Smart Hospital

By

Author's name

Department of Engineering Mathematics
University of Bristol

A dissertation submitted to the University of Bristol in accordance with the requirements of the degree of Doctor of Philosophy in the Faculty of Engineering.

April 2013

Word count: ten thousand and four

Abstract

Here write the abstract

Dedication and acknowledgements

Here goes the dedication.

Author's declaration

I declare that the work in this dissertation was carried out in accordance with the requirements of the University's Regulations and Code of Practice for Research Degree Programmes and that it has not been submitted for any other academic award. Except where indicated by specific reference in the text, the work is the candidate's own work. Work done in collaboration with, or with the assistance of, others, is indicated as such. Any views expressed in the dissertation are those of the author.

SIGNED: ..... DATE: .....

Table of Contents

List of Tables

Table **Page**

List of Figures

Chapter 1

Introduction

1.1 Motivation

The program will build a virtual hospital website called Smart Hospital, which aims to provide better access to medical care for residents in remote or medically underserved areas in Africa. Due to geographical limitations, transportation difficulties, and insufficient medical facilities, many residents often cannot receive good medical care. Through this virtual hospital website, residents in these areas will have the opportunity to consult with specialists remotely and receive more medical assistance and health support.
This project was initially part of a collaboration with the Africa Virtual Hospital initiative, which is dedicated to addressing healthcare disparities in remote and underserved communities across Africa. The initiative aims to connect these underserved regions with doctors and experts from around the world through its virtual hospital platform. However, despite the termination of the collaboration due to certain factors, our team chose to independently continue this project. In the context of rising global healthcare pressures, including an aging population, increasing rates of chronic diseases, and uneven distribution of medical resources, we recognize that establishing a digital healthcare website to provide telemedicine remains highly meaningful, particularly for regions with insufficient medical resources or limited access to healthcare.

1.2 Client Brief

This project was initially undertaken in collaboration with Africa Virtual Hospital. As the client, Africa Virtual Hospital proposed the development of a virtual healthcare software platform comprising the following five main components: Doctor Application, WhatsApp Chatbots for Patients and Service Providers, Caregiver Application, Nurse Application, and Electronic Health Record. Within this framework, our team's primary focus was on the doctor application, which was expected to run on multiple devices, including smartphones, tablets, and laptops.
The Doctor Application component was expected to include the following elements:

- Registration
- SSO/Welcome
- Home page (Appointments and Reviews)
- Calendar (Availability)
- Notifications
- Patient Profile
- Connect (Video and Voice to Patient WhatsApp)
- Treatment Plan and Orders
- Patient Information Feed
- My Supporting Staff
- My Profile

During the collaboration period, we developed the various webpages and features of the doctor application based on the design and technical resources provided by the client. However, with the termination of the collaboration, our team also lost the original technical support. And then our new client is our University, but given the constraints on development time and resources, our team adjusted the scope based on existing resources and the original proposal, re-planned the website architecture and front-end design, and prioritized retaining core functional modules such as the doctor-patient interaction interface and basic medical information viewing, to ensure the project's feasibility under current conditions.

1.3 Aims and Objectives

The aim of this project is to design and develop an easy-to-use, cross-device virtual hospital platform to promote telemedicine and doctor-patient communication, thereby improving medical accessibility in remote areas.
Based on the aim, we have established the following specific implementation items:
Develop an intuitive user interface and create a web-based system that supports physician operations. Implement core system functions, including appointment scheduling, patient record access, and basic remote consultation workflows. Establish an integrated patient data interface to help physicians understand the patient's overall condition, thereby improving diagnostic efficiency and decision-making quality.

1.4 Challenges

This project faced three main challenges: unexpected termination of cooperation, limited development time, and difficulties in obtaining back-end data resources.

First, the project began in June and was expected to end in early September. Under the original plan, we were to collaborate with Africa Virtual Hospital to complete the development of the Doctor Application. Although most team members lacked website development experience, the development work proceeded smoothly under the guidance of Africa Virtual Hospital in the early stages.

However, the partnership was terminated prematurely in mid-July, resulting in the team losing access to technical support and shared data, and significantly compressing the remaining development timeline. With only about one and a half months left for development, the team had to independently complete tasks such as system design and construction, leading to an extremely tight schedule.

Additionally, the team had originally planned to utilize existing data from Africa Virtual Hospital, such as patient basic information and medical records. However, due to the termination of the collaboration, our backend development team faced difficulties in accessing patient data and medical records, which are highly sensitive, thereby imposing some limitations on website functionality verification and data simulation.

Despite these challenges, our team adjusted the project scope, reallocated resources, and prioritized the implementation of core functions to ensure the platform's feasibility.

Chapter 2

Background

2.1 Relative Work

In many hospitals, patients need to wait for a long time just to have a short consultation with a doctor. This can be exhausting, especially for people who are already sick. In some places like the countryside, the problem is worse because there are not enough doctors or clinics [4]. Some people need to travel very far to get help, which is hard for old people or people with long-term illness.

During COVID-19, many doctors started using phone or video to talk to patients. In the UK, it went from 15% to 48% in just a few weeks [8]. It helped people talk to doctors from home and also saved time.

Our Smart Hospital system also uses phone and video consultations. We also have Health QA, where people can ask simple health questions and get advice more easily. These help people who live far away or cannot go outside, and they also make the waiting time in hospitals shorter.

Our system supports the United Nations' SDG Goal 3, which is about helping people stay healthy and live well [9]. Among all the targets, Goal 3.8 and Goal 3.4 are especially important to our project. Goal 3.8 is about giving basic and affordable health services to everyone. Goal 3.4 is about helping people with long-term sickness, like heart disease or mental problems. Our system lets them contact doctors or get advice online more easily, so they can get help earlier before the problem becomes worse.

2.2 Existing Solutions

2.2.1 Teladoc Health

Teladoc Health is a popular online medical platform[**?**] that offers services like video calls with doctors and support for mental health. It is used in many countries and is helpful for people

who find it hard to go to a hospital. But one problem is that it's not fully connected to real hospitals. This means doctors on Teladoc often can't see a patient's full medical records from other clinics or hospitals. As a result, some information might be missing, which can make it harder to give the right diagnosis or follow-up care.

