HealthHub: A Healthcare Data Management System

Paolo Palumbo, Francesco Panattoni, Nedal Hadam

June 17, 2025

Contents

Chapter 1

Introduction

Healthhub is a web-based platform designed to simplify and centralize the management of medical appointments. It offers dedicated features for both patients and healthcare professionals. Regular users can search for doctors, book visits, and leave reviews, while medical professionals can manage the services they offer and track their appointments.

- Allow users to search for medical professionals and retrieve information on their services
- Allow users to book appointments and review doctors after a visit
- Allow users to manage their appointment history and cancel or modify bookings
- Give users recommendations on doctors based on their location and past activity
- Allow doctors to specify their specializations, availability, and offered services
- Allow doctors to view and manage their upcoming appointments

Chapter 2

Dataset and Web Scraping

To populate the dataset for our application, we used a combination of web scraping, synthetic data generation, and logical inference based on real-world patterns of user behavior.

2.1 Web Scraping

The initial data was scraped from a platform containing medical reviews, MioDottore[1]. This platform provides a comprehensive database of medical professionals, including their specializations, locations, and user reviews. The scraping process involved extracting the following information:

- Doctor names and specializations
- Locations (cities and regions)
- User reviews and ratings
- Contact information (where available)
- Service offerings (e.g., types of consultations, treatments)

2.2 Synthetic Data Generation

To enhance the dataset, we generated synthetic data to simulate a more realistic user base and appointment history. This involved several steps:

- **Unique Users**: We extracted all unique usernames from the scraped reviews. For each user, we generated a detailed profile including demographic and geographical information.
- **Appointments**: For every review, we created a corresponding medical appointment. These were dated in the weeks preceding the review, to simulate a realistic flow where users leave feedback shortly after being treated.

[1] https://www.miodottore.it/

- **Likes**: To simulate engagement features such as likes on reviews, we analyzed the provinces of the doctors each user had interacted with. We then randomly selected other doctors in the same provinces and associated a number of likes comparable to the number of reviews written by each user.

2.3 Dataset Characteristics: Velocity and Variety

2.3.1 Velocity

The application context is inherently dynamic, and our dataset reflects this through the following characteristics:

- **High review frequency**: The dataset reflects an average of at least 100 new reviews per day, simulating the continuous flow of patient feedback that a live system would experience.
- **Growing doctor base**: Based on trends from medical boards, we estimate around 400 to 500 new doctors would register on the platform annually, contributing to the system's dynamism and evolving content.

2.3.2 Variety

The dataset incorporates diverse data types, which contribute to its heterogeneity:

- **Multi-format records**: Structured user and doctor profiles, unstructured textual reviews, and timestamped interaction logs (likes, appointments) are all represented.
- **Diverse entities**: The inclusion of users, doctors, reviews, likes, and appointments enables multi-relational analysis and feature richness.

2.4 Resulting Dataset

The original scraped dataset, stored in `scraped.json` (265 MB), contains the raw information collected from the web. This dataset served as the foundation for the data generation process, providing:

- 699,987 reviews
- 214,682 unique reviewers
- 87,632 doctors

Building upon this, the final dataset is stored in JSON format, totaling approximately 960 MB, and includes both real and synthetic data entries. Its main components are:

- **Doctors** (`doctors.json`, 289 MB) Contains 87,632 healthcare professionals, each with profile data and linked to 699,987 reviews. Reviews include timestamps and patient feedback.
- **Users** (`users.json`, 66 MB) Comprises 214,682 unique users extracted from the original dataset and extended with synthetic profiles.
- **Appointments** (`appointments.json`, 422 MB) Each review is connected to a synthetic appointment, scheduled in the weeks preceding the review date. Appointments simulate realistic scheduling and clinic visit patterns.
- **Templates** (`templates.json`, 162 MB) Stores template structures for appointment scheduling logic, such as available time slots, weekdays, and timing constraints.
- **User Likes** (`user_likes.json`, 22 MB) Represents user interactions with doctors, generated according to the geographic distribution of the doctors reviewed by each user.

Overall, this dataset offers a realistic simulation of user and doctor activity on a healthcare review platform, reflecting both volume and behavioral complexity.

Chapter 3

Design

3.1 Actors

The application involves three primary categories of actors, each with distinct roles and permissions:

- *Non-Authenticated User (Guest User):* refers to an anonymous individual who accesses the application without logging in. This actor is allowed to register, authenticate using existing credentials, or explore the platform by searching for doctors and viewing their public profiles.
- *Patient (User):* represents the end-user of the service. Once authenticated, the patient can book appointments with doctors, confirm or cancel them, and subsequently provide feedback through reviews.
- *Doctor:* a professional figure who offers medical appointments. The doctor can manage their availability schedule, making it accessible to patients, and consult an analytics dashboard displaying aggregated data such as earnings, reviews, and the number of visits performed.

3.2 Functional Requirements

The following section outlines the functional requirements.

3.2.1 Guest Users

Guest users are allowed to:

- Register for an account within the application;
- Log in using their credentials;
- Initiate the password recovery process in case of forgotten credentials;
- Search for doctors and access their public profiles.

