Exercices App

Team 6

Kyriaki Charalambous Xingrui Gu Najma Haji

COMP0067 App Engineering

MSc Computer Science

November 7, 2023

Department of Computer Science

University College London

Abstract

Physiotherapy and health exercises are vital components of rehabilitation and wellness programs for patients with various conditions. However, there is a lack of a centralized platform that allows users to access, record, and utilize physiotherapy exercises provided by the UCL MotionInput and UCL Exercises apps, in collaboration with NHS England and several charities, including the ALS MND Association. Furthermore, there is a need for an efficient method to search and categorize exercises, along with instructional YouTube videos to help users perform the movements correctly. Clinicians also require an effective tagging system for selecting appropriate exercises and creating custom exercise chains for patients. To address these challenges, we will develop a responsive web application that hosts JSON files of movements, provides cross-platform access, and integrates a searchable database and a gallery of YouTube links to enhance user experience and meet the specific needs of both clinicians and patients.

After brainstorming within our team of three, we have developed a responsive web application called ”UCL Exercises Web App” to address the aforementioned challenges. Built using React.js to ensure cross-platform compatibility, the app serves as a comprehensive platform for accessing, recording, and using physiotherapy exercises. We have established a centralized database that enables users to easily browse, search, and select exercises, as well as filter them by tags. Additionally, a search functionality allows users to find suitable exercises for patients using keywords. Each exercise includes a designated YouTube video link pointing to an instructional tutorial, ensuring maximum comprehension for patients. Moreover,

clinicians can create templates based on patients' needs, allowing them to combine multiple exercises into a customized exercise chain.

The development of the "UCL Exercises Web App" has brought significant achievements and positive impacts to the physiotherapy community. With its cross-platform compatibility, the app provides a comprehensive and user-friendly platform for accessing, recording, and using physiotherapy exercises. The centralized database, along with the search and filter functionalities, simplifies the process of finding and selecting exercises tailored to individual patient needs. By incorporating designated YouTube video links for each exercise, patients can better understand and correctly perform the exercises, improving the effectiveness of their rehabilitation programs. Furthermore, the app empowers clinicians to create customized exercise chains using templates based on patients' specific needs, leading to more personalized and targeted treatment plans. The platform also includes a patient portal, where patients can leave their own rehabilitation reviews and ratings, providing valuable insights for other patients. Overall, the "UCL Exercises Web App" enhances the accessibility, efficiency, and personalization of physiotherapy exercises for both patients and clinicians alike.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Project Background

The field of physiotherapy plays a crucial role in promoting health and well-being for patients suffering from various physical conditions and disorders. Rehabilitation exercises are an essential component of physiotherapy programs, designed to help patients regain strength, mobility, and overall function. However, accessing and managing personalized physiotherapy exercises can be a challenge for both patients and clinicians. This difficulty is further exacerbated by the lack of a centralized platform that allows users to access, record, and utilize physiotherapy exercises provided by various sources and applications, such as the UCL MotionInput and UCL Exercises apps.

Recognizing this need, UCL Computer Science collaborated with NHS England and several charities, including the ALS MND Association, to develop a solution that would address these challenges and improve the overall physiotherapy experience for patients and clinicians alike. The goal was to create a responsive web application that not only hosts and organizes physiotherapy exercises, but also offers a user-friendly interface, efficient search capabilities, and additional resources such as instructional videos.

The development of the ”UCL Exercises Web App” marks an important step towards achieving this goal, providing a comprehensive platform that bridges the gap between patients, clinicians, and physiotherapy resources. By bringing together

exercises, video tutorials, and customization options, the app aims to enhance the accessibility, efficiency, and personalization of physiotherapy exercises, ultimately improving patient outcomes and satisfaction.

1.1.1 The Client

Introducing the ”UCL Exercises Web App” - a user-friendly, responsive web application designed to simplify and enhance the physiotherapy experience for both patients and clinicians. Developed in collaboration with leading organizations, this innovative platform offers a centralized database of exercises, advanced search capabilities, video tutorials, and personalized exercise chains. Experience seamless access, efficient management, and customized physiotherapy exercises for improved patient care and satisfaction with the ”UCL Exercises Web App”.

1.1.2 Problem Statement

The lack of a centralized and user-friendly platform for accessing, managing, and utilizing physiotherapy exercises presents challenges for both patients and clinicians, leading to inefficiencies in treatment planning and execution, limited patient comprehension, and reduced personalization of physiotherapy programs. The need for a comprehensive solution that bridges the gap between patients, clinicians, and physiotherapy resources is crucial to enhance the accessibility, efficiency, and personalization of physiotherapy exercises, ultimately improving patient outcomes and satisfaction.

1.1.3 Project Goals

After meeting with our clients and partners, we gained a clear understanding of the project’s essential requirements. Utilizing the MoSCoW method, we prioritized features into ”Must-have,” ”Should-have,” ”Could-have,” and ”Won’t-have” categories. We also created user personas for various roles to further identify and confirm user pain points and address existing issues. Subsequent meetings refined the software’s functional and non-functional requirements. The collected require-

ments allowed the team to prioritize which features to implement first. Our goal is to create a user-centric application that is people-focused, easy to use, and capable of delivering all core functionalities identified during the requirements gathering process.

1.2 Project Outline

This project is primarily divided into two main phases: the design phase and the development phase. In the design phase, we first employed user research methods, such as the MoSCoW method, to analyze user needs. After confirming the requirements, we created early iterative prototypes using sketches and wireframes. Once the prototypes and page layouts were confirmed, we proceeded with wireframe iterations and drafted the website's flow to illustrate the relationships and connections between each page. Finally, in the design phase, we used Figma to create a user-centric interactive prototype, complete with corresponding interaction effects, to better visualize the application and intuitively demonstrate its usage and operation to clients. Through multiple iterations, we continually delved into user experience, aiming to enhance the product's usability and provide the best user experience possible. In the development phase, we utilized various front-end and back-end technologies. Node.js and Express were used for server-side applications and database calls, while MySQL served as the database management system. The application was hosted on an Azure virtual machine. For front-end development, we employed React, HTML, and CSS. By combining these technologies and methodologies, we created an intuitive and user-friendly web application that caters to the needs of both patients and clinicians.

1.3 Project Management

1.3.1 Development Team

Our team is made up of three talented individuals

Team Leader: Kyriaki Charalambous:

Team Member: Xingrui Gu (Hayden):

Xingrui Gu earned a BSc Mathematics with Statistics from King's College London. Prior to this, Xingrui Gu studied R, Java, C++, Python, MySQL and has limited experience with HTML, React, Node.js, and PHP. Additionally, Xingrui Gu has extensive internship experience in human-computer interaction, excelling at identifying user pain points and skillfully creating prototypes using Figma.

Team Member: Najma Haji:

1.3.2 Scheduling

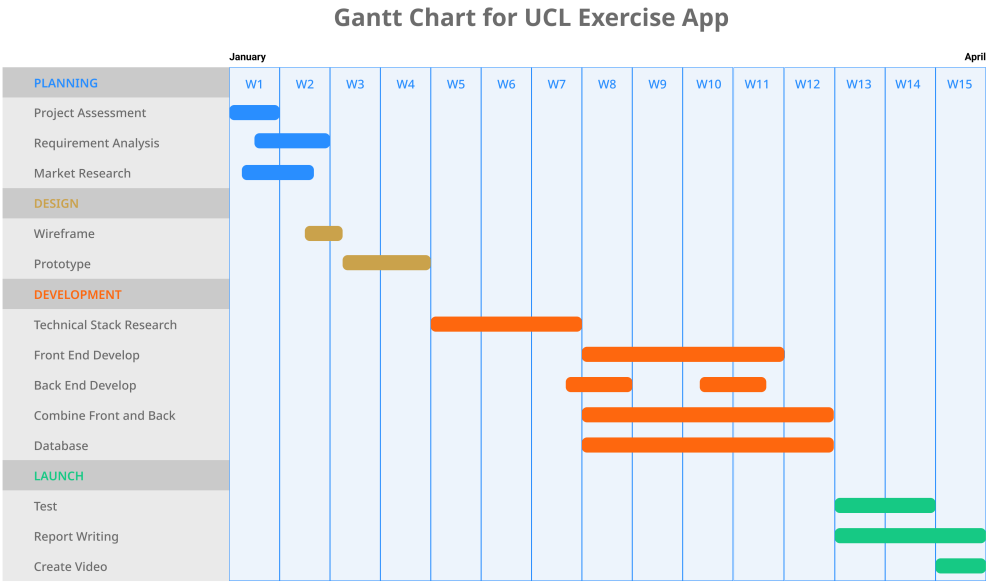

Figure 1.1: Gantt Chart

Chapter 2

Requirements

2.1 Requirements Gathering

To gather the requirements for our web-based exercise app, we employed a combination of methods, including client meetings, user surveys, and iterative feedback during the development process[1][2]. We also carried out a lot of patient research in order to design the product from the patient's point of view. In addition to this research, we also carried out a competitor analysis of similar products to determine the needs and pain points of users.[3]

Our initial meeting with the client helped us gain a better understanding of the project objectives and required functionalities. Through the early stages of the meeting, we obtained a preliminary understanding and summary of the client's needs and pain points. We then employed the MoSCoW method to analyze user requirements and prioritize them, categorizing them as must-haves, should-haves, could-haves, and won't-haves. This framework allowed us to focus on core features and determine which were essential for the minimum viable product (MVP) and which were optional or out of scope.

In terms of user analysis, we also received extensive input from Dr. Yun Fu, discussing various issues related to the project and its motivation in detail. After conducting the initial MoSCoW analysis, we realized that the weight of our must-

haves was quite significant, indicating that we had not captured user pain points effectively. We held a second meeting with our client, and after multiple iterative discussions, we finally confirmed and integrated a more comprehensive MoSCoW requirements analysis to better address the project's objectives and user needs.

2.2 Personas

A well-crafted user profile provides designers and developers with a deeper understanding of user needs and pain points. Consequently, throughout our multiple iterations, we created three distinct virtual personas. Although each persona doesn't represent a specific individual, they collectively illustrate the needs and pain points of the stakeholders we aim to address. These personas encompass their goals, interests, and motivations, all of which significantly influence our software product design. By conceptualizing the software's purpose and the users' needs, these core aspects continuously guide the implementation and design phases of the project. These personas also serve as a constant reminder of the diverse expectations of different users. Our team members consistently practice the customer-centric development and design philosophy, striving to create an app that will thoroughly satisfy our users.

pR

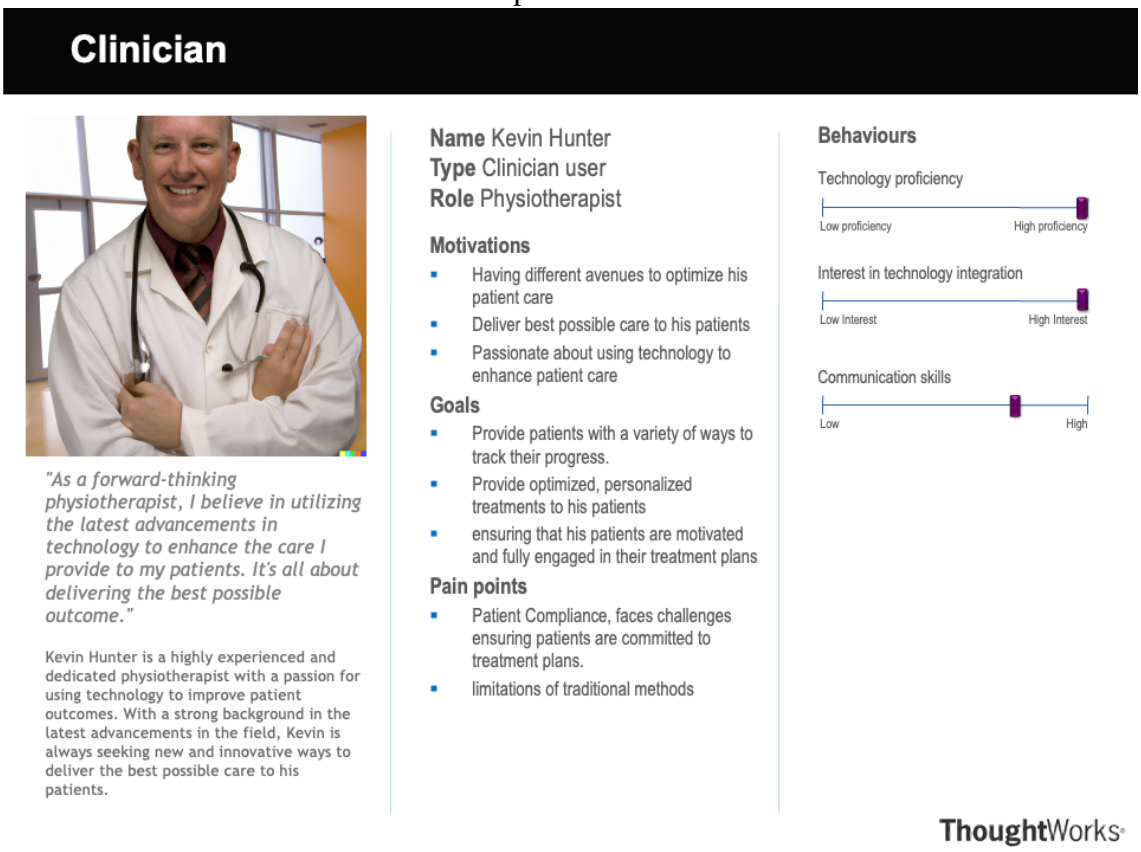

Figure 2.1: Persona 1 - Kevin Hunter

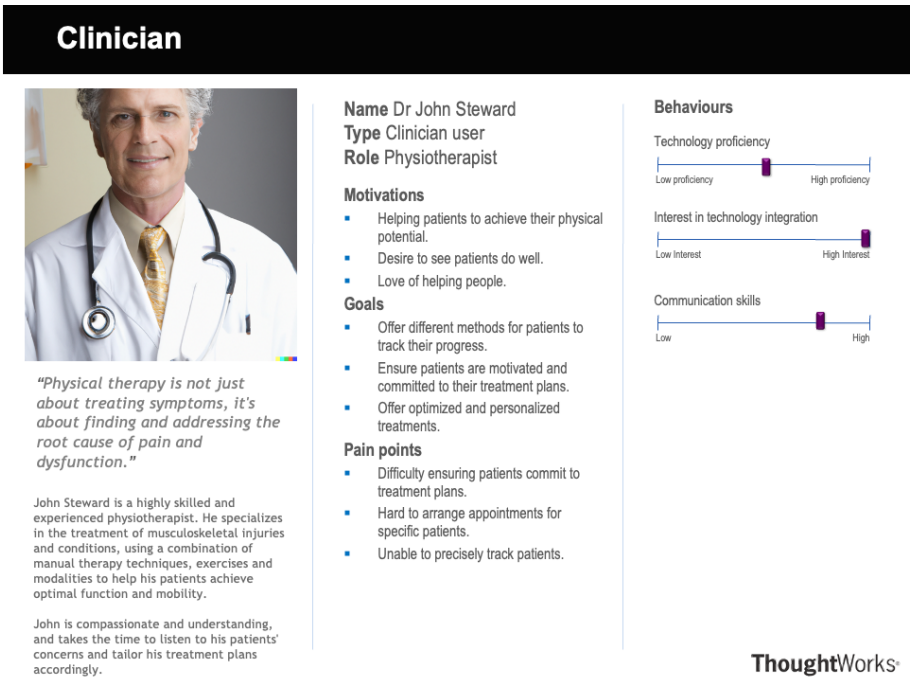

Figure 2.2: Persona 2 - John Steward

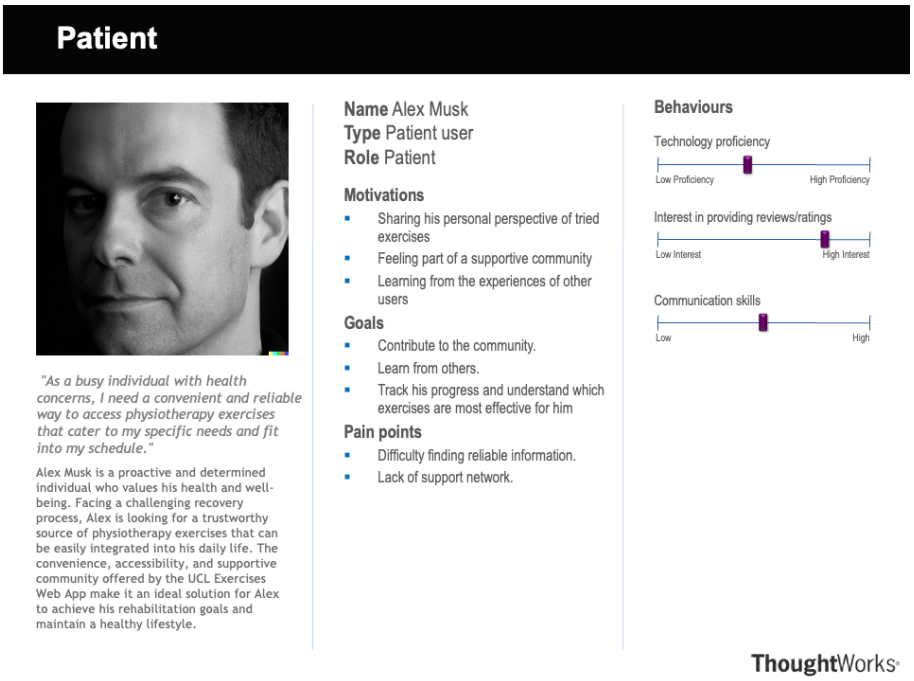

Figure 2.3: Persona 3 - Alex Musk

2.3 MoSCoW requirement list

We generated many versions of the requirements in consultation with the client over the course of the project. In the end, this was what was agreed on. The following tables enlist the final set of requirements accordingly.

2.3.1 Functional Requirement Specifications

Table 2.1: MoSCoW Requirements – Functional

ID	Requirement	Priority
R1	Patients can access the Web App without login and rate exercises/Templates.	Must
R2	Patients can leave comments on each exercise or templates.	Should
R3	Clinicians can login using Google or UCL authentication	Must
R4	Clinicians can select exercises from a gallery of exercises (from our database) and create programs for their patients	Must
R5	Clinicians are able to drag selected exercises and rearrange the order of their program	Should

R6	Clinicians can adjust the repetitions for each exercise and the frequency the program should be performed.	Should
R7	Clinicians are able to paste in a text field the emails of up to 50 patients and forward the program they created in an email	Must
R8	Clinicians have the option of saving a program as a template without having to send it to patients.	Could
R9	Clinicians are able to filter exercises based on specific filters such as Joint/Region, Movement and Condition	Must
R10	Clinicians should be able to search exercises based on exercise name, description, and tag name.	Should
R11	Clinicians are able to search templates based on timeline name, exercises, and tag name.	Should
R12	Clinicians have the ability to search though the library tab based on Name, Trust, Location, Date and Version.	Should
R13	Clinicians are able to upload JSON files of movements that would go in our database, write a description and have the ability to add tags	Must
R14	Clinicians are able to upload PDF files of .., write a description and have the ability to add trust author etc	Must
R15	Clinicians are able to view and edit templates.	Could
R16	Clinicians are able to view the description and watch a YouTube video of every exercise	Should

2.3.2 Non-Functional Requirement Specifications

Table 2.2: MoSCoW Requirements – Non-Functional

ID	**Requirement**	**Priority**
R17	A secondary app is able to retrieve certain information from our database	Must

R18	The Web App is responsive	Must
R19	User information on both Clinician and patient side is not stored	Must

2.4 Use Cases

We obtained the use cases after careful analyses of the MoSCoW requirements. For the use case analysis, we identified two primary actors and their interactions with different parts of the application. They are:

1. Clinician (C)

2. Patient (P)

The following table enlists the use cases which we had identified from the requirements. Each use case code depends on which user-base category is expected to be completing this action, either [C]linician or [P]atient.

Table 2.3: Use Case Listing

ID	Use Case
UCP1	Make Review/Rating on Exercises
UCP2	Make Review/Rating on Timelines
UCP3	Login via NHS Oauth login
UCC1	Browse Exercises
UCC2	Browse Timelines
UCC3	Browse *PDF's*
UCC4	Send Program to patients
UCC7	Choose and download a PDF
UCC8	Upload a new Exercise
UCC9	Upload a PDF

2.5 Use Case Diagram

The following figure contains a use case diagram which visualises the interactions between all actors and the application.

