **Behaviors:** Needs efficient tools to manage patient flow.
  - **Goals:** Reduce administrative tasks, seamless access to patient appointment schedules.
- **Persona 3: Linda (Reception Staff)**
  - **Demographics:** Age 38, familiar with basic IT tools.
  - **Behaviors:** Manages patient registration and billing.
  - **Goals:** Streamline patient check-ins, update waiting list.

Main Scenarios

Here are the key scenarios that the system will need to support:

- **Scenario 1: Booking an Appointment (Emma)**

    - Emma logs into *Patient*, searches for a dermatologist at the nearest hospital, views available slots, and books an appointment.
- **Scenario 2: Patient Check-in (Linda)**
    - Linda uses *Staff* to check in arriving patients, verifies their details, and updates their status in the system.
- **Scenario 3: Calling Patients to Consultation (Dr. Harris)**
    - Dr. Harris uses *Staff* to view his next patients to be called.

2.3 Project epics and priorities

The development process is organized into three primary epics, each encompassing a broad set of functionalities critical to the overall system. The implementation is planned incrementally over several iterations or releases, focusing on delivering essential features early and refining or expanding the capabilities in subsequent phases. Below is an indicative plan for the incremental implementation of these epics, highlighting the functionalities to be achieved in each.

Epic 1: Patient Management

Objective: Enhance patient experience by providing efficient tools for managing their healthcare interactions.

Priorities and Incremental Implementation:

Phase 1 (Initial Release):
Develop basic patient profile management, including registration and data update functionalities.
Implement core appointment booking functionality with basic search filters (e.g., by specialty, date).

Phase 2:
Enable access to and management of personal care files, allowing patients to view their medical history and past appointment details.
Integrate patient feedback mechanisms directly into the care file access features.

Epic 2: Administrative Interface

Objective: Streamline administrative processes to improve efficiency at the reception desk and reduce patient wait times.

Priorities and Incremental Implementation:

Phase 1 (Initial Release):
Set up basic patient registration and check-in at the reception.

Phase 2:
Enhance the administrative dashboard with real-time queue management tools.

Phase 3:
Integrate administrative tools with the patient management system for seamless data flow and improved data accuracy.
Add reporting capabilities for administrative tasks to aid in decision-making and operational efficiency.

Epic 3: Information Display System

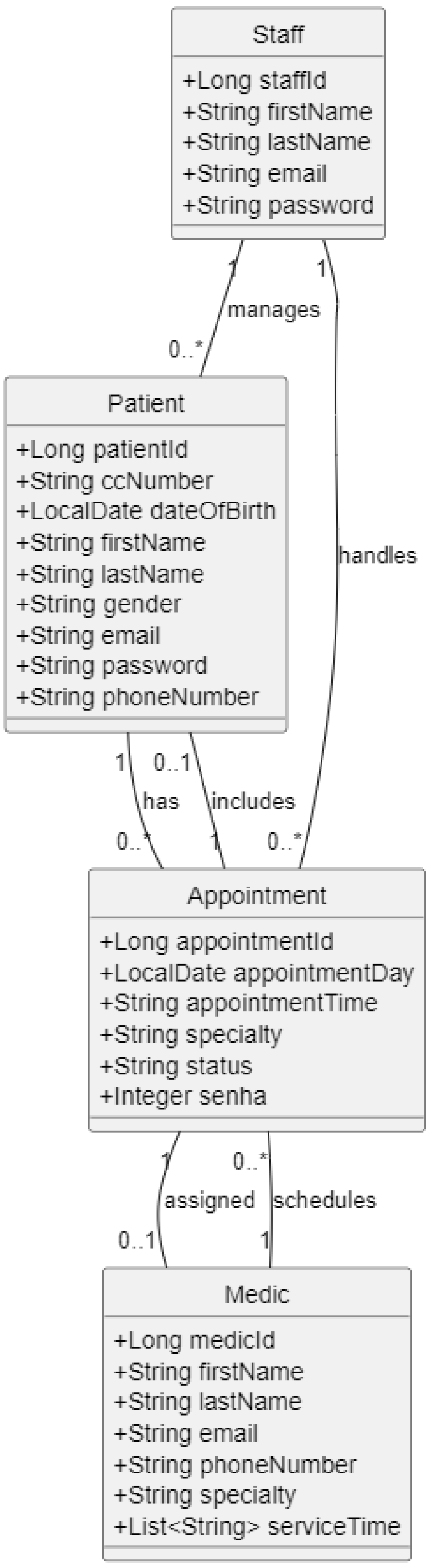
Objective: Provide real-time information to both staff and patients through digital displays, improving communication and operational transparency.

