CI601 The Computing Project

DentalConnect: A web application for aggregated dental appointments.

Daniel Bennett

Student Number: 22860985
Supervisor: Jennie Harding
Second reader: Thar Shamsa
Word Count: 10904

Abstract

The UK's dental care crisis, caused by a severe shortage of NHS dental appointments, prevents patients from accessing timely care, leaving them to either seek private dentistry or do nothing, which fuels a growing public healthcare issue. Patients face significant barriers to private dentistry, primarily cost, with no centralised platform to compare appointment and treatment prices across practices. Additionally, there is no aggregated solution for finding last-minute or emergency appointments located nearby and available at short notice. Independent dental practices also lack a platform to list unused appointments, leading to operational inefficiencies and underutilisation of appointments. This project addresses these challenges by developing a web application that enables dental practices to list their unused appointments, providing an aggregated solution for patients to search, filter and book appointments. The web application implements a three-tier architecture, leveraging TypeScript, React, Next.js, Hasura GraphQL Engine and a PostgreSQL database. The project was managed following Agile Feature-Driven sprints, focusing on delivering user-facing functionality while prioritising a Minimum Viable Product (MVP). All Must-Have functional and non-functional requirements were implemented and tested successfully, along with several Should-Have features, forming a complete and validated MVP. While the solution is limited to private dentistry and does not address systemic NHS issues or fully eliminate affordability barriers, it provides a scalable foundation for improving access and efficiency in the private dental sector.

Table of Contents

System Access Information

Web Application URL: https://ci601.vercel.app/home

GitHub Repository: https://github.com/danbennett239/CI601

Role	Email	Password
User (Patient)	user@dentalconnect.com	39N)+e2kL?
Admin	admin@dentalconnect.com	nyTL2(397!
Verified Practice	vp@dentalconnect.com	LY23C6YnT£
Unverified Practice	uvp@dentalconnect.com	B4P6Gk3+2?

Refer to the project's GitHub repository ReadMe for detailed local development environment setup instructions. The following environment variables must be defined in a .env file:

Variable	Value
HASURA_GRAPHQL_URL	http://localhost:8080/v1/graphql
HASURA_ADMIN_SECRET	myadminsecretkey
NEXT_PUBLIC_BASE_URL	http://localhost:3000
JWT_SECRET	this_is_a_very_secure_secret_key_123456
MAILJET_API_KEY	eb63f348dfc60a8a1ce38a9fbf1e74cb
MAILJET_API_SECRET	71c28e89d78b7629bcf11cdcd1e8eddd
MAILJET_SENDER_EMAIL	danbennettuni@outlook.com
AWS_ACCESS_KEY_ID	AKIARYEUCXV7ELEH45XB
AWS_SECRET_ACCESS_KEY	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
AWS_REGION	eu-north-1
AWS_S3_BUCKET_NAME	ci601-dental-practice-images
GOOGLE_MAPS_API_KEY	AIzaSyAPopYtvDqAm-IYz47tHaLnOUb00fVsrFY

Ensure running local development on port 3000 as third-party integrations are URL scoped.

1. Introduction

1.1 Problem Statement

Access to timely dental care in the UK has significantly declined due to the lack of NHS dental appointments, creating a mounting public healthcare issue and driving patients to seek private dental care. For patients, cost is the largest barrier to private dental care, with no centralised platform available to compare pricing of available appointments across practices. In addition to cost, patients also face difficulties in securing last-minute or emergency appointments that are affordable, available at short notice and geographically convenient. Meanwhile, independent private dental practices lack a unified digital solution for listing unused appointments, leading to inefficiencies. Existing solutions, such as Bupa, only provide an online aggregated appointment booking service for practices within their network and do not provide clear, easily comparable information about appointment availability, pricing and location.

1.2 Solution Approach

To address the problem outlined in Section 1.1, this project focuses on the development of a web application that enables dental practices to list unused appointments, whilst providing patients with a centralised, user-friendly and mobile-responsive solution to search, compare and book appointments. By improving appointment visibility and accessibility, the web application aims to optimise appointment utilisation for practices, while also improving appointment availability and reducing access barriers for patients. The solution cannot eliminate financial barriers for all patients or resolve systemic NHS issues, but it provides a step towards improving access and efficiency within private dental care.

1.3 Project Scope

The project is limited to the development of a web application for listing, searching and booking dental appointments. Certain functionality has been excluded from the project scope to ensure feasibility with the available resources and project timeframe. Specifically, the web application will not integrate with pre-existing third-party dental appointment booking systems and will not provide health care advice, treatment recommendations or guidance through contact with healthcare professionals or AI-driven agents.

2. Project Aim, Objectives and Deliverables

2.1 Project Aim

The aim of this project is to develop a web application that improves access to private dental care through enabling users to search, compare and book appointments online, and optimising appointment utilisation for independent private dental practices. The project's aim was shaped through research into the decline of NHS dental availability, barriers to private dental care, inefficiencies in private dentistry, and gaps identified in existing solutions.

2.2 Project Objectives

To achieve the aim of developing a web application that enhances access to private dental care and optimises appointment utilisation, six objectives were defined. These objectives, informed by the research detailed in Section 6, translate the project's aim into actionable software development goals. Objectives O1 through O3 focus on the core functionality of the web application, while objectives O4 to O6 outline technical considerations and best practices the web application must follow to ensure an accessible, user-friendly, and maintainable solution. Collectively, these objectives provide a structured foundation for defining the project's requirements, ensuring alignment with the project's aim, and guiding both design and implementation.

- **O1: Develop a Booking System**: Enables dental practices to create and manage appointments, and allows users to book available appointments and receive instant confirmations.
- **O2: Implement User and Practice Login and Registration**: Create secure login and registration functionality for both users and dental practices. Users will be able to register and sign in, manage their accounts, preferences, and bookings. Practices can register and sign in, manage their practice details, preferences and appointment availability.
- **O3: Create Filterable Appointment Listings**: Build a filterable and sortable list of available appointments. Users will be able to filter appointments based on appointment type, price, location and date, and sort based on time, price or distance to find a suitable appointment quickly.
- **O4: Ensure Responsive Design**: Develop the application to be fully accessible, user-friendly and optimised for a range of devices, including mobiles and desktops.
- **O5: Evaluate Usability and Accessibility**: Conduct usability and accessibility testing to ensure that the application meets user needs and adheres to accessibility standards.
- **O6: Implement CI/CD Deployment Pipeline**: Set up a Continuous Integration and Continuous Deployment (CI/CD) pipeline to automate testing and deployment, ensuring consistent and reliable application updates throughout development while maintaining high availability for users.

2.3 Project Deliverables

The deliverables provide tangible outputs required to achieve the project's aim and objectives.

- **Web Application**: Functional web application for listing, searching and booking dental appointments.

- **Wireframe User Interface Designs**: Low-fidelity UI designs to guide development (Appendix D).

- **Technical Documentation**: System architecture, implementation details, API and service references, and setup instructions (Documented through the report and in Appendix E).

- **Testing Documentation**: Test cases, results, and validation of functional and non-functional requirements (Documented through the report and in Appendix G).

3. Project Requirements

Following the definition of the project objectives, requirements were identified to guide the development of the web application. Requirements can be defined as specifications that a system must satisfy to fulfil business objectives and user expectations (Wiegers & Beatty, 2013).

3.1 Requirement Gathering and Prioritisation

To ensure that the requirements aligned with the project's aim, the Minimum Viable Product (MVP) model was applied to define the critical functionality for the web application. The MVP focuses on delivering core functionality that meets the essential needs of users, allowing for early validation of the product with minimal resources (Ries, 2011). In this project, the MVP specifically focuses on delivering the features required for users to search, compare and book dental appointments, as well as enabling dental practices to create and manage appointments.

The MoSCoW prioritisation technique was applied alongside the MVP model to categorise the functional and non-functional requirements into prioritised groups based on their importance for the project's success (Clegg and Barker, 1994). This assures the implementation of the most critical requirements first, with additional requirements addressed if time and other resources permit (Stapleton, 2003). The four MoSCoW prioritisation categories are defined as follows (Ahmad et al., 2017):

- **Must-Have**: Requirements that must be included in the final software product.
- **Should-Have**: High-priority requirements that should be included, if possible, within the delivery time frame.
- **Could-Have**: Desirable or nice-to-have requirements that could be included without incurring too much effort or cost.
- **Won't-Have**: Requirements that stakeholders want to have, but all the stakeholders have the consensus that the requirements will not be implemented in the current version.

The complete list of functional and non-functional requirements, detailed in Sections 3.2 and 3.3, was derived from expanding upon the project's objectives and through further research into appointment types and durations, competitor applications and conducting questionnaires, detailed in Section 6. The Must-Have requirements represent the MVP, while the Should-Have and Could-Have requirements constitute valuable enhancements if time and resources allow. The Won't-Have requirements identify features that fall outside the current scope, but would be suitable in future iterations of the web application.

The combination of the MVP and MoSCoW methodologies enabled the creation of well-defined, prioritised requirements that align with the project's aim, scope, and timeframe. This approach was particularly suited to the project due to development time constraints associated with working as a solo developer, requiring focus on delivering core functionality.

3.2 Functional Requirements

The functional requirements for the web application are listed below, prioritised using MoSCoW and linked to project objectives from Section 2.2.

Requirement ID	Description	MoSCoW Priority	Linked Objective(s)
FR1	**Booking System**: Enable users to book appointments and receive confirmations.	Must-Have.	O1
FR2	**User Registration**: Allow users to create an account.	Must-Have.	O2
FR3	**User Login**: Enable users to log in securely to access and manage their accounts	Must-Have.	O2
FR4	**Practice Registration**: Allow dental practices to sign up securely.	Must-Have.	O2
FR5	**Practice Login**: Enable dental practices to log in securely to access and manage their accounts.	Must-Have.	O2
FR6	**Appointment Search**: Display available appointments with filters (distance, date, price) and sort options.	Must-Have.	O3
FR7	**Appointment Management**: Practices can create and manage appointments with dynamic durations, supporting the selection of one or more appointment types: Check-Up, Cleaning, Whitening, Filling, Extraction and Emergency.	Must-Have.	O1, O3
FR8	**Admin Account**: Admin users can log in and access administration features	Must-Have.	O2
FR9	**Practice Verification**: Admin can verify registered practices.	Must-Have.	O2

FR10	**User Profile Management**: Allow users to manage their details and preferences.	Must-Have.	O2
FR11	**Practice Profile Management**: Allow practices to manage their details and preferences.	Must-Have.	O2
FR12	**Notifications for New Appointments**: Notify users via email or text when appointments become available.	Should-Have.	O3
FR13	**Review and Feedback System**: Enable users to leave reviews and feedback about their experiences.	Should-Have.	O1
FR14	**Basic Analytics**: Provide dental practices with simple metrics to overview their performance.	Should-Have.	O2
FR15	**Automatic Location Detection**: Automatically detect and apply the user's location for appointment search.	Should-Have.	O3
FR16	**Email Integration**: Integration with a third-party email provider for "Forgot password" and email verification.	Should-Have.	O2
FR17	**Practices Invitations**: Practices can invite additional users to their practice.	Should-Have.	O2
FR18	**Dispute Dashboard**: Admin can view disputes raised by users.	Should-Have.	O2
FR19	**Appointment Disputes**: Users can raise disputes related to appointments.	Should-Have.	O1
FR20	**Application Status**: Practices can view the status of their application.	Should-Have.	O2
FR21	**Map View**: Display available appointments on an interactive map for easier navigation.	Could-Have.	O3

FR22	**Calendar Integration**: Allow users to add bookings to third-party calendars.	Could-Have.	N/A
FR23	**Chatbot Support**: Provide basic assistance for common questions or issues.	Could-Have.	N/A
FR24	**Public API**: Provide and document an API that allows third-party applications to access appointment data.	Could-Have.	N/A
FR25	**API Key Generation**: Enable practices to generate an API key securely.	Could-Have.	N/A
FR26	**Third-Party Authentication**: Integration with third-party authentication providers (e.g., Google).	Could-Have.	O1

Table 3.1: Functional Requirements with MoSCoW Priorities and Linked Objectives

The project's Won't-Have requirements are detailed in Appendix H.1. A table indicating functional requirement implementation status is in Appendix F.1.

3.3 Non-Functional Requirements

The non-functional requirements for the web application are listed below, prioritised using MoSCoW and linked to project objectives from Section 2.2.

Requirement ID	Description	MoSCoW Priority	Linked Objective(s)
NFR1	**Accessible and Responsive Design**: Ensure the website is fully accessible and user-friendly on both desktop and mobile devices.	Must-Have.	O4, O5
NFR2	**CI/CD Pipeline**: Set up an automated pipeline for testing, building, and deploying updates.	Must-Have.	O6
NFR3	**Performance**: The system must respond to user actions (e.g., booking, filtering) within 2 seconds.	Must-Have	O4, O5
NFR4	**Security (Password Encryption)**: All user and practice passwords must be stored securely using encryption (e.g., bcrypt).	Must-Have.	O2
NFR5	**Developer Alerting**: The system should notify developers via email of failed deployments or test runs.	Should-Have.	O6
NFR6	**Availability**: The system must maintain 99.9% uptime, with justification provided in Appendix H.2.	Should-Have.	O6

Table 3.2: Non-Functional Requirements with MoSCoW Priorities and Linked Objectives

A table indicating non-functional requirement implementation status is in Appendix F.2.

4. Methodology

4.1 Project Management Approach

Choosing an appropriate project management methodology is essential for a project to be managed and completed successfully (Project Management Institute, 2010). Waterfall and Agile, two widely used methodologies, were compared to evaluate their suitability for this project.

Waterfall, introduced by Royce (1970), is a linear approach to project management where each of the five phases progresses sequentially: requirements, design, implementation, verification and maintenance. Each phase must be completed before moving on to the next stage (Royce, 1970; Atlassian, 2024b).

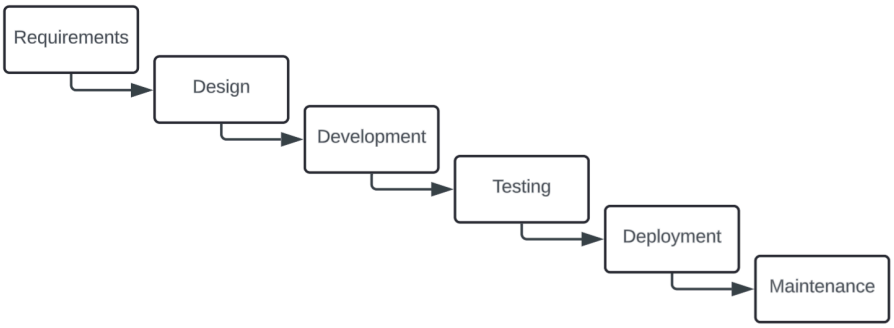


Figure 4.1: The Waterfall Model

Agile emphasises a flexible and iterative delivery through an incremental process, designed to be adaptable to dynamic requirements while delivering functional increments throughout the project lifecycle (Agile Alliance, 2001; Atlassian, 2024a).

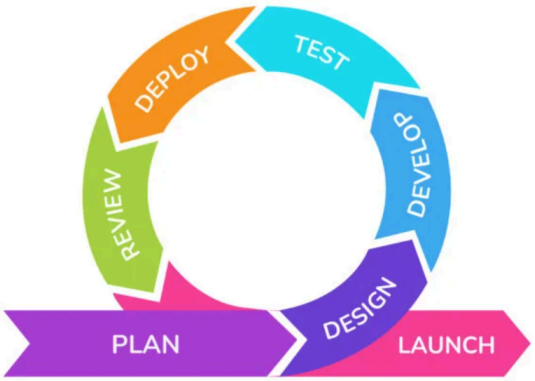


Figure 4.2: Agile Development Cycle (Krasamo, 2022)

Agile was selected as the project management approach as it is better suited for quickly delivering an MVP and will better accommodate the project's evolving requirements (Highsmith, 2009). Furthermore, adhering to biweekly sprints and an iterative approach enables flexibility, allowing for the de-scoping and reprioritisation of requirements frequently (Cohn, 2009). Additionally, with a sole developer on the project, the adaptability provided reduces the impact of scope creep and technical challenges, which are likely to arise when working with new technologies (Dybå and Dingsøyr, 2008; Moyo and Mnkandla, 2020). The sequential nature of the Waterfall methodology would amplify these issues, further reinforcing it as an inappropriate approach.

4.2 Sprint Planning and Project Plan

Following the selection of Agile as the project management methodology, development was structured into sprints following the Feature-Driven Development (FDD) approach. FDD is an iterative approach that focuses on delivering frequent, fully functional client-facing features (Coad, Lefebvre and De Luca, 1999). The approach was chosen for this project as it aligns with an Agile workflow and is suitable for solo development, as it emphasises delivering the core, working functionality of a feature as quickly as possible (Stanke, 2024).

Functional and Non-Functional requirements, identified in Section 3.2 and 3.3, were grouped into features. Each feature contained a mixture of Must-Have, Should-Have, and Could-Have requirements, enabling flexibility in de-scoping lower-priority requirements in cases of time or resource constraints. Each feature became a sprint within the project plan, with the features required for the MVP prioritised first. A total of 9 sprints were defined, each lasting two weeks, which provided frequent intervals for reviewing progress and performing reprioritisation. Figure 4.3 outlines the feature of each sprint, with the commensurate requirements.

Sprint	Feature	Requirement(s)
0	Project Setup	NFR1, NFR2, NFR4
1	User Login & Registration	FR2, FR3, FR10, FR16
2	Practice Registration & Verification	FR4, FR5, FR8, FR9, FR11
3	Appointment Creation	FR7
4	Appointment Listing	FR6, FR12, FR15
5	Booking System	FR1
6	Additional Practice Tools	FR13, FR14, FR17
7	Additional User Tools	FR18, FR19, FR20
8	Application Enhancements	FR21, FR22, FR23, FR24, FR25, FR26

Figure 4.3: Feature-Driven sprints.

Development and progress tracking were managed through a Gantt chart, which broke down requirements into specific developmental tasks and tracked their corresponding completion status. Each task was given an estimated effort value measured in days. Each sprint contained 2 days at the end as a contingency buffer to accommodate challenges or delays. Figure 4.4 illustrates a sprint from the project plan. The full project plan is available within Appendix C.

Sprint 1 - User Login & Registration		04-01-25	17-01-25	14
Create users table in Hasura with fields and role permissions	Complete	04-01-25	04-01-25	0.5
Set up register API route and service methods	Complete	04-01-25	04-01-25	0.5
Implement password hashing with bcrypt in registration API	Complete	05-01-25	05-01-25	0.5
Create GraphQL mutation to insert new users	Complete	05-01-25	05-01-25	0.5
Build registration form and connect to API	Complete	06-01-25	06-01-25	0.5
Implement Toast notifications for registration flow	Complete	06-01-25	06-01-25	0.5
Set up login API route and service methods	Complete	07-01-25	07-01-25	0.5
Implement password check and JWT generation	Complete	07-01-25	08-01-25	1
Build login form and connect to API	Complete	08-01-25	09-01-25	1
Implement Toast notifications for login flow	Complete	09-01-25	09-01-25	0.5
Create useUser custom react hook	Complete	10-01-25	10-01-25	1
Set up profile API routes and service methods	Complete	11-01-25	11-01-25	1
Build user profile page and connect to API	Complete	12-01-25	12-01-25	0.5
Integrate third-party email provider (MailJet)	Complete	12-01-25	12-01-25	0.5
Create password_reset_tokens table in Hasura	Complete	13-01-25	13-01-25	0.5
Implement forgot/reset password API routes	Complete	13-01-25	13-01-25	0.5
Build forgot/reset password pages and connect APIs	Complete	14-01-25	14-01-25	1.5
Test all authentication and profile flows	Complete	15-01-25	15-01-25	0.5
Implement GraphQL Codegen	Incomplete	15-01-25	15-01-25	0.5
Contingency	N/A	16-01-25	17-01-25	2

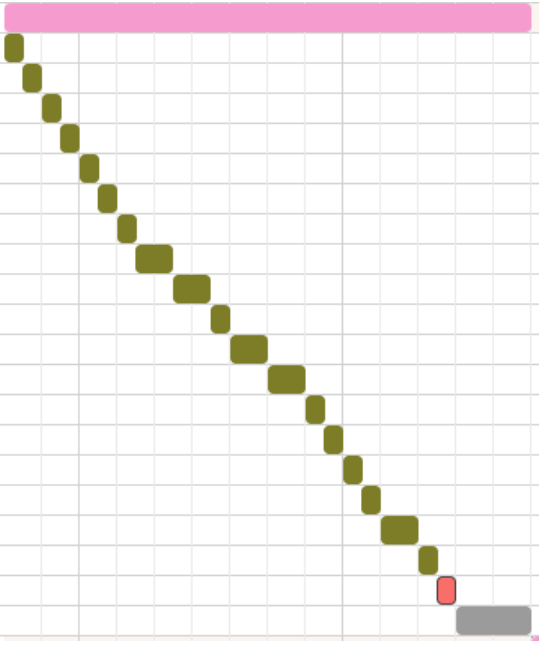


Figure 4.4: Sprint 1 User Login & Registration Gantt chart

4.3 Technology Choices

This section documents the comparison and choices of technologies, evaluating their suitability for the project's web application.

4.3.1 Programming Language

JavaScript and TypeScript are two of the most popular programming languages, ranking within the top 5 amongst professional developers in the Stack Overflow 2024 Developer survey (Stack Overflow, 2024).

Figure 4.5: Most popular programming languages (Stack Overflow, 2024)

TypeScript is a syntactic superset of JavaScript that adds static typing (TypeScript, 2025). The languages were compared to determine the most appropriate language for the web application. Figure 4.6 details a comparison of the two languages.

Feature	TypeScript	JavaScript
Typing	Provides static typing	Dynamically typed
Tooling	Comes with IDEs and code editors	Limited built-in tooling
Syntax	Similar to JavaScript, with additional features	Standard JavaScript syntax
Compatibility	Backward compatible with JavaScript	Cannot run TypeScript in JavaScript files
Debugging	Stronger typing can help identify errors	May require more debugging and testing
Learning curve	Can take time to learn additional features	Standard JavaScript syntax is familiar

Figure 4.6: Comparison of JavaScript and TypeScript (GeeksForGeeks, 2024).

The additional features and syntax in TypeScript result in a more significant learning curve. However, the initial time commitment is a worthwhile trade-off due to the more robust code provided by TypeScript's static typing. Static typing enables errors that occur at runtime in JavaScript to be detected at compile time in TypeScript. This reduces the potential and frequency of unexpected behaviour and errors at runtime, providing substantial time savings throughout the project's development lifecycle, significantly outweighing the initial time commitment (Microsoft, 2025). The web application spans multiple domains (appointments, practices, users, etc.), therefore benefiting from the code predictability provided by types, interfaces, enums, and generics, which help manage the complexity of large projects and future-proof the application for expansions and scaling (GeeksForGeeks, 2024).

4.3.2 Frontend Technologies

React and Angular are popular options for creating modern and responsive user interfaces, ranking first and second in the State of JavaScript 2024 survey (State of JavaScript, 2024). React is a front-end library that enables the creation of user interfaces via reusable components, with unidirectional data binding and a virtual Document Object Model (DOM). Angular is a fully-fledged front-end framework that enforces the Model-View-Controller (MVC) model, with two-way data binding and a real DOM, with support for features like dependency injection (Siddhpara, 2025). An explanation of the technical terminology is available in Appendix I.1.

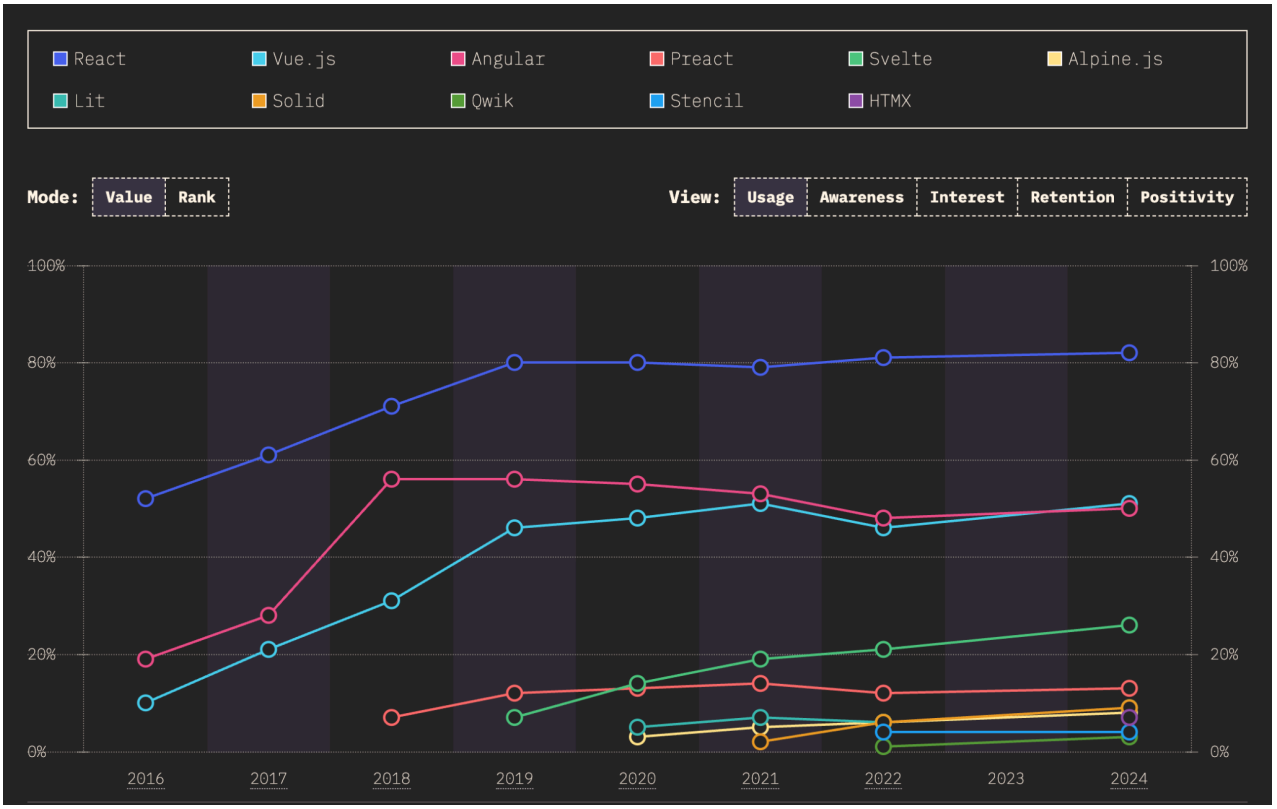


Figure 4.7: Front-end Framework popularity (State of JS, 2024).

Due to its simplicity, flexibility, and performance advantages, React was selected as the more appropriate front-end solution for the web application. Being a library rather than a framework, React has a simpler learning curve than Angular, which suits solo development and the project's timeframe (Siddhpara, 2025). The component-based architecture produces reusable and modular code that aligns with the project's requirements, FDD and the iterative nature of the Agile development lifecycle (Reis and Figueiredo, 2024). An example of component reuse is the sign-in and registration components, which are required on the sign-in page and within the booking flow (FR1, FR2, FR3). This reuse reduces development time, accelerating the delivery of an MVP.

A standout feature of React is its support for conditional rendering, which allows dynamic interface changes based on application state. This feature is especially applicable because the web application requires user interface (UI) elements, such as the navigation bar, to change based on a user's authentication status and role (FR3, FR5, FR8) (React, 2025a). Additionally, React is more performant, partly due to its virtual DOM, which improves rendering efficiency (Siddhpara, 2025). This is particularly beneficial given the application's performance requirements (NFR3).

However, as React is a library, it lacks routing and data fetching support. To address these issues when building a full-stack application, the creators of React recommend also using Next.js (React, 2025b).

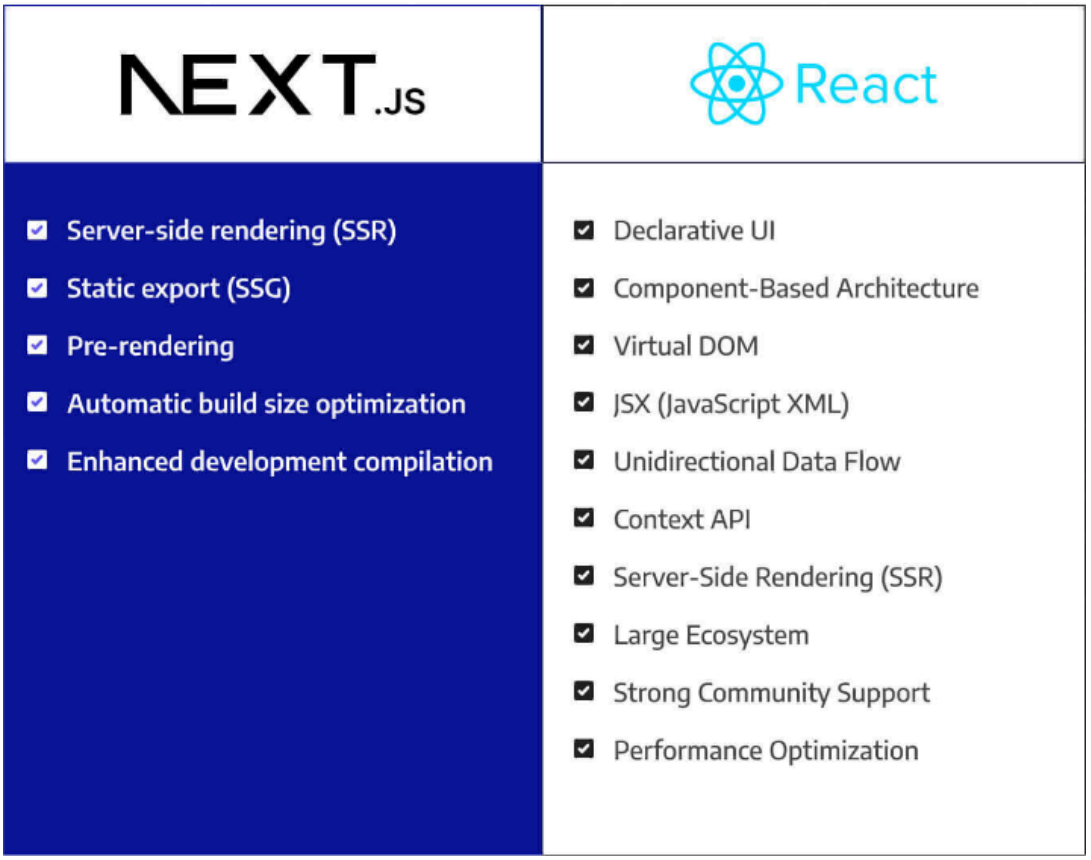


Figure 4.8: Next.js and React Features
(Nijhawan, 2022)

Next.js addresses React's limitations by automatically creating routes based on directory structure, eliminating the need for manual configuration (Vercel, 2025). It excels in Server-Side Rendering (SSR) and static site generation (SSG), which can improve performance and Search Engine Optimisation (SEO) (Vercel, n.d.c). SSR is especially beneficial for web application homepages, providing no loading screens with instant content rendering, thereby improving User Experience (UX) (Emadamerho-Atori, 2024). Next.js enforces several security features, including server-side execution of sensitive API calls, helping to protect user data and application secrets, aligning with industry best practices and the project's security requirements (NFR4) (Vercel, n.d.b). The combination of React and Next.js aligns with the need for rapid development, a quick delivery of an MVP, and the project's requirements to create a responsive, secure, and performant full-stack web application (NFR1, NFR3, NFR4).

Additional libraries will be used to enhance the web application. Zod, a TypeScript-first schema validation library, will be used to ensure data integrity and improve security by preventing the submission of harmful data. Toast notifications will provide real-time user feedback, improving UX.

4.3.3 Backend Technologies

PostgreSQL and MySQL are two of the most widely used Relational Database Management Systems (RDBMS), ranking first and second in popularity in the Stack Overflow 2024 developer survey (Figure 4.9) (Stack Overflow, 2024). Both store data in interrelated tables, use SQL as an interface for interacting with data and are open source (AWS, 2025). However, the two RDBMS excel in different use cases, detailed in Figure 4.10.

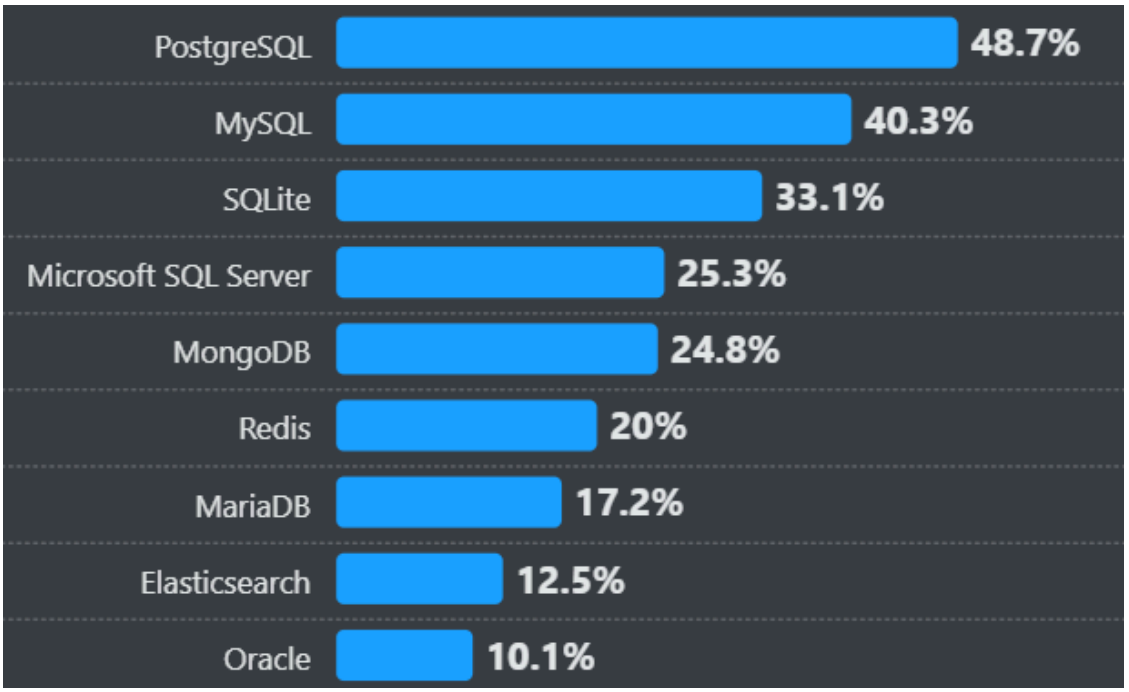


Figure 4.9: Database popularity (Stack Overflow, 2024).

Use case	PostgreSQL	MySQL
Large-scale enterprise applications	Excellent, with robust scalability and complex query support	Good, but better suited for lightweight tasks
Applications requiring geospatial data support	Ideal (PostGIS support)	Basic support
JSON and NoSQL-like data support	Strong (supports JSON and JSONB types)	Limited (basic JSON support)
High read performance (e.g., web apps)	Decent, but not as optimized for reads	Excellent (InnoDB engine, row-level locking)
Complex, concurrent read-write operations	Superior (multi-version concurrency control)	Good, but it may experience locking issues
Easy setup for small web projects	Takes longer to configure	Fast and easy to set up
Data warehousing and analytical processing	Excellent, with powerful indexing and parallel queries	Decent, but lacks advanced features

Figure 4.10 PostgreSQL vs MySQL use case comparison (Roach, 2024).

PostgreSQL was selected as the RDBMS for the project's web application because it is better suited for complex and large-scale applications. Flexibility and efficiency are provided for queries involving aggregate analytics or filtered queries, applicable within practice analytics (FR14) and appointment searching (FR6), through the mature support of window functions, Common Table Expressions (CTEs), and materialised views (Christensen, 2023). The geospatial data support through PostGIS is critical for enabling users to search for appointments based on location (FR6). The web application will store dynamic data, such as appointment types (FR7), making PostgreSQL's JSONB an ideal data type as it handles variable structures without compromising query performance through indexing (PostgreSQL Global Development Group, n.d.).

The Hasura GraphQL Engine will be implemented on top of the PostgreSQL database to provide a flexible API layer. Hasura automatically creates a GraphQL API based on the existing database schema (Hasura, 2025). This enables the web application to request the exact data required without needing to write complex queries due to the simpler declarative syntax that resembles object notation provided by GraphQL (GraphQL, n.d.). Figure 4.11 illustrates Hasura's integration, while Appendix I.2 explains the implementation within this project. Additionally, Hasura's support for Code Generator complements TypeScript by automatically generating TypeScript types based on the GraphQL schemas (The Guild, 2025). This ensures type safety for data returned from the database throughout the application, improving code robustness and reducing the likelihood of runtime errors.

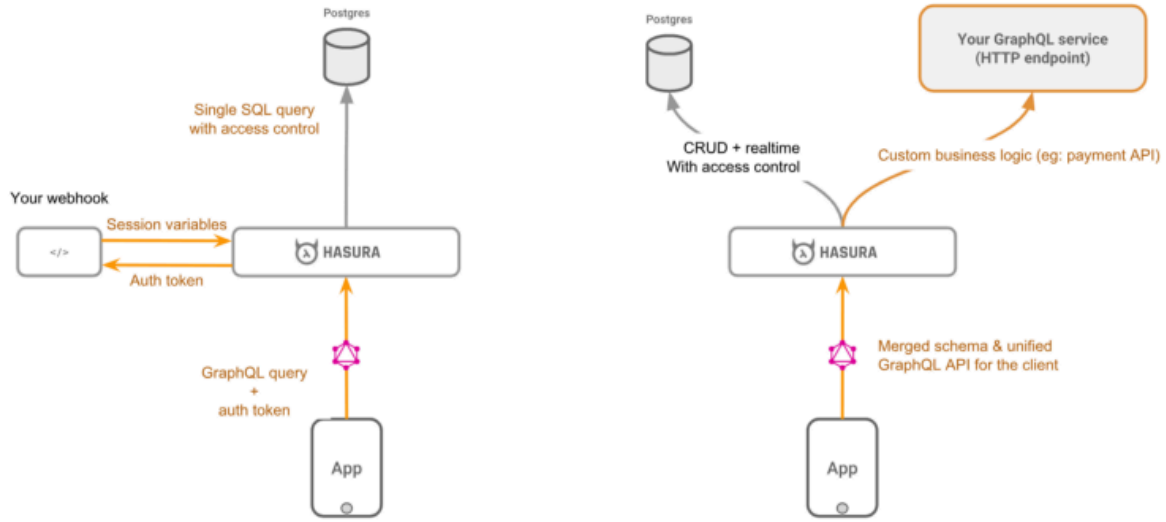


Figure 4.11: How Hasura works in your stack (Hasura, 2025).

Figure 4.12: GraphQL Code Generator Weekly NPM Downloads (NPM, n.d.).

4.3.4 Hosting, Deployment and Development Environments

The web application is hosted on Vercel, with a cloud-hosted Neon PostgreSQL database and a cloud instance of Hasura GraphQL Engine. Vercel was picked over alternatives like Netlify because it was founded by the same company that created Next.js, resulting in excellent native support for Next.js applications (Ikius, 2024). The Neon PostgreSQL database was selected due to native integration with Hasura, stemming from a partnership between the two companies (Hasura, 2020). Vercel and Hasura are connected to the project's repository via their GitHub integrations, enabling a CI/CD pipeline to automate builds, testing and deployment.

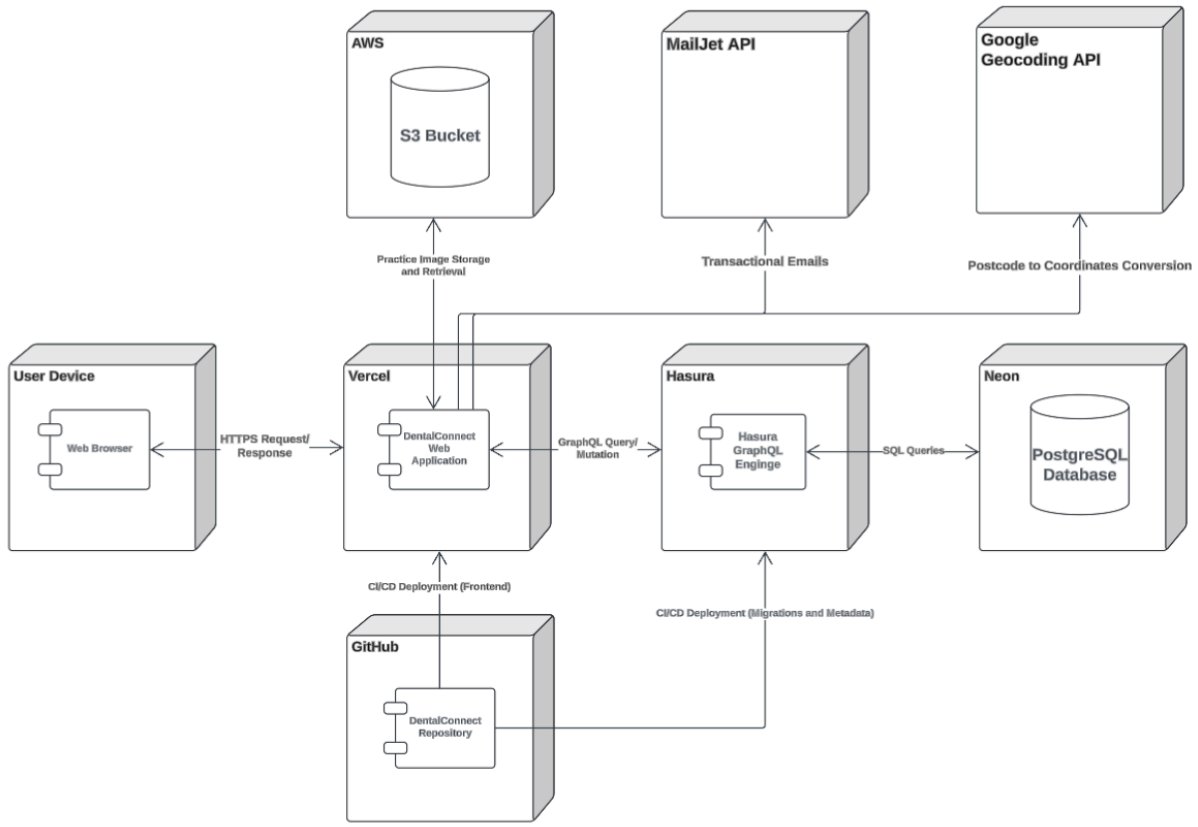


Figure 4.13: Deployment Diagram.

The web application is developed within a local development environment, allowing for immediate feedback on front and backend changes and safe testing of data schema changes. The environment runs via `npm run dev` and consists of a Dockerised PostgreSQL instance and Hasura GraphQL Engine, with environment variables stored in a `.env` file. Code changes and database schema migrations, which are tracked via the Hasura CLI, are committed to the project's GitHub repository. When database schema changes are pushed to GitHub, the Hasura GitHub integration automatically applies them to the cloud instance of Hasura and cloud-hosted Neon database, ensuring a consistent database schema across environments. The CI/CD steps are illustrated in Figure 4.14.

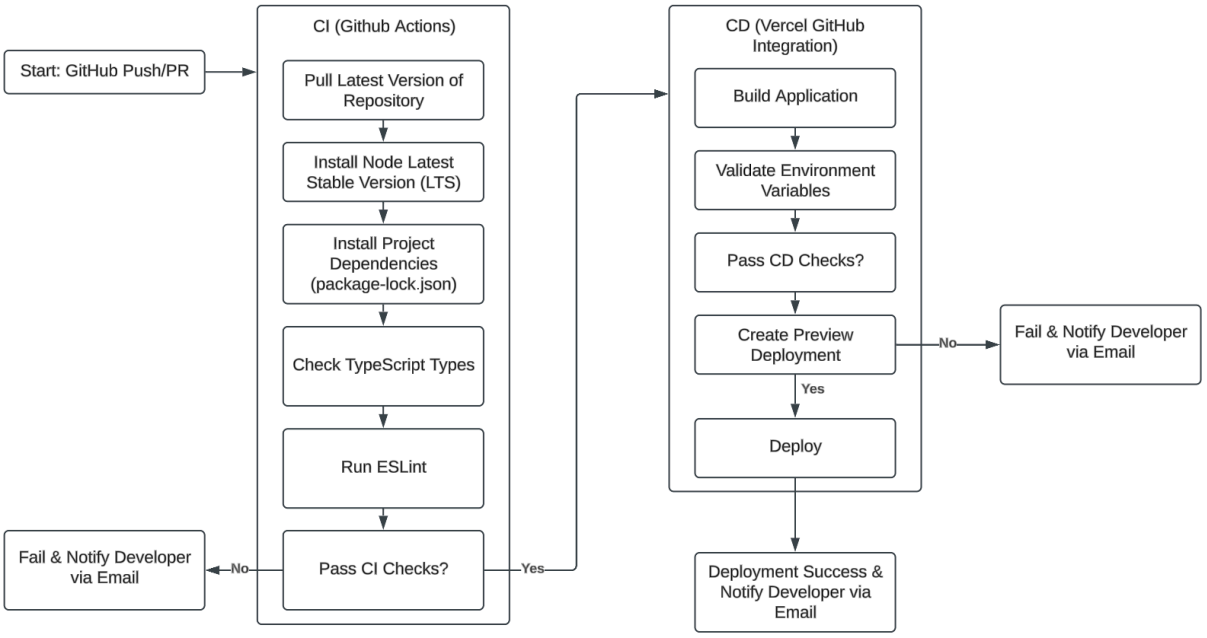


Figure 4.14: CI/CD Pipeline.

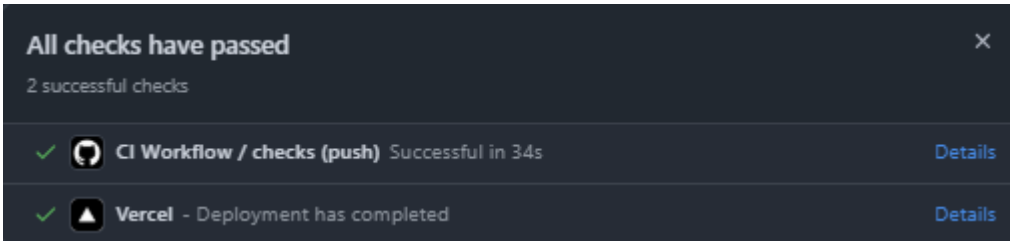


Figure 4.15: Example CI/CD pass.

This workflow ensures that code is tested and validated before deployment, ensuring the availability of the web application. Please refer to Appendix I.3 for additional information on the CI/CD pipeline setup, examples of failing pipelines, and details regarding GitHub integration.

4.3.5 Evaluation, Testing Tools and Techniques

Testing tools were selected based on three criteria: simple setups, minimal configuration, and clear, extensive documentation. Jest was used for unit testing, Cypress for end-to-end, and Google Lighthouse and Chrome Devtools, and the W3C validation for accessibility and performance. Further comparison and selection rationale is available in Appendix I.4.

Further information regarding the web application’s validation is available within Section 5.2.1.3. Usability testing results are available within Section 6.4.3.

The web application’s test traceability matrix, test case list, and example implementations are available in Appendix G.

4.4 Risk Analysis

A risk analysis is the process of identifying, measuring, and mitigating adverse events that may negatively affect a project's success. Its purpose is to quantify a project’s risks, promoting better informed decisions and to plan for contingency before issues arise (Hayes, 2025; Thomson Reuters, 2024).

Within this project, a 5x5 risk matrix was employed, following an Asana template, to assess and prioritise risks. A risk matrix analyses a project’s risks, assigning each a severity score based on the gravity of the consequences and a likelihood score based on the probability of the risk occurring. The two scores are multiplied to calculate an overall risk impact score, allowing risks to be categorised as High, Medium, or Low, enabling the prioritisation of risks accordingly (Team Asana, 2025). This supports effective risk management by highlighting the risks that require the most attention throughout the development lifecycle.

The definitions for severity, likelihood and impact ratings are provided in Appendix I.5. The project’s identified risks are available in Appendix I.6. Figure 4.16 presents the project’s risk matrix.

	1 Negligible	2 Minor	3 Moderate	4 Major	5 Catastrophic
5 Very Likely	5	10	15 Challenges with new technologies Unexpected behaviours and bugs	20	25
4 Probably	4	8	12	16 Scope Creep	20 Time Management
3 Possible	3	6 Deployment issues	9 Issues with external services	12 Illness	15
2 Not Likely	2	4	6	8	10
1 Very Unlikely	1	2	3	4	5 Loss of physical hardware

Figure 4.16: Project Risk Matrix illustrating identified risks plotted by severity and likelihood scores. Risk impact levels are colour-coded: Green for Low Risk (1–6), Orange for Medium Risk (8–12), and Red for High Risk (15–25).

5. Product Description

This section presents the project's developed web application, DentalConnect, starting with a system overview before detailing system architecture, design and implementation.

5.1 Introduction to Software Artefact

Figure 5.1 presents a system overview of DentalConnect, with Figure 5.2 outlining system-wide, role-specific use cases. These use cases inform the user journeys defined in Appendix J.1. The following subsections outline authentication and how appointments are created, searched and booked.

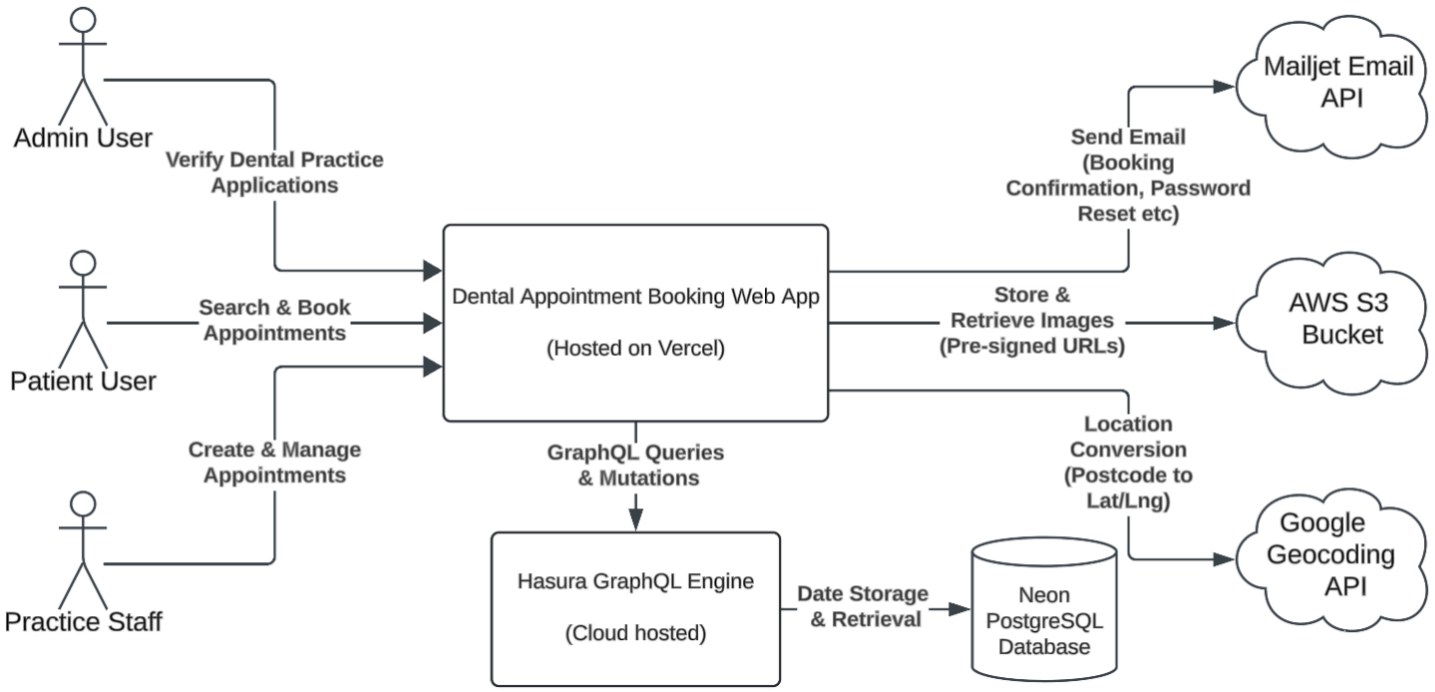


Figure 5.1: DentalConnect System Overview Diagram.

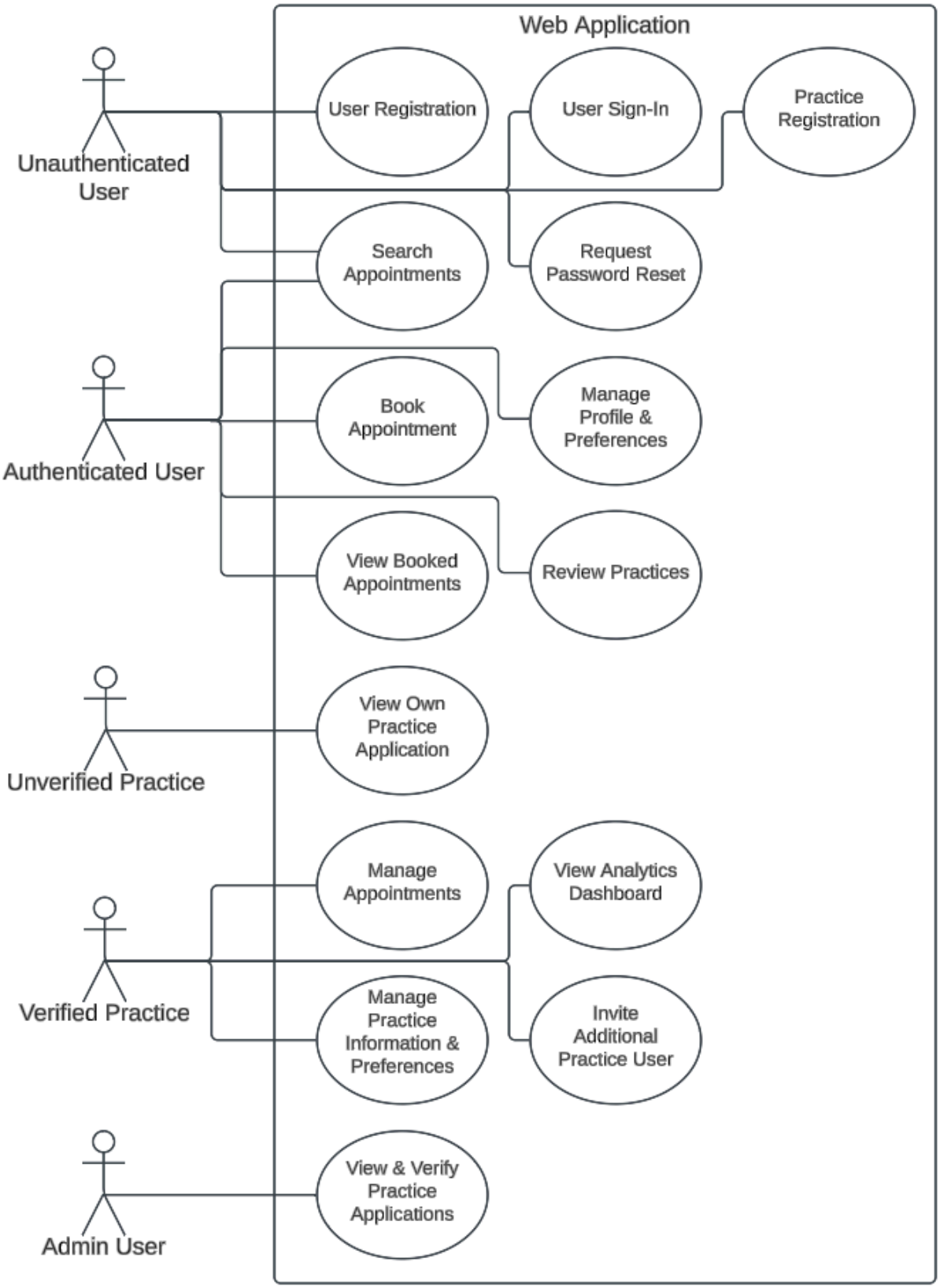


Figure 5.2: DentalConnect Use Case Diagram.

5.1.1 Authentication and Role-Based Access

System authentication and role-based access control are implemented using JSON Web Tokens (JWT) with a custom React hook, `useUser`. The hook provides the user's authentication status to enforce role-based access, details in Appendix J.2.

When a user signs in, credentials are securely validated via the `/api/auth/login` endpoint. Passwords are hashed using `bcrypt` and compared against the hash stored within the database to ensure secure authentication. Upon successful authentication, access and refresh tokens are issued and stored in HTTP-only cookies. If "Remember Me" is selected, access token expiry extends from one hour to seven days, supporting persistent sessions while maintaining security.

The `useUser` hook retrieves and monitors authentication status, user role and session details via the `/api/auth/me` endpoint. These states are leveraged for conditional rendering, dynamically populating navigation bar options and page content based on user authentication status and role. See Figure 5.3 for navigation bar content dependent on user authentication status and role.