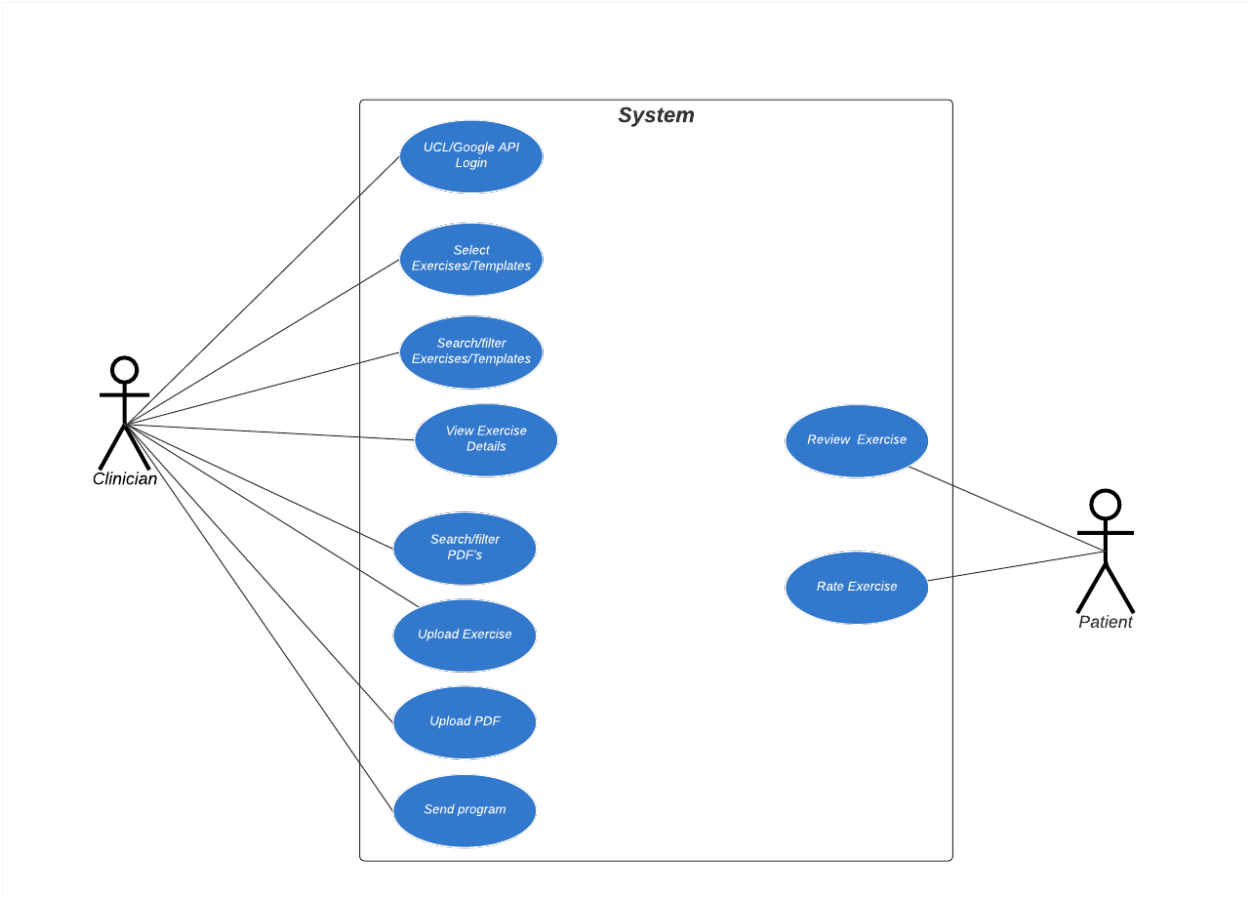

Figure 2.4: Use Case Diagram

Chapter 3

Research

3.1 Related Applications

Before starting our prototype design, we conducted a competitor analysis of physiotherapy and exercise-related apps currently on the market. Here are three relevant apps widely used by large organizations, followed by a detailed competitor analysis:

3.1.1 Physitrack:

Physitrack is a comprehensive app that offers physiotherapy exercises, patient education, and remote monitoring. It caters to healthcare professionals and patients, providing a platform for clinicians to create personalized exercise programs and track their patients' progress.

Competitor Analysis: Physitrack boasts an extensive range of exercise content and robust tracking features, along with a clear page layout and UI design. However, it may cater more to healthcare professionals rather than end-users, potentially making it less user-friendly for patients.

3.1.2 MyPhysioRehab:

MyPhysioRehab is an app that enables physiotherapists to create customized exercise programs for their patients. It offers a library of over 3,000 exercises, and patients can access their prescribed programs and track their progress.

Competitor Analysis: MyPhysioRehab has an extensive exercise database, this

provides us with a good basis for building a solid database of movements. But it may not have the same user-centric design as our app. Additionally, it seems to lack a keyword-based search feature and a tagging system for exercises.

3.1.3 HEP2go:

HEP2go is a web-based platform that allows healthcare professionals to create home exercise programs for their patients. It features a large library of exercises and enables customization of programs with instructional videos and images.
Competitor Analysis: HEP2go provides a web-based platform with a vast range of exercises, as an example of how exercise programs can be created and shared with patients, but it also highlights the importance of user-centric design and seamless integration with other platforms for an optimal user experience. However, it may lack user-friendliness and cross-platform compatibility. It also doesn't mention any integration with YouTube video links for a more comprehensive understanding of the exercises.

In summary, our analysis of the three main competitors, Physitrack, MyPhysioRehab, and HEP2go, has shown that while each app offers valuable features such as extensive exercise libraries and program customization, they also have their limitations. Areas for improvement include user-friendliness, keyword-based search functionality, exercise tagging systems, and cross-platform compatibility. By addressing these gaps, our program can provide a more comprehensive and user-centric solution for both healthcare professionals and patients.

3.2 Related Technologies

3.2.1 Front End Technology

- **HTML:**

  Abilities: HTML (HyperText Markup Language) is the standard markup language used for creating web pages and web applications.

  Pros: Easy to learn and understand; widely supported across browsers.

Cons: Limited to static content; needs to be combined with other technologies for dynamic content.

- **CSS:**

  Abilities: CSS (Cascading Style Sheets) is used for styling web pages and controlling the appearance of HTML elements.

  Pros: Allows for separation of presentation from content; easy to maintain; efficient for styling multiple pages.

  Cons: Can be verbose; browser compatibility issues may arise; limited programming capabilities.

- **JavaScript:**

  Abilities: JavaScript is a powerful scripting language used for adding interactivity and dynamic content to web pages.

  Pros: Widely supported by browsers; versatile for various tasks; large community support.

  Cons: May cause performance issues if not optimized; differences in browser implementations can lead to inconsistencies.

- **jQuery:**

  Abilities: jQuery is a popular JavaScript library that simplifies HTML document traversal, event handling, and animation.

  Pros: Easy to learn and use; cross-browser compatibility; large plugin library.

  Cons: Can be slower compared to vanilla JavaScript; not as relevant due to modern browser improvements and other libraries.

- **React:**

  Abilities: React is a popular JavaScript library for building user interfaces, particularly single-page applications.

  Pros: Encourages reusable components; efficient performance with virtual DOM; strong community support.

  Cons: Steeper learning curve; requires additional libraries for more complex applications; JSX syntax can be confusing.

- **Vue:**

  Abilities: Vue.js is a progressive JavaScript framework for building user interfaces and single-page applications.

  Pros: Easy to integrate with existing projects; lightweight; supports reusable components; easy learning curve.

  Cons: Smaller community compared to React; less mature than other frameworks; limited resources and third-party libraries.

- **Bootstrap:**

  Abilities: Bootstrap is a popular CSS framework for building responsive, mobile-first websites.

  Pros: Easy to use; responsive design; extensive pre-built components; strong community support.

  Cons: Can lead to similar-looking websites; limited customization options; may include unnecessary code.

3.2.2 Back End Technology

- **Node.js:**

  Abilities: Node.js is a JavaScript runtime built on Chrome's V8 engine for building scalable network applications.

  Pros: Non-blocking, asynchronous I/O; large package ecosystem; full-stack development in JavaScript.

  Cons: Not suitable for CPU-intensive tasks; callback hell can occur; single-threaded nature.

- **Express:**

  Abilities: Express is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications.

  Pros: Easy to learn and use; middleware support; great for RESTful API development; large community support.

  Cons: Less opinionated, which might lead to poor design decisions; manual configuration.

- **MySQL:**

  Abilities: MySQL is an open-source relational database management system based on SQL (Structured Query Language).

  Pros: Easy to use; great performance; strong community support; ACID-compliant transactions.

  Cons: Limited support for NoSQL data storage; less advanced features compared to some competitors.

- **MongoDB:**

  Abilities: MongoDB is a popular open-source NoSQL database that stores data in flexible, JSON-like documents.

  Pros: Schema-less design; horizontal scaling; good for handling large amounts of unstructured data.

  Cons: Limited support for ACID transactions; may require additional memory and storage.

- **EJS:**

  Abilities: EJS (Embedded JavaScript) is a simple templating engine that lets you generate HTML markup with plain JavaScript.

  Pros: Easy to learn and use; integrates seamlessly with Express.js; fast rendering.

  Cons: Limited feature set compared to other templating engines; less attractive syntax.

- **Pug:**

  Abilities: Pug (formerly Jade) is a high-performance template engine that emphasizes a clean, minimal syntax.

  Pros: Clean, whitespace-sensitive syntax; extensible through filters and mixins; great with Express.js.

  Cons: Steeper learning curve; unique syntax may not be suitable for all developers.

- **PHP:**

Abilities: PHP is a widely-used open-source scripting language especially suited for web development.

Pros: Easy to learn; large community support; many available frameworks; good CMS integration.

Cons: Inconsistent library and API; performance may not be as good as some other languages.

- **Django:**

  Abilities: Django is a high-level Python web framework that encourages rapid development and clean, pragmatic design.

  Pros: Encourages best practices; batteries-included approach; great for rapid development; strong security.

  Cons: Monolithic architecture;

- **Django:**

  Abilities: Django is a high-level Python web framework that encourages rapid development and clean, pragmatic design.

  Pros: Encourages best practices; batteries-included approach; great for rapid development; strong security.

  Cons: Monolithic architecture; steep learning curve; may feel too opinionated for some developers.

- **Azure:**

  Abilities: Microsoft Azure is a cloud computing service for building, testing, deploying, and managing applications and services.

  Pros: Scalable and flexible; wide range of services; good integration with other Microsoft products; strong security.

  Cons: Can be expensive; steep learning curve; potential vendor lock-in.

- **AWS:**

  Abilities: Amazon Web Services (AWS) is a cloud computing platform that offers a wide range of services, including computing power, storage, and databases.

Pros: Extensive range of services; highly scalable and reliable; strong security and compliance features; large community and ecosystem.

Cons: Can be expensive; steep learning curve; complex pricing structure; potential vendor lock-in.

3.3 Comparison & Review of Selected Solutions

3.3.1 Front End:

After Dr. Yun Fu's patient introduction and explanation during the class, our team carefully discussed and decided to adopt HTML, CSS, JavaScript, and React as our frontend technology stack. Since React primarily uses a syntax similar to HTML, we build pages using functional components and utilize JSX for page content. We then apply CSS for UI beautification to achieve an excellent user experience. Per our client's request for a responsive interface, we use relative positioning in CSS for parameterized adjustments.

3.3.2 Back End:

For backend development, we ultimately chose the Node.js and Express framework for several reasons. First, since both React and Node.js are based on JavaScript, using a single language stack allows us to use the same programming language for both frontend and backend, reducing learning and maintenance costs. Second, Node.js provides high-performance advantages through its event-driven and non-blocking I/O model, which is essential for our web application when handling numerous concurrent connections. Additionally, Node.js boasts a rich ecosystem with an extensive selection of third-party libraries and frameworks, such as Express, which offers us convenient functionality implementation and broad possibilities for our app's development and future enhancements. Finally, to promote code reusability, the combination of Node.js and React enables code sharing between frontend and backend, which helps to reduce development time and improve code maintainability.

3.3.3 Database:

In the realm of databases, we have chosen MySQL as our database management system for creating the Exercise dataset after thorough deliberation. This decision was based on several factors, including MySQL's status as a mature and stable database system, its open-source licensing, which aids in reducing project operational costs, and the extensive database development community it supports. This community provides an abundance of resources, documentation, and forums, thus lowering the technical threshold for future app maintenance and development.Furthermore, MySQL offers exceptional cross-platform compatibility and high-performance query processing capabilities. In addition to these advantages, we employ MySQL Workbench for database management and maintenance, which simplifies tasks for developers and maintenance staff involved in database operations, ensuring efficient and seamless management.

3.3.4 Web Hosting:

Regarding project deployment, we chose Azure as our application hosting platform based on user requirements. This decision mainly considers the following factors. First, Azure is a globally renowned cloud service provider that delivers high availability, scalability, and low-latency infrastructure, ensuring stable application performance. Second, Azure offers a wealth of tools and services that support rapid deployment and management of applications, enhancing development efficiency. Furthermore, Azure allows for automatic scaling and contraction of applications, dynamically adjusting resources according to performance demands. In addition, Azure provides a range of security measures and compliance certifications to safeguard application and data security. Lastly, Azure's excellent integration support for Node.js and Express makes it convenient to deploy our application on the platform.

3.4 Summary

In summary, our team has selected a robust technology stack for the project after thorough deliberation and guidance from Dr. Yun Fu. Despite our team's extensive experience in PHP, we have opted for Node.js and React as our primary development frameworks to benefit from lower maintenance costs and a more versatile development environment. For the frontend, we employ HTML, CSS, JavaScript, and React, taking advantage of functional components and JSX for efficient page construction and using CSS to create a responsive and visually appealing interface. The backend is built using Node.js and Express, capitalizing on the benefits of a unified language stack, high performance, an abundant ecosystem, and code reusability. In terms of database management, we have chosen MySQL for its maturity, open-source licensing, vast community support, and exceptional compatibility and performance. We also utilize MySQL Workbench for streamlined database management and maintenance. Finally, we have selected Azure as our web hosting platform, leveraging its high availability, scalability, comprehensive tools and services, security measures, and seamless integration with our chosen technologies.

Chapter 4

HCI Design

4.1 Design Principal

Before beginning the design process, we established several guiding principles to help us create a user-centered and easy-to-use application. These principles ensure that our app caters to the needs of various stakeholders, including clinicians and patients:

1. **Consistency**: Ensuring a consistent interface in terms of color scheme, typography, and design style helps users establish familiarity and reduces the learning curve[4]. Our design aims to maintain highly similar interface styles and interaction methods between the patient and clinician sides.

2. **Hover states, feedback, and responsiveness**: The app should provide hover states for buttons, images, and text, as well as feedback upon user interaction. Responsiveness is also crucial, ensuring smooth and efficient performance without lag or delay[5]. These design standards enhance user experience and prevent accidental interactions.

3. **Clarity and simplicity**: The app's interface should be easy to use and understand. Users should effortlessly navigate the app and complete their tasks without confusion. This can be achieved by using clear labels, concise instructions, and consistent layouts throughout the application.

4. **Affordance**: The software design should always adhere to a user-centric phi-

losophy, ensuring that users can easily understand the purpose and functionality of interface elements through intuitive and discoverable design principles[6].

5. **User control and error prevention**: Empower users by providing control over their actions and the ability to undo or redo tasks when necessary. Guide users towards correct interactions with clear instructions and error messages to effectively reduce mistakes[5].

6. **Aesthetics and visual appeal**: A visually appealing app enhances the overall user experience and user engagement[7]. Utilize a harmonious, elegant color scheme, a professional logo, and a visually pleasing design consistent with your brand image.

7. **Flexibility and customization**: Our app should be responsive and adaptable to various device types and screen sizes[8], adjusting its layout for different orientations. Additionally, the app should grant users maximum control, making them the true owners of the software.

8. **Accessibility**: The app's design should cater to users with different abilities and usage scenarios, including those with visual, auditory, cognitive, or motor impairments. Implement features such as text-to-speech, speech-to-text, and adjustable font sizes to ensure accessibility for all users[9].

4.2 Design Cycle

4.2.1 Hand-drawn sketches

Upon identifying the client's needs and establishing our product design paradigm and principles, our team conducted a series of brainstorming sessions to sketch ideas for each page of the application. These sessions were informed by various design requirements and insights gathered from competitor analysis. Additionally, we created a site map to visually depict the hierarchy and relationships between different pages available to users.

During the brainstorming sessions, we utilized sketchpads and iPad drawing boards to conceptualize and discuss user requirements and user flows. Each team member contributed their ideas, gradually bringing to life the login screen, main Exercise screen, main Templates screen, browsing screen, uploading screen, and others. Although these initial sketches were simple, consisting of lines and squares without color, our team consistently adhered to our design principles of style and simplicity.

As we worked on the details of each specific interface, we repeatedly considered user needs, delving deeper into user pain points to uncover additional requirements and opportunities for improving the overall user experience. After a three-hour brainstorming session, we identified six basic interfaces for the software product: role selection, login, registration, main interface, message creation, and Templates.

By maintaining a strong focus on user-centered design and adhering to our established design principles, we aim to develop an application that effectively addresses user needs and provides an intuitive and satisfying user experience.

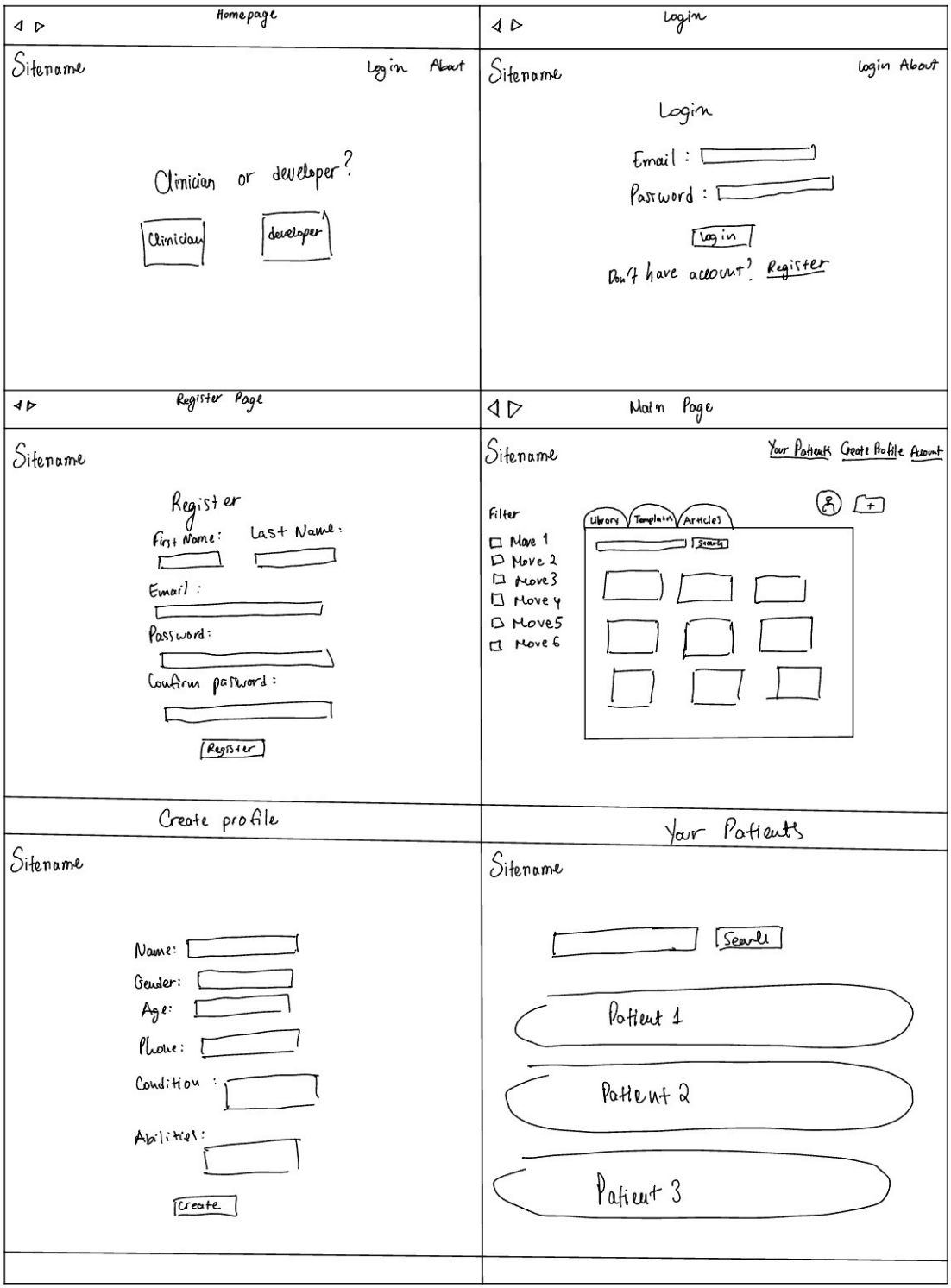

Figure 4.1: Hand Drawn Sketches

4.2.2 Wireframes

Simultaneously, our interaction designers and team members engaged in multiple iterations of the wireframes, striving to refine and improve our product design by incorporating a broader range of ideas and concepts. In the first iteration, the interaction design was primarily based on intuition, which can be reliable and similar to experience in certain situations. As a result, many of the intuitive elements were retained in subsequent iterations.

