COMP6251: Web and Cloud Applications Development Coursework

Team 01

Semester II

Team Members

Sr. No.	Student Id	Name	Email
1	34875441	Rithin Menezes	rrm1n23@soton.ac.uk
2	35117311	Tanvi Patil	tp3n23@soton.ac.uk
3	35431563	Toshna Rane	tr4u23@soton.ac.uk

Table of Contents

1. Prototype Functionality Description

1.1 Overview of Implemented Features

This web application enhances healthcare management by facilitating secure and efficient interactions among patients, doctors, practitioners, and administrators.

1.1.1 Patient Functions:

- **Create an Account:** Register by agreeing to GDPR compliance.
- **View Practices:** Access and register with multiple health practices.
- **Book Appointments:** Schedule appointments and describe health concerns.
- **View Appointments:** Monitor appointment requests and statuses.
- **Medical Records:** Access uploaded medical records per appointment.
- **GDPR Compliance Management:**
    - **View, Update Data**: Manage personal data as per GDPR guidelines.
    - **Email Notifications:** Receive automated emails for various activities.
    - **Receive Test Results:** Obtain test results via email.
- **Pharmacy Locator:** Locate pharmacies using NHS data and Google Maps.
- **Email Notifications:** Receive automated emails for various activities.
- **Receive Test Results:** Obtain test results via email.
- **Forgot Password:** Reset password during sign-in if forgotten.

1.1.2 Doctor Functions:

- **View Appointment Requests**: Access all appointment requests from patients.
- **Upload Prescription**: Provide prescriptions for patients.
- **Update Medical History**: Maintain and update patients' medical history.
- **Order Tests**: Arrange necessary tests for comprehensive care.

1.1.3 Practitioner Functions:

- **Approve/Decline Appointments**: Assign appointments to doctors based on description.
- **Offer Alternatives**: Suggest alternative appointment options.
- **Manage Test Appointments**: Notify patients about test results and update medical history.

1.1.4 Admin Functions:

- **Approve/Decline Practice Registrations**: Manage practice registration requests.
- **Account Management**: Create and update accounts for Practices, Practitioners, and Doctors.

1.2 Assumptions and Interpretations:

- **Data Security and Integrity:** Users provide accurate personal details during registration.
- **Single Practice Registration:** Patients register to one practice; can change practice if visiting a different location.
- **GDPR Compliance:** Features for viewing and updating personal data. Future enhancement: ability to delete personal data.
- **Continuous Internet Access:** Required for application functionality.
- **Valid Information:** Must be provided during registration.
- **Practice Structure:** Each practice has one practitioner and multiple doctors.
- **Communication of Test Results:** Handled by practitioners.
- **Medical Tasks:** Doctors handle prescriptions, test orders, and medical history updates.

This application streamlines healthcare management processes, ensuring secure and efficient interaction while maintaining data security and regulatory compliance. It is designed to be robust, user-friendly, and meet the diverse needs of its users.

2. Tools and Techniques

2.1 Tools and Frameworks Used

This project leverages modern technologies across various domains to build a responsive web application.

2.1.1 Tools and Technologies:

- **React:** Dynamic rendering and state management with a component-based structure.
- **Tailwind CSS:** Utility-first styling for rapid UI development.
- **Firebase:** Comprehensive backend services including authentication, analytics, and hosting.
- **Node.js:** Efficient server-side operations.
- **Firestore:** Real-time NoSQL database for seamless data synchronisation.
- **Axios:** Handles HTTP requests with a promise-based structure.
- **Nodemailer:** Sends emails from Node.js applications.

2.1.2 Development Environment:

- **Visual Studio Code (VS Code):** Extensive extensions, robust debugging, and built-in Git support.
- **GitHub:** Version control, development management, collaboration, and continuous integration.

2.2 Justification for Tool Choices

1. **React and Tailwind CSS:**
    - **React:** Enables responsive, dynamic user interfaces with reusable, modular components, essential for scalability and maintainability.
    - **Tailwind CSS:** Provides a utility-first approach to styling, accelerating development and ensuring consistent design with minimal custom CSS.
2. **Firebase and Node.js:**
    - **Firebase:** Streamlines development with built-in authentication, real-time databases, and hosting. Its integration capabilities are crucial for developing secure, compliant healthcare applications.
    - **Node.js:** Efficiently handles asynchronous I/O operations, vital for backend services requiring real-time data processing and scalability.
3. **Firestore:**
    - **Real-Time Data Synchronisation:** Essential for collaborative healthcare applications, ensuring instant updates and seamless data management.
    - **Scalability:** Supports high volumes of patient data, providing a reliable, efficient database solution integrated seamlessly within the Firebase ecosystem.
4. **Axios:**
    - **HTTP Requests:** Ensures robust and reliable network request handling, critical for managing sensitive health data and enabling real-time client-server communication.
5. **Visual Studio Code:**
    - **Productivity:** Enhances development workflow with powerful editing, debugging, and extension capabilities, ideal for both novice and experienced developers.
6. **GitHub:**
    - **Version Control:** Facilitates effective collaboration and code management through version control and continuous integration, ensuring code quality and project efficiency.

The selected technologies offer robustness, high performance, and scalability, essential for modern web applications. The integration of these technologies ensures a smooth development process and a scalable, maintainable final product.

3. Design and Implementation Overview

3.1 Application Architecture

- **Client-Side Functions:**
  - **UI Rendering:** Utilises React for dynamic and responsive user interfaces.
  - **Styling:** Employs Tailwind CSS for utility-first styling and consistent design.
- **Server-Side Functions:**
  - **Authentication:** Manages user authentication with Firebase.
  - **Data Storage:** Uses Firestore for real-time data synchronisation and storage.
  - **Hosting:** Deploys the application using Firebase hosting.
- **Data Handling:**
  - **API Requests:** Axios handles HTTP requests for external API integration.
  - **Direct Data Management:** Firebase integration allows direct data handling within React components.

3.2 Design Principles

- **Separation of Concerns:** Independent development, testing, and updates for logic.
- **Modularity:** Reusable, React components reduce redundancy and improve maintainability.
- **DRY (Don't Repeat Yourself):** Reduces code repetition, enhancing scalability.
- **Code Consistency and Quality:** ESLint and Prettier enforce consistent, high-quality code.

3.3 Design Patterns

- **Factory Pattern:** Used for dynamic component rendering, allowing the application to fluidly handle various user roles and permissions.
- **Builder Pattern:** Utilised for assembling complex user profiles and health records, allowing flexible and extensible data structures.

3.4 Database Design

- **Collections:**
  - **admin:** Stores admin user details and permissions.
  - **appointment_booking:** Contains appointment details, including patient info, doctor assigned, date, and status.
  - **doctors:** Records details of doctors, including specialties and availability.
  - **patient:** Stores patient personal information and medical history.
  - **patient_practice_registration:** Tracks patient registrations to various practices.
  - **practice:** Contains healthcare practices information, location and services offered.
  - **practitioner:** Stores practitioner details, roles, and assigned patients.
  - **prescriptions:** Contains prescription details, medication prescribed, and associated patient and doctor.
  - **user_type:** Manages different user roles and access levels.

3.5 Implementation Overview

- **Architectural Strategy:**
  - **Minimised Server Management:** Firebase reduces the need for traditional server-side management, focusing on front-end development and real-time data interactions.
  - **Real-Time Interactions:** Ensures seamless user experience in healthcare applications through real-time updates.
  - **Code Quality:** ESLint and Prettier maintain high code quality and reduce bugs.