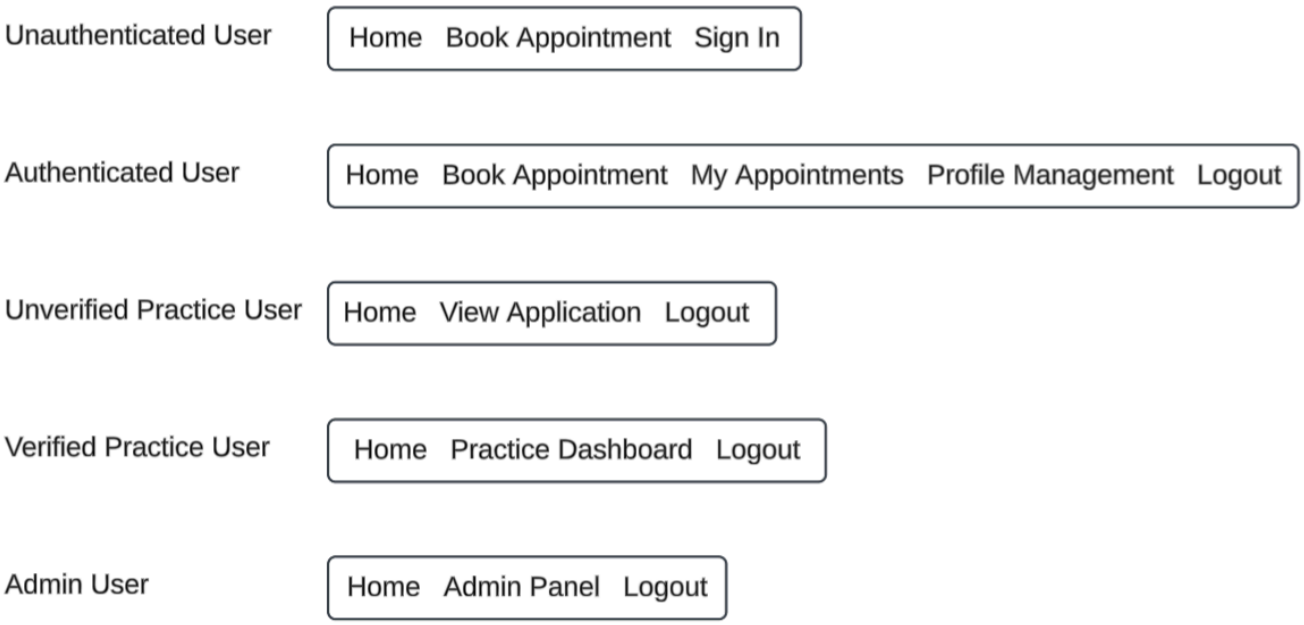


Figure 5.3: Navigation bar content based on user authentication.

Access control is enforced throughout the application via client-side checks. For example, access to the Practice Dashboard is restricted to verified practice users (Figures 5.4 and 5.5). This ensures that unauthorised users cannot access protected content, even if attempting to directly navigate.

```
if (!user || user.role !== "verified-practice") {
  return <div>Access Denied: You are not authorised to view this page.</div>;
}
```

Figure 5.4: Client-side role-based access control.

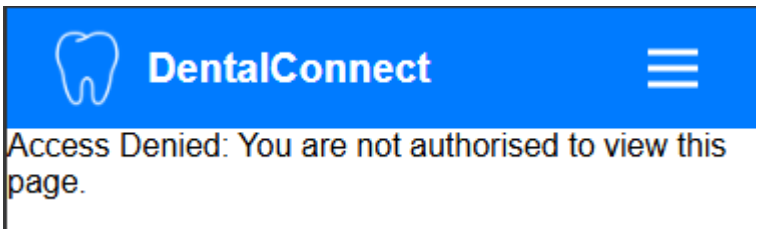


Figure 5.5: Access Denied message.

JWT-based session management ensures a secure, scalable, and user-friendly authentication system. HTTP-only cookies mitigate common security risks such as cross-site scripting (XSS), combined with conditional rendering, client-side and server-side checks, prevent unauthorised access. Passwords are enforced to require a minimum length and the inclusion of special characters. This approach aligns with industry best practices for secure web development and provides a seamless experience tailored to each user's role.

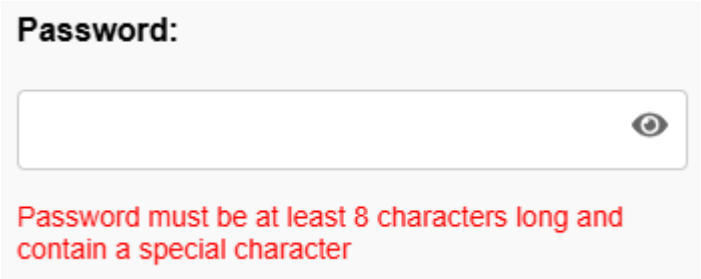


Figure 5.6: Password field validation.

5.1.2 Appointment Management

During practice registration, users configure their practice's opening times (Figure 5.7), select the appointment types they offer, and set corresponding prices (Figure 5.8). The form is validated via Zod to ensure data integrity, the schema is available in Appendix J.3.

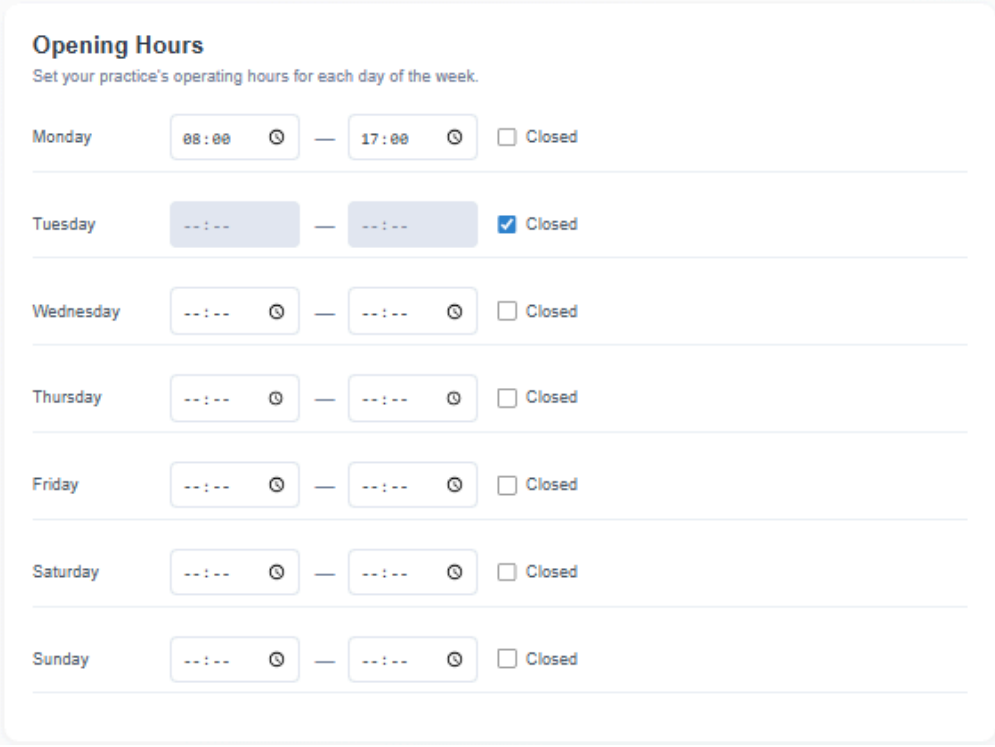


Figure 5.7: Practice opening hours from the practice registration form.

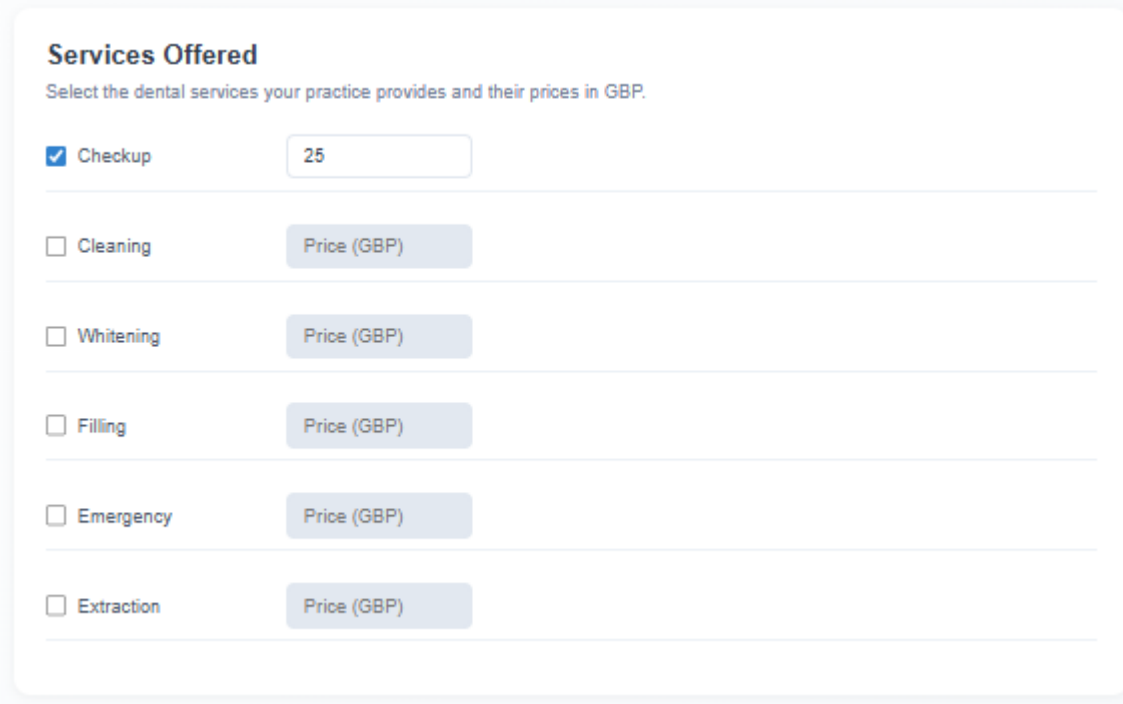


Figure 5.8: Services offered section from the practice registration form.

Once verified by an admin, the practice user gains access to the practice dashboard (Figure 5.9). All practice-related pages use a custom React hook, `usePractice`, to populate the page with the user's practice-specific data, details in Appendix J.4. This approach ensures that page content is tailored to the logged-in practice user, improving UX and performance through efficient data handling.

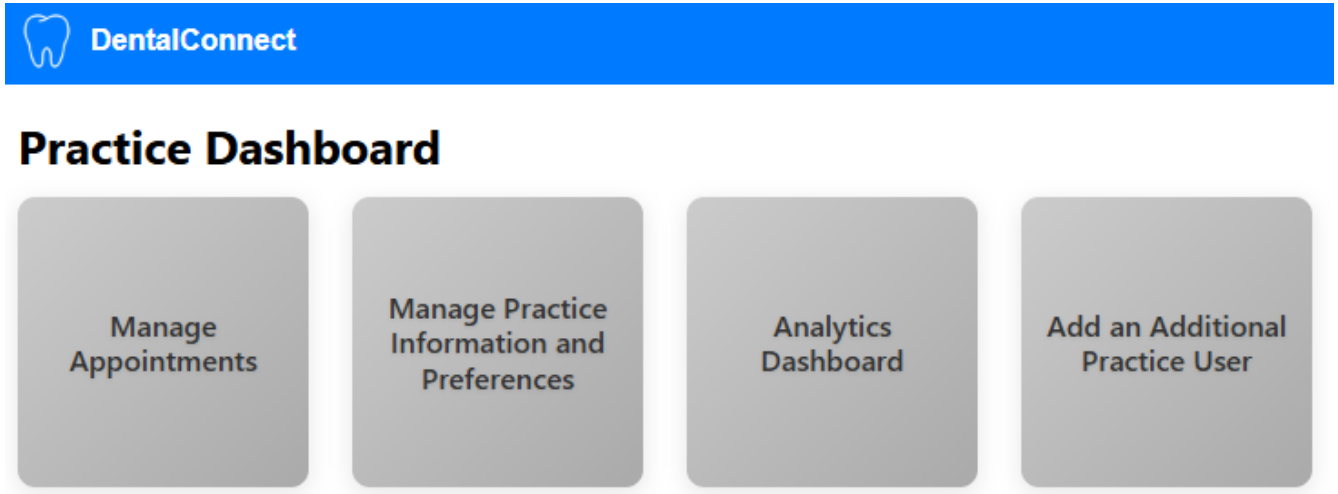


Figure 5.9: Practice Dashboard

The practice dashboard contains access to the Manage Appointments page. This page contains a calendar interface where light-grey and dark-grey boxes represent the practice's configured opening hours, enhancing usability (Figure 5.10).

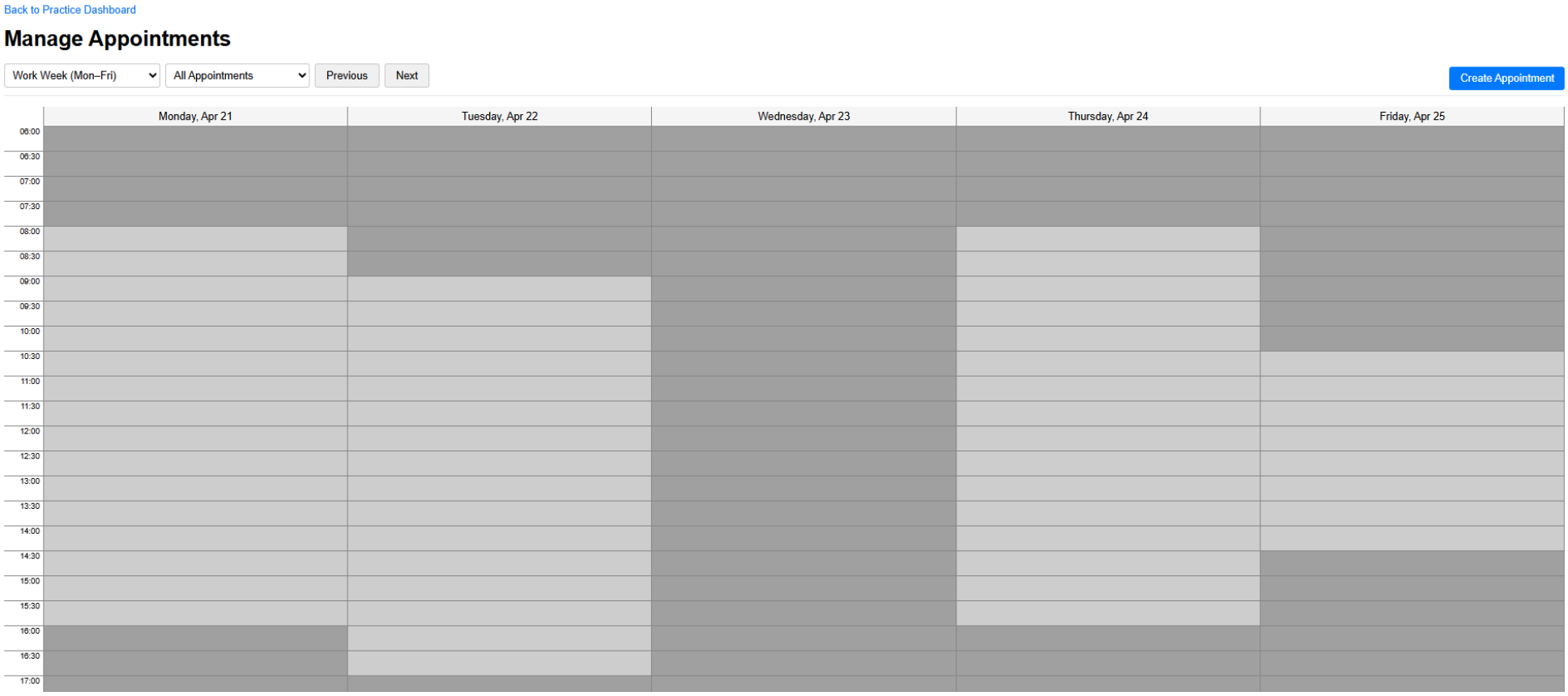


Figure 5.10: Manage Appointments page.

The calendar supports day, work week, calendar week, or month views with filtering for booked or unbooked appointments.

Appointments are created through the "Create Appointment" button or by clicking a time slot within the calendar.

A pop-up prompts the user to specify start and end times, defaulted at 30 minutes, and presents checkboxes of the appointment types supported by the practice. The practice user selects which services to offer for the specific appointment (Figures 5.11 and 5.12). Once created, the calendar updates in real-time and supports overlapping appointments where applicable (Figure 5.13 and Figure 5.14).

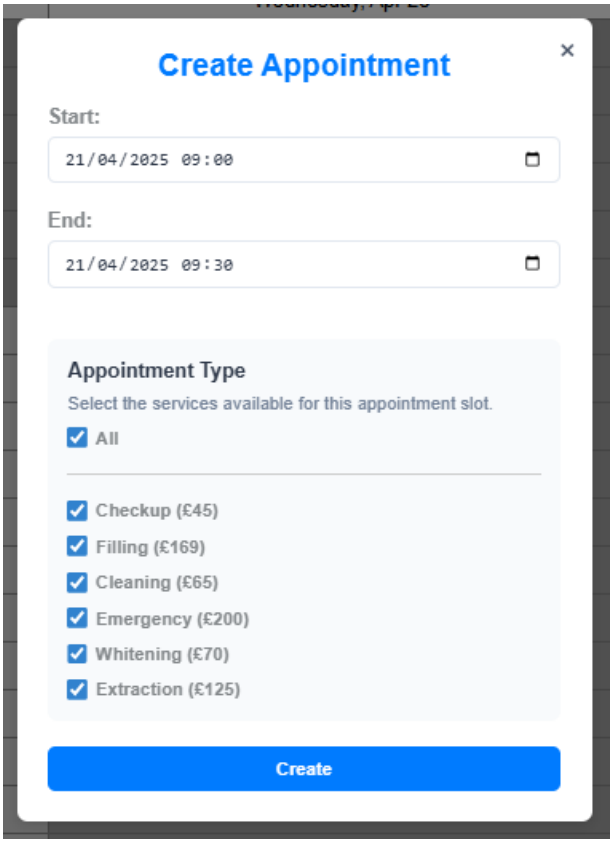


Figure 5.11: Create Appointment pop-up.

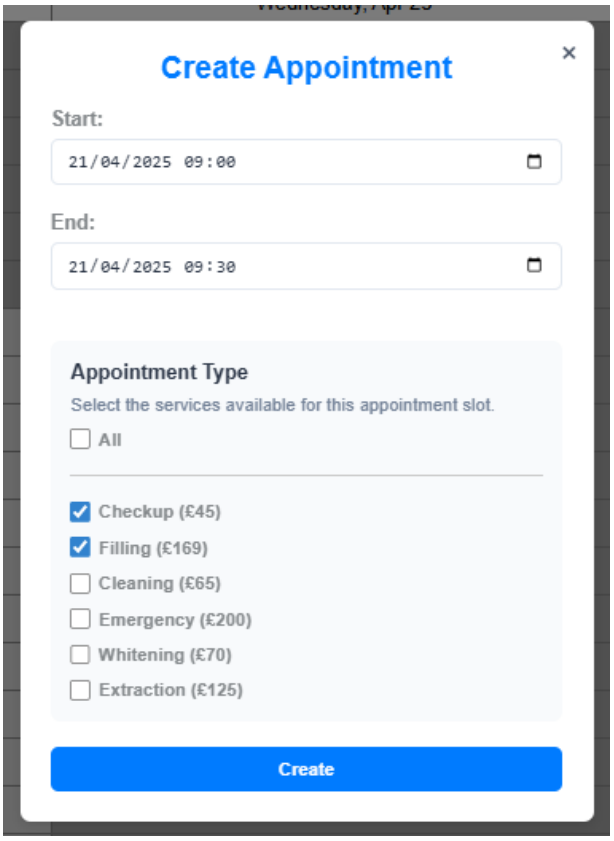


Figure 5.12: Create Appointment pop-up, Checkup and Filling selected.

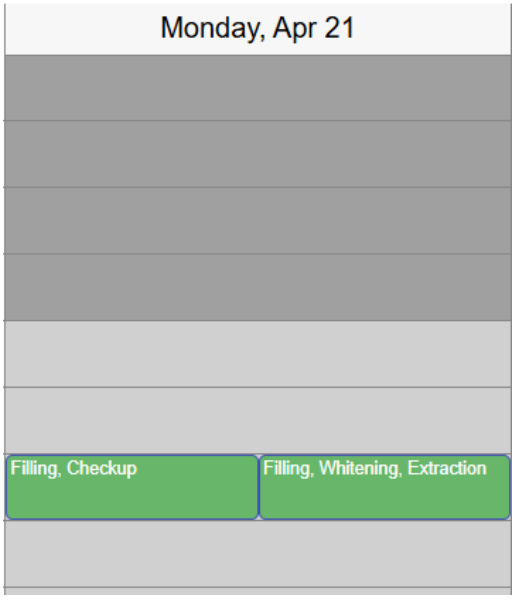


Figure 5.13: Appointment Management calendar displaying two overlapping appointments.

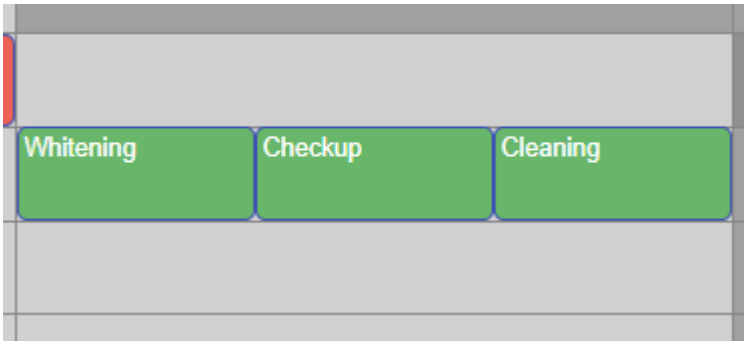


Figure 5.14: Appointment Management calendar displaying three overlapping appointments.

Zod ensures valid appointment times, and at least one appointment type is selected (Figure 5.15).

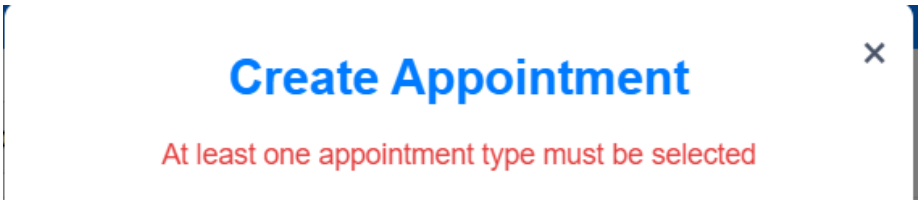
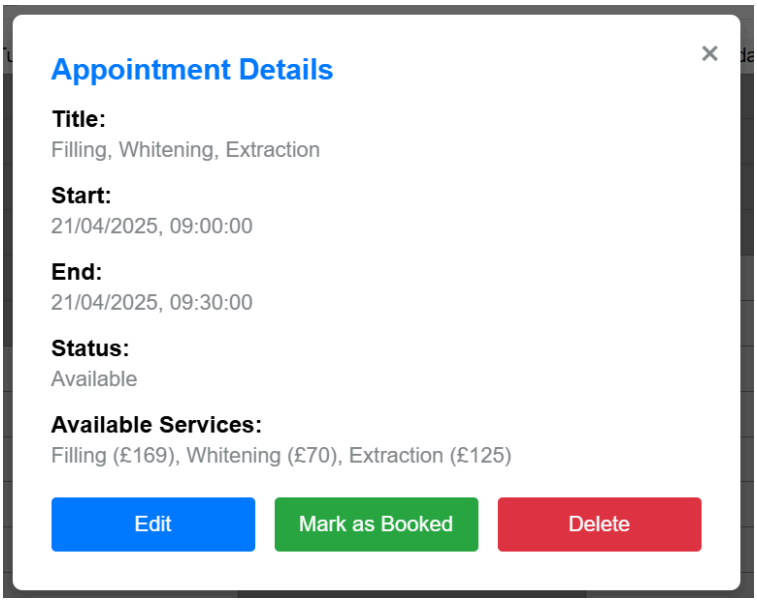


Figure 5.15: Create Appointment pop-up, no services selected form validation.

Practices can view appointment details by clicking on appointments in the calendar, with options to edit, manually mark as booked, or delete (Figure 5.16-5.18). Editing and deletion of an appointment is supported if it is unbooked.



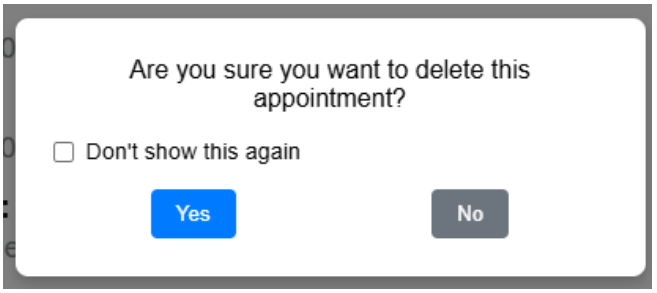


Figure 5.17: Confirmation pop-up when deleting an appointment.

Figure 5.16: Appointment details pop-up displaying appointment information.

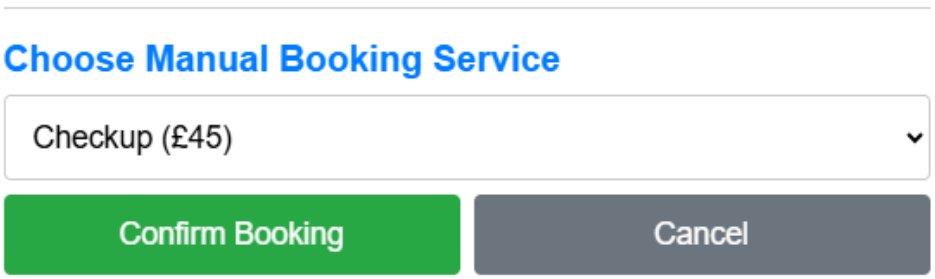


Figure 5.18: Manual appointment booking interface.

Once an appointment is booked, it changes from green to red within the calendar, and upon clicking into the appointment, the patient's name and contact details are available (Figures 5.19 and 5.20).

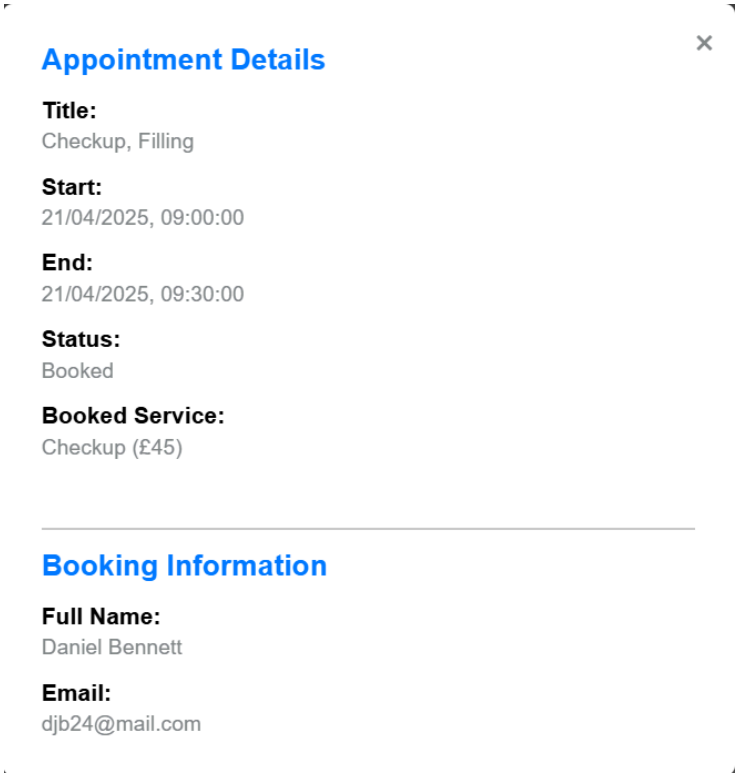


Figure 5.19: Appointment details pop-up for booked appointment showing patient information.

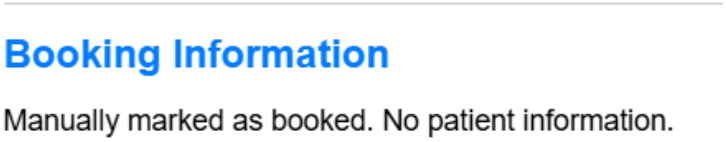


Figure 5.20: Appointment details pop-up for booked appointment showing patient information when manually marked as booked.

The system enforces constraints to maintain data integrity:

- Appointments cannot be created outside of opening hours (Figure 5.21).
- Practices cannot modify opening hours if a conflicting appointment exists.

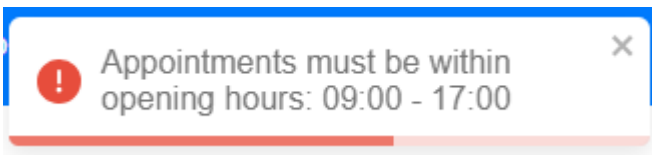


Figure 5.21: Toast error message shown when attempting to create an appointment outside of practice opening hours.

5.1.3 Appointment Searching and Booking

The user navigates to the search page by either the “Search for Appointments” button within the homepage or via the ”Book Appointments” button in the navigation bar. By default, a minimum date range is automatically set to the current date and time. Users select filters to search for appointments (Figure 5.22). Matching appointments are returned and displayed in cards, otherwise, a user-friendly message is displayed (Figure 5.23 and Figure 5.24).

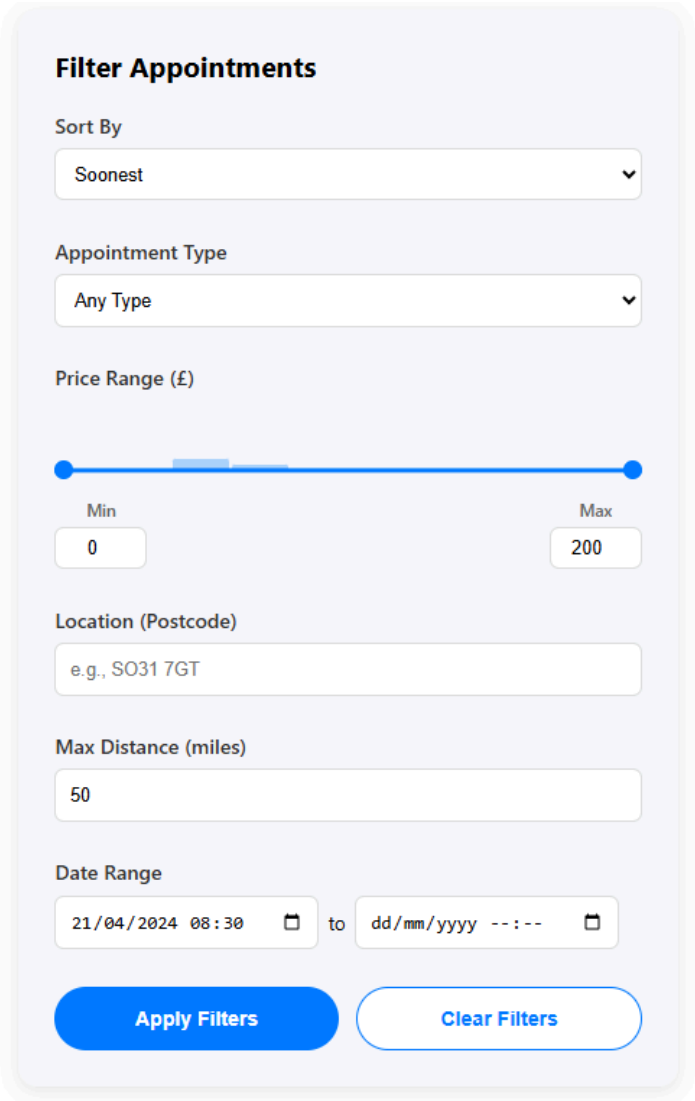


Figure 5.22: Search page filter component with default minimum date range.

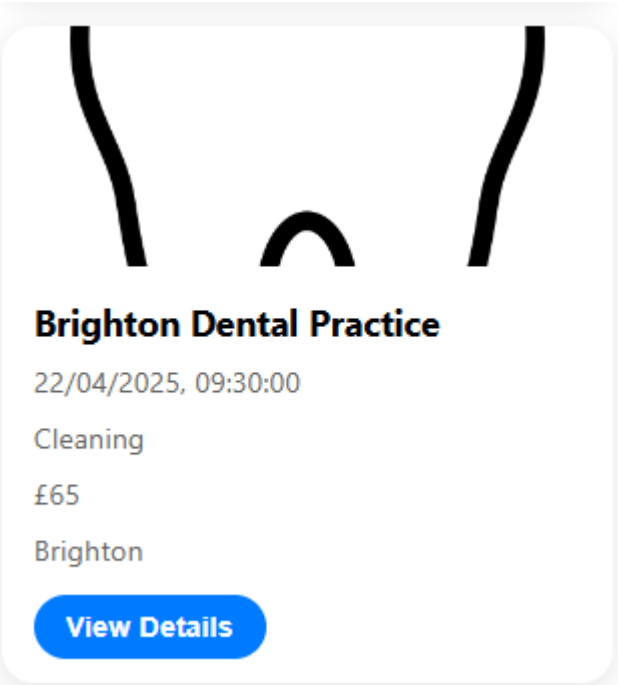


Figure 5.23: Search page appointment card.

Figure 5.24: Message displayed when no appointments match the selected filters.

The user selects a desired appointment and is taken to an overview page showing appointment and practice details (Figure 5.25). They select a service and proceed to checkout (Figure 5.26).

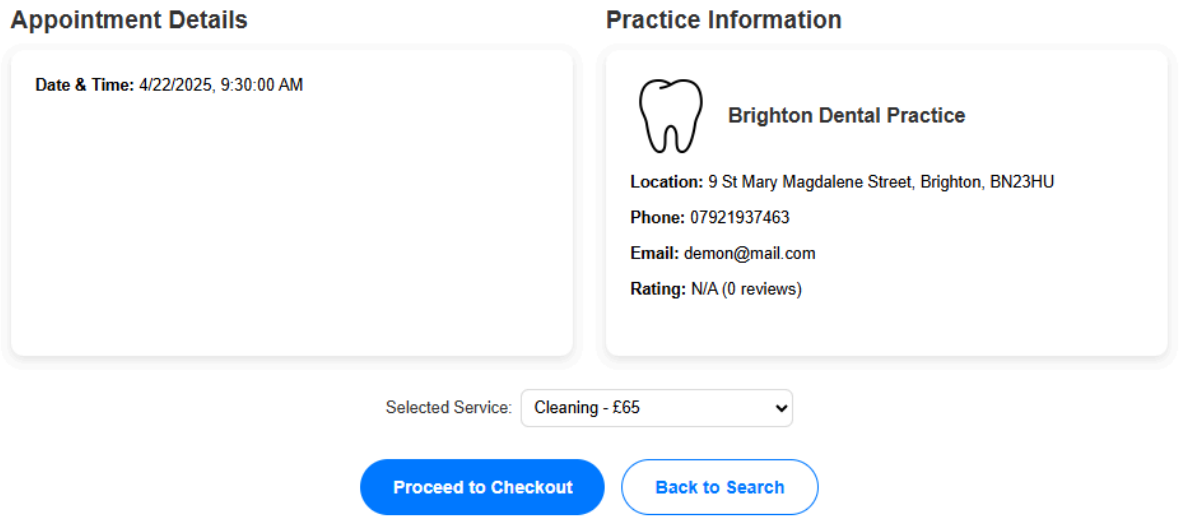


Figure 5.25: Overview page.

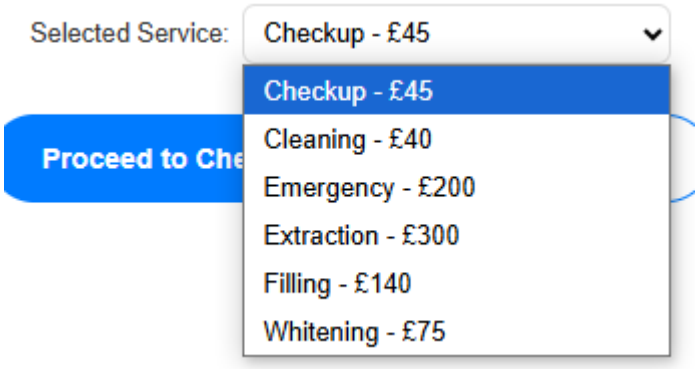


Figure 5.26: Service selection.

If not signed in, the user is prompted to sign in or alternatively register. The user is then taken to a booking summary page. This improves UX by helping to reduce booking errors. Upon booking confirmation, the user receives an email and can view their booked appointment within the “My appointments” page.

5.2 Architectural Design and Implementation

The web application adheres to a three-tiered architecture consisting of a presentation, application and data tier (IBM, n.d.). The architecture promotes maintainability via the separation of concerns and supports scalability as bottlenecks can be addressed on a per-tier basis (GeeksforGeeks, 2021). The modularity provided by the architecture enables each tier to be developed independently, simplifying modification and additions, and aligning the approach well with FDD and an agile workflow (Chiaramonte, 2024). The architecture is illustrated in Figure 5.27.

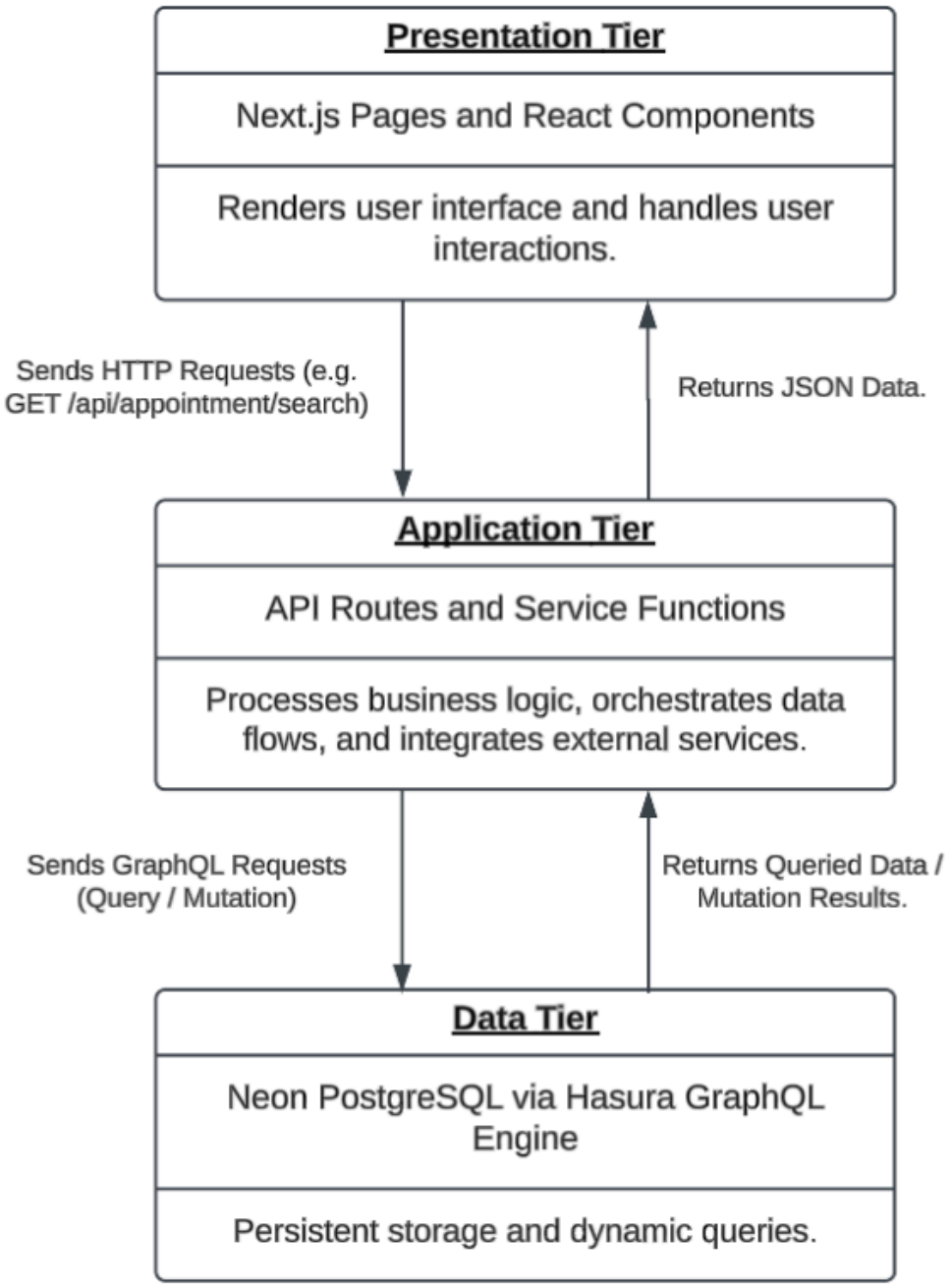


Figure 5.27: DentalConnect three-tier architecture.

5.2.1 Presentation Tier

The presentation tier is responsible for the web application's UI (IBM, n.d.). Dynamic and static content is displayed via React components within Next.js pages via the Next App router. The UI is designed to be accessible and responsive on a range of devices and aligns with both O4: Ensure Responsive Design and O5: Evaluate Usability and Accessibility. Figure 5.2.2 illustrates a web navigation diagram of the web application.

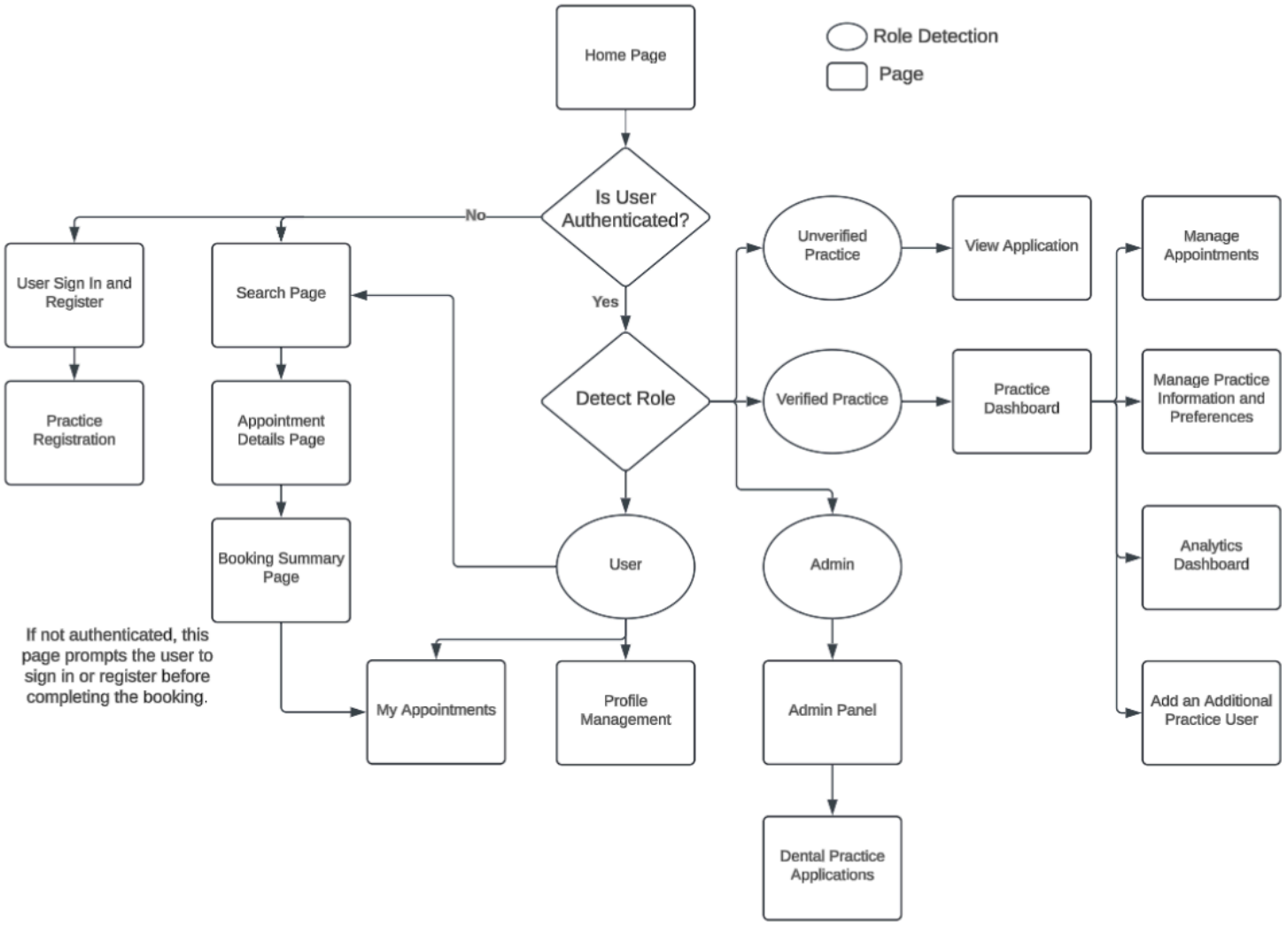


Figure 5.28: Web navigation diagram.

5.2.1.1 CSS Implementation

Within the web application, the UI is styled using CSS Modules via `{page/componentName}.module.css`, such as `HomeDashboard.module.css`, to maintain modularity and avoid style leakage (CSS Modules, 2023). Style leakage is a prominent issue in globally styled React applications, leading to verbose class names and difficult-to-debug code due to accidental overrides (Eightwalk Technology, 2023). Dynamic units, like viewport height (VH) and root em (REM), are utilised over fixed units to achieve responsiveness where the scaling of UI elements is required for different screen sizes (W3Schools, n.d.b). When the UI layout needs to be adjusted based on screen size, media queries are implemented with breakpoints to provide responsiveness (W3Schools, n.d.a). An example of the implementation of media queries is collapsing the navigation bar into a hamburger menu for mobile devices. This improves user experience (UX) by hiding the navigation, which otherwise would be cluttered on a smaller device, maintaining a clean and accessible user interface (UI). The desktop homepage is shown in Figure 5.29 and the mobile in Figure 5.30. The open hamburger menu is shown in Figure 5.31. Full code implementation can be viewed in Appendix J.5, and CSS dynamic unit examples in Appendix J.6.

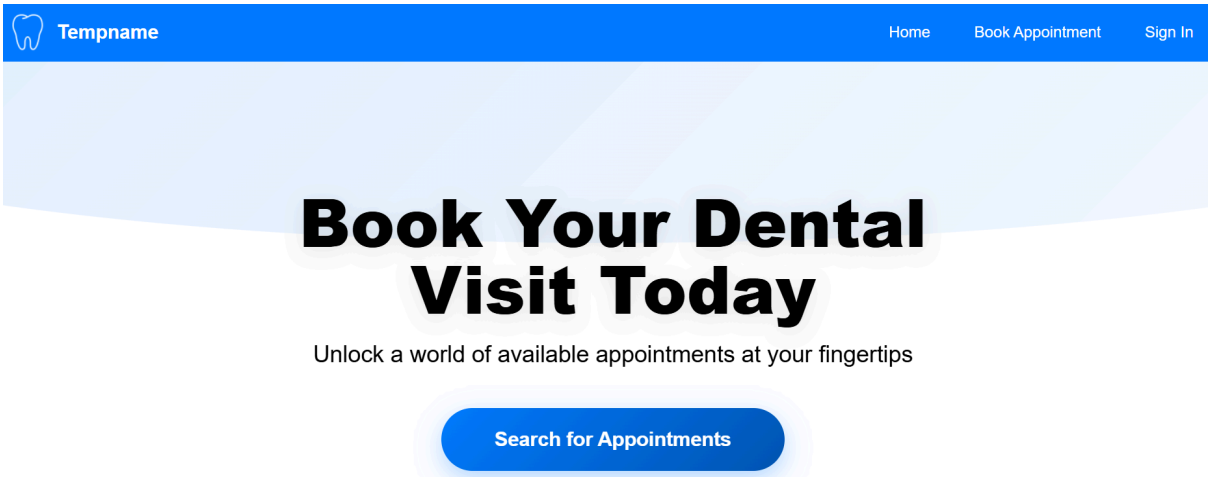


Figure 5.29: Web Application Homepage on Desktop

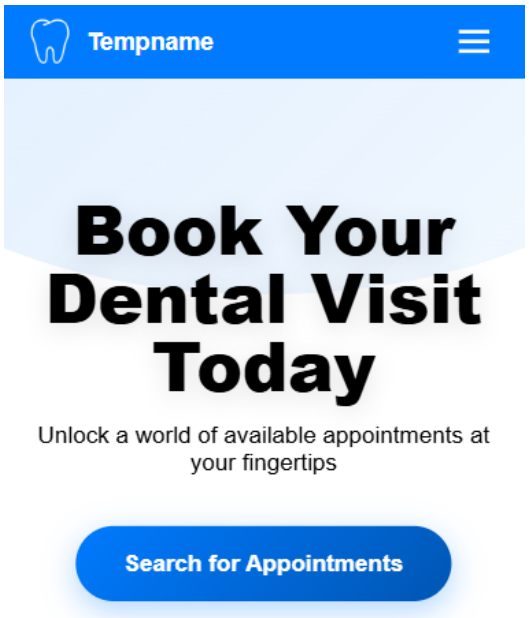


Figure 5.30: Web Application Homepage on mobile.

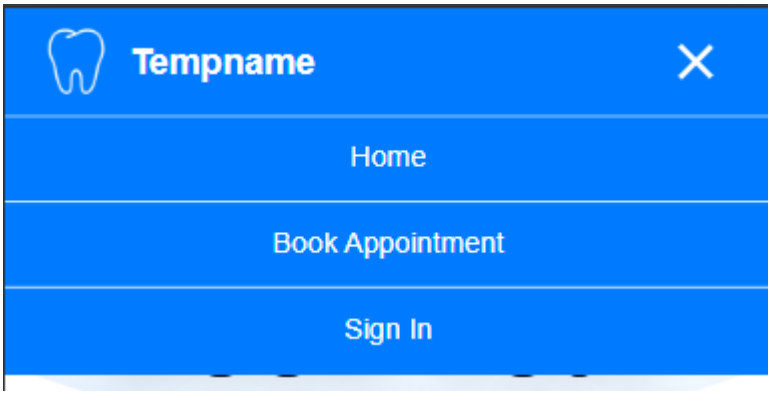


Figure 5.31: Open Hamburger Menu on mobile.

5.2.1.2 Server-Side Render (SSR)

Server-side rendering (SSR) is utilised throughout the web application for enhanced performance, increased SEO and an improved user experience (UX) (Vercel, n.d.c). Notably, SSR is applied within the homepage of the web application. SSR operates by processing requests on the server: when a user navigates to the web application's homepage, the Next.js App Router's async page function fetches data from both the `/api/appointment/home` as well as the `/api/practice/home` API routes to retrieve appointment and practice data from the database. Then the HTML is pre-rendered with the dynamic data returned from the database before being delivered to the user's browser. The SSR flow of the homepage is visualised via a sequence diagram in Figure 5.32, showing from the user request through to the HTML rendering and a class diagram in Figure 5.33 illustrating where and how the data is rendered via React components.

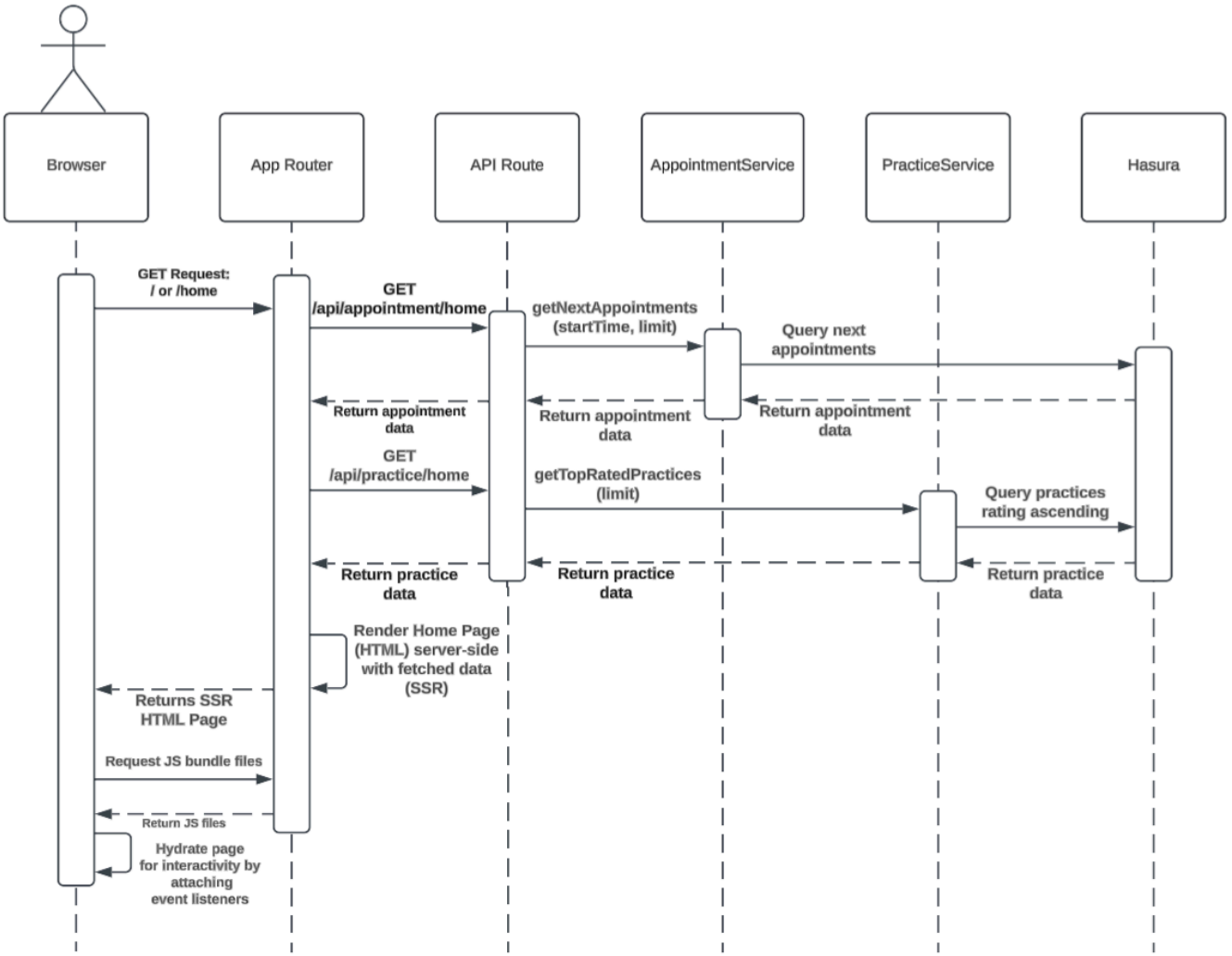


Figure 5.32: Homepage SSR sequence diagram.

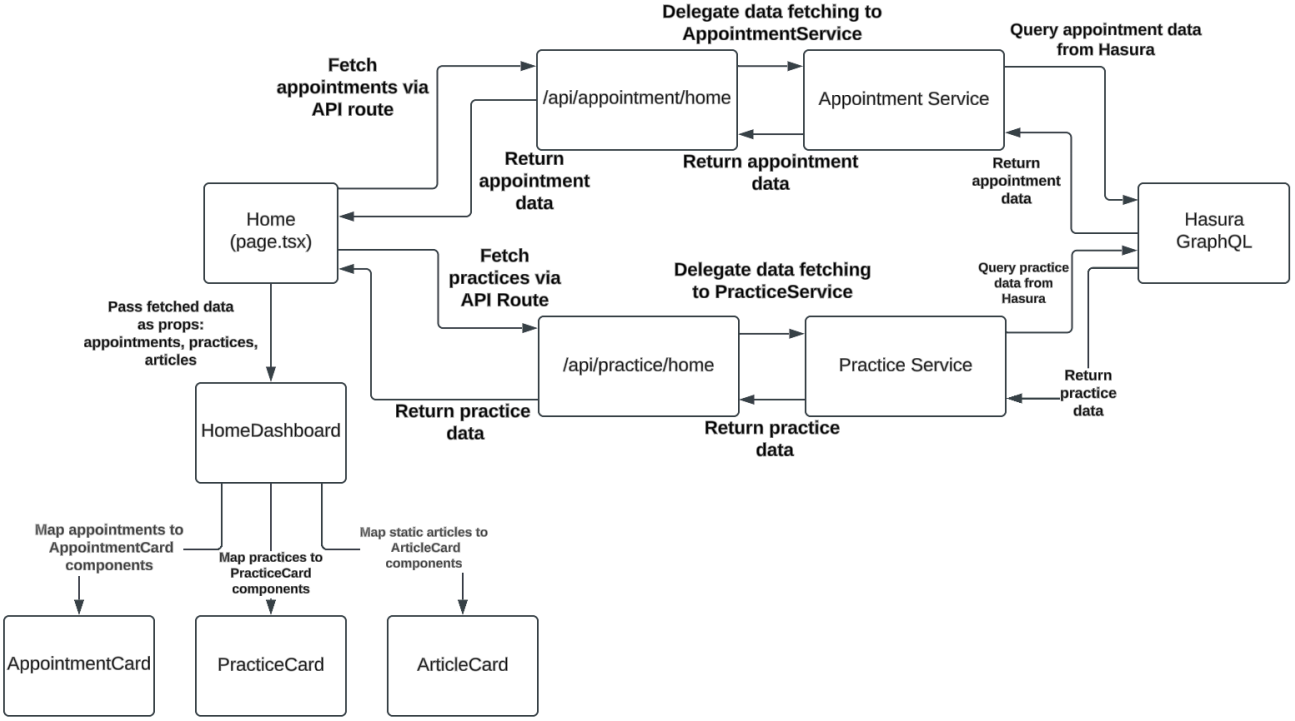


Figure 5.33: Homepage class diagram

SSR is chosen over the alternative of client-side rendering (CSR) due to the near-instant initial render, as verified by Google Lighthouse's first contentful paint performance metric to typically be 1 second, shown in Figure 5.34 (Emadamerho-Atori, 2024; Google Developers, 2019). Additionally, if the web application's homepage were to implement CSR, the user would be met with a loading screen, decreasing the UX (Emadamerho-Atori, 2024). This supports O4 Responsive Design through consistent performance across a range of devices, and O5 Usability and Accessibility by ensuring immediate content access.