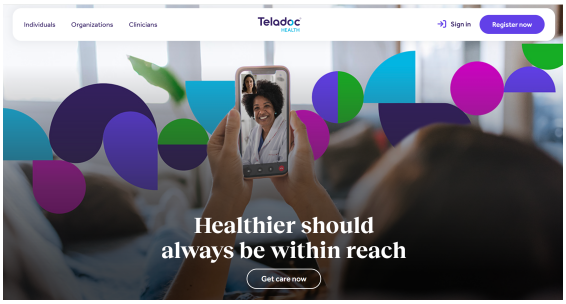


Figure 2.1: Homepage of Teladoc Health.

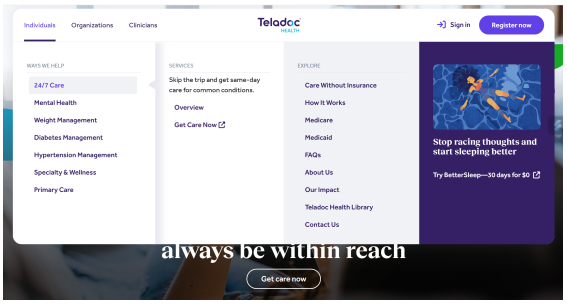


Figure 2.2: Service menu of Teladoc Health.

2.2.2 LINE Hospital Services

In Taiwan, many hospitals have their own official LINE accounts to provide online services. Patients can add the hospital's LINE to access booking and other basic functions. It is very easy to use, since most people in Taiwan already use LINE in their daily life. This makes it a common way for hospitals to offer simple medical services without asking people to download another app.

Figure 2.3: Screenshot of LINE services from Far Eastern Memorial Hospital, showing options like booking, announcements, and personal info links [3].

These features are quite easy to use. However, each hospital has its own design and menu. Some functions even open in separate websites. There is no unified system, so it can be confusing for some patients, especially older people or those not familiar with technology.

2.3 Requirements

The system is designed as a web-based platform for doctors, nurses, and patients to facilitate efficient clinical workflows and improve access to healthcare services. The main objective is to allow healthcare staff to manage patient information, record essential clinical data, and provide consultations remotely, thereby reducing unnecessary hospital visits. Patients are also supported with features to access their health records and receive guidance without needing to attend the hospital in person.

To ensure a functional and effective experience for hospital staff and patients, the system was designed to support a core set of features. These include secure user login and authentication for doctors, nurses, and patients; the ability to record patient vitals and view historical data trends; and an interface for remote consultation to minimise unnecessary hospital visits.

Drawing on the team's previous experience with similar healthcare platforms, including the Virtual Hospital Africa system, we referenced their approach as a conceptual foundation while adapting the design to our own project scope and requirements.

The system is initially populated with pre-existing mock patient data to facilitate testing and demonstration. In addition to these accounts, the platform supports new patient registration, enabling patients to create their own accounts and access the same core features, including secure login, health query submission, and medical record downloads.

Non-functional requirements include secure authentication with role-based access control, ensuring that only authorised users can access or modify clinical records. The system is designed for stable operation with fast response times to support real-time clinical workflows. In addition, the user interface is kept clear and intuitive to reduce training time for hospital staff. These system capabilities are closely aligned with the user stories presented below.

As a...	**I want...**	**So that...**	**Technical ability**
Patient	To ask health-related questions online	I can get medical guidance without visiting the hospital	2–3
Patient	To download my complete medical record	I can share it with a pharmacy or another healthcare provider	2–3
Nurse	To record patient vitals and update profile information	I can ensure accurate data is available for diagnosis	2–4
Doctor	To review patient history and vitals trends	I can make informed clinical decisions	3–5
Doctor	To add clinical notes and prescriptions	I can provide clear treatment guidance for the patient	3–5

Table 2.1: Smart Hospital User Stories

Chapter 3

Design and Implementations

Begins a chapter. Example: When the beloved cellist (Christopher Walken - outstanding) of a world-renowned string quartet receives a life-changing diagnosis, the group's future suddenly hangs in the balance: suppressed emotions, competing egos and uncontrollable passions threaten to derail years of friendship and collaboration. Featuring a brilliant ensemble cast (including Philip Seymour Hoffman, Catherine Keener and Mark Ivanir as the three other quartet members), it is a fascinating look into the world of working musicians, and an elegant homage to chamber music and the cultural world of New York. The music, of course, is ravishing (the score is the work of regular David Lynch collaborator Angelo Badalamenti): A Late Quartet hits all the right notes.

3.1 Methodology

3.1.1 Work Flow

3.1.2 Front-end Tools

Figma

In our project, our goal is to design a website for medical use. As a team, we all agree that a high-quality website should include several essential factors. Among them, user interface (UI) and user experience (UX) are the most crucial. A well-designed website should maintain a consistent and visually appealing style across all pages, while also being intuitive to use and free from confusing workflows. Another key consideration is the design process itself. Since we are working as a team, it is important to choose a tool and workflow that supports efficient collaboration. First, team members should be allowed to work independently or collaboratively while still ensuring consistency across design. Second, we need a real-time collaborative environment to prevent any individual's progress from being delayed by others.

For achieving our project goals, we evaluate two tools, Canva and Figma. Both are useful for collaborative website development and support real-time commenting to improve teamwork.

However, the key differences lie in several areas. First is layout design. Figma allows for the detailed design of every element on each page, including font size and CSS box models. In contrast, Canva cannot adjust all elements individually and focuses more on overall visual design. Second is interactive prototyping. Figma features in demonstrating the website's workflow by allowing us to define the behavior of elements like buttons and links to fully simulate the website experience. Canva, on the other hand, can only create simple links between pages. Third is version control. Figma automatically records a comprehensive history of all changes, while Canvas offers relatively basic version control functions. Ultimately, Figma is our final choice for this project because it provides all the features we need for professional web development.

As our final decision, Figma improved our work due to these features:

- **Real-time Collaboration and Efficiency**: Figma enabled us to work simultaneously in the cloud, with live editing and immediate feedback eliminating unnecessary back-and-forth communication. This transparency allowed everyone to see ongoing progress, making it easier to coordinate tasks and iterate on designs much faster.
- **Guaranteed Visual Consistency**: By utilizing shared elements, colors, and font styles, we successfully maintained a coherent and consistent visual language across the entire interface.
- **Streamlined Development Handoff**: One of the most practical benefits was the automatic generation of CSS code for design properties. This feature greatly enhanced our efficiency by simplifying the process of translating designs into front-end code.
- **Clear Accountability and Version Control**: The ability to tag team members in comments kept us aligned on assigned sections. Furthermore, the comprehensive version control function provided a clear history of all changes, ensuring accountability and allowing us to review progress at any stage.
- **Fostered Collaborative Ownership**: Ultimately, Figma didn't just reduce confusion and latency; it fostered a strong sense of shared ownership over the design. It ensured that our team worked as a cohesive unit, building a product with a unified vision.