However, through multiple iterations, we continuously discarded bland and outdated layouts and designs. Gradually, our focus shifted towards enhancing the interface aesthetics without compromising usability. After several iterations, we successfully struck a balance between aesthetic appeal and functionality.

Finally, during a meeting with our client, we presented and explained the wireframes, soliciting their feedback and opinions. By incorporating the client's insights, we were able to further refine our design, ensuring that it met both their expectations and the needs of our target users, all while adhering to our established design principles and maintaining a distinct, modern visual identity.

1. Firstly, according to our client's needs, our product faces two different user groups, so a role-selected login screen is a necessity

Figure 4.2: Choose Role

2. If the user selects Clinician, both Google and UCL login options will appear.

Figure 4.3: Log in

3. By logging in with own account we come to the main page, which is a Library page, this is the core function of our products

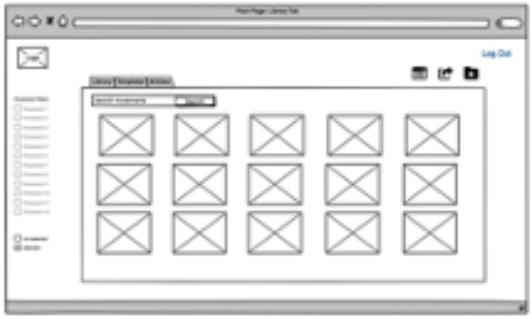

Figure 4.4: Main page

4. Then there is the Template interface. Template to allow the user to better manage the timeline of a combination of actions

5. The Article page is designed as a long bar, this is mainly to take into account the fact that unlike Exercise, which does not need to show images, Article needs to show a lot of documentation details

Figure 4.5: Template

Figure 4.6: Article page

6. In the top right corner of the page, there is a share button, which comes from the user action logic, where the user can copy this link and send it to the patient

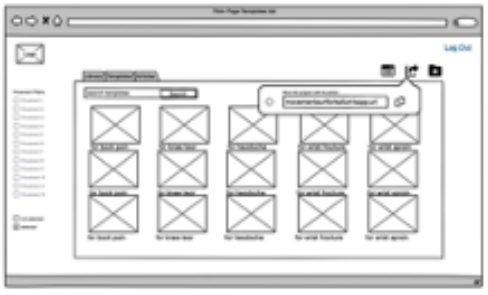

Figure 4.7: Share

7. Then there is the upload screen, where we have added a large number of entries based on user demand

Figure 4.8: Upload

8. Then there is the Program Editor, which, in line with our design principles, observes the maximum autonomy based on the user

Figure 4.9: Program Editor

9. We have also set up a developer interface according to customer requirements and Kallum's needs

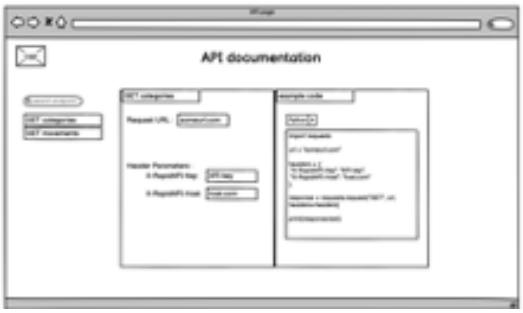

Figure 4.10: Developer Mode

4.2.3 Interactive Prototype

In the high-fidelity interactive prototype design process, we explored a variety of cases and styles for palette selection, as we firmly believe in providing the best "aesthetics and minimalist design" for an optimal user experience. The exploration process was also a highly heuristic evaluation process, stemming from insights obtained through Nielsen's ten usability principles.

Ultimately, through the relentless efforts of our interaction designers and team members, we chose a blue and white color scheme of #F0F4F7, #0380FE, and #FFFFFF, which aligns with the mainstream style of most medical app designs. According to color psychology, blue and white evoke a sense of tranquility and serenity, meeting the psychological needs of users for medical-related apps. Simul-

taneously, we followed our feedback and hover design principles, adding mouse-hover states to each block and button, facilitating a more intuitive user interaction learning and usage experience.

Furthermore, we added corresponding background blue light to each Exercise and Template block, providing intuitive feedback for every user interaction and enhancing the overall user experience. To improve the visual appeal of the interface, we incorporated numerous shadow designs, offering users a better sense of depth. Additionally, we added a timeline on the right side, displaying the sequence of Exercises as users add them, and enabling drag-and-drop functionality, which aligns with users' natural interaction habits and enhances their experience.

To adhere to simplicity and consistency design principles, we used an abundance of icons in the software, improving visual recognition and interface cleanliness. Overall, we conducted a successful design attempt. Although it still has some shortcomings, we received positive feedback in subsequent client reviews.

The final prototype design was structurally similar to the earlier wireframes but was given color, simplified layout, and cleaner interface, taking minimalism into account. The completion of the final prototype marked the end of all design iterations. Even though the design was significantly adjusted during the web application development process, the final product's layout closely resembled the final prototype.

The final prototype design is structurally basic to the previous wireframes, but given colour, and we simplified the layout, made the interface neat and tidy, and took minimalism into account. Final" means that this is the end of all design iterations. Even though the design was still tweaked considerably during the development of the web application, the final product can be seen to be quite similar to the final prototype, at least in terms of layout.

The Link of the Interaction Prototype: `https://www.figma.com/proto/hrYwqX1pUSuKsjOSRslsDj/App-Engineering?node-id=168%3A420&scaling=scale-down&page-id=0%3A1&starting-point-node-id=168%3A420`

Figure 4.11: Choose Choice

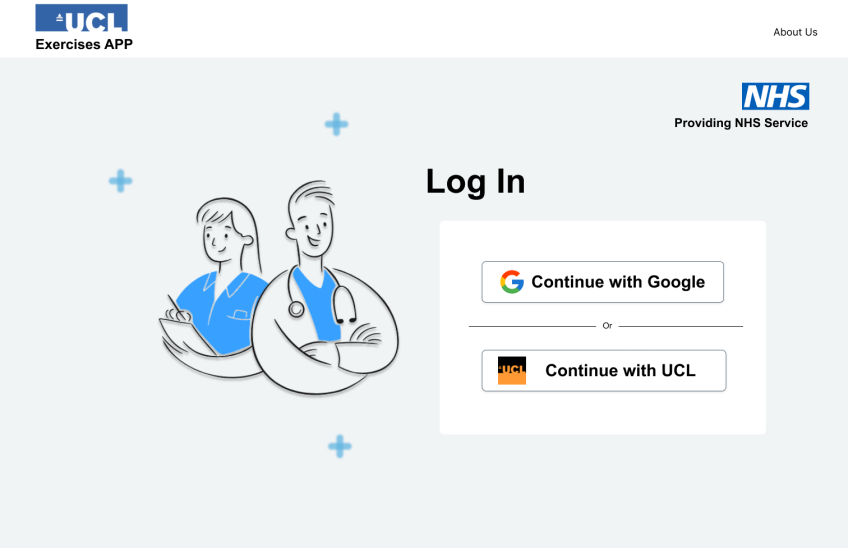

Figure 4.12: Log In

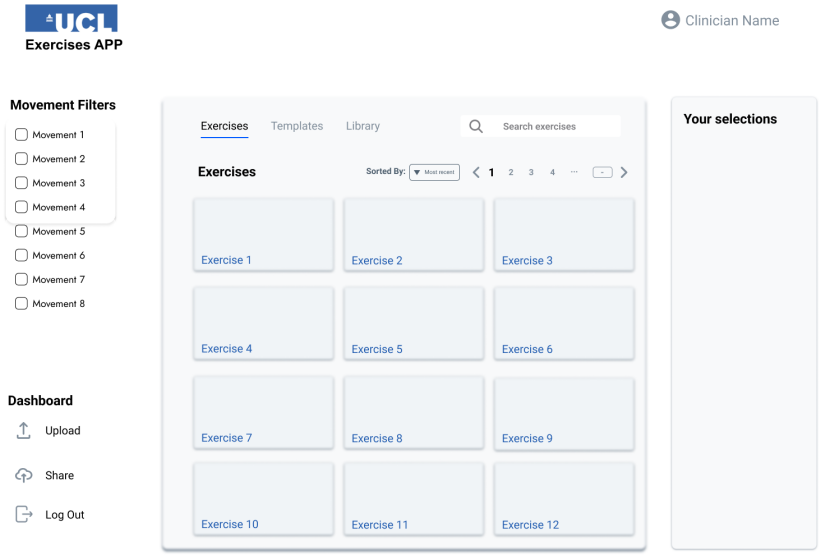

Figure 4.13: Exercise Main Page

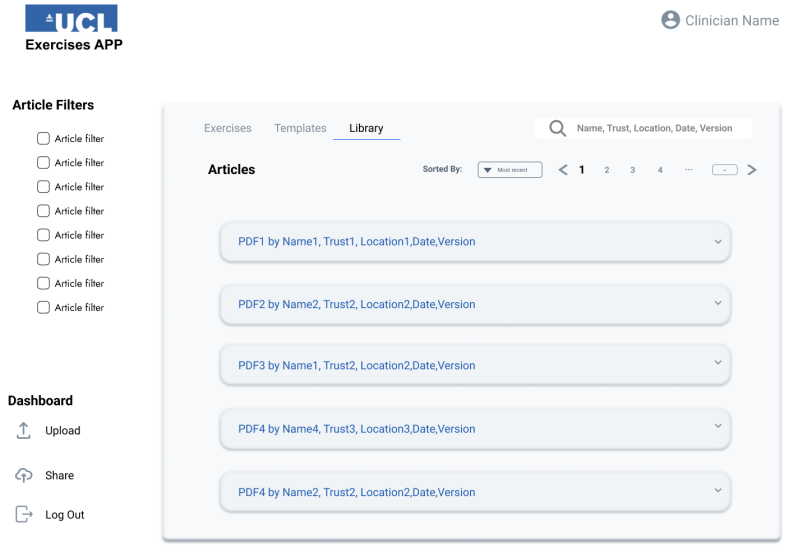

Figure 4.14: Article Main Page

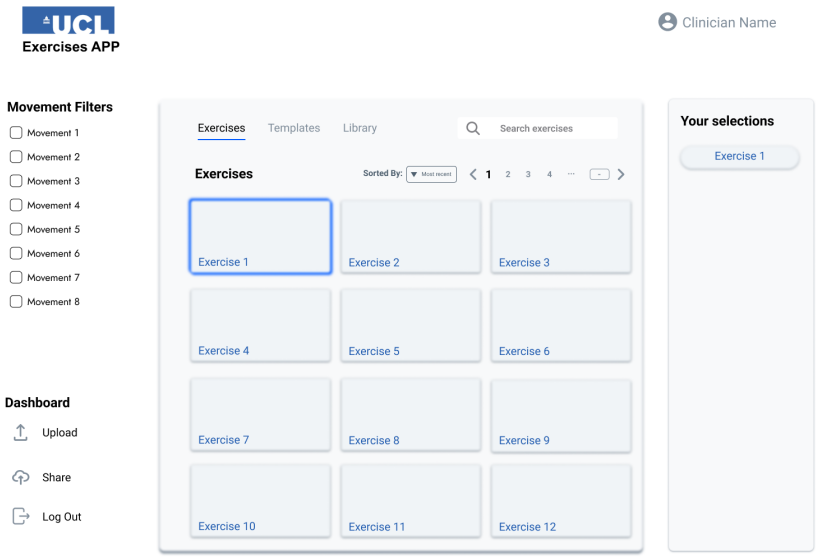

Figure 4.15: Exercise Main Page(Choose)

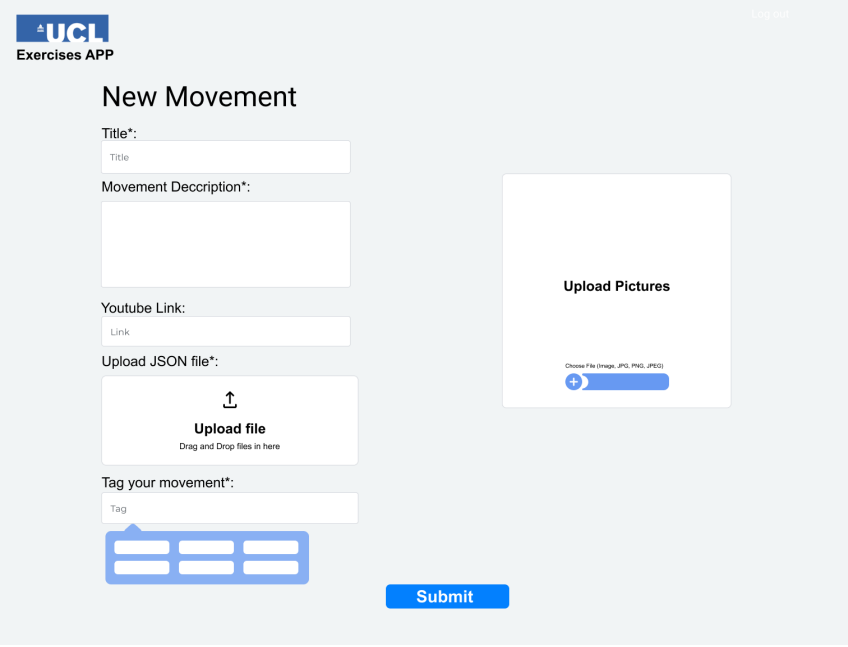

Figure 4.16: Exercise Add

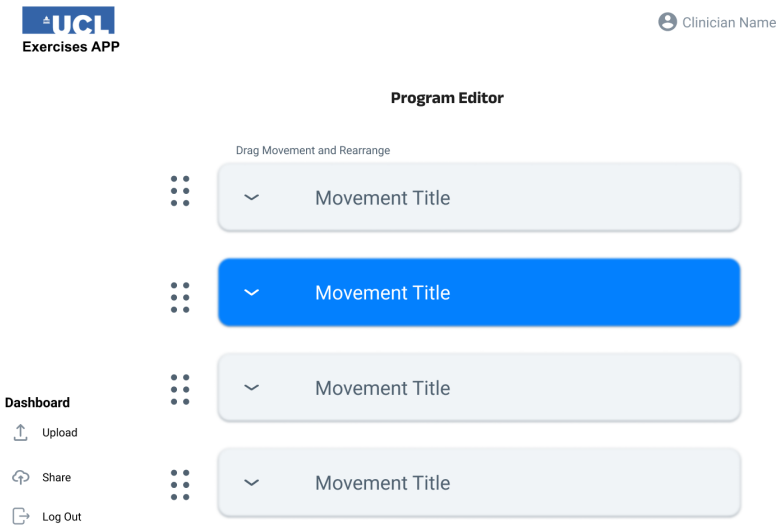

Figure 4.17: Program Editor

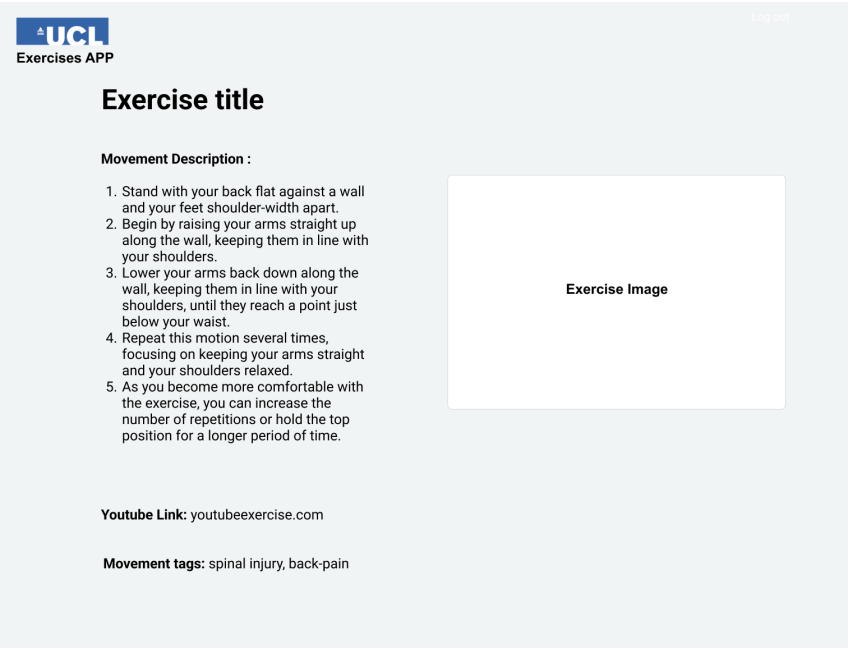

Figure 4.18: Exercise Detail

Chapter 5

System Design

5.1 System architecture

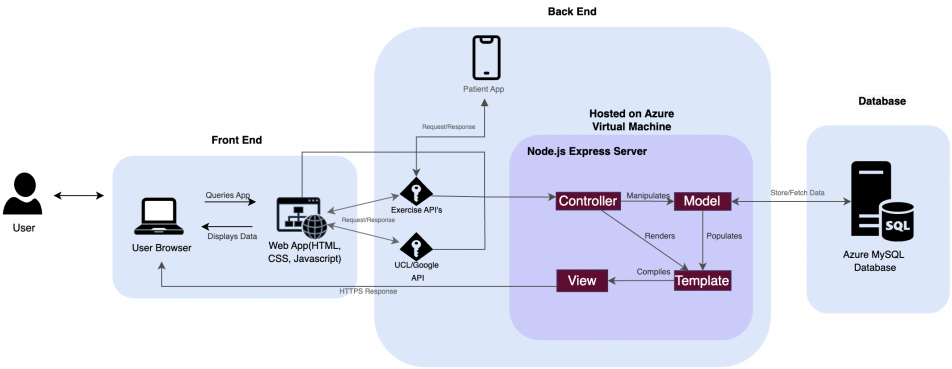

Figure 5.1: System Detail

Our web application's system architecture is segmented into four primary components, namely users, front end, back end, and the database, following the architectural design principles outlined by Garlan and Shaw [10].

1. Users: Our application is specifically tailored to accommodate two distinct user roles - Clinicians and Patients. We've developed unique user interfaces to cater to the specific needs of each group.

2. Frontend: The frontend serves as the visual interface of our application, delivering an intuitive and interactive user experience. It presents the application's content through a user-friendly graphical user interface (GUI) [11], facilitating seamless interaction between the users and the application.

3. Backend: The backend, which could be thought of as the central processing unit of our application, manages the application logic and handles critical functionalities such as user login and email operations [11]. It also interfaces with the database, fetching data to supply the frontend and processing user requests.

4. Database: We employ Azure as the primary hosting platform for our MySQL database [11], ensuring the secure and efficient storage and retrieval of user data.

Our web application has been designed with a specific aim - to enable clinicians to create personalized physiotherapy exercise programs for their patients. The application is built on a robust Node.js and Express server, utilizing Google and UCL authentication APIs to ensure secure user access. The application is hosted on Azure and is connected to a MySQL database that stores user-selected exercises.

The only external application interfacing with our web application is the Patient App. This app allows patients to access their prescribed physiotherapy exercise programs. The system utilizes a single API to deliver information from our database to the Patient App. This API fetches the exercise programs stored in the database as timeline JSON files, making them readily accessible for patients via their app.

The backend components of our system primarily comprise Node.js as the runtime environment, Google API login and UCL account API login for secure user authentication [12][13], Express as a web application framework, and a MySQL database [14] for secure data storage. Furthermore, we incorporate session management to maintain user data across consecutive requests. These components, when combined, create a reliable, secure, and easily maintainable backend architecture, delivering high-quality services to users while ensuring superior performance and excellent user experiences.

In conclusion, our system architecture efficiently manages the flow of data between these different components, ensuring a seamless and user-friendly experience.

5.2 Site Map

Creating a sitemap is a crucial aspect of website design and development, as it offers numerous benefits to both website owners and users.

First and foremost, a sitemap serves as a blueprint for your website, allowing you to plan and organize content and structure more effectively. By visualizing the hierarchy and relationships between different pages, you can ensure that your website is easy to navigate, free from dead ends or orphaned pages. This leads to a more user-friendly experience, as visitors can quickly locate the information they seek.

Secondly, sitemaps are an essential tool for Search Engine Optimization (SEO). By submitting your sitemap to search engines like Google, Bing, or Yahoo, you provide them with an easy-to-understand guide to your website's structure. This enables search engine crawlers to index your site more efficiently, increasing the likelihood of your content appearing in relevant keyword search results. As a result, a well-organized sitemap can lead to higher organic search rankings and greater visibility for your website.