Figure 5.34: Google Lighthouse First Contentful Paint Performance Result.

5.2.1.3 Validation

The effectiveness of the presentation tier was validated via automated tooling and manual testing. Google Lighthouse, an automated tool that measures metrics including performance, accessibility and SEO, was run for mobile and desktop devices (Google Developers, 2016). The results are shown in Figure 5.35 and Figure 5.36. The W3 validator was used to ensure the application's HTML was standards-compliant, with results shown in Figure 5.37 (W3C, n.d.). Manual testing on a variety of device sizes, both physical and emulated within Chrome DevTools, was carried out following the user journeys of the web application (Google Developers, 2024). The results from this combined approach confirm the efficacy of the presentation tier, with Google Lighthouse scores of 95 or above on all metrics, zero errors detected within the W3 validator, and successful manual testing. This confirms the fulfilment of both O4: Responsive Design and O5: Usability and Accessibility.

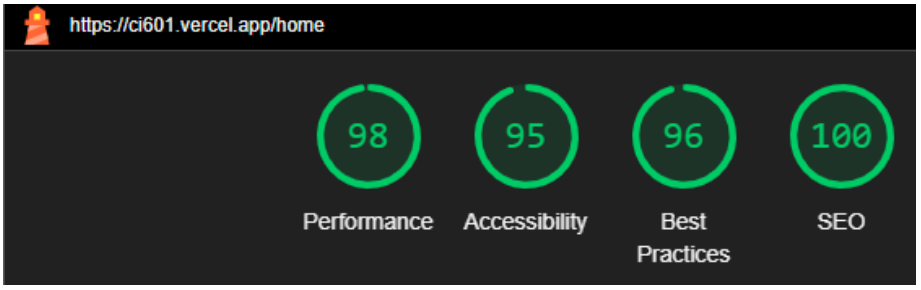


Figure 5.35: Google Lighthouse Desktop Result

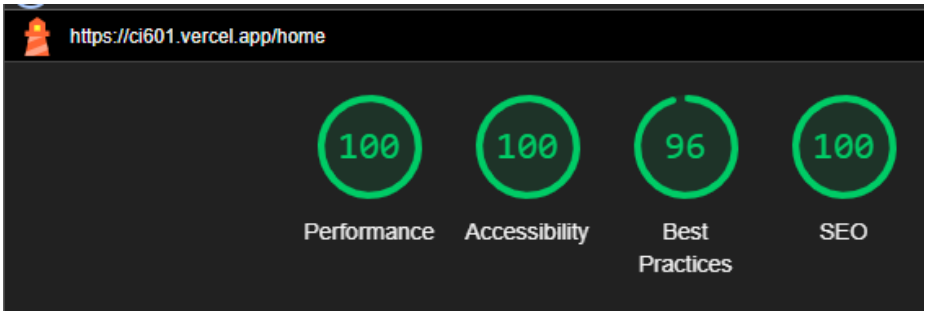


Figure 5.36: Google Lighthouse Mobile Result

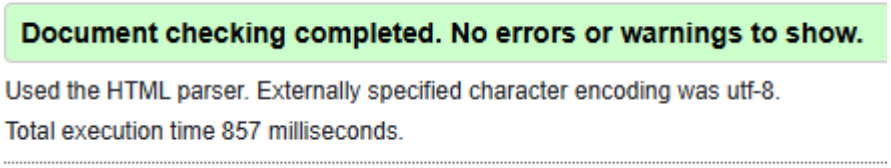


Figure 5.37: W3 Validator Result

5.2.2 Application Tier

The application tier serves as an intermediary between the presentation and data tiers for managing and performing business logic (IBM, n.d.). Next.js API routes handle incoming requests from the presentation tier and orchestrate required logic by delegating tasks to related services. Each specialised service is responsible for operations related to one aspect of the web application, for example, appointments, dental practices or authorisation.
All functionality, such as user authorisation via `/api/auth/login`, flows through the application tier, enabling adherence to a strict separation of concerns (Naumov, 2020). The approach produces a modular, easily maintainable, and scalable application tier aligning with both feature-driven development as well as an agile workflow (GeeksforGeeks, 2021).

5.2.2.1 Routes

Next.js API routes serve as an entry point to the application tier, implemented as serverless functions within the `/api/` directory. Folders and files within the directory map to API endpoints. For example, the `{BaseURL}/api/appointment/search` route file structure is detailed within Figure 5.39, with a successful GET request to the endpoint shown in Figure 5.40. Full code implementation of the appointment search route is available in Appendix J.7.

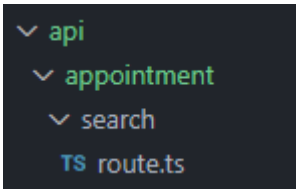


Figure 5.39: API directory structure.

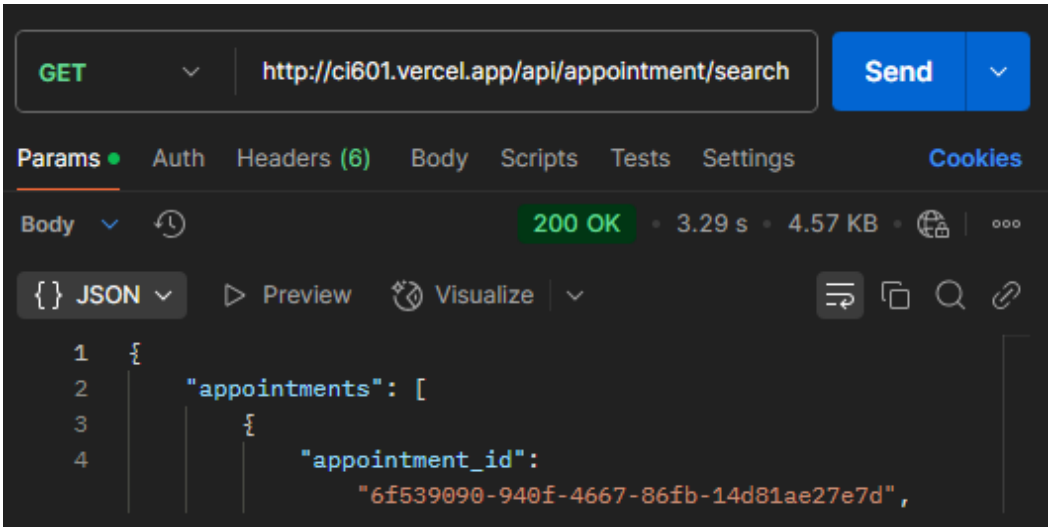


Figure 5.40: Postman GET request to search route.

Each Next.js route receives a request (`req`), extracts parameters, invokes methods in relevant services and then returns a response (`res`) (Vercel, n.d.a). For instance, the route handling appointment booking performs basic validation and then offloads the booking to the `bookAppointment()` method within `appointmentService.ts` (FR1). The full code implementation can be viewed within Appendix J.8, and a visual representation of the logic is illustrated in Figure 5.41.

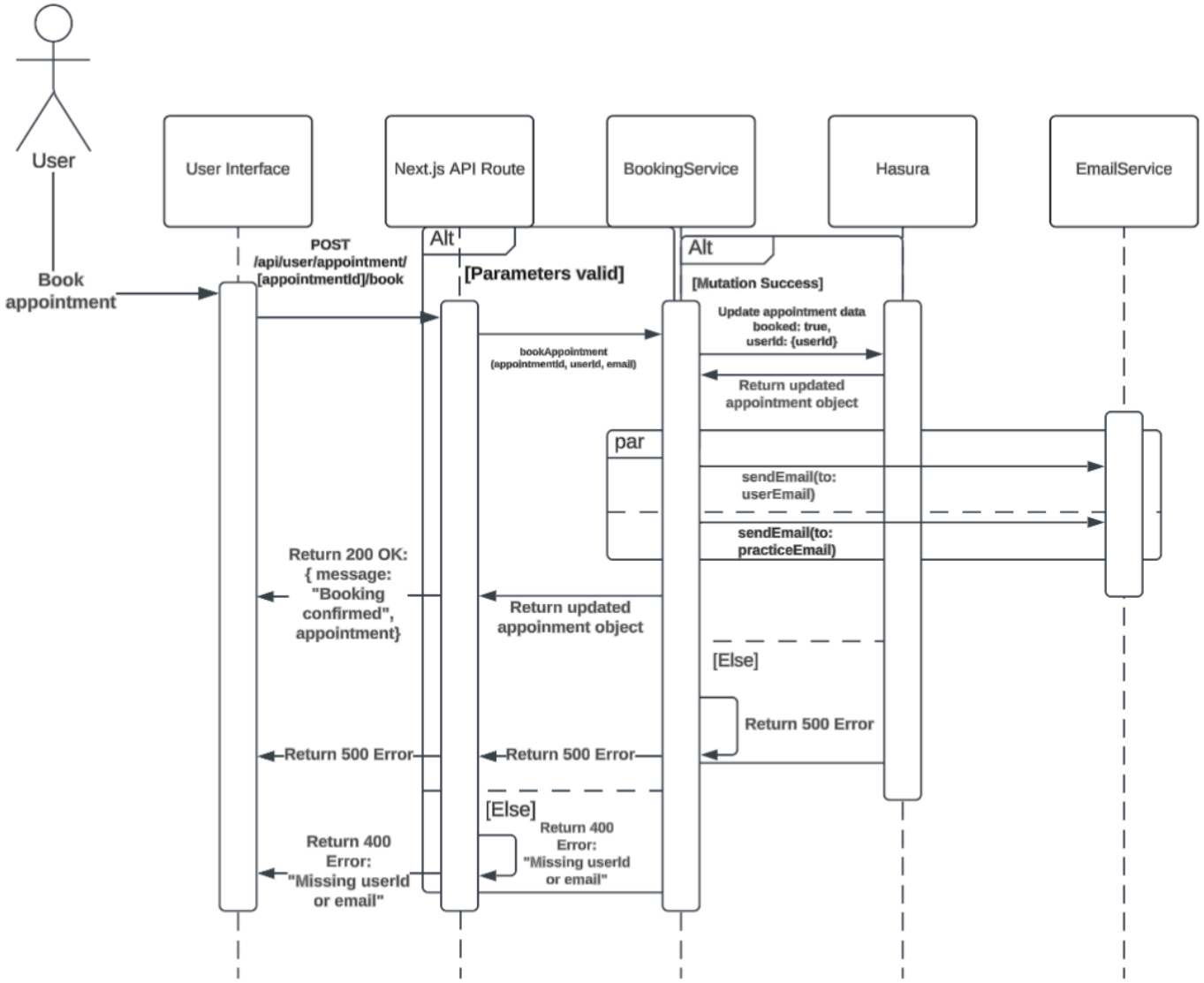


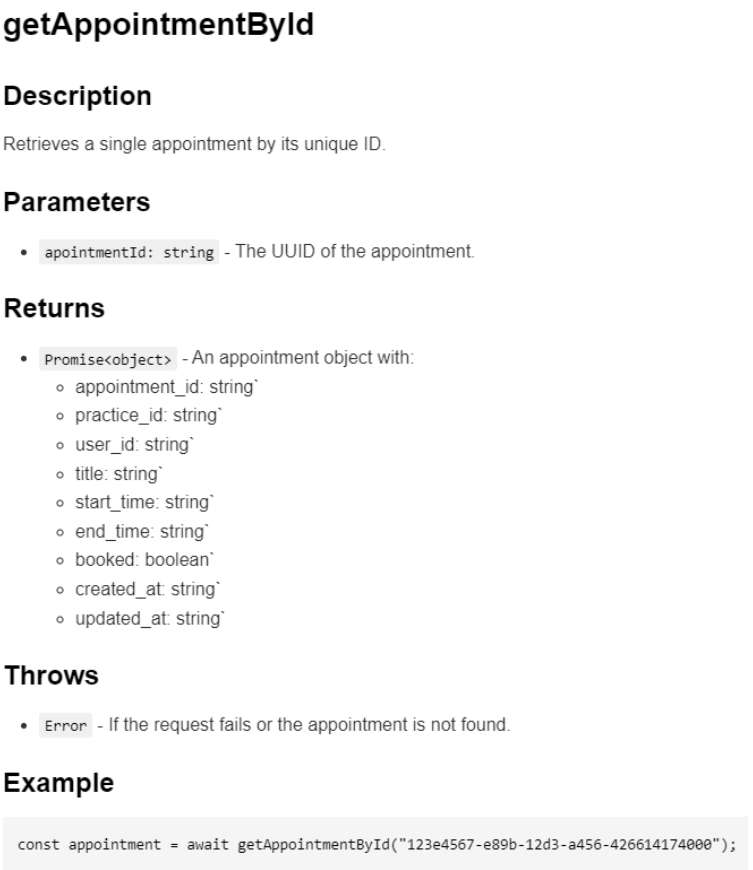
Figure 5.41: Booking sequence diagram.

Next.js's architecture enforces server-side execution of API calls to confine sensitive logic and API keys, in turn enhancing security (Vercel, n.d.b). Without routes, client-side code would be unable to call the methods provided by the services. Within the web application, thin routes, which contain minimal business logic and delegate to services, are used over thick routes, which are functions containing all business logic. This choice follows the separation of concerns (SOC) principle, request handling and business logic are strictly separate, and the single responsibility principle (SRP), each component (route and service) has a single job (Dijkstra, 1974; Martin, 2003). By adhering to these principles, the routes have increased testability, maintainability and scalability as well as better support code reuse and reduce duplicate code (GeeksforGeeks, 2021).

5.2.2.2 Services

Services encapsulate the business logic within the application tier, implemented as TypeScript modules grouped by domain within the `/services/` directory. The full directory structure is detailed in Appendix J.9. Each service exports domain-specific methods for routes to invoke, such as the `appointmentService.ts` providing methods for booking and searching appointments (FR1, FR6), and the `userService.ts` providing user authentication functionality (FR2, FR3). The modular service approach is particularly well suited to the web application, as it spans across multiple domains (e.g. appointments, users, practices). This approach promotes FDD as each domain can expand independently.

Some methods act as orchestrators by leveraging functionality from other services, like `sendEmail()` from `emailService.ts` for email confirmations, or utility functions, such as `generateToken()` from `authUtils.ts` for JSON web token (JWT) creation. Other methods execute GraphQL queries and mutations for the retrieval, modification and creation of data within Hasura GraphQL. Confining the web application's business logic solely within the services adheres to the separation of concerns (SOC) principle as well as the single responsibility principle, as each service serves a distinct role (Dijkstra, 1974; Martin, 2003). For full documentation of services, please refer to Appendix H. An example documented method is shown in Figure 5.42.



getAppointmentById

Description

Retrieves a single appointment by its unique ID.

Parameters

- `apointmentId: string` - The UUID of the appointment.

Returns

- `Promise<object>` - An appointment object with:
  - appointment_id: string`
  - practice_id: string`
  - user_id: string`
  - title: string`
  - start_time: string`
  - end_time: string`
  - booked: boolean`
  - created_at: string`
  - updated_at: string`

Throws

- `Error` - If the request fails or the appointment is not found.

Example

```
const appointment = await getAppointmentById("123e4567-e89b-12d3-a456-426614174000");
```

Figure 5.42: Example service documentation

5.2.3 Data Tier

The data tier, built on a Neon cloud-hosted PostgreSQL database accessed via a cloud instance of Hasura GraphQL, provides persistent storage for all aspects of the web application (IBM, n.d.). For development, a local dockerized PostgreSQL database and Hasura instance mirrored the production cloud instances, with schema changes being tracked via metadata and migrations using Hasura's CLI (Hasura, n.d.c). The schema migration history is available within Appendix J.10, and the final database schema is visualised by an entity-relationship diagram (ERD) in Figure 5.43.

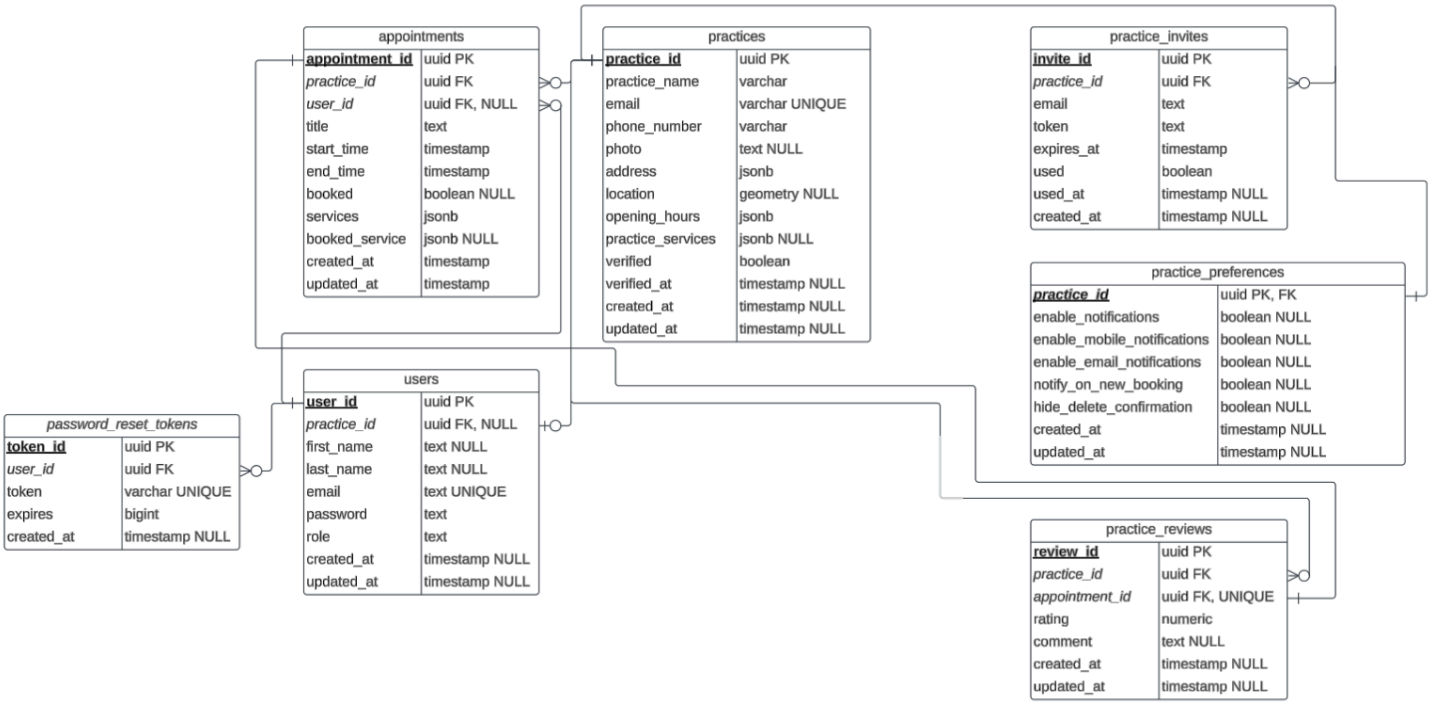


Figure 5.43: Database Entity-Relationship Diagram

5.2.3.1 Views, Functions and Filtering

Functional Requirement 6 (FR6) for the web application requires that a user can filter available appointments on multiple parameters, including appointment type, distance, date, and price. To follow best practices and optimise performance, database-side filtering is required (Codd, 1990). If filtering were to be performed within the web application it would require all appointments to be returned from the database, leading to an increase in latency and bandwidth usage thereby worsening the UX (Nielsen, 1993).

Hasura excels with database-side filtering on static fields, like `start_time`, with dynamically populated user parameters via GraphQL queries. For a user to filter appointments by distance, a computed field `distance` is required. A computed field is dynamically calculated at query time and not stored as a column within the database (Hasura, n.d.a). The `distance` value represents the distance between the user's location, latitude and longitude coordinates calculated dynamically from the postcode provided within the search page via the Google Maps API, and the practice's `location` column, latitude and longitude coordinates calculated from the postcode provided during signup via the Google Maps API (Google Developers, 2025). For a full breakdown of the coordinate calculation via the Google Maps API integration and the geometry `location` column, please see Appendix J.11. A standard approach for filtering on a computed field within a PostgreSQL database is to include it within the filtering logic of a query. However, filtering on a computed field is not supported within Hasura. An alternative solution is to create a custom function, but Hasura dictates that the result of a function can only include columns of a given table, therefore blocking the inclusion of the computed field `distance` as it is not a column (Hasura, n.d.b).

To work around Hasura's constraints, the `appointments_with_distance` view was created. A view is a virtual table that is created by fetching data from one or more tables by an SQL query. The content in a view is dynamically produced from the referencing tables (Hasura, 2021). Within the view, a dummy `distance` column, with a default value of 0, fulfils Hasura's requirement that a function must return table columns (Hasura, n.d.b). An additional benefit of including the dummy column within the view is that the schema clarity of the `appointments` table is preserved (Elmasri and Navathe, 2015). The structure of the appointments_with_distance view, derived from the appointments and practices tables, is shown via an ERD in Figure 5.44. The SQL definition is available in Appendix J.12.

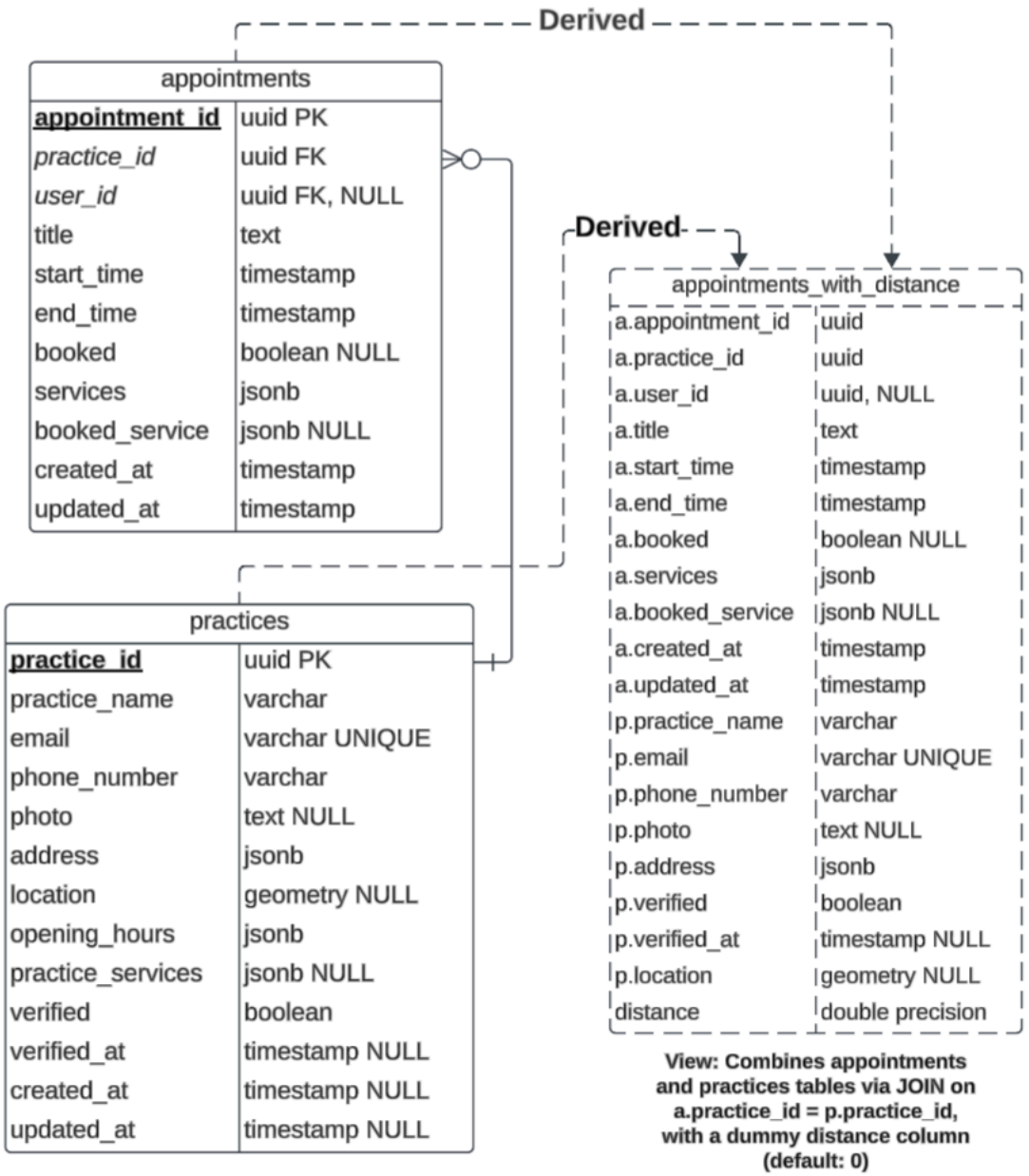


Figure 5.44: Entity-Relationship Diagram showing the `appointments_with_distance` view

The `get_nearby_appointments` function (Appendix J.13) calculates the actual value for distance using the PostGIS ST_Distance function (PostGIS, n.d.b). Thereby enabling the filtering of appointments by distance with an additional optional parameter `max_distance` to limit the radius of the returned appointments. By calculating the distance dynamically from the database side within PostgreSQL, spatial indexing is leveraged to avoid application tier overhead (PostGIS, n.d.a). For the steps taken to enable PostGIS within PostgreSQL please see Appendix J.14. The data flow diagram in Figure 5.45 illustrates appointment creation by a practice and the subsequent searching of appointments by a user, utilising the `get_nearby_appointments` function.

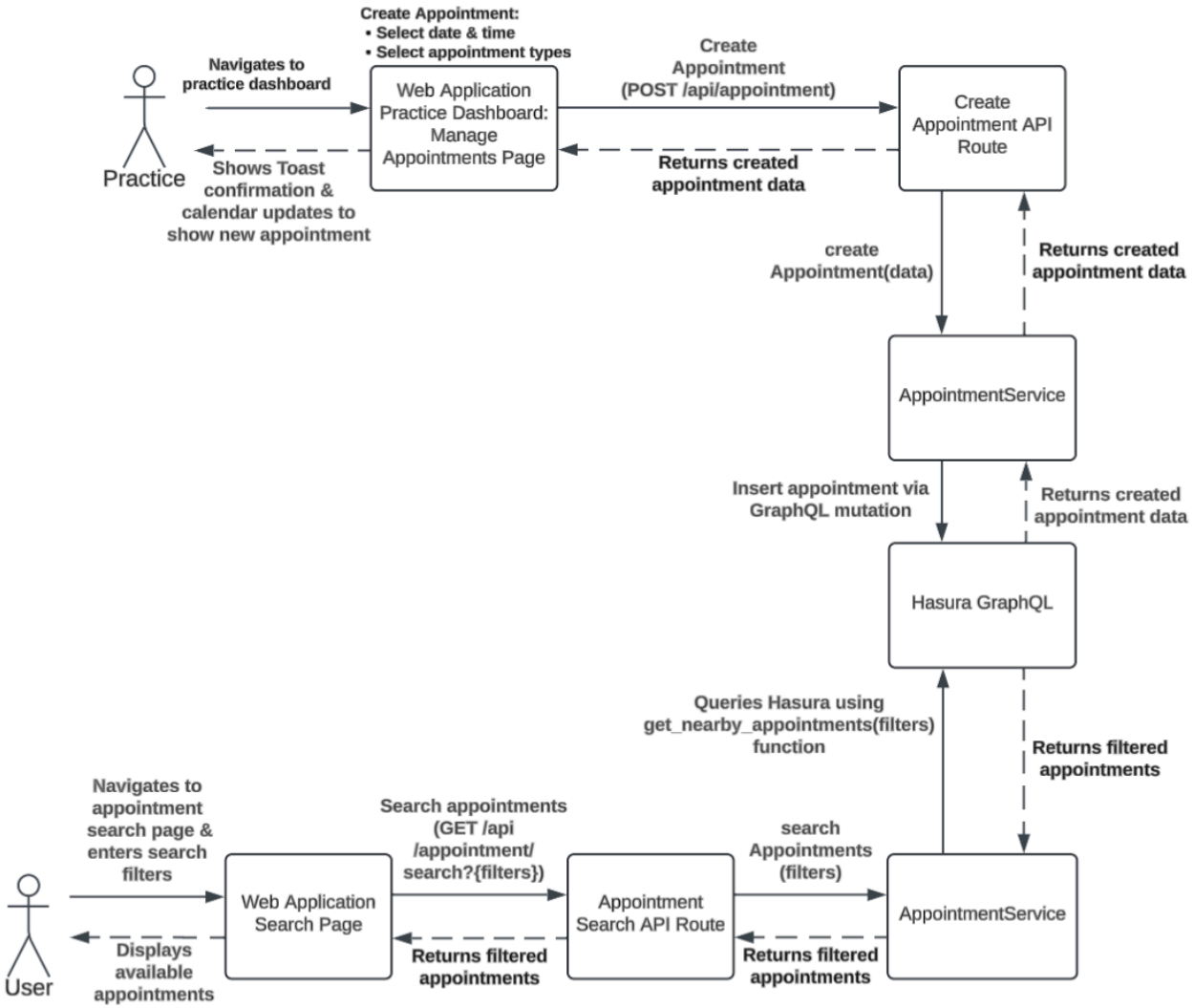


Figure 5.45: Data flow diagram of appointment creation and searching.

Appointment type and prices are stored as key-value pairs within a JavaScript Object Notation Binary (JSONB) column, `services`, within the `appointments` table. This enables dental practices to offer multiple appointment types, such as a checkup and teeth whitening, for a single appointment, example shown in Figure 5.46. As a result, the appointment appears in the search results of a user searching for either a checkup or teeth whitening. The user then selects the type of appointment to book during checkout from the available options. This enhances flexibility and availability for both dental practices and users. JSONB was selected due to its adaptability and performance over alternative column types. Unlike fixed columns, JSONB can adapt to updates to appointment types without schema changes (PostgreSQL Global Development Group, n.d.). Additionally, JSONB supports rich queries, utilised within the `get_nearby_appointments` function, such as type existence (`a.services ? 'checkup'`) and price range filtering (`a.services ->> 'checkup):: numeric < 50`). The binary format of the field partnered with GIN indexing outperforms a standard JSON field due to being able to perform operations without parsing, enabling O(log n) time complexity for key existence (PostgreSQL Global Development Group, n.d.). For an example query, see Appendix J.15.

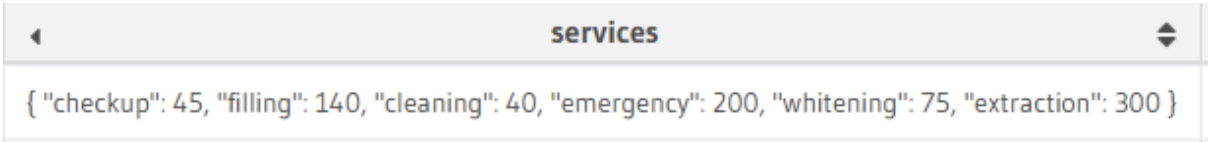


Figure 5.46: services JSON

5.3 Data Orchestration and Rendering

Data orchestration and rendering follows the container-presenter pattern (see Appendix J.16), with dynamic routing used for booking flow persistence (see Appendix J.17) to improve maintainability and UX.

5.4 Third-Party Integrations

The web application integrates with three third-party services:

- **Mailjet**
- **Amazon S3 Bucket**
- **Google Geolocation**

Appendix J.18 details the setup and implementation of each integration, provides validation for the approach and discusses security considerations through role-based and scoped URL access.

6. Research

This section presents the research undertaken and its influence on the project. It includes a literature review, research into appointment types, a competitor analysis, and user research through questionnaires.

6.1 Literature Review: Challenges in UK Dentistry

This literature review explores the current state of both NHS and private dentistry by examining the access barriers to NHS services, the shift towards private dental care, operational inefficiencies, and the consequences of delayed dental treatment. It aims to contextualise and provide validation for a web application designed to enhance appointment utilisation and access.

6.1.1 The NHS Dental Access Crisis

A lack of access to NHS dental appointments is a widely documented issue in the UK, with recent investigations and surveys highlighting the scale and progression of the crisis. A 2022 BBC investigation, contacting 6,880 dental practices, found that 90% of NHS dental practices across the UK were not accepting new adult patients (BBC News, 2022). In certain areas, such as the south-west of England, the percentage of practices not accepting new adult NHS patients increased to 98%. Access was similarly limited for children, with 80% of NHS practices not accepting new child patients (BBC News, 2022).

More recent data from 2024 further illustrates the growing magnitude of the problem. The 2024 GP Survey, run by Ipsos and analysed by the British Dental Association (BDA), revealed that the unmet need for NHS dentistry in England now stands at 13 million people, or approximately 28% of the adult population, an increase of over 8% when compared with 2023 (BDA, 2024a). The NHS appointment availability crisis is projected to worsen, as 74% of practices state an intention to reduce, or further reduce, their NHS work, and 43% indicate an intention to go fully private (BDA, 2023a). General Dental Practice Committee (GDPC) chair Shawn Charlwood warns, “NHS dentistry is running out of road. Every day a broken system remains in force, we lose dentists, while millions struggle to access care”, highlighting a system in decline with no immediate signs of improvement (BDA, 2023a).

6.1.2 Shift to Private Dentistry

The decline in availability of NHS dentistry has forced patients to seek private dental care or to neglect their oral health. The 2024 Experiences of NHS healthcare services in England survey, conducted by the Office for National Statistics and analysed by the BDA, found that 96.9% of those without a dentist who tried to access NHS dental care were unsuccessful (BDA, 2024b; ONS, 2024). Of those who failed to secure care, 11% resorted to private dental care, whilst 78.5% took no action, with 41% citing cost as the major barrier (BDA, 2024b; Office for Health Improvement and Disparities, 2024a; Oral Health Foundation, 2024).

A 2022 poll of 2,026 adults, carried out by Yonder Data Solutions on behalf of Healthwatch England, provides further evidence of the shift to a reliance on private dentistry. The poll found that 17% of respondents felt pressured to pay private fees to access treatment and 24% had to pay privately to receive all the treatment they required (Healthwatch England, 2022).

6.1.3 Operational Inefficiencies in Private Practices

Despite the increasing reliance on private dentistry, many practices face significant operational challenges. A 2019 British Dental Journal article reported on a BDA survey of identified dental practice owners in England, which found that 75% of practices were struggling to fill staff vacancies. This figure has increased from 50% in 2016 and 68% in 2017, indicating a trend of increasing staffing difficulties (British Dental Journal, 2019). The operational difficulties have continued post-COVID, with a 2023 BDA report stating that only 21% of practices have returned to pre-COVID-19 capacity, illustrating ongoing constraints in service delivery (BDA, 2023b). Financial pressures magnify the operational challenges, with the 2024 Dentistry Census concluding that practice profits have dropped by 5% when compared with the previous census conducted in 2021 (Dentistry.co.uk, 2024).

The operational challenges are exacerbated by inefficiencies in appointment utilisation. Although there is no publicly available data indicating the number of missed appointments in private dentistry, NHS outpatient data provides an insight into the appointment utilisation across general healthcare. NHS England reported in 2021/22 that of the 103 million outpatient appointments in England, 7.6% resulted in ‘Did Not Attend’ (NHS England, 2023). This equates to 7.8 million missed appointments. Patient behavioural patterns cannot be presumed to be identical, as private dental practices enforce stricter cancellation and no-show policies that often result in charges. However, appointment utilisation inefficiencies are still likely to be present to some degree, further straining private practices.

Additionally, NexHealth claims that 77% of patients prefer a provider that offers online booking and scheduling, however, only 26% of practices currently offer it (NexHealth Insights, 2025). Upon reviewing the provenance of these figures, it becomes evident that these figures are presented in a manner that supports NexHealth’s commercial interest, as the company provides online booking solutions for dental practices. The 77% figure is drawn from a 2013 survey of 9,015 individuals aged over 65, so it may not reflect current preferences and isn’t representative of all demographics (Accenture, 2013). The 26% figure stems from NexHealth’s State of Dental 2022 survey of 1,271 dental staff (NexHealth, 2022). While these limitations reduce the reliability of the exact figures, they highlight a gap between patient preferences and practice offerings, presenting an opportunity for the adoption of a digital solution to streamline operations and improve patient satisfaction. This combination of operational, financial and technical challenges limits private dental practices’ ability to cater for patients unable to access NHS services.

6.1.4 Consequences of Delayed Dental Care

The operational inefficiencies in private practices, coupled with limited access to NHS dental appointments, have led to widespread delays in dental care, with significant public health implications. As discussed in Section 6.1.2, 78.5% of patients took no action after being unsuccessful in securing NHS dental care, presenting a widespread issue of untreated oral conditions (BDA, 2024b). Statistics published by the Oral Health Foundation (OHF) and surveys conducted by Public Health England (2020) illustrate the extent of this issue, reporting that 31% of adults suffer from tooth decay, with the average number of teeth with untreated decay standing at 2.1 (OHF, n.d.; Public Health England, 2020). Each year, 6 million UK adults experience pain lasting over two weeks caused by a toothache (Oral Health Foundation, n.d.). This not only affects quality of life, but has a notable economic impact. The Denplan 2023 Oral Health Survey of 5,101 adults found that 21% of adults reported severe oral health issues that led them to take time off work, resulting in an estimated 23 million working days being taken as sick leave due to dental pain. As this figure is based on extrapolation from self-reported data, it should be interpreted cautiously (Denplan, 2023). However, it still highlights the wider economic consequences of delayed dental care.

Untreated manageable oral conditions often progress and worsen, leading to more severe issues that require more extensive and expensive dental interventions, such as extractions (Orchard Cottage Dental, 2023). Patients frequently seek care from non-dental healthcare professionals, driven by the limited access to dental care, with 600,000 GP consultations and 200,000 visits annually being accounted for by dental problems costing the NHS an estimated £26 million (British Dental Association, 2016; {my}dentist, 2021). This further strains an already burdened NHS (BDA, 2016).

Children face particularly alarming consequences, with tooth decay being the leading cause for hospital admissions among 5-to-9-year-olds (Royal College of Surgeons of England, 2024). A 2024 survey by the Office for Health Improvement and Disparities found that among five-year-olds with decay, the average number of decayed teeth is 3.5, with 81.4% being left untreated (Office for Health Improvement and Disparities, 2024b). Public Health England reported that every 10 minutes, a child is admitted to hospital for a tooth extraction caused by preventable tooth decay (Public Health England, 2018). These findings show the severity of neglected oral health in children, with tooth decay being largely preventable through early intervention, and many of the further consequences being avoidable (Office for Health Improvement and Disparities, 2024b). This underscores the need for more accessible dental care to ensure timely treatment, prevent unnecessary suffering and invasive treatments, and reduce the strain on the NHS.

6.1.5 Conclusion and Influence on the Project

The literature consistently highlights a UK dental care system under strain, characterised by limited NHS dental access, with 13 million patients unable to access care, increased reliance on private dental practices suffering from staffing and capacity challenges, and the serious public health and financial consequences of delayed dental care (BDA, 2024a; BDA, 2024b; British Dental Journal, 2019). With 31% of UK adults currently having untreated tooth decay, leading to an estimated 23 million workdays lost annually due to dental issues (Denplan, 2023; OHF, n.d.). These findings shaped the project's aim and objectives, while providing validation for a solution that enhances private dental appointment visibility and accessibility, addressing the challenges faced by patients and practices.

This project responds to these issues by developing a centralised web application for searching, comparing and booking appointments, designed to provide appointment availability and reduce access barriers for patients, while optimising appointment utilisation for private practices. The web application plays a role in alleviating the pressure on NHS services by improving access to timely dental care, increasing early interventions and reducing the likelihood of preventable conditions escalating into severe issues that require expensive treatment.

The web application does not address the systemic issues faced by NHS dentistry or eliminate financial barriers for all patients. However, the solution provides a step towards optimising the private sector resources to better meet patient demand, indirectly easing the strain on NHS resources.

For future research, surveys should be conducted with private dental practices to obtain accurate figures of operational inefficiencies and challenges to aid future feature prioritisation in subsequent expansions.

6.2 Appointment Types and Durations

Market research was conducted to determine the frequent procedures offered by various UK dental practices. Full details of the research are available within Appendix K.1.

6.2.1 Influence on Web Application

The analysis of appointment types and their duration directly informed several aspects of the web application:

- **FR7**: The web application will support 6 appointment types: Check-Up, Cleaning, Whitening, Filling, Extraction and Emergency.
- **FR7**: The scheduling system within the web application will support dynamic appointment length.
- **FR7, FR29**: The implementation of the appointment types will be flexible, supporting multiple appointment types per appointment, and expandable, enabling future iterations of the web application to support additional appointment types without breaking the existing functionality.

The pricing for an appointment will be based on the appointment type, with each practice setting its price for a given treatment.

6.3 Competitor Analysis

The full competitor analysis is available within Appendix K.2. This section highlights a selection of key influences.

6.3.1 Bupa Dental Care

Bupa Dental Care operates a network of over 350 NHS and private dental practices across the UK (Bupa Dental Care, 2025). The web application enables users to search for a range of treatment options and to book an appointment with a dental practice within the Bupa network.

6.3.1.1 Influence on DentalConnect:

- **Practice Registration**: Analysis of the information that Bupa provided about a given practice informed the data fields required during DentalConnect's practice registration, as illustrated within the wireframe in Appendix D.3.
- **Practice Verification**: Bupa's web application only supports practices within their network. DentalConnect supports independent dental practices, resulting in a verification process, performed by a system admin, post practice registration to ensure only legitimate practices are able to list appointments.
- **User Registration**: Bupa validated that dental appointments can be booked with basic personal data (name and contact information). This informed the data collected during DentalConnect's user registration, as seen in the wireframe in Appendix D.2.
- **Appointment Search Functionality**: Bupa's practice-based search functionality to find appointments highlighted a key limitation by not allowing users to directly filter, view and compare appointments. DentalConnect addresses this by allowing users to search appointments directly, applying filters and sorting upfront to streamline viewing available appointments and providing price comparison across practices.
- **Review System**: Bupa's web application allows users to leave a review for a practice. DentalConnect will contain a review system enabling users to share feedback and help other users make informed decisions.
- **Tooltips**: Bupa's web application contains tooltips which offer additional context and guidance, improving the user experience. DentalConnect will use tooltips where appropriate, such as explaining options within preference settings.

6.3.1 {my}dentist

A web application created by the UK's largest dental provider, {my}dentist, with over 500 NHS and private dental practices ({my}dentist, 2025). It enables users to search for practices and treatments, overlapping the project's scope by facilitating the searching of practices and treatments. However, it does not allow a user to book a specific appointment through the web application. The {my}dentist web application also supports searching for orthodontists, however this is out of the project scope.

6.3.2.1 Influence on DentalConnect:

- **User Interface Design**: The card view employed by {my}dentist informed DentalConnect's approach to displaying appointments within search results. The cards provide responsiveness by adapting their layout and size across a range of devices, including desktop and mobile. The cards will dynamically update to display a distance field when a user provides location data, as shown in Figures 6.1 and 6.2.
- **Performance**: {my}dentist's web application was not performant and slow to navigate, as verified by Google Lighthouse testing detailed in 6.3. This reinforces the need for DentalConnect to be performant.

Figure 6.1: {my}dentist practice card view

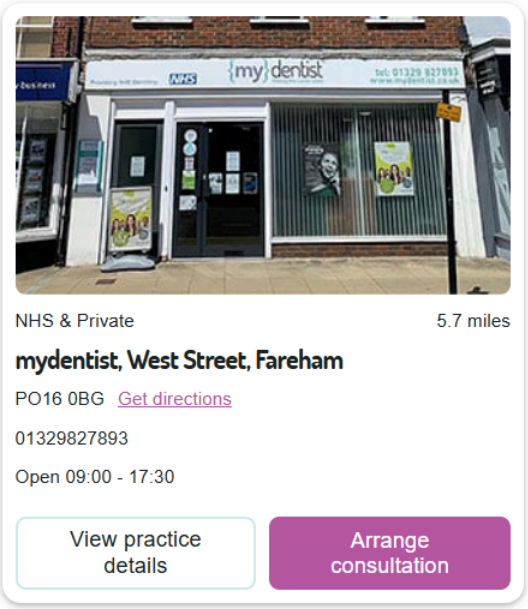


Figure 6.2: {my}dentist practice card view with distance

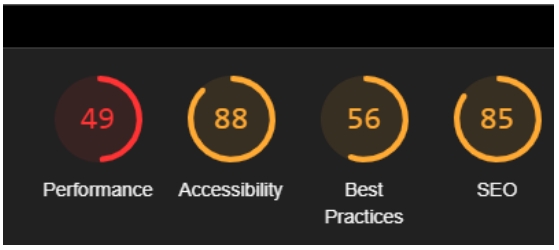


Figure 6.3: {my}dentist Homepage Google Lighthouse Result (Desktop)

6.4 User Research and Validation

User research was conducted through three questionnaires to validate the project, guide design decisions and evaluate usability. Participant consent forms were used within the guidance of CI601.

6.4.1 Preliminary User Research Questionnaire

This questionnaire highlighted the accessibility challenges and for demand for an aggregate dental appointment web application. Full details are available within Appendix K.3.

6.4.2 Design Questionnaire

The design questionnaire informed UI design to align user preferences with full details in Appendix K.4. These insights informed the UI wireframes in Appendix D.

6.4.3 Usability Questionnaire

To evaluate the intuitiveness of the web application, a usability questionnaire was conducted following the user journeys. Full details are in Appendix K.5.

The key takeaways are summarised within Figure 6.4.

Issue Identified	Solution Implemented
Confusion with the meaning of certain options within the practice preferences page.	Tooltip coverage has been expanded to every setting. Hovering over the "i" icon displays a brief explanation of the preference (Figure 6.5).
Unsaved changes are lost without warning when exiting both the user and practice manage details and preference pages.	A confirmation pop-up now shows if unsaved changes are detected when attempting to navigate away from the page. The pop-up provides the option to confirm or stay on the page (Figure 6.6).
Appointment search filters reset to default upon returning back to the search page after clicking on an appointment.	Search filters are saved to browser local storage, allowing them to persist when the user navigates away and returns to the search page (Figure 6.7).

Figure 6.4: Usability issues identified and solutions implemented.

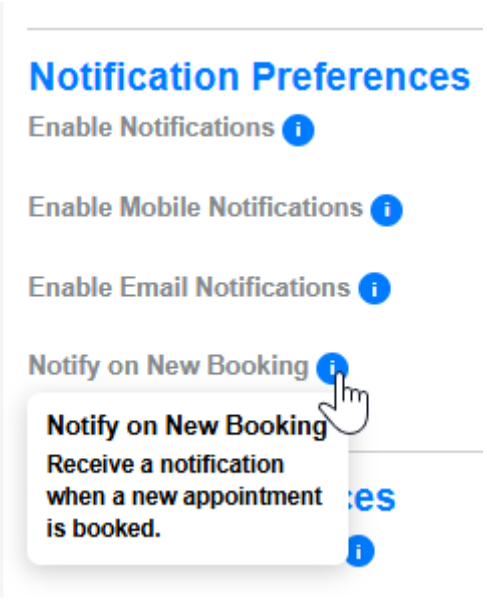


Figure 6.5: Expanded Tooltip coverage example.

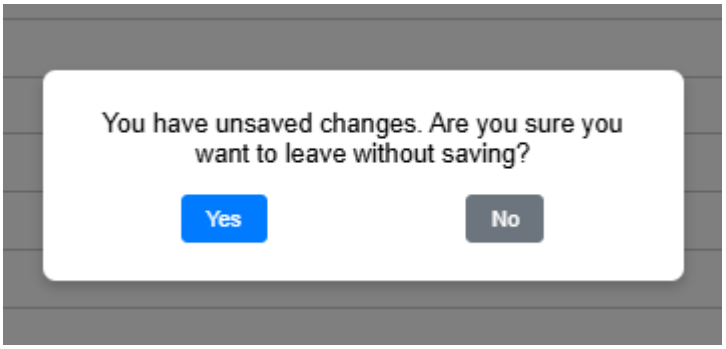


Figure 6.6: Unsaved changes warning pop-up.

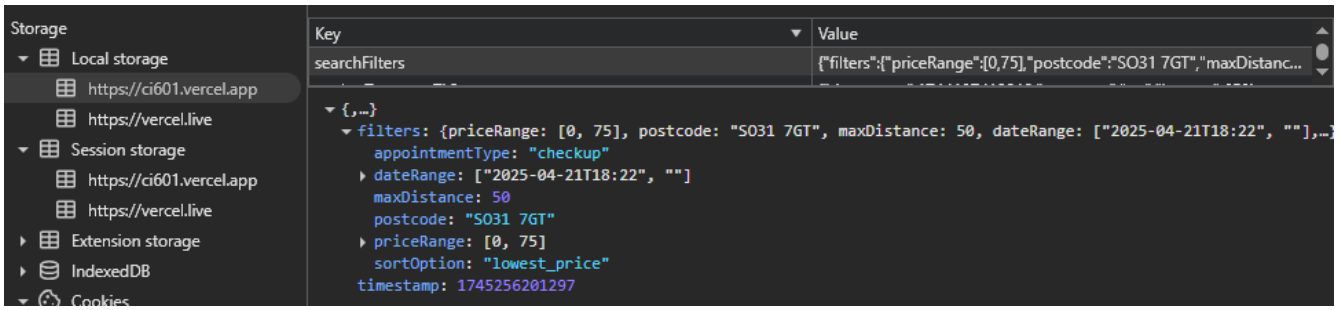


Figure 6.7: Search filters stored in browser local storage.

7. Critical Review

This section provides a critical evaluation of the project, assessing the software artefact against its objectives and requirements. It highlights key successes, lessons learned and areas for improvement..

7.1 Evaluation of Project Success

The project can be considered successful as the web application achieved its primary aim of improving access to private dental care by enabling users to search, compare and book available appointments, while allowing practices to list unused appointments. The MVP was delivered through the successful implementation of all Must-Have functional requirements, ensuring that all user journeys were fully operational. The FDD Agile sprints accommodated the evolving project, from a last-minute appointment web application to an appointment aggregator, and provided ample time for a majority of Should-Have requirements to be fulfilled. Additionally, all non-functional requirements were achieved, contributing to an accessible, responsive, and performant web application.

In total, 100% of the Must-Have functional requirements and non-functional requirements were implemented successfully. 56% of the Should-Have requirements were implemented, extending features beyond the MVP. Although no Could-Have requirements were completed due to time constraints, this highlights the effectiveness of the MoSCoW prioritisation strategy and focus on delivering the essential user-facing functionality. This delivery rate reflects effective sprint planning and showcases the flexibility provided by an Agile approach, which supported adaptive development throughout the project lifecycle.

Beyond meeting the project's requirements, the web application adheres to industry best practices through the use of multiple architectural patterns. It follows a modular three-tier architecture, supporting clear separation of concerns, conforms to the single responsibility principle and employs the container-presenter pattern. This produced a clean, maintainable codebase designed to support scalability and long-term extensibility.

A significant achievement throughout development was the 0% error and timeout rate of the web application's Vercel deployment. This is attributed to a broad test case coverage and a robust CI/CD pipeline. The pipeline ensured changes were not deployed if they broke the web application's deployment build or did not follow industry best practices.

7.2 Lessons Learned

A key lesson learnt during this project was recognising the difference between learning the fundamentals of a new technology and fully understanding the best practices and how to implement them. This was particularly apparent when learning TypeScript and Next.js, with initial development feeling intuitive and features being delivered on time within early sprints. However, limitations within the codebase became apparent, with areas lacking structure. As the project scaled, issues such as duplicate TypeScript types and disorganised Next.js routes and services emerged, highlighting the need for significant refactoring.

The refactoring process was time-consuming, particularly as it involved learning best practices simultaneously. Although refactoring is a natural part of learning and using new technologies, insufficient time was allocated for it within the project plan. In future projects, more time will be allocated for iterative refactoring within the project timeline, and research into technology-specific best practices will be performed earlier. An example where a full refactor was not performed due to time constraints, is the absence of a dedicated `hasuraService` for handling the execution of GraphQL queries and mutations, resulting in boilerplate and repetitive code across services.

Delivering core functionality was prioritised to accelerate the delivery of an MVP. Although a fine prioritisation as it aligns with the principle of the project plan, it resulted in the unnecessary de-scoping of certain requirements and development tasks when moving onto the next feature preemptively. A notable example is the de-scoping of GraphQL Code Generation (codegen) during early sprints due to the high perceived overhead of setup and learning, with concerns that it may delay MVP delivery. However, as development continued, this decision introduced significant technical debt. Without codegen, TypeScript types had to be manually defined and maintained, eliminating one of the major advantages of using Hasura GraphQL with TypeScript. Post-MVP, the time and effort required to retroactively implement codegen were too great within the remaining project timeframe. In subsequent projects, greater confidence will be placed in the project plan, which had allocated sufficient time for learning and implementing codegen while still providing early MVP delivery. This experience highlights the importance of recognising when an initial time investment in tooling and best practices can significantly reduce long-term complexity, even if it slows progress in the short term. It also shows the importance of not automatically assigning a de-scoped task the lowest priority in the subsequent sprint as it leads to repeated deferral.

The technology choices were appropriate, aligned with the web application's requirements, and contributed to the production of a performant, responsive and maintainable web application. If tasked with a similar project in the future, Hasura would likely not be used. While Hasura's schema management was streamlined through the CLI and auto-generated a developer-friendly GraphQL API which helped with rapid development, its limitations became more apparent as more complex functionality was required. A key challenge arose from Hasura's handling of computed fields and custom functions, specifically the inability to directly filter on computed fields. This was required for users to be able to filter and sort appointments by distance while searching. This constraint resulted in a complex workaround, fully detailed in Section 5.2.3, and delayed development. Although the issue was resolved, it showed the importance of thoroughly evaluating a tool and investigating any limitations. In future projects, further research and prototyping will be conducted to assess how well a technology accommodates complex, domain-specific requirements beyond initial development speed and ease of use.

7.3 Future Improvements and System Limitations

The project successfully delivered a fully functional web application, encompassing the core MVP functionality alongside the majority of Should-Have. However, due to time constraints, certain requirements had to be de-scoped. The following two key requirements would have significantly enhanced the web application's value and applicability.

- **Public Facing API**: An API would have enabled pre-existing practice booking software to integrate with DentalConnect, making appointment creation and management an automated process, providing more applicability within industry.
- **Payment Integration**: Payment integration would have made DentalConnect fully self-serving. However, this requirement was realised too late within development. The nature of the web application, with disputes, would require a holding and distribution of payments system.

Additional improvements would include expanding test-case coverage to Should-Have requirements by placing an emphasis on adding tests throughout development. Although tests were run manually, adding the automated tests to the CI/CD pipeline would have ensured consistent validation. Documentation should have been maintained throughout development to improve coverage and long-term maintainability. Future project plans will also account more realistically for other academic deadlines, as overlapping university assignments led to the de-scoping of most of Sprint 7 and all of Sprint 8.

References

Accenture, 2013. Consumer health survey: Senior citizens – 2013 Accenture consumer survey on patient engagement. Available at: https://www.criticaleye.com/inspiring/insights-servfile.cfm?id=3869 (Accessed: 23 April 2025).

Agile Alliance, 2001. Manifesto for agile software development. Available at: https://agilemanifesto.org/ (Accessed: 28 October 2024).

Ahmad, K.S., Ahmad, N., Tahir, H. and Khan, S., 2017. Fuzzy_MoSCoW: A fuzzy based MoSCoW method for the prioritization of software requirements. 2017 International Conference on Intelligent Computing, Instrumentation and Control Technologies (ICICICT). Available at: https://ieeexplore.ieee.org/document/8342602 (Accessed: 7 November 2024).

Amazon Web Services (AWS), n.d. Amazon Simple Storage Service User Guide. Available at: https://docs.aws.amazon.com/AmazonS3/latest/userguide/Welcome.html (Accessed: 1 May 2025).

Amazon Web Services (AWS), 2025. What's the difference between MySQL and PostgreSQL? Available at: https://aws.amazon.com/compare/the-difference-between-mysql-vs-postgresql/ (Accessed: 20 April 2025).

Angular, 2022. Two-way binding. Available at: https://v17.angular.io/guide/two-way-binding (Accessed: 29 April 2025).