HTML

We aim to provide a clean, accessible and responsive interface for medical users. To achieve this goal, we requires tools that help us build structured, semantic layouts while maintaining visual consistency across devices and screen sizes. When evaluating which front-end tools were most suitable for our design, we considered several combinations, including HTML with Bootstrap, HTML with TailwindCSS, and SCSS-based modular design systems. According to the specific characteristics of our project and our situation. We all agree that using HyperText Markup Language(HTML) and Cascading Style Sheets(CSS) was the most appropriate choice for our

needs after discussion. In the following section, we provide a brief introduction to these two technologies and explain how they contributed to the development of our prototype.

HTML is a necessary and standard core language that can be interpreted by any modern web browsers. In web development, HTML is used to arrange the content of the website and provide basic structural syntax for page design. As a fundamental tool, HTML has a relatively simple and easy-to-learn syntax, allowing developers to create websites with clear semantic structures that enhances both accessibility and user-friendliness. HTML wild ranges of labels, such as `<head>`, `<p>`, and `<div>`, which are used to organise the layout of a page. These elements support essential functions like hyperlinks, image embedding, list creation and tables formatting. With just basic syntax, developers can easily achieve a certain level of responsive functionality. Although various tools exists to assist in web development—such as page builders or frameworks—all of them ultimately compile down to HTML. As a result, HTML is irreplaceable in the front-end development process. Due to the situation our team has encountered, we all agree that the cost of learning alternative tools would beyond their benefits. Therefore, we decide to directly use plain HTML as our primary front-end tool. However, as mentioned above, HTML only provides basic structure and limiting styling capabilities. Managing layout and appearance directly within HTML would lead to inefficiencies and inconsistency. To address this issue, we employed CSS to handle visual design and layout, which will be discussed in next section.

CSS

One crucial issue in web development is how to manage code efficiently and make it maintainable. A widely adopted and effective solution is to separate the content, functionality and the style into distinct layers. While tools such as SCSS, Tailwind CSS and Bootstrap provide advanced styling features, we ultimately chose to use standard CSS for this project. This decision was based on the simplicity and clarity that CSS provides–especially valuable for a small team working on a focused prototype. CSS allows us to directly translate our Figma designs into code without introducing additional syntax or dependencies. This made it easier for all team members to understand and contribute to the styling process. Besides, one of the key advantages of CSS is its clear separation of content and presentation, which improves maintainability and enables us to update visual styles without modifying the fundamental structure. CSS also plays a crucial role in transforming our HTML structure into a clean, professional, and user-friendly interface. Furthermore, CSS enhances our collaboration by allowing us to define a shared layout and style system. Each members can easily apply consistent styles across pages, significantly reducing time spent on detail adjustments, and improving the efficiency of our workflow.

Spring Boot and Thymeleaf

Before selecting our technology stack for the Smart Hospital web application, we weighed a number of backend and frontend framework possibilities with consideration of equating development speed, maintainability, and integration efficiency. The top contenders included

Spring Boot with Thymeleaf, Node.js with Express and React, and Django with HTML templates. Table 3.1 summarizes the strengths and weaknesses of each stack based on thoroughly documented pros and cons according to credible sources.

Technology Stack	Advantages	Disadvantages
Spring Boot + Thymeleaf	• Single Java ecosystem simplifies development and reduces context switching. • Auto-configuration, embedded server (Tomcat), and REST API support simplify backend setup. • Native backend and frontend integration for faster delivery.	• Higher memory consumption compared to lightweight frameworks. • Less responsive UI compared to client-side frameworks like React.
Node.js + Express + React	• Well-suited for real-time and event-driven applications. • Large JavaScript ecosystem with reusable components. • Highly responsive UIs using React.	• Requires two codebases to be maintained for frontend and backend. • Steeper deployment complexity for integrated full-stack solutions
Django + HTML Templates	• Strong security features and fast development. • Built-in admin interface speeds up backend configuration. • Simple server-side rendering workflow.	• Requires Python expertise, which was outside our team's core competence. • Less native compatibility with Java-based tools and services.

Table 3.1: Technology Stack Alternatives Comparison

We weighed these options and opted to go with Spring Boot in the backend and Thymeleaf in the frontend. This Java-focused stack provided us with a consistent and streamlined development environment, which was particularly crucial for our small team of individuals working on tight deadlines. Because both frameworks are based on the Java ecosystem, we were able to build full-stack functionality in one integrated project. This reduced the complexity that would have otherwise been caused by having two separate frontend and backend codebases and improved coordination among team members.[7]

Spring Boot is a well-used enterprise-level framework that streamlines Java web application development. It provides an embedded Tomcat server, auto-configuration, and dependency management, allowing us to concentrate on core functionality instead of boilerplate setup.[2] Its innate RESTful API support enabled us to plan a scalable, modular backend architecture, with Controller classes as the basis for API endpoints, a core feature for connecting the client interface to backend services.

At the frontend, Thymeleaf was selected as a server-side templating engine that has native support for Spring Boot's MVC framework. Its dynamic binding of backend data in HTML templates ensured tight coupling of presentation logic with backend processes. This eliminated the need for extra JavaScript frameworks like React or Angular, reducing the learning curve and allowing the team to progress faster. Thymeleaf's human-readable syntax also made collaboration easier and the codebase more maintainable.[5]

Alternatives such as Node.js with Express and React were also considered, which would have offered highly interactive UIs and would have leveraged JavaScript's extensive ecosystem, but would also have meant two codebases to manage and deployment complexity.[7] Django with HTML templates, knowingly able to achieve rapid development and secure by design [2], was also a consideration but would have required the team to learn Python, an undesirable learning curve considering the time constraints we had.

Finally, our academic supervisor recommended picking up Spring Boot with Thymeleaf for its combination of performance, integration ease, and suitability for a Java-experienced team. This also simplified deployment, as both backend and frontend could be deployed together

in a single application. Moving further to the backend implementation phase, Spring Boot Controller classes will accept incoming HTTP requests and will form the basis of our API layer, which, later on, will be tested by tools discussed in the Backend Tools section.