Lastly, sitemaps aid in website maintenance and updates. As your website grows and evolves, tracking its structure and content becomes increasingly challenging. Sitemaps offer a clear overview of the existing architecture, making it easier for developers to identify areas that require improvement or updates. In this way, sitemaps help ensure that your website remains functional, up-to-date, and appealing to users.

In conclusion, creating a sitemap is a key step in website development, as it enhances usability, supports SEO, and simplifies website maintenance. Therefore, we have collectively created a sitemap for our app's structure, which primarily consists of two parts - two distinct structures for the Clinician and Patient interfaces.

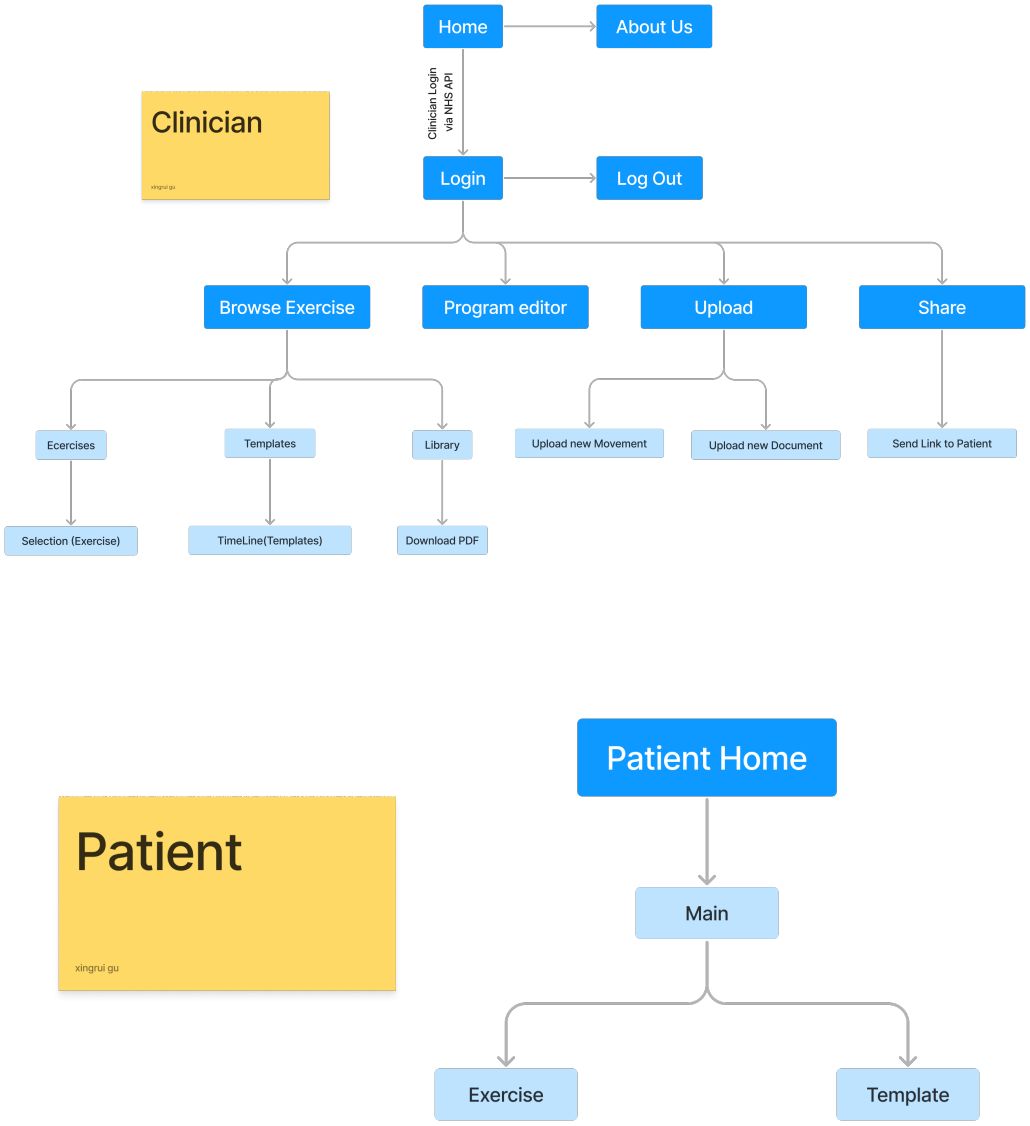

Figure 5.2: Site Map Clinicians and Patients

5.3 E-R Diagram

Entity-Relationship (ER) diagrams play a vital role in the design and development of databases, as they provide a visual representation of the data model and its structure.

The database "exercisesapp" consists of seven tables: `comments`, `exercises`, `likes`, `pdfs`, `reviews`, `tags`, and `timelines`. Below is a detailed description of each table:

- **comments:** This table includes four attributes. The `NewCommentId` attribute is an auto-incremented integer that acts as a primary key in conjunction with `CommentId`. The `Text` attribute is a string of maximum length 7000, representing the comment text. The `UserType` attribute is a string of maximum length 200, representing the type of user that made the comment.

- **exercises:** This table contains five attributes. The `Title` attribute, which acts as the primary key, is a string of maximum length 400. The `Description` attribute is a string of maximum length 10000, describing the exercise. The `Link` attribute is a string of maximum length 300, providing a link related to the exercise. The `JSON` attribute is a string of maximum length 50000, storing the JSON data related to the exercise. The `CreatedAt` attribute is a datetime value, indicating the creation time of the record.

- **likes:** This table contains two attributes. The `CommentId` attribute is an integer acting as a primary key and a foreign key referencing the `CommentId` in the `reviews` table. The `Likes` attribute is an integer, representing the number of likes a comment has received.

- **pdfs:** This table consists of seven attributes. The `PDFTitle` attribute acts as the primary key and is a string of maximum length 400. The `Author`, `Trust`, `Location`, and `PDFDescription` attributes are strings of maximum length 300, 300, 300, and 10000 respectively. The `PDF` attribute is a longblob type, storing the actual PDF data. The `PDFCreatedAt` attribute is a datetime value, representing the creation time of the record.

- **reviews:** This table comprises five attributes. The `CommentId` attribute is an auto-incremented integer that acts as a primary key in conjunction with `ExerciseTitle`. The `ExerciseTitle` attribute is a string of maximum

length 255. The `Rating` attribute is an integer, representing the rating given to the exercise. The `Comment` attribute is a string of maximum length 7000, containing the review comment. The `CreatedTime` attribute is a datetime value, indicating the time the review was created.

- **tags:** This table includes two attributes. The `Title` and `Tag` attributes are strings of maximum length 400. The `Title` attribute acts as a primary key and a foreign key referencing the `Title` in the `exercises` table. The `Tag` attribute also acts as a primary key and represents the tag associated with the exercise.

- **timelines:** This table consists of three attributes. The `TimelineTitle` attribute is a string of maximum length 400 and acts as the primary key. The `TimelineJSON` attribute is a text data type, storing the JSON data of the timeline. The `Created_At` attribute is a datetime value, recording the creation time of the timeline record.

All tables in the ”exercisesapp” database are stored in the InnoDB storage engine and use the latin1 character set.

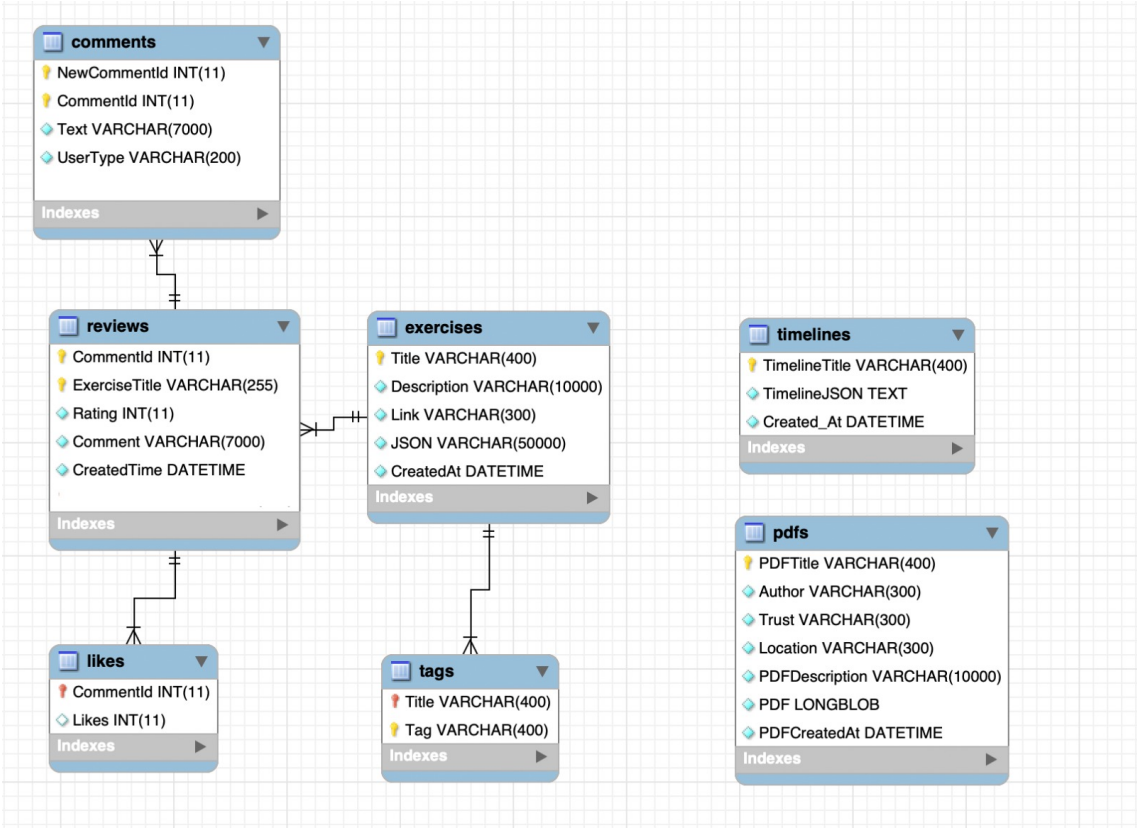

Figure 5.3: ER diagram

Chapter 6

Implementation

6.1 User Authentication Implementation

Our login system is developed using the React framework, a popular and efficient JavaScript library for building user interfaces. The code snippet provided showcases the implementation of the Login component, which serves as the entry point for user authentication.

The Login component is designed as a functional component that renders a login page with two separate login options: Google and UCL (University College London) authentication.

- **Google Authentication:** A button is displayed on the login page, which, when clicked, triggers the `google()` function.This function opens a new browser window or tab to the specified URL (`/auth/google`), directing the user to the Google authentication page.

- **UCL Authentication:** Similarly, a button for UCL authentication is provided. Clicking the button calls the `ucl()` function, which navigates the user to the UCL authentication page at (`/auth/ucl`).

Both the `google()` and `ucl()` functions utilize the `window.open()` JavaScript method, which opens a new browser window or tab with the specified URL. The "`_self`" parameter ensures that the current window or tab is replaced with the new URL.

In addition to the login buttons, the Login component also renders a visual layout, including the NHS (National Health Service) logo, a "Providing NHS Service" label, and other relevant images. The layout is styled using CSS classes such as `login-page`, `login-wrapper`, and `login-div`.

Totally, the Login component provides a straightforward and user-friendly interface for users to authenticate themselves via either Google or UCL accounts, ensuring secure access to the application. The implementation leverages React and JavaScript to render the component and handle user interactions.

```
const google = () => {
  window.open("http://localhost:5000/auth/google", "_self");
};

const ucl = () => {
  window.open("http://localhost:5000/auth/ucl", "_self");
};
```

Figure 6.1: User Authentication Implementation

In addition to the frontend Login component, our application also requires a backend implementation to handle user authentication. The backend is implemented using Node.js, an open-source, cross-platform JavaScript runtime environment that executes JavaScript code outside of a browser. For this particular implementation, we are using the Express.js framework, a popular web application framework for Node.js.

To manage authentication, we use the Passport.js middleware, which is a flexible and extensible authentication middleware for Node.js applications. Passport.js supports various authentication mechanisms, known as strategies, including Google and UCL authentication, which are used in our application.

Here is a brief explanation of the code snippet provided for the backend implementation:

- The required libraries and modules are imported, including Passport.js, Ex-

press.js, and the authentication route module (`authRoute`).

- Express-session is used to manage user sessions. A session middleware is configured with a secret key to sign the session ID cookie.
- Passport.js is initialized and configured to use session management.
- Cross-origin resource sharing (CORS) is enabled, allowing the frontend application running on `localhost:3000` to communicate with the backend server.
- The authentication route module (`authRoute`) is registered as middleware for the `/auth` route.
- The Express.js server is started, listening on port 5000 for incoming requests.

In summary, the backend implementation for the login system manages user authentication and session management, leveraging the power of Node.js, Express.js, and Passport.js. This implementation seamlessly integrates with the frontend Login component, providing a secure and efficient login process for users.

```
//needed for login
const passportSetup = require("./passport");
const passport = require("passport");
const authRoute = require("./routes/auth");
const session = require('express-session')
```

Figure 6.2: User Authentication Implementation Back end

```
//login things
app.use(session({
  secret: 'cats'
}));

app.use(passport.initialize());
app.use(passport.session());

app.use(
  cors({
    origin: "http://localhost:3000",
    methods: "GET,POST,PUT,DELETE",
    credentials: true, // allows us to send sessions via client server reqests
  })
);
app.use(cors());

app.use("/auth", authRoute);

app.listen("5000", () => {
  console.log("Server is running!");
});
```

Figure 6.3: Log in thing Back end

6.2 Upload Exercises and PDFS

One of the core features of this project is the implementation of file upload functionality, including JSON files (exercise data) and PDF files. To achieve this functionality, we created a component named *UploadModal*, which contains two sub-components: *Upload* and *UploadPDF*. Below, we will discuss the implementation of these two sub-components in detail.

6.2.1 Upload Component

The *Upload* component is responsible for uploading JSON files. In this component, we used the `useState` Hook to manage states, such as `title`, `description`, `youtubeLink`, `file`, and `tags`. We used the `useRef` Hook to obtain a reference to the file input, so we can trigger a click event on the file input. We also used the `useCallback` Hook to handle drag and drop events.

```
const onDragOver = useCallback((e) => {
  e.preventDefault();
  if (!dragging) {
    setDragging(true);
  }
}, []);

const onDragLeave = useCallback((e) => {
  e.preventDefault();
  setDragging(false);
}, []);

const onDrop = useCallback(
    (e) => {
      e.preventDefault();
      setDragging(false);
      if (e.dataTransfer.files && e.dataTransfer.files.length > 0) {
        setFile(e.dataTransfer.files[0]);
      }
    },
    []
);
```

Figure 6.4: Drag File Function

The upload form contains the following fields:

- Title
- Action description
- YouTube link
- Tags
- Upload JSON file

Upon form submission, we send the form data as a `FormData` object to the backend API, with the API address being `/main`.

```
const handleSubmit = () => {
  const formData = new FormData();
  formData.append("title", title);
  formData.append("description", description);
  formData.append("youtubeLink", youtubeLink);
  formData.append("file", file);
  formData.append("tags", JSON.stringify(tags));

  Axios.post("http://localhost:3001/main", formData).then(() => {
    console.log("Data submitted successfully");
  });
};
```

Figure 6.5: Sumbit

Additionally, we implemented a function called `checkTitleExists` to check whether the title entered by the user already exists. If it exists, we display an error message asking the user to change the title.

```
const checkTitleExists = async (titleToCheck) => {
  try {
    const response = await Axios.get("http://localhost:3001/check-title", {
      params: { title: titleToCheck },
    });
    if (response.data.exists) {
      setTitleExists(true);
      setTitleError("The Title has been occupied");
    } else {
      setTitleExists(false);
      setTitleError("");
    }
  } catch (error) {
    console.error("Error checking title:", error);
  }
};
```

Figure 6.6: Check Exist for Title

6.2.2 UploadPDF Component

The *UploadPDF* component is responsible for uploading PDF files. Similar to the *Upload* component, we used the `useState`, `useRef`, and `useCallback` Hooks to manage states and event handling.

The upload form contains the following fields:

- Title
- Author
- Trust
- Location
- Description
- Upload PDF file

Upon form submission, we send the form data as a `FormData` object to the backend API, with the API address being `/create-pdf`. We also implemented

```
const handleSubmit = () => {
  const formData = new FormData();
  formData.append("title", title);
  formData.append("author", author);
  formData.append("trust", trust);
  formData.append("location", location);
  formData.append("description", description);
  formData.append("file", file);

  Axios.post("http://localhost:3001/create-pdf", formData).then(() => {
    console.log("Data submitted successfully");
  });
};
```

Figure 6.7: Upload PDF

the `checkTitleExists` function in this component to check whether the title entered by the user already exists. If it exists, we display an error message asking the user to change the title.

```
//check whether the title has been existed
const checkTitleExists = async (titleToCheck) => {
  try {
    const response = await Axios.get("http://localhost:3001/check-title", {
      params: { title: titleToCheck },
    });
    if (response.data.exists) {
      setTitleExists(true);
      setTitleError("The Title has been occupied");
    } else {
      setTitleExists(false);
      setTitleError("");
    }
  } catch (error) {
    console.error("Error checking title:", error);
  }
};
```

Figure 6.8: Check PDF Exist for Title

6.2.3 UploadModal Component

The *UploadModal* component integrates the *Upload* and *UploadPDF* components into a single pop-up window. We used a state variable called `selected` to determine which sub-component (*Upload* or *UploadPDF*) should be displayed in the pop-up window. Users can switch the value of the `selected` state variable by clicking the icons in the pop-up window, thus switching between the two sub-components.

By integrating these two functionalities into a single pop-up window, we achieved a clean and easy-to-use user interface, allowing users to easily upload JSON and PDF files.

6.2.4 Implementation of Upload Functionality in Backend

In this subsection, we will describe the implementation of the key feature, file upload functionality, in our web application's backend. The implementation relies on the following frameworks, libraries, and plugins:

- **Express:** A popular Node.js web application framework, which simplifies the creation of a web server and handling HTTP requests and responses.

- **Multer:** A middleware for handling multipart/form-data, primarily used for uploading files.
- **MySQL:** A widely-used open-source relational database management system, utilized to store and manage the uploaded data.
- **File System (fs):** A built-in Node.js module, used to perform file system operations such as reading and writing files.
- **Path:** Another built-in Node.js module, used to handle and transform file paths.

6.2.4.1 File Upload with Multer

To handle the file uploads, we use Multer middleware in our Express app. Multer is responsible for handling the incoming multipart/form-data requests and saving the uploaded files. The following code snippet demonstrates how we configure Multer:

We create a custom storage engine using ‘multer.diskStorage‘ and define the destination and filename properties. The destination property specifies the folder where the uploaded files will be saved, while the filename property sets a unique name for each uploaded file.

6.2.4.2 Handling File Upload Route

We create an Express route for handling the file upload request:

```
// upload page posting
app.post("/create", upload.single("file"), (req, res) => {
  const { title, description, youtubeLink } = req.body;
  const file = req.file;

  if (!file) {
    res.status(400).send("Error: JSON file is required.");
    return;
  }
```

Figure 6.9: Handling File Upload Route

The ‘upload.single(”file”)‘ middleware processes the incoming request, extracts the uploaded file, and saves it to the specified destination. The ‘file‘ object is then available in the request object (‘req.file‘).

6.2.4.3 Reading and Storing JSON Content

Once the file is uploaded, we use the File System (fs) module to read the content of the uploaded JSON file:

```
// Read the file and convert it to a string
const filePath = path.join(__dirname, "./uploads", file.filename);
fs.readFile(filePath, "utf8", (err, jsonString) => {
  if (err) {
    console.log("Error reading file:", err);
    res.status(500).send("Error reading JSON file");
    return;
  }
```

Figure 6.10: Reading and Storing JSON Content

After reading the file, we convert its content into a string and insert it into the MySQL database:

```
// Insert the JSON content as a string into the database
db.query(
  "INSERT INTO exercises (Title, Description, Link, JSON) VALUES (?, ?, ?, ?)",
  [title, description, youtubeLink, jsonString],
  (err, result) => {
    if (err) {
      console.log(err);
      res.status(500).send("Error inserting value");
    } else {
      res.send("Value inserted");
    }
  }
);
```

Figure 6.11: Insert Into Database

6.2.5 Sending Tags to Database

We create another route to handle the tags associated with the uploaded files:

```
// send tags to database
app.post("/add-tags", async (req, res) => {
  const { title, tags } = req.body;

  // Function to insert a single tag into the database
  const insertTag = (title, tag) => {
    return new Promise((resolve, reject) => {
      db.query(
        "INSERT INTO tags (Title, Tag) VALUES (?, ?)",
        [title, tag],
        (err, result) => {
          if (err) {
            reject(err);
          } else {
            resolve(result);
          }
        }
      );
    });
  };
```

Figure 6.12: Sending Tags to Database

The ‘insertTag‘ function is used to insert a single tag into the MySQL database

6.3 Exercise Interface Implementation

The `Exercise` component is responsible for fetching and displaying the exercise details, including exercise name, description, video link, tags, image URL, and comments. It also allows users to submit their comments, rate the exercise, and like other comments.