Angular, 2025. Dependency injection. Available at: https://angular.dev/guide/di (Accessed: 29 April 2025).

Assembly Dental, n.d. 4 common types of general dentistry services. Available at: https://somervilledentist.com/blog/common-types-general-dentistry-services-cip133/ (Accessed: 4 April 2025).

Atlassian, 2024a. The agile coach: Atlassian's guide to agile development. Available at: https://www.atlassian.com/agile (Accessed: 31 October 2024).

Atlassian, 2024b. Waterfall project management methodology. Available at: https://www.atlassian.com/agile/project-management/waterfall-methodology (Accessed: 31 October 2024).

BBC News, 2022. Full extent of NHS dentistry shortage revealed by far-reaching BBC research. Available at: https://www.bbc.co.uk/news/health-62253893 (Accessed: 6 April 2025).

British Dental Association (BDA), 2016. NHS charges are masking cuts and driving patients to GPs, say dentists. 6 September. British Dental Association. Available at: https://www.nature.com/articles/sj.bdj.2016.670.pdf (Accessed: 6 April 2025).

British Dental Association (BDA), 2023a. Half of dentists have cut NHS commitment, with more to come. Available at: https://www.bda.org/news-and-opinion/news/half-of-dentists-have-cut-nhs-commitment-with-more-to-come/ (Accessed: 28 March 2025).

British Dental Association (BDA), 2023b. Dentistry: COVID impact on scale unseen in any other part of NHS. Available at: https://www.bda.org/media-centre/dentistry-covid-impact-on-scale-unseen-in-any-other-part-of-nhs/ (Accessed: 6 April 2025).

British Dental Association (BDA), 2024a. 13 million unable to access NHS dentistry. Available at: https://www.bda.org/media-centre/13-million-unable-to-access-nhs-dentistry/ (Accessed: 28 March 2025).

British Dental Association (BDA), 2024b. Dentists: 97% of new patients unable to access NHS care. 10 October. Available at: https://www.bda.org/media-centre/dentists-97-of-new-patients-unable-to-access-nhs-care/ (Accessed: 6 April 2025).

British Dental Journal, 2019. Three quarters of NHS dental practices failing to fill vacancies. British Dental Journal, 226, p. 480. Available at: https://www.nature.com/articles/s41415-019-0221-y (Accessed: 6 April 2025).

Bupa Dental Care, 2025. Dental practices. Available at: https://www.bupa.co.uk/dental/dental-care/practices (Accessed: 1 April 2025).

Cheslyn Hay Dental Practice, n.d. How long does professional teeth whitening take? Walsall: Cheslyn Hay Dental Practice. Available at: https://cheslynhaydental.co.uk/how-long-does-professional-teeth-whitening-take/ (Accessed: 4 April 2025).

Chiaramonte, M., 2024. What is a 3-tier application architecture? Definition and examples. vFunction. Available at: https://vfunction.com/blog/3-tier-application/ (Accessed: 23 March 2025).

Christensen, E., 2023. Postgres subquery powertools: Subqueries, CTEs, materialized views, window functions, and LATERAL join. Available at: https://www.crunchydata.com/blog/postgres-subquery-powertools-subqueries-ctes-materialized-views-window-functions-and-lateral (Accessed: 20 April 2025).

Chung, L., Nixon, B.A., Yu, E. and Mylopoulos, J., 2012. Non-functional requirements in software engineering. New York: Springer Science & Business Media.

Clegg, D. and Barker, R., 1994. Case method fast-track: A RAD approach. Boston: Addison-Wesley Longman Publishing Co.

Coad, P., Lefebvre, E. and De Luca, J., 1999. Java modeling in color with UML: Feature-driven development. Available at: https://csis.pace.edu/~marchese/CS616/Agile/FDD/fdd2.pdf (Accessed: 16 April 2025).

Codd, E.F., 1990. The relational model for database management: Version 2. Boston: Addison-Wesley. Available at: https://dl.acm.org/doi/pdf/10.5555/77708 (Accessed: 23 March 2025).

Cohn, M., 2009. Succeeding with agile: Software development using Scrum. Boston: Addison-Wesley.

Crescent Heights Dental Clinic, n.d. 10 most common dental procedures and how they work. Available at: https://www.crescentdental.ca/10-most-common-dental-procedures-and-how-they-work/ (Accessed: 4 April 2025).

CSS Modules, 2023. CSS modules README. GitHub. Available at: https://github.com/css-modules/css-modules/blob/master/README.md (Accessed: 23 March 2025).

Denplan, 2023. Oral health survey 2023. Available at: https://www.denplan.co.uk/the-core/industry-and-company-news/oral-health-survey-23 (Accessed: 6 April 2025).

Dentistry.co.uk, 2024. 2024 dentistry census: Key findings. 8 February. Available at: https://dentistry.co.uk/2024/02/08/2024-dentistry-census-key-findings/ (Accessed: 6 April 2025).

Dijkstra, E.W., 1974. On the role of scientific thought. EWD447. Available at: https://www.cs.utexas.edu/users/EWD/ewd04xx/EWD447.PDF (Accessed: 23 March 2025).

Dybå, T. and Dingsøyr, T., 2008. Empirical studies of agile software development: A systematic review. Information and Software Technology, 50(9–10), pp.833–859. Available at: https://www.sciencedirect.com/science/article/pii/S0950584908000256 (Accessed: 20 April 2025).

Elightwalk Technology, 2023. What are the best practices for styling in React? Medium. Available at: https://medium.com/@elightwalk/what-are-the-best-practices-for-styling-in-react-e9816e7912d4 (Accessed: 23 March 2025).

Elmasri, R. and Navathe, S.B., 2015. Fundamentals of database systems. 7th ed. Boston: Pearson. Available at: https://www.auhd.edu.ye/upfiles/elibrary/Azal2020-01-22-12-28-11-76901.pdf (Accessed: 23 March 2025).

Emadamerho-Atori, N., 2024. Client-side rendering (CSR) vs. server-side rendering (SSR). Prismic. Available at: https://prismic.io/blog/client-side-rendering-vs-server-side-rendering (Accessed: 23 March 2025).

GeeksforGeeks, 2021. Advantages and disadvantages of three-tier architecture in DBMS. Available at: https://www.geeksforgeeks.org/advantages-and-disadvantages-of-three-tier-architecture-in-dbms/ (Accessed: 23 March 2025).

GeeksforGeeks, 2024. Difference between TypeScript and JavaScript. Available at: https://www.geeksforgeeks.org/difference-between-typescript-and-javascript/ (Accessed: 3 November 2024).

GeeksforGeeks, 2025a. Difference between Virtual DOM and Real DOM. Available at: https://www.geeksforgeeks.org/difference-between-virtual-dom-and-real-dom/ (Accessed: 29 April 2025).

GeeksforGeeks, 2025b. ReactJS unidirectional data flow. Available at: https://www.geeksforgeeks.org/reactjs-unidirectional-data-flow/ (Accessed: 29 April 2025).

GeeksforGeeks, 2025c. ReactJS Virtual DOM. Available at: https://www.geeksforgeeks.org/reactjs-virtual-dom/ (Accessed: 29 April 2025).

Google Developers, 2016. Lighthouse overview. Available at: https://developer.chrome.com/docs/lighthouse/overview (Accessed: 23 March 2025).

Google Developers, 2019. First contentful paint. Available at: https://developer.chrome.com/docs/lighthouse/performance/first-contentful-paint (Accessed: 23 March 2025).

Google Developers, 2024. Device mode: Simulate mobile devices with device emulation. Available at: https://developer.chrome.com/docs/devtools/device-mode (Accessed: 23 March 2025).

Google Developers, 2025. Geocoding API overview. Available at: https://developers.google.com/maps/documentation/geocoding/overview (Accessed: 23 March 2025).

GraphQL, n.d. Introduction to GraphQL. Available at: https://graphql.org/ (Accessed: 12 November 2024).

Hasura, n.d.a. Computed fields. Available at: https://hasura.io/docs/2.0/schema/bigquery/computed-fields/ (Accessed: 23 March 2025).

Hasura, n.d.b. Custom functions. Available at: https://hasura.io/docs/2.0/schema/postgres/custom-functions/ (Accessed: 23 March 2025).

Hasura, n.d.c. Quickstart: Hasura CLI. Available at: https://hasura.io/docs/2.0/hasura-cli/quickstart/ (Accessed: 23 March 2025).

Hasura, 2020 What is Hasura? Hasura. Available at: https://hasura.io/blog/what-is-hasura-ce3b5c6e80e8 (Accessed 14 Apr. 2025).

Hasura, 2021. Views in Microsoft SQL Server. Available at: https://hasura.io/learn/database/microsoft-sql-server/views/ (Accessed: 23 March 2025).

Hayes, A., 2025. Risk analysis: Definition, types, limitations, and examples. Investopedia. Available at: https://www.investopedia.com/terms/r/risk-analysis.asp (Accessed: 14 April 2025)..

Healthwatch England, 2022. Key findings from Healthwatch England's national dental polling in 2022. Available at: https://www.healthwatch.co.uk/sites/healthwatch.co.uk/files/Key%20findings%20from%20Healthwatch%20England%E2%80%99s%20national%20dental%20polling%20in%202022.pdf (Accessed: 23 April 2025).

Highsmith, J., 2009. Agile project management: Creating innovative products. 2nd ed. Boston: Addison-Wesley. Available at: https://books.google.co.uk/books?id=VuFpkztwPaUC (Accessed: 20 April 2025).

IBM, n.d. What is three-tier architecture? Available at: https://www.ibm.com/think/topics/three-tier-architecture (Accessed: 23 March 2025).

IBM, 2024. What is high availability? Available at: https://www.ibm.com/think/topics/high-availability (Accessed: 29 April 2025).

iCare Dental, n.d. 7 common procedures and their average appointment lengths at the dentist. London: iCare Dental. Available at: https://icaredental.co.uk/7-common-procedures-of-appointment-lengths-at-the-dentist/ (Accessed: 4 April 2025).

Ikius, 2024. Vercel vs. Netlify: Which platform is best for your needs? Available at: https://ikius.com/blog/vercel-vs-netlify (Accessed: 1 November 2024).

International Organisation for Standardisation (ISO), 2018. ISO 9241-11:2018 – Ergonomics of human-system interaction – Part 11: Usability: Definitions and concepts. Available at: https://www.iso.org/standard/63500.html (Accessed: 30 April 2025)

Krasamo, 2022. The agile development process for mobile apps. Krasamo. Available at: https://www.krasamo.com/agile-development-process/ (Accessed: 30 April 2025).

Martin, R.C., 2003. Agile software development: Principles, patterns, and practices. Upper Saddle River, NJ: Prentice Hall.

Mailjet, n.d. Email API Developer Documentation. Available at: https://dev.mailjet.com/ (Accessed: 30 April 2025).

MDN Web Docs, 2024. Base64. Available at: https://developer.mozilla.org/en-US/docs/Glossary/Base64 (Accessed: 16 December 2024).

Microsoft, 2025. The TypeScript handbook: Introduction. Available at: https://www.typescriptlang.org/docs/handbook/intro.html (Accessed: 20 April 2025).

Mintel, 2018. Nip, tuck or fill: 31% of Brits are interested in having cosmetic surgery in the future. Available at: https://www.mintel.com/press-centre/beauty-and-personal-care/nip-tuck-or-fill-31-of-brits-are-interested-in-having-cosmetic-surgery-in-the-future (Accessed: 4 April 2025).

Moyo, S. and Mnkandla, E., 2020. A novel lightweight solo software development methodology with optimum security practices. IEEE Access, 8, pp.117573–117584. Available at: https://ieeexplore.ieee.org/document/8978533 (Accessed: 20 April 2025).

{my}dentist. (2021) The Great British Oral Health Report. {my}dentist. Available at: https://dentistry.co.uk/app/uploads/2022/01/the-great-british-oral-health-report-2021.pdf (Accessed 4 Apr. 2025).

{my}dentist, 2025. {my}dentist | The UK’s largest dental provider. Available at: https://www.mydentist.co.uk/ (Accessed: 1 April 2025).

Naumov, A., 2020. Separation of concerns in software design. Available at: https://nalexn.github.io/separation-of-concerns/ (Accessed: 23 March 2025).

NexHealth, 2022. State of dental 2022: The year of the patient. Available at: https://www.nexhealth.com/stateofdental2022 (Accessed: 23 April 2025).

NexHealth Insights, 2025. Top 7 dental practice marketing statistics. Available at: https://www.nexhealth.com/resources/dental-marketing-statistics (Accessed: 23 April 2025).
NHS Digital, 2023. NHS dental statistics for England, 2022–23: Annual report. 24 August. Available at: https://digital.nhs.uk/data-and-information/publications/statistical/nhs-dental-statistics/2022-23-annual-report (Accessed: 4 April 2025).

NHS England, 2023. Reducing did not attends (DNAs) in outpatient services. Available at: https://www.england.nhs.uk/long-read/reducing-did-not-attends-dnas-in-outpatient-services/ (Accessed: 6 April 2025).

Nielsen, J., 1993. Usability engineering. San Francisco: Morgan Kaufmann. Available at: https://dl.acm.org/doi/pdf/10.5555/2821575 (Accessed: 23 March 2025).

Nijhawan, S., 2022. Next.js vs. React: Which framework is better for frontend? Available at: https://www.vlinkinfo.com/blog/next-js-vs-react-which-framework-is-better-for-frontend/ (Accessed: 7 November 2024).

NPM, n.d. @graphql-codegen/cli. Available at: https://www.npmjs.com/package/@graphql-codegen/cli (Accessed: 14 April 2025).

Office for Health Improvement and Disparities, 2024a. Adult oral health survey 2021: Service use and barriers to accessing care. London: GOV.UK. Available at: https://www.gov.uk/government/statistics/adult-oral-health-survey-2021/adult-oral-health-survey-2021-service-use-and-barriers-to-accessing-care (Accessed: 4 April 2025).

Office for Health Improvement and Disparities, 2024b. National dental epidemiology programme (NDEP) for England: Oral health survey of 5 year old schoolchildren 2024. GOV.UK. Available at: https://www.gov.uk/government/statistics/oral-health-survey-of-5-year-old-schoolchildren-2024/national-dental-epidemiology-programme-ndep-for-england-oral-health-survey-of-5-year-old-schoolchildren-2024 (Accessed: 6 April 2025).

Office for National Statistics (ONS), 2024. Experiences of NHS healthcare services in England: Wave 1, September 2024. [Dataset]. Available at: https://www.ons.gov.uk/peoplepopulationandcommunity/healthandsocialcare/healthcaresystem/datasets/experiencesofnhshealthcareservicesinengland (Accessed: 16 April 2025).

Open Minds, 2025. The 9s of availability: decoding the metrics of uptime. Available at: https://www.openminds.co.uk/the-9s-of-availability-decoding-the-metrics-of-uptime/ (Accessed: 29 April 2025).

Oral Health Foundation (OHF), n.d. Oral health statistics in the UK. Available at: https://www.dentalhealth.org/oral-health-statistics (Accessed: 4 April 2025).

Oral Health Foundation (OHF), 2024. UK could face dental health crisis as costs soar, says Oral Health Foundation. 18 June. Available at: https://www.dentalhealth.org/news/uk-could-face-dental-health-crisis-as-costs-soar-says-oral-health-foundation (Accessed: 6 April 2025).

Orchard Cottage Dental, 2023. What are the risks of neglecting dental hygiene? Available at: https://orchardcottagedental.co.uk/the-risks-of-neglecting-dental-hygiene/ (Accessed: 6 April 2025).

PostGIS Project, n.d.a. Introduction to PostGIS: 15. Spatial indexing. Available at: https://postgis.net/workshops/postgis-intro/indexing.html (Accessed: 23 March 2025).

PostGIS Project, n.d.b. PostGIS 3.4.2dev manual: ST_Distance. Available at: https://postgis.net/docs/ST_Distance.html (Accessed: 23 March 2025).

PostGIS Project, 2023. About PostGIS. Available at: https://postgis.net/ (Accessed: 23 March 2025).

PostgreSQL Global Development Group, n.d.a. JSON types. PostgreSQL Documentation. Available at: https://www.postgresql.org/docs/current/datatype-json.html (Accessed: 23 March 2025).

PostgreSQL Global Development Group, n.d.b. GIN indexes. PostgreSQL Documentation. Available at: https://www.postgresql.org/docs/current/gin.html (Accessed: 23 March 2025).

Project Management Institute, 2010. The value of project management. Available at: https://www.pmi.org/-/media/pmi/documents/public/pdf/white-papers/value-of-project-management.pdf?rev=893cecac1fec4ea9b8557b80aab3213b (Accessed: 16 April 2025).

Public Health England, 2018. Every 10 minutes a child in England has a rotten tooth removed. 6 April. GOV.UK. Available at: https://www.gov.uk/government/news/every-10-minutes-a-child-in-england-has-a-rotten-tooth-removed (Accessed: 6 April 2025).

Public Health England, 2020. Oral health survey of adults attending general dental practices 2018. Available at: https://www.gov.uk/government/publications/oral-health-survey-of-adults-attending-dental-practices-2018 (Accessed: 6 April 2025).

React, 2025a. Conditional rendering – Learn React. Meta Platforms, Inc. Available at: https://react.dev/learn/conditional-rendering (Accessed: 14 April 2025).

React, 2025b. React: The library for web and native user interfaces. Meta Platforms, Inc. Available at: https://react.dev/ (Accessed: 14 April 2025).

Reils, E., 2011. The lean startup. Available at: https://ia800509.us.archive.org/7/items/TheLeanStartupErickRies/The%20Lean%20Startup%20-%20Erick%20Ries.pdf (Accessed: 25 October 2024).

Reis, J. and Figueiredo, R., 2024. Angular vs React: A comparison of both frameworks. Imaginary Cloud. Available at: https://www.imaginarycloud.com/blog/angular-vs-react (Accessed: 14 April 2025).

Roach, J., 2024. PostgreSQL vs. MySQL: Choosing the right database for your project. DataCamp. Available at: https://www.datacamp.com/blog/postgresql-vs-mysql (Accessed: 14 April 2025).

Royal College of Surgeons of England, 2024. Dental surgeons: Too many children admitted to hospital for tooth decay. Available at: https://www.rcseng.ac.uk/news-and-events/media-centre/press-releases/fds-dental-admissions-sept-24 (Accessed: 6 April 2025).

Royce, W., 1970. Managing the development of large software systems. Available at: https://blog.jbrains.ca/assets/articles/royce1970.pdf (Accessed: 3 November 2024).

Siddhpara, V., 2025. React vs Angular: Which JS framework to choose for front-end development? Radixweb. Available at: https://radixweb.com/blog/react-vs-angular (Accessed: 14 April 2025).

Stack Overflow, 2024. Stack Overflow developer survey 2024. Available at: https://survey.stackoverflow.co/2024/technology (Accessed: 14 April 2025).

Stanke, B., 2024. Feature-driven development: The pros, cons, and how it compares to Scrum. Available at: https://www.bobstanke.com/blog/feature-driven-development (Accessed: 16 April 2025).

Stapleton, J., 2003. DSDM: Business focused development. 2nd ed. Harlow: Pearson Education. Available at: https://books.google.co.uk/books?id=OwTlDyM0O-0C (Accessed: 24 April 2025).

State of JavaScript, 2024. Front-end frameworks. Available at: https://2024.stateofjs.com/en-US/libraries/front-end-frameworks/ (Accessed: 14 April 2025).

Team Asana, 2025. Risk matrix template: How to assess risk for project success (with examples). Asana. Available at: https://asana.com/resources/risk-matrix-template (Accessed: 14 April 2025).

The Guild, 2025. GraphQL codegen. Available at: https://the-guild.dev/graphql/codegen (Accessed: 12 November 2024).

The Yardley Clinic, n.d. What are the most common dental treatments? Available at: https://theyardleyclinic.co.uk/blog/what-are-the-most-common-dental-treatments/ (Accessed: 4 April 2025).

Thomson Reuters, 2024. Risk analysis: An overview. Thomson Reuters Legal. Available at: https://legal.thomsonreuters.com/blog/what-is-risk-analysis/ (Accessed: 14 April 2025).

TypeScript, 2025. TypeScript: JavaScript with syntax for types. Microsoft. Available at: https://www.typescriptlang.org/ (Accessed: 14 April 2025).

Vercel, n.d.a. API routes. Next.js Documentation. Available at: https://nextjs.org/docs/pages/building-your-application/routing/api-routes (Accessed: 23 March 2025).

Vercel, n.d.b. Composition patterns. Next.js Documentation. Available at: https://nextjs.org/docs/app/building-your-application/rendering/composition-patterns (Accessed: 23 March 2025).

Vercel, n.d.c. Server components. Next.js Documentation. Available at: https://nextjs.org/docs/app/building-your-application/rendering/server-components (Accessed: 23 March 2025).

Vercel, 2024. Enterprise Service Level Agreement. Available at: https://vercel.com/legal/sla (Accessed: 29 April 2025).

Vercel, 2025. Routing – Next.js documentation. Available at: https://nextjs.org/docs/pages/building-your-application/routing (Accessed: 14 April 2025).

W3C, n.d. Developer tools. Available at: https://www.w3.org/developers/tools/ (Accessed: 23 March 2025).

W3Schools, n.d.a. CSS responsive – Media queries. Available at: https://www.w3schools.com/css/css_rwd_mediaqueries.asp (Accessed: 23 March 2025).

W3Schools, n.d.b. CSS units. Available at: https://www.w3schools.com/css/css_units.asp (Accessed: 23 March 2025).

Wiegers, K.E., 2000. 10 requirements traps to avoid. Originally published in Software Testing & Quality Engineering, Jan/Feb 2000. Reprinted with modifications by Process Impact. Available at: https://users.csc.calpoly.edu/~csturner/courses/300f06/readings/reqtraps.pdf (Accessed: 24 April 2025).

Wiegers, K.E. and Beatty, J., 2013. Software requirements. 3rd ed. Boston: Pearson Education. Available at: https://ptgmedia.pearsoncmg.com/images/9780735679665/samplepages/9780735679665.pdf (Accessed: 15 December 2024).

Appendix A: Record of Engagement

A.1 Meetings Log

This appendix documents the meetings held with the project supervisor throughout the CI601 project timeline. Table A.1 provides a record of both group and one-to-one meetings, outlining the date, time, type and a brief summary of each meeting.

Date	Time	Type	Details
1st November 2024	10:00-10:45	Group	Group meeting introducing CI601, project timeline, initial questions and discussion of interim report and project viva.
8th November 2024	09:30-09:50	1-2-1	Initial meeting discussing project idea, interim report and project viva.
15th November 2024	09:30-10:00	1-2-1	Discussing proposed technology choices and approach for the project. Feedback on initial plan for interim report.
11th December 2024	12:00-12:30	1-2-1	Project viva. Main piece of project report feedback was to improve the flow of the project aim into the objectives and into requirements.
10th February 2025	11:30-12:00	1-2-1	Feedback on report post viva feedback updates
20th February 2025	11:30-12:30	Group	Discussion of how to present the documentation of the full software development lifecycle.
27th February 2025	12:00-12:30	1-2-1	Showcase and discussion of current progress of web application. Talked about the way that the appointment dispute system could work.
13th March 2025	12:00-12:30	1-2-1	Showcase of completed MVP with additional features. Further discussion of what to include within the Product Description section of the report.
26th March 2025	10:00-10:30	1-2-1	Feedback on the Product Description section of the report. Going through the specific requirement alterations and evidence of test, fail, fix and repeat. Discussion of Testing and Research sections.

27th March 2025	11:30-12:30	Group	Going through the mark scheme to aid in our understanding.
10th April 2025	11:00-11:30	1-2-1	Questions, feedback and discussion of work in progress Methodology as well as finished Product Description and Research sections.
17th April 2025	10:00-10:30	1-2-1	Read through and feedback on the finished Methodology section.
23rd April 2025	13:00-13:30	1-2-1	Questions, feedback and discussion of updated order of sections, literature review, questionnaires and Abstract.
24th April 2025	11:30-12:30	Group	Final group Q&A session.

Table A.1: CI601 Meetings Log

A.2 Email Chains

This appendix details the email contact the project supervisor through the CI601 project. Where appropriate it also shows the email correspondence with the second reader.A.2.1 Supervisor Request.

A.2.1 Supervisor Request

This appendix presents the initial CI601 supervisor request email.

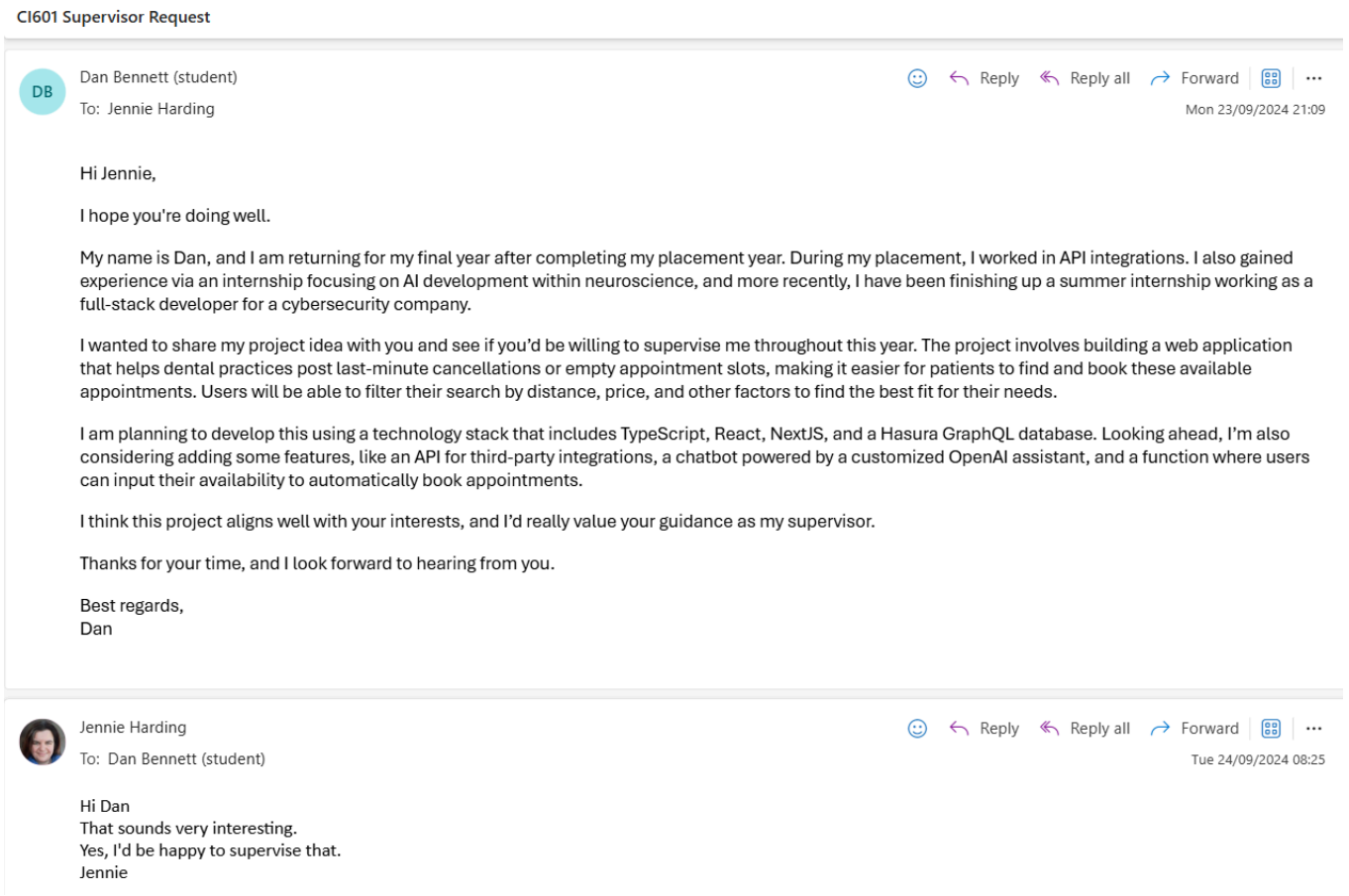
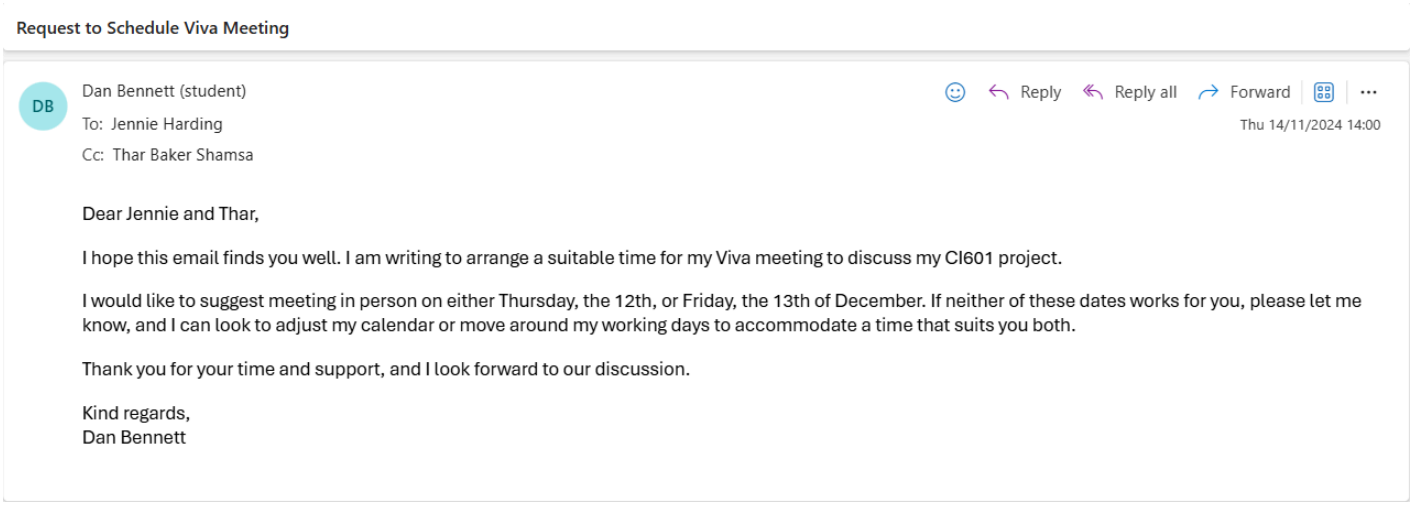


Figure A.1: CI601 supervisor request email.

A.2.2 Viva Organisation

This appendix details the organisation of the CI601 Viva, showing correspondence with the project supervisor and second reader.



Request to Schedule Viva Meeting

DB Dan Bennett (student)

To: Jennie Harding

Cc: Thar Baker Shamsa

Reply Reply all Forward

Thu 14/11/2024 14:00

Dear Jennie and Thar,

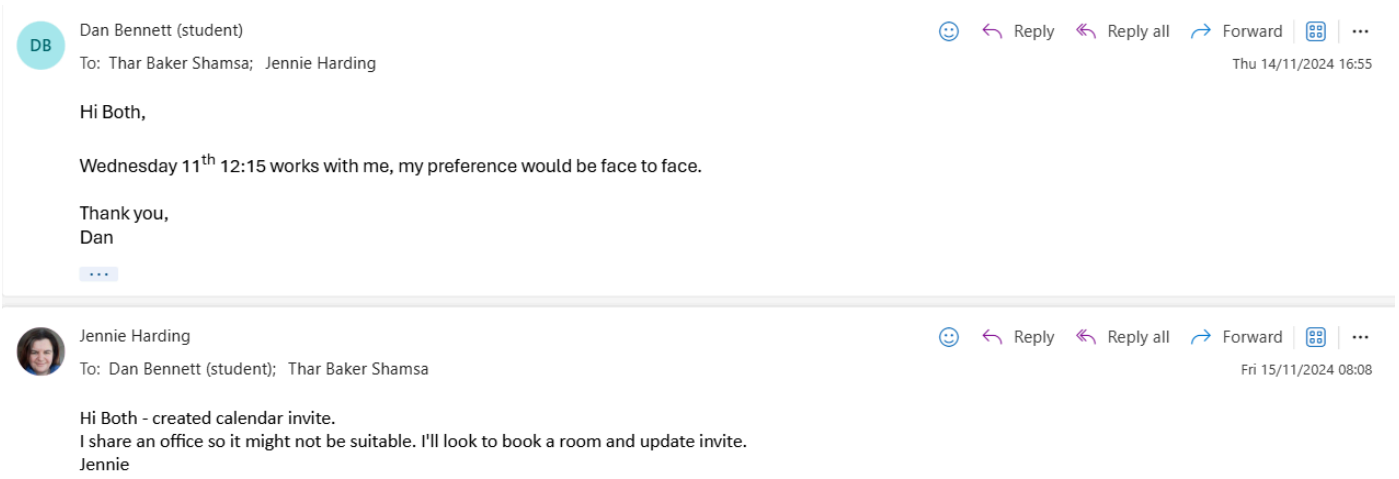
I hope this email finds you well. I am writing to arrange a suitable time for my Viva meeting to discuss my CI601 project.

I would like to suggest meeting in person on either Thursday, the 12th, or Friday, the 13th of December. If neither of these dates works for you, please let me know, and I can look to adjust my calendar or move around my working days to accommodate a time that suits you both.

Thank you for your time and support, and I look forward to our discussion.

Kind regards,
Dan Bennett

Figure A.2: CI601 initial Viva organisation email.



DB Dan Bennett (student)

To: Thar Baker Shamsa; Jennie Harding

Reply Reply all Forward

Thu 14/11/2024 16:55

Hi Both,

Wednesday 11th 12:15 works with me, my preference would be face to face.

Thank you,
Dan

...

Jennie Harding

To: Dan Bennett (student); Thar Baker Shamsa

Reply Reply all Forward

Fri 15/11/2024 08:08

Hi Both - created calendar invite.
I share an office so it might not be suitable. I'll look to book a room and update invite.
Jennie

Figure A.3: CI601 Viva confirmation email.

A.2.3 First Meeting Request of Semester 2

This appendix shows the email correspondence organising the first one-to-one meeting of Semester Two.

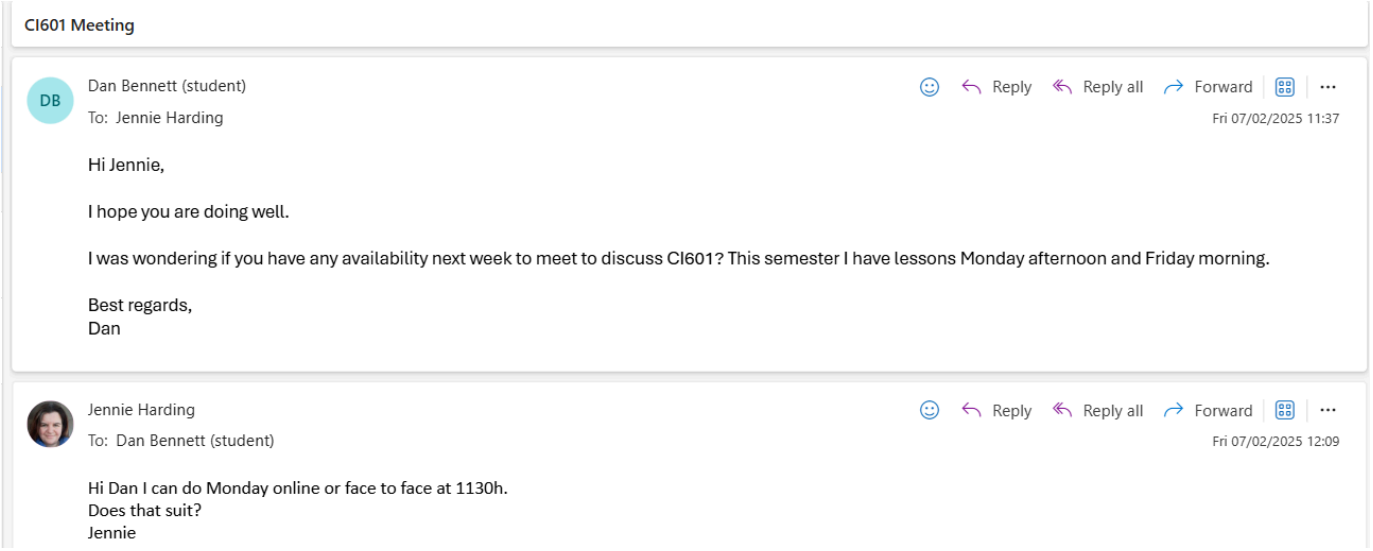


Figure A.4: Email arranging the first one-to-one meeting of Semester Two.

Appendix B: Ethics Checklist

The following checklist should be used to determine whether your project falls within the scope of the University's Research Ethics Policy, and whether ethical review will be required (see section 3 of the Research Ethics Policy for further information on the types of research that falls within its scope).

School of Architecture, Technology and Engineering	
Student Name	Daniel Bennett
Project Title	A Dentistry Appointment Booking Web Application

	Question	**Yes**	**No**
1	Will the project involve the participation of humans (e.g. interviews, surveys, focus groups, observations, photography, audio or video recording, physical activity or invasive/intrusive procedures)?	X	
2	If NO to 1, please explain what, if any, data you plan to collect and confirm that it is NOT from human subjects.		
3	Will the research involve the use of bodily materials derived or obtained from humans?		X
4	Will the research require access to, collection of or use of (non-personal) sensitive or confidential data?		X
5	Does the research have the potential to expose any person, whether or not participating in the research, to physical or psychological harm?		X
6	Does the research have the potential for significant negative impact on or damage to the natural environment?		X
7	Does the research have the potential for significant negative impact on culture or cultural heritage?		X
8	Will the research involve the use of or study of animals of any kind? (Note: animal research should be reviewed by the Animal Welfare and Ethical Review Board).		X
9	Are there are any other ethical issues raised by this research project that in the opinion of the applicant would warrant ethical review?		X

If you have answered 'yes' to any questions from 3 to 9, then an individual research ethics application though BREAM would be necessary. **This is not practical for an undergraduate project.** *You should discuss with your supervisor how data collection*

can be amended to fit under the umbrella of the module-wide research ethics application without compromising your project.

Start date (if recruiting participants)	1 October 2024
End date (if recruiting participants)	6 June 2025

I declare that all data collection will be covered by the module-wide research ethics application.

I confirm that I will NOT be collecting:

- Personal data on vulnerable adults
- Personal data on adults who are not CI601 students (this includes data on things such as names, addresses, photographs of the respondents etc)
- Personal data that could identify the respondents
- Personal data that is classified as sensitive

I understand that I will need to make an individual research ethics application though BREAM if my data collection goes beyond the umbrella of the CI601 research ethics application.

Signed:
Daniel Bennett

Appendix C: Project Plan

This appendix presents the full project plan, showing each of the nine two-week sprints. Each section details a sprint's focus, outcome and the corresponding Gantt chart. Figure C.1 outlines the key for the Gantt chart.

When a task was planned to be added to the next sprint it was marked as incomplete. When a task was de-scoped complete it was marked as backlogged.

Figure C.1: Gantt chart key

C.1: Sprint 0 - Project Setup

All tasks were completed apart from implementing GraphQL Codegen, which was added to the next sprint.

Activity	Status	Start	End	Effort (Days)
Sprint 0 - Project Setup		**05-12-24**	**18-12-24**	**14**
Initialise GitHub Repository	Complete	05-12-24	05-12-24	1
Set up React and Next.js project with TypeScript	Complete	06-12-24	06-12-24	1
Configure ESLint for consistent code style	Complete	07-12-24	07-12-24	1
Create .env file and define environment variables	Complete	08-12-24	08-12-24	1
Set up Docker environment for local PostgreSQL and Hasura engine	Complete	09-12-24	09-12-24	1
Initialise local Hasura project and connect to local PostgreSQL	Complete	10-12-24	10-12-24	1
Set up Hasura migrations and metadata tracking via Hasura CLI	Complete	11-12-24	11-12-24	0.5
Set up Hasura Cloud instance	Complete	11-12-24	11-12-24	0.5
Set up Neon Cloud PostgreSQL database	Complete	12-12-24	12-12-24	1
Connect Hasura Cloud to Neon PostgreSQL	Complete	13-12-24	13-12-24	1
Create and configure Vercel project and set up environment variables	Complete	14-12-24	14-12-24	0.5
Connect Hasura and Vercel to GitHub repository via their integrations	Complete	14-12-24	14-12-24	0.5
Create GitHub Actions CI workflow	Complete	15-12-24	15-12-24	1
Configure Hasura and Vercel deployments	Complete	16-12-24	16-12-24	0.5
Implement GraphQL Codegen	Incomplete	16-12-24	16-12-24	0.5
Contingency	N/A	17-12-24	18-12-24	2

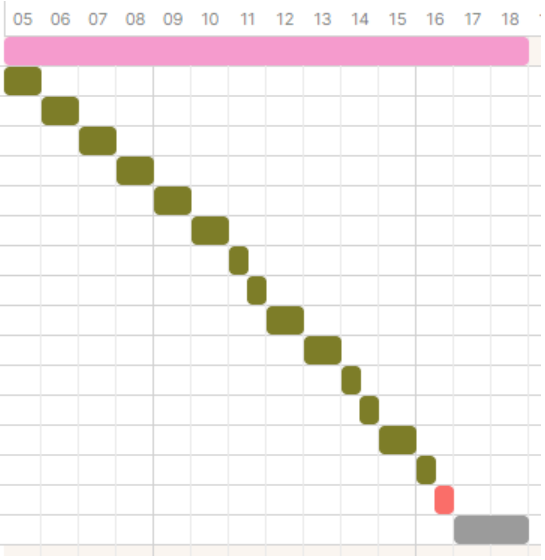


Figure C.2: Gantt Chart for Sprint 0 - Project Setup

C.2: Sprint 1 - User Login and Registration

All tasks were completed apart from implementing GraphQL Codegen, which was added to the next sprint.

Sprint 1 - User Login & Registration		04-01-25	17-01-25	14
Create users table in Hasura with fields and role permissions	Complete	04-01-25	04-01-25	0.5
Set up register API route and service methods	Complete	04-01-25	04-01-25	0.5
Implement password hashing with bcrypt in registration API	Complete	05-01-25	05-01-25	0.5
Create GraphQL mutation to insert new users	Complete	05-01-25	05-01-25	0.5
Build registration form and connect to API	Complete	06-01-25	06-01-25	0.5
Implement Toast notifications for registration flow	Complete	06-01-25	06-01-25	0.5
Set up login API route and service methods	Complete	07-01-25	07-01-25	0.5
Implement password check and JWT generation	Complete	07-01-25	08-01-25	1
Build login form and connect to API	Complete	08-01-25	09-01-25	1
Implement Toast notifications for login flow	Complete	09-01-25	09-01-25	0.5
Create useUser custom react hook	Complete	10-01-25	10-01-25	1
Set up profile API routes and service methods	Complete	11-01-25	11-01-25	1
Build user profile page and connect to API	Complete	12-01-25	12-01-25	0.5
Integrate third-party email provider (MailJet)	Complete	12-01-25	12-01-25	0.5
Create password_reset_tokens table in Hasura	Complete	13-01-25	13-01-25	0.5
Implement forgot/reset password API routes	Complete	13-01-25	13-01-25	0.5
Build forgot/reset password pages and connect APIs	Complete	14-01-25	14-01-25	1.5
Test all authentication and profile flows	Complete	15-01-25	15-01-25	0.5
Implement GraphQL Codegen	Incomplete	15-01-25	15-01-25	0.5
Contingency	N/A	16-01-25	17-01-25	2

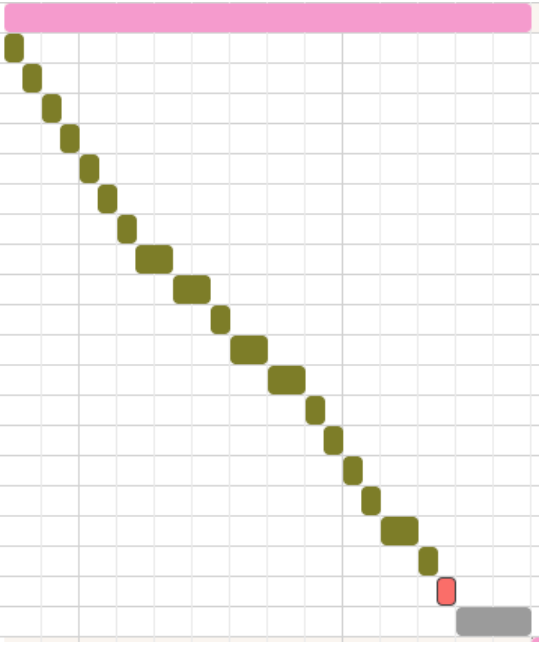


Figure C.3: Gantt Chart for Sprint 1 - User Login and Registration

C.3: Sprint 2 - Practice Registration and Verification

All tasks were completed apart from implementing GraphQL Codegen, which was added to the next sprint.

Sprint 2 - Practice Registration & Verficiation		18-01-25	31-01-25	14
Create practices table in Hasura	Complete	18-01-25	18-01-25	1
Set up Amazon S3 bucket for presigned URL image uploads	Complete	19-01-25	19-01-25	0.5
Implement practice registration route and service methods	Complete	19-01-25	19-01-25	0.5
Define Zod validation schema for practice registration	Complete	20-01-25	20-01-25	0.5
Build practice registration form and connect to API	Complete	20-01-25	20-01-25	0.5
Implement Toast notifications for practice registration flow	Complete	21-01-25	21-01-25	1
Set up practice login route and service methods (reuse user login)	Complete	22-01-25	22-01-25	0.5
Build practice login form and connect to API	Complete	22-01-25	22-01-25	0.5
Implement usePractice custom React hook	Complete	23-01-25	23-01-25	1
Implement practice verification routes and service methods	Complete	24-01-25	24-01-25	1
Build admin dashboard	Complete	25-01-25	25-01-25	1
Implement practice verification within admin dashboard	Complete	26-01-25	26-01-25	0.5
Implement Toast notifications for admin actions	Complete	26-01-25	26-01-25	0.5
Set up practice profile API routes and service methods	Complete	27-01-25	27-01-25	1
Build practice profile management page and connect to API	Complete	28-01-25	28-01-25	1
Test registration, login, profile, verification, and image upload	Complete	29-01-25	29-01-25	0.5
Implement GraphQL Codegen	Incomplete	29-01-25	29-01-25	0.5
Contingency	N/A	30-01-25	31-01-25	2

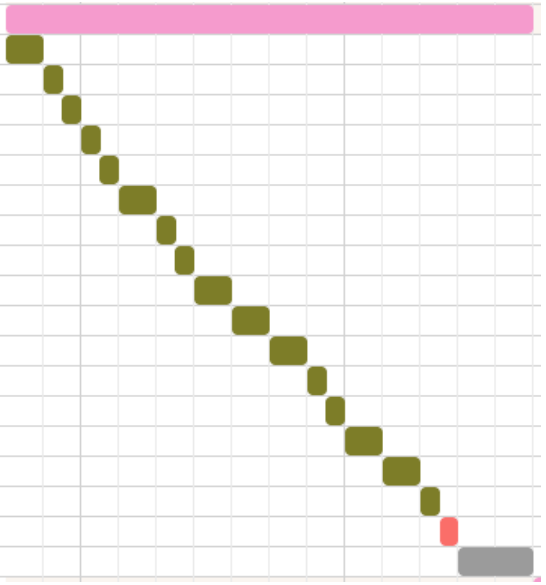


Figure C.4: Gantt Chart for Sprint 2 - Practice Registration and Verification

C.4: Sprint 3 - Appointment Creation and Management

All tasks were completed apart from implementing GraphQL Codegen, which was added to the next sprint.

Sprint 3 - Appointment Creation & Management		01-02-25	14-02-25	14
Create appointments table in Hasura	Complete	01-02-25	01-02-25	1
Define appointment types and durations	Complete	02-02-25	02-02-25	1
Expand practice signup form to support appointment types.	Complete	03-02-25	03-02-25	1
Set up appointment API routes and service methods (create, edit, delete)	Complete	04-02-25	04-02-25	1
Define Zod validation schema for appointment creation & editing	Complete	05-02-25	05-02-25	1
Build Manage Appointments page and connect to AP	Complete	06-02-25	06-02-25	1
Build calendar view within practice dashboard	Complete	07-02-25	07-02-25	1
Build create appointment popup and connect to API	Complete	08-02-25	08-02-25	1
Build edit appointment popup and connect to API	Complete	09-02-25	09-02-25	1
Expand edit popup to include delete appointment functionality	Complete	10-02-25	10-02-25	1
Implement Toast notifications for appointment actions	Complete	11-02-25	11-02-25	1
Test appointment creation, editing, and deletion flows	Complete	12-02-25	12-02-25	0.5
Implement GraphQL Codegen	Incomplete	12-02-25	12-02-25	0.5
Contingency	N/A	13-02-25	14-02-25	2

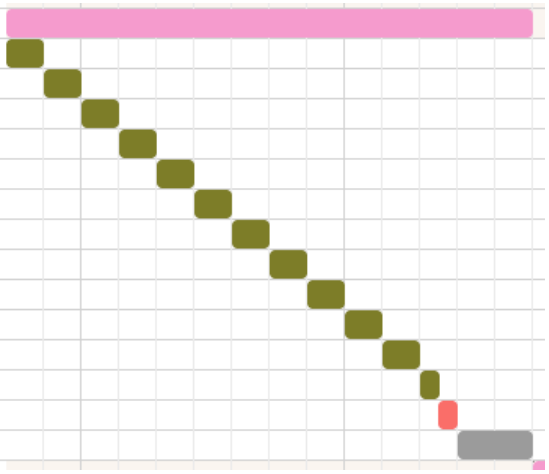


Figure C.5: Gantt Chart for Sprint 3 - Appointment Creation and Management

C.5: Sprint 4 - Appointment Listing

Most of the tasks were completed. Each incomplete task was prioritised with de-descoped completely and marked as backlogged, with those marked as incomplete being added to the next sprint.

Sprint 4 - Appointment Listing		15-02-25	28-02-25	14
Enable PostGIS extension in Hasura database	Complete	15-02-25	15-02-25	0.5
Integrate Google Geocoding API for distance calculations	Complete	15-02-25	15-02-25	0.5
Create appointment search view & function in Hasura	Complete	16-02-25	16-02-25	1
Set up appointment listing API route and service methods	Complete	17-02-25	17-02-25	1
Implement custom React hook for automatic location detection	Complete	18-02-25	18-02-25	1
Build appointment search page with appointment cards	Complete	19-02-25	20-02-25	2
Build filter component	Complete	21-02-25	21-02-25	1
Implement sorting options	Complete	22-02-25	22-02-25	1
Set up notification API route and service for new appointments	Backlogged	23-02-25	23-02-25	1
Build user notification preferences UI	Backlogged	24-02-25	25-02-25	2
Integrate SMS provider for sending notifications	Incomplete	26-02-25	26-02-25	0.5
Implement GraphQL Codegen	Incomplete	26-02-25	26-02-25	0.5
Contingency	N/A	27-02-25	28-02-25	2

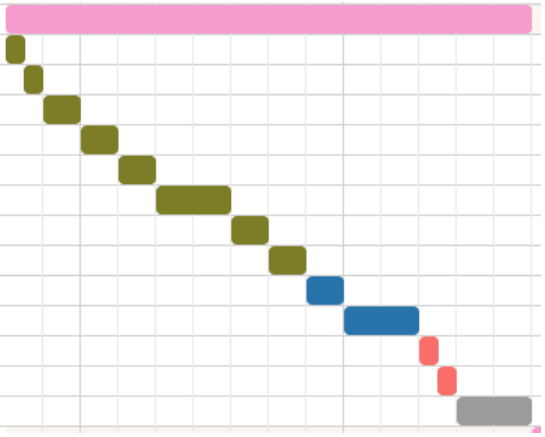


Figure C.6: Gantt Chart for Sprint 4 - Appointment Listing

C.6: Sprint 5 - Booking System

All tasks were completed apart from implementing GraphQL Codegen and integrating with an SMS provider, which was added to the next sprint..

Sprint 5 - Booking System		01-03-25	14-03-25	14
Expand appointments table for booking status and service	Complete	01-03-25	01-03-25	1
Set up booking API route and service methods	Complete	02-03-25	02-03-25	1
Build booking pages and components for flow	Complete	03-03-25	05-03-25	3
Build booking confirmation logic (check availability, prevent double-booking)	Complete	06-03-25	06-03-25	1
Build authentication check for booking flow	Complete	07-03-25	07-03-25	1
Reuse login/register components for unauthenticated booking attempts	Complete	08-03-25	08-03-25	1
Generate booking confirmation message (on-screen & email)	Complete	09-03-25	09-03-25	0.5
Integrate email provider for booking confirmation emails	Complete	09-03-25	09-03-25	0.5
Build user booking history page	Complete	10-03-25	10-03-25	1
Set up API route and service for fetching user bookings	Complete	11-03-25	11-03-25	1
Implement GraphQL Codegen	Incomplete	12-03-25	12-03-25	0.5
Integrate SMS provider for sending notifications	Incomplete	12-03-25	12-03-25	0.5
Contingency	N/A	13-03-25	14-03-25	2

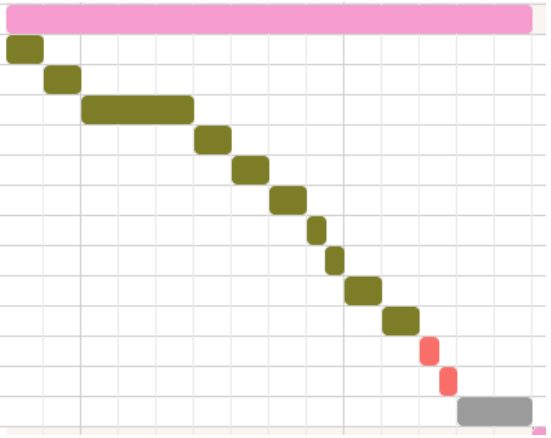


Figure C.7: Gantt Chart for Sprint 5 - Booking System

C.7: Sprint 6 - Addition Practice Tools

All tasks were completed apart from implementing GraphQL Codegen and integrating with an SMS provider. Both of these tasks were backlogged due to the technical overhead that would be required to retroactively implement them into the web application.

Sprint 6 - Additional Practice Tools		15-03-25	28-03-25	14
Create practice_reviews table	Complete	15-03-25	15-03-25	0.5
Build review and feedback form in My Appointments UI	Complete	15-03-25	16-03-25	1.5
Set up routes and services for submitting reviews and fetching aggregate ratings	Complete	17-03-25	17-03-25	0.5
Update major pages to display aggregate ratings (e.g., search, dashboard)	Complete	17-03-25	19-03-25	2.5
Create aggregate endpoints for number of appointments booked	Complete	20-03-25	20-03-25	0.5
Integrate graphing library for basic analytics	Complete	20-03-25	21-03-25	1.5
Build analytics section in practice dashboard	Complete	22-03-25	22-03-25	1
Create practice_invites table in Hasura	Complete	23-03-25	23-03-25	0.5
Implement token generation logic for practice invitations	Complete	23-03-25	23-03-25	0.5
Set up routes and services for handling invites	Complete	24-03-25	24-03-25	1
Build join practice pages and UI components	Complete	25-03-25	25-03-25	1
Implement GraphQL Codegen	Backlogged	26-03-25	26-03-25	0.5
Integrate SMS provider for sending notifications	Backlogged	26-03-25	26-03-25	0.5
Contingency	N/A	27-03-25	28-03-25	2

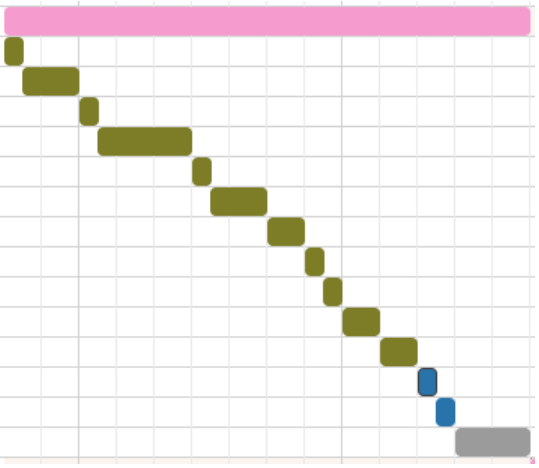


Figure C.8: Gantt Chart for Sprint 6 - Additional Practice Tools

C.8: Sprint 7 - Additional User Tools

All incomplete tasks were immediately de-scoped due to time and resource restrictions imposed by other University assignments.