3.1.3 Back-end Tools

3.1.4 Testing Tools

3.2 Front-end Design and Implementations

3.2.1 Early Stage-Virtual Hospital Africa(VHA)

During the initial discussions with the client, it was identified that the **Patient Intake page** contained too many information fields. This led to poor user experience, with many testers abandoning the form midway (Figure **??**). To address this, the client designed a second version of a multi-step form interface (Figure **??**). Finally, after our team's feedback, a third version was created (Figure **??**).

3.2.1.1 Key Improvements in Patient Intake Page

1. **Left-side navigation bar:** Enabled non-linear navigation, allowing users to jump to any section freely, improving flexibility and control.
2. **Optimized field grouping:** Consolidated fields such as name, gender, title, language, and ID number into a "General" group. Added profile photo upload with instant preview.
3. **Unified navigation buttons:** Replaced inconsistent buttons with clear *Back/Next*, improving predictability and reducing disorientation.

3.2.1.2 Patient Profile Page Improvements

1. **Enhanced patient information card:** Added profile picture, gender/age tags, patient status, quick contact and download buttons, and last edit time.
2. **Expanded information modules:** Unified the top tab bar and added a secondary menu (General, Primary Care, Contacts, Biometrics, Insurance).
3. **Editable forms:** Personal data presented in structured fields supporting direct editing.

3.2.1.3 Vitals Page Redesign

1. **Grouped layout:** Divided vital signs into "Required" and "Optional" sections, aligning with clinical practices.

2. **Optimized workflow:** Replaced single "Continue" button with *Back/Next*, allowing users better control.

3. **Integration with patient panel:** Displayed patient summary and treatment timeline alongside data entry.

These improvements adhered to usability principles such as *Match between system and the real world*, *Visibility of system status*, and *Recognition rather than recall*, reducing cognitive load, improving efficiency, and minimizing errors.

3.2.2 Smart Hospital

Design Process

From Concept to User Needs: Ideation and Story Mappingigma

Bridging Stories and Systems: Workflow Design

Visualizing Functionality: Prototyping with Figma

After we completed the workflow diagram, we used it as the foundation to divide tasks among team members. Each member is responsible for designing specific pages for different parts of the website while working all within a single file, shown as Figure 3.1, simplified cross-reviewing and allowed for seamless handoffs instead of manually sending assets or screenshots . Next, we will introduce our layout design.

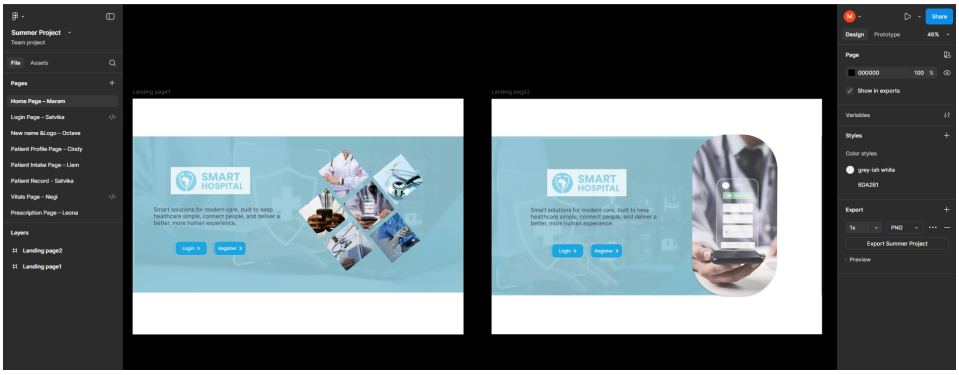


Figure 3.1: Shared Figma file displaying landing pages by Maram.

The primary goal of our layout is to ensure intuitive user interaction. For example when doctors are diagnosing a patient, they typically start by reviewing the patient's profile to gain a comprehensive understanding of the patient's condition. This information is displayed on the

"Patient Profile" page. At the same time, doctors may need access to the patient's visits history, so we design a clear and concise list that displays the date of each visit, a brief summary of the diagnosis, and the attending physician. If more detailed information is needed, doctors can click the "view" button to open a pop-up window showing the full report. Another example is during the diagnosis process–doctors may want to take temporary notes before writing the final prescription or summary. To support this, we included a text area on the "Clinical Notes" page where doctors can write their notes down. These features are all designed to enhance the experience for medical professionals interacting with our platform.

Color selection plays a vital role in website design. In our project, we choose a range of blue tones as the primary color scheme, as blue often represents inclusiveness and trust. To make the overall atmosphere feel more approachable and friendly, we incorporate some earthy tones into the blue. Meanwhile, white and gray is used for elements like buttons and text areas to create a clean and orderly appearance. Through these color combinations, we aim to convey a sense of stability and warmth to our users. The overall visual design reflects the spirit of our platform, to serve as a strong and trustworthy support for users of all ages, genders, physical conditions and cultural background.

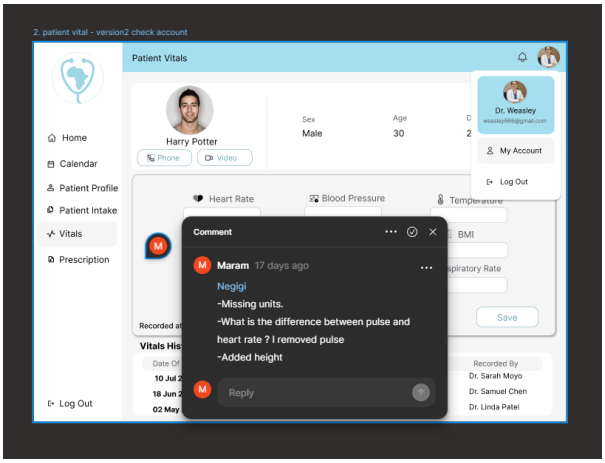
We continually review each other's work using the comment feature which is one of the most useful features we employed, to ensure real-time feedback and update.Throughout the development of each screen, we actively left comments for one another, either suggesting improvements or pointing out issues to be addressed. These comments were visible directly on the relevant design elements, helping us avoid miscommunication and track suggested changes efficiently.

We often initiated discussions either directly within Figma or first during our meetings, and then documented the agreed-upon feedback using the comment feature to ensure accountability. This helped prevent any feedback from being forgotten and gave each designer a clear to-do list for updates. For instance, comments could be left directly on the design (Figure 3.2(a)) or used to track resolved issues after a meeting (Figure 3.2(b)).