Priorities and Incremental Implementation:

Phase 1:
Develop basic digital signage to display patient calls and general queue information.

3 Domain model

The domain model manages information about patients, medics, staff, and appointments. Patients are linked to appointments, which are scheduled and managed by medics, while staff members handle administrative tasks such as managing patient records and overseeing appointment logistics.


Staff
+Long staffId
+String firstName
+String lastName
+String email
+String password
1
1
manages
0..*
handles
Patient
+Long patientId
+String ccNumber
+LocalDate dateOfBirth
+String firstName
+String lastName
+String gender
+String email
+String password
+String phoneNumber
1
0..1
has
includes
0..*
1
0..*
Appointment
+Long appointmentId
+LocalDate appointmentDay
+String appointmentTime
+String specialty
+String status
+Integer senha
1
0..*
assigned
schedules
0..1
1
Medic
+Long medicId
+String firstName
+String lastName
+String email
+String phoneNumber
+String specialty
+List<String> serviceTime

4 Architecture notebook

4.1 Key requirements and constrains

The MedConnect system is designed to streamline patient appointments and admissions while excluding clinical records for now. Several key requirements and constraints significantly impact the architecture and design of the system:

Integration with Existing Systems

1. **Legacy Systems**:
    - **Patient Records**: The MedConnect system must interface with the existing legacy patient records system to retrieve and update patient information. This interface should support the existing data formats and database management systems used by the legacy system.
    - **Billing System**: Integration with the existing billing system is essential to handle billing for medical services. This interface is defined by the Billing Interface Specification and must be adhered to ensure seamless operation.

Performance Requirements

2. **System Performance and Load Handling**:
    - **Concurrent Users**: The system must efficiently handle multiple concurrent users, including doctors, patients, and administrative staff, without significant degradation in performance.
    - **Data Retrieval**: Rapid retrieval of patient and appointment data is crucial to ensure a smooth user experience, especially in a healthcare setting where timely access to information can be critical.

Security Requirements

3. **Data Security and Privacy**:
    - **Authorization and Authentication**: The system must implement robust authentication mechanisms to ensure that only authorized users can access sensitive medical data. This includes user identification and password control for all remote accesses.
    - **Data Protection**: All patient and medical data must be protected against unauthorized access and breaches. This involves implementing encryption for data at rest and in transit, as well as ensuring secure access protocols.

User Platform Requirements

4. **Multi-Platform Support**:
    - **Web and Mobile Access**: The system must be accessible via web browsers on both desktop and mobile devices. This ensures that users can access the system from various environments, including on-the-go scenarios.
    - **Responsive Design**: The user interface should be responsive to provide an optimal experience on different screen sizes, from mobile phones to large desktop monitors.