Sprint 7 - Additional User Tools		29-03-25	11-04-25	14
Add dispute option to My Appointments UI	Complete	29-03-25	29-03-25	1
Implement logic to restrict disputes to completed appointments within 5 days	Complete	30-03-25	30-03-25	0.5
Create appointment_disputes table in Hasura	Incomplete	30-03-25	30-03-25	0.5
Set up routes and services for submitting disputes	Incomplete	31-03-25	31-03-25	1
Implement Toast notifications for dispute submission	Incomplete	01-04-25	01-04-25	0.5
Create dispute dashboard UI for admin	Backlogged	01-04-25	02-04-25	1.5
Set up routes and services for fetching and managing disputes	Backlogged	03-04-25	03-04-25	1
Implement basic dispute management actions (view, mark as resolved)	Backlogged	04-04-25	07-04-25	4
Update navigation logic to link to application status page	Complete	08-04-25	08-04-25	0.5
Set up service to fetch practice application status	Complete	08-04-25	08-04-25	0.5
Build application status page for practices	Incomplete	09-04-25	09-04-25	1
Contingency	N/A	10-04-25	11-04-25	2

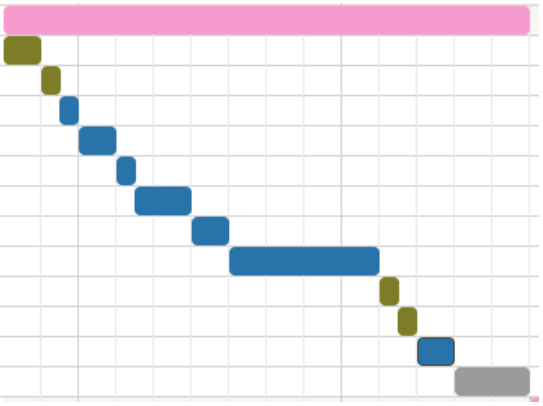


Figure C.9: Gantt Chart for Sprint 7 - Additional User Tools

C.9: Sprint 8 - Application Enhancements

All tasks were de-scoped due to time and resource restrictions imposed by other University assignments.

Sprint 8 - Application Enhancements		12-04-25	25-04-25	14
FR21 Map View	Backlogged	12-04-25	13-04-25	2
FR22 Calendar Integration	Backlogged	14-04-25	15-04-25	2
FR23 Chatbot Support	Backlogged	16-04-25	17-04-25	2
FR24 Public API	Backlogged	18-04-25	19-04-25	2
FR25 API Key Generation	Backlogged	20-04-25	21-04-25	2
FR26 Third-Part Authentication	Backlogged	22-04-25	23-04-25	2
Contingency	N/A	24-04-25	25-04-25	2

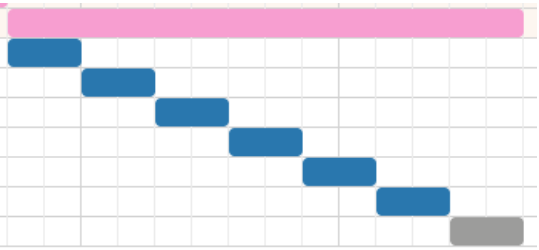


Figure C.10: Gantt Chart for Sprint 8 - Application Enhancements

Appendix D: Wireframes

This appendix details the user interface wireframes for DentalConnect, directly informed through the design questionnaire (Section 6.4.2 and Appendix K.4) and the competitor analysis (Section 6.3 and Appendix K.2).

D.1 User Sign In

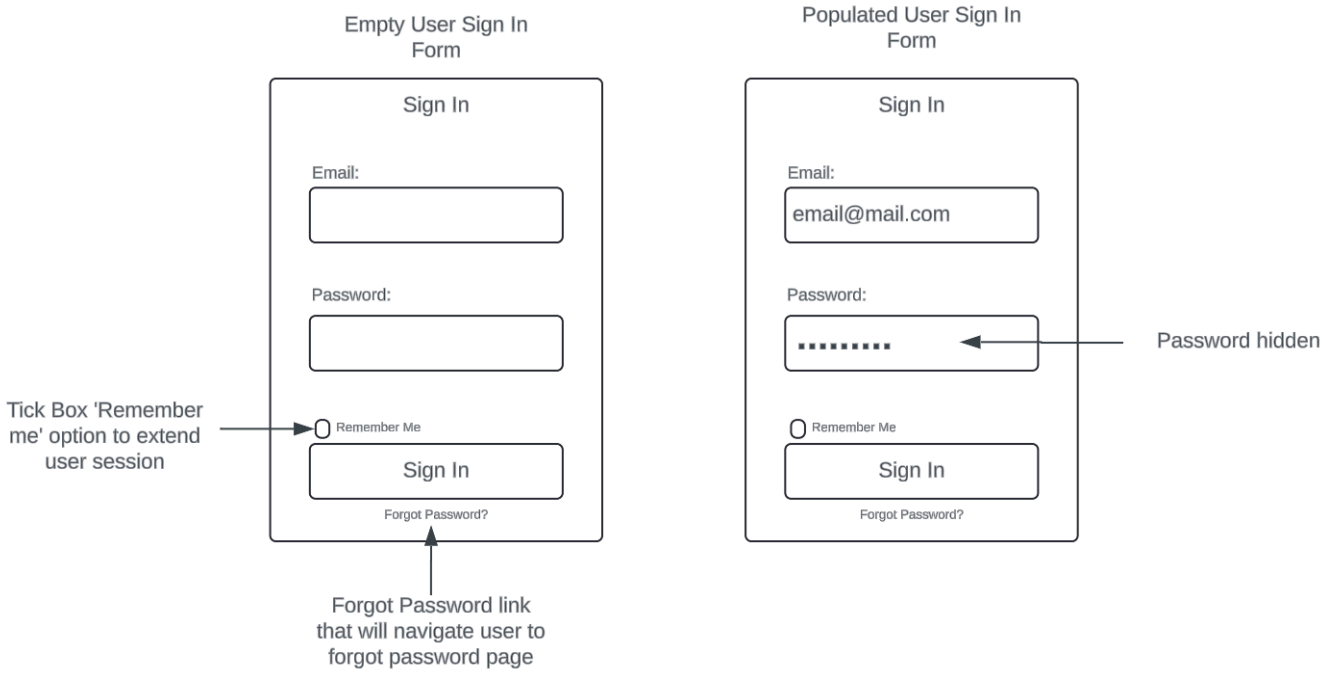


Figure D.1: User Sign components, empty form and populated form.

D.2 User Registration

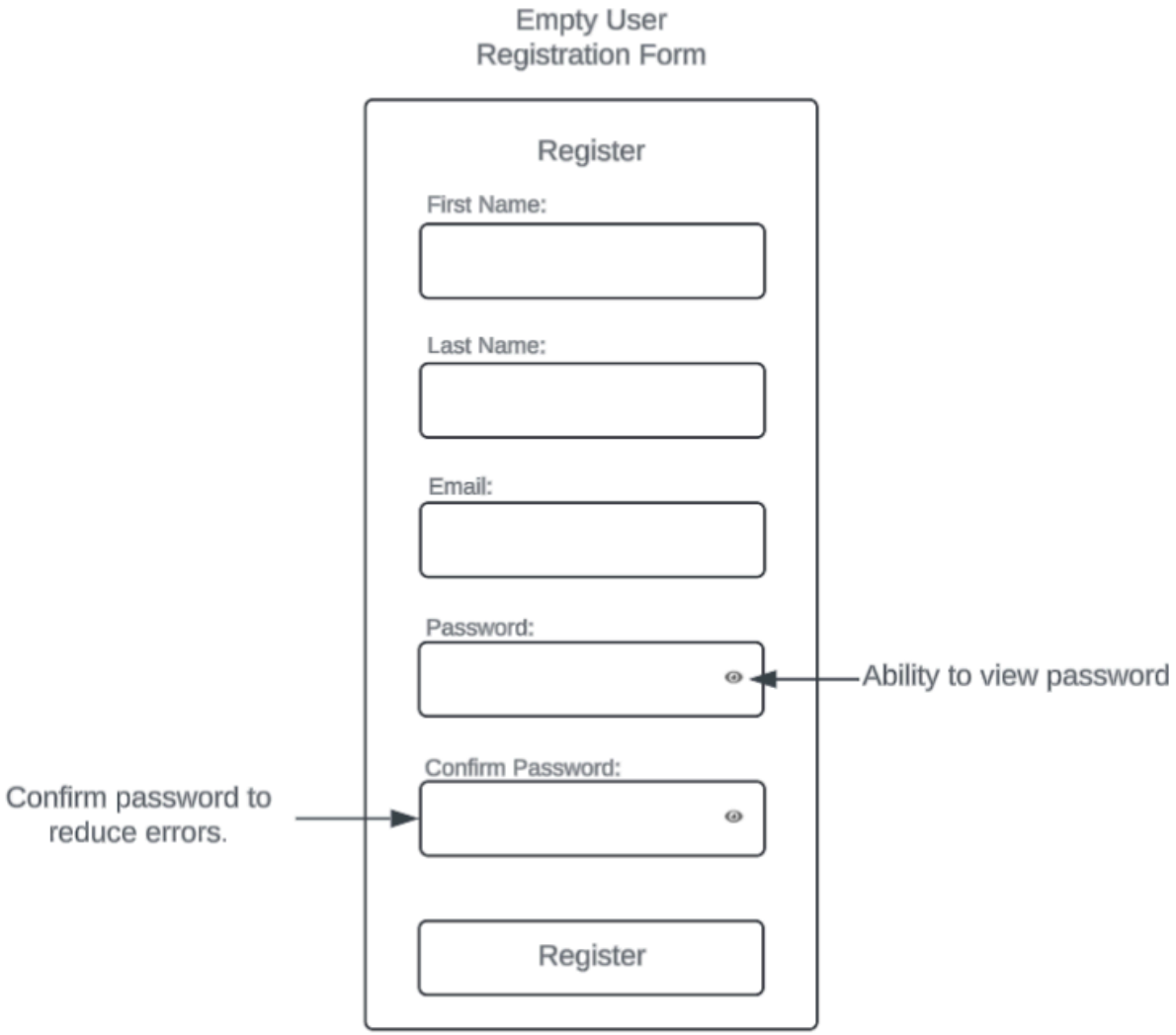


Figure D.2: User registration component with confirm password and button enabling a user to view the entered password.

D.3 Practice Registration

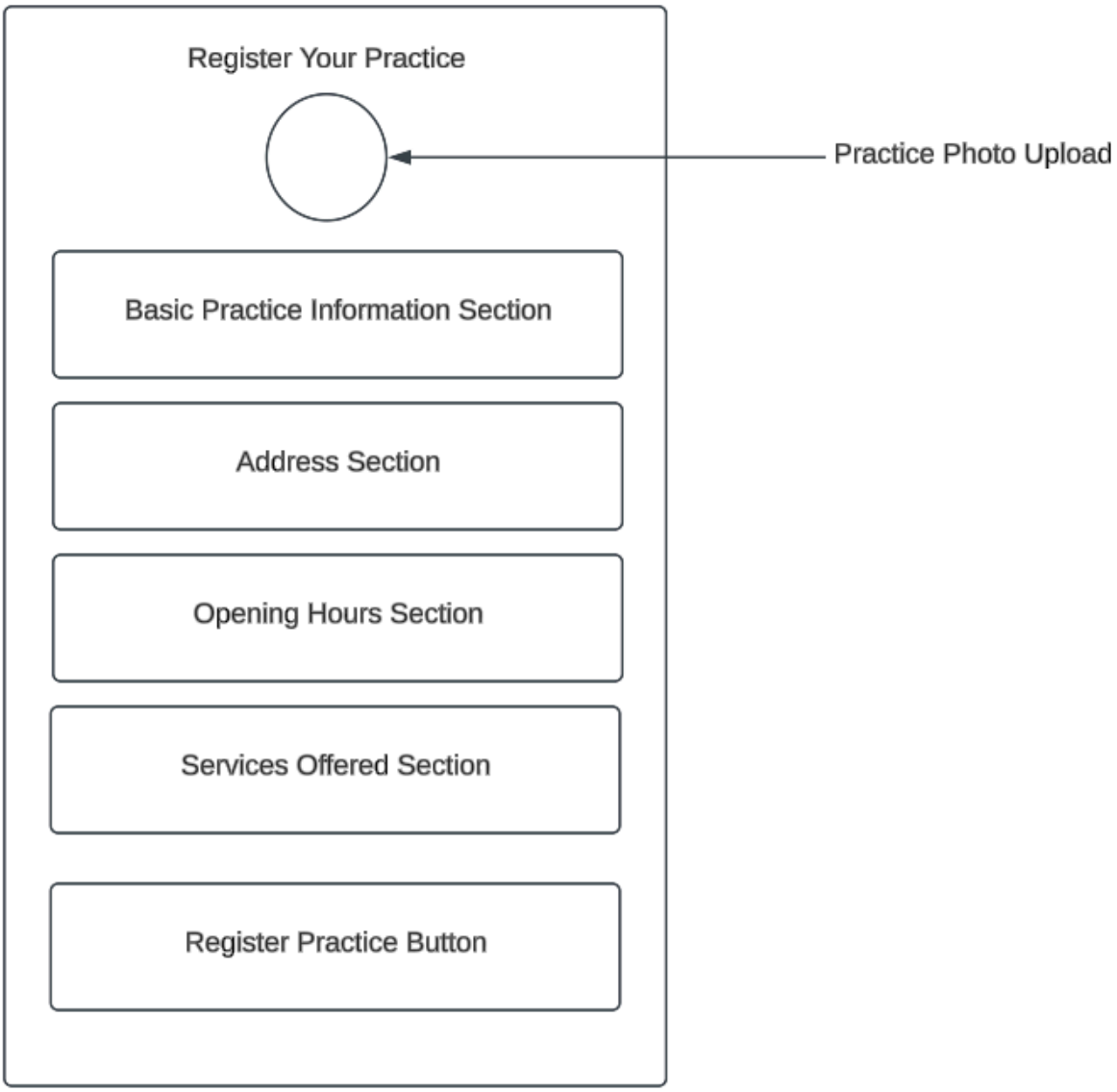


Figure D.3: Practice registration component overview.

D.4 Sign-In Page

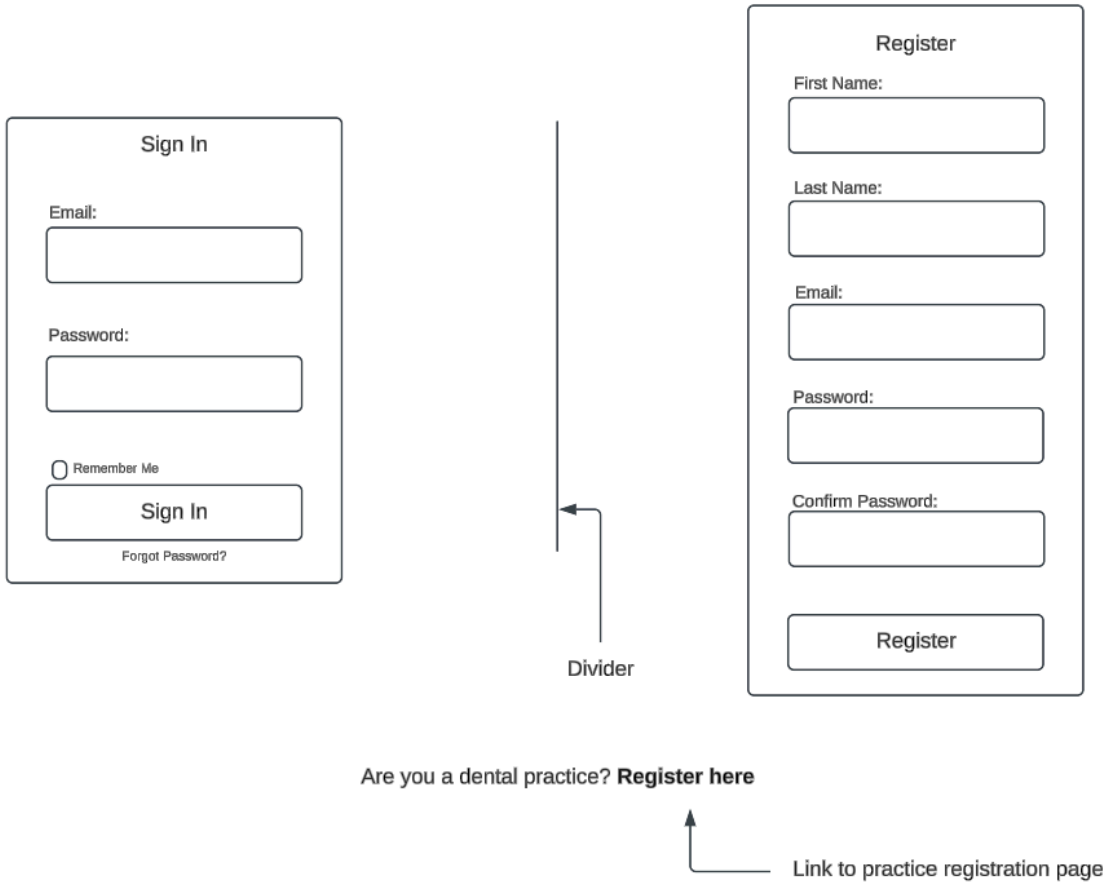


Figure D.4: DentalConnect Sign In Page (Desktop)

D.5 Homepage

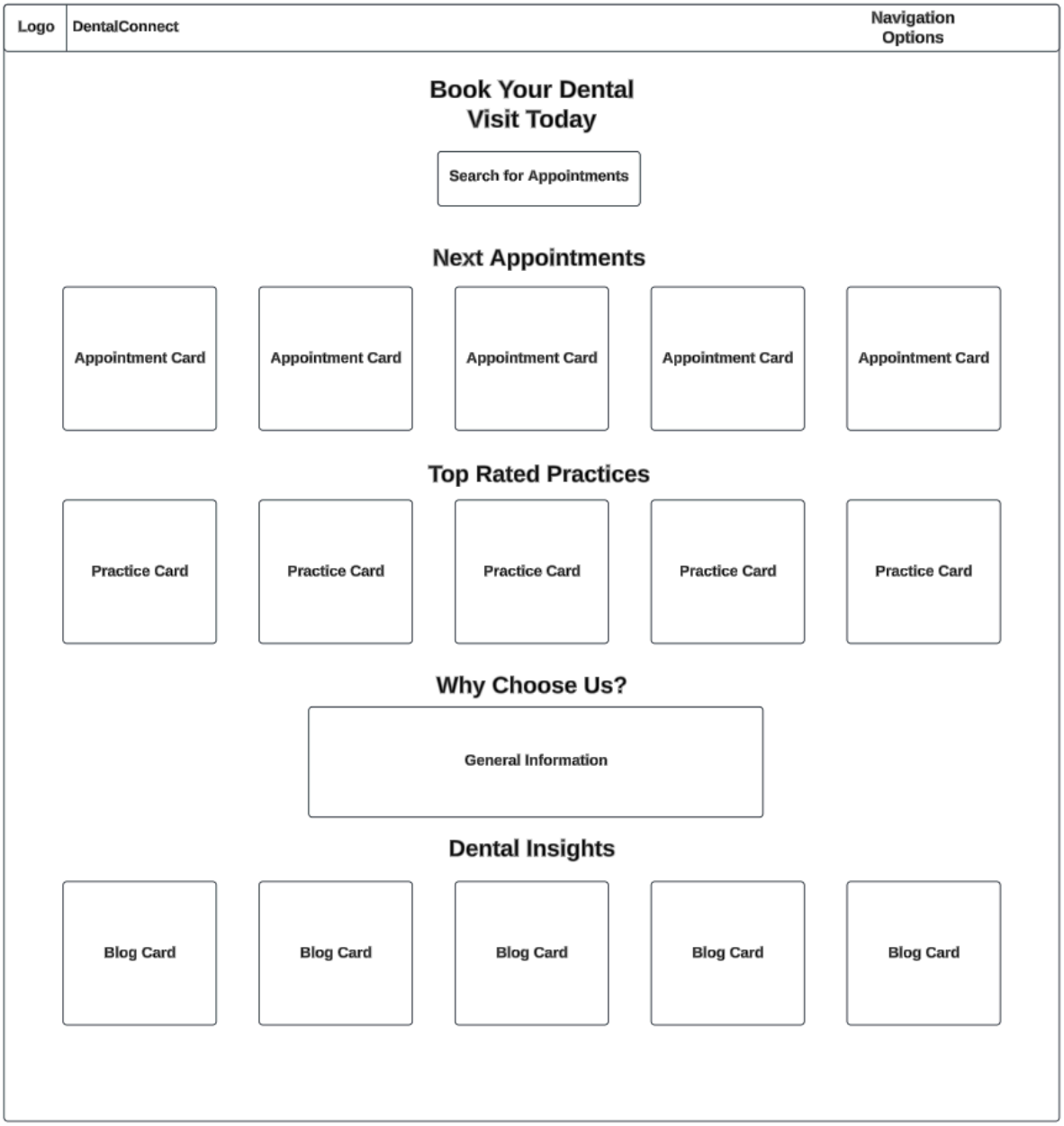


Figure D.5: DentalConnect home page (Desktop)

D.6 Search Page

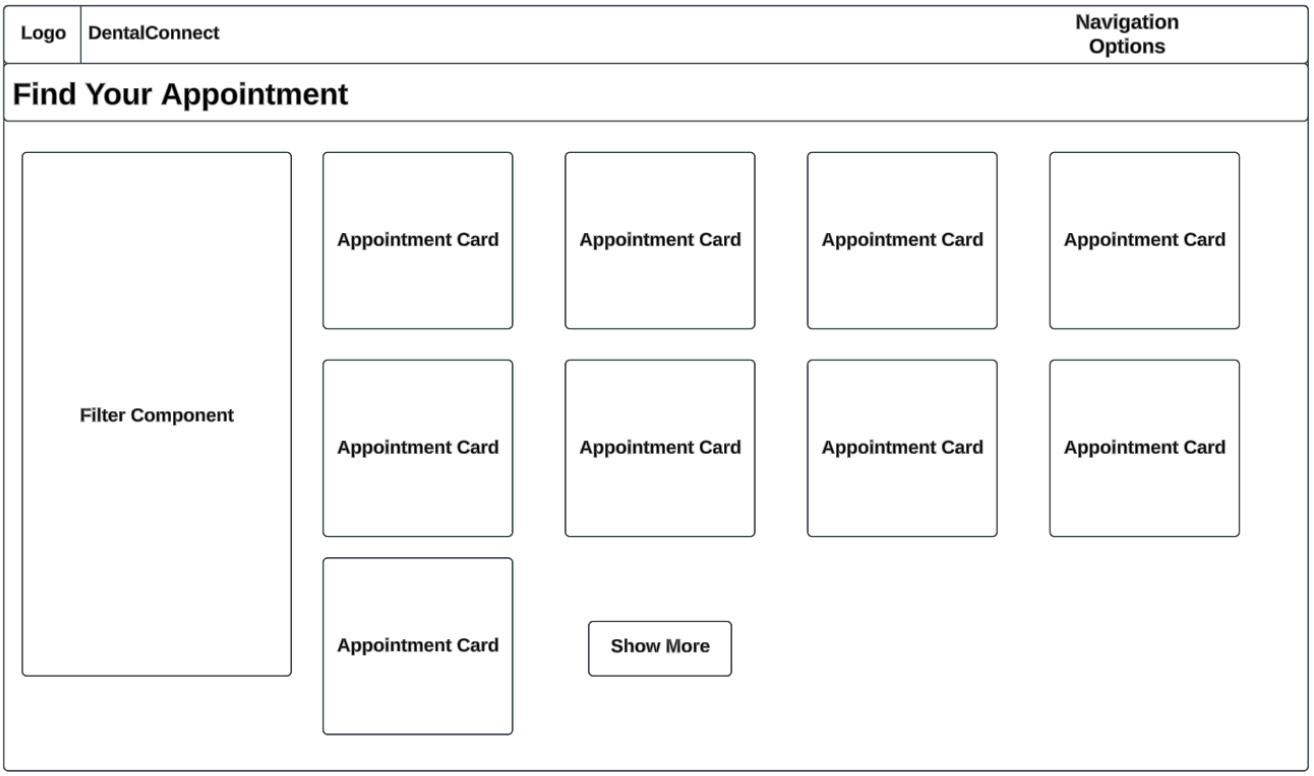


Figure D.6: DentalConnect search page (Desktop)

D.7 Appointment Management

Figure D.7: DentalConnect Appointment Management page within Practice Dashboard

D.8 Navigation Bar

Figure D.8: DentalConnect navigation bar.

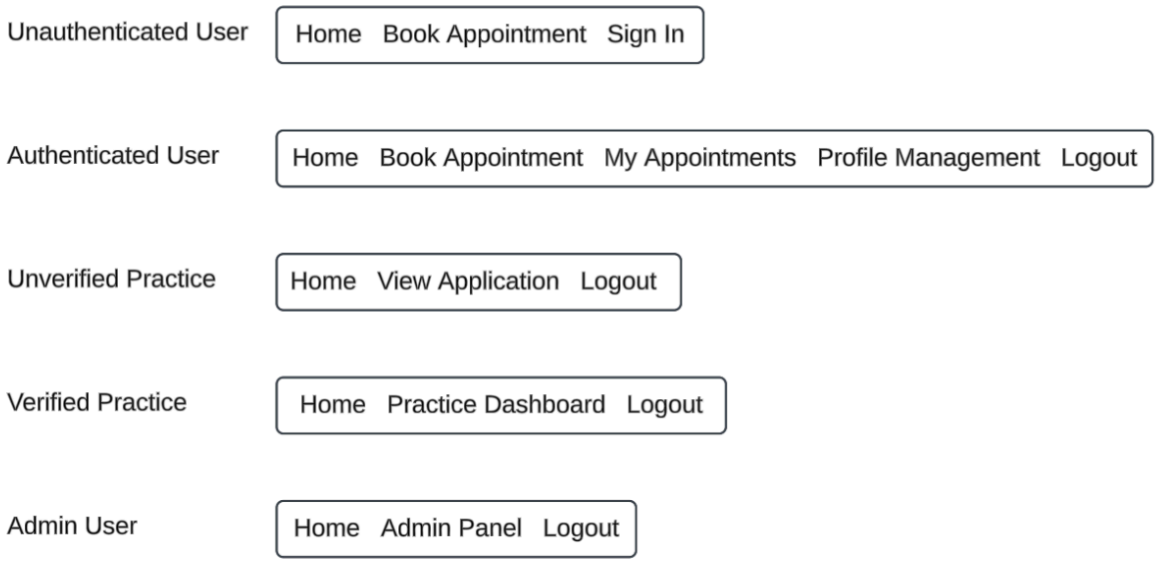


Figure D.9: DentalConnect navigation bar button options, based on authentication status and per role.

Appendix E: Technical Documentation

The local development setup instructions are available within the GitHub repository's ReadMe, available at:

https://github.com/danbennett239/CI601

Further project documentation of API routes and services can be accessed at:

https://github.com/danbennett239/CI601/tree/main/Documentation

Appendix F: Implemented Requirements

This appendix details the implementation status of the function and non-functional requirements of the project.

F.1: Functional Requirements

Table F.1 shows functional requirement implementation status. Green represents implemented, red represents unimplemented.

Requirement ID	Description	MoSCoW Priority	Linked Objective(s)
FR1	**Booking System**: Enable users to book appointments and receive confirmations.	Must-Have.	O1
FR2	**User Registration**: Allow users to create an account.	Must-Have.	O2
FR3	**User Login**: Enable users to log in securely to access and manage their accounts	Must-Have.	O2
FR4	**Practice Registration**: Allow dental practices to sign up securely.	Must-Have.	O2
FR5	**Practice Login**: Enable dental practices to log in securely to access and manage their accounts.	Must-Have.	O2
FR6	**Appointment Search**: Display available appointments with filters (distance, date, price) and sort options.	Must-Have.	O3
FR7	**Appointment Management**: Practices can create and manage appointments with dynamic durations, supporting the selection of one or more appointment types: Check-Up, Cleaning, Whitening, Filling, Extraction and Emergency.	Must-Have.	O1, O3

FR8	**Admin Account**: Admin users can log in and access administration features	Must-Have.	O2
FR9	**Practice Verification**: Admin can verify registered practices.	Must-Have.	O2
FR10	**User Profile Management**: Allow users to manage their details and preferences.	Must-Have.	O2
FR11	**Practice Profile Management**: Allow practices to manage their details and preferences.	Must-Have.	O2
FR12	**Notifications for New Appointments**: Notify users via email or text when appointments become available.	Should-Have.	O3
FR13	**Review and Feedback System**: Enable users to leave reviews and feedback about their experiences.	Should-Have.	O1
FR14	**Basic Analytics**: Provide dental practices with simple metrics to overview their performance.	Should-Have.	O2
FR15	**Automatic Location Detection**: Automatically detect and apply the user's location for appointment search.	Should-Have.	O3
FR16	**Email Integration**: Integration with a third-party email provider for "Forgot password" and email verification.	Should-Have.	O2
FR17	**Practices Invitations**: Practices can invite additional users to their practice.	Should-Have.	O2
FR18	**Dispute Dashboard**: Admin can view disputes raised by users.	Should-Have.	O2
FR19	**Appointment Disputes**: Users can raise disputes related to appointments.	Should-Have.	O1
FR20	**Application Status**: Practices can view the status of their application.	Should-Have.	O2

FR21	**Map View**: Display available appointments on an interactive map for easier navigation.	Could-Have.	O3
FR22	**Calendar Integration**: Allow users to add bookings to third-party calendars.	Could-Have.	N/A
FR23	**Chatbot Support**: Provide basic assistance for common questions or issues.	Could-Have.	N/A
FR24	**Public API**: Provide and document an API that allows third-party applications to access appointment data.	Could-Have.	N/A
FR25	**API Key Generation**: Enable practices to generate an API key securely.	Could-Have.	N/A
FR26	**Third-Party Authentication**: Integration with third-party authentication providers (e.g., Google).	Could-Have.	O1
FR27	**Advanced AI Chatbot**: A chatbot with image upload and symptom analysis for appointment suggestions.	Won't-Have.	N/A
FR28	**Advanced Analytics**: In-depth metrics and reports for analysing booking trends.	Won't-Have.	O2
FR29	**Custom Appointment Types**: Enable practices to define their own appointment types.	Won't-Have.	O1
FR30	**Payment System**: Integrate with a third-party payment provider to securely process and distribute payments to practices. The system holds funds temporarily to facilitate the dispute process.	Won't-Have	O1

Table F.1: Functional Requirement Implementation Status. Green indicates implemented requirements. Red indicated unimplemented requirements.

F.2: Non-Functional Requirements

Table F.2 shows functional requirement implementation status. Green represents implemented, red represents unimplemented.

Requirement ID	Description	MoSCoW Priority	Linked Objective(s)
NFR1	**Accessible and Responsive Design**: Ensure the website is fully accessible and user-friendly on both desktop and mobile devices.	Must-Have.	O4, O5
NFR2	**CI/CD Pipeline**: Set up an automated pipeline for testing, building, and deploying updates.	Must-Have.	O6
NFR3	**Performance**: The system must respond to user actions (e.g., booking, filtering) within 2 seconds.	Must-Have	O4, O5
NFR4	**Security (Password Encryption)**: All user and practice passwords must be stored securely using encryption (e.g., bcrypt).	Must-Have.	O2
NFR5	**Developer Alerting**: The system should notify developers via email of failed deployments or test runs.	Should-Have.	O6
NFR6	**Availability**: The system must maintain 99.9% uptime, with justification provided in Appendix H.2.	Should-Have.	O6

Table F.2: Functional Requirement Implementation Status. Green indicates implemented requirements. Red indicated unimplemented requirements.

Appendix G: Testing Documentation

G.1 Functional Requirement Test Case Descriptions

Table G.1 details step-by-step test cases that verify the system meets its functional requirements. These test cases are implemented as end-to-end (E2E) tests and are stored within the `/cypress` directory of the project.

Test ID	Requirement ID	Test Description	Test Type	File Name	Steps
F1	FR1	Verify that a user can book an appointment end-to-end and receive an on-screen confirmation.	E2E	bookingFlow	1. Start as an unauthenticated user. 2. Visit /home and click the "Search Appointments" link. 3. Wait for appointments to load on the /search page. 4. Click the first available appointment card. 5. On the appointment details page, click "Proceed to Checkout". 6. When redirected to the booking page (/appointments/:id/book), confirm that a login form is displayed. 7. Submit valid login credentials. 8. Once authenticated, confirm that the booking page reloads. 9. Click "Confirm Booking". 10. Assert that the "Booking Confirmed" message is shown.

F2	FR2	Verify that the user registration form handles mismatched passwords and successful registration properly.	E2E	userRegister	1. Visit the sign-in page (/signin). 2. Fill in the registration form with mismatched passwords. 3. Click the "Register" button. 4. Confirm that the error message “Passwords do not match” is shown. 5. Correct the repeated password to match. 6. Click the "Register" button again. 7. Confirm that the user is redirected to the homepage (/home).
F3	FR3	Verify that a registered user can log in with correct credentials.	E2E	userLogin	1. Visit the sign-in page (/signin). 2. Enter a valid email and incorrect password. 3. Click the "Sign In" button. 4. Confirm that an error message ("Invalid credentials") is displayed and that the URL remains /signin. 5. Enter the correct email and password: 6. Click the "Sign In" button again. 7. Confirm that the user is redirected to /home and a session cookie is set.
F4	FR4	Verify that a dental practice can complete the registration process.	E2E	practiceRegister	1. Visit /signin. 2. Click the "Register here" link to navigate to the practice registration page. 3. Fill out the practice registration form. 4. Test again with mismatched passwords 5. Assert that a "Passwords do not match" error is displayed and no redirect occurs 6. Update with matching passwords.

					7. Assert that the user is redirected to /home after successful registration.
F5	FR5	Verify that a verified practice can log in and access the dashboard.	E2E	verifiedPracticeLogin	1. Visit /signin. 2. Fill in the login form with valid verified practice credentials. 3. Click the "Sign In" button. 4. Assert that the user is redirected to /home. 5. Assert that the "Practice Dashboard" banner is visible. 6. Assert that the session cookie is set.
F6	FR5	Verify that an unverified practice can log in and see correct navigation options.	E2E	unverifiedPracticeLogin	1. Visit /signin. 2. Fill in the login form with valid unverified practice credentials. 3. Click the "Sign In" button. 4. Assert that the user is redirected to /home. 5. Assert that the "View Application" banner is visible. 6. Assert that the session cookie is set.
F7	FR6	Verify that search filters correctly filter and display appointment results.	E2E	searchAppointment	1. Navigate to /home and click the "Search Appointments" link to go to the search page. 2. Verify that 3 appointment cards are initially displayed. 3. Enter filters. 4. Click the "Apply Filters" button. 5. Verify that the filtered results show only 1 appointment card.

					6. Assert that the appointment shown is for "Smile Clinic".
F8	FR7	Verify that practices can create appointments with multiple appointment types and view them in the appointment calendar.	E2E	createAppointment	1. Log in as a verified practice and land on /home. 2. Click the "Practice Dashboard" banner and navigate to the appointments page. 3. Click the "Create Appointment" button to open the popup form. 4. Fill in the start and end time within valid opening hours. 5. Uncheck the “All” appointment types option. 6. Check the "Cleaning" and "Checkup" appointment types. 7. Click the "Create Appointment" button inside the popup. 8. Assert that the appointment popup closes successfully, indicating creation.
F9	FR8	Verify that an admin user can log in.		adminLogin	1. Visit /signin. 2. Fill in the admin login credentials 3. Submit the form 4. Assert that the user is redirected to /home. 5. Confirm that the “Admin Panel” banner is visible. 6. Assert that a session cookie is set
F10	FR8	Verify that an admin user can log in and access the admin dashboard.	E2E	adminPanelAccess	1. Visit the sign-in page. 2. Enter valid admin credentials and submit the form. 3. Assert redirect to /home and visibility of the "Admin Panel" link.

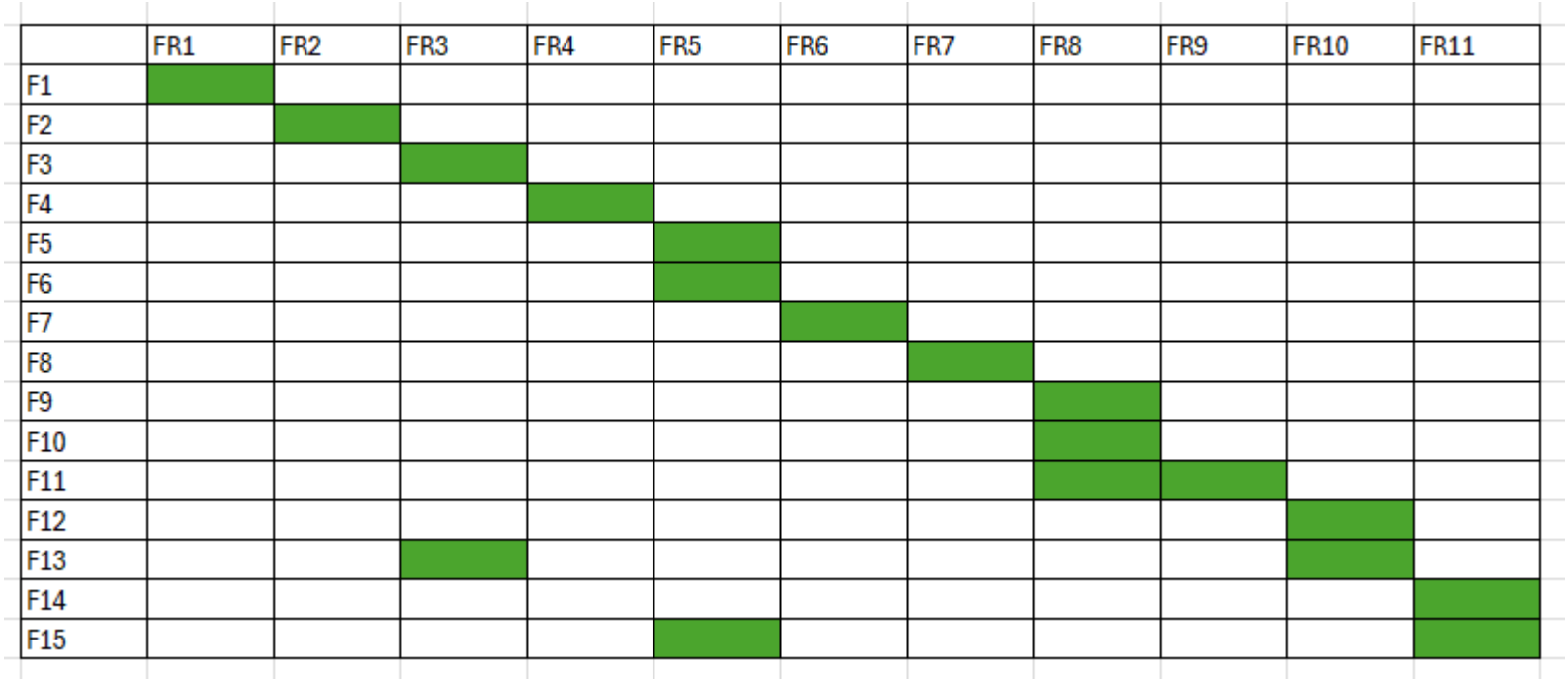
					4. Click "Admin Panel" and confirm redirect to /admin-panel. 5. Assert that the admin dashboard content is visible.
F11	FR8, FR9	Verify that an admin can login, view unverified practices and mark them as verified.	E2E	adminApprovePractice	1. Log in with valid admin credentials via the sign-in page. 2. Navigate to the Admin Panel. 3. Open the “Dental Practice Applications” section. 4. Confirm a pending practice is listed. 5. Click the practice to open the details modal. 6. Click the "Approve" button. 7. Assert that the approval is successful and the practice is removed from the list.
F12	FR10	Verify that users can view and update their profile.	E2E	updateUserSettings	1. Navigate to the profile page as a logged-in user. 2. Clear the existing email field and enter a new valid email address. 3. Click the save button when enabled. 4. Confirm that a success message is displayed. 5. Verify that the input field reflects the updated email value.
F13	FR3, FR10	Verify that a user can log in and update their email address in their profile, receiving confirmation and seeing the new email reflected.	E2E	userLoginAndUpdateSettings	1. Visit /signin. 2. Fill in correct user credentials and submit the login form. 3. Assert that the user is redirected to /home and the session cookie is set. 4. Navigate to the /profile page. 5. Clear the email input and enter a new valid email. 6. Click the save button when it becomes enabled.

					7. Assert that a success message appears and the email input now displays the updated value
F14	FR11	Verify that practices can view and update their practice details and preferences.	E2E	verifiedPracticeUpdateSettings	1. Navigate to the practice settings page as a logged-in verified practice. 2. Clear the existing practice name input and enter a new name. 3. Click the Save button. 4. Wait for the update to apply. 5. Verify that a success message appears and the new name remains in the input field.
F15	FR5, FR11	Verify that verified practices can login and then view and update their practice details and preferences.	E2E	verifiedPracticeLoginAndUpdatePracticeSettings	1. Visit /signin and log in with valid verified practice credentials. 2. Assert redirect to /home. 3. Click the Practice Dashboard banner link. 4. On the dashboard, click the Manage Practice Information and Preferences card. 5. Assert redirect to /practice-dashboard/settings. 6. Clear the current name, type a new practice name. 7. Click Save. 8. Wait for the update request to complete. 9. Assert a success toast is shown and the updated name remains in the input field

Table G.1: Functional Requirement test cases

G.2 Test Traceability Matrix

A matrix mapping each functional requirement (FR) to its corresponding tests.

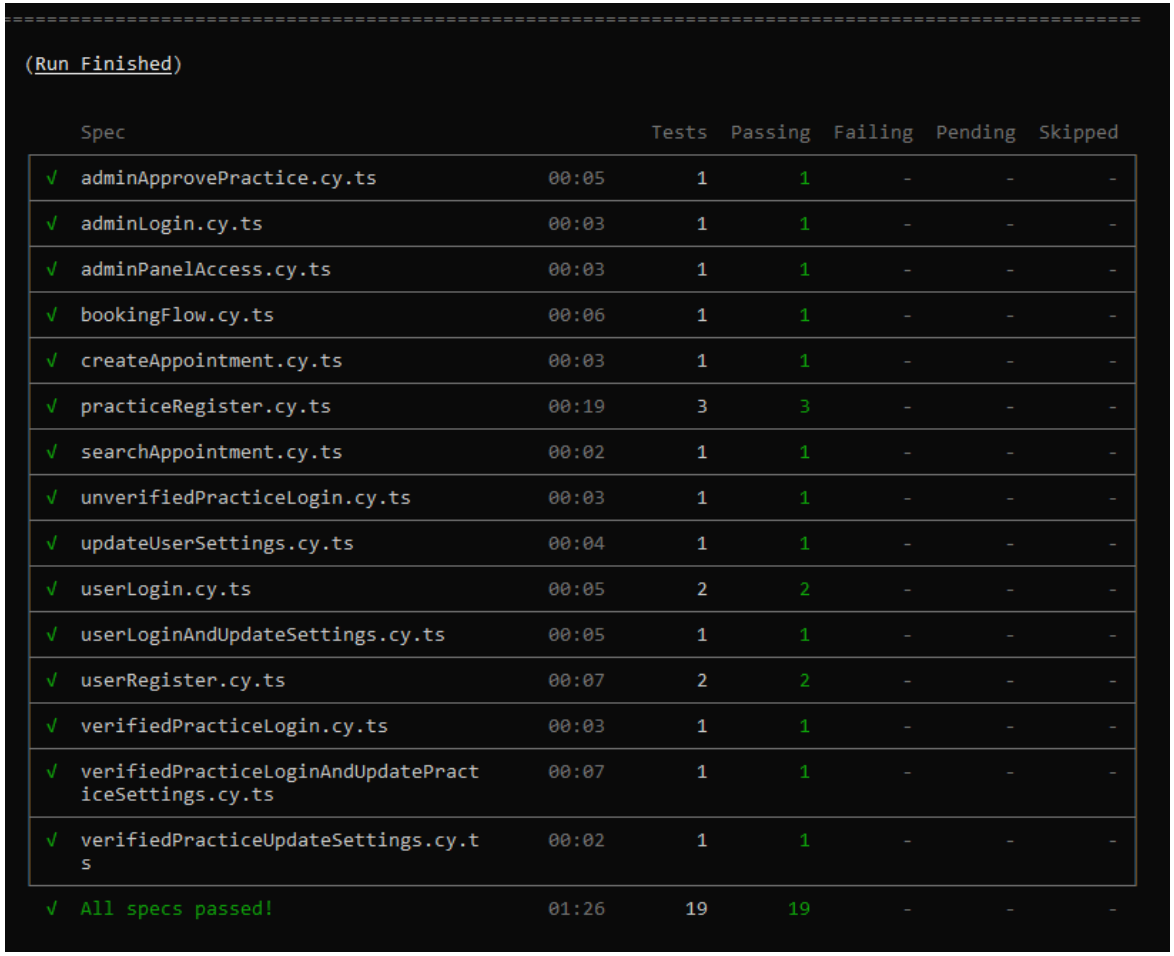


	FR1	FR2	FR3	FR4	FR5	FR6	FR7	FR8	FR9	FR10	FR11
F1	■										
F2		■									
F3			■								
F4				■							
F5					■						
F6					■						
F7						■					
F8							■				
F9								■			
F10								■			
F11								■	■		
F12										■	
F13			■							■	
F14											■
F15					■						■

Figure G.1: Test Traceability Matrix

G.3 End-To-End Test Results

Figure G.2 shows all E2E tests passing successfully after running `npx cypress run`. The booking flow test requires appointments to be populated to pass due to server-side components.



(Run Finished)

	Spec		Tests	Passing	Failing	Pending	Skipped
√	adminApprovePractice.cy.ts	00:05	1	1	-	-	-
√	adminLogin.cy.ts	00:03	1	1	-	-	-
√	adminPanelAccess.cy.ts	00:03	1	1	-	-	-
√	bookingFlow.cy.ts	00:06	1	1	-	-	-
√	createAppointment.cy.ts	00:03	1	1	-	-	-
√	practiceRegister.cy.ts	00:19	3	3	-	-	-
√	searchAppointment.cy.ts	00:02	1	1	-	-	-
√	unverifiedPracticeLogin.cy.ts	00:03	1	1	-	-	-
√	updateUserSettings.cy.ts	00:04	1	1	-	-	-
√	userLogin.cy.ts	00:05	2	2	-	-	-
√	userLoginAndUpdateSettings.cy.ts	00:05	1	1	-	-	-
√	userRegister.cy.ts	00:07	2	2	-	-	-
√	verifiedPracticeLogin.cy.ts	00:03	1	1	-	-	-
√	verifiedPracticeLoginAndUpdatePracticeSettings.cy.ts	00:07	1	1	-	-	-
√	verifiedPracticeUpdateSettings.cy.ts	00:02	1	1	-	-	-
√	All specs passed!	01:26	19	19	-	-	-

Figure G.2: Cypress E2E all test cases passing

G.4 Non-Functional Requirement Test Descriptions

Table G.2 outlines tests validating performance, usability, and other non-functional qualities of the application.

Test ID	Requirement ID	Test Description	Test Type
N1	NFR1	Run Google Lighthouse against key pages (home, search, booking) to ensure they meet accessibility and responsive design standards.	Performance
N2	NFR2	Verify via CI logs that on each push/PR, the pipeline executes linting, type-checking, unit tests, and deploy steps.	Integration
N3	NFR3	Perform automated performance tests to confirm page load and API responses complete within 2s.	Performance
N4	NFR4	Test that all password-hashing functions use bcrypt and never store plaintext.	Security
N5	NFR6	Simulate a failed deployment in the pipeline and verify that a notification email is sent to the dev team.	Integration
N6	NFR7	Monitor uptime via Vercel and confirm 99.9% availability over a 30-day window.	Monitoring

Table G.2: Non-Functional Requirement Test cases

G.4 Non-Functional Requirement Test Results

Each non-functional requirement test was validated:

N1: Detailed within Section 5.2.1.3
N2: Available here: https://github.com/danbennett239/CI601/actions
N3: Detailed within Section 5.2.1.2
N4: Detailed in Figure G.3.
N5: Detailed in Figure G.4.
N6: Detailed in Figure G.5.

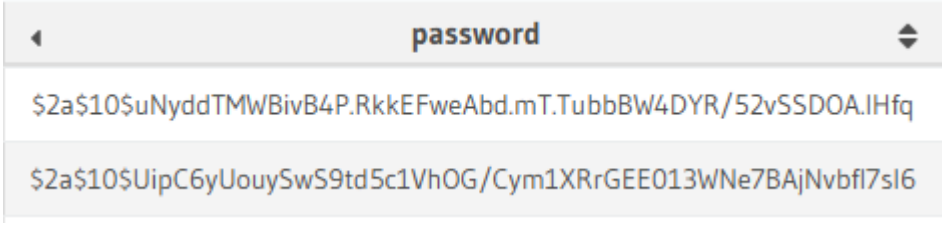


Figure G.3: Hashed passwords in Hasura.

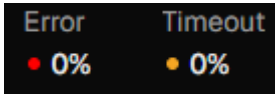


Figure G.5: Vercel availability analytics.

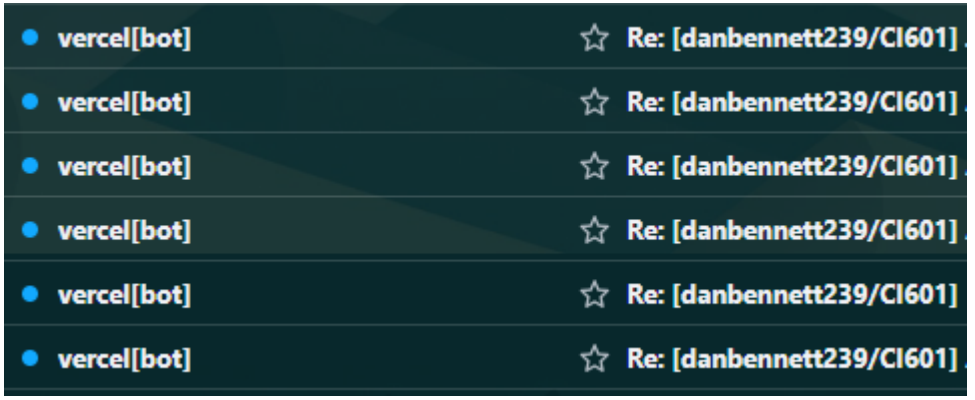


Figure G.4: Emails from Vercel regarding deployment success and errors

Appendix H: Supporting Material for Section 3: Project Requirements

H.1: Won't-Have Functional Requirements

Table H.1 details the project's Won't-Have requirements.

Requirement ID	Description	MoSCoW Priority	Linked Objective(s)
FR28	**Advanced Analytics**: In-depth metrics and reports for analysing booking trends.	Won't-Have.	O2
FR29	**Custom Appointment Types**: Enable practices to define their own appointment types.	Won't-Have.	O1
FR30	**Payment System**: Integrate with a third-party payment provider to securely process and distribute payments to practices. The system holds funds temporarily to facilitate the dispute process.	Won't-Have	O1

Table H.1: DentalConnect Won't-Have requirements.

H.2: Web Application Availability Justification

Vercel, the hosting platform of the DentalConnect web application, provides a 99.99% uptime guarantee for its service (Vercel, 2024). This equates to 4.32 minutes of downtime per month (30 days), or 52.56 minutes per year. DentalConnect will strive for 99.9% availability, adhering to the three-nines availability rule, which is considered the standard for business services (IBM, 2024; Open Minds, 2025). Three-nines allows for 43.2 minutes per month (30 days), or 525.6 minutes per year (8.76 hours). This is an achievable level through a robust CI/CD pipeline and a 99.99% hosting provider. Additionally, Vercel offers preview deployments to help test pushes before merging onto production. It also supports easy and fast rollbacks of deployments, meaning changes can be undone quickly if they break production. Future iterations of the web application can strive for four zero availability (99.99%), but this would require multiple instances, for backups, of Hasura GraphQL Engine and Neon database instances.

Appendix I: Supporting Material for Section 4: Methodology

I.1: Technical Terminology

To help understand the technical distinctions and features of React and Angular, Table I.1 provides definitions for the key terms referenced within the discussion presented in Section 4.3.2.

Term	Definition
Unidirectional data binding	Unidirectional data flow means that data moves in a single direction. In React, this means from the parent component to the child components via props (GeeksforGeeks, 2025b).
Two-way data binding	Two-way data binding gives components in an application a way to share data. Use two-way binding to listen for events and update values simultaneously between parent and child components (Angular, 2022).
Virtual Document Object Model (Virtual DOM)	The Virtual Document Object Model is an in-memory representation of the Real Document Object Model. React uses this lightweight JavaScript object to track changes in the application state and efficiently update the Real Document Object Model where necessary (GeeksforGeeks, 2025c).
Real Document Object Model (Real DOM)	The Real Document Object Model represents the structure of an HTML document in the form of a tree, where each node corresponds to an element in the document. It is an interface that allows programming languages to manipulate and interact with the content, structure, and style of a webpage (GeeksforGeeks, 2025a).
Dependency Injection	Dependency Injection is a design pattern and mechanism for creating and delivering some parts of an application to other parts of an application that require them (Angular, 2025).

Table I.1: Definitions of technical terminology used within the comparison of React and Angular.

I.2: Hasura Integration Explanation

Figure I.1 below repeats the diagram shown in Figure 4.11 to support the project-specific explanation of Hasura's integration within the DentalConnect application.

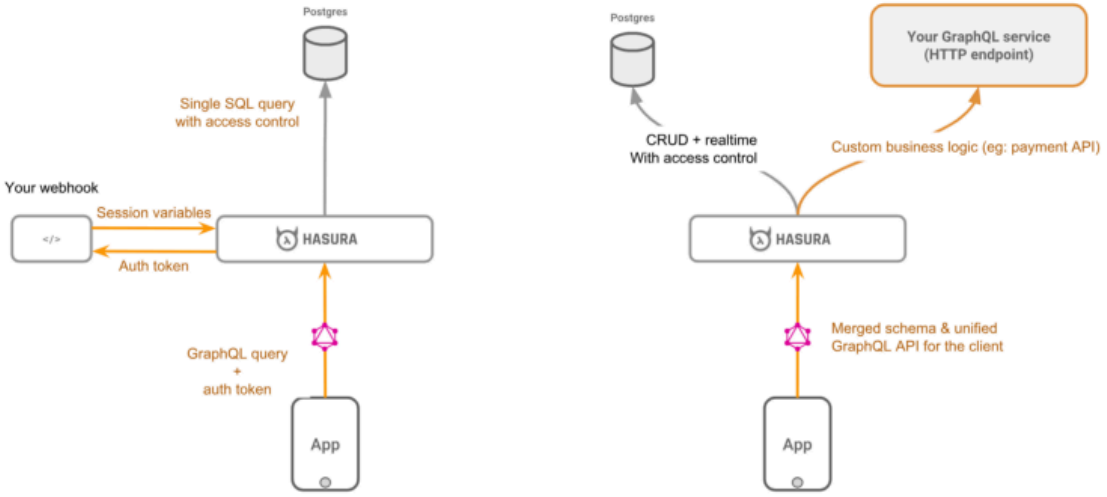


Figure I.1: How Hasura works in your stack (Hasura, 2025)

Figure I.1 illustrates how the Hasura GraphQL Engine integrates and interacts with an application, a PostgreSQL database and external services. The left side of Figure I.1 illustrates the approach DentalConnect implements. The app issues GraphQL queries that are authenticated using tokens and session variables. Hasura uses these to enforce access control and resolve the request by generating a secure SQL query. The right side of Figure I.1 shows the process of Hasura interacting with remote schemas, such as an external service like a payment API, to create a unified GraphQL API. This would possibly be used within the future expansion of DentalConnect.

I.3: CI/CD Pipeline Additional Information

This appendix provides expanded details on the CI/CD pipeline. It outlines the GitHub Actions CI workflow, Vercel and Hasura GitHub integrations, and an example of a failing CI/CD pipeline with the resolution.

I.3.1 GitHub CI Workflow

The following YAML code block defines the custom GitHub Actions CI workflow that has been implemented within the project's GitHub repository.

```yaml
name: CI Workflow

on:
  push:
    branches:
      - main
  pull_request:
    branches:
      - main

jobs:
  checks:
    runs-on: ubuntu-latest

    steps:
      # Checkout the code
      - name: Checkout code
        uses: actions/checkout@v3

      # Set up Node.js environment
      - name: Set up Node.js
        uses: actions/setup-node@v3
        with:
          node-version: 'lts/*'
          cache: 'npm'

      # Install dependencies
      - name: Install dependencies
        run: npm ci

      # Run Type Checking
      - name: Run TypeScript Type Checking
        run: npm run type-check

      # Run Linting
      - name: Run ESLint
        run: npm run lint
```

The full history of all workflows is available here:

https://github.com/danbennett239/CI601/actions

Figure I.2 shows a successful run of the GitHub Actions CI workflow.