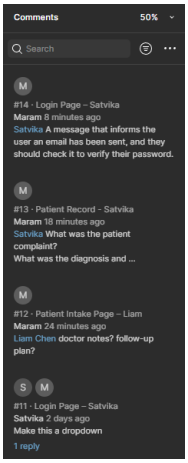
To evaluate whether the designs meet our requirements, we test various user scenarios using Figma's prototyping feature to link different steps and assess the overall flow. Also, we shared links to Figma with our academic supervisor during our weekly review sessions so that they could view the prototype live. During these review sessions, we made use of the "Follow" feature so that we were all concentrated on the same element of the design so that it was simpler for the supervisor to provide targeted, contextual feedback on specific areas.

This direct communication also allowed us to quickly clarify and allowed us to reduce uncertainty about comments such as "this needs to be more visible" or "consider redesigning this layout."

Moreover, Figma's version history feature allowed us to continuously improve our design by reflecting on what worked and what didn't. Although we didn't manually take snapshots, Figma



(a) Providing feedback via comments.



(b) Tracking and resolving design feedback.

Figure 3.2: Using Figma's comment feature for collaborative design reviews.

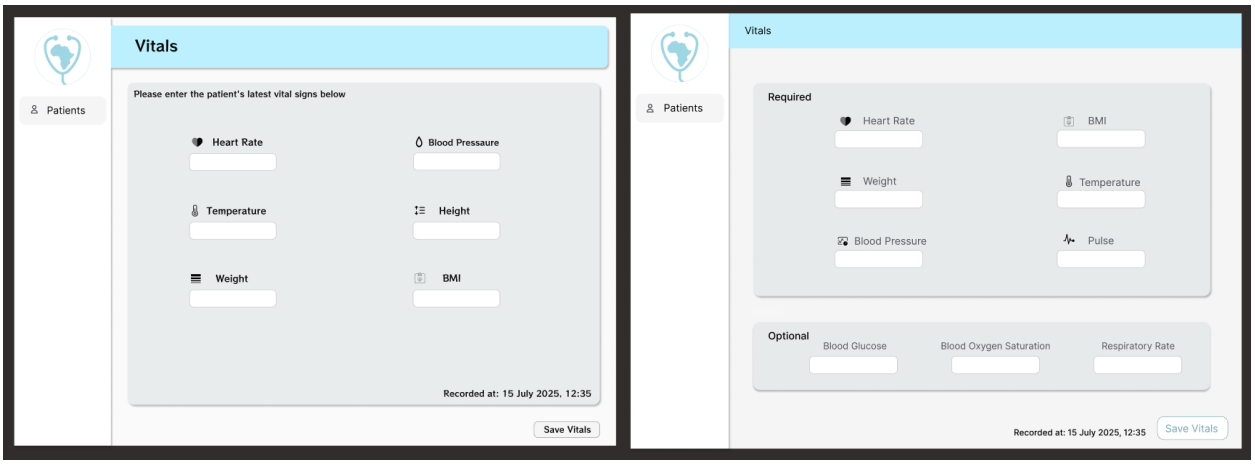
automatically saved our progress, which made it easy to revisit previous versions at any point. This proved especially helpful when comparing earlier designs with more refined iterations, as it allowed us to clearly see what had improved and what needed further adjustment.

A great example of this is the evolution of the Vitals page, which went through at least six major visual and functional changes, shown in Figure 3.3. In the earliest version, the layout lacked structure, had no sidebar for navigation, and missed key interactive elements such as the "Save Vitals" button, as shown in Figure 3.3(a). As feedback was gathered through team discussions and supervisor input, we iteratively refined the page. By the sixth version (Figure 3.3f), the design included:

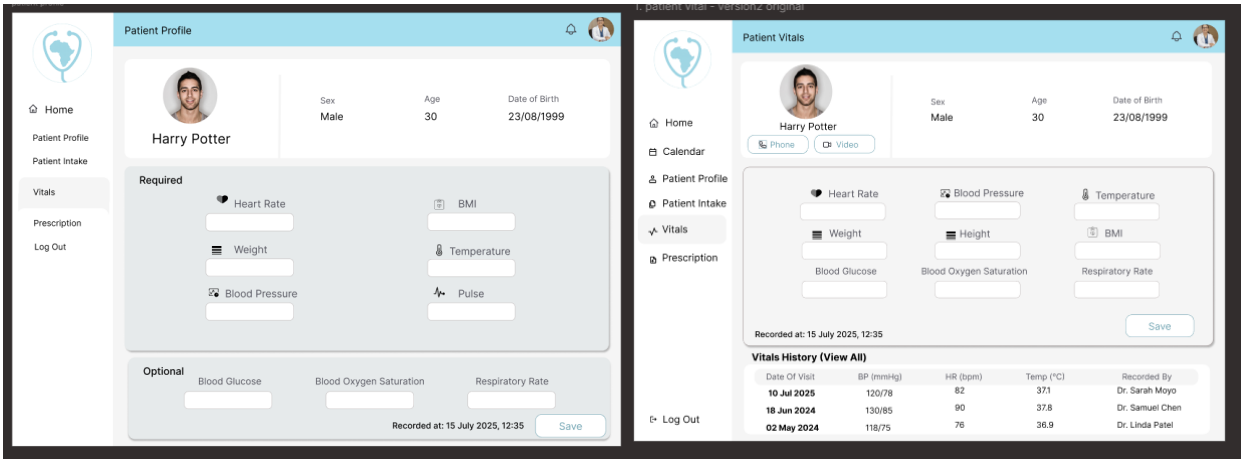
- An aligned, visually consistent layout that matches the rest of the system.
- A patient card was added at the top to remind clinicians whose record they were updating.
- A "Save Vitals" button to improve usability.
- A vitals history table, allowing easy access to previously recorded data.


Vitals
Heart Rate
Blood Pressaure
Temperature
Height
Weight
BMI
Vitals
Please enter the patient's latest vital signs below
Heart Rate
Blood Pressaure
Temperature
Height
Weight
BMI
Recorded at: 15 July 2025, 12:35
Save Vitals

(a) (b)


Patients
Vitals
Please enter the patient's latest vital signs below
Heart Rate
Blood Pressaure
Temperature
Height
Weight
BMI
Recorded at: 15 July 2025, 12:35
Save Vitals
Patients
Vitals
Required
Heart Rate
BMI
Weight
Temperature
Blood Pressure
Pulse
Optional
Blood Glucose
Blood Oxygen Saturation
Respiratory Rate
Recorded at: 15 July 2025, 12:35
Save Vitals

(c) (d)



(e) (f)

Figure 3.3: The iterative design process of the Vitals page, showing its evolution through six key versions. (a)Basic layout without a sidebar or patient card. (b)Introduction of a navigation sidebar. (c)Adding a patient information card at the top. (d)Refining the layout and adding a history table. (e)Improving visual consistency and button placement. (f)Final design with all key components integrated.