3.2.2 Patients

Authenticated users identified as **patients** are granted the following capabilities:

- Manage and update their personal profiles;
- Search for doctors and access their public profiles;
- Book appointments and optionally include a note for the doctor;
- Endorse a doctor as a form of support or recommendation;
- Submit reviews regarding their medical experience;
- View their appointments, including past, current, and upcoming ones;
- Search for doctors through the recommendation feature provided by the system.

3.2.3 Doctors

Authenticated users identified as **doctors** are provided with the following functionalities:

- Manage and update their personal and professional profiles;
- Search for other doctors and access their public profiles;
- Define and configure the types of visits offered, along with the associated pricing;
- Oversee appointments by identifying scheduled patients and appointments for the current day;
- Configure and manage appointment availability through customizable scheduling templates;
- Monitor and respond to patient reviews;
- Access analytics, including visual representations of revenue, newly acquired patients, recent reviews, and a summary of completed visits.

3.3 Non-Functional Requirements

The following non-functional requirements address the quality attributes of the system, ensuring its performance, reliability, security, scalability, and maintainability.

3.3.1 Performance and Scalability

- The system shall maintain **acceptable response times** for common operations such as viewing medical records and scheduling appointments;
- The system shall efficiently **handle increased workload** during peak usage periods without degradation of service.

3.3.2 Availability and Reliability

- The application shall be **available** to all users **24/7**, minimizing downtime through redundancy and failover mechanisms;
- The system shall implement **backup and recovery procedures** to preserve data integrity and support rapid restoration after failures.
- The system shall tolerate occasional data staleness in non-critical views, ensuring high availability even under degraded conditions.

3.3.3 Security and Privacy

- The system shall enforce **secure, authenticated access for all users**, employing strong password policies and session management;
- All sensitive data shall be encrypted both in transit and at rest;
- The system provides defenses against injections.

3.3.4 Usability

- The user interface shall be intuitive and **user-friendly**, enabling users to perform tasks with minimal learning curve.
- The application shall exhibit **low latency** in user interactions to maintain a responsive experience.

3.3.5 Portability and Flexibility

- The application shall be **deployable on multiple operating systems** (e.g., Windows, macOS, Linux) without requiring behavioral changes.
- The system's architecture shall support the addition of new attributes or modules (e.g., new appointment types) with minimal code modification.

3.3.6 Maintainability

- The codebase shall follow **Object-Oriented design** principles, promoting modularity and ease of comprehension.
- The system shall **minimize single points of failure** through component decoupling and redundancy.
- The application shall include comprehensive documentation and code comments to facilitate future enhancements and debugging.

3.4 UML Class Diagram

...

Chapter 4

Implementation

4.1 Front-End

The front-end of our application has been developed as a traditional web-based interface, utilizing a combination of **JavaScript**, **HTML** and **CSS**. This technology stack enables the creation of a responsive and interactive user interface that can be accessed from any modern web browser without the need for additional installations. The application communicates asynchronously with the back-end server through **AJAX** (Asynchronous JavaScript and XML) requests, allowing data to be fetched and updated dynamically without requiring full page reloads. This results in a smoother user experience and improved performance, particularly when handling operations such as appointment booking, profile management, and real-time data visualization. The separation of concerns between presentation and logic also enhances the maintainability and scalability of the application.

4.2 Back-End

The back-end is implemented using **Java** due to its robustness, platform independence, and strong ecosystem. Java offers a well-established set of libraries and frameworks that support rapid development of secure, scalable, and maintainable web applications. Its object-oriented nature encourages clean architecture and modular design, which are essential for complex systems like healthcare platforms. Additionally, Java's widespread use in enterprise environments ensures long-term support and community-driven innovation, making it a reliable choice for production-grade applications.

4.3 Spring Boot

As the core of our application, we adopted the **Spring Boot** framework, which has significantly streamlined both the development and deployment processes. One of the primary advantages of Spring Boot lies in its built-in support for an embedded **Tomcat** server, allowing the creation of a fully functional web application without requiring external configuration. This design choice has enhanced our ability to construct responsive and maintainable

RESTful APIs.

In addition, to support our data management needs, we integrated **Spring Data Neo4j** and **Spring Data MongoDB**. The former facilitates interaction with the **Neo4j graph database** through a consistent repository-based abstraction, ideal for managing highly interconnected medical data such as relationships between doctors and patients. The latter offers seamless integration with **MongoDB**, **a document-oriented NoSQL database**, while preserving the flexibility and idiomatic programming style of the broader Spring ecosystem. Collectively, these technologies have provided a robust, scalable and modular architecture well-suited to the requirements of our application.

The application follows the layered architecture presented during lectures, organizing the codebase into clearly defined packages to promote modularity and separation of concerns. Specifically:

- The **Config** package contains configuration classes;
- The **Controller** package handles API endpoint mappings, the **Model** package defines the entity classes corresponding to the database schema. The Controller also contains the **APIs** with which the front-end communicates;
- The **Repository** package provides data access functionality through Spring Data JPA interfaces;
- The **Resources** directory includes external configuration files such as `application.properties`;
- The **Service** layer encapsulates the core business logic;

Bibliography