4. Techniques for Testing and Deploying Web Applications

4.1 Testing Methods

4.1.1 Test-Driven Development (TDD):

- **Approach:** Write tests before implementation to ensure thorough testing.
- **Benefits:** Identifies edge cases early, ensuring a robust and bug-free codebase.

4.1.2 Jest and React Testing Library:

- **Jest:**
  - **Unit Tests:** Test individual components/functions in isolation.
  - **Integration Tests:** Check interactions between different parts of the application.
  - **Snapshot Tests:** Capture and compare rendered output to detect unintended UI changes.
- **React Testing Library:**
  - **Rendering Tests:** Ensure components render correctly with given props.
  - **Event Tests:** Simulate user interactions to verify component responses.

4.1.3 Sample Test Cases:

- **Patient Registration Component:**
  - **Unit Test:** Verify registration form renders correctly with all input fields.
  - **Event Test:** Simulate form submission and check if correct functions are called.
- **Appointment Booking Component:**
  - **Snapshot Test:** Capture and compare rendered output.
  - **Integration Test:** Ensure booking data is correctly saved to Firestore.
- **Pharmacy Locator:**
  - **Snapshot Test:** Capture and compare map with nearby pharmacies.
  - **Integration Test:** Ensure pharmacy dataset is correctly displayed by searching respective cities.

4.2 Deployment Techniques

4.2.1 Deployment Strategy:

- **Firebase Project:** Create in Firebase Console to manage backend services.
- **Hosting Setup:** Initialise Firebase Hosting with firebase init.
- **Build and Deploy:** Use npm run build to generate static files, deployed with firebase deploy.

4.2.2 Continuous Integration (CI):

- **GitHub Actions:** Automates build and deployment on every push to the main branch, ensuring code quality and automatic deployment if tests pass.

4.2.3 Deployment Steps:

1. **Initialise Firebase:** firebase init
2. **Build the Application:** npm run build
3. **Deploy to Firebase:** firebase deploy
   Links for deployed Project: 1. Frontend 2. Backend

4.2.4 Portability and Business Logic Testing:

- **Cross-Browser Testing:** Tested on Chrome, Firefox, Safari for compatibility.
- **Responsiveness Testing:** Ensured application works on various devices and screen sizes via manual and automated tests.
- **Business Logic Tests:** Thoroughly tested critical functionalities like appointment scheduling, user authentication, and data management.

By adopting these rigorous testing and deployment practices, the application is ensured to be robust, reliable, and ready for real-world use in a healthcare environment.

5. Relevant Statistics and Continuous Integration

5.1 Project Statistics

- **Lines of Code (LoC):** Over 10,000 lines of code, indicating the scale and complexity of the application.
- **Code Documentation:** Inline comments provide clarity on functionality, with detailed descriptions of functions and modules.
- **External Sources:**
    - **Third-Party Libraries:**
        - **React:** For building the user interface.
        - **Tailwind CSS:** For styling the application.
        - **React Redux:** For state management.
        - **React DatePicker:** For date selection in forms.
        - **React Router:** For routing within the application.
        - **Lucide React:** For icons.
        - **Google Maps API:** For displaying pharmacy locations.
        - **Axios:** For making HTTP requests.
        - **Nodemailer:** For sending emails.
    - **Code Snippets:** For different dashboards

5.2 Test Coverage:

Nearly 80% test coverage, ensuring all functions and pathways are verified, reducing bugs and enhancing reliability.

5.3 Continuous Integration (CI) Practices

- **Private GitHub Repository:** Central hub for development, enabling collaboration, code reviews, branch management, and version tracking.
    - **Commit History:** ***Figure 1:*** *Commit history showing regular updates and detailed commit messages.*
- **Branch Management:** Dedicated branches for features, bug fixes, and experiments ensure the main branch remains stable and deployable.
- **Pull Requests and Code Reviews:** Mandatory code reviews enhance quality by catching issues early.
- **Automated Testing:** Unit, integration, and functional tests run on every commit to any branch using Jest and React Testing Library.
- **Build Automation:** GitHub Actions automate builds and tests on every push, maintaining high code quality and readiness for deployment.

5.4 Evidence of Version Control and Continuous Integration

- **GitHub Repository:** Managed with a private repository. Screenshot demonstrate commit history.

Detailed statistical tracking and robust continuous integration practices are essential for a high-quality development process. They enhance application reliability, foster efficient team collaboration, and uphold professional standards crucial for complex projects like healthcare management systems.

6. Critical Evaluation of the Web Application

6.1 Application Architecture:

6.1.1 Evaluation of Client-Side Technologies (React)

- **Modularity:** Component-based structure for reusable UI components, simplifying maintenance and enhancing scalability.
- **Performance:** Virtual DOM optimises rendering, improving performance and user experience.

6.1.2 Evaluation of Server-Side Technologies (Firebase)

- **Real-Time Data Synchronisation:** Firestore ensures updates are instantly reflected across all clients.
- **Scalability:** Serverless architecture accommodates varying loads without manual intervention.
- **Integrated Services:** Firebase Authentication, Firestore, and Hosting backend development.

6.2 Effectiveness of Development Approaches (TDD)

- **Early Bug Detection:** Identifies and fixes bugs early, reducing critical issues in production.
- **High Test Coverage:** Ensures reliable and stable functionality.
- **Documentation:** Tests serve as documentation, clarifying code usage.

6.2.1 Performance and Scalability

- **Performance:** Firebase's serverless architecture ensures low-latency data access and efficient resource utilisation.
- **Scalability:** Automatic handling of scaling, managing increased loads without additional infrastructure.

6.2.2 Security Measures

- **Data Protection:** Secure authentication and data access through Firebase Authentication and Firestore security rules.
- **GDPR Compliance:** Adheres to guidelines by obtaining user consent and providing data management functionalities.

6.3 User Experience

- **Responsive Design:** Adaptability across devices with Tailwind CSS.
- **User Value:** Pharmacy Locator with Google Maps API enhances user convenience.

6.4 Adaptability and Future Proofing

- **Modular Architecture:** React components and Firebase services allow easy feature addition and modification.
- **Continuous Integration:** CI/CD pipelines with GitHub Actions ensure rapid and reliable updates.
- **Future Enhancements:**
    - **Appointment Reminders:** Automated email reminders for upcoming appointments.
    - **Enhanced Reporting:** Additional reports for patient and appointment statistics.
    - **User Feedback:** Implement a feedback system for users to rate their experience.
    - **Improved Notification System:** Enhance notification system for real-time updates.

A healthcare application built with React and Firebase offers responsive, real-time functionality. Using TDD and continuous integration, it ensures high code quality and rapid updates. This systematic approach supports scalability and adaptability, meeting evolving healthcare needs and maintaining professional development standards.