Deployment Considerations

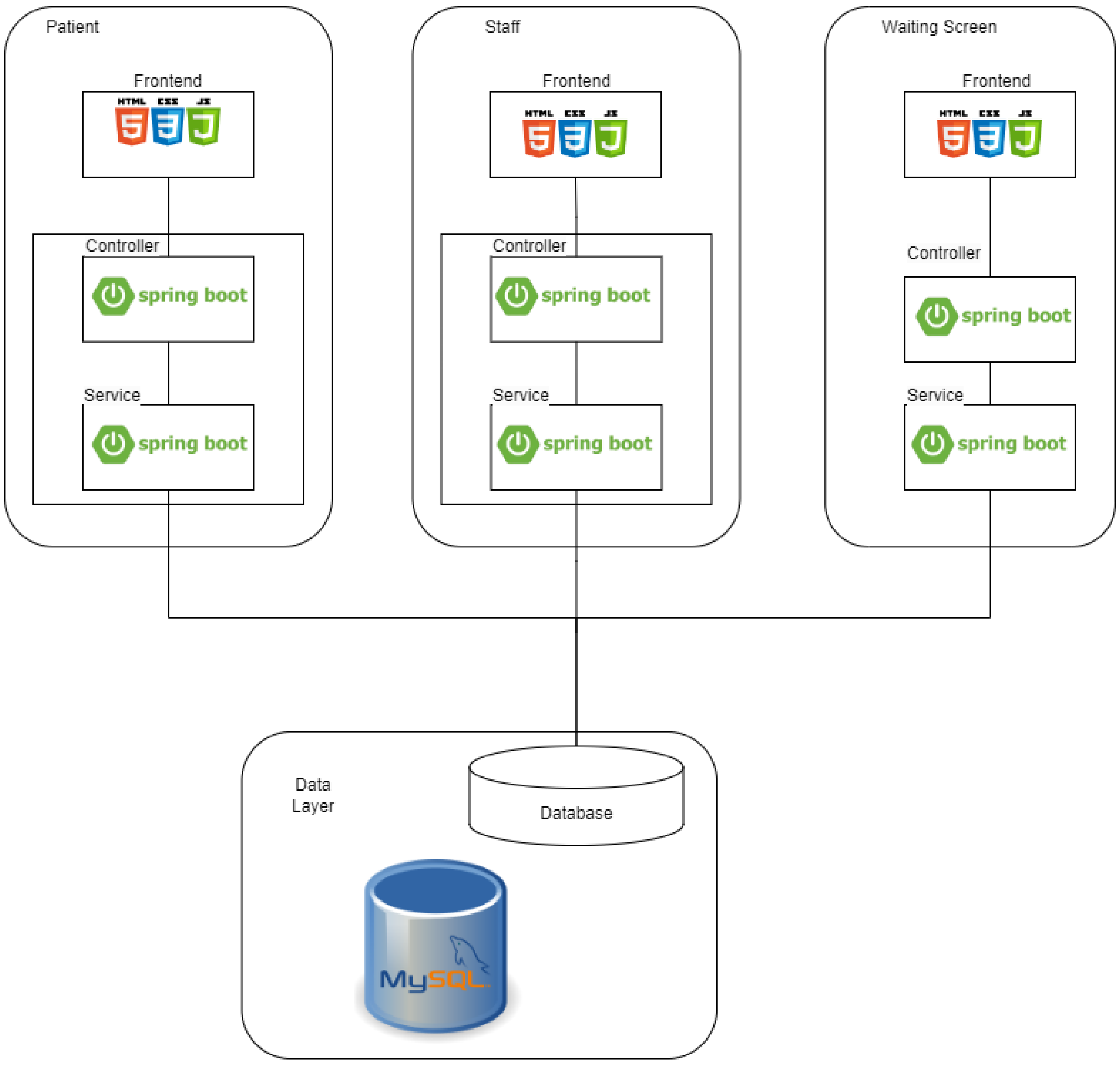
5. **Deployment Architecture**:
    - **Client-Server Model**: The MedConnect system will be implemented using a client-server architecture. The client portion, accessible via web browsers, interacts with the server portion, which processes and stores data.
    - **Scalability**: The server infrastructure must be scalable to accommodate increasing numbers of users and data over time. This may involve the use of cloud services to provide elasticity and high availability.

Additional Constraints

6. **Regulatory Compliance**:
    - **Healthcare Regulations**: The system must comply with healthcare regulations such as HIPAA (Health Insurance Portability and Accountability Act) to ensure the confidentiality, integrity, and availability of patient data.
    - **Data Retention and Auditing**: There must be mechanisms in place for data retention and auditing to meet legal and regulatory requirements for medical records.
7. **Robustness and Maintenance**:

- **Long-Term Maintenance**: The system must be designed with maintainability in mind to facilitate long-term support and upgrades. This includes using modular design principles and thorough documentation.
- **Error Handling and Recovery**: Robust error handling and data recovery mechanisms must be implemented to ensure the system can recover gracefully from failures and continue to operate under adverse conditions.

4.2 Architecture view



- **Patient (Patient WebApp)**:
  - The patient interacts with the web application to book appointments.
  - The client app communicates with the server through the controller, which handles HTTP requests and interacts with the service layer for business logic.
  - The service layer accesses the database to fetch or update patient information.
- **Staff (Staff WebApp)**:
  - Staff members use this application to check patient appointments, schedule appointments, and handle administrative tasks.
  - Similar to the client app, the staff web application uses Spring's controller and service components to process requests and interact with the database.
- **Display (Waiting Screen)**:

  - Used to display real-time information such as appointment statuses and patient queues.
  - The display app uses RabbitMQ for real-time messaging and updates, ensuring that the information shown is current.
- **Database (MySQL)**:
  - All applications interact with a centralized MySQL database to store and retrieve data related to patients, appointments and staff.