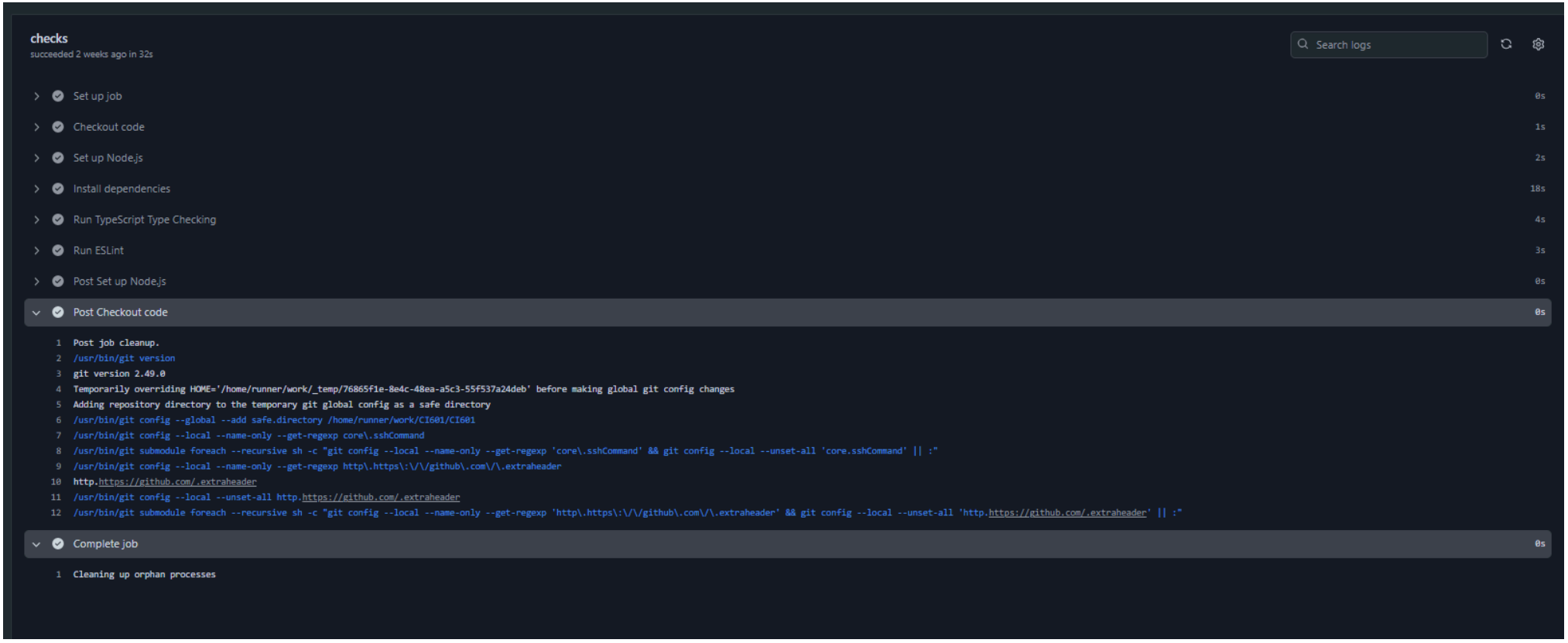


Figure I.2: Successful GitHub Actions CI workflow.

I.3.2 Vercel GitHub Integration

The Vercel GitHub integration has been set up to provide a preview deployment and validates the code. This runs on both commits and pull requests. On both success and error there are email notifications. Figure I.3 shows an example pull request with both GitHub Actions and the Vercel checks passing, providing a preview deployment.

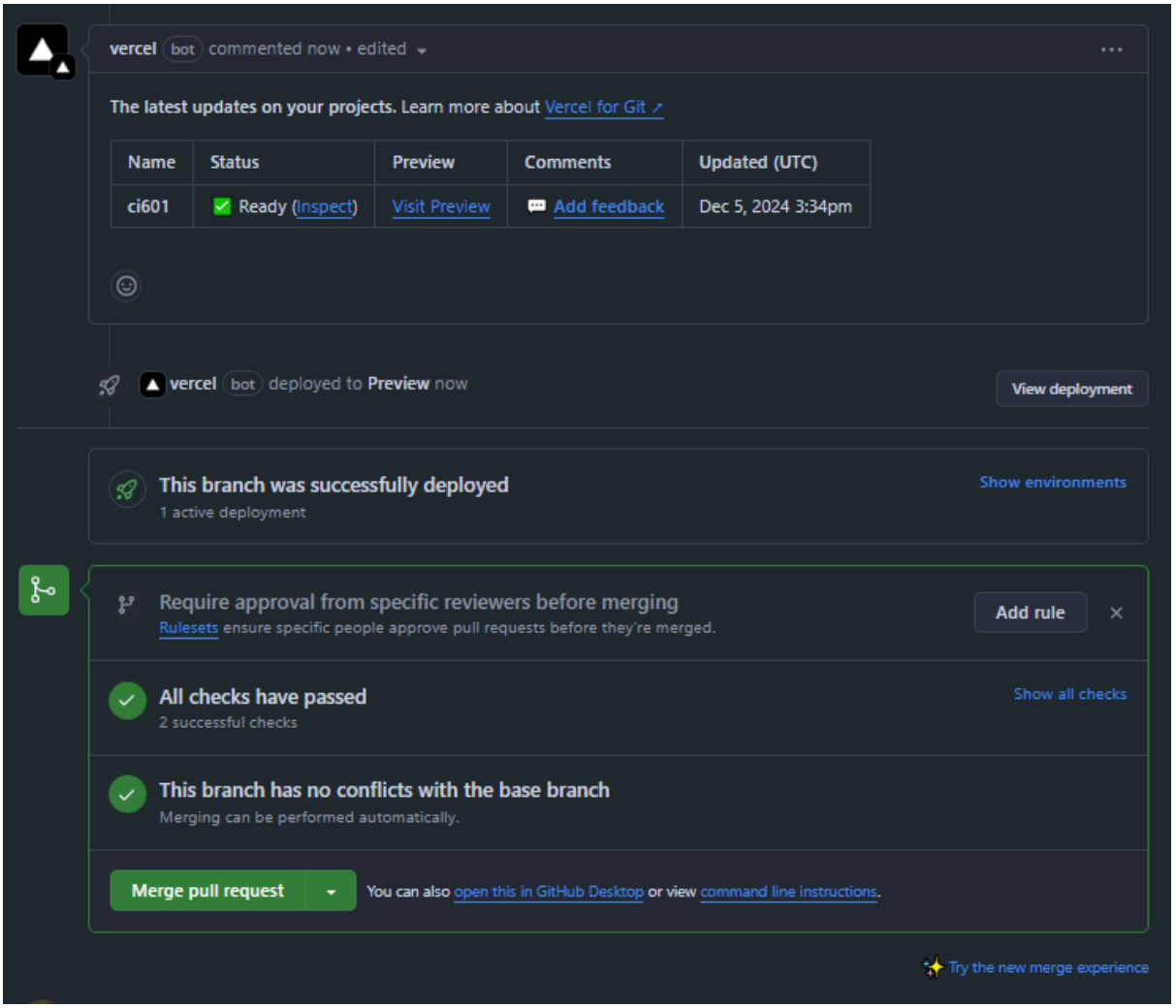


Figure I.3: Vercel GitHub integration, passing all checks and providing a preview deployment.

I.3.3 Hasura GitHub Integration

Hasura provides a GitHub integration that enables for migrations and metadata tracked via the Hasura CLI to be deployed when pushed to the GitHub repository. This means that database alterations in the local development environment that are committed are automatically added to the cloud instances of Hasura and the cloud-hosted Neon database. Figure I.4 provides an example deployment.

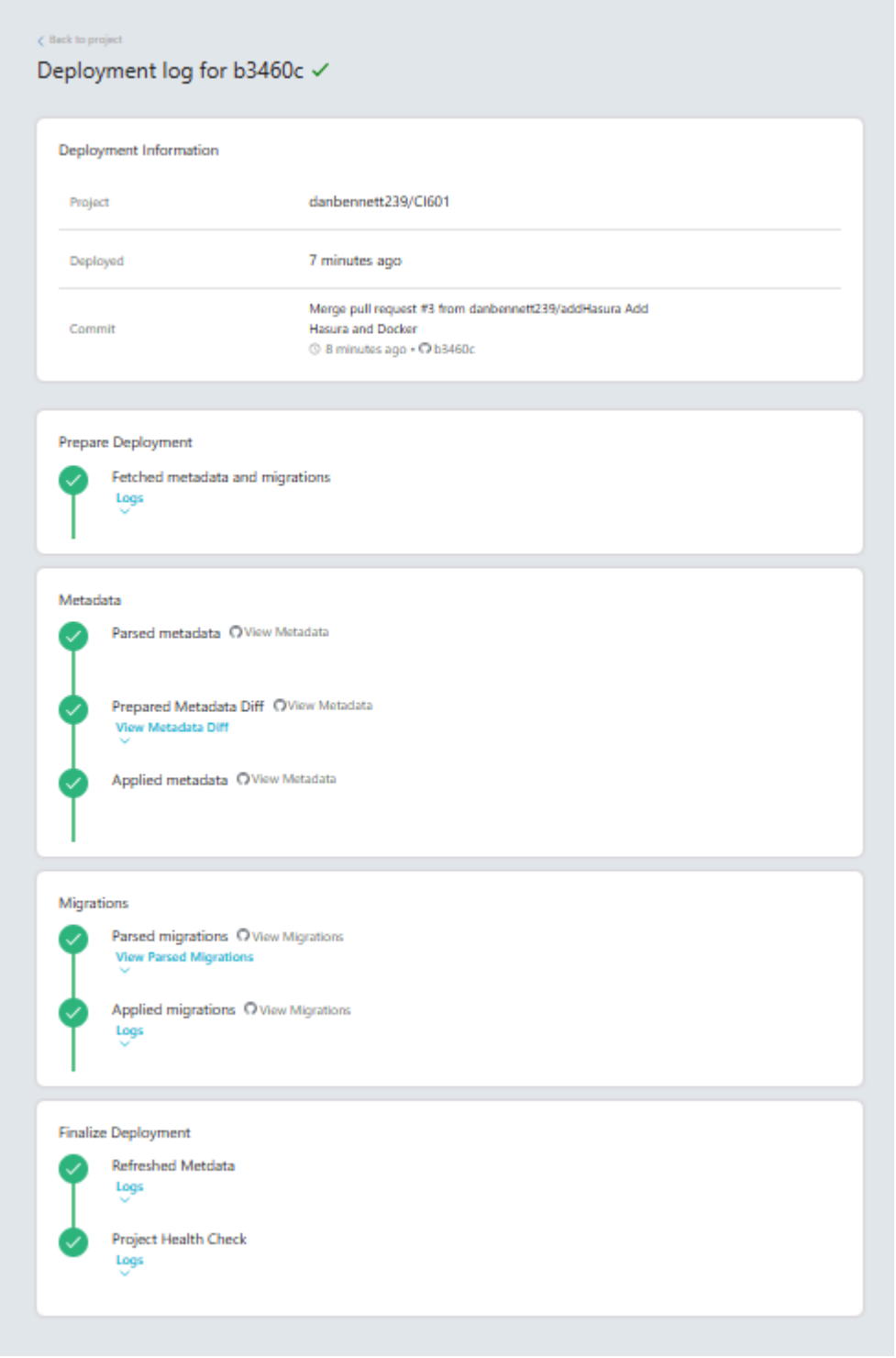


Figure I.4: Hasura GitHub integration, deploying metadata and migrations pushed to GitHub repository.

I.3.4 CI/CD Failure Example

Figure I.5 shows a CI workflow failing due to ESLint flagging an unused variable. This failure is an example of the CI/CD pipeline enforcing industry best practices. Figure I.6 presents the fix to this ESLint error.

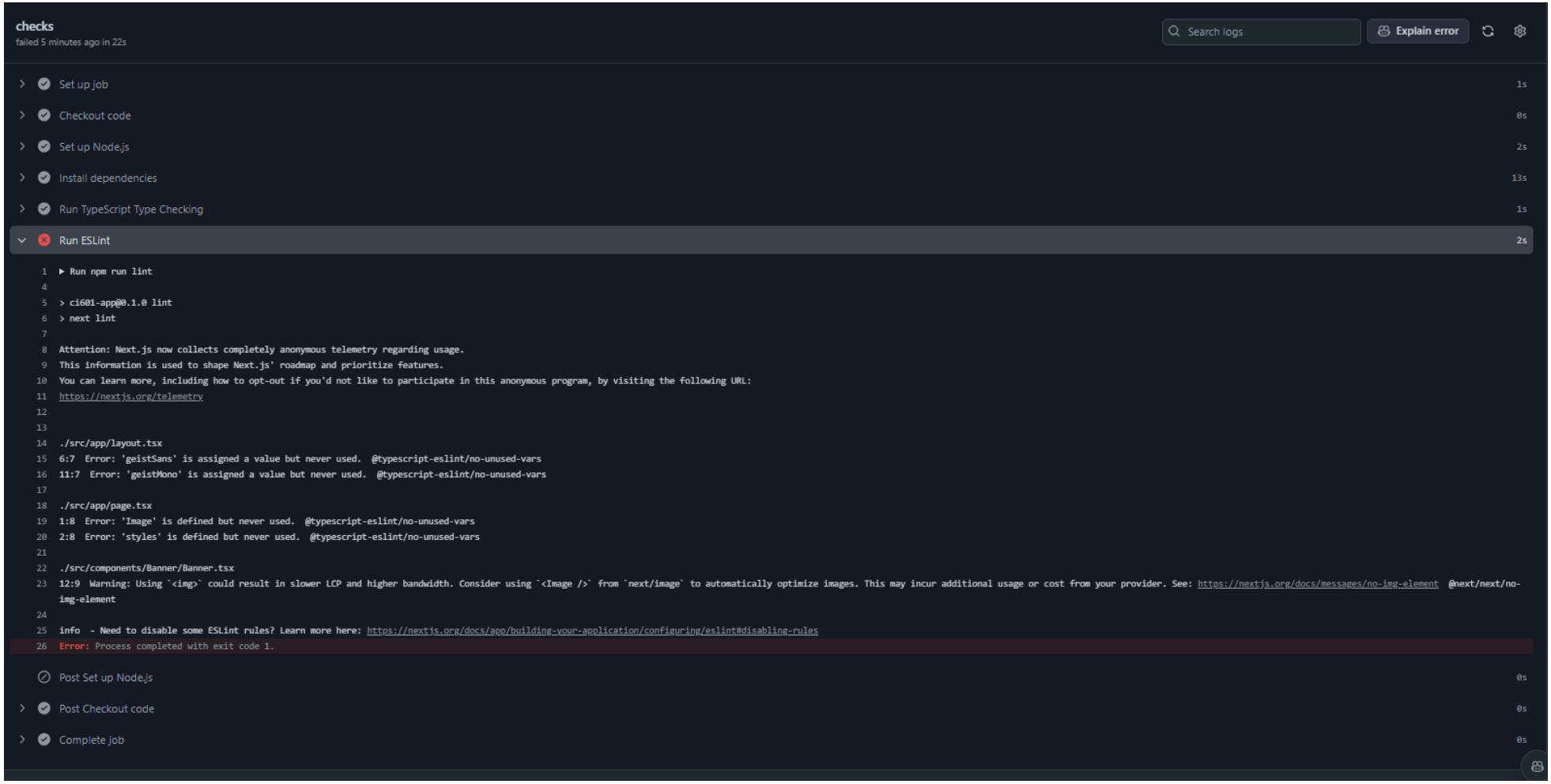


Figure I.5: ESLint unused variable failure within CI/CD pipeline.

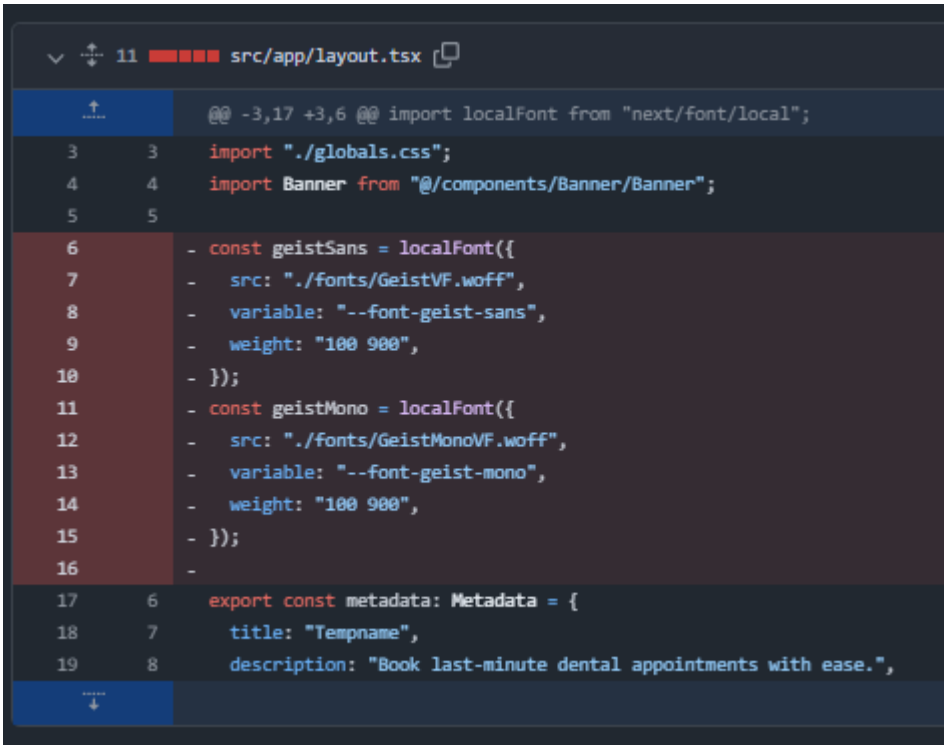


Figure I.6: Fix for ESLint unused variable.

I.4: Evaluation, Testing Tools and Techniques Expansion

Testing tools were selected based on three criteria: simple setups, minimal configuration, and clear, extensive documentation. Selecting tools based on these three criteria resulted in tools that were simple to learn and implement.

I.4.1 Unit Testing

Jest was chosen as the Unit testing tool after trialling both it and Mocha. Mocha required additional libraries and extra configuration, whereas Jest offered all the required functionality out of the box (Jest, n.d.; Mocha, n.d.).

I.4.2 End-to-end Testing

Cypress was selected as the End-To-End (E2E) testing tool after comparing it with Selenium. Cypress provides an integrated testing environment with a simple setup and minimal configuration, whereas Selenium requires additional tools to achieve the same functionality (Cypress, n.d.; Selenium, n.d.).

I.4.3 Accessibility and Performance Testing

To perform validation of the project's non-functional requirements, automated tools and manual testing were utilised:

- **Google Lighthouse**: Audit key pages on both desktop and mobile for performance, accessibility, best practices and SEO (Google Developers, 2016).
- **W3C HTML Validation**: Ensure compliance with web standards (W3C, n.d.).

Chrome DevTools: Manual testing of a range of emulated screen sizes and devices (Google Developers, 2016).

I.5: Severity, Likelihood and Impact Rating Definitions

Severity	Description
Negligible (1)	The risk will have little consequence if it occurs.
Minor (2)	The consequences of the risk will be easy to manage.
Moderate (3)	The consequences of the risk will take time to mitigate.
Major (4)	The consequences of this risk will be significant and may cause long-term damage.
Catastrophic (5)	The consequences of this risk will be detrimental and may be hard to recover from.

Table I.2: Risk severity levels (Team Asana, 2025).

Risk Impact	Description
Low (1-6)	Low-risk events that likely won't happen, and if they do, they won't cause significant consequences for the project.
Medium (7-12)	Medium-risk events are a nuisance and can cause project hiccups, but if you take action during project planning to prevent and mitigate these risks, you'll set yourself up for project success. You shouldn't ignore these risks, but they also don't need to be a top priority.
High (13-25)	High-risk events can derail your project if you don't keep them top of mind during project planning. Because these risks are likely to happen and have serious consequences, they are the most important in your risk management plan.

Table I.3: Risk impact ratings (Team Asana, 2025).

Likelihood	Description
Very unlikely (1)	It's a long shot that this risk will occur.
Not likely (2).	There's a good chance this risk won't occur.
Possible (3)	This risk could happen, but it might not. This risk has split odds.
Probable (4)	There's a good chance this risk will occur.
Very likely (5)	You can be pretty sure this risk will occur at some point in time.

Table I.4: Risk likelihood levels (Team Asana, 2025).

I.6: Identified Risks

Table I.5 details the project's identified risks, with their corresponding severity, likelihood, risk impact scores, as well as their mitigation strategies.

Risk	Severity	Likelihood	Risk Impact	Mitigation Strategy
Scope creep	4	4	16	Closely follow MoSCow prioritisation and regularly review project scope to ensure that Must-Have requirements are a priority. Refer to the project plan to monitor progress.
Time management	5	4	20	Use 2-week Agile sprints, set realistic deadlines and track tasks with Trello. Ensure ample time is given for unexpected delays or tasks. Review progress at the end of each sprint to ensure alignment with requirements and timeline.
Challenges with new technologies	3	5	15	Allocate sufficient time for research, prototyping and learning within early sprints.
Unexpected behaviour and bugs	3	5	15	Carry out regular testing and build time into the project schedule to fix any identified bugs. For singular components, conduct unit testing and end-to-end testing for the application.
Issues with external services	3	3	9	Test third-party services early and have backup services in mind.
Deployment issues	2	3	6	Implement a Continuous Integration and Continuous Deployment (CI/CD) pipeline, testing on local and staging environments, and automate error detection.
Loss of physical hardware	5	1	5	Commit code to version control (Github) and back up files in cloud storage.
Illness	3	4	12	Build a flexible project schedule and prioritise Must-Have requirements early.

Table I.5: CI601 identified risks.

Appendix J: Supporting Material for Section 5: Product Description

J.1 User Journeys

The uses cases of DentalConnect across all user roles, along with the Must-Have requirements and the functionality of the MVP informed the creation of the following 9 core user journeys:

- User registration.
- User sign-in.
- Managing user details and preferences.
- Searching for an appointment.
- Booking an appointment.
- Practice registration.
- Practice sign-in.
- Managing practice details and preferences.
- Managing appointments.

J.2 useUser Custom React Hook

The `useUser` custom React hook retrieves the current user's authentication status and role. It is used through the web application for conditionally rendering components and enforcing role-based access control.

J.2.1 useUser Implementation

The following code block provides the implementation of the `useUser` custom React hook, calling the `/api/auth/me` endpoint when invoked.

```
// hooks/useUser.ts
import { useEffect, useState } from "react";

export interface User {
  id: string;
  first_name: string;
  last_name: string;
  email: string;
  role: string;
  practice_id?: string;
}

export function useUser() {
  const [user, setUser] = useState<User | null>(null);
  const [loading, setLoading] = useState<boolean>(true);
```

```
  useEffect(() => {
    async function fetchUser() {
      try {
        // This endpoint should return a JSON object with a "user" property
        const res = await fetch("/api/auth/me", { credentials: "include" });
        if (res.ok) {
          const data = await res.json();
          setUser(data.user);
        } else {
          setUser(null);
        }
      } catch (error) {
        console.error("Error fetching user:", error);
        setUser(null);
      }
      setLoading(false);
    }
    fetchUser();
  }, []);

  return { user, loading };
}
```

The server-side Next.js route (`/api/auth/me`) then checks the user's cookie-based session and returns the user object if valid.

```
import { NextResponse } from 'next/server';
import { getUserFromCookies } from '@/lib/utils/auth';

export async function GET() {
  const user = await getUserFromCookies();
  if (!user) {
    return NextResponse.json({ error: "Unauthorized" }, { status: 401 });
  }
  return NextResponse.json({ user });
}
```

This route calls the `getUserFromCookies` function, which retrieves the `accessToken` and `refreshToken` from the user's cookies. If neither is found, it returns `null`. If an `accessToken` exists, it attempts to verify it via the `verifyToken` function.

```
/**
 * Returns the user from currently set cookies (accessToken/refreshToken).
 * If tokens are missing or invalid, returns null.
 * This function does NOT handle refreshing. It simply returns user data if valid.
 */
export async function getUserFromCookies(): Promise<UserPayload | null> {
  // If cookies() returns a Promise in your environment, await it:
  const cookieStore = await cookies();
  const accessToken = cookieStore.get('accessToken')?.value;
  const refreshToken = cookieStore.get('refreshToken')?.value;

  if (!accessToken && !refreshToken) return null;

  // Verify access token
  if (accessToken) {
    const user = verifyToken(accessToken);
    if (user) {
      return user;
    }
  }

  // If no valid access token, return null. Refreshing is handled elsewhere.
  return null;
}

export function verifyToken(token: string): UserPayload | null {
  try {
    return jwt.verify(token, JWT_SECRET) as UserPayload;
  } catch {
    return null;
  }
}
```

The `verifyToken()` function checks the JWT using the project's secret key, retrieved from either Vercel Environment Variables or local `.env`. If the token is valid, the decoded user payload is returned. If invalid, then `null` is returned. If no valid token is found, the API returns a 401 response, and the user is treated as unauthenticated.

J.2.2 useUser Example Usage

The following code block is an example taken from the `Banner` component within the web application. The `useUser` hook is invoked and the banner content is conditionally rendered depending on the user's role.

```
const { user, loading } = useUser();

{user.role === "user" && (
  <>
    <Link href="/my-appointments" onClick={closeMenu}>
      <div className={styles.navItem}>My Appointments</div>
    </Link>
    <Link href="/profile" onClick={closeMenu}>
      <div className={styles.navItem}>Profile Management</div>
    </Link>
  </>
)}
```

J.3 Practice Registration Zod Schema

The following schema is used to validate the input of the practice registration form. It ensures that required fields are provided, images are of a valid size and file type and UK postcodes are formatted correctly. It checks that at least one service with a corresponding price is populated, and one open day with valid opening and closing times is specified.

```
import { z } from "zod";

const allowedImageTypes = ["image/png", "image/jpeg", "image/jpg"];
const ukPostcodeRegex = /^(GIR
?0AA|[A-PR-UWYZ]([0-9]{1,2}|([A-HK-Y][0-9]|[A-HK-Y][0-9]([0-9]|[ABEHMNPRV-Y]))|[0-9][A-HJKPSTUW])
?[0-9][ABD-HJLNP-UW-Z]{2})$/i;

export const practiceRegistrationSchema = z
  .object({
    practiceName: z.string().min(1, "Practice name is required"),
    email: z.string().email("Invalid email address"),
    password: z.string().min(6, "Password must be at least 6 characters"),
    repeatPassword: z.string().min(1, "Please confirm your password"),
    phoneNumber: z.string().min(1, "Phone number is required"),
    photo: z
      .instanceof(File, { message: "Practice photo is required" })
      .refine((file) => file.size <= 10 * 1024 * 1024, {
        message: "File size must be less than 10MB",
      })
      .refine((file) => allowedImageTypes.includes(file.type), {
        message: "File must be a PNG or JPEG image",
      }),
    address: z.object({
      line1: z.string().min(1, "Address Line 1 is required"),
      line2: z.string().optional(),
      line3: z.string().optional(),
      city: z.string().min(1, "City/Town is required"),
      county: z.string().optional(),
      postcode: z.string().regex(ukPostcodeRegex, "Invalid UK postcode"),
      country: z.string().min(1, "Country is required"),
    }),
    openingHours: z
      .array(
        z.object({
          dayName: z.string(),
          open: z.string().optional(),
          close: z.string().optional(),
          closed: z.boolean(),
        })
      )
      .refine((hours) => hours.some((day) => !day.closed), {
        message: "At least one day must be open",
        path: ["openingHours"],
      })
      .refine(
        (hours) => {
          const timeRegex = /^\d{2}:\d{2}$/;
          return hours.every((day) => {
            if (!day.closed) {
              return (
                day.open &&
                day.close &&
                timeRegex.test(day.open) &&
                timeRegex.test(day.close) &&
```

```
                timeToMinutes(day.close) > timeToMinutes(day.open)
              );
            }
            return true;
          });
        },
        {
          message: "For open days, both opening and closing times are required and closing must be
after opening",
          path: ["openingHours"],
        }
      ),
    practice_services: z.record(z.string(), z.number().nonnegative("Price must be
non-negative")).refine(
      (services) => Object.keys(services).length > 0,
      { message: "At least one service must be selected with a price" }
    ),
  })
  .refine((data) => data.password === data.repeatPassword, {
    message: "Passwords do not match",
    path: ["repeatPassword"],
  });
```

J.4 usePractice Custom React Hook

The `usePractice` custom React hook retrieves details about a specific dental practice using its `practiceId`.

J.4.1 usePractice Implementation

The following code block provides the implementation of the `usePractice` hook. The hook calls the `/api/practice/[practiceId]` API route, which then queries the Hasura database for the practice and practice preference data. The hook provides a `refreshPractice` function to re-fetch practice data when called.

```
// hooks/usePractice.ts
"use client";

import { useEffect, useState } from 'react';
import { Practice } from '@/types/practice';

interface UsePracticeReturn {
  practice: Practice | null;
  loading: boolean;
  error: string | null;
  refreshPractice: () => Promise<void>;
}

export function usePractice(practiceId?: string): UsePracticeReturn {
  const [practice, setPractice] = useState<Practice | null>(null);
  const [loading, setLoading] = useState<boolean>(false);
  const [error, setError] = useState<string | null>(null);

  const fetchPractice = async () => {
    if (!practiceId) {
      setPractice(null);
      return;
```

```
    }
    setLoading(true);
    setError(null);

    try {
      const res = await fetch(`/api/practice/${practiceId}`);
      if (!res.ok) {
        const errorData = await res.json();
        throw new Error(errorData.error || 'Failed to fetch practice');
      }
      const data = await res.json();
      setPractice(data.practice);
    } catch (err: unknown) {
      const message = err instanceof Error ? err.message : 'Failed to fetch practice';
      setError(message);
      setPractice(null);
    } finally {
      setLoading(false);
    }
  };

  useEffect(() => {
    fetchPractice();
  }, [practiceId]);

  // Expose a refresh function to re-fetch manually
  const refreshPractice = async () => {
    await fetchPractice();
  };

  return { practice, loading, error, refreshPractice };
}
```

J.4.2 usePractice Example Usage

In the Appointments page, the usePractice hook is used alongside the useUser hook. The practiceId is extracted from the logged-in user's session and passed intousePractie to fetch the associated practice's profile and preferences. The practice object is then used to render dynamic content for the practice.

```
const { user, loading: userLoading } = useUser();
const { practice, loading: practiceLoading, error: practiceError, refreshPractice } =
usePractice(user?.practice_id);
```

An example usage of the `refreshPractice` method is when a preference is updated. The following code block details the hide delete confirmation model implementation. After the preference is saved, `refreshPractice` is called to update the local state with the latest practice preferences from the database.

```
const handleDontShowAgain = async (dontShow: boolean) => {
  if (!practice?.practice_id) return;
  try {
    if (dontShow) {
      await updatePracticePreferences(practice.practice_id, {
        hide_delete_confirmation: true,
      });

      // Re-fetch updated practice preferences
      await refreshPractice();
      toast.success("Preference updated: Hide delete confirmation");
    }
  } catch (err: unknown) {
    toast.error("Error updating preference");
  }
};
```

J.5 Homepage Hamburger Menu Code Implementation

This appendix details the implementation of the hamburger menu within the web application's navigation bar. The logic is contained within the `Banner` component, which is used within the root `layout.tsx` file of the project so that it is visible on all pages. The following code is a snippet of relevant logic from `Banner.tsx` and `Banner.module.css`.

```
// Banner.tsx
import React, { useState } from "react";
import styles from "./Banner.module.css";

const Banner: React.FC = () => {
  const [menuOpen, setMenuOpen] = useState(false);
  const toggleMenu = () => setMenuOpen(!menuOpen);

  return (
    <div className={`${styles.banner} ${menuOpen ? styles.menuOpen : ''}`}>
      <div className={styles.bannerContent}>
        <button className={styles.hamburger} onClick={toggleMenu} aria-label="Toggle menu">
          <span className={styles.hamburgerIcon}></span>
        </button>
      </div>
      <div className={`${styles.mobileNav} ${menuOpen ? styles.mobileNavOpen : ''}`}>
        <div className={styles.navItem}>Home</div>
        {/* Other nav items omitted */}
      </div>
    </div>
  );
};

export default Banner;
```

The `Banner` component uses the `menuOpen` state, set via the `toggleMenu` function, to show and hide the menu when the hamburger button is clicked.

```css
/* Banner.module.css */
.hamburger {
  display: none;
  background: none;
  border: none;
  cursor: pointer;
}

.hamburgerIcon {
  width: 25px;
  height: 3px;
  background-color: var(--white);
  position: relative;
}

.hamburgerIcon::before,
.hamburgerIcon::after {
  content: '';
  position: absolute;
  width: 25px;
  height: 3px;
  background-color: var(--white);
  transition: all 0.3s ease-in-out;
}

.hamburgerIcon::before { top: -8px; }
.hamburgerIcon::after { top: 8px; }

.menuOpen .hamburgerIcon { background-color: transparent; }
.menuOpen .hamburgerIcon::before { transform: rotate(45deg) translate(5.5px, 5.5px); }
.menuOpen .hamburgerIcon::after { transform: rotate(-45deg) translate(5.5px, -5.5px); }

.mobileNav {
  max-height: 0;
  overflow: hidden;
  transition: max-height 0.3s ease-in-out;
}

.mobileNavOpen { max-height: 500px; }

@media (max-width: 768px) {
  .navSection { display: none; }
  .hamburger { display: block; }
}
```

The CSS uses media queries to hide the desktop navigation and show the mobile navigation (hamburger menu) on devices with a width smaller than 768px. Additionally, it animates the hamburger icon into an X when the menu is open.

J.6 CSS Dynamic Unit Example

This appendix shows the use of dynamic CSS units within `HomeDashboard.module.css` for responsive UI element scaling.

```
/* HomeDashboard.module.css */
.title {
  font-size: 4.25rem;
  margin-bottom: 1.275rem;
}

@media (max-width: 768px) {
  .title {
    font-size: 3.5rem;
  }
}
```

The `.title` class uses `rem` units to dynamically set font size proportionally to the root font size of the device. This ensures consistent scaling across screen sizes. A media query is used to reduce the `.title` to `3.5rem` on devices with a width smaller than 768px.

J.7 Appointment Search Route

This appendix details the implementation of the web application's appointment search `route.ts`.

```
// /api/appointment/search/route.ts
import { NextRequest, NextResponse } from "next/server";
import { searchAppointments } from "@/lib/services/appointment/appointmentService";

export async function GET(request: NextRequest) {
  const { searchParams } = new URL(request.url);
  const filters = {
    userLat: searchParams.get("lat") ? parseFloat(searchParams.get("lat")!) : undefined,
    userLon: searchParams.get("lon") ? parseFloat(searchParams.get("lon")!) : undefined,
    maxDistance: searchParams.get("maxDistance") ? parseFloat(searchParams.get("maxDistance")!) :
undefined,
    limit: searchParams.get("limit") ? parseInt(searchParams.get("limit")!, 10) : undefined,
    offset: searchParams.get("offset") ? parseInt(searchParams.get("offset")!, 10) : undefined,
    appointmentType: searchParams.get("appointmentType")?.trim() || undefined,
    priceMin: searchParams.get("priceMin") ? parseFloat(searchParams.get("priceMin")!) : undefined,
    priceMax: searchParams.get("priceMax") ? parseFloat(searchParams.get("priceMax")!) : undefined,
    dateStart: searchParams.get("dateStart") || undefined,
    dateEnd: searchParams.get("dateEnd") || undefined,
    sortBy: searchParams.get("sortBy") as "lowest_price" | "highest_price" | "closest" | "soonest" |
undefined,
  };

  try {
    const appointments = await searchAppointments(filters);
    return NextResponse.json({ appointments });
  } catch (error: unknown) {
    console.error("Error searching appointments:", error);
    const message = error instanceof Error ? error.message : "Failed to search appointments";
    return NextResponse.json({ error: message }, { status: 500 });
  }
}
```

The route processes incoming GET requests to the `{BaseUrl}/api/appointment/search` endpoint by parsing optional query parameters into the `filters` object. This object is then passed to the `searchAppointments` method from the `appointmentService` to perform a database query. The logic is contained in a try-catch block to return either the appointment data as JSON or an error message with a status code of 500.J.8 Appointment Booking Implementation
This appendix outlines the backend implementation of the appointment booking functionality within the DentalConnect web application.

J.8.1 API Route

The following route handles the HTTP POST request when a user books an appointment. It performs basic validation and then delegates the booking functionality to the `bookAppoinment` function within the `bookingService`.

```
import { NextRequest, NextResponse } from "next/server";
import { bookAppointment } from "@/lib/services/appointment/bookingService";

interface Appointment {
  appointment_id: string;
  practice_id: string;
  title: string;
  start_time: string;
  end_time: string;
  booked: boolean;
  practice: {
    practice_name: string;
    email: string;
  };
}

interface BookingRequestBody {
  userId: string;
  email: string;
}

export async function POST(
  request: NextRequest,
  context: { params: Promise<{ appointmentId: string }> }
) {
  try {
    const { appointmentId } = await context.params;
    const { userId, email } = (await request.json()) as BookingRequestBody;

    if (!userId || !email) {
      return NextResponse.json({ error: "Missing userId or email" }, { status: 400 });
    }

    const appointment: Appointment = await bookAppointment(appointmentId, userId, email);
    return NextResponse.json({ message: "Booking confirmed", appointment }, { status: 200 });
  } catch (error: unknown) {
    const message = error instanceof Error ? error.message : "Failed to book appointment";
    console.error("Error in booking route:", message);
    return NextResponse.json({ error: message }, { status: 500 });
  }
}
```

J.8.2 bookAppointment Service Method

The function updates the specified appointment with the Hasura database to be booked and links it with the user. It then sends a booking confirmation to both the user and the practice. The emails are only sent if the booking is successful within the database and are sent using `Promise.all` so that if either of the emails being sent errors or fails it does not cause the booking to fail for the user.

```
export async function bookAppointment(appointmentId: string, userId: string, userEmail: string):
Promise<BookingAppointment> {
  const mutation = `
    mutation BookAppointment($appointmentId: uuid!, $userId: uuid!) {
      update_appointments_by_pk(
        pk_columns: { appointment_id: $appointmentId },
        _set: { booked: true, user_id: $userId }
      ) {
        appointment_id
        practice_id
        title
        start_time
        end_time
        booked
        practice {
          practice_name
          email
        }
      }
    }
  `;

  try {
    const response = await fetch(HASURA_GRAPHQL_URL, {
      method: "POST",
      headers: {
        "Content-Type": "application/json",
        "x-hasura-admin-secret": HASURA_ADMIN_SECRET,
      },
      body: JSON.stringify({
        query: mutation,
        variables: { appointmentId, userId },
      }),
    });

    if (!response.ok) {
      throw new Error(`HTTP error: ${response.status}`);
    }

    const result = await response.json();
    if (result.errors) {
      throw new Error(result.errors[0]?.message || "Failed to book appointment");
    }

    const appointment: BookingAppointment = result.data.update_appointments_by_pk;

    const practiceEmail = appointment.practice.email;
    const subject = "Appointment Confirmation";

    const userText = `
      Dear User,
      Your appointment with ${appointment.practice.practice_name} is confirmed:
      - Service: ${appointment.title}
```

```
      - Date & Time: ${new Date(appointment.start_time).toLocaleString()}
      Thank you for booking with us!
    `;
    const userHtml = `
      <p>Dear User,</p>
      <p>Your appointment with <strong>${appointment.practice.practice_name}</strong> is
confirmed:</p>
      <ul>
        <li><strong>Service:</strong> ${appointment.title}</li>
        <li><strong>Date & Time:</strong> ${new Date(appointment.start_time).toLocaleString()}</li>
      </ul>
      <p>Thank you for booking with us!</p>
    `;

    const practiceText = `
      Dear ${appointment.practice.practice_name},
      A user has booked an appointment:
      - Service: ${appointment.title}
      - Date & Time: ${new Date(appointment.start_time).toLocaleString()}
      Please prepare accordingly.
    `;
    const practiceHtml = `
      <p>Dear ${appointment.practice.practice_name},</p>
      <p>A user has booked an appointment:</p>
      <ul>
        <li><strong>Service:</strong> ${appointment.title}</li>
        <li><strong>Date & Time:</strong> ${new Date(appointment.start_time).toLocaleString()}</li>
      </ul>
      <p>Please prepare accordingly.</p>
    `;

    await Promise.all([
      sendEmail({
        to: userEmail,
        subject,
        textPart: userText,
        htmlPart: userHtml,
      }),
      sendEmail({
        to: practiceEmail,
        subject,
        textPart: practiceText,
        htmlPart: practiceHtml,
      }),
    ]);

    return appointment;
  } catch (error: unknown) {
    const message = error instanceof Error ? error.message : "Failed to book appointment";
    console.error("Error in bookAppointment:", message);
    throw new Error(message);
  }
}
```

J.9 Services Directory Structure

This appendix shows the `/services/` file directory. This directory includes all services within the web application grouped into folders by domain.

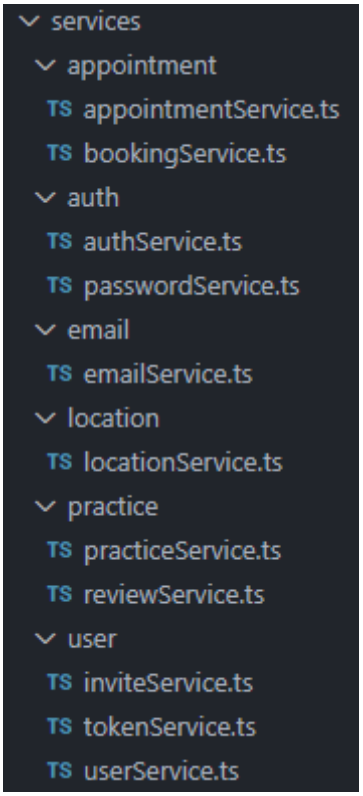


Figure J.1: Services directory.

J.10 Hasura Schema Migration History

Hasura migrations are tracked via the command line interface (CLI) from the local development Hasura instance. They are applied to the cloud-hosted instance through the GitHub integration. Full migration history with migration comments and commit comments can be viewed in the project's GitHub repository:

https://github.com/danbennett239/CI601/tree/main/hasura/migrations/default

The list of applied migrations from the Hasura console is shown in Figure J.2.

Applied Migrations

Migration Version	Database name
1741431541949	default
1738435266404	default
1738437857537	default
1740761718796	default
1741012640160	default
1741259622409	default
1738434488378	default
1741211273725	default
1741212224217	default
1741260730695	default
1733426094153	default
1741258100262	default
1737489695581	default
1739725954762	default
1740930911904	default
1741212265187	default
1740761772215	default
1740762587099	default
1740930967610	default
1733584423616	default
1738942507669	default
1738943126723	default
1739984466304	default
1740673857480	default
1741212239112	default
1741278720648	default
1741025270124	default
1741211958686	default
1738352659876	default
1738942752669	default
1739717502170	default
1739454250814	default
1739981044975	default
1740749187556	default

Figure J.2: Hasura migration history.

J.11 Coordinate Calculation

This appendix outlines the implementation of converting a user entered postcode into latitude and longitude coordinates using the Google Maps Geocoding API. This is used within the Search page for location-based appointment filtering.

J.11.1 Search Page Implementation

When the user enters a postcode, the frontend of the web application sends a request to the /api/location/geocode API route. If successful, the postcode is converted to latitude and longitude coordinates and added as parameters used to filter appointments.

```
if (filters.postcode) {
      try {
        const response = await
fetch(`/api/location/geocode?postcode=${encodeURIComponent(filters.postcode)}`);
        const data = await response.json();
        if (!response.ok) {
          throw new Error(data.error || `Geocode failed with status: ${response.status}`);
        }
        const coords: Coordinates = data;
        params.set('lat', coords.latitude.toString());
        params.set('lon', coords.longitude.toString());
        params.set('maxDistance', (filters.maxDistance * 1.60934).toString());
      } catch (err: unknown) {
        console.error('Geocode error details:', err);
        setError(err instanceof Error ? err.message : 'Failed to geocode postcode');
        setLoading(false);
        setAppointments([]);
        return;
      }
    }
```

J.11.2 API Route

The API route extracts the postcode from the query string and delegates the conversion to the `geocodePostcode` method from the `locationService`, which returns the latitude and longitude in JSON format.

```
export async function GET(request: NextRequest) {
  const { searchParams } = new URL(request.url);
  const postcode = searchParams.get('postcode');

  if (!postcode) {
    return NextResponse.json({ error: 'Postcode is required' }, { status: 400 });
  }

  try {
    const coords: Coordinates = await geocodePostcode(postcode);
    return NextResponse.json(coords);
  } catch (error: unknown) {
    console.error('Error in geocode route:', error);
    const message = error instanceof Error ? error.message : 'Failed to geocode postcode';
    return NextResponse.json({ error: message }, { status: 500 });
  }
}
```

J.11.3 Location Service

The `locationService` `geocodePostcode` method makes the API request to the Google Maps Geocoding API, returning the latitude and longitude if successful. It utilises the `GOOGLE_MAPS_API_KEY` environment variable, which is securely stored in either the local `.env` file or within the Vercel Environment Variables. This approach aligns with security best practices.

```
const GOOGLE_MAPS_API_KEY = process.env.GOOGLE_MAPS_API_KEY!;

export interface Coordinates {
  latitude: number;
  longitude: number;
}

// Geocode postcode to lat/long using Google Maps API
export async function geocodePostcode(postcode: string): Promise<Coordinates> {
  const url =
`https://maps.googleapis.com/maps/api/geocode/json?address=${encodeURIComponent(postcode)}&key=${GOO
GLE_MAPS_API_KEY}`;
  try {
    const response = await fetch(url);
    if (!response.ok) {
      throw new Error(`HTTP error: ${response.status}`);
    }
    const data = await response.json();
    if (data.status !== 'OK' || !data.results[0]) {
      throw new Error('Invalid postcode or no results found');
    }
    const { lat, lng } = data.results[0].geometry.location;
    return { latitude: lat, longitude: lng }; // Matches Coordinates interface
  } catch (error: unknown) {
    console.error('Error geocoding postcode:', error);
    throw new Error(error instanceof Error ? error.message : 'Failed to geocode postcode');
  }
}
```

J.12 appointments_with_distance View SQL Definition

This appendix shows the `appointments_with_distance` view SQL definition. The appointments and practices tables are combined via a `JOIN` on `a.practice_id = p.practice_id` and a dummy `distance` column with a default value of 0.

```
CREATE
OR REPLACE VIEW "public"."appointments_with_distance" AS
SELECT
  a.appointment_id,
  a.practice_id,
  a.user_id,
  a.title,
  a.start_time,
  a.end_time,
  a.booked,
  a.created_at,
  a.updated_at,
  a.services,
  a.booked_service,
  p.practice_name,
  p.email,
  p.phone_number,
  p.photo,
  p.address,
  p.verified,
  p.verified_at,
  p.location,
  (0) :: double precision AS distance
FROM
  (
    appointments a
    JOIN practices p ON ((a.practice_id = p.practice_id))
  );
```

J.13 get_nearby_appointments Function SQL Definition

The `get_nearby_appointments` function enables the filtering and sorting of upcoming unbooked appointments by user-defined criteria. It supports pagination and multiple sorting options and is used within the search appointments page.

PostGIS geospatial functions calculate the distance between user-specified coordinates and practice locations. If coordinates are provided, appointments can be filtered by proximity using `ST_DWithin` and sorted by distance using `ST_Distance`.

The function returns appointments via the `appointments_with_distance` view, which includes appointment and practice data along with the calculated `distance` column.

The full SQL definition is shown below:

```
CREATE
OR REPLACE FUNCTION public.get_nearby_appointments(
  user_lon double precision DEFAULT NULL :: double precision,
  user_lat double precision DEFAULT NULL :: double precision,
  max_distance double precision DEFAULT 10,
  limit_num integer DEFAULT 20,
  offset_num integer DEFAULT 0,
  appointment_type text DEFAULT NULL :: text,
  price_min numeric DEFAULT NULL :: numeric,
  price_max numeric DEFAULT NULL :: numeric,
  date_start timestamp without time zone DEFAULT NULL :: timestamp without time zone,
  date_end timestamp without time zone DEFAULT NULL :: timestamp without time zone,
  sort_by text DEFAULT 'soonest' :: text
) RETURNS SETOF appointments_with_distance LANGUAGE sql STABLE AS $ function $
SELECT
  a.appointment_id,
  a.practice_id,
  a.user_id,
  a.title,
  a.start_time,
  a.end_time,
  a.booked,
  a.created_at,
  a.updated_at,
  a.services,
  a.booked_service,
  p.practice_name,
  p.email,
  p.phone_number,
  p.photo,
  p.address,
  p.verified,
  p.verified_at,
  p.location,
  CASE
    WHEN user_lon IS NOT NULL
    AND user_lat IS NOT NULL THEN ST_Distance(
      ST_SetSRID(p.location, 4326) :: geography,
      ST_SetSRID(ST_MakePoint(user_lon, user_lat), 4326) :: geography
    ) / 1000 -- Distance in km
    ELSE 0 :: double precision
  END AS distance
FROM
  appointments a
  JOIN practices p ON a.practice_id = p.practice_id
WHERE
  a.booked = false
  AND (
    date_start IS NULL
    OR a.start_time >= date_start
  )
  AND (
    date_end IS NULL
    OR a.start_time <= date_end
  )
  AND (
    appointment_type IS NULL
    OR a.services ? appointment_type
  )
  AND (
    price_min IS NULL
    OR (
      (a.services ->> appointment_type) :: numeric >= price_min
```

```
    )
  )
  AND (
    price_max IS NULL
    OR (
      (a.services ->> appointment_type) :: numeric <= price_max
    )
  )
  AND (
    user_lon IS NULL
    OR user_lat IS NULL
    OR ST_DWithin(
      ST_SetSRID(p.location, 4326) :: geography,
      ST_SetSRID(ST_MakePoint(user_lon, user_lat), 4326) :: geography,
      max_distance * 1000 -- Convert km to meters
    )
  )
ORDER BY
  CASE
    WHEN sort_by = 'lowest_price'
    AND appointment_type IS NOT NULL THEN (a.services ->> appointment_type) :: numeric
  END ASC,
  CASE
    WHEN sort_by = 'highest_price'
    AND appointment_type IS NOT NULL THEN (a.services ->> appointment_type) :: numeric
  END DESC,
  CASE
    WHEN sort_by = 'closest'
    AND user_lon IS NOT NULL
    AND user_lat IS NOT NULL THEN ST_Distance(
      ST_SetSRID(p.location, 4326) :: geography,
      ST_SetSRID(ST_MakePoint(user_lon, user_lat), 4326) :: geography
    ) / 1000
  END ASC,
  CASE
    WHEN sort_by = 'soonest' THEN a.start_time
  END ASC
LIMIT
  limit_num OFFSET offset_num;
$ function $
```

J.14 PostGIS PostgreSQL Setup

To support the distance filtering, geospatial functionality from PostGIS was enabled within the Neon PostgreSQL instance. Initially, geospatial queries failed due to the extension not being installed. This was confirmed by checking the `installed_version` column value of PostGIS (Figure J.3). After installation, this value was checked again to confirm success (Figure J.4 and Figure J.5).

```sql
SELECT name, default_version, installed_version
FROM pg_available_extensions
WHERE name LIKE 'postgis%';
```

Figure J.3: SQL query to validate PostGIS installation status.

```sql
CREATE EXTENSION IF NOT EXISTS postgis;
```

Figure J.4: SQL query to enable PostGIS.

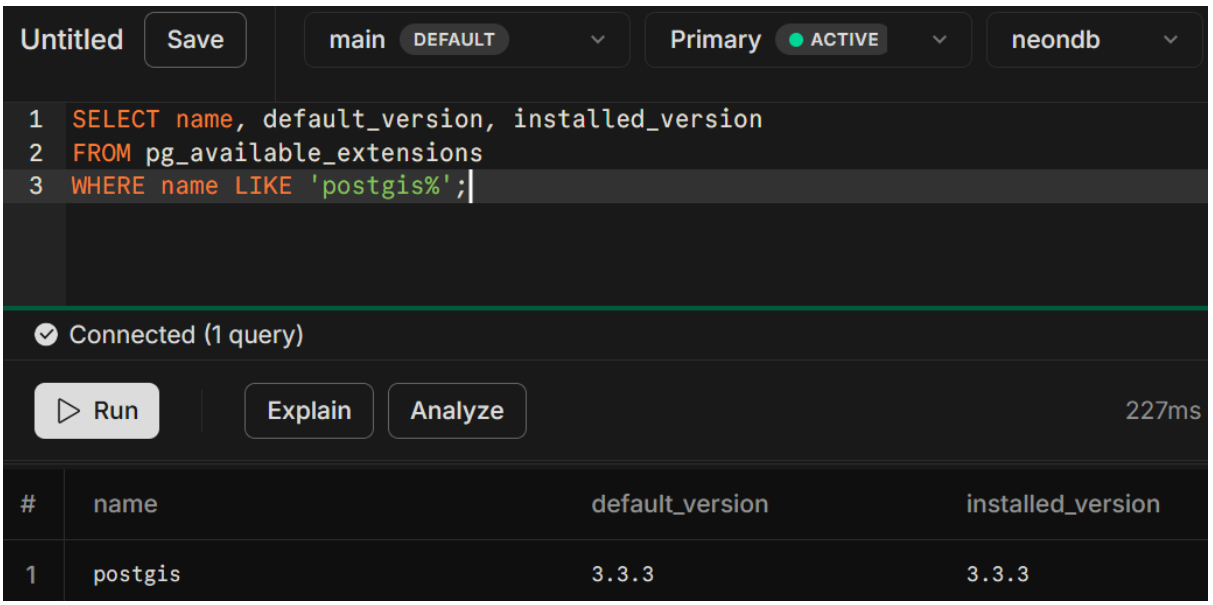


Figure J.5: SQL query executed to confirm successful PostGIS installation.

PostGIS was chosen for the storage of location data and for distance calculations for several reasons (PostGIS Project, 2023):

- **Performance**: Geospatial queries are a core requirement (e.g., nearest appointments, max distance filtering). A separate geometry column with a GiST index outperforms JSONB extraction, especially as your dataset grows.
- **Simplicity**: Hasura/PostGIS handles geometry filtering natively, avoiding complex JSONB parsing in queries.
- **Future-Proofing**: If you expand to other geospatial features (e.g., polygons for practice service areas), a dedicated column aligns better with PostGIS capabilities.

This approach ensures efficient, scalable, and maintainable geospatial functionality within the application for distance-based filtering.

J.15 Search Filtering Example

The following example demonstrates how user-entered filters are translated into the `get_nearby_appointments` SQL function.

The user searches for:

- **Appointment type**: checkup
- **Price range**: 30 - 60
- **Postcode** (converted to coordinates): lat: 50.9, long: -1.3
- **Distance**: 10 miles
- **Date range**: May 1 to May 31, 2025

These are sent to the search appointment API endpoint:

```
GET /api/appointment/search?appointmentType=checkup
  &priceMin=30
  &priceMax=60
  &lat=50.9
  &lon=-1.3
  &maxDistance=16.0934 — Miles converted to KM by front-end
  &dateStart=2025-05-01
  &dateEnd=2025-05-31
  &sortBy=soonest
```

Which results in the following SQL call:

```
SELECT * FROM public.get_nearby_appointments(
  user_lon := -1.2,
  user_lat := 50.9,
  max_distance := 16.0934,  -- 10 miles in km
  limit_num := 20,
  offset_num := 0,
  appointment_type := 'checkup',
  price_min := 30,
  price_max := 60,
  date_start := '2025-05-01',
  date_end := '2025-05-31',
  sort_by := 'soonest'
);
```

J.16 Container-Presenter Pattern

The container-presenter pattern is applied in the web application through Next.js pages and React components. The pages are responsible for the orchestration of API calls, data filtering, state management and the distribution of data to React components with the sole responsibility of rendering. An example implementation of the pattern is within the search functionality of the web application (FR6). The search page (`/search/page.tsx`) acts as the container, processing the user-inputted filter selections from the `FilterComponent` via the `applyFilters()` callback function, passed as a prop. The filters are saved to local storage so that when a user navigates into an `AppointmentCard` or further into the booking flow and returns to the search page the filters are persisted. The callback function triggers the `fetchAppointments()` method which queries `/api/appointment/search ?{userFilters}`and the returned data is passed to the presenter component, `AppointmentCard`, for rendering. The approach was selected due to the simplicity, scalability and reusable presenters which align with the evolving requirements. In addition, the separation of concerns and sole responsibility improves maintainability and abridges debugging due to the encapsulated logic.

J.17 Next.js Dynamic Routing

Next.js dynamic `[id]` routing is leveraged for the booking flow (FR1) within `/appointment/[id]/page.tsx` to ensure persistence and enable flexible navigation for an enhanced user experience. When a user selects an appointment from the search results they are directed to a unique booking page with the URL `/appointment/{appointmentId}`. This approach provides a persistent state throughout the booking flow, enabling users to navigate forwards and backwards through steps. Additionally, the server-side data fetching ensures that booked appointments cannot be booked again, with an error message shown where needed. The appointment type and source page are sent as query parameters, enabling the “Select Service” dropdown to be pre-populated with the user searched appointment type, and for the “Return” button to conditionally navigate back to either the search page or homepage, further improving user experience.