7. Appendix

Included code samples, UML diagrams, screenshots of the application interface:

UML Diagrams:

Fig 1: Flowchart:

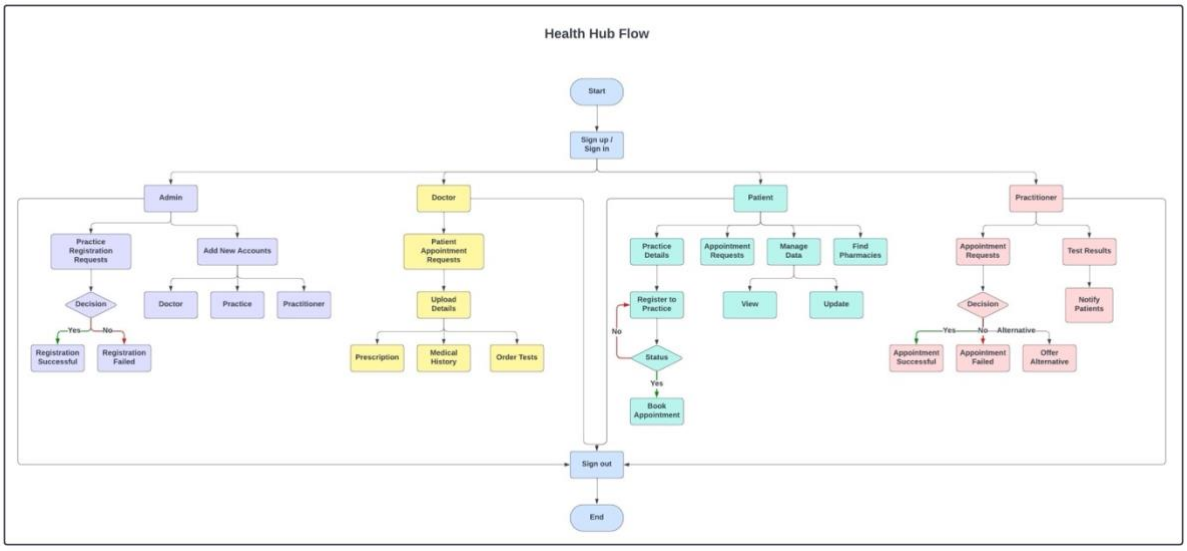


Fig 2: Use Case Diagram

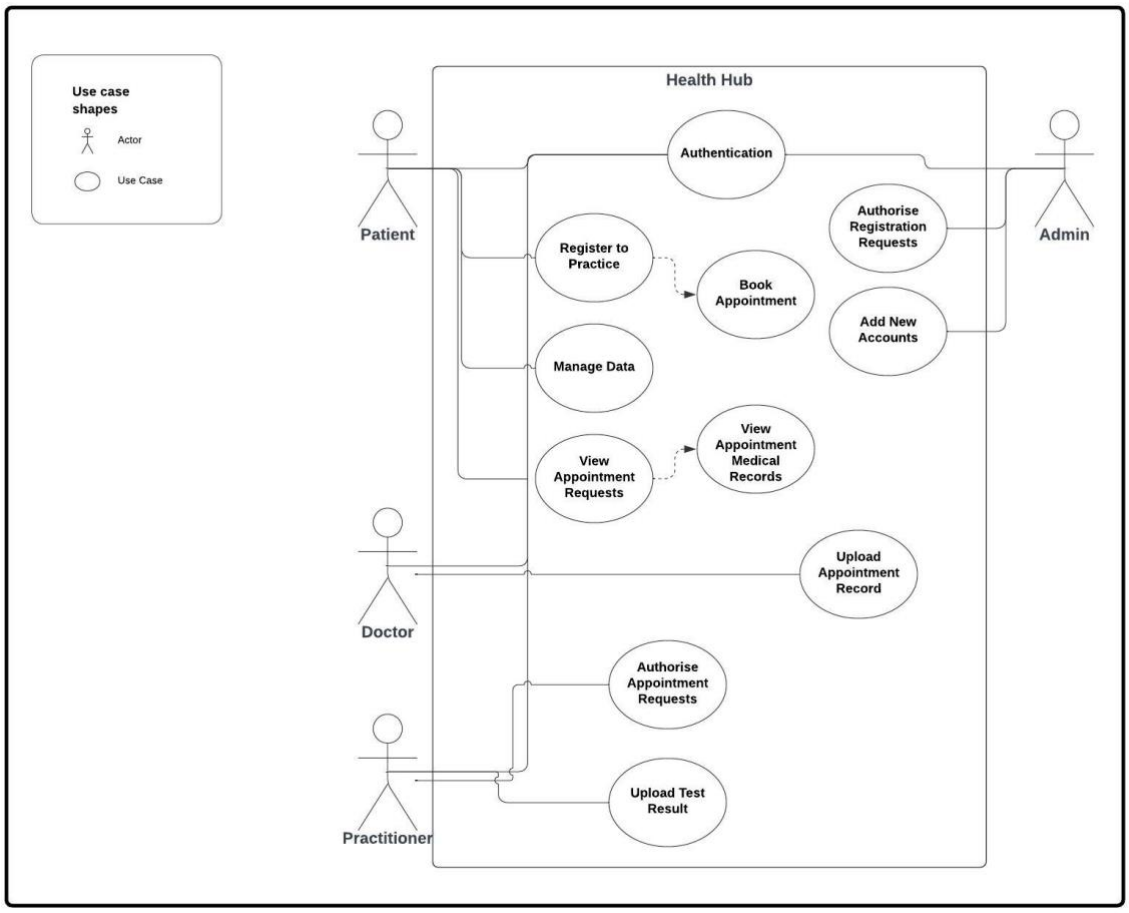


Fig 3. Wireframes:

Fig 3.1 Sign up Wireframe:

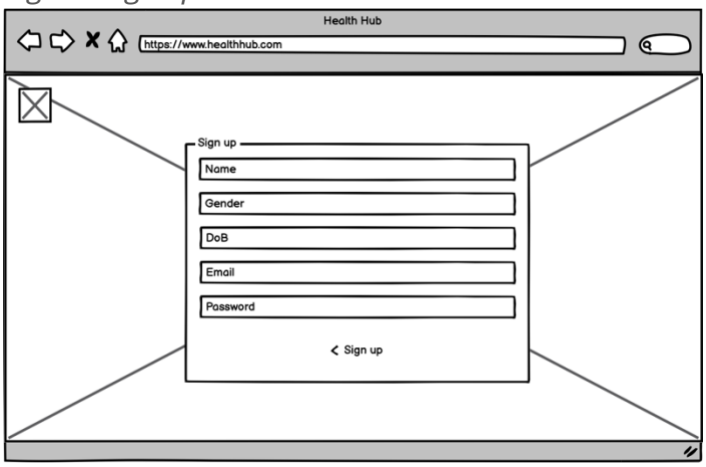


Fig. 3.2 Sign in Wireframe

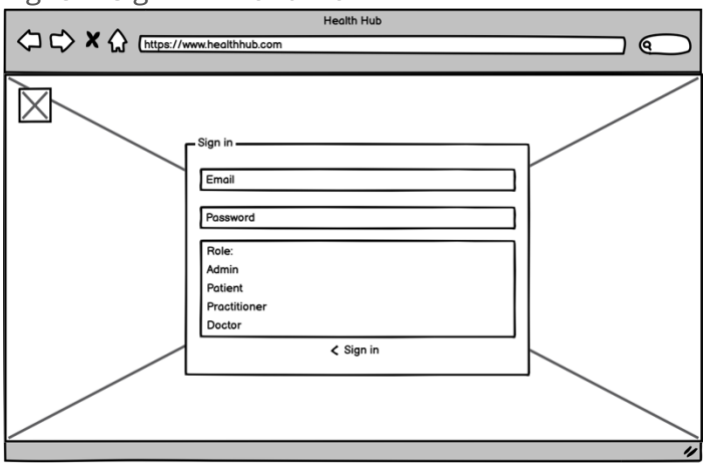


Fig 3.3 Patient Practice Wireframe

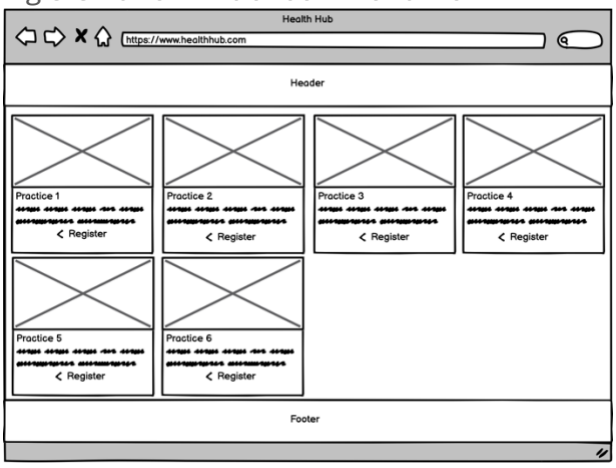


Fig 3.4 Admin Wireframe

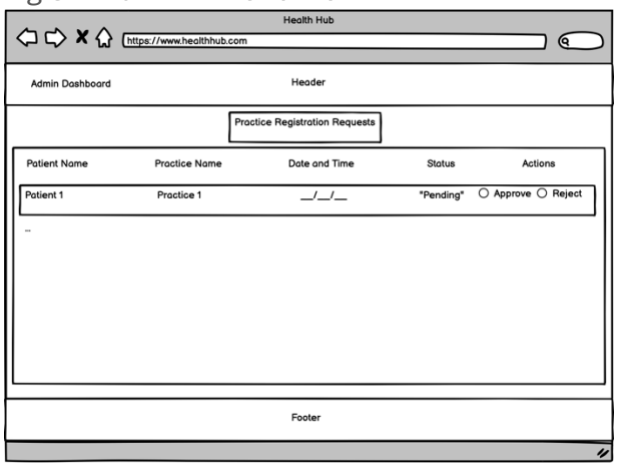


Fig 3.5 Patient Appointment Wireframe

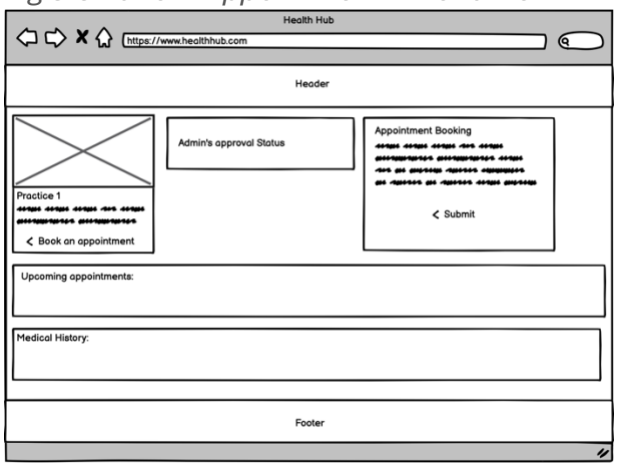


Fig 3.6 Practitioner Wireframe

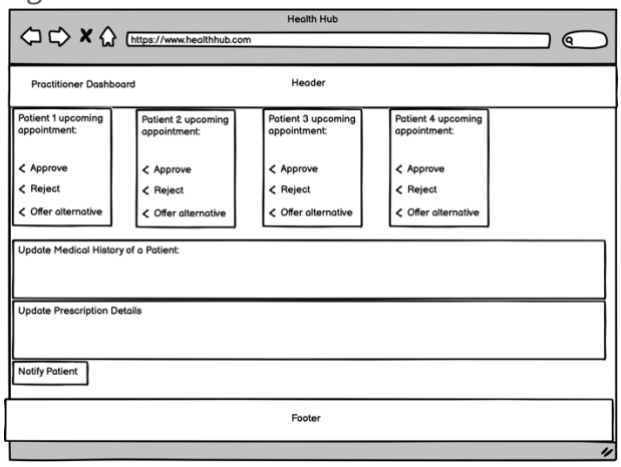


Fig 3.7 Doctor Wireframe

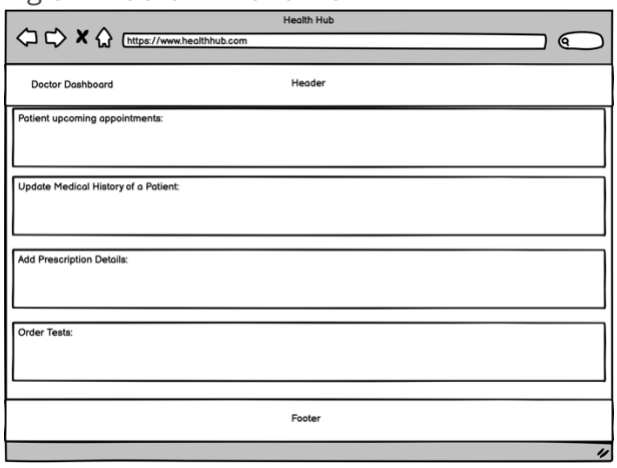


Fig 4. Screenshots:

Fig 4.1 Sign up

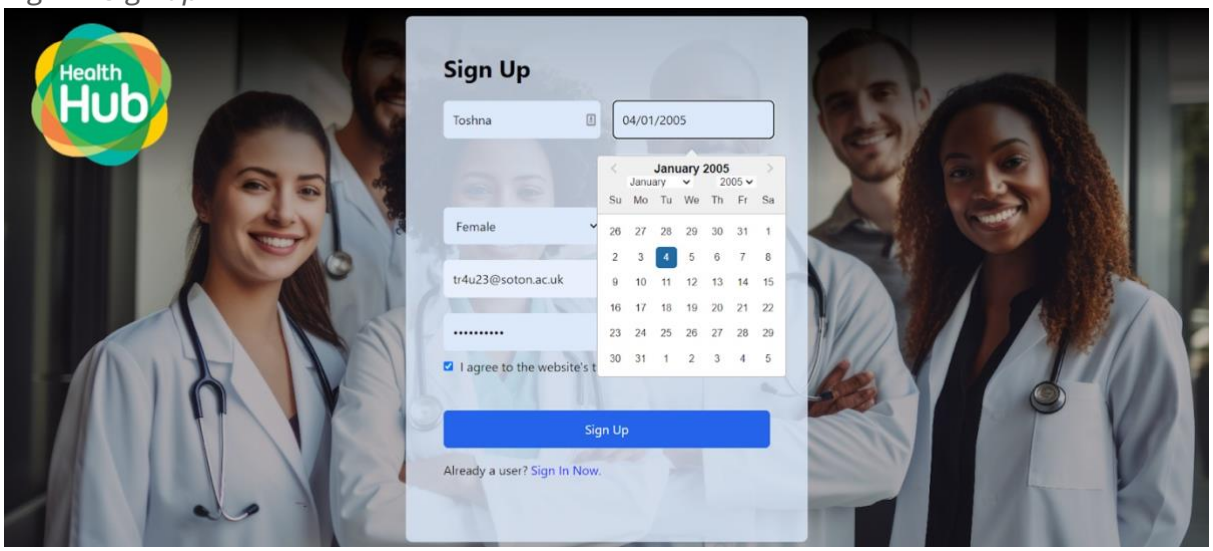


Fig 4.2 Sign in

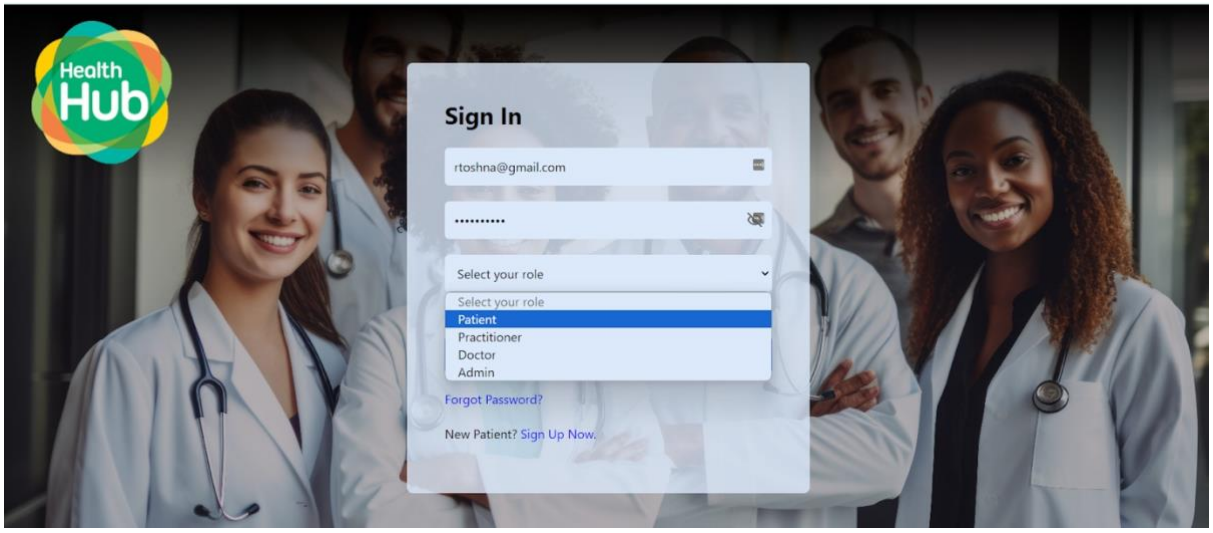


Fig 4.3 Practices

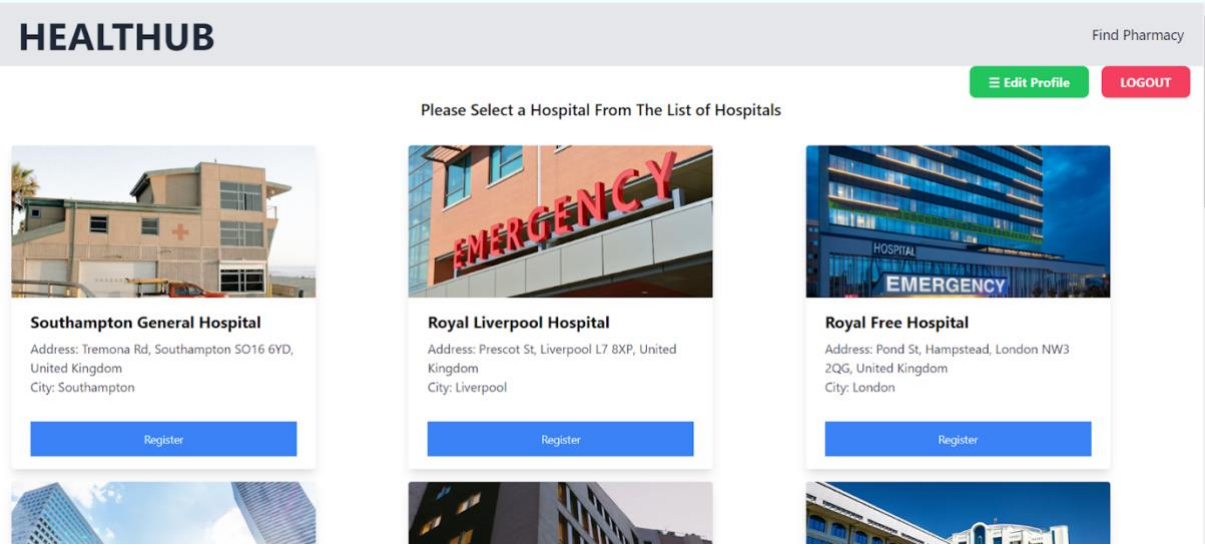
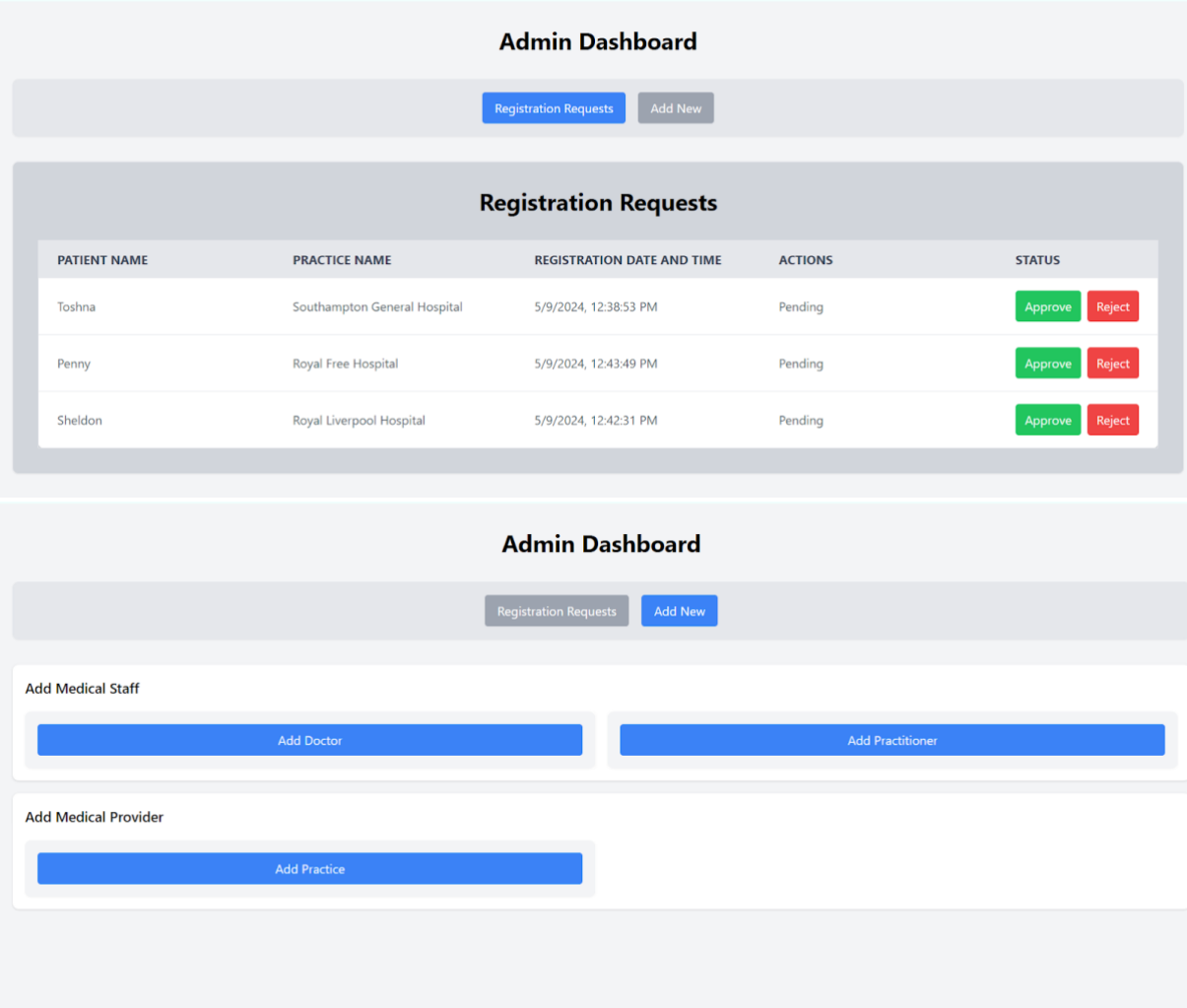