These changes not only enhanced the interface visually, but also made the page much more usable in a real clinical workflow. Since healthcare professionals often move quickly between patient records, having the patient card consistently displayed at the top of all patient-specific pages helps reduce confusion, reminding them of whose data they are viewing or editing.

We made multiple adjustments to several pages based on weekly feedback from our supervisor and internal team reviews. This ensured a more user-friendly and functional design across the platform.

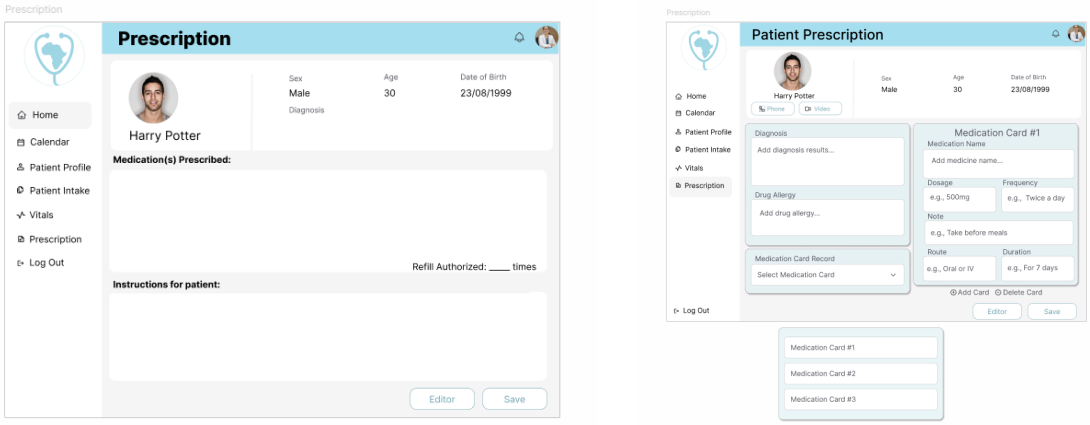
One of the most significant design iterations in our project involved the Prescription Page. During a weekly meeting with our supervisor, we received feedback that the initial layout appeared empty and lacked the essential clinical details typically expected in a real-world healthcare system. Based on this feedback, we conducted research into existing healthcare system standards to understand what prescription interfaces should contain, with a particular focus on NHS guidelines and clinical UX principles.

We referred to the *Medication and Allergy Standards for structured records*, published by the Royal College of Physicians in collaboration with the Health and Social Care Information Centre (HSCIC) in 2013. These standards recommend that digital prescriptions should include key information such as medication name, dosage, frequency, route, duration, and notes, all of which are fields we implemented in our updated interface [6].

In terms of user interface design, we followed the NHS Design System's guidance on card components, which recommends grouping related form fields into visually distinct blocks. This improves usability and reduces cognitive load by allowing clinicians to focus on one section at a time. Inspired by this, we introduced structured "Medication Cards" in our interface, each

containing fields for medication name, dosage (e.g., 500mg), frequency (e.g., twice a day), route (e.g., oral or IV), duration (e.g., for 7 days), and clinical notes (e.g., take before meals). We also added a dropdown for selecting saved medication records, improving both usability and completeness of the page [1].

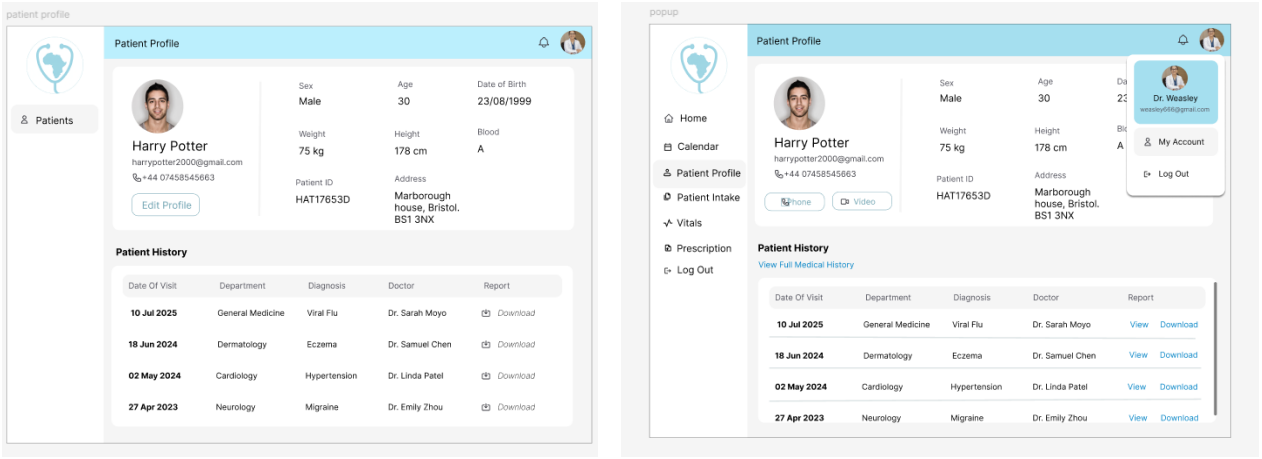
These changes transformed the original sparse interface into a more complete, user-friendly, and clinically appropriate design. They also align our application with industry standards for both content structure and interface design, as shown in Figure 3.4.



(a) (b)

Figure 3.4: Comparison of the Prescription Page before and after design iteration based on supervisor feedback. (a)Initial design of the Prescription Page. (b)Redesigned page with structured "Medication Cards" and comprehensive fields, following NHS guidelines.

Similarly, during one of our internal design review meetings, we identified a gap in the Patient Profile Page. Initially, the Patient Profile Page featured the doctor's profile picture, but it lacked interactivity. In a later iteration, we enhanced the design by implementing a functional popup menu. This allowed doctors to either access their account settings or securely log out, improving both usability and navigation consistency, as shown in Figure 3.5. These feedback-driven design iterations not only improved usability but also aligned the interface with real-world clinical workflows and recognized healthcare UI standards.