6.3.1 Fetching Exercise Data

We use the `useLocation` hook to retrieve the `exerciseTitle` from the passed state. Next, we utilize the `React.useEffect` hook to fetch data for the specific exercise using an asynchronous function named `fetchData`. Inside

this function, we make an HTTP GET request using Axios to the `/exercises/${exerciseTitle}` endpoint to retrieve the exercise data.

Once we receive the data, we use the `setExerciseName`, `setExerciseDescription`, `setExerciseVideoLink`, `setExerciseTags`, and `setExerciseImageUrl` functions to update the respective state variables.

```
// Fetch data for the specific exercise
React.useEffect(() => {

  const fetchData = async () => {
    try {
      const response = await axios.get(`http://localhost:3001/exercises/${exerciseTitle}`);
      const data = response.data;

      if (data) {
        setExerciseName(data.title);
        setExerciseDescription(data.description);
        setExerciseVideoLink(data.link);
        setExerciseTags(data.tags.join(', '));
        const thumbnailUrl = getVideoThumbnailUrl(data.link);
        if (thumbnailUrl) {
          setExerciseImageUrl(thumbnailUrl);
        }
      } else {
        console.log("No exercise data found");
      }
    } catch (error) {
      console.log(error);
    }
  };
  fetchData();
}, [exerciseTitle]);
```

Figure 6.13: Fetching Exercise Data

6.3.2 Fetching, Displaying, and Submitting Comments

We use another `React.useEffect` hook to fetch comments associated with the exercise by calling the `fetchComments` function. This function sends an HTTP GET request to the `/reviews/${exerciseTitle}` endpoint to retrieve the comments data. The fetched comments are then processed and stored in the `commentsList` state variable. Users can submit their comments using the `handleSubmitComment` function. This function sends an HTTP POST request

```
const fetchComments = async () => {
  try {
    const response = await axios.get(`http://localhost:3001/reviews/${exerciseTitle}`);
    const commentsData = response.data;

    const processedCommentsData = commentsData.map(comment => ({
      ...comment,
      CreatedTime: new Date(comment.CreatedTime),
    }));

    setCommentsList(processedCommentsData);
    const totalRatings = processedCommentsData.reduce((total, comment) => total + Number(comment.rating), 0);
    setAverageRating(processedCommentsData.length > 0 ? totalRatings / processedCommentsData.length : 0);

    // Set initial likeCounts and likedComments states
    const initialLikeCounts = processedCommentsData.reduce((acc, comment, index) => {
      acc[index] = comment.likes;
      return acc;
    }, {});
    setLikeCounts(initialLikeCounts);
    setLikedComments({});
    setOriginalCommentsList(processedCommentsData);
  } catch (error) {
    console.log(error);
  }
};
```

Figure 6.14: Fetch Comment

to the `/reviews` endpoint with the `exerciseTitle`, `rating`, and `comment` data. The newly submitted comment is then added to the `commentsList` state.

```
const handleSubmitComment = async () => {
    try {
        const response = await axios.post('http://localhost:3000/reviews', {exerciseTitle, rating, comment });
        const id = response.data.id;
        const newComment = {
            id,
            text: comment,
            rating: rating,
            date: new Date(),
        };
        setCommentsList([...commentsList, newComment]);
        setComment("");
        setRating(0);
        const newAverageRating =
            (averageRating * commentsList.length + rating) /
            (commentsList.length + 1);
        setAverageRating(newAverageRating);
    } catch (error) {
        console.log(error);
    }
```

Figure 6.15: Fetch Comment

6.3.3 Liking Comments

Users can like comments using the `handleLikeComment` function. This function updates the `likedComments` and **likeCounts** state variables based on the index of the comment being liked or unliked. It then calls the `updateLikeCount` function to send an HTTP POST request to the `http://localhost:3001/likes` endpoint to update the like count for the specific comment.

6.3.4 Filtering and Searching Comments

The `Exercise` component provides functionality for filtering comments based on their rating using the `handleBarClick` function. When a rating bar is clicked, this function sends an HTTP GET request to the `http://localhost:3001/reviews/${exerciseTitle}` endpoint with the `rating` parameter to fetch comments with the specified rating. The fetched comments are then processed and stored in the `commentsList` state variable.

```
const handleLikeComment = (index) => {
  setLikedComments((prevState) => ({
    ...prevState,
    [index]: !prevState[index],
  }));

  const newLikeCount = likeCounts[index]
    ? (likedComments[index] ? likeCounts[index] - 1 : likeCounts[index] + 1)
    : 1;

  setLikeCounts((prevState) => ({
    ...prevState,
    [index]: newLikeCount,
  }));

  // Get the current likeCounts state before making the API call
  const currentLikeCounts = { ...likeCounts, [index]: newLikeCount };
  updateLikeCount(commentsList[index].CommentId, currentLikeCounts[index]);
};

const updateLikeCount = async (commentId, newLikeCount) => {
    try {
      await axios.post(`http://localhost:3001/likes`, { commentId, newLikeCount });
    } catch (error) {
      console.log(error);
    }
  };

  const handleLikeComment = (index) => {
    setLikedComments((prevState) => ({
      ...prevState,
      [index]: !prevState[index],
    }));

    const newLikeCount = likeCounts[index]
      ? (likedComments[index] ? likeCounts[index] - 1 : likeCounts[index] + 1)
      : 1;

    setLikeCounts((prevState) => ({
      ...prevState,
      [index]: newLikeCount,
    }));

    // Get the current likeCounts state before making the API call
    const currentLikeCounts = { ...likeCounts, [index]: newLikeCount };
    updateLikeCount(commentsList[index].CommentId, currentLikeCounts[index]);
  };
```

Figure 6.16: Handle and Update Like

```
const handleBarClick = async (rating) => {
  if (selectedBar === rating) {
    setSelectedBar(null);
    setCommentsList(originalCommentsList);
    return;
  }
  setSelectedBar(rating);

  try {
    const response = await axios.get(`http://localhost:3001/reviews/${exerciseTitle}`, { params: { rating } });
    const commentsData = response.data;

    const processedCommentsData = commentsData.map((comment) => ({
      ...comment,
      CreatedTime: new Date(comment.CreatedTime),
    }));

    setCommentsList(processedCommentsData);
    setTimeout(() => {
      scrollToCommentsList();
    }, 100);
  } catch (error) {
    console.log(error);
  }
};
```

Figure 6.17: Handle Bar Click Function

Users can also search for comments using the `executeSearch` function. This function sends an HTTP GET request to the `/reviews/search/${exerciseTitle}/${inputSearchTerm}` endpoint to search for comments containing the search term. The search results are then stored in the `filteredCommentsList` state variable.

```
const executeSearch = async (event) => {
  event.preventDefault(); // Prevent default form submission behavior

  if (inputSearchTerm === "") {
    setFilteredCommentsList([]);
  } else {
    try {
      const response = await axios.get(
        `http://localhost:3001/reviews/search/${exerciseTitle}/${inputSearchTerm}`
      );
      const commentsData = response.data;

      const processedCommentsData = commentsData.map((comment) => ({
        ...comment,
        CreatedTime: new Date(comment.CreatedTime),
      }));

      setFilteredCommentsList(processedCommentsData);
    } catch (error) {
      console.log(error);
    }
  }
};
```

Figure 6.18: Searching Comments

6.3.5 Sharing Exercise and Copying Video Link

The `Exercise` component allows users to share the exercise via a popup that is toggled using the `toggleSharePopup` function. Users can copy the YouTube video link to their clipboard using the `copyToClipboard` function, which utilizes the `navigator.clipboard.writeText` method.

```
// for share popup
const [showSharePopup, setShowSharePopup] = useState(false);
const toggleSharePopup = () => {
  setShowSharePopup(!showSharePopup);
};


// Copy YouTube link to clipboard
  const copyToClipboard = () => {
      navigator.clipboard.writeText(exerciseVideoLink).then(
      () => {
          console.log('Link copied to clipboard');
      },
      (err) => {
          console.error('Could not copy link to clipboard: ', err);
      }
      );
  };
```

Figure 6.19: Sharing Exercise and Copying Video Link

6.3.6 Smooth Scrolling

The **Exercise** component provides smooth scrolling functionality to various sections of the page. The `scrollToCommentSection`, `scrollToStarBars`, and `scrollToCommentsList` functions are used to scroll smoothly to the comment section, star bars, and comments list, respectively. These functions use the `window.scrollTo` method with the behavior option set to 'smooth'.

```
// Function to scroll to the comment section
const scrollToCommentSection = () => {
    const commentSectionTop = commentSectionRef.current.getBoundingClientRect().top + window.pageYOffset;
    window.scrollTo({ top: commentSectionTop, behavior: 'smooth' });
  };

  const scrollToStarBars = () => {
    const starBarsTop = starBarsRef.current.getBoundingClientRect().top + window.pageYOffset;
    window.scrollTo({ top: starBarsTop, behavior: 'smooth' });
  };

const scrollToCommentsList = () => {
    if (commentListRef.current) {
        const commentsListTop = commentListRef.current.getBoundingClientRect().top + window.pageYOffset;
        window.scrollTo({ top: commentsListTop, behavior: 'smooth' });
    }
};
```

Figure 6.20: Smooth Scrolling

6.4 Core Functions Implementation In Main

6.4.1 Email Sending Functionality

We implemented an asynchronous function called `sendEmail` to enable the email sending feature. This function takes two arguments: an email address (`email`) and email content (`content`).

- First, we try to send an HTTP POST request to a local server endpoint at `/send-email` using the Fetch API. The request headers include 'Content-Type': 'application/json', and the request body contains the JSON stringified data of `email` and `content`.
- If the request is sent successfully and the response status is 'ok', we output a message using `console.log` indicating that the email was sent successfully. The function returns `true`.
- If the response status is not 'ok', we output an error message using `console.error` indicating that there was an error while sending the email. The function returns `false`.
- If an error occurs while sending the request, we catch the error and output a message using `console.error` indicating that there was an error while

sending the email. The function returns `false`.

```
// Call saveTemplate and get the saved timeline JSON
try {
  const savedTimelineJson = await saveTemplate(timelineTitle);

  const emailContent = {
    timelineTitle,
    timelineJson: JSON.stringify(savedTimelineJson),
    notes,
  };

  // Call the sendEmailToPatients function to send the email to patients
  await sendEmail(patientEmails, emailContent);

  // Display success message
  setOperationStatus({
    type: "success",
    message: "Email sent successfully",
  });
```

Figure 6.21: Send Email

6.4.1.1 Email Address Validation

To ensure that our target address for sending emails is a valid email address, we use a function called `validateEmail` to validate it. This function takes an email address as an argument and checks whether it conforms to the rules of a valid email address using a regular expression. If it is valid, it returns `true`; otherwise, it returns `false`.

```
function validateEmail(email) {...}
```

Figure 6.22: Validate Email

6.4.2 Timeline Implementation

In the provided code, we implemented a Timeline feature that allows users to manage their exercises more efficiently. The Timeline is designed to display exercises in chronological order, and users can interact with the Timeline to modify the order and repetition of exercises.

6.4.2.1 Selecting and Ordering Exercises

```
const handleExerciseNameClick = (exercise) => {
  const isSelected = selectedExercises.find(
    (selectedExercise) => selectedExercise.exerciseTitle === exercise.title
  );

  if (isSelected) {
    setSelectedExercises(
      selectedExercises.filter(
        (selectedExercise) => selectedExercise.exerciseTitle !== exercise.title
      )
    );
  } else {
    const newExercise = {
      exerciseTitle: exercise.title,
      repetitions: 1,
      exerciseLink: exercise.link,
    };

    setSelectedExercises([...selectedExercises, newExercise]);
  }
};
```

Figure 6.23: Select Exercise

We use the `selectedExercises` state to store an array of selected exercises. Whenever the user selects an exercise, the exercise is added to the `selectedExercises` array. To enable the user to change the order of exercises within the Timeline, we implemented a drag-and-drop functionality.

To achieve this, we can use a library such as react-dnd (React Drag and Drop) or react-beautiful-dnd. With these libraries, we can wrap our exercise components in draggable containers and define drop zones within the Timeline. This allows users to click and drag exercises to rearrange their order within the Timeline. When

the user drops an exercise in a new position, we update the `selectedExercises` array accordingly, ensuring that the new order is reflected in the application state.

6.4.2.2 Adjusting Repetition

In addition to rearranging the order of exercises, users can also adjust the repetition of exercises within the Timeline. To implement this feature, we can add a repetition input field to each exercise component within the Timeline. Users can input the desired number of repetitions for each exercise.

We can store the repetition count as a property of each exercise object within the `selectedExercises` array. When the user updates the repetition count, we update the corresponding exercise object in the `selectedExercises` array.

```
Repetitions:
<div className="repetition-selector">
<button
  className="minus-button"
  onClick={() => {...}}
...>

  <input
    type="number"
    value={exercise.repetitions}
    onChange={(e) => {...}}
    style={{ width: '30px' }} // You can adjust the width to your preference
  />
  <button
    className="plus-button"
    onClick={() => {...}}
  >
    +
  </button>

</div>
```

Figure 6.24: Adjusting Repetition

Overall, we implemented a Timeline feature that allows users to manage their exercises effectively. Users can select exercises, modify their order within the Timeline using a drag-and-drop functionality, and adjust the repetition count for

each exercise. We used React and additional libraries, such as react-dnd or react-beautiful-dnd, to create an interactive and user-friendly interface for managing exercises within the Timeline.

6.4.3 Filtering and Sorting Items

We also provided filtering and sorting functionality for the items in the application. For this, we use the `filteredItems` variable to store the filtered items. The filtering process is done by checking whether the `title` property of the item contains the `searchTerm`. The sorting process is based on the chosen `sortMethod`, such as sorting alphabetically (A-Z) or by creation date (newest items first).

```
const filteredItems = selectedItems
  .filter((item) => {
    const stringToFilter = item.title; // Access the 'title' property
    if (typeof stringToFilter !== "string") {
      console.error("Non-string item found:", item);
      return false;
    }
    return stringToFilter.toLowerCase().includes(searchTerm.toLowerCase());
  })
  .sort((a, b) => {
    switch (sortMethod) {
      case "A-Z":
        return a.title.localeCompare(b.title);
      case "Newest":
        return new Date(b.createdAt) - new Date(a.createdAt);
      // Add more sorting options if needed
      default:
        return 0;
    }
  });
```

Figure 6.25: Filter

```
const handleSortMethodChange = (event) => {
  const sortValue = event.target.value;
  console.log("Selected sort method:", sortValue);
  if (activeTabIndex === 0) {
    setSortMethod(sortValue);
  } else {
    setSortMethodTemplates(sortValue);
  }
};
```

Figure 6.26: Sort

6.4.4 Pagination Functionality

To provide a better user experience, we implemented pagination functionality. This way, we can display a smaller number of items at a time instead of showing all items on the page. We do this by using the `paginatedItems` variable to store the items for the current page and the `itemsPerPage` variable to set the number of items to display per page. We use the `slice` method to extract the items for the current page from the filtered items list.

Similarly, we implemented pagination functionality for the template data using the `paginatedTemplates` variable and the `templatesPerPage` variable.

```
{/* Pagination */}
  <div className="pagination-container">
    <Paginate
      pageCount={Math.ceil(filteredItems.length / itemsPerPage)}
      onPageChange={handlePageChange}
      containerClassName={'custom-pagination'}
      activeClassName={'active'}
      forcePage={currentPage}
      previousLabel={"Prev"}
      nextLabel={"Next"}
      breakLabel={null}
      pageRangeDisplayed={0}
      marginPagesDisplayed={0}
    />

  </div>
```

Figure 6.27: Pagination

6.4.5 Searching and Sorting Templates

We also implemented searching and sorting functionality for the template data. For this, we created the `filteredTemplates` variable to store the filtered template data. The filtering process is done by checking whether the `TimelineTitle` property of the template contains the `searchTextTemplates`. We also implemented a function called `handleSearchTextTemplatesChange` that updates the `searchTextTemplates` variable when the user inputs search text.

6.4.6 Handling User Interface Interactions

We provided some functions to handle interactions with the user interface, such as changing the sorting method, pagination, and searching. These functions are:

- `handleSortMethodChange`: When the user selects a sorting method, this function updates the `sortMethod` or `sortMethodTemplates` variable (depending on the currently active tab).

- `handlePageChangeTemplates`: When the user switches to another page, this function updates the `currentPageTemplates` variable.
- `handleSearchTextTemplatesChange`: When the user inputs search text, this function updates the `searchTextTemplates` variable.

6.4.7 Youtube Thubnail

In the given code, we also provided other features such as handling selected exercises and templates, displaying YouTube video thumbnails using the `YoutubeThumbnail` component, and more. These functionalities are aimed at enhancing the user experience and functionality of the application.

```
const YoutubeThumbnail = ({ videoUrl }) => {
  function extractVideoId(link) {
    const videoIdMatch = link.match(
      /(?:youtube\.com\/(?:[^/]+\/[^/]+\/|(?:v|e(?:mbed)?)\/|.*[?&]v=)|youtu\.be\/)([^"&?/ ]{11})/i
    );
    return videoIdMatch ? videoIdMatch[1] : null;
  }

  const videoId = extractVideoId(videoUrl);
  const thumbnailUrl = `https://img.youtube.com/vi/${videoId}/0.jpg`;

  return (
    <img
      src={thumbnailUrl}
      alt={`Thumbnail for ${videoUrl}`}
      style={{ width: "100%", height: "100%", objectFit: "cover" }}
    />
  );
};
```

Figure 6.28: Thumbnail

In summary, we used React, Axios, and other libraries to create a feature-rich application, including key functionalities like sending emails, validating email addresses, filtering and sorting items, paginating data, handling user interface interactions, and more. We provide detailed documentation and code snippets to help other developers understand our implementation.

Chapter 7

Testing

7.1 Testing Strategy

This section outlines the test strategy designed for our app. We will perform the following tests to ensure the quality and performance of the application:

7.1.1 Unit and Integration Testing

To ensure robustness and reliability of our web app, throughout the development process, we continuously integrated unit tests, enabling early detection and resolution of any issues. This approach not only minimized the risk of bugs but also improved the overall quality of the application. By breaking down the application into separate components and understanding their inputs, outputs, and expected behaviors, we were able to create test cases that addressed various scenarios, including edge cases and error handling.We then executed these tests manually, analyzing the results and making adjustments as necessary.

Testing Framework

We additionally wrote test scripts and utilised Jest framework and partly the React Testing Library. Our test scripts can be found in our source code under src/page/<ComponentName>.test.js.