Fig 4.5 Admin Dashboard



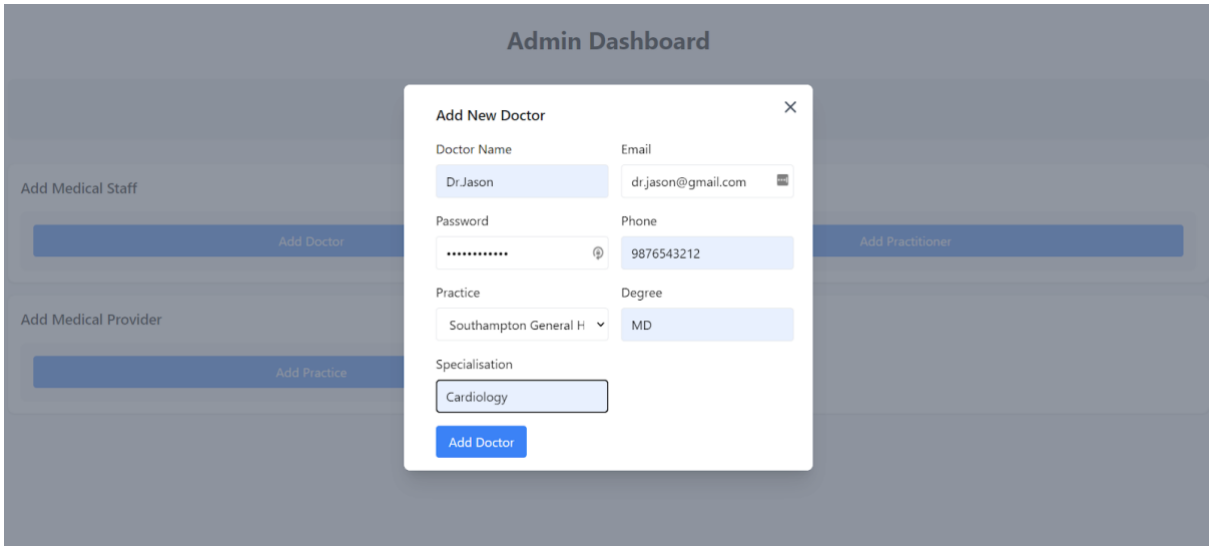


Fig 4.4 Admin Approval

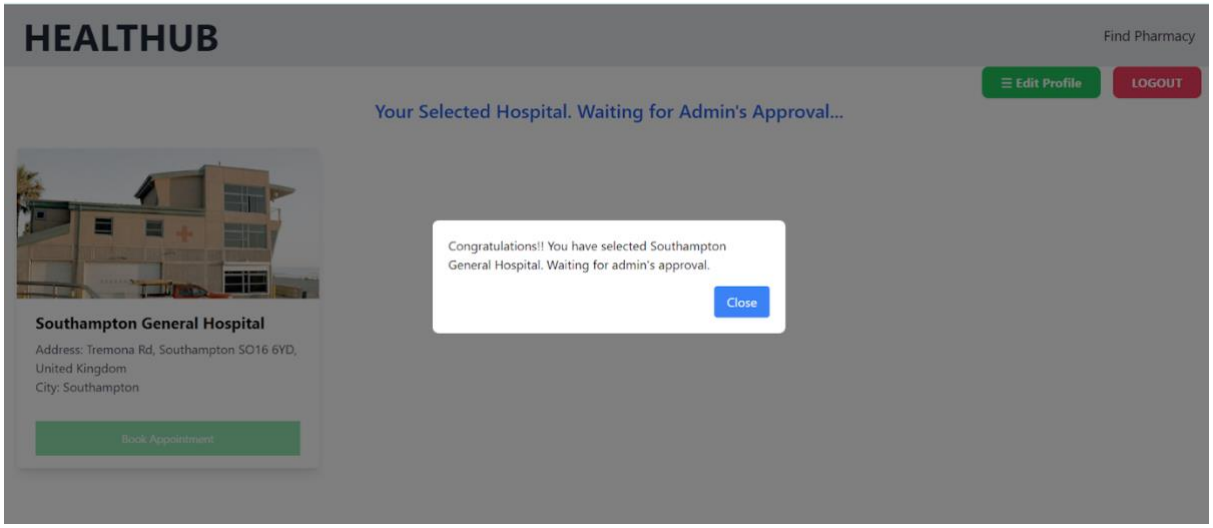
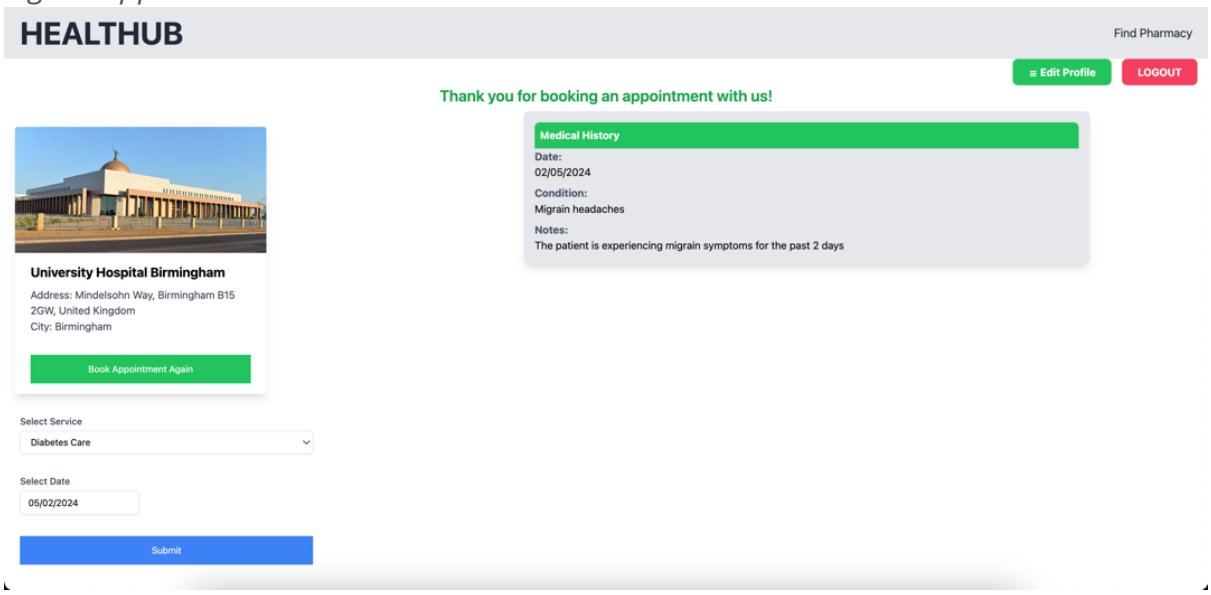
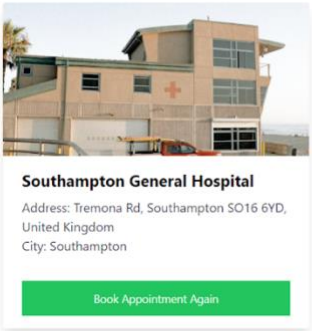