(a) (b)

Figure 3.5: Evolution of the doctor's profile menu on the Patient Profile Page. (a)Initial design with a static, non-interactive doctor profile picture. (b)Enhanced design with a functional popup menu for account settings and logout.

Front-end Implementation

In the front-end implementation section, we illustrate the process of how we created, organised, and tested our front-end code. We divide this process into 5 stages. First, we describe how we structured and developed the codebase. Then, we explain how we integrate shared layouts using Thymeleaf to improve consistency and maintainability across pages. Next, we explain how different pages are connected to form a smooth user flow. We also discuss the steps we took to test each function to ensure the website behaves as intended. Finally, we reflect on the main challenges we encountered during the front-end development and how we addressed them. These topics are covered in following subsections: Code Structure and Component Organisation, Integrating Shared Layouts with Thymeleaf, Page Navigation and User Flow, User Flow and Functional Testing, and Challenges Encountered During Frontend Development, respectively.

Code Structure and Component Organisation

In this section, we describe how we constructed the front-end codebase. As mentioned earlier, each member was responsible for designing different parts of the website in Figma. We continued the pattern during implementation–each member developed their assigned parts using HTML and CSS, focusing on accurately reproducing the layout and visual style from our Figma design.

At this stage, our goal is to replicate the layout and visual style of our Figma design. Most interactive elements, such as buttons, pop-up windows, and form submissions, have not yet been functionalized. We have also not yet integrated a shared layout system using Thymeleaf. These shared elements are currently duplicated across individual pages, and we plan to organize them in the next development stage to improve maintainability and consistency.

Integrating Shared Layouts with Thymeleaf

After developing all the pages, we noticed that many structural elements–such as headers, sidebars, and forms–were repeated across different pages. To reduce redundancy and to make the Smart Hospital UI consistent and maintainable, we wrapped repeated interface components (sidebar, top navigation, and patient header) into reusable Thymeleaf fragments and consumed them across pages with th:replace. As shown in Figure 3.6, the vitalsHistoryPage.html file defines an empty container, CSS links, outer grid, and main content area, while common UI components are injected at render time from layouts/sharedLayout.html.

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0"/>
    <title>Vitals History</title>
    <link rel="stylesheet" th:href="@{/css/sharedLayout.css}">
    <link rel="stylesheet" th:href="@{/css/vitalsHistoryPage.css}">
</head>
<body>

<div class="layout-container">
  <!-- Sidebar -->
  <aside th:replace="layouts/sharedLayout :: vitalsSidebar"></aside>

  <!-- Main Content Area -->
  <div class="main-content">
    <!-- Top Navigation -->
    <header th:replace="layouts/sharedLayout :: topNav"></header>

    <!-- Vitals History Content -->
    <div class="vitals-content">
        <!-- Patient Info Card -->
        <section th:replace="layouts/sharedLayout :: patientCard"></section>

        <!-- Vitals History Section -->
        <div class="vital-history">
            <div class="history-header">
                <div class="header-left">
                    <button class="back-btn" onclick="history.back()">
```

Figure 3.6: vitalsHistoryPage.html consuming shared fragments (vitalsSidebar, topNav, patientCard) via th:replace.

For example, the Vitals page loads the active vitalsSidebar fragment (highlighting the Vitals tab), the topNav fragment (with dynamic pageTitle provided via the model), and a patientCard fragment (so clinicians can always see the patient context at the top of patient-specific pages). This server-side composition makes our design system dry, updating the sidebar icon set or the patient header once applied everywhere without touching individual pages. Figure 3.7 illustrates part of the sharedLayout.html file, displaying the sidebar fragment and its logo and most crucial navigation links.

```
<!-- Logo at the top of the sidebar -->
<img th:src="@{/SharedLayoutImages/LogoCircle.png}" alt="Logo" class="logo">

<!-- 🔸 Navigation Links -->
<nav class="sidebar-nav">
  <div class="tab" data-page="home">
    <a href="/dashboard">
      <img th:src="@{/SharedLayoutImages/homeIcon.svg}" alt="Home Icon" class="tab-icon">
      <span>Home</span>
    </a>
  </div>

  <div class="tab">
    <a href="/patientProfile">
      <img th:src="@{/SharedLayoutImages/profileIcon.svg}" alt="Profile Icon" class="tab-icon">
      <span>Patient Profile</span>
    </a>
  </div>

  <div class="tab">
    <a href="/patientintake">
      <img th:src="@{/SharedLayoutImages/intakeIcon.svg}" alt="Intake Icon" class="tab-icon">
      <span>Patient Intake</span>
    </a>
  </div>