Unit and Integration Testing

Multiple unit tests were carried out for the majority of the pages. Specifically, for

the Login.js page we tested:

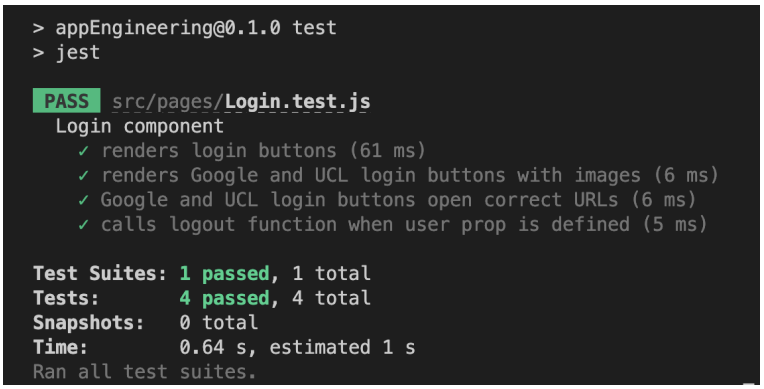

Figure 7.1: Login page tests

For Home.js we tested:

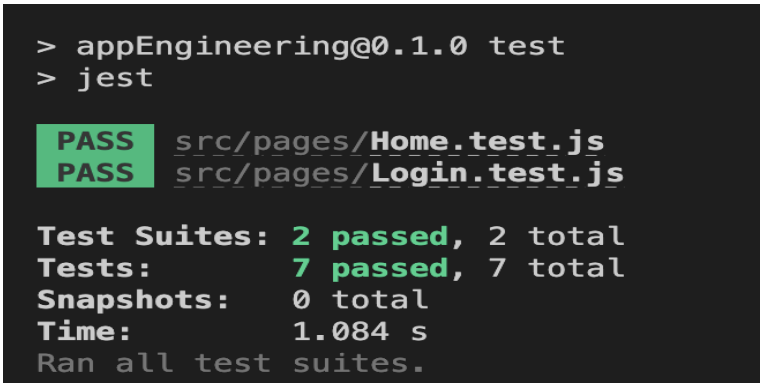

Figure 7.2: Home page tests

For Exercise.js:

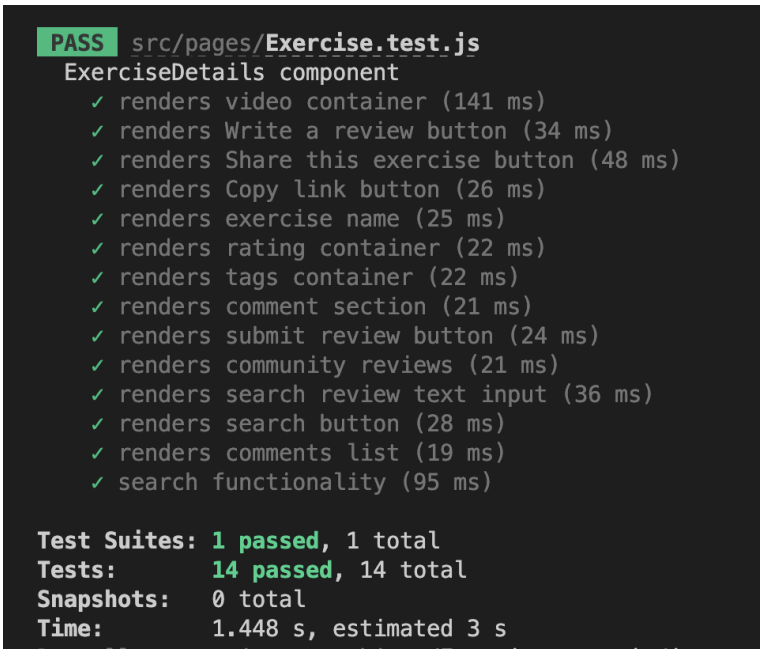

Figure 7.3: Exercise page tests

In our React application, we wrote unit tests for various components, utility functions, and hooks. These tests focused on ensuring that each unit behaves as expected, handles edge cases correctly, and produces the expected output given specific input or state.

Example unit test:

```
test('renders Google and UCL login buttons with images', () => {
  render(
      <BrowserRouter>
        <Login user={null} logout={mockLogout} />
      </BrowserRouter>
    );
  expect(screen.getByText('Google')).toBeInTheDocument();
  expect(screen.getByText('UCL')).toBeInTheDocument();
  // expect(screen.getByAltText('')).toHaveLength(2);
  expect(screen.getByAltText('Google login icon')).toBeInTheDocument();
  expect(screen.getByAltText('UCL login icon')).toBeInTheDocument();

});
```

Figure 7.4: Example unit test from Login.js

We also used integration tests to examine how different parts of the system work together and verify that they integrate correctly, and test the flow of data and state across the application. We wrote integration tests for scenarios where multiple components interact with each other, API requests and responses, and user interactions.

These tests ensure that our application functions correctly as a whole and that different parts work seamlessly together.

Example integration test:

```
test('search functionality', async () => {
  // Mock the API call for searching comments
  const searchResult = [
  ];
  const history = createMemoryHistory();

  const mockAxios = new MockAdapter(axios);
  mockAxios
    .onGet('http://localhost:3002/api/reviews/search/exerciseTitle/searchInput')
    .reply(200, searchResult);

  mockAxios
    .onGet('http://localhost:3002/api/reviews/')
    .reply(200, []);

  mockAxios
  .onGet('http://localhost:3002/api/exercises/')
  .reply(200, []);

  render(
      <BrowserRouter>
        <Exercise />
      </BrowserRouter>
      );

  // Find the search input and type a search term
  const searchInput = waitFor(() => screen.getByTestId('search-input'));

  waitFor(() => userEvent.type(searchInput, 'searchTerm'));

  // Check if the state updates correctly
  waitFor(() => expect(searchInput.value).toBe('searchTerm'));
  // Find the search button and click it
  const searchButton =  waitFor(() =>screen.getByTestId('search-button'));
  waitFor(() => fireEvent.click(searchButton));
  // Wait for the API call to complete and the state to update
  waitFor(() => expect(screen.getByText('searchResultComment')).toBeInTheDocument());
});
});
```

Figure 7.5: Example integration test from Exercise.js

Overall our development workflow involved continues unit and integration testing to ensure that our application remains stable and reliable throughout the development lifecycle.

7.1.2 Responsive Design Testing

To ensure that our APP displays correctly on various devices, we will perform responsive design testing for at least 3 screen sizes:

- Laptop: 13.3 in, 14 in, 16in
- Tablet: width 768px
- Mobile phone: width 480px

The first step in the responsive design testing process involved a thorough examination of the application's source code, specifically the CSS and HTML structure. Our objective was to identify components that required adjustments to enhance responsiveness, ensuring that the application caters to different devices and screen sizes.

Next, we implemented media queries to target specific breakpoints and modify the CSS properties accordingly. This approach enabled the layout and design to adapt seamlessly to various screen sizes and devices, providing an optimal user experience for all users, regardless of the device they access the application on.

Finally, we conducted extensive testing of the application on a wide range of devices, screen sizes, and browsers to ensure consistent behavior and appearance across all platforms. This was achieved by utilizing browser developer tools and device emulators, as well as performing tests on actual devices. This comprehensive testing process allowed us to identify and address any potential issues, ensuring that the application's responsiveness meets the highest standards and provides an exceptional user experience.

7.1.3 Compatibility Testing

In the compatibility testing section of our report, we focused on ensuring that the application functions as expected across various platforms, browsers, and devices. This is crucial for providing an accessible and consistent user experience, regardless of the user's device or browser preferences.

To achieve this, we performed the following steps:

Identifying target platforms and browsers: We determined the most popular browsers and platforms that our target audience uses, including Chrome, Firefox, Safari, and Edge, as well as mobile platforms such as iOS and Android. This information helped us prioritize our testing efforts and allocate resources efficiently.

Creating test cases: We developed a comprehensive set of test cases that covered various aspects of the application, including layout, design, functionality, and performance. These test cases ensured that we systematically checked every aspect of the application for compatibility issues.

Testing on multiple platforms and browsers: We utilized a combination of physical devices and browser emulators to test the application on different platforms and browsers. This allowed us to identify and address any inconsistencies or issues that arose due to specific browser or platform configurations.

Addressing compatibility issues: As we encountered compatibility issues (mostly CSS related) during testing, we documented the problem and worked closely with the development team to implement appropriate solutions. This iterative process helped us to identify and resolve issues more efficiently, ensuring that the application functioned smoothly across all platforms and browsers.

Re-testing and validation: After addressing the identified compatibility issues, we re-tested the application to validate that the implemented solutions were effective. This step ensured that all identified issues were resolved, and the application was ready for deployment.

By carrying out this comprehensive compatibility testing process, we were able to ensure that the application functions consistently and provides an exceptional user experience across a wide range of devices, platforms, and browsers.

7.1.4 Stress Testing

In the stress testing section of our report, we assessed the application's performance and stability under various conditions to ensure a consistent user experience. Our goal was to identify any potential issues that might arise under increased load or usage and address them accordingly.

The following steps were taken to conduct stress testing for the application:

Identifying key areas for testing: We focused on the critical components of the application that required stress testing, such as user interactions, response times, and resource usage. Creating simple test scenarios: We developed test scenarios simulating increased user traffic or heavy workloads, ensuring that our testing efforts were relevant to real-world situations. Executing stress tests: Using basic testing tools and techniques, we applied increased load to the application, monitoring its performance and stability throughout the process. Analyzing test results: We reviewed the stress testing results to identify any issues or areas for improve-

ment, enabling us to address potential bottlenecks or performance degradations. Implementing improvements: Working with the development team, we made any necessary adjustments to optimize the application's performance and stability under stress. Re-testing and validation: We conducted additional stress tests to confirm the effectiveness of the changes and ensure the application could handle increased load without issues.

By following this simplified stress testing process, we verified that the application could maintain its performance and stability under various conditions, providing a consistent and reliable user experience.

7.1.5 User Acceptance Testing

User Acceptance Testing (UAT) was conducted to ensure that the application met end-users' needs and expectations. The process involved defining acceptance criteria, preparing test scenarios, and recruiting a diverse group of test participants. We carried out UAT sessions where users interacted with the application and provided feedback on its usability, functionality, and overall experience. The test results and feedback were analyzed to identify areas for improvement. Necessary adjustments were made, and additional UAT sessions were conducted to confirm the changes' effectiveness. This streamlined approach ensured a successful deployment and a positive user experience.

Specifically for the main page, Users reported that the "Select All" checkbox in the "Templates" modal was not working properly.We fixed the issue with the checkbox and made it functional. Same happened with our pagination, it used to create infinite amount of pages but now it's working properly. Users requested a way to sort exercises and templates in alphabetical order and so we added that feature.

7.1.6 Test Reporting

For each test, we will explain why we perform this test, which test tool we use, how we conduct the test, what results we obtain, and our analysis or conclusion of the results.

By executing the above test strategy, we will ensure that our APP achieves the highest standards in terms of functionality, performance, and usability, providing an exceptional experience for users.

Chapter 8

Conclusion and Future Work

8.1 Summary of achievements

Overall, our project maintained a high level of integrity and success, but due to staff shortages and time constraints, we focused more on functional implementation as our main goal. Due to our shortage of staff, we have always maintained a high level of cooperation and everyone has contributed spontaneously to the project in an outstanding way. Overall we have completed all the features but some of them are not yet optimised for optimal experience, but this does not detract from the fact that we have worked hard to make a successful app.

Table 8.1: Achievement table

ID	Requirement	Priority	State	Contributor
R1	Patients can access the Web App without login and rate exercises/Templates.	Must	Done	Xingrui
R2	Patients can leave comments on each exercise or templates.	Must	Done	Xingrui
R3	Clinicians can login using Google or UCL authentication.	Must	Done	Najma
R4	Clinicians can select exercises from a gallery of exercises (from our database) and create programs for their patients.	Must	Done	Najma

R5	Clinicians are able to drag selected exercises and rearrange the order of their program.	Should	Done	Najma
R6	Clinicians can adjust the repetitions for each exercise and the frequency the program should be performed.	Should	Done	Kyria
R7	Clinicians are able to paste in a text field the emails of up to 50 patients and forward the program they created in an email	Must	Done	Kyria
R8	Clinicians have the option of saving a program as a template without having to send it to patients.	Could	Done	Najma
R9	Clinicians are able to filter exercises based on specific filters such as Joint/Region, Movement and Condition.	Must	Done	Xingrui
R10	Clinicians should be able to search exercises based on exercise name, description, and tag name.	Should	Done	Kyria
R11	Clinicians are able to search templates based on timeline name, exercises, and tag name.	Should	Done	Kyria
R12	Clinicians have the ability to search though the library tab based on Name, Trust, Location, Date and Version.	Must	Done	Kyria
R13	Clinicians are able to upload JSON files of movements that would go in our database, write a description and have the ability to add tags.	Must	Done	Xingrui

R14	Clinicians are able to upload PDF files, write a description and have the ability to add trust author etc.	Must	Done	Xingrui
R15	Clinicians are able to view and edit templates.	Could	Done	Najma
R16	Clinicians are able to view the description and watch a YouTube video of every exercise.	Should	Done	Xingrui
R17	A secondary app is able to retrieve certain information from our database.	Must	Done	Kyria
R18	The Web App is responsive.	Must	Partly Completed	Najma
R19	User information on both Clinician and patient side is not stored.	Must	Done	Kyria
Key Functionalities (Must have and Should Have)			100% Completed	
Optional Functionalities (Could Have)			100% Completed	

Table 8.2: Known Bug List

ID	Bug Description	Priority
1	Some display issues in a responsive interface.	Low
2	About the different component sizes in Timeline.	Low
3	Pagginate is displayed in different Page.	Low
4	After uploading, the window needs to be manually closed again and the page refreshed.	Low

Table 8.3: Individual Contribution Table

Work Package	**Kyriaki Charalambous**	**Najma Haji**	**Xingrui Gu**
Client liaison	33.34%	33.33%	33.33%
Requirement Analysis	33.34%	33.33%	33.33%
Research	33.34%	33.33%	33.33%
User Interface Design	33.34%	33.33%	33.33%
Coding	33.34%	33.33%	33.33%
Testing	33.34%	33.33%	33.33%
Bi-weekly report	33.34%	33.33%	33.33%
Poster Design	33.34%	33.33%	33.33%
Project Report	33.34%	33.33%	33.33%
Video Editing	33.34%	33.33%	33.33%
Overall Contribution	33.34%	33.33%	33.33%
Roles	Team leader, Back End Developer, Client liaison, bi-weekly report editor, Researcher	Back End Developer, Client liaison, Researcher, bi-weekly report editor	Front End Developer, Researcher, UI Designer, bi-weekly report editor, Final Report Editor

8.2 Critical evaluation of the project

Overall, our team has delivered a relatively satisfactory product to our clients. Throughout the user interface design, we have adopted a clean and user-friendly layout, which reduces the learning curve for our target audience, enabling them to navigate through different pages effortlessly based on their existing habits. This achievement is attributed to our iterative design of interface prototypes, which have been constantly refined to meet client needs, as well as the valuable feedback from experts and users regarding the appearance of our application. Maintaining a customer-centric approach and involving clients in the design process is a crucial foundation for ensuring the success of our product. However, as we lack experience in professional application software development, there are numerous areas in which we still need to improve.

8.2.1 User Interface design and user experience

In terms of user experience details, we have incorporated hover states in a multitude of buttons throughout the interface. Hover states employ visual effects to provide users with feedback, navigation assistance, and a reduction in erroneous actions. However, in certain overlooked areas, we might have inadvertently omitted the implementation of hover states in our UI design. In our project, we have not been able to maintain a high degree of uniformity in the UI due to some objective factors, but have achieved similarity. To further optimize the UI and user experience, we may consider the following aspects:

1. **Consistency**: Ensuring a uniform style and operational logic throughout the interface can facilitate users in adapting to and mastering the application's usage more rapidly. This involves maintaining congruity in design elements, such as typography, color schemes, and interactive components.

2. **Simplicity**: Minimizing superfluous elements and functionalities can render the interface more streamlined, enabling users to swiftly locate the required information and perform relevant actions. This can be achieved by adhering to a minimalist design approach, prioritizing content hierarchy, and employing white space effectively.

3. **Customizability**: Offering users a certain degree of customization options, such as altering theme colors or font sizes, allows them to tailor the interface according to their preferences and requirements. This enhances user satisfaction and promotes a sense of ownership.

4. **Enhanced feedback**: In addition to hover states, other methods of providing users with operational feedback can be explored, such as click animations, audio cues, or haptic feedback. These can augment the user's interactive experience and bolster user engagement.

5. **Adaptability**: Contemplating the adaptability of the interface to different devices and screen sizes is crucial in ensuring that the UI maintains excellent

usability and visual appeal across various contexts. This involves utilizing responsive design techniques, flexible layouts, and scalable UI elements.

Not only in these general aspects, but also from the beginning of our design concept, we aimed to create a platform that could accommodate various user groups. This means that we needed to consider a multitude of accessibility design principles. As a result, we did not conduct user testing for details such as fonts and themes. Moreover, in our Exercise detail interface, which was primarily designed for patients, we did not carry out any design testing or gather feedback from patients. After patients leave comments, we did not design a comment messaging feature to allow them to share messages with one another. This is in contrast to our decision to enable non-logged-in users to access our platform, which has enhanced the technical implementation.

In the detail interface, we can visibly notice a certain degree of delay in page loading, especially in the YouTube video module. In the main interface, we should consider integrating the pagination display with the page number input, but this idea was not implemented due to technical and practical reasons. On the clinician side, the presentation of Exercise Repetition in the timeline is worth reconsidering, and the size of each Exercise Timeline block should be kept consistent.

Even though our interface has many flaws from a user perspective, the majority of these issues are due to the challenges brought by the implementation of backend code technology.

8.2.2 Functionality

Most of our requirements based on the MoSCoW list are almost complete. All of the requirements of Must and key features in our MoSCoW list have already been implemented, there are a few additional features that would enhance our application and user experience that could be added to our platform development, but overall prioritising our user requirements by using MoSCoW requirements analysis is an app that develops open line solutions for users in a limited time frame Effective method. The results of the tests showed that most of the functionality worked.

8.2.3 Stability

To improve the stability of our application, we have implemented comprehensive error handling mechanisms. This includes input validation checks, preventing invalid data from being processed, and displaying informative error messages to guide users in rectifying mistakes. Additionally, we have employed thorough testing methodologies, such as unit tests, integration tests, and stress tests, to identify and resolve any underlying issues before deployment. By proactively addressing potential errors and bugs, we have enhanced the overall stability and reliability of our application.

8.2.4 Efficiency

In terms of efficiency, we have made conscious decisions to optimize our application's performance. For instance, we have designed our database structure with scalability and speed in mind, utilizing indexing, caching, and query optimization techniques to minimize latency and enhance responsiveness. On the front-end, we have employed modern web development frameworks, such as React, to ensure a smooth and responsive user interface. Furthermore, we have utilized code minification, image optimization, and lazy loading techniques to decrease loading times and improve the overall user experience.

8.2.5 Compatibility

To guarantee compatibility across various devices and screen sizes, we have embraced responsive design principles. Our application's layout, typography, and UI elements have been designed to adapt seamlessly to different screen resolutions and orientations, ensuring a consistent and enjoyable experience for all users, regardless of their device. By employing flexible layouts, fluid grids, and media queries, we have ensured that our application's interface remains visually appealing and functional across a wide range of devices, from let to desktop computers. However,

we did not test all screen sizes and browser platforms in our responsive testing and given the future of mobile digital health, the main application scenario for this platform in the future should be Tablet rather than PC so at this point we need to further improve our testing and do full and comprehensive testing to determine the full compatibility of the application. However, given the time and number of people on the project, we are generally satisfied with our compatibility so far.

8.2.6 Maintainability

With the component-based design of the different interfaces and functions, our application is very easy to maintain and update, but the fragmentation between the different components requires a lot of reading and understanding time costs for further development by people who are not familiar with the code in the future. (BackEnd). And our platform is hosted using Git and Azure servers, a combination that means our backend is very easy to maintain.

8.2.7 Project Management

Our project employed efficient project management methods, which greatly facilitated internal communication and collaboration within our team. To achieve this, we utilized remote working tools such as WhatsApp groups and Notion, which not only enhanced our workflow but also enabled a more inclusive environment for our team member with a disability to participate effectively. The bi-weekly reports played a pivotal role in promoting efficient work and project management. These regular updates allowed us to stay on track, focus on deadlines, and effectively allocate tasks to team members.

Additionally, we capitalized on each individual's strengths for better personnel management, allowing team members to excel in their respective areas. This approach encouraged the development of each person's skills, and in retrospect, we can see that everyone continuously strived to improve their abilities. However, due to our general lack of experience in application development, we spent a consider-

able amount of time learning new languages and frameworks.
In conclusion, our project management was outstanding, as it facilitated effective communication, collaboration, and skill development among team members. Despite the challenges we faced, the overall management strategy contributed significantly to the success of our project.

8.3 Future Work

Through analyzing user experiences of patients and clinicians, as well as continuously gathering feedback from our target audience, we can identify several areas for improvement in our platform. Here are seven valuable ideas our team would consider if more time were available:

1. Firstly if we had extra ample time, we would first refine the presentation of all our features, such as responsive design, detailed UI design of each button for the best user experience.

2. Enhance communication between the two main user groups, patients and doctors. In our current system, contact between patients and doctors is maintained through email, and doctors can view patients' comments under each exercise. In future iterations, we hope to implement an account system, allowing doctors and patients to communicate directly through the platform, thus increasing the potential for remote medical guidance.

3. Optimize the clinicians' interface regarding academic papers by providing appropriate and relevant tags that link to our Exercise Tags. This will enhance the user experience for clinicians.

4. Implement an improved feedback system. While our current platform already includes a feedback system for patients to leave valuable comments, we aim to develop a feature that allows patients to interact and share their thoughts and experiences regarding specific exercises. This functionality will foster

communication among patients and enable them to learn from each other's experiences.

5. Strengthen the platform's adaptability to different languages and cultures, broadening its reach and ensuring that it caters to a diverse user base.

6. Introduce gamification elements, such as points, badges, or progress tracking, to increase user engagement and motivation to complete exercises and adhere to treatment plans. This would help maintain patient loyalty to our platform and potentially enhance the effectiveness of doctors' treatment plans and supervision.