4.3 Deployment architecture

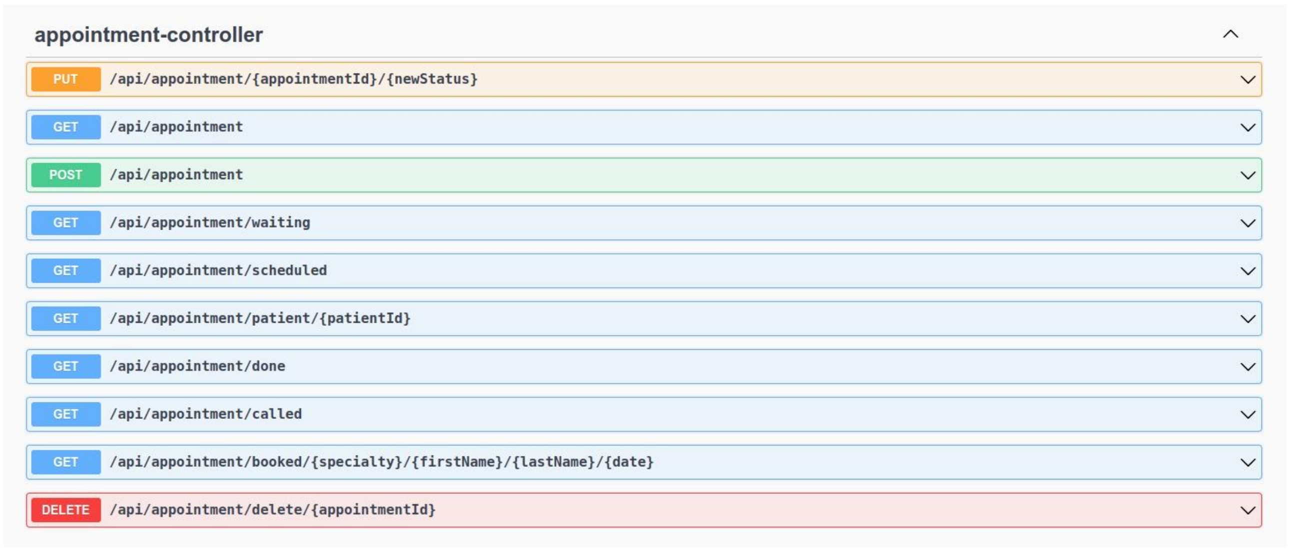
System Architecture
For the MedConnect system, consider a three-tier architecture consisting of the presentation layer, the business logic layer, and the data access layer.

- **Presentation Layer:** This includes the *Patient* and *Staff* interfaces and digital signage (*Boards*). We will use html as framework.
- **Business Logic Layer:** This layer will handle all the business rules and processes. We will be using Spring Boot to manage the logic.
- **Data Access Layer:** Here, data interactions are handled. We will be using MySQL database.
- **Integration Layer:** Using RESTful APIs for communication between the front-end and the backend.

5 API for developers

Appointment

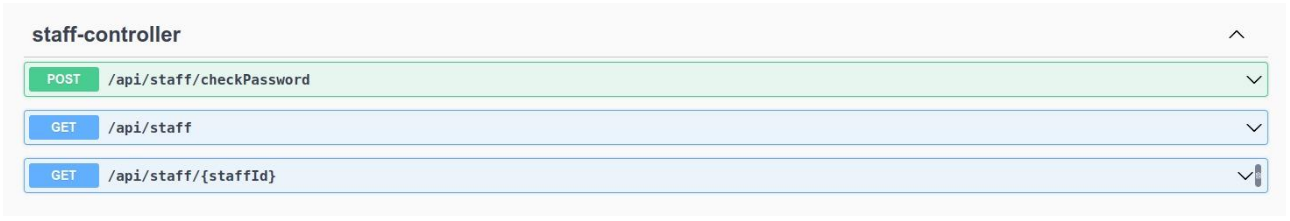
- **Create, Update, and Delete Appointments**: Developers can create new appointments, update the status of existing ones, and delete appointments using endpoints.
- **Retrieve Appointment Information**: The API allows retrieval of all appointments, specific appointments by patient ID, and appointments based on status (waiting, scheduled, done, called)
- **Booked Appointments**: Developers can fetch booked appointments for a specific specialty and date


appointment-controller

```
PUT     /api/appointment/{appointmentId}/{newStatus}
GET     /api/appointment
POST    /api/appointment
GET     /api/appointment/waiting
GET     /api/appointment/scheduled
GET     /api/appointment/patient/{patientId}
GET     /api/appointment/done
GET     /api/appointment/called
GET     /api/appointment/booked/{specialty}/{firstName}/{lastName}/{date}
DELETE  /api/appointment/delete/{appointmentId}
```