  <!-- Active tab styling applied here -->
```

Figure 3.7: Snippet from sharedLayout.html showing the sidebar fragment, including the logo at the top and navigation links.

We also separated shared and page-specific styling concerns to avoid cascade conflicts. Fragment-level styles for global pages live in `sharedLayout.css` and are loaded ahead of each page's own stylesheet (e.g., `vitalsHistoryPage.css`). Fragments leverage Thymeleaf's URL syntax (e.g., `th:src="@{/SharedLayoutImages/}"`) to refer to assets so paths get resolved properly no matter the environment. Where a page needs alternative "active" navigation, we provide a purpose-specific fragment (e.g., `sidebar` vs. `vitalsSidebar`) to facilitate correct tab state without hardcoded per-page logic. This approach assisted in achieving consistency (the same chrome everywhere), maintainability (one change affects all pages), and clarity. As shown in Figure 3.8, clinicians always have navigation context and the patient card visible when navigating between modules.

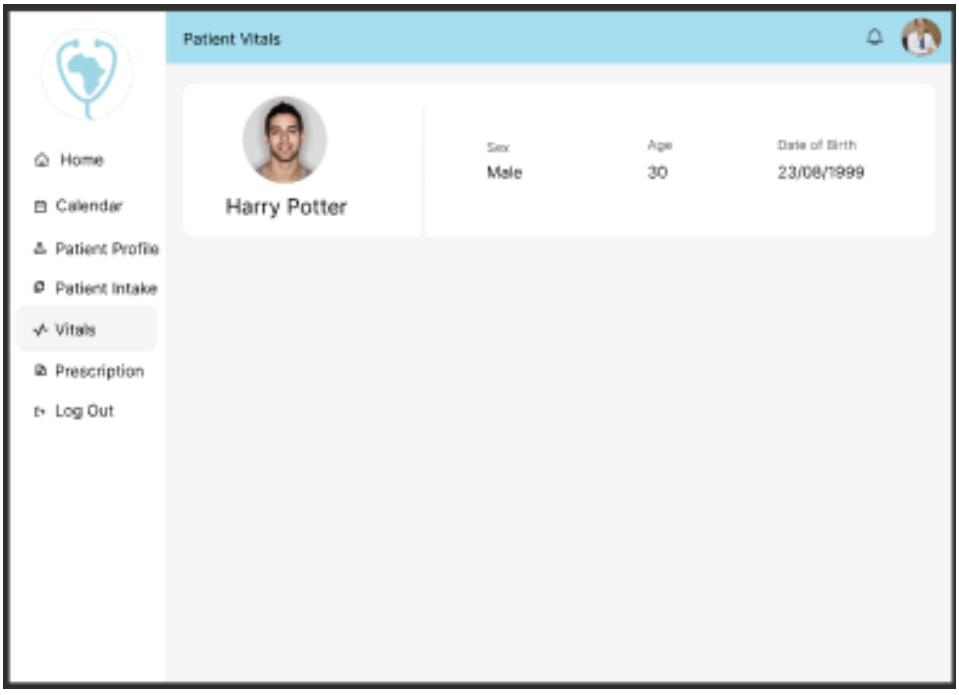


Figure 3.8: Rendered UI in the browser showing the shared sidebar, patient card, and top navigation in the Vitals module.

Page Navigation and User Flow

In this stage, we began to functionalize interactive elements such as buttons, hyperlinks to ensure the website provides a smooth and intuitive experience. The overall user flow had already been defined in our early-stage Figma design, a clear user flow that reflects typical interactions between patients and doctors in our early-stage Figma design, reflecting typical interactions between patients and doctors. Each page was connected according to the logical steps users would take when navigating through the platform. For example, clicking the "Register" button on the homepage directs users to the registration form, and buttons on the "Dashboard" page are linked to corresponding pages. These workflows were clearly visualized in our Figma prototype. The main goal of this stage is to ensure that all the interactive elements function as the origin design. In the next stage, we plan to test the whole website work appropriately and collect the disadvantages that can be improved.

User Flow and Functional Testing

Challenges Encountered During Frontend Development

One of the biggest challenges encountered while performing frontend development was how to break down the Figma-based design system into Thymeleaf templates. Even though the Figma designs created a solid visual reference for layout, typography, and spacing between

elements, the server-side rendering approach of Thymeleaf required the interface to be split into smaller, more manageable parts. It introduced additional difficulty in the process of converting pixel-perfect designs into coherent HTML templates with alignment, responsiveness, and style consistency preserved. Visual parity usually needed to be delivered by iterative adjustments of the HTML structure as well as accompanying CSS.

An analogous problem involved the handling of shared styles and bits within the multi-page Thymeleaf application. Shared interface pieces, the sidebar navigation, header, and footer, were included as reusable fragments to foster consistency and reduce redundancy. With this came, however, the risk of side effects: a modification of a shared fragment would likely disrupt the styling or layout of multiple pages. To tackle this, the development team implemented a formalized update process, such as fragment-specific testing and peer review prior to merge of changes. This procedure ensured global component updates maintained visual correctness without causing regressions in other parts of the system.

3.3 Back-end Design and Implementations

Chapter 4

Evaluation and Testing

Begins a chapter. Example: When the beloved cellist (Christopher Walken - outstanding) of a world-renowned string quartet receives a life-changing diagnosis, the group's future suddenly hangs in the balance: suppressed emotions, competing egos and uncontrollable passions threaten to derail years of friendship and collaboration. Featuring a brilliant ensemble cast (including Philip Seymour Hoffman, Catherine Keener and Mark Ivanir as the three other quartet members), it is a fascinating look into the world of working musicians, and an elegant homage to chamber music and the cultural world of New York. The music, of course, is ravishing (the score is the work of regular David Lynch collaborator Angelo Badalamenti): A Late Quartet hits all the right notes.

Chapter 5

Conclusion

Begins a chapter. Example: When the beloved cellist (Christopher Walken - outstanding) of a world-renowned string quartet receives a life-changing diagnosis, the group's future suddenly hangs in the balance: suppressed emotions, competing egos and uncontrollable passions threaten to derail years of friendship and collaboration. Featuring a brilliant ensemble cast (including Philip Seymour Hoffman, Catherine Keener and Mark Ivanir as the three other quartet members), it is a fascinating look into the world of working musicians, and an elegant homage to chamber music and the cultural world of New York. The music, of course, is ravishing (the score is the work of regular David Lynch collaborator Angelo Badalamenti): A Late Quartet hits all the right notes.

Chapter 6

Reference

Appendix A

Appendix A

Begins an appendix

Bibliography

[1] N. Digital, *Nhs service manual – card component.*
https://service-manual.nhs.uk/design-system/components/card, 2023.
Online.

[2] GeeksforGeeks, *Django vs spring boot.*
https://www.geeksforgeeks.org/blogs/django-vs-spring-boot/, 2025.
Online.

[3] F. E. M. Hospital, *Official line account.*
https://line.me/R/ti/p/@femh, 2025.
Screenshot taken by the author.

[4] International Labour Organization, *Decent work for rural health workers.*
https://www.ilo.org/global/topics/economic-and-social-development/rural-development/WCMS_768518/lang--en/index.htm, 2021.
Accessed: 2025-08-02.

[5] JavaGuides, *Thymeleaf vs react js.*
https://www.javaguides.net/2024/08/thymeleaf-vs-react-js.html, 2024.
Online.

[6] R. C. of Physicians, Health, and S. C. I. C. (HSCIC), *Medication and allergy standards: Structured record standards for the nhs.*
https://developer.nhs.uk/apis/gpconnect/pages/accessrecord_structured/use_cases/Medication_record.pdf, 2013.
Online.

[7] Simplilearn, *Node.js vs spring boot.*
https://www.simplilearn.com/node-js-vs-spring-boot-article, 2025.
Online.

[8] N. Trust, *Digital primary care: Where next?*
https://www.nuffieldtrust.org.uk/research/digital-primary-care-where-next, 2022.

Accessed: 2025-08-02.

[9] United Nations, *Ensure healthy lives and promote well-being for all at all ages.* `https://sdgs.un.org/goals/goal3`, 2022.
Accessed: 2025-08-02.