7. As our platform is both professional and academic, it is essential to develop a comprehensive onboarding process, including tutorial videos or guided walkthroughs, to help new users quickly understand the platform's features and navigate the system with ease.

8. Continuously optimize and expand the database of exercises and treatment options, ensuring that the platform remains current with the latest research and best practices in the field of rehabilitation.

Bibliography

[1] Barbara Farbey. Exploring requirements: Quality before design. *European Journal of Information Systems*, 1(2):145–147, 1991.

[2] Martin Maguire. Methods to support human-centred design. *International journal of human-computer studies*, 55(4):587–634, 2001.

[3] Gunner Haag, Rebekah Kuhlman, and Devante Mante. About face 3: the essentials of interaction design. 2007.

[4] William Lidwell, Kritina Holden, and Jill Butler. *Universal principles of design, revised and updated: 125 ways to enhance usability, influence perception, increase appeal, make better design decisions, and teach through design*. Rockport Pub, 2010.

[5] Jakob Nielsen. *Usability engineering*. Morgan Kaufmann, 1994.

[6] Don Norman. *The design of everyday things: Revised and expanded edition*. Basic books, 2013.

[7] Noam Tractinsky, Avivit Cokhavi, Moti Kirschenbaum, and Tal Sharfi. Evaluating the consistency of immediate aesthetic perceptions of web pages. *International journal of human-computer studies*, 64(11):1071–1083, 2006.

[8] Brett S Gardner. Responsive web design: Enriching the user experience. *Sigma Journal: Inside the Digital Ecosystem*, 11(1):13–19, 2011.

[9] Richard Rutter, Patrick H Lauke, Cynthia Waddell, Jim Thatcher, Shawn Lawton Henry, Bruce Lawson, Andrew Kirkpatrick, Christian Heilmann,

Michael R Burks, Bob Regan, et al. *Web accessibility: Web standards and regulatory compliance*. Apress, 2007.

[10] David Garlan and Mary Shaw. An introduction to software architecture. In *Advances in software engineering and knowledge engineering*, pages 1–39. World Scientific, 1993.

[11] Roy Thomas Fielding. *Architectural styles and the design of network-based software architectures*. University of California, Irvine, 2000.

[12] Google apis, 2023. Accessed: 2023-04-14.

[13] Ucl api, 2023. Accessed: 2023-04-26.

[14] Mysql, 2023. Accessed: 2023-04-01.

fancyvrb listings

Appendix A

User Manual

A.1 Introduction

Welcome to the ”UCL Exercise App” Rehabilitation Exercise Platform User Manual! Our platform is designed to assist both clinicians and patients in managing rehabilitation exercises and tracking progress throughout treatment. This user manual provides a comprehensive guide on how to navigate and utilize the platform's features.

A.2 Getting Started

A.2.1 Role Selection

To access our platform, you will need to select a role: Clinician or Patient.

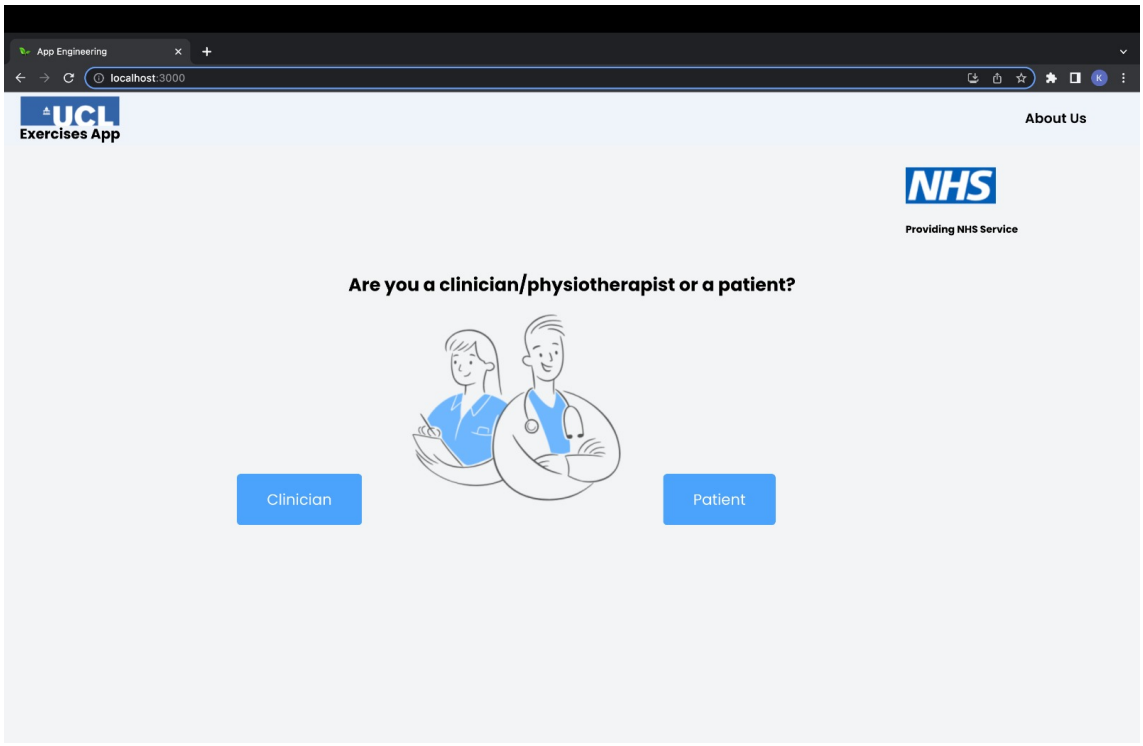

Figure A.1: Role Selection

A.2.2 Logging In

For clinicians, you can create an account and log in using your Google or UCL account. For patients, there is no need to associate any external accounts with our platform. Once you have an account, you can log in using your chosen method. After logging in, you will be directed to the appropriate interface depending on your user type (clinician or patient).

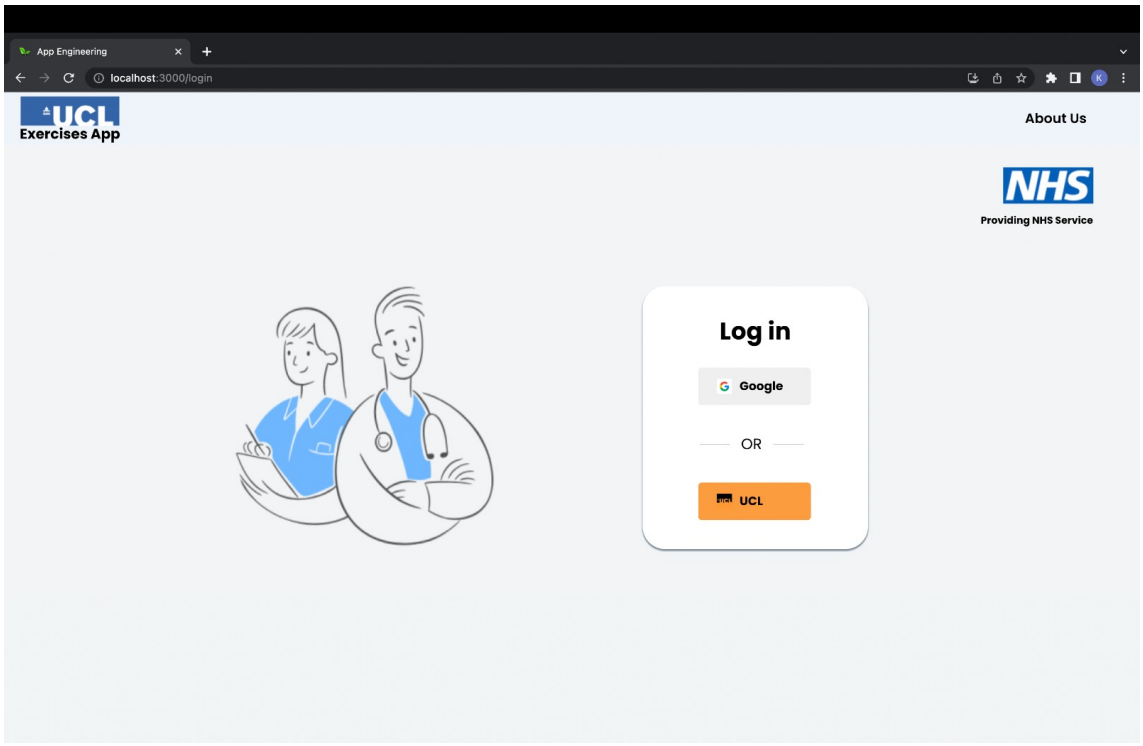

Figure A.2: Log in

A.3 Patient Interface

A.3.1 Main - Patient

The main patient interface allows you to view, search, and filter any Exercise and Template. You can also click on an item to review the details of each Exercise. Moreover, you can see all the Templates created by attentive Clinicians in the past.

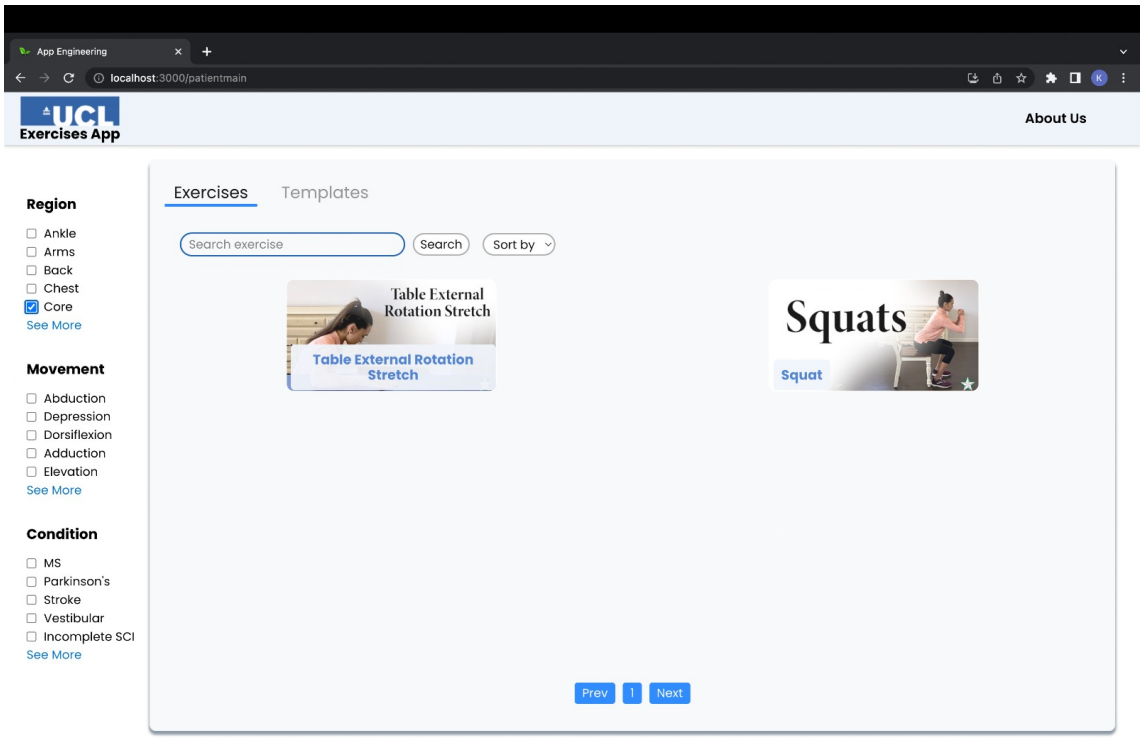

Figure A.3: Patient Main Filter

A.3.2 Exercise Library

Browse and search for specific exercises in the Exercise Library. Each exercise includes a title, description, video demonstration, and instructions on how to perform it correctly.

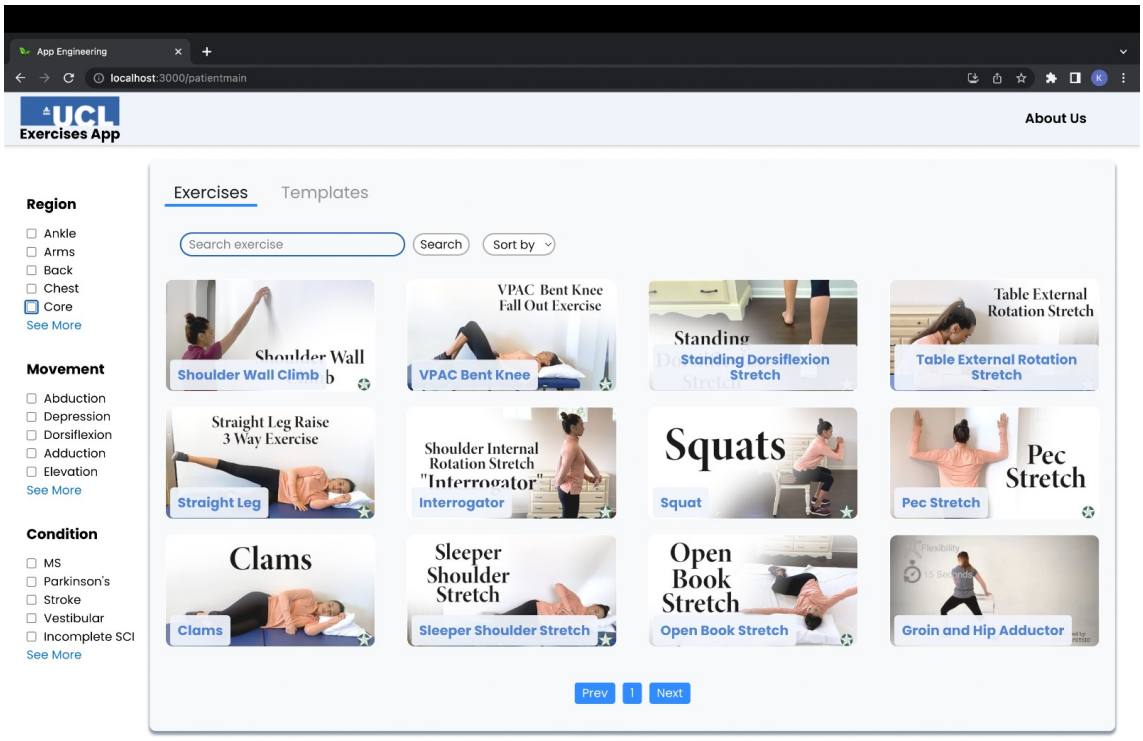

Figure A.4: Patient Main

A.3.3 Template Library

Browse and search for specific templates in the Template Library. Each template consists of a collection of exercises, and you can click on any exercise directly for more information.

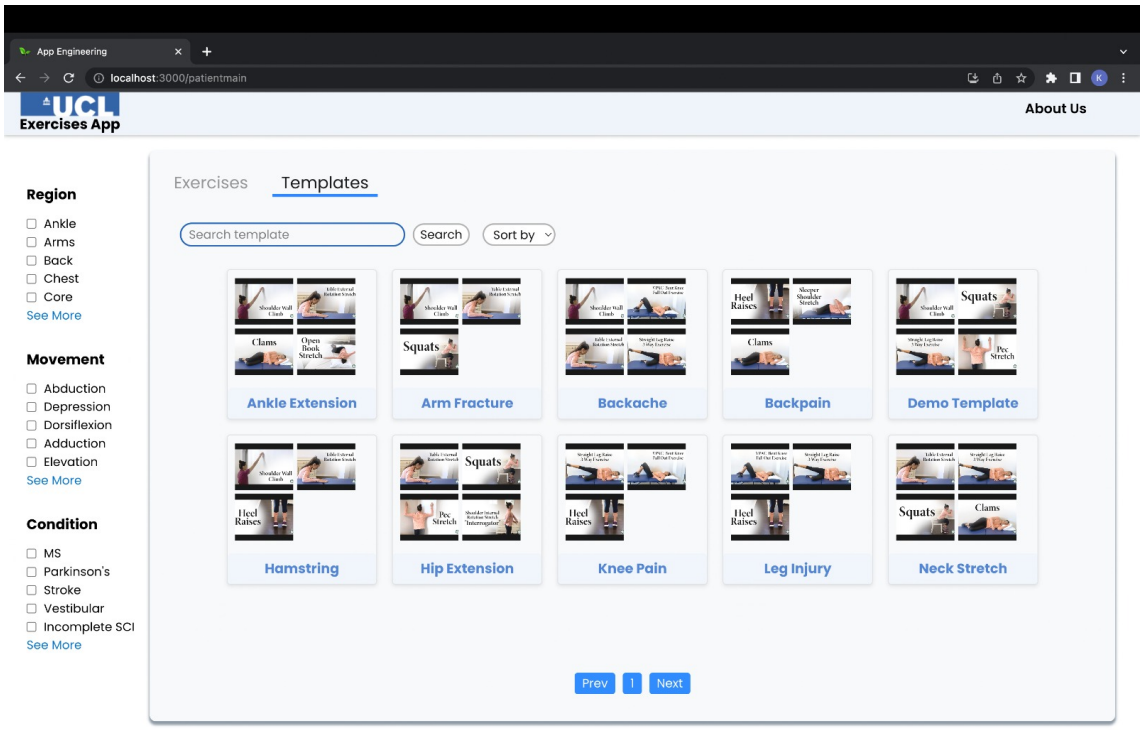

Figure A.5: Patient Template

A.3.4 Exercise

The Exercise content interface allows you to follow a video guide for rehabilitation movements. This feature helps you visualize the correct execution of rehabilitation exercises and stay motivated throughout the rehabilitation process. You will also be able to see the title, description, tags, and rating of a specific Exercise.

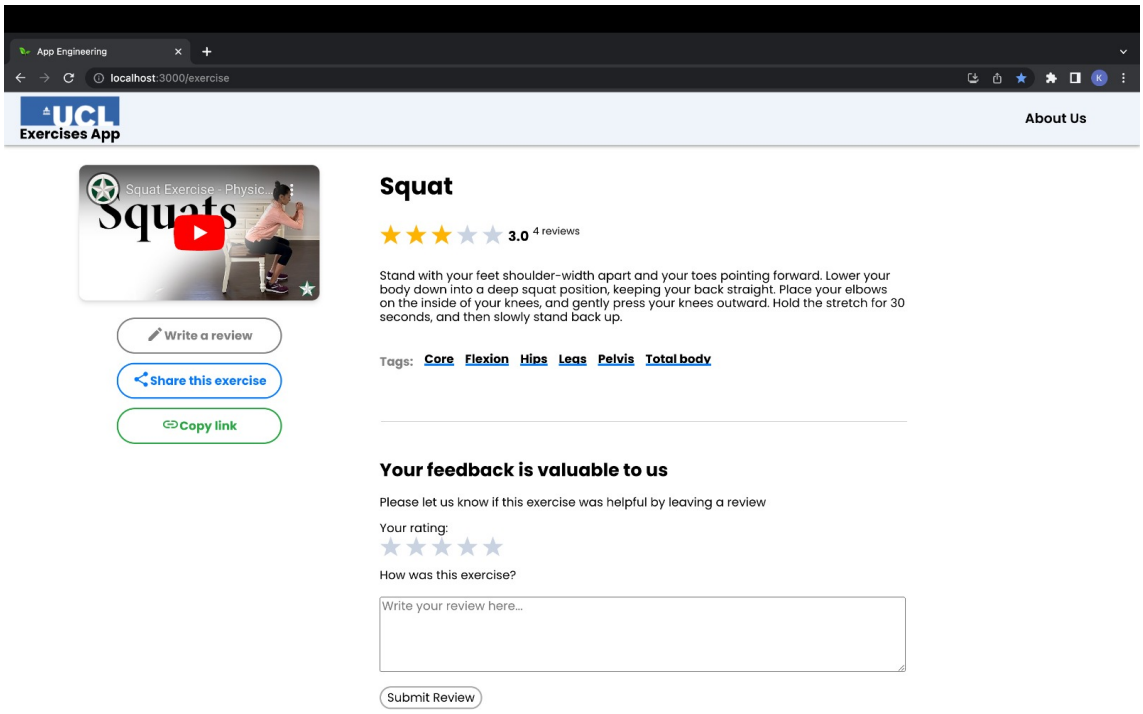

Figure A.6: Exercise Detail

A.3.5 Rating

In each particular Exercise, we have a clear rating system that shows the current distribution of reviews. This allows both our patients and clinician users to get a rough idea of the existing Exercise reviews.

A.3.6 Comment

Patients can leave valuable comments, which can be used by every patient in need of exercise rehabilitation. These comments allow users to share their experiences and provide additional insights into the effectiveness of a specific exercise. The Clinician can also follow up with comments after each of the patient's comments.