Fig 4.6 Appointment





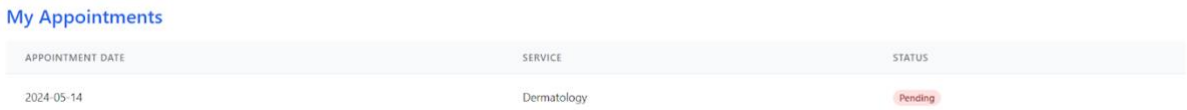


Fig 4.7 GDPR

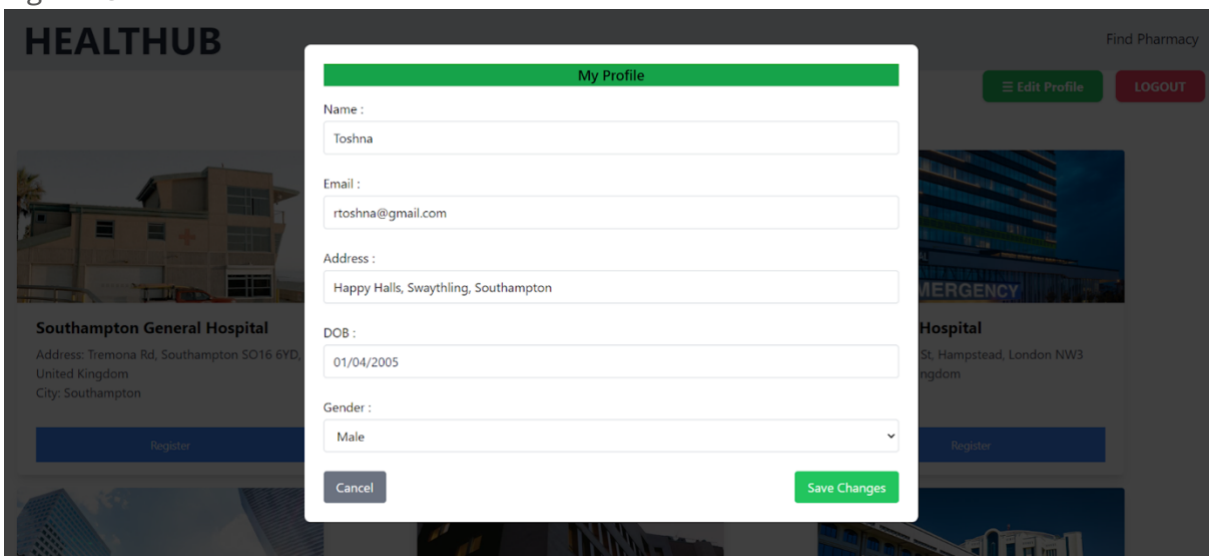


Fig 4.8 Practitioner

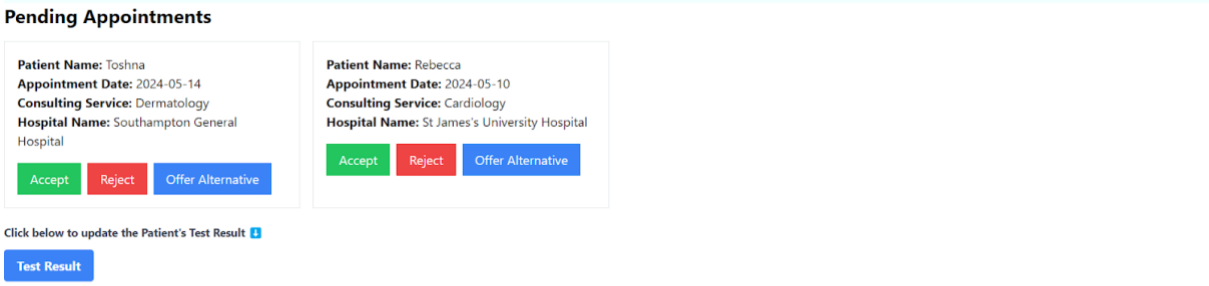


Fig 4.9 Doctor

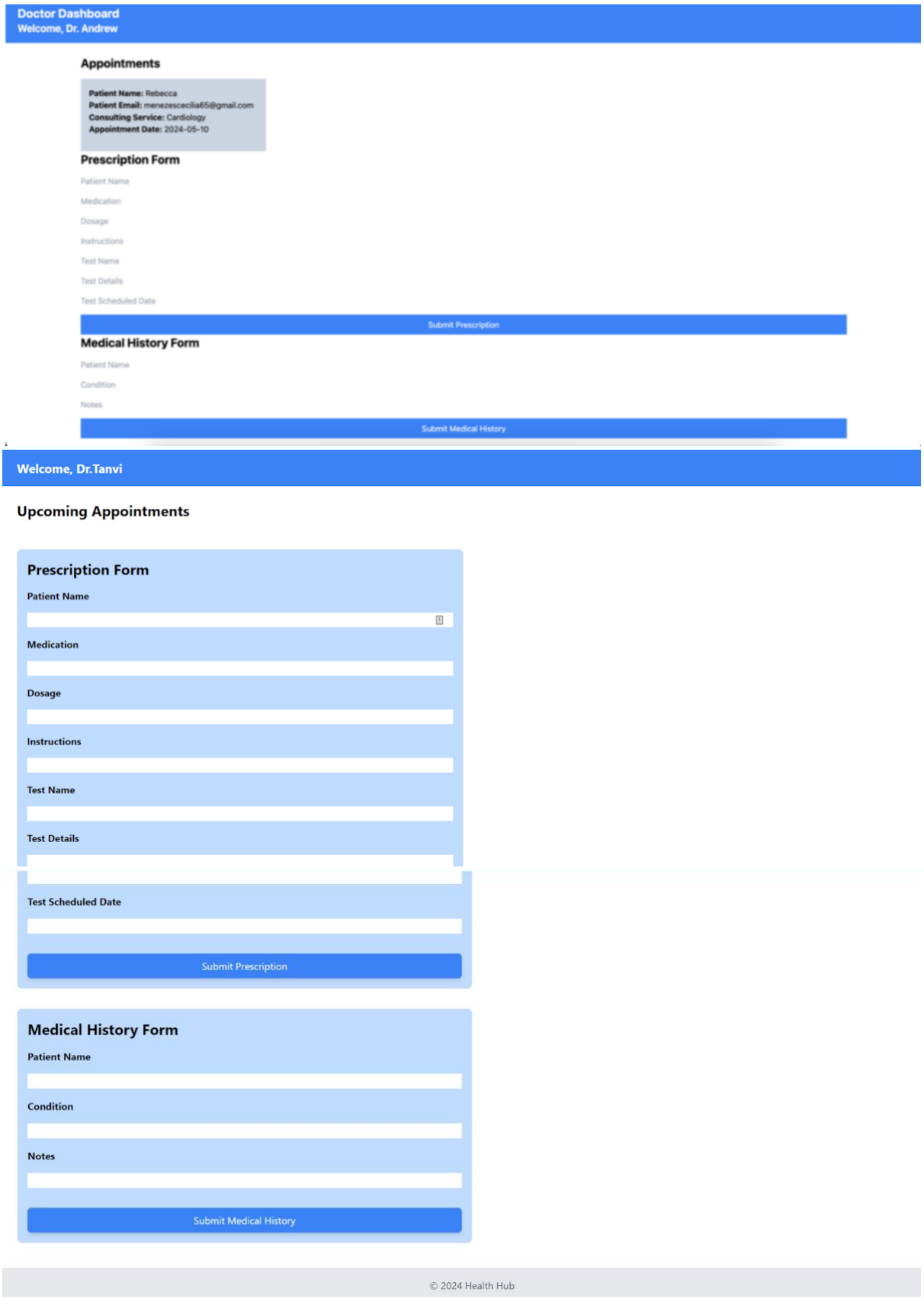


Fig 4.10 Pharmacy

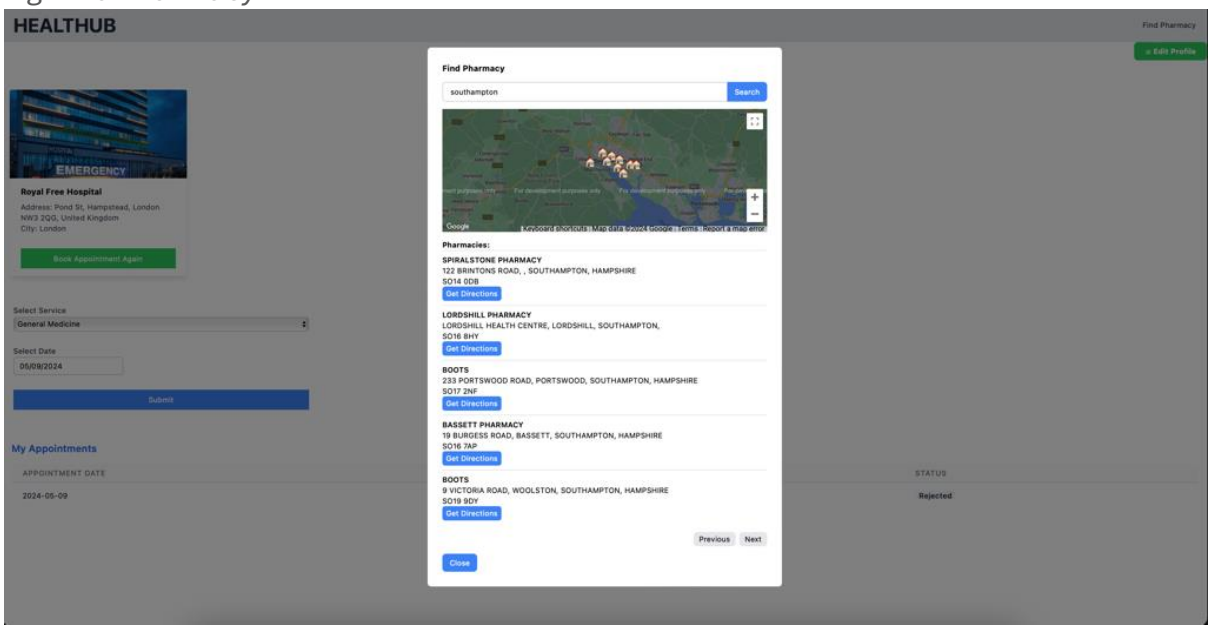


Fig 5. Code Snippets

Fig 5.1 Practitioner and Hospital

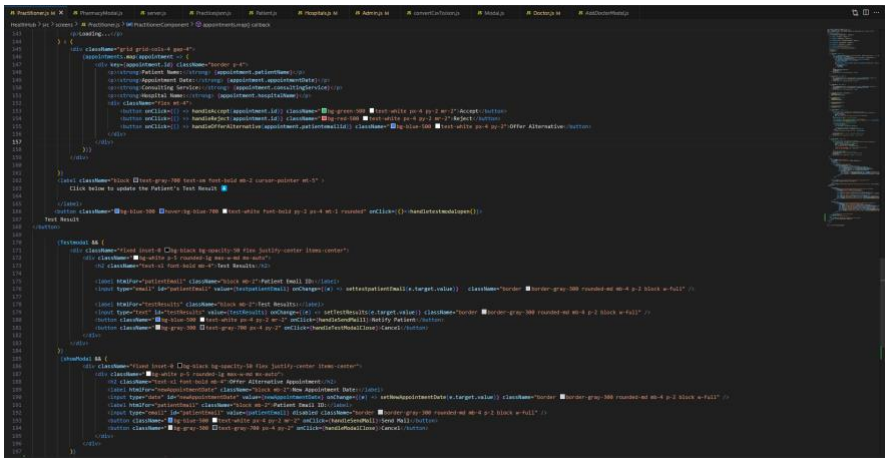


Fig 5.2 Admin Dashboard

Fig 5.4 Doctor Dashboard

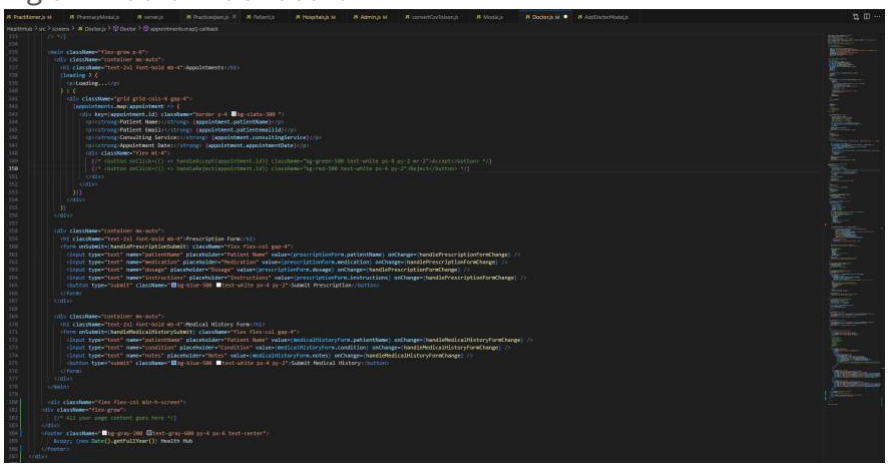


Fig 5.5 Sign up / Sign in

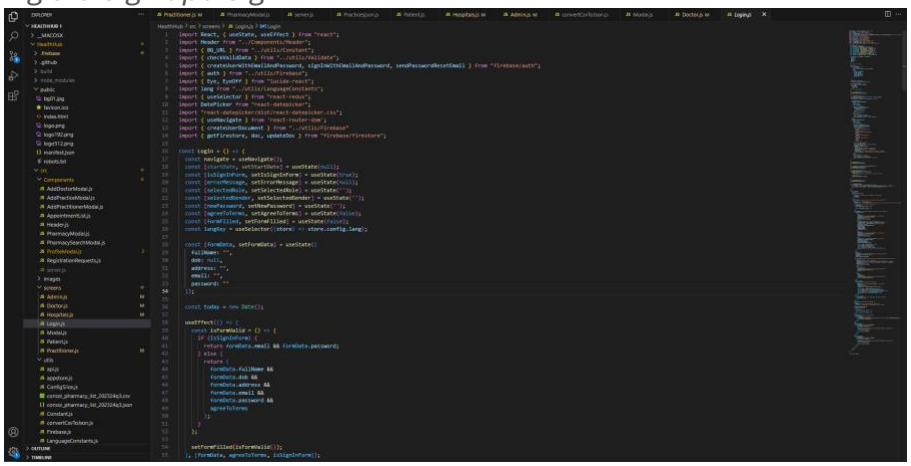


Fig 5.6 Mail

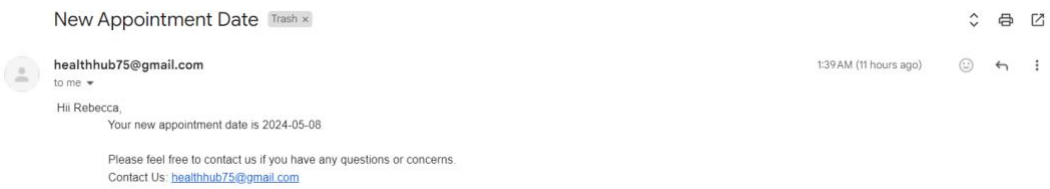


Fig 6. Github

5.4 Evidence of Version Control and Continuous Integration