J.18: Third Party Integrations

J.18.1 Mailjet

Mailjet was selected as the project's third-party email provider due to its simplicity in setup and usage. It is used to send transactional emails such as password reset links and booking confirmations. An API key and secret were generated through the Mailjet dashboard and securely stored as environment variables (Mailjet, n.d.). DentalConnect's emails are handled via a centralised `emailService` that receives an email object and processes the request through the Mailjet API, shown within the following code block. This approach enables the functionality to be reused through the application, reducing boilerplate code and increasing codebase maintainability.

```
export async function sendEmail(options: SendEmailOptions): Promise<void> {
  const mailjet = Mailjet.apiConnect(
    process.env.MAILJET_API_KEY ?? "",
    process.env.MAILJET_API_SECRET ?? ""
  );

  await mailjet.post("send", { version: "v3.1" }).request({
    Messages: [
      {
        From: {
          Email: process.env.MAILJET_SENDER_EMAIL ?? "",
          Name: "DentalConnect",
        },
        To: [
          {
            Email: options.to,
          },
        ],
        Subject: options.subject,
        TextPart: options.textPart,
        HTMLPart: options.htmlPart,
      },
    ],
  });
}
```

J.18.2 Amazon S3 Bucket

Amazon Simple Storage Service (S3) is used to store the uploaded dental practice images due to its scalability, efficiency, and performance (AWS, n.d.). Storing images directly in a database using base64 encoding is a poor architectural choice, as the encoding of binary image data inflates the file size by approximately 33% (MDN Web Docs, 2024). This increases the storage requirements, bloats the database unnecessarily, and impacts read and write operation performance. S3 decouples the media store from the web application, leading to a lean and performant database. S3 is designed to handle large, unstructured data efficiently, and supports high availability and duality (AWS, n.d.).

Image uploads are handled in a dedicated `s3Service` that uses the `PutObjectCommand` from `@aws-sdk/client-s3` to send file buffers to S3. A unique file key is generated using a timestamp, and the public file URL is returned and stored within the practice record after a successful upload:

```
const key = `uploads/${Date.now()}-${fileName}`;
await s3.send(new PutObjectCommand({
  Bucket: BUCKET_NAME,
  Key: key,
  Body: fileBuffer,
  ContentType: fileType,
}));
const fileUrl = `https://${BUCKET_NAME}.s3.${REGION}.amazonaws.com/${key}`;
```

A dedicated IAM user is granted scoped permissions for only the necessary actions within the `uploads/` folder of the S3 bucket. The policy allows uploading (`PutObject`), retrieval (`GetObject`), and optional deletion (`DeleteObject`) of files within the `uploads/` prefix, but explicitly denies listing all objects to enhance security. This ensures that only direct access to known files is permitted, and the contents of the entire bucket cannot be enumerated. (AWS, n.d.)