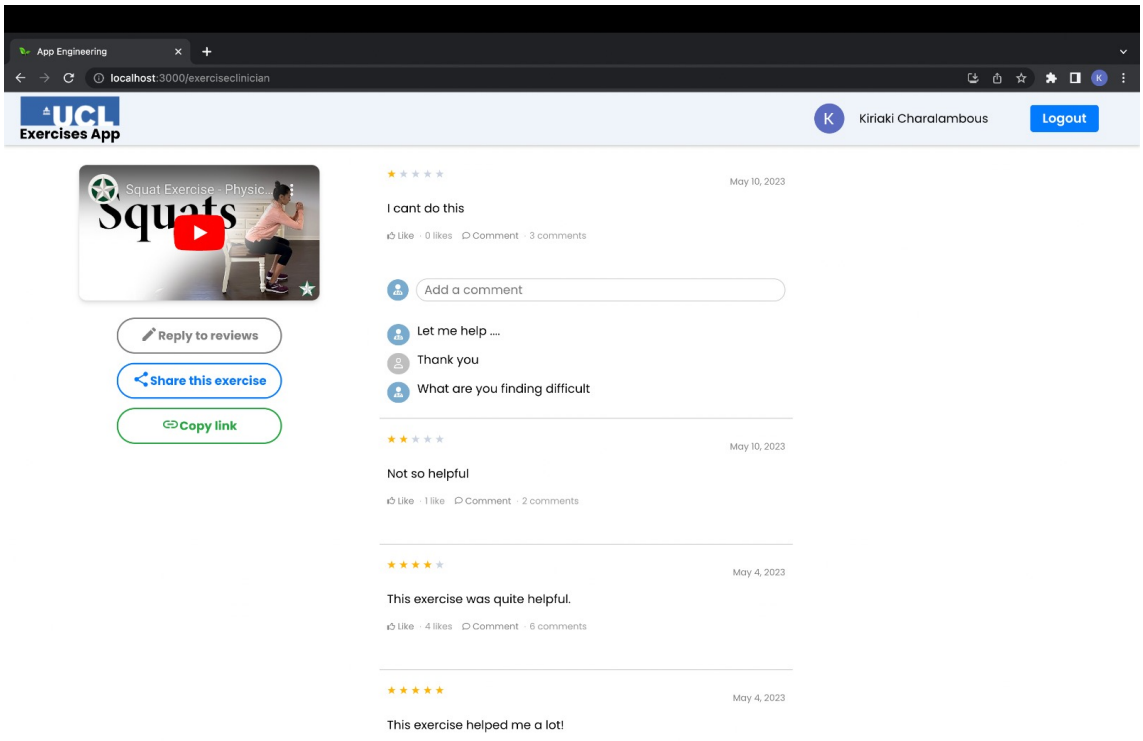

Figure A.7: Comment

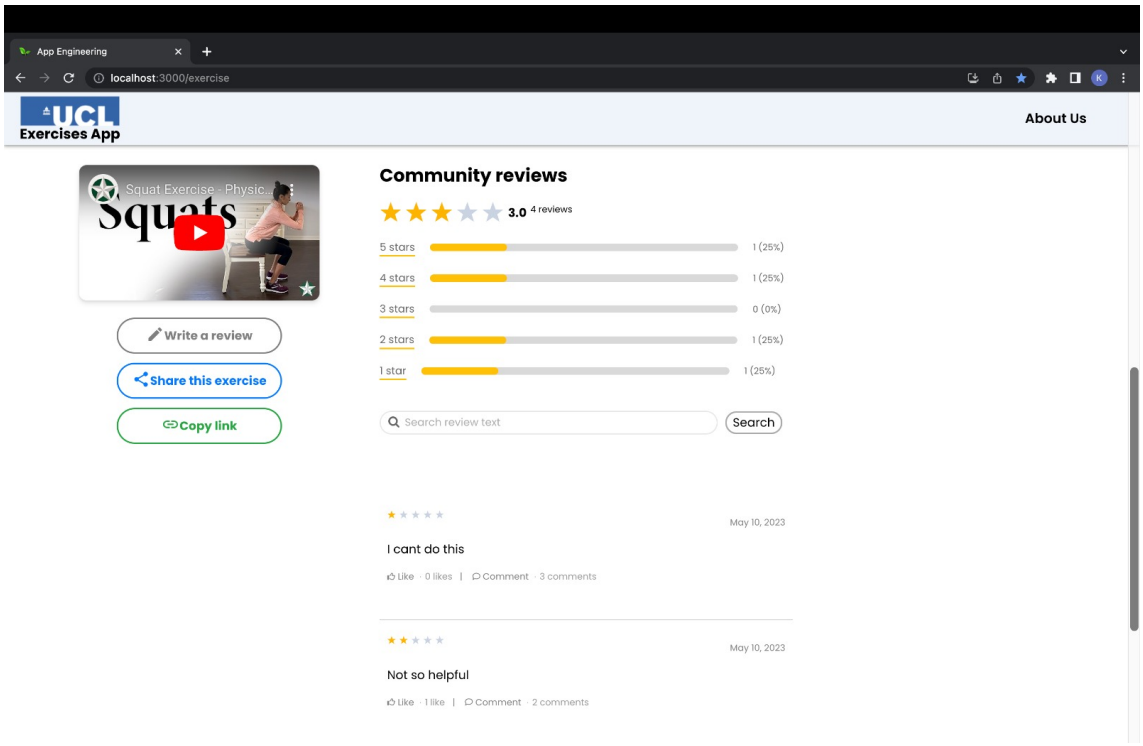

Figure A.8: Comment List

A.4 Clinician Interface

This user manual provides a detailed guide on how to use the MainPage application. The MainPage is designed to manage exercises and templates, allowing users to interact with the application to search, filter, sort, and send emails.

A.4.1 Main Page Interface

The Main Page interface consists of the following components:

1. Exercise and template tabs: Users can switch between the exercise and template tabs to view and manage exercises or templates.

2. Search bar: Users can search for exercises or templates by entering a search term.

3. Sort options: Users can sort exercises or templates by different criteria, such as alphabetical order or creation date.

4. Pagination controls: Users can navigate between pages of exercises or templates using the pagination controls at the bottom of the page.

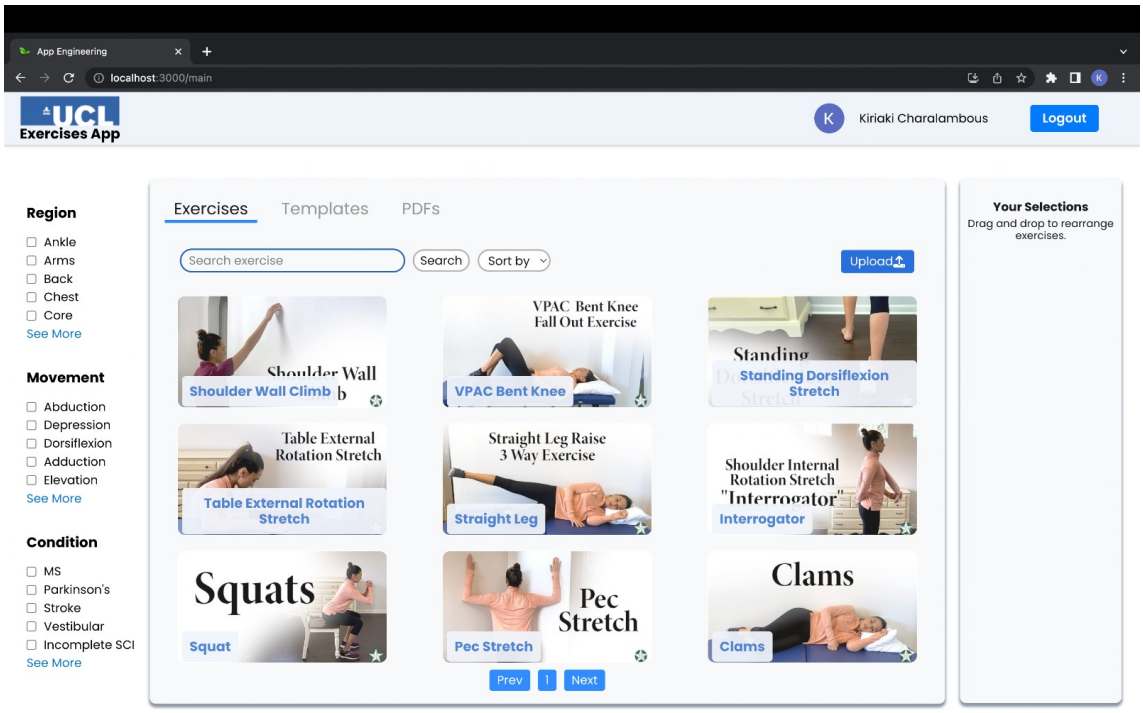

Figure A.9: Main Clinician

A.4.2 Using the Main Page Application

A.4.2.1 Viewing Exercises and Templates

To view exercises or templates, follow these steps:

1. Click on the ”Exercises” or ”Templates” tab to switch between the exercise and template views.

2. Use the pagination controls at the bottom of the page to navigate between pages of exercises or templates.

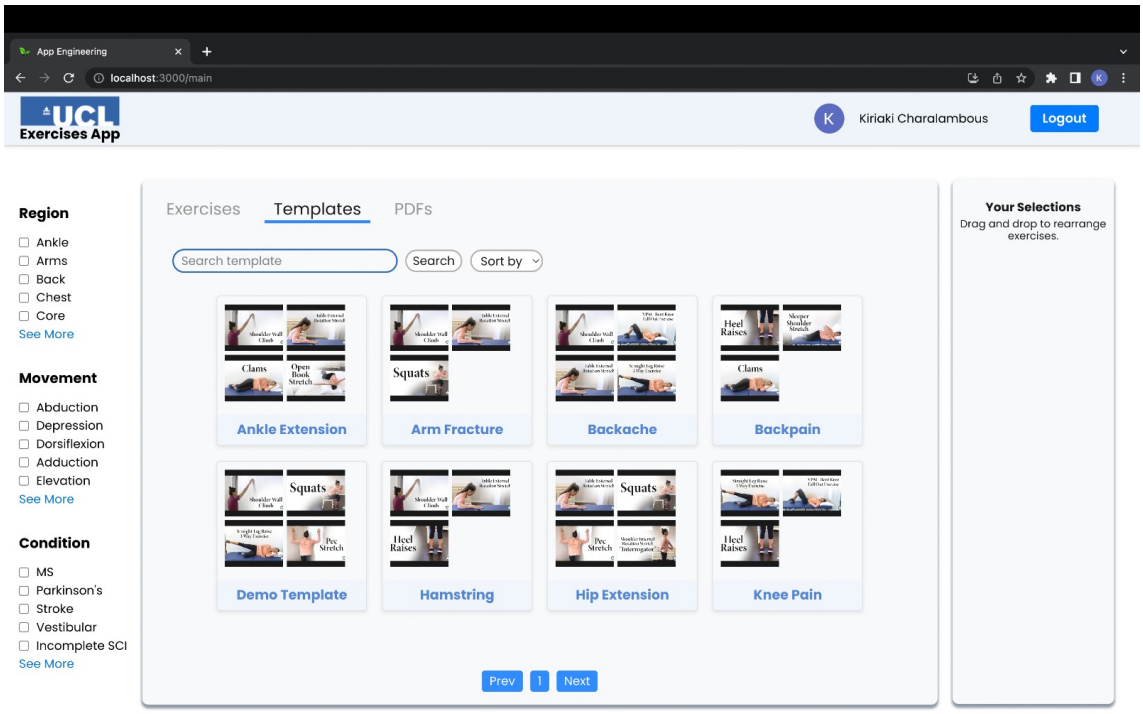

Figure A.10: Main(Template) Clinician

A.4.2.2 Searching for Exercises and Templates

To search for exercises or templates, follow these steps:

1. Click on the ”Exercises” or ”Templates” tab to switch to the desired view.

2. Enter a search term in the search bar. The list of exercises or templates will be filtered based on the entered search term.

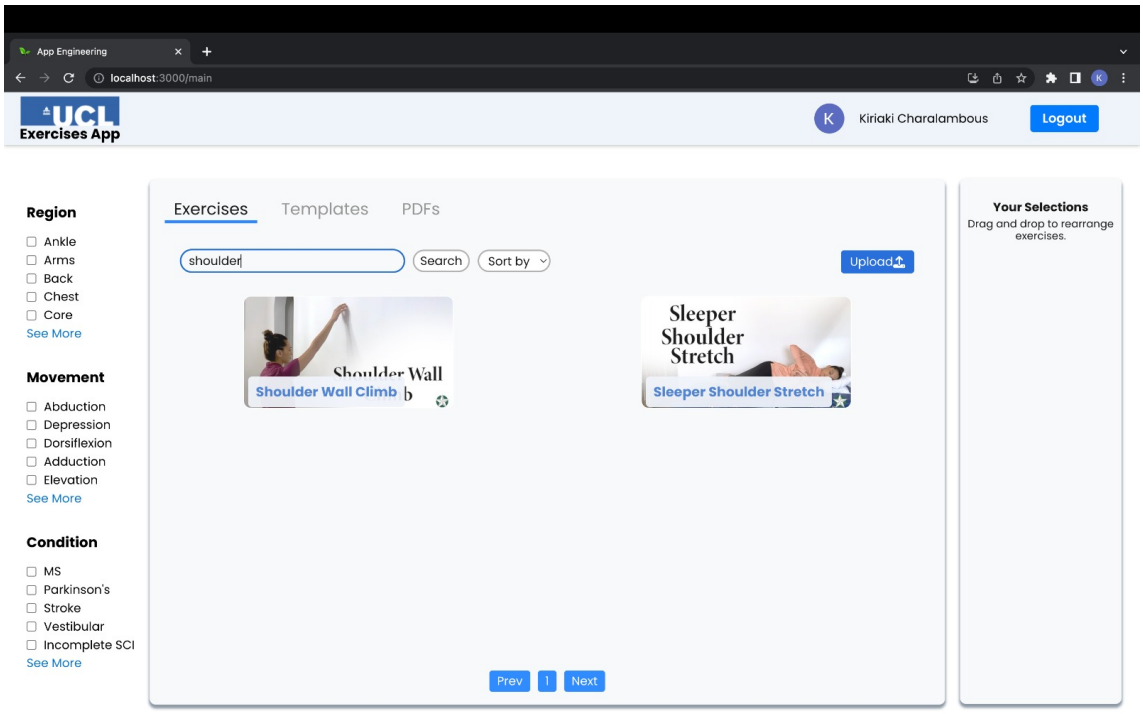

Figure A.11: Main(Template) Clinician

A.4.2.3 Sorting Exercises and Templates

To sort exercises or templates, follow these steps:

1. Click on the ”Exercises” or ”Templates” tab to switch to the desired view.
2. Select a sorting option from the drop-down menu. The list of exercises or templates will be sorted based on the selected option.

A.4.2.4 Using the Timeline

The Timeline feature allows users to manage their exercises more efficiently. Users can select exercises, modify their order within the Timeline using a drag-and-drop functionality, and adjust the repetition count for each exercise.

1. Select exercises from the exercise list to add them to the Timeline.
2. Click and drag exercises within the Timeline to rearrange their order.
3. Adjust the repetition count for each exercise using the input field provided within the exercise component.

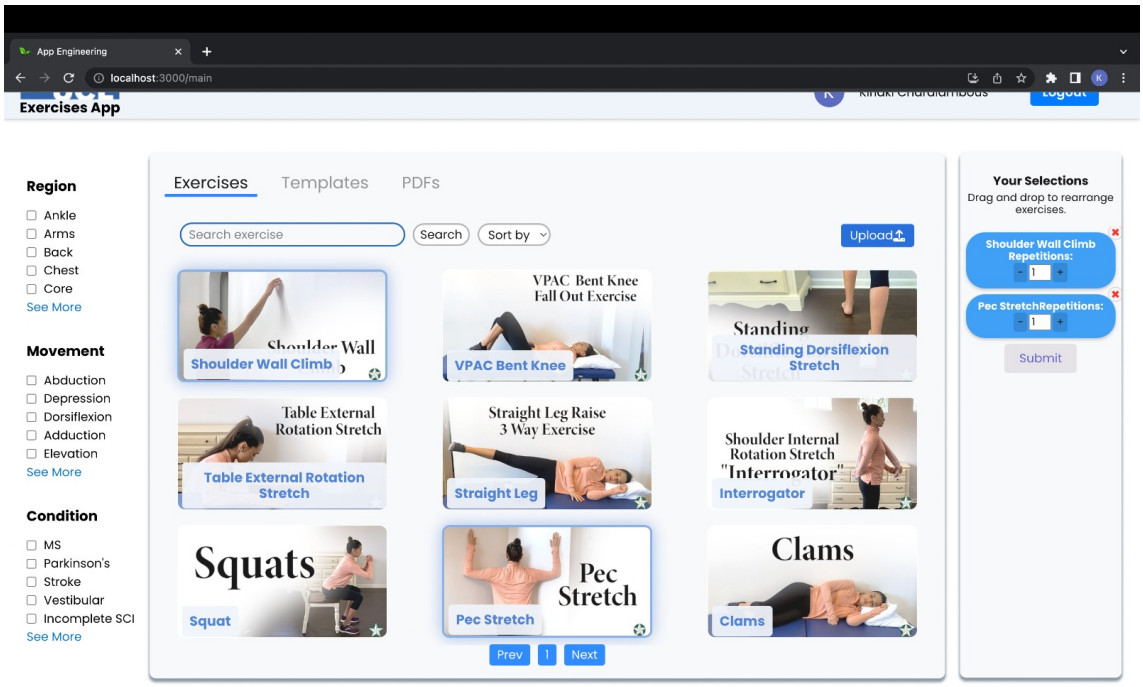

Figure A.12: Timeline

A.4.2.5 Sending Emails

To send an email, follow these steps:

1. Ensure that the recipient's email address is valid. The application uses the `validateEmail` function to verify the email address format.

2. Compose the email content.

3. Click the "Send Email" button. The application will attempt to send the email using the `sendEmail` function. A success or error message will be displayed based on the outcome.

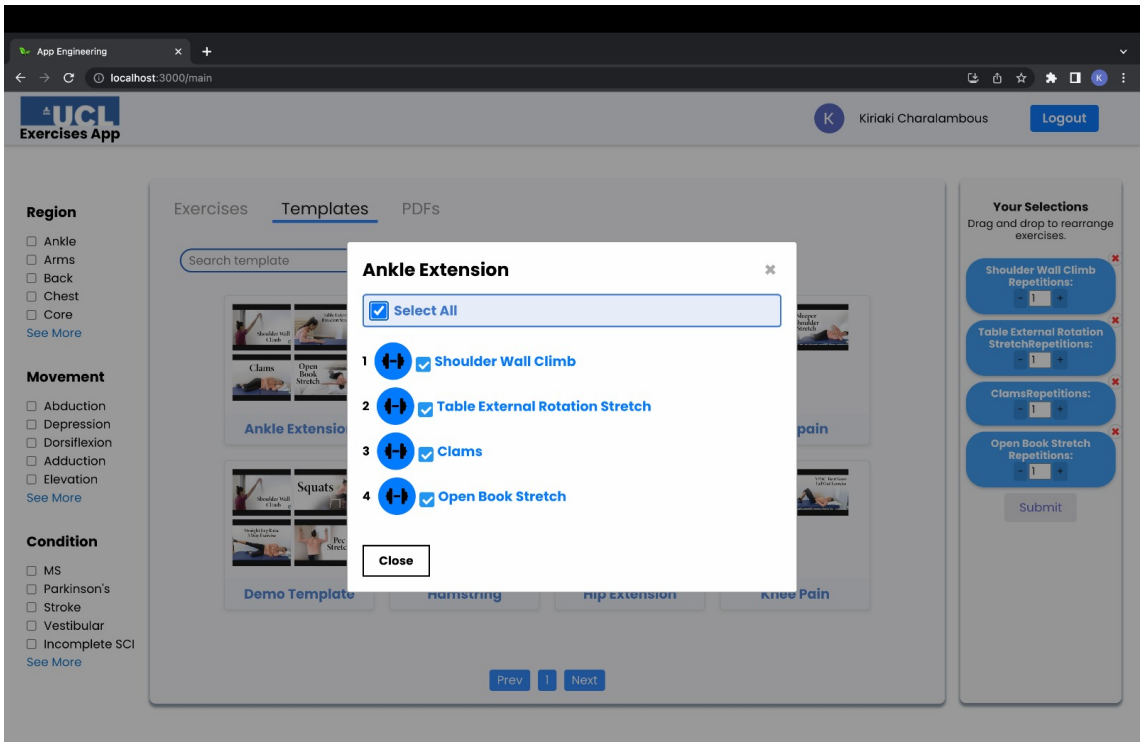

Figure A.13