Staff

- **Staff Information**: Retrieve information about staff members
- **Staff Authentication**: Verify staff credentials

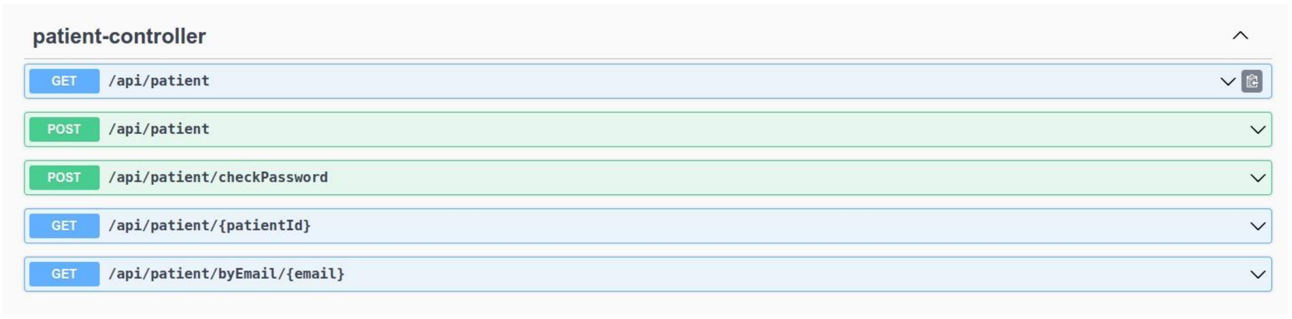


staff-controller

```
POST /api/staff/checkPassword
GET  /api/staff
GET  /api/staff/{staffId}
```

Patient

- **Patient Information Retrieval**: provide detailed information about patients
- **Patient Authentication**: endpoint is used to verify patient credentials

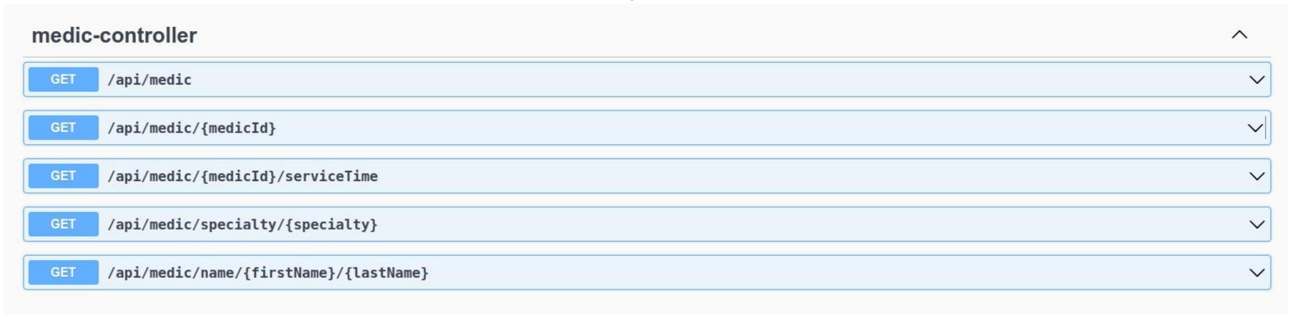


patient-controller

```
GET  /api/patient
POST /api/patient
POST /api/patient/checkPassword
GET  /api/patient/{patientId}
GET  /api/patient/byEmail/{email}
```

Medic

- **Medic Details and Specialties**: Fetch detailed information about medics and their specialties
- **Service Time**: Retrieve the service time for specific medics



medic-controller

```
GET /api/medic
GET /api/medic/{medicId}
GET /api/medic/{medicId}/serviceTime
GET /api/medic/specialty/{specialty}
GET /api/medic/name/{firstName}/{lastName}
```

6 References and resources

Libraries and Frameworks

1. **Spring Boot**: Used for building the backend services. Spring Boot simplifies the development of stand-alone, production-grade Spring-based applications by providing defaults for code and annotation configuration to quickly set up a Spring application. Documentation: Spring Boot Documentation
2. **MySQL**: A popular open-source relational database management system. MySQL is used as the primary database for storing application data in the MedConnect project. Documentation: MySQL Documentation
3. **JUnit & Mockito**: These are used for testing purposes. JUnit is a framework for writing and running tests, while Mockito is used for mocking dependencies in unit tests.

Web Services and Tools

1. **Swagger**: An open-source software framework backed by a large ecosystem of tools that helps developers design, build, document, and consume RESTful web services. Swagger is used for API documentation.