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:PutObject",
        "s3:GetObject",
        "s3:DeleteObject"
      ],
      "Resource": "arn:aws:s3:::ci601-dental-practice-images/uploads/*"
    }
  ]
}
```

J.18.3 Google Geocoding

To protect the Google Maps Geocoding API key, HTTP referrer restrictions were applied via the Google Cloud Console. This ensures that only requests originating from the DentalConnect's web application can successfully access the API via the credentials. This mitigates unauthorised usage and aligns with best practices for securing third-party APIs (Google Developers, 2025). For full implementation details, please refer to Appendix J.11.

Appendix K: Supporting Material for Section 6: Research

K.1 Appointment Types and Duration

This research involved reviewing articles from the Crescent Heights Dental Clinic and The Yardley Clinic detailing the most common types of dental procedures (Crescent Heights Dental Clinic, n.d.; The Yardley Clinic, n.d.). The analysis, along with an article from Assembly Dentist, led to grouping the procedures into four main categories: diagnostic, preventative, restorative and cosmetic (Assembly Dental, n.d.). With categories defined, six appointment types were selected to be supported within the web application providing coverage across the categories and ensuring the web application supports routine and urgent care. The appointment types are detailed within Table K.1.

Appointment Type	Category	Further Rationale
Check-Up	Diagnostic	63% of adults attend regular check-ups, with a further 15% attending occasionally (Office for Health Improvement and Disparities, 2024a).
Cleaning	Preventative	33.1 percent of adult clinical treatments included Scale and Polish (NHS Digital, 2023).
Whitening	Cosmetic	43% of UK adults are interested in non-surgical cosmetic procedures, with teeth whitening being the most commonly considered (Mintel, 2018).
Filling	Restorative	67% of the British population had at least 1 filling, and over 50% had more than 3 (Oral Health Foundation, n.d.).
Extraction	Restorative	74% of adults have had a tooth extracted (Oral Health Foundation, n.d.).
Emergency	Other	200,000 A&E visits take place for patients with dental problems every year ({my}dentist, 2021).

Table K.1: Selected Appointment Types for the Web Application

Having established the appointment types the web application will offer, additional research was undertaken to determine the typical duration of each appointment. Understanding the average length of each appointment is crucial to ensuring the web application's scheduling system enables dental practices to create and manage appointments effectively. The typical appointment duration for each appointment type is presented in Table K.2 (Cheslyn Hay Dental Practice, n.d.; iCare Dental, n.d.).

Appointment Type	Typical Duration (Minutes)
Check-Up	30 to 60
Cleaning	30 to 45
Whitening	60 to 120
Filling	20 to 60
Extraction	20 to 60
Emergency	Variable

Table K.2: Typical Durations of Selected Appointment Types

K.2 Competitor Analysis

Competitor analysis evaluated how other web applications implement a solution for online dental appointment aggregation, searching and booking. Although there are currently no web applications providing a solution for the same scope as this project, two web applications, Bupa Dental Care and {my}dentist, have some overlap in the functionality and service they provide. Both web applications do not integrate with independent dental practices, opting only to support dental practices within their network. The findings influenced the web application through identifying features to adopt and gaps to address, which helped to create and shape the project objectives, functional and non-functional requirements.

K.2.1 Full Bupa Dental Care Competitor Analysis

Bupa Dental Care operates a network of over 350 NHS and private dental practices across the UK (Bupa Dental Care, 2025). The web application enables users to search for a range of treatment options and to book an appointment with a dental practice within the Bupa network. The web application also offers Bupa's other services including health insurance, health subscriptions and care homes, however these are outside of the scope of the project.

Category	Details
Key Features	**Searching for a practice**: The user is able to search for nearby dental practices by entering either a postcode or by granting location permissions. Alternatively the user is able to browse dental practices by region and city. After entering location, dental practices, sorted ascendingly by distance, are displayed via cards containing an image, practice name, address, opening hours and contact information (Appendix K.2.1 and Appendix K.2.2). The search results can be filtered by treatments, services and patient acceptance (NHS or private). Alongside the dental practice cards a map with pins for each dental practice in the search results is displayed (Appendix K.2.3). Upon clicking into a practice, additional information about the practice, treatments and the corresponding prices are shown (Appendix K.2.4 and Appendix K.2.5). The user is presented with the option to book an appointment via a button which moves them into the booking flow. **Booking an appointment**: Booking an appointment follows on from the searching for a practice flow. The web application prompts the user to either sign in or create an account. The available treatment options are

	displayed and after selection a price is shown. An option is given to select any dentist or a specific dentist and after selection available appointment slots are shown sorted by date. Upon choosing an appointment slot a summary popup including the appointment time and type, dentist's name and practice's location appears with a confirm booking button. After confirming the booking the user receives a confirmation email and is about to view their booked appointments within the web application.
Strengths	**Searching for a practice**: ● The web application allows for additional filtering of dental practices after the initial location filter. **Booking an appointment**: ● The web application displays a booking summary popup for a final review of details before confirming the booking, reducing the likelihood of a booking error. ● Requires user sign-inn ensures secure access to booking features and allows users to manage appointments. **General**: ● The web application provides blog posts containing general information on dental care and fill space on the homepage. ● The web application is mobile responsive and accessible. ● The web application allows for users to review a practice via a feedback form. ● The user interface (UI) contains tooltips which provide additional information on hover. This keeps the UI minimal while also offering additional context and guidance.
Weaknesses	**Searching for a practice**: ● The web application lacks 'sort by' functionality for dental practice search results, only sorting by distance ascending. ● To view prices for a treatment navigating into a specific dental practice is required. There is no way to quickly view a price or to compare prices between practices. **Booking an appointment**: ● The viewing of appointment availability is too late within the booking flow, requiring a user to restart the process if slots do not meet their needs. For example, a user who needs an appointment within 3 days may find the practice they select only has slots in 2 weeks after selecting the practice, treatment and dentist.

	• Navigation within the booking flow is poor, when attempting to return to the practice page the user gets stuck at the sign in or create an account page. **General**: • The web application is not performant and is very slow to use. Verified via a Google Lighthouse test, results are available in Appendix K.2.6 and Appendix K.2.7.
Influence on project	The analysis of the Bupa Dental Care web application informed several aspects of the web application: **Practice Registration** • Bupa Dental Care's web application provided an insight into the information that will be collected during dental practice registration through looking at the information provided to users about a given practice. The fields can be seen within the wireframes of the practice registration within Appendix D.3. • DentalConnect supports independent dental practices, whereas all practices listed on Bupa Dental Care's web application are within the Bupa network. As a result DentalConnect will require a verification process, performed by a system admin, post practice registration to ensure that only legitimate practices are on the DentalConnect web application. **User Registration** • Bupa Dental Care's web application inspired the information that will be collected during user registration. It validated that dental appointments can be booked with basic information, name and contact details, with no further information required such as medical data. The fields can be seen within the wireframes of the user registration within Appendix D.2. **Practice Appointment Management** • DentalConnect's appointment management system within the practice dashboard will display the associated patient contact details (provided during user registration) within a booked appointment. **Searching for an appointment**: • Search results will be shown in a card view, similar to Bupa Dental Care, as it clearly conveys information. Additionally, it will be responsive across a range of devices without making major user interface alterations. • Bupa Dental Care's web application informed the practice and appointment information that will be shown to the user within the card view.

	• Within the Bupa Dental Care's web application, the user searches for a practice where they then select a treatment and view available appointments. DentalConnect will allow users to search for an appointment directly, with the ability to filter by distance, price, date etc. This approach enables the user to instantly view and compare appointment availability and prices. **Booking an appointment**: • The user will be required to sign in during the booking flow, which will enable a user to view and manage their booking. • Within the same manage bookings screen the user will be able to leave reviews for practices they have had an appointment with. • Requiring sign-in for booking will also enable the practices to view the contact details of the user that has booked a given appointment • At the end of the booking flow a summary screen will be shown to the user to reduce the likelihood of miss-booking appointments. • When using Bupa's booking flow the user gets stuck. DentalConnect will ensure that the user can return to a previous stage in the booking flow and not get stuck. This will be verified in testing. • Appointments will not be taken out of search results while being viewed as it is required for the user to view extra details. However, logic checks will be performed before booking to ensure no duplicate bookings, and error messages will be shown to the user as needed. **General**: • Bupa Dental Care's use of a map view informed the decision to include a map view to provide geographic visualisation within appointment search results. • Bupa Dental Care's use of blog posts to provide general dental care information informed the decision to include a similar feature with DentalConnect. This will promote user engagement, patient awareness and Search Engine Optimisation (SEO) optimisation. • The Bupa Dental Care web application allows users to leave a review for a practice. This project's web application will contain a review system enabling users to share feedback and help other users make informed decisions. • The Bupa Dental Care web application contained tooltips which offer additional context and guidance, improving the user experience. This project's web application will use tooltips where appropriate, such as explaining options in preference menus.

Table K.3: Bupa Dental Care Competitor Analysis

K.2.2 Bupa Dental Care Competitor Analysis Supporting Screenshots

K.2.1: Screenshot of Bupa Dental Care Practice Search Page (Desktop With Location Filter Applied)

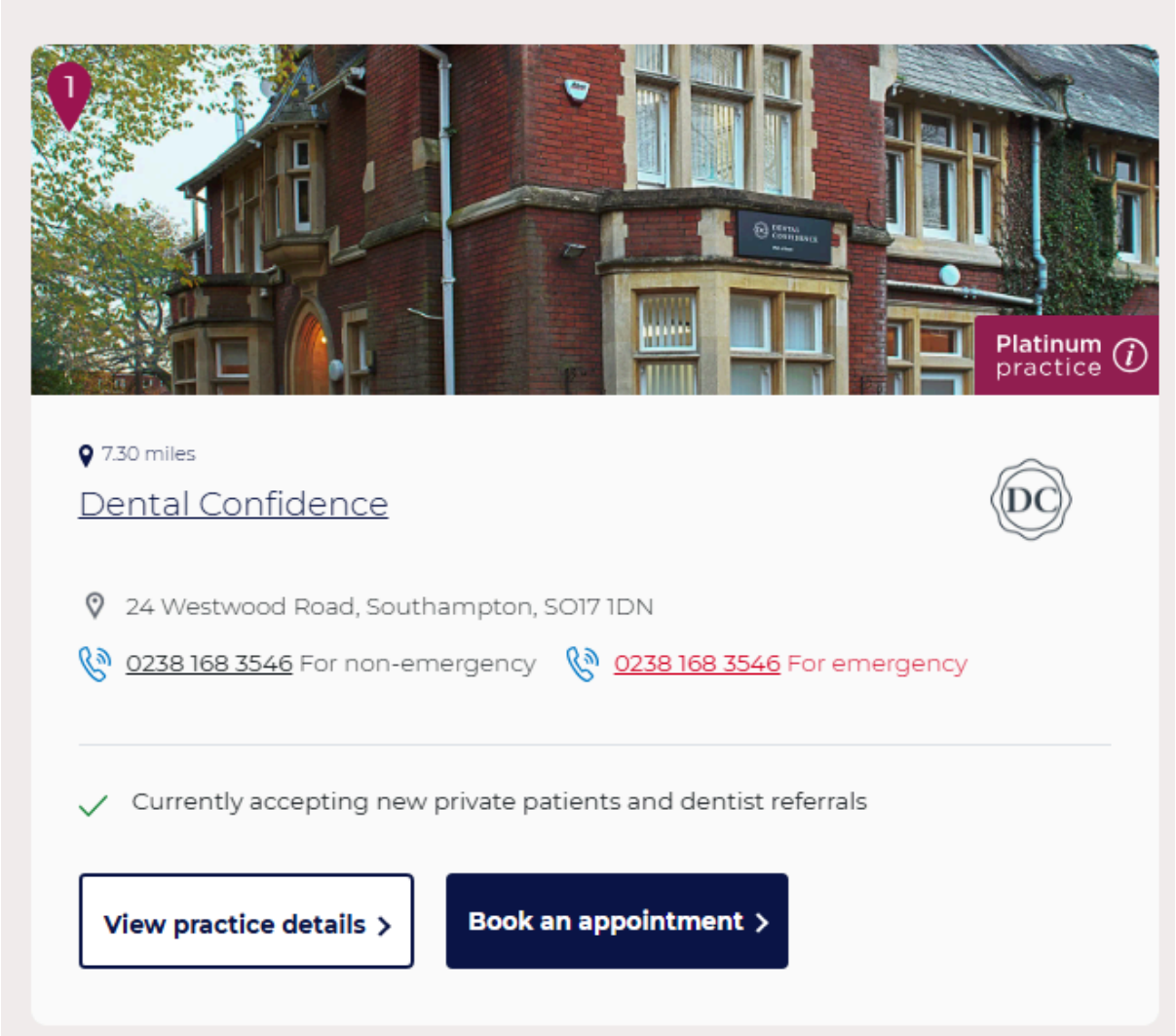


Figure K.1: Practice Search Page (Desktop).

K.2.2: Screenshot of Bupa Dental Care Practice Search Page (Mobile With Location Filter Applied)

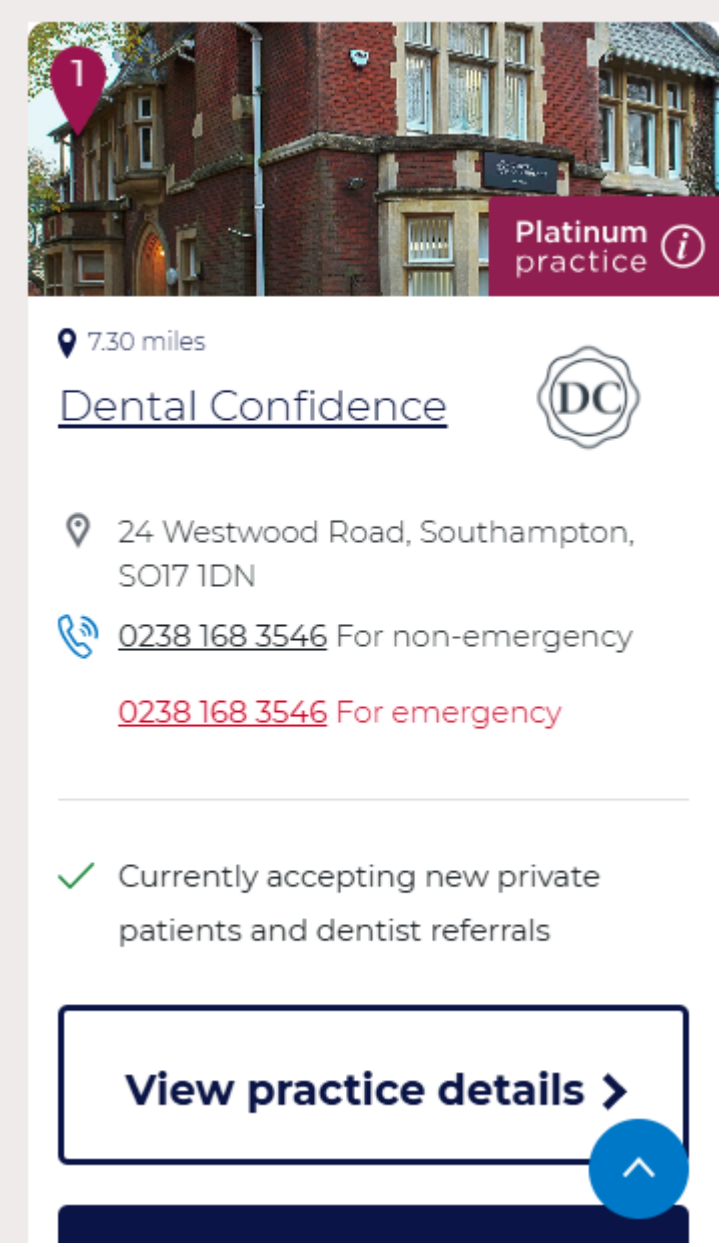


Figure K.2: Practice Search Page (Mobile).

K.2.3: Map View (Desktop)

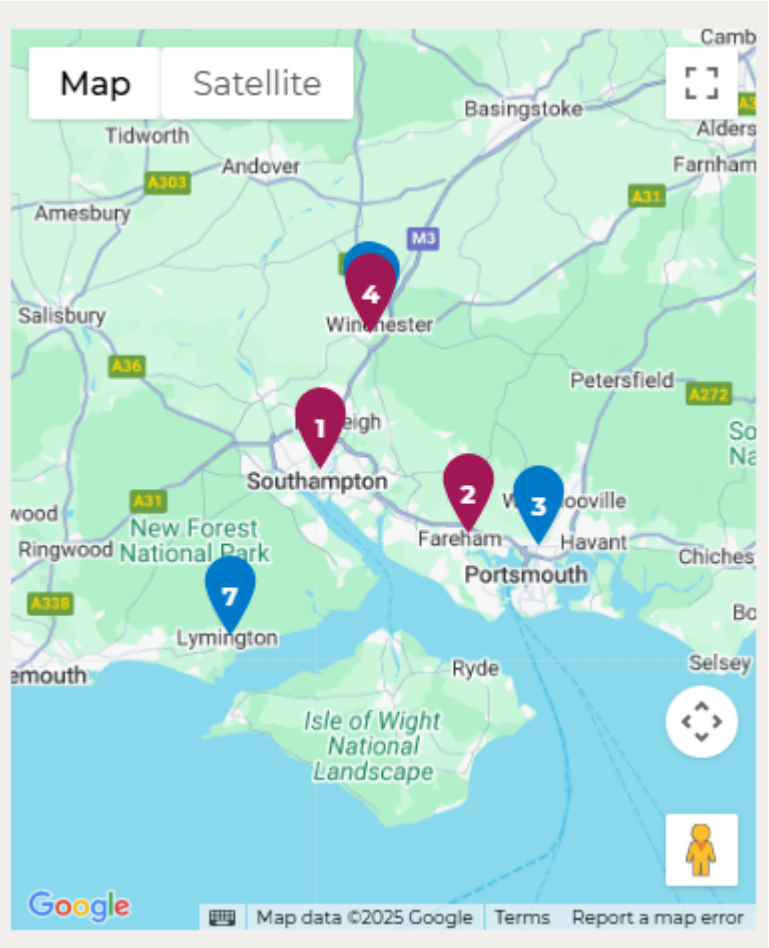


Figure K.3: Map View.

K.2.4: Screenshot of Bupa Dental Care Practice Details Page (Desktop)

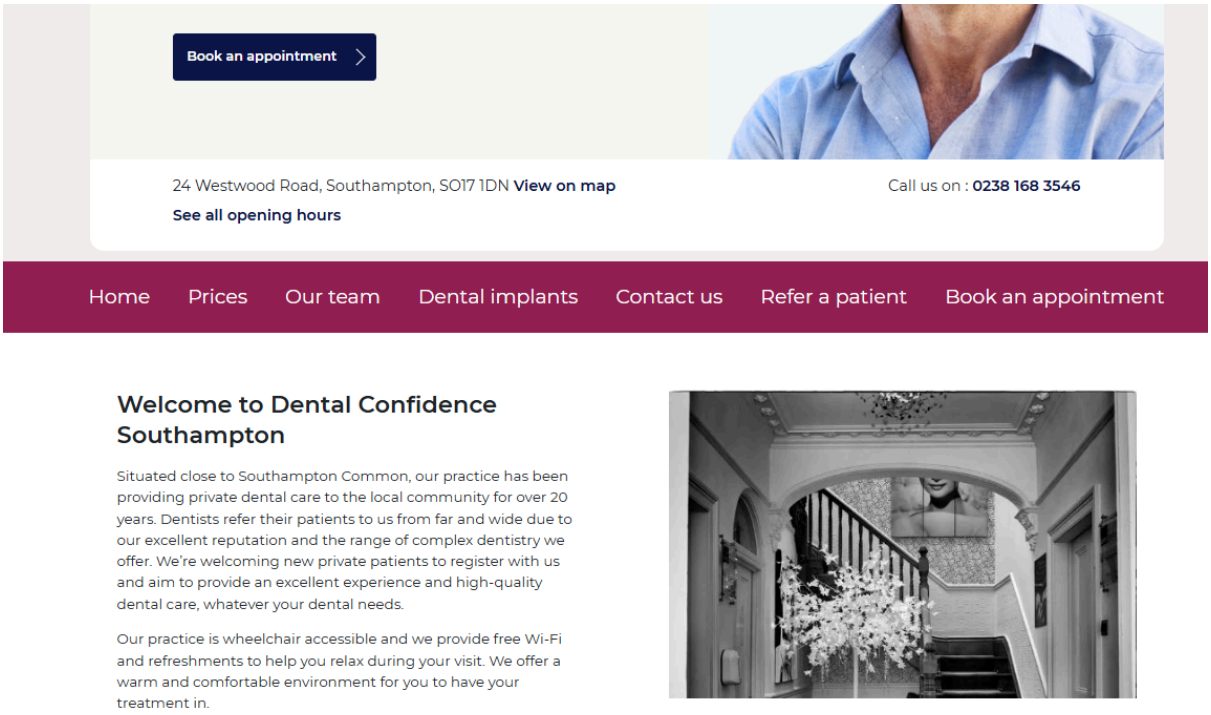


Figure K.4: Practice Details Page (Desktop).

K.2.5: Screenshot of Bupa Dental Care Practice Details Page (Mobile)

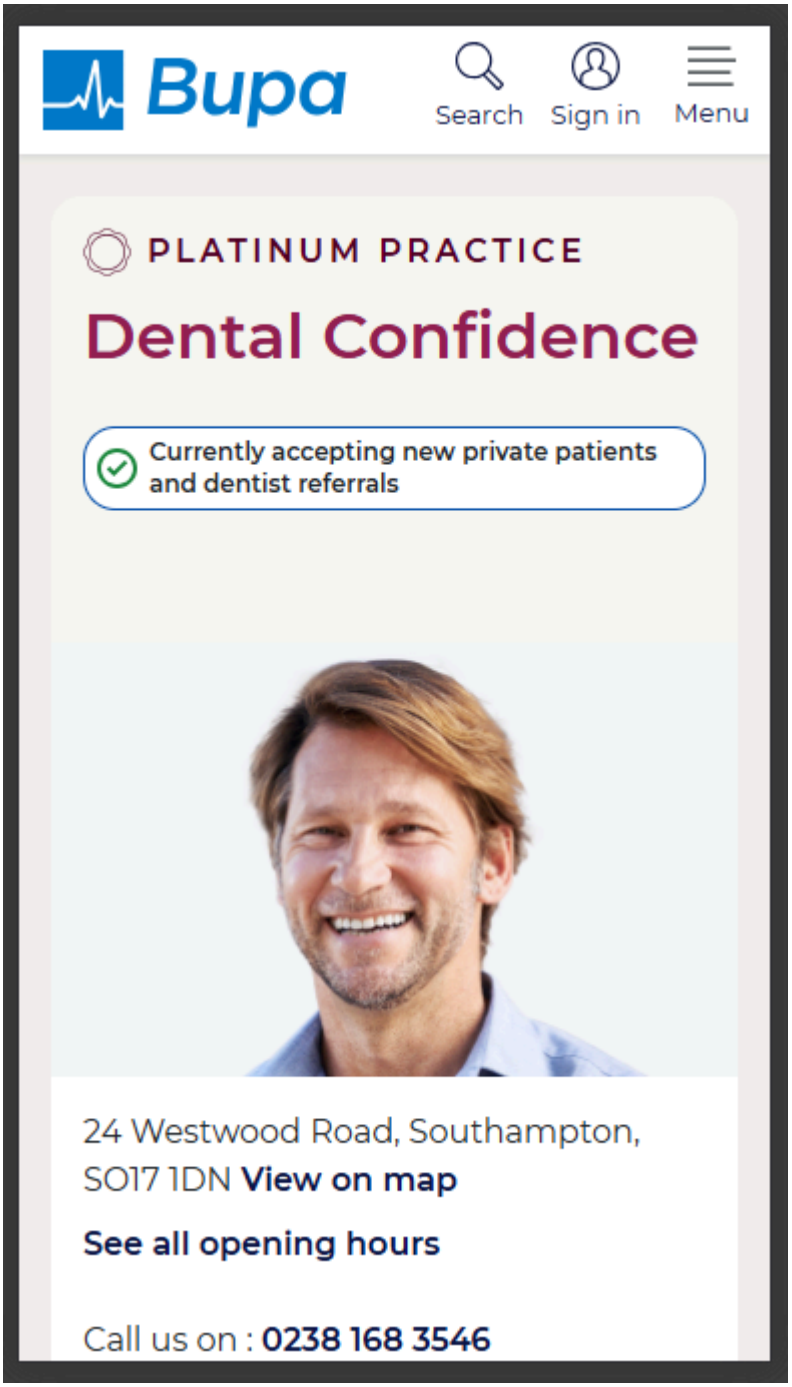


Figure K.5: Practice Details Page (Mobile).

K.2.6: Bupa Dental Care Google Lighthouse (Desktop)

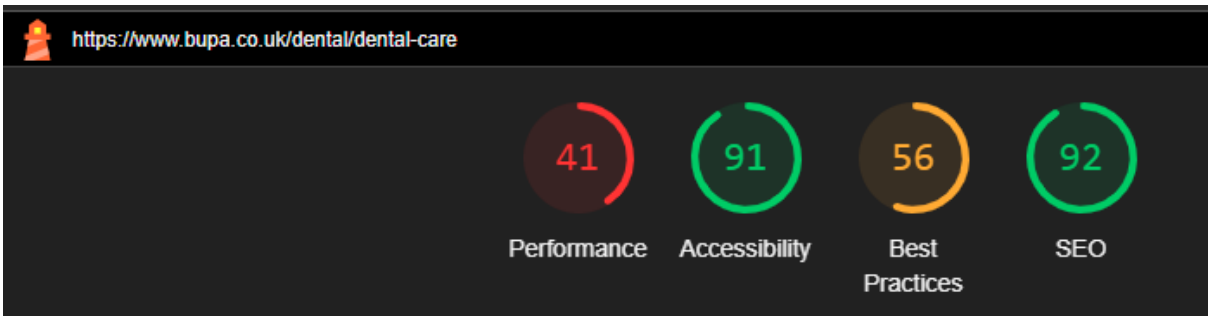


Figure K.6: Google Lighthouse Performance Test (Desktop).

K.2.7: Bupa Dental Care Google Lighthouse (Mobile)

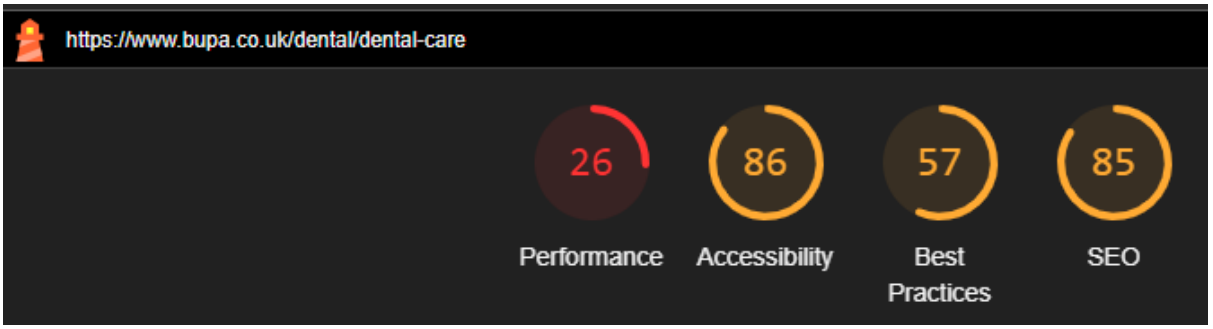


Figure K.7: Google Lighthouse Performance Test (Mobile).

K.2.3 Full {mydentist} Competitor Analysis

A web application created by the UK's largest dental provider, {my}dentist, with over 500 NHS and private dental practices ({my}dentist, 2025). It enables users to search for practices and treatments, overlapping the project's scope by facilitating the searching of practices and treatments. However, it does not allow a user to book a specific appointment through the web application. The {my}dentist web application also supports searching for orthodontists, however this is out of the project scope.

Category	Details
Key Features	**Searching for a practice**: The user is able to search for dental practices with the ability to filter by location, practice name and whether the practice is accepting new patients. Without filters applied, random practices are displayed in cards with an image of the practice, indicator of if it is NHS or private, practice name, location (postcode), contact details (phone number), opening hours and buttons for additional practice details and to arrange a consultation (Appendix K.2.4.1 and K.2.4.2). Upon entering location (postcode, town or city) the search results update to display practices sorted in ascending order by distance, and an additional field showing distance away in miles is added to the cards (Appendix K.2.4.3 and K.2.4.4). **Searching for a treatment**: The user is able to search for treatments by name, with a quick select for common treatments such as 'Routine dentistry' and 'Teeth whitening'. Upon searching for a treatment type, the web application displays the available types of services. For example 'Teeth whitening' results include multiple different variations and brands (Appendix K.2.4.5). The user is able to find out more details or enquire about booking each of the services by their respective cards within the search results (Appendix K.2.4.7 through K.2.4.9).
Strengths	**Searching for a practice**: ● The user is able to search for practices with relevant filters. ● The practice cards show relevant information about the practices. ● The practice cards are visually consistent and well structured, helping to provide a user-friendly browsing experience. **Searching for a treatment**: ● The cards are clear and concise.

	General: ● The web application is mobile responsive.
Weaknesses	**Searching for practice**: ● The filters are limited, when viewing further information for a given practice there are additional attributes such as a 1-5 stars rating from reviews. Being able to filter on this could be beneficial. **Searching for treatment**: ● There is poor error handling when searching for a treatment by name. If the search does not match the exact search term expected by the web application then no results are shown and there is a blank space with no error message (Appendix K.2.4.6). ● The user does not have the ability to directly book a treatment, the practice has to contact the user ● There is no price shown for each service. ● There is no indication of appointment availability. **General**: ● The web application doesn't support direct booking, the user enters contact information and the practice has to contact the user. ● The web application is not performant and is very slow to use. Verified via a Google Lighthouse test, results are available in Appendix K.2.4.10 through K.2.4.13.
Influence on the project	● **FR6**: The search functionality of the web application must support relevant filters, including distance. ● **FR6**, **FR15**: The user should be able to enter a postcode or have their location automatically detected. ● **FR6**: Upon a user entering location information the cards will update to show the distance to the practice. ● **FR6**: The cards will display relevant information about the appointment and practices. ● **NFR1**: The web application and cards must be mobile responsive. ● **NFR3**: The web application must be performant.

Table K.4: {my}dentist Competitor Analysis.

K.2.4 {my}dentist Competitor Analysis Supporting Screenshots

K.2.4.1: Screenshot of {my}dentist Practice Search Page (Desktop Without Filters)

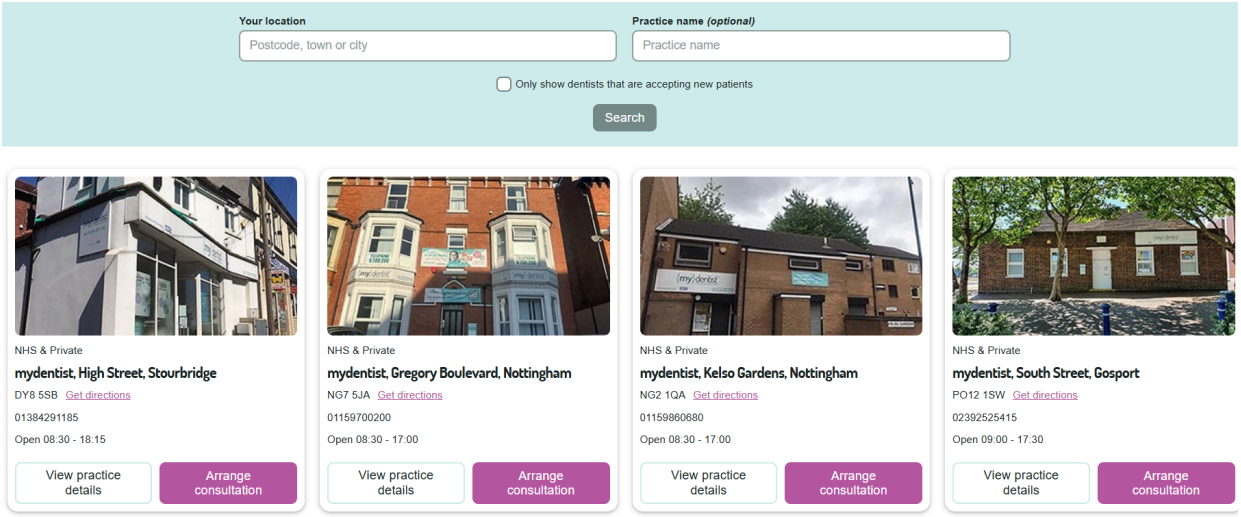


Figure K.8: Search Page without filters applied (Desktop).

K.2.4.2: Screenshot of {my}dentist Practice Search Page (Mobile Without Filters)

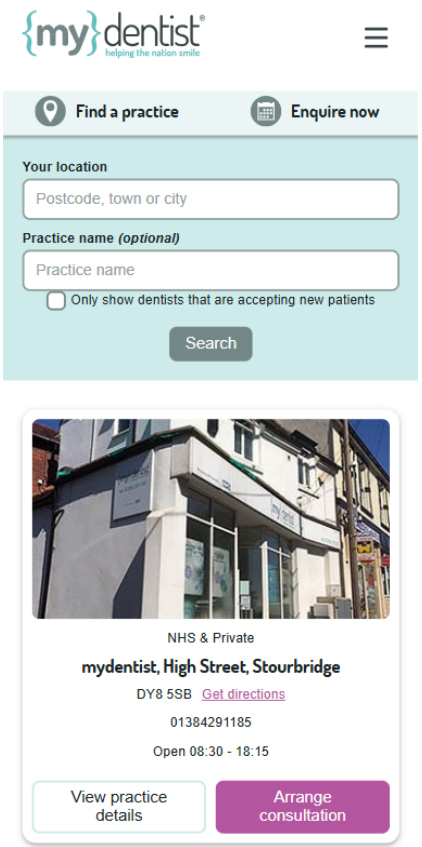


Figure K.9: Search Page without filters applied (Mobile).

K.2.4.3: Screenshot of {my}dentist Practice Search Page (Desktop With Location Filter Applied)

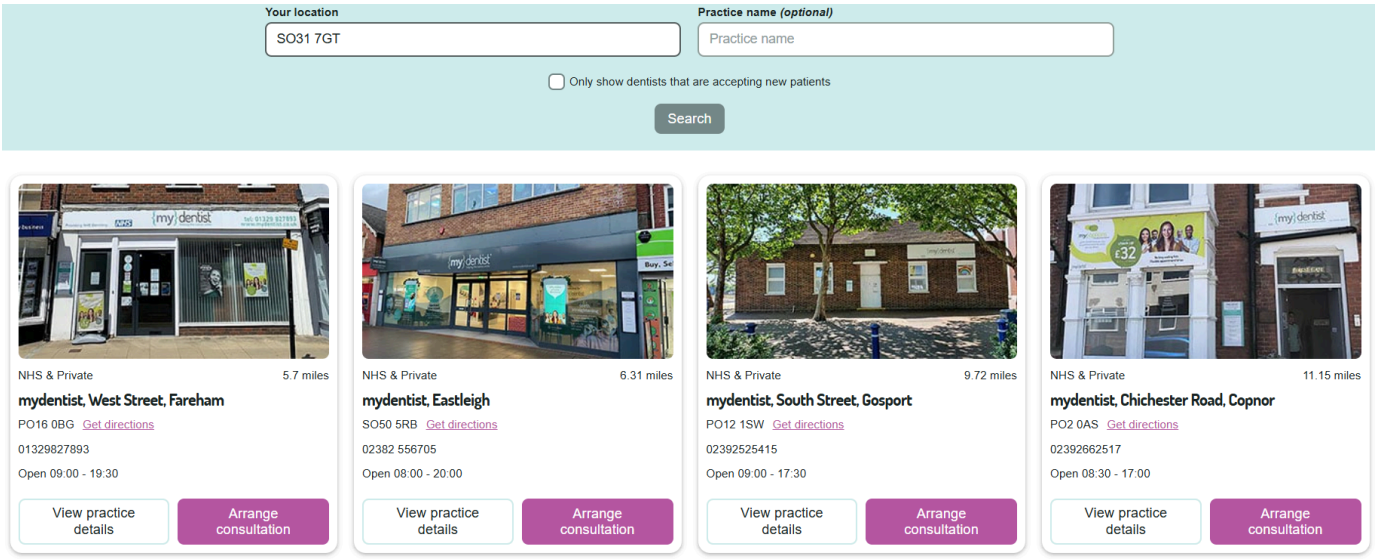


Figure K.10: Search page with location applied (Desktop).

K.2.4.4: Screenshot of {my}dentist Practice Search Page (Mobile With Location Filter Applied)

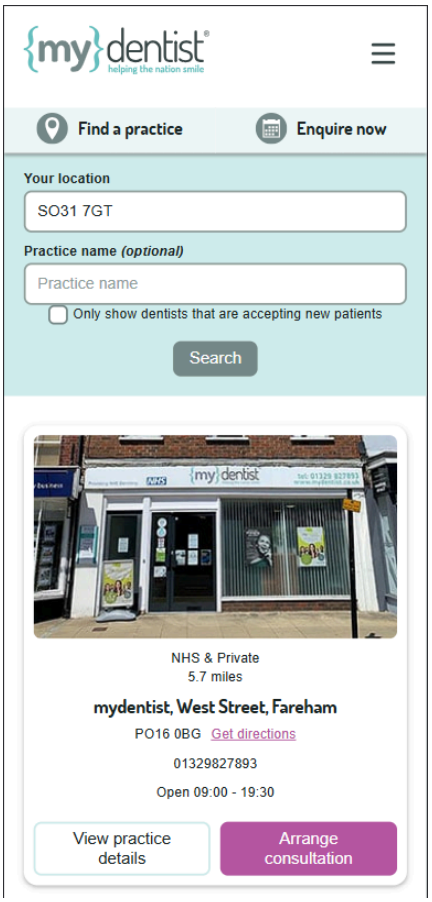


Figure K.11: Search page with location applied (Mobile).

K.2.4.5: Screenshot of {my}dentist Treatment Search Results for ‘Teeth Whitening’

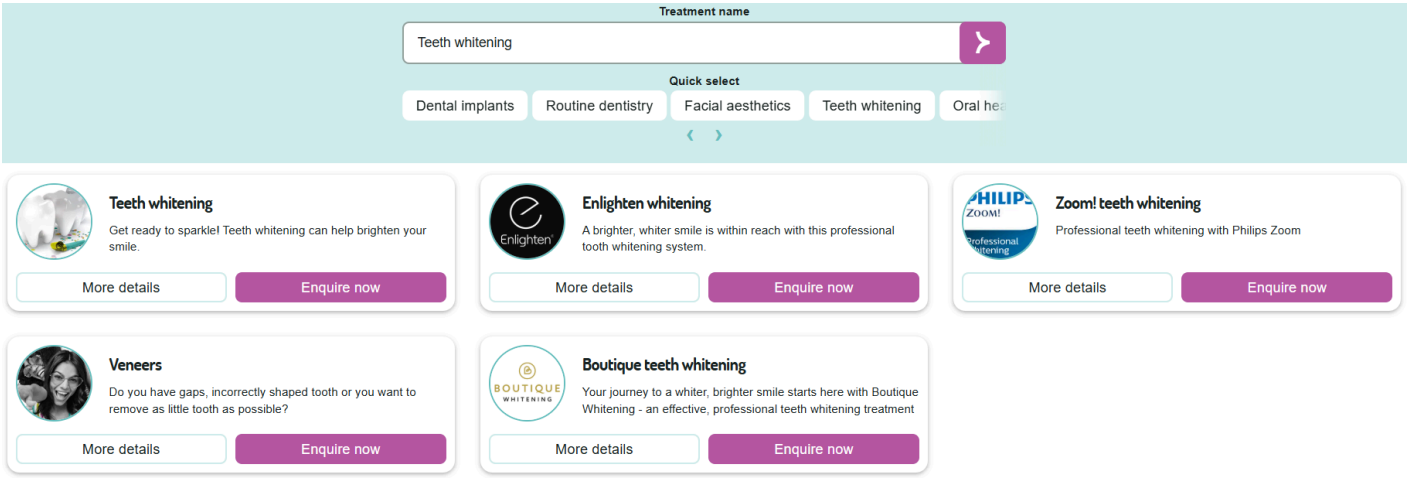


Figure K.12: Treatment page search results for ‘Teeth Whitening’.

K.2.4.6: Screenshot of {my}dentist Treatment Search Results Error Handling

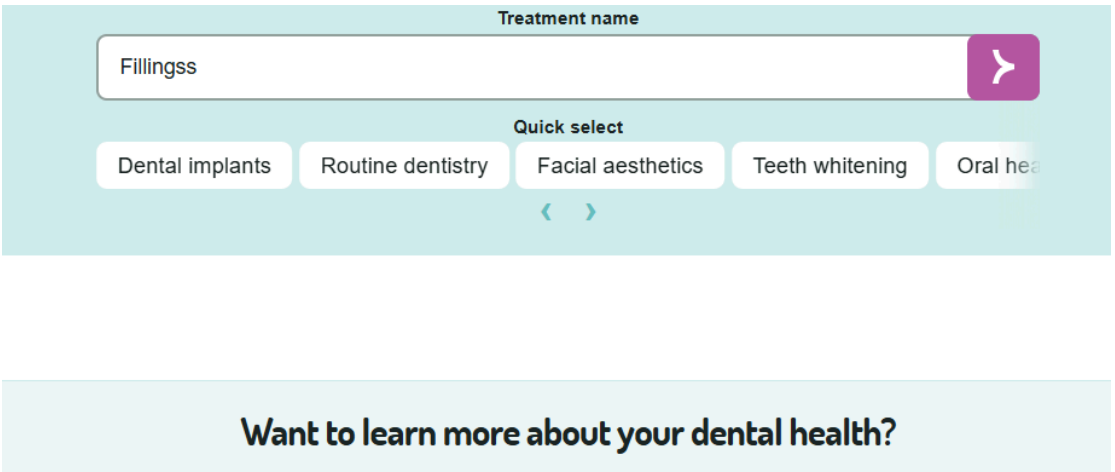


Figure K.13: Treatment page with erroring search term.

K.2.4.7: Treatment Search Result Card

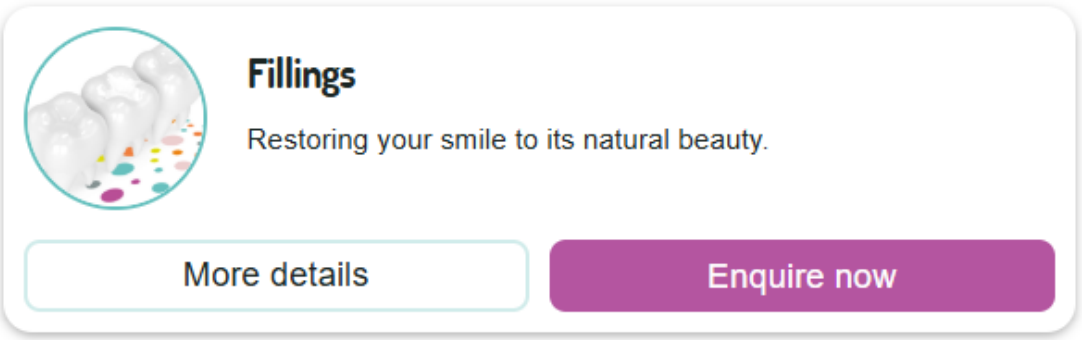


Figure K.14: Treatment search result card.

K.2.4.8: Treatment Search Results More Details

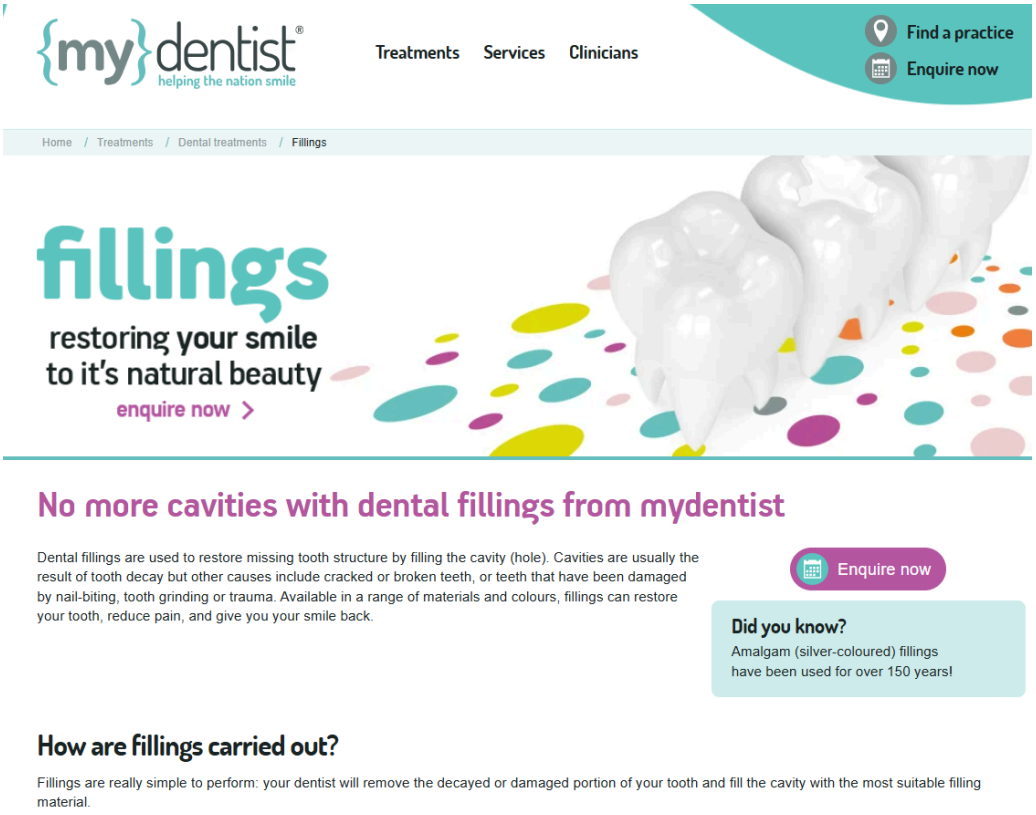


Figure K.15: Treatment search results more details page.

K.2.4.9: Treatment Search Results Enquire Now

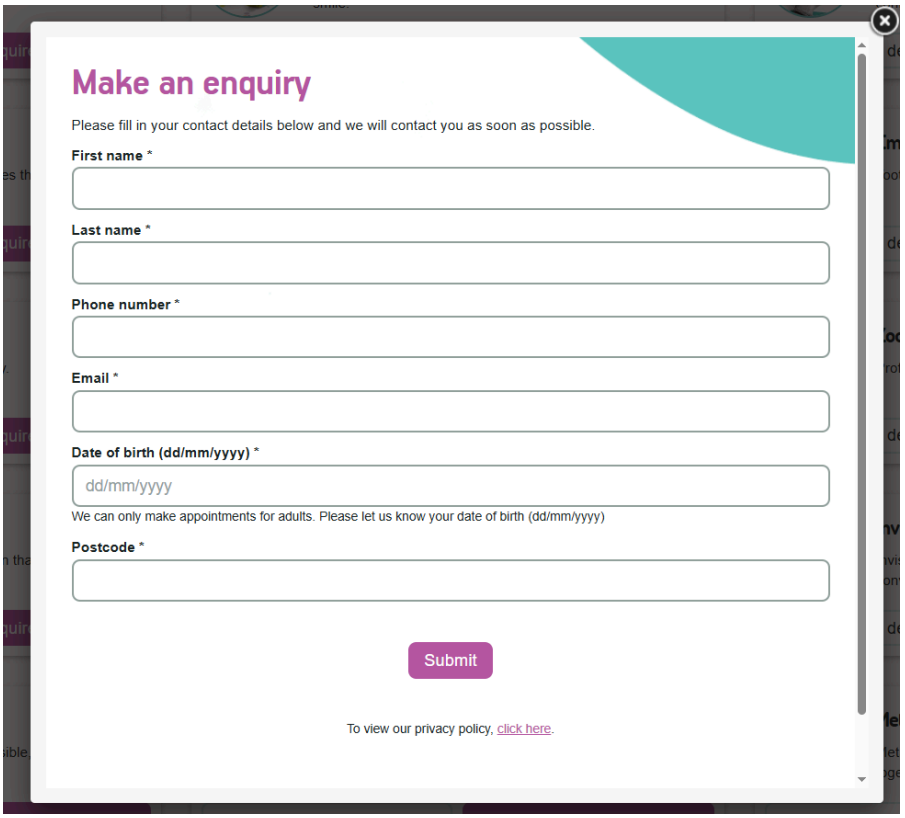


Figure K.16: Treatment search results enquire now pop-up.

K.2.4.10: Google Lighthouse Test Results for {my}dentist (Desktop Homepage)

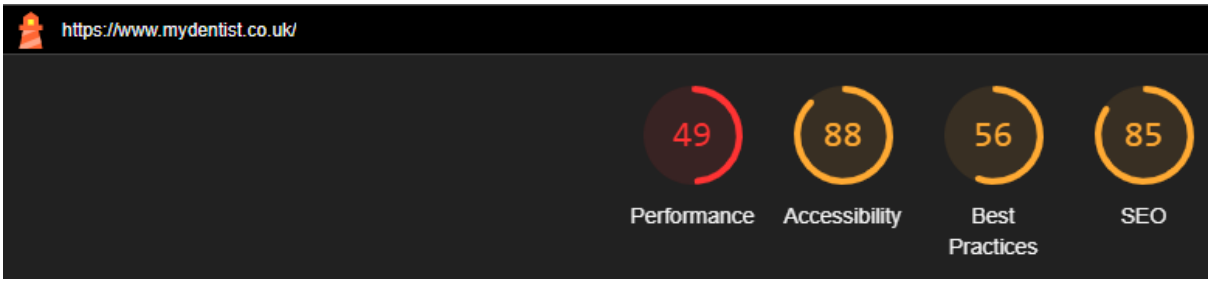


Figure K.17: Google Lighthouse results {my}dentist homepage (Desktop).

K.2.4.11: Google Lighthouse Test Results for {my}dentist (Mobile Homepage)

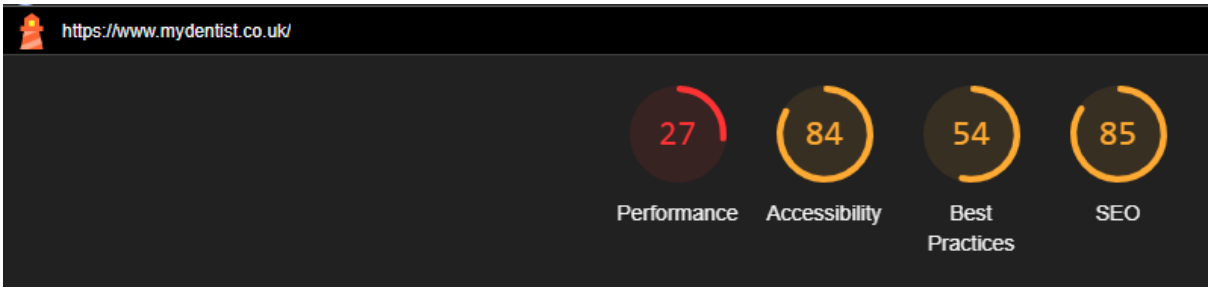


Figure K.18: Google Lighthouse results {my}dentist homepage (Mobile).

K.2.4.12: Google Lighthouse Test Results for {my}dentist (Desktop Practice Information)

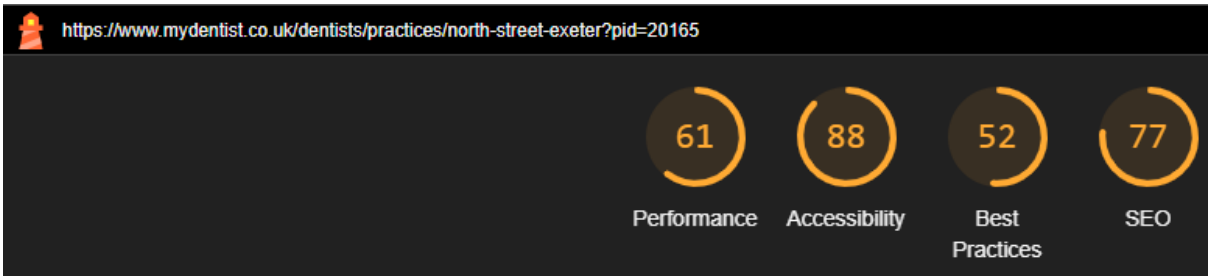


Figure K.19: Google Lighthouse results {my}dentist practice information page (Desktop).

K.2.4.13: Google Lighthouse Test Results for {my}dentist (Mobile Practice Information)

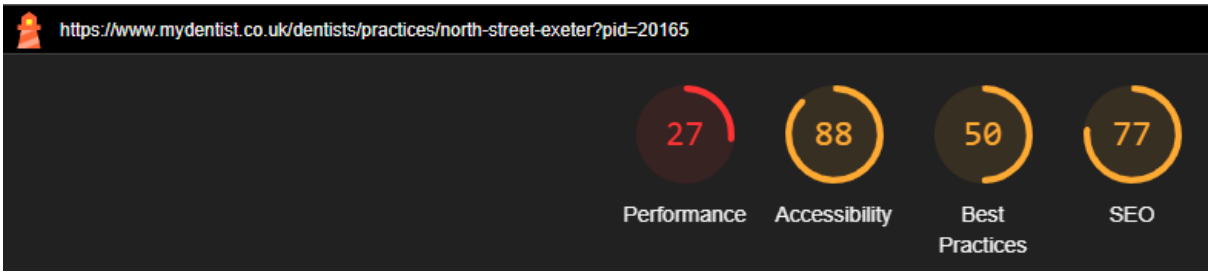


Figure K.20: Google Lighthouse results {my}dentist practice information page (Mobile).

K.3 Preliminary User Research

This online questionnaire was conducted to validate the proposed web application and confirm assumptions about user demand, behaviours and preferences when booking dental appointments. The information sheet and consent forms are available here:

https://github.com/danbennett239/CI601/tree/main/Questionnaire

K.3.1 Preliminary User Research Questionnaire

Have you ever needed to book a dental appointment at short notice (e.g., within a few days due to a cancellation or urgent need)?

- ☐ Yes, often
- ☐ Yes, occasionally
- ☐ No

If yes, what challenges did you face when trying to find a short-notice appointment?

Would you use a service that aggregates dental appointments from multiple practices, allowing you to compare prices and availability upfront?

- ☐ Yes
- ☐ Maybe, depending on the features
- ☐ No

What factors are most important to you when booking a dental appointment?
(Rank the following from 1 = Most Important to 5 = Least Important)

- ☐ Price
- ☐ Distance to the practice
- ☐ Appointment timing (e.g., within a few days, specific times like evenings)
- ☐ Dentist rating/reputation
- ☐ Availability of specific treatments (e.g., cleaning, whitening)

How far would you be willing to travel for a dental appointment?

- ☐ Less than 5 miles
- ☐ 5–10 miles
- ☐ 10–20 miles
- ☐ 20+ miles

Would you find it useful to receive notifications about newly available appointments that match your preferences (e.g., treatment type, distance)?

- ☐ Yes
- ☐ No

What would be your preferred way to filter and sort available appointments?
(Select all that apply)

- ☐ By price
- ☐ By distance
- ☐ By appointment time
- ☐ By dentist rating
- ☐ By treatment type

Would you prefer to use this service on a mobile device, desktop, or both?

- ☐ Mobile device
- ☐ Desktop/laptop
- ☐ Both

What additional features would make this service more valuable to you?

Do you have any concerns about using a service that aggregates dental appointments from multiple practices?

K.3.2 Preliminary User Research Responses

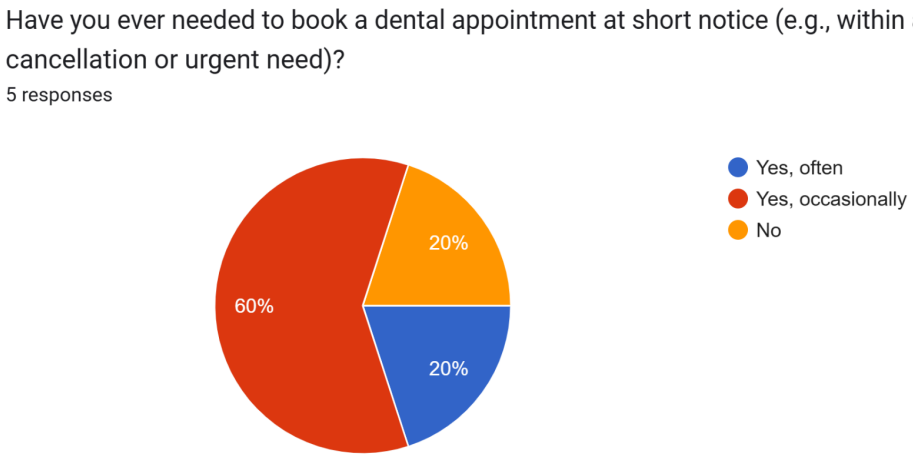


Figure K.21: Preliminary questionnaire question 1 responses pie chart.

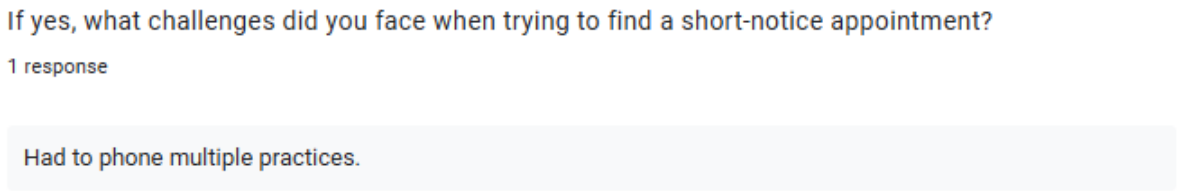


Figure K.22: Preliminary questionnaire question 2 responses.

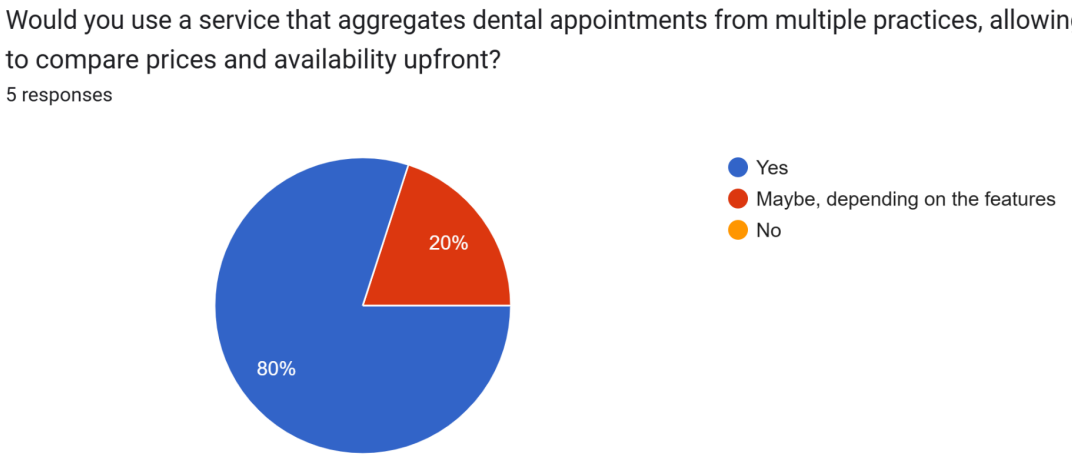


Figure K.23: Preliminary questionnaire question 3 responses pie chart.

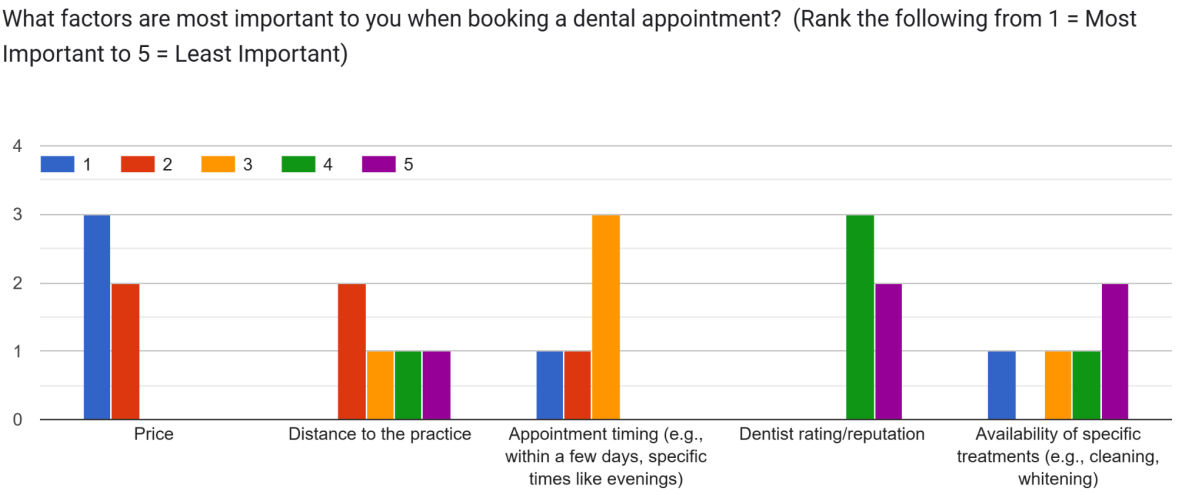


Figure K.24: Preliminary questionnaire question 4 responses bar graph.

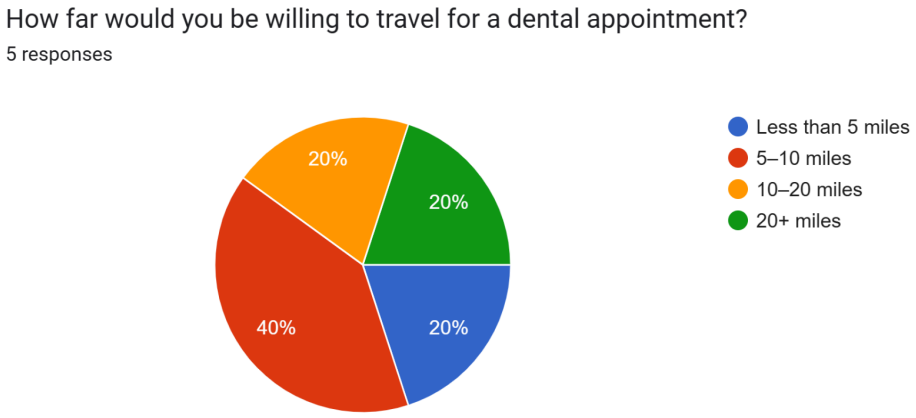


Figure K.25: Preliminary questionnaire question 5 responses pie chart.

Would you find it useful to receive notifications about newly available appointments that match your preferences (e.g., treatment type, distance)?
5 responses

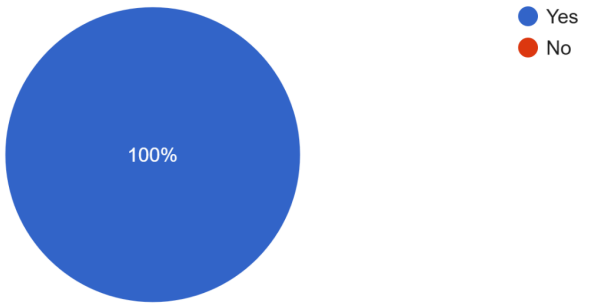


Figure K.26: Preliminary questionnaire question 6 responses pie chart.

What would be your preferred way to filter and sort available appointments? (Select all that apply)
5 responses

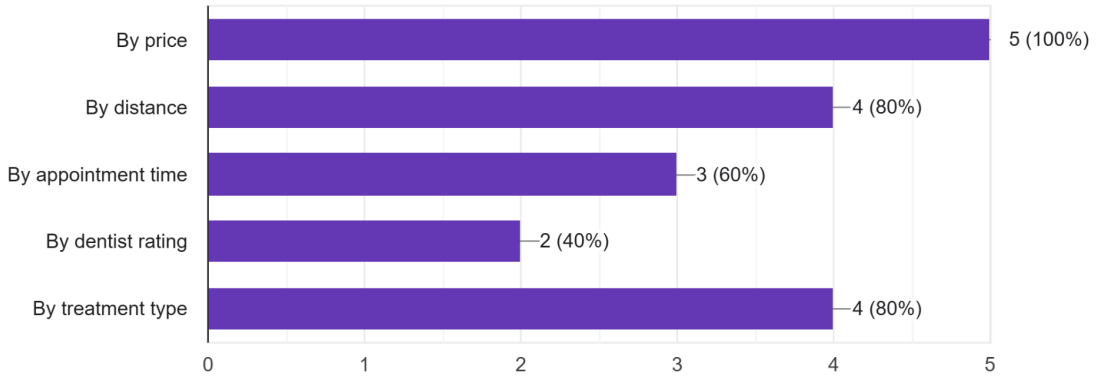


Figure K.27: Preliminary questionnaire question 7 responses bar graph.

Would you prefer to use this service on a mobile device, desktop, or both?
5 responses

Figure K.28: Preliminary questionnaire question 8 responses pie chart.

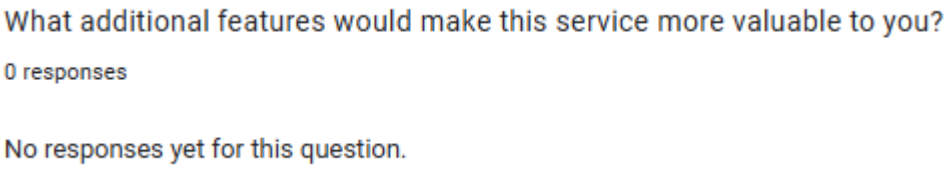


Figure K.29: Preliminary questionnaire question 9 responses.

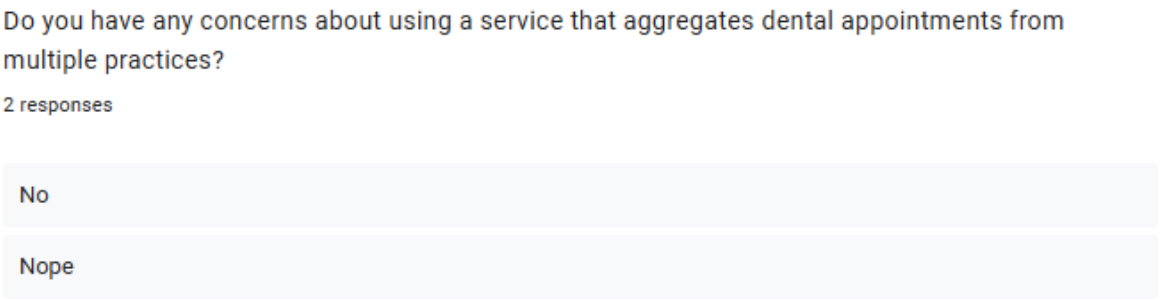


Figure K.30: Preliminary questionnaire question 10 responses.

K.3.3 Preliminary User Research Outcome

The preliminary user research questionnaire revealed that 80% of respondents had previously needed a short-notice dental appointment, with all showing interest in an aggregated booking web application. Price was the most important booking factor, with distance and appointment type being the second most important. All respondents wanted to be able to access the web application on desktop and mobile devices. No concerns were raised about using an aggregated platform.

This provides validation for the project's web application. It informed development for both mobile and desktop devices, the filters user's desire and features, including notifications for new appointments (FR12).

K.4 Design Questionnaire

This short design questionnaire was conducted to refine and inform specific design choices within the project. It focused on user preferences of layout and color scheme. The information sheet and consent forms are available here:

https://github.com/danbennett239/CI601/tree/main/Questionnaire

K.4.1 Design Questionnaire

Question 1: Sign In and Registration
Which user interface design do you prefer for Signing In and Registration?

- Option 1: Separate Pages:
    - Sign-In and Registration are on separate pages, linked by buttons (e.g., “Don’t have an account? Register” or “Already have an account? Sign In”) at the bottom of each modal. Users click “Sign In” in the navigation banner to access the Sign-In page, and can navigate to the Registration page from there.

- Option 2: Single Page with Divider:
    - Sign-In and Registration are displayed on the same page, separated by a visual divider (e.g., left and right sections). The page is accessed via the “Sign In” button in the navigation banner, and users can choose to sign in or register immediately.

Question 2: Web Application Colour Scheme
Which colour scheme do you prefer for a dental appointment booking web application?

- Blue and White
- Green and Tan

K.4.2 Design Questionnaire Responses

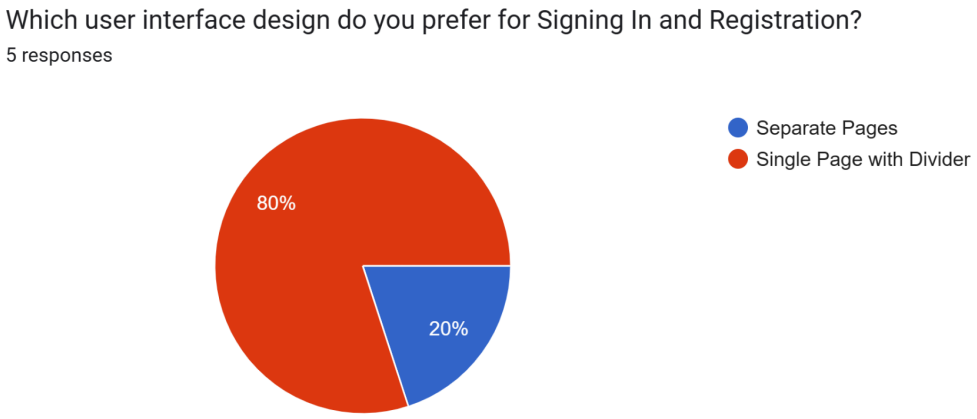


Figure K.31: Design questionnaire question 1 responses pie chart.

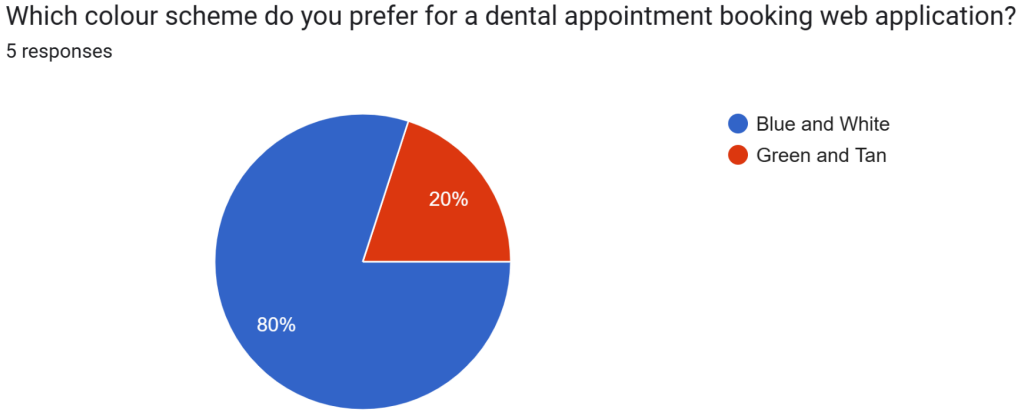


Figure K.32: Design questionnaire question 2 responses pie chart.

K.4.3 Design Questionnaire Outcome

The design questionnaire directly informed the sign in and registration page design. This is reflected within the UI wireframe in Appendix D.4. Additionally it informed the colour scheme of the web application being blue and white.

K.5 Usability Questionnaire

A usability questionnaire was conducted to validate that each of the following nine user journeys (Defined in Appendix J.1) within the web application was clear, effective and user-friendly:

- User registration.
- User sign-in.
- Managing user details and preferences.
- Searching for an appointment.
- Booking an appointment.
- Practice registration.
- Practice sign-in.
- Managing practice details and preferences.
- Managing appointments.

The questionnaire aimed to identify any usability issues that could impact the user experience or prevent successful task completion.

Participants were asked to complete each of the user journeys, rating each on effectiveness, efficiency, and satisfaction, using a Likert scale, with an additional optional free-text field. This aligns with the ISO 9241-11 standard (International Organisation for Standardisation, 2018). The full questionnaire and responses are detailed in Appendix K.2.2 and Appendix K.2.3.

The outcome of the questionnaire is in Section 6.3.4.

The information sheet and consent forms are available here:

https://github.com/danbennett239/CI601/tree/main/Questionnaire

K.2.1 Usability Questionnaire Full Questionset

CI601 Usability Questionnaire
Please complete the following tasks in the web application. After each task, answer the short set of questions to reflect your experience.

For each question, use the scale:
1 - Strongly Disagree
2 - Disagree
3 - Neutral
4 - Agree
5 - Strongly Agree

You may also provide optional comments after each section.

1. User Registration
Task: Create a new user account on the system using your email and a secure password.

- The registration process was easy to complete. (1–5)
- The instructions and fields were clear. (1–5)
- I am satisfied with the overall registration experience. (1–5)
- Optional comments (open response): _____

2. User Sign-In
Task: Sign in using the account you just created.

- Signing in was straightforward. (1–5)
- The sign-in process was clear and understandable. (1–5)
- I am satisfied with the sign-in experience. (1–5)
- Optional comments (open response): _____

3. Managing User Details and Preferences
Task: Update your contact information and notification preferences.

- It was easy to find and update my user details. (1–5)
- The available settings were clear and understandable. (1–5)
- I am satisfied with how I manage my user preferences. (1–5)
- Optional comments (open response): _____

4. Searching for an Appointment
Task: Use the appointment search page to find an available appointment near you.

- Searching for an appointment was intuitive. (1–5)
- The filters and sorting options were clear. (1–5)
- I am satisfied with the appointment search experience. (1–5)
- Optional comments (open response): _____

5. Booking an Appointment
Task: Book an appointment from your search results.

- The booking process was easy to follow. (1–5)
- The information shown during booking was clear. (1–5)
- I am satisfied with the appointment booking process. (1–5)
- Optional comments (open response): _____

6. Practice Registration
Task: Register as a dental practice.

- The practice registration process was simple. (1–5)
- The information required was clear and appropriate. (1–5)
- I am satisfied with the process of registering a practice. (1–5)
- Optional comments (open response): _____

7. Practice Sign-In
Task: Sign in as the dental practice you just created.

- Signing in was straightforward. (1–5)
- The sign-in process was clear and understandable. (1–5)
- I am satisfied with the sign-in experience. (1–5)
- Optional comments (open response): _____

8. Managing Practice Details and Preferences
Task: Update your practice's contact info and appointment types.

- It was easy to find and update practice details and preferences. (1–5)
- The detail and preference options were understandable. (1–5)
- I am satisfied with how practice details and preferences are managed. (1–5)
- Optional comments (open response): _____

9. Managing Appointments
Task: Create, edit, and delete an appointment.

- Managing appointments was intuitive. (1–5)
- The system made it clear how to edit or delete an appointment. (1–5)
- I am satisfied with the appointment management features. (1–5)
- Optional comments (open response): _____

K.2.2 Usability Questionnaire Responses

Each of the following pie-charts represent the answers given by the 5 participants of the usability questionnaire detailed in Appendix K.2.1.

K.2.3.1 User Registration

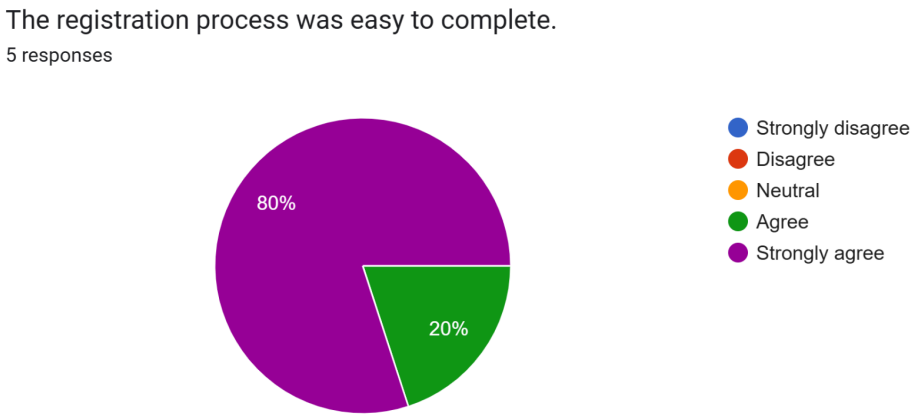


Figure K.33: Usability questionnaire user registration question 1 responses pie chart.

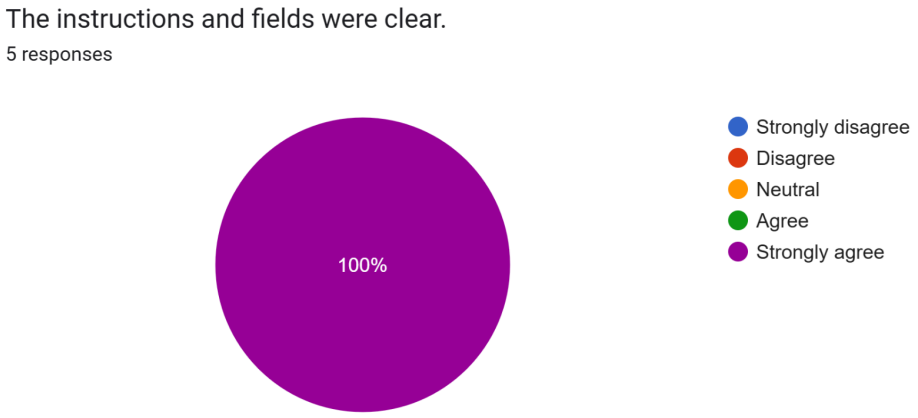


Figure K.34: Usability questionnaire user registration question 2 responses pie chart.

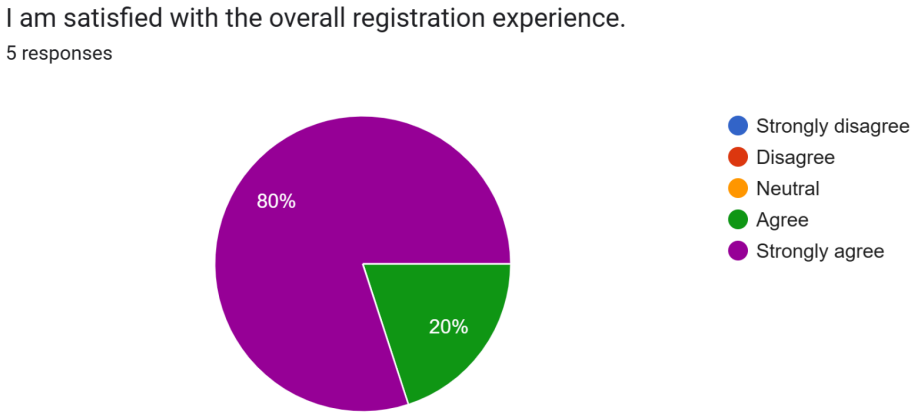


Figure K.35: Usability questionnaire user registration question 3 responses pie chart.

K.2.3.2 User Sign-In

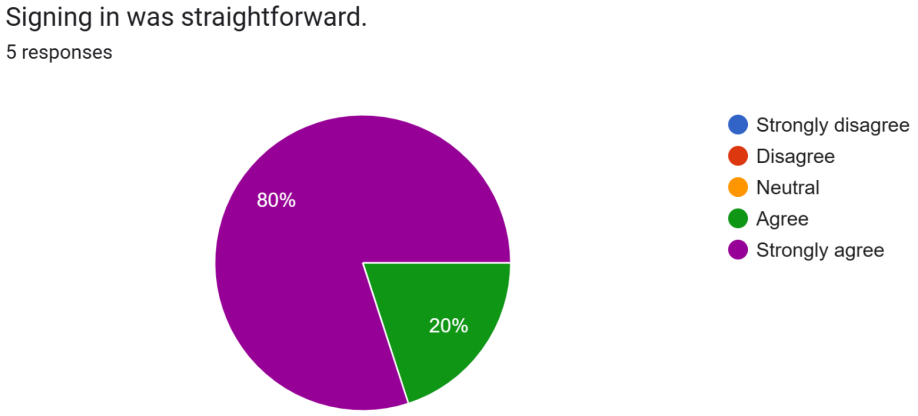


Figure K.36: Usability questionnaire user sign-in question 1 responses pie chart.

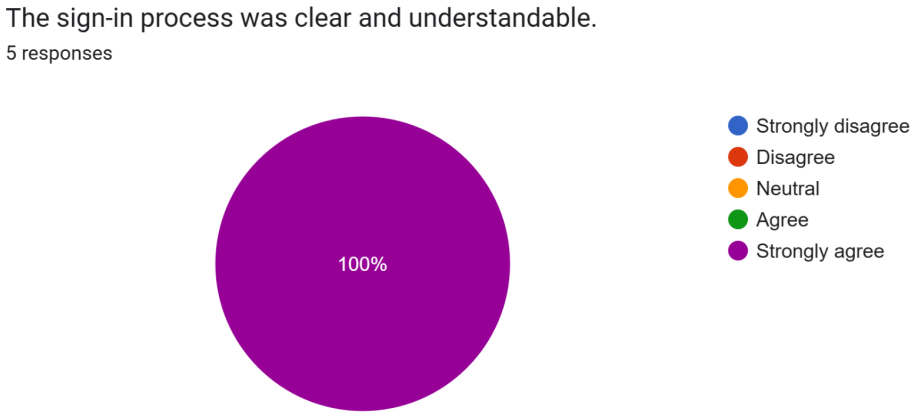


Figure K.37: Usability questionnaire user sign-in question 2 responses pie chart.

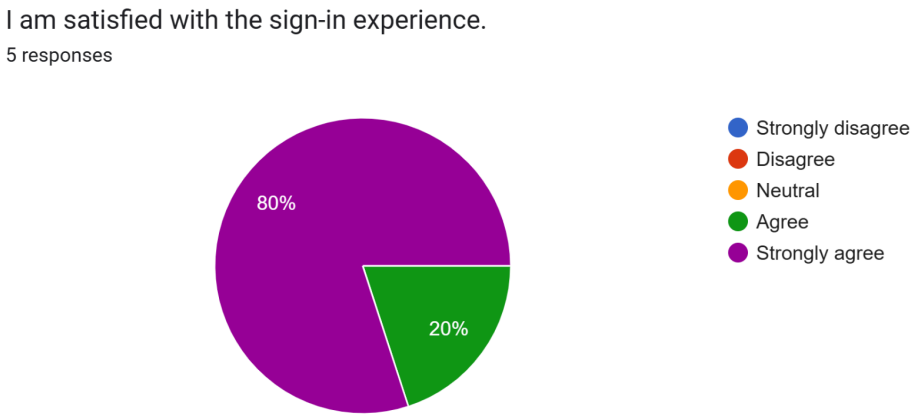


Figure K.38: Usability questionnaire user sign-in question 3 responses pie chart.

K.2.3.3 Managing User Details and Preferences

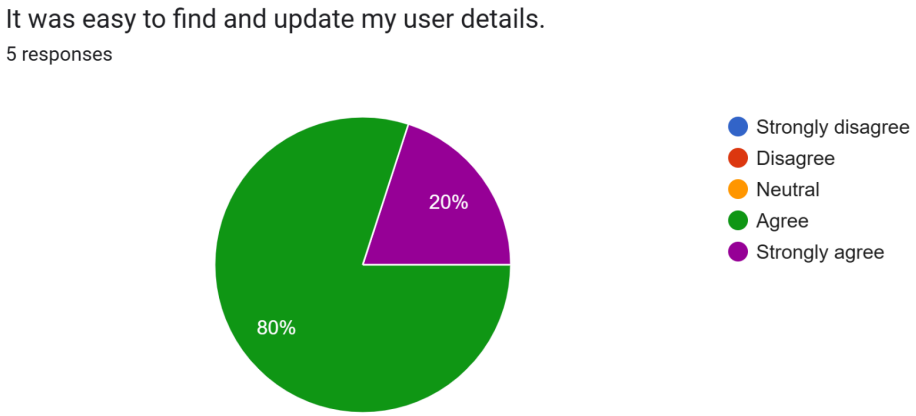


Figure K.39: Usability questionnaire managing user details and preferences question 1 responses pie chart.

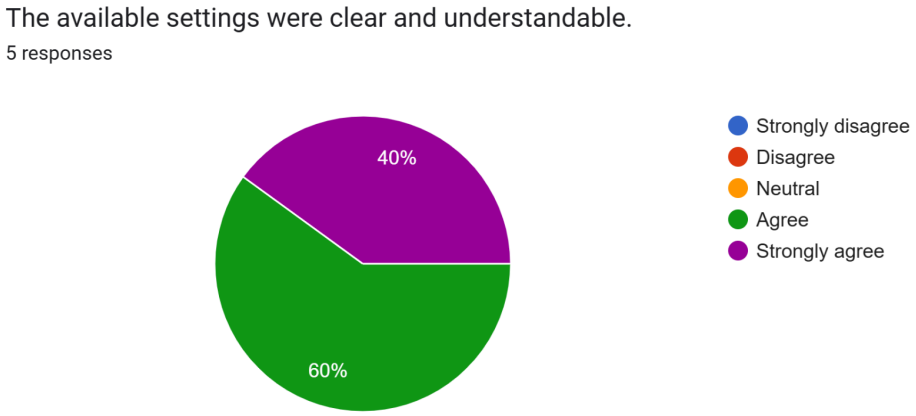


Figure K.40: Usability questionnaire managing user details and preferences question 2 responses pie chart.

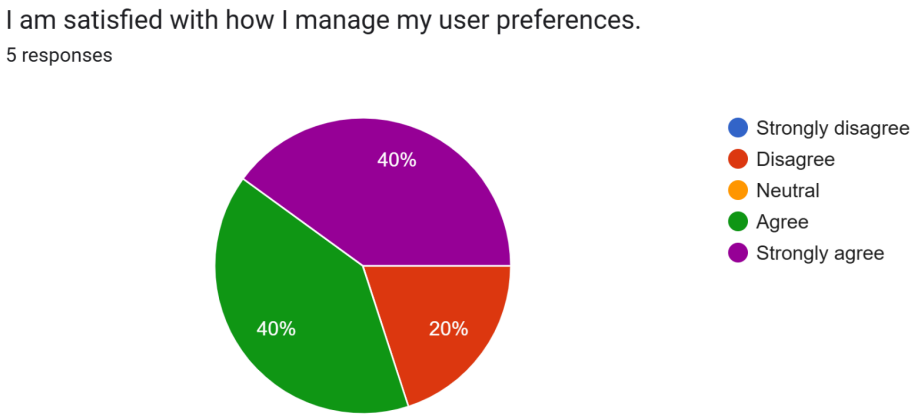


Figure K.41: Usability questionnaire managing user details and preferences question 3 responses pie chart.

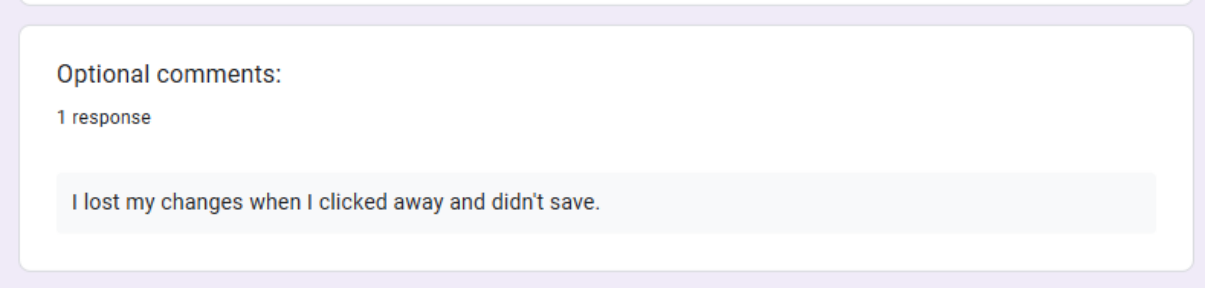


Figure K.42: Usability questionnaire managing user details and preferences optional comments responses.

K.2.3.4 Searching for an Appointment

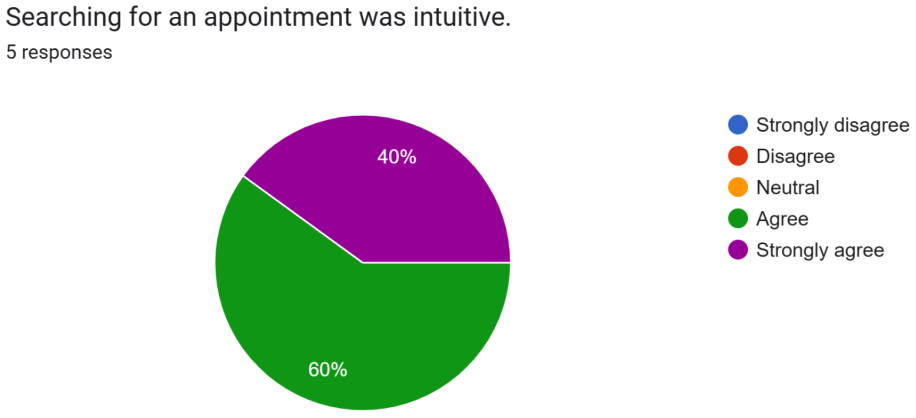


Figure K.43: Usability questionnaire searching for an appointment question 1 responses pie chart.

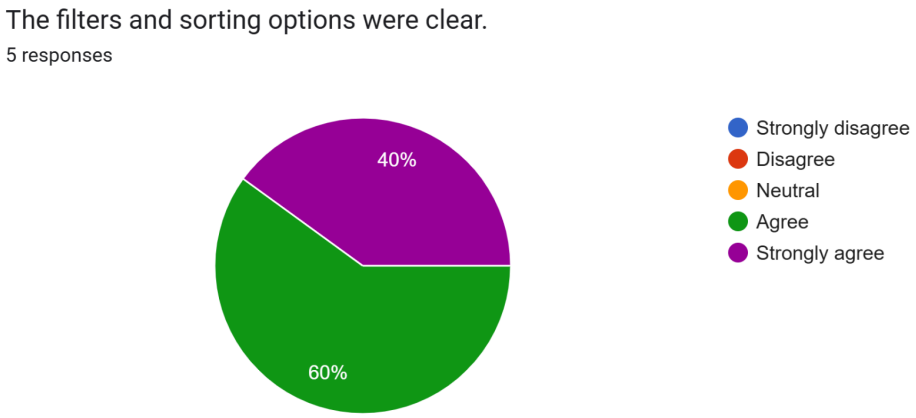


Figure K.44: Usability questionnaire searching for an appointment question 2 responses pie chart.

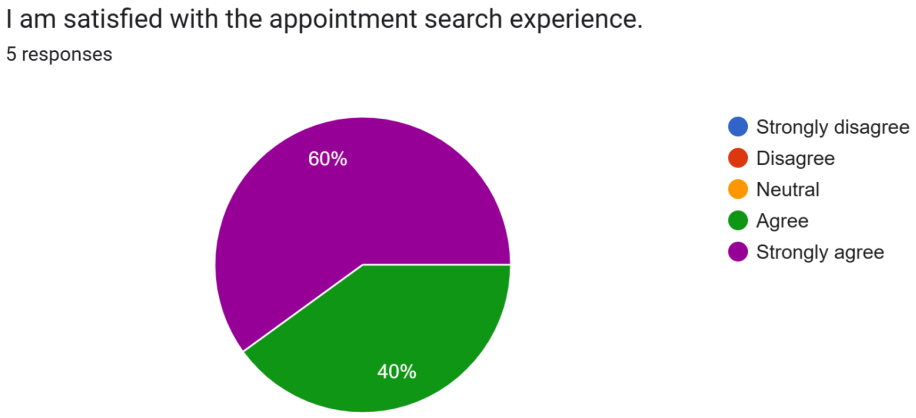


Figure K.44: Usability questionnaire searching for an appointment question 3 responses pie chart.

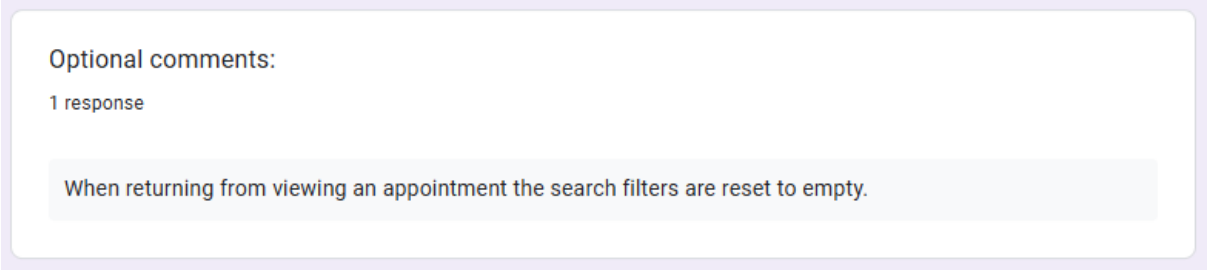


Figure K.45: Usability questionnaire searching for an appointment optional comments responses.

K.2.3.5 Booking an Appointment

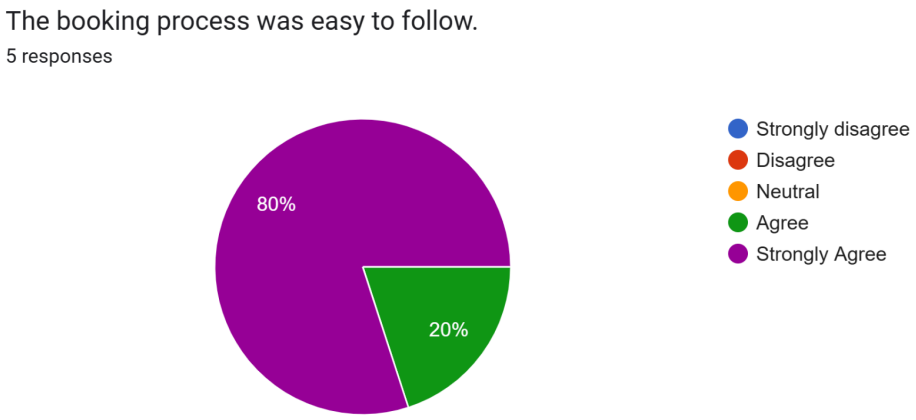


Figure K.46: Usability questionnaire booking an appointment question 1 responses pie chart.

Figure K.47: Usability questionnaire booking an appointment question 2 responses pie chart.

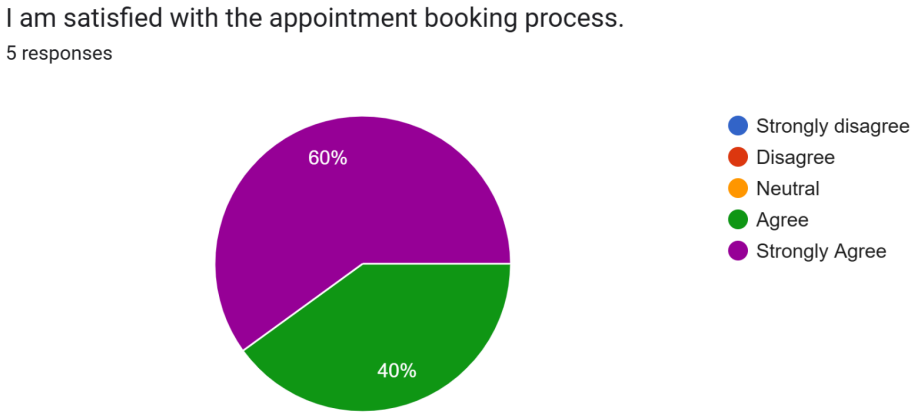


Figure K.48: Usability questionnaire booking an appointment question 3 responses pie chart.

K.2.3.6 Practice Registration

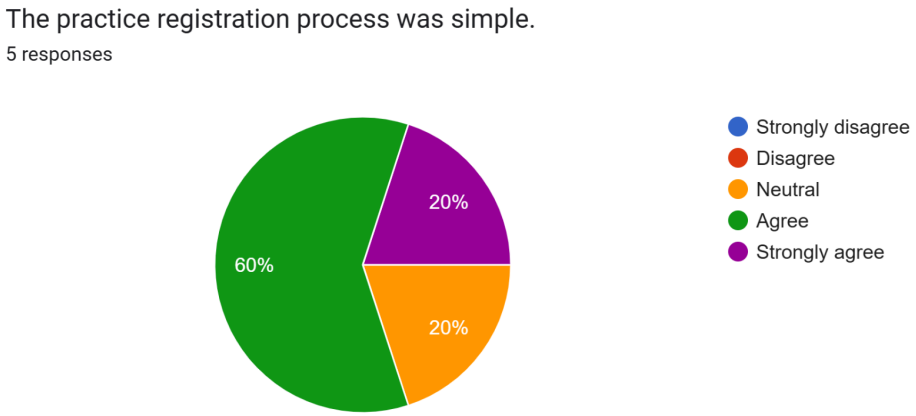


Figure K.49: Usability questionnaire practice registration question 1 responses pie chart.

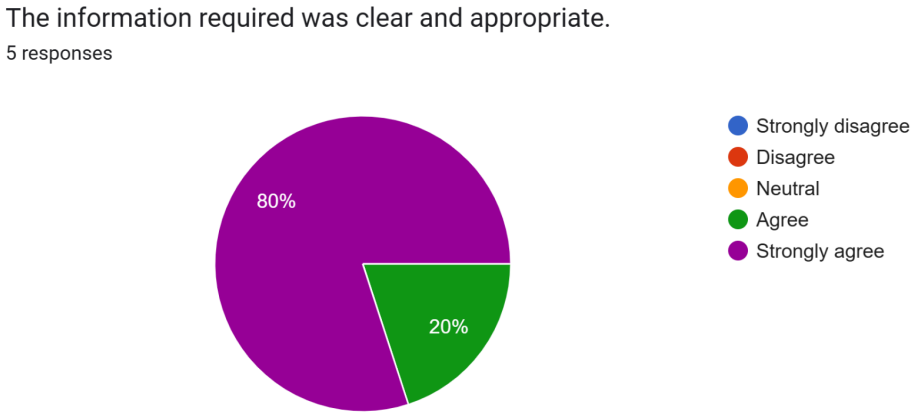


Figure K.50: Usability questionnaire practice registration question 2 responses pie chart.

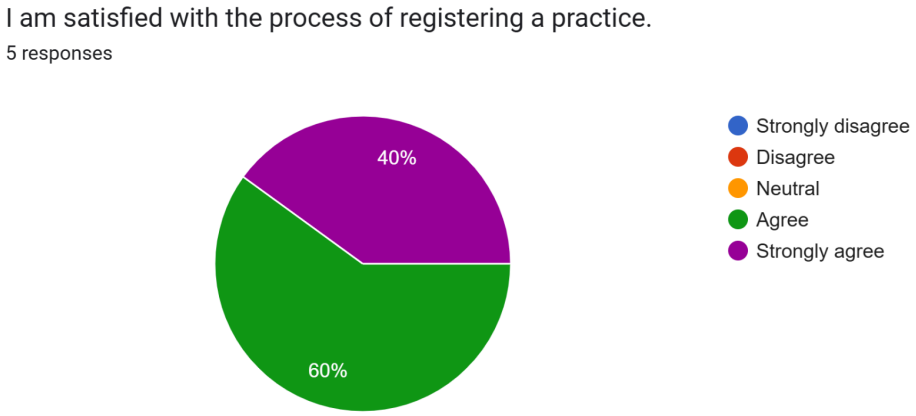


Figure K.51: Usability questionnaire practice registration question 3 responses pie chart.

K.2.3.7 Practice Sign-In

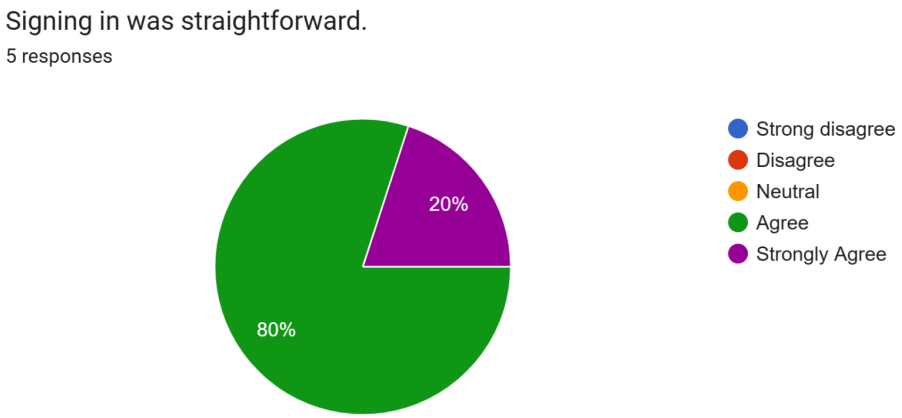


Figure K.52: Usability questionnaire practice sign-in question 1 responses pie chart.

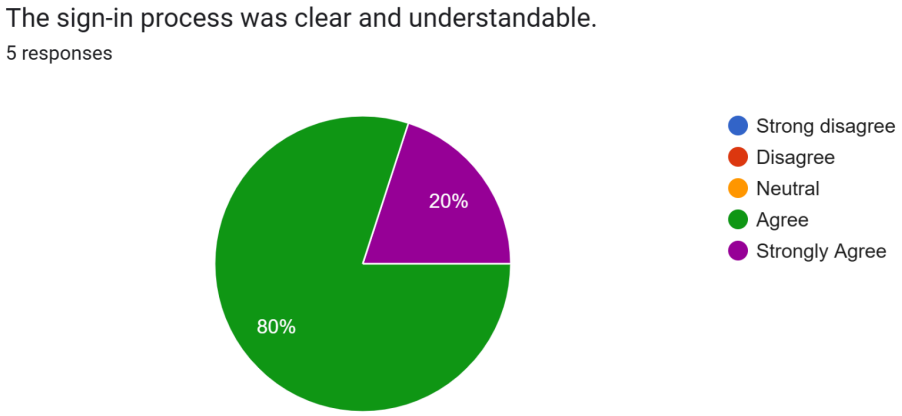


Figure K.53: Usability questionnaire practice sign-in question 2 responses pie chart.

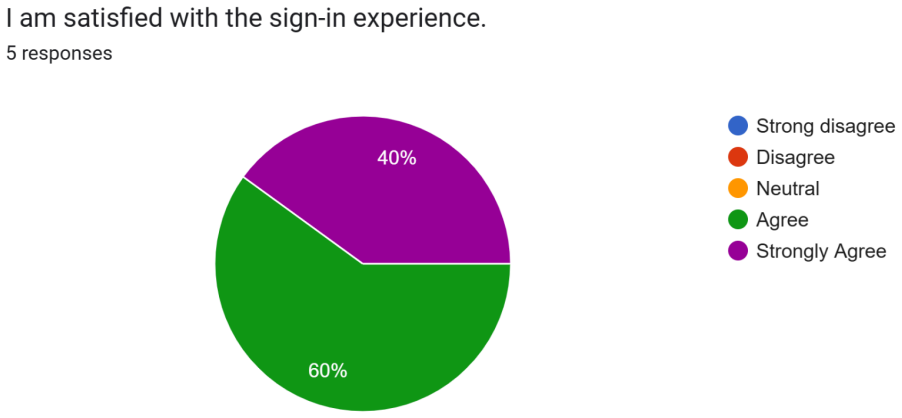


Figure K.54: Usability questionnaire practice sign-in question 3 responses pie chart.

K.2.3.8 Managing Practice Details and Preferences

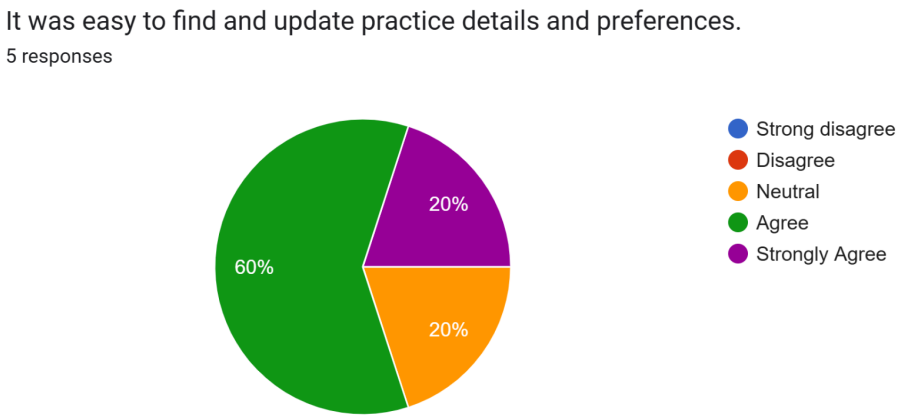


Figure K.55: Usability questionnaire managing practice details and preferences question 1 responses pie chart.

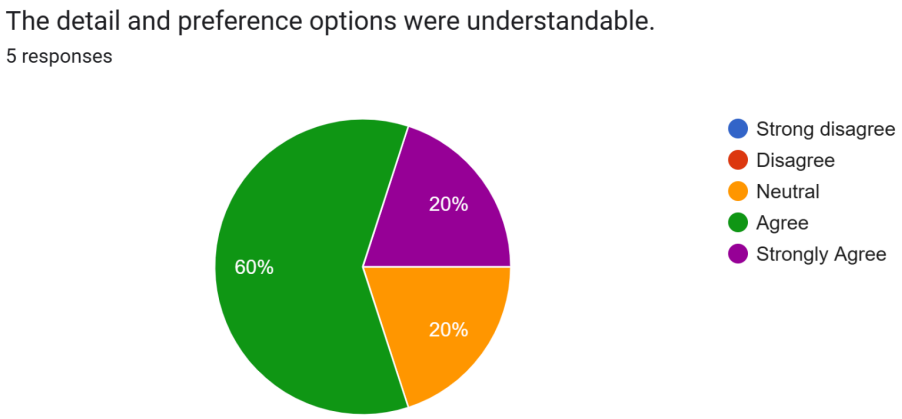


Figure K.56: Usability questionnaire managing practice details and preferences question 2 responses pie chart.

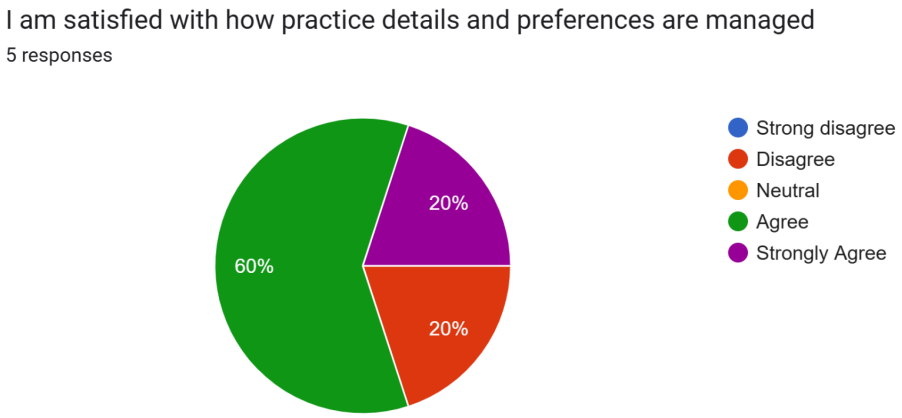


Figure K.57: Usability questionnaire managing practice details and preferences question 3 responses pie chart.

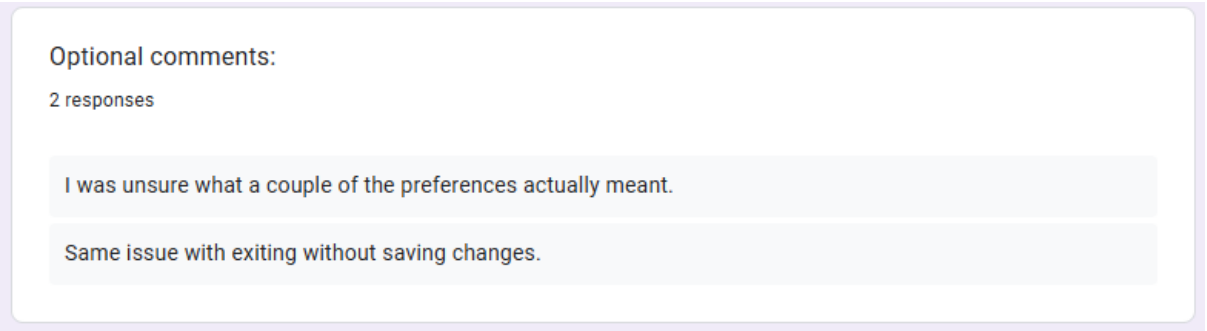


Figure K.58: Usability questionnaire managing practice details and preferences optional comments responses.

K.2.3.9 Managing Appointments

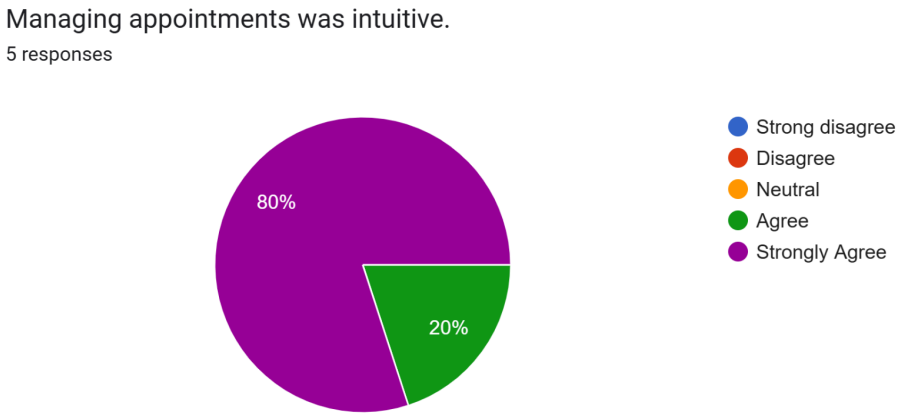


Figure K.59: Usability questionnaire managing appointments question 1 responses pie chart.

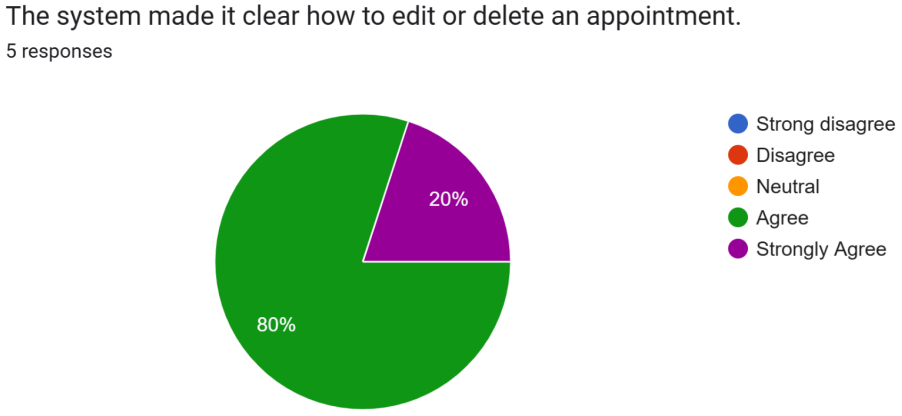


Figure K.60: Usability questionnaire managing appointments question 2 responses pie chart.

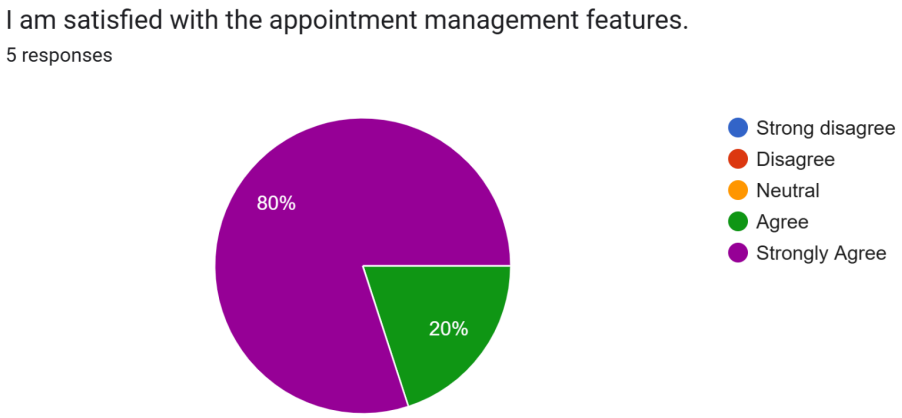


Figure K.61: Usability questionnaire managing appointments question 3 responses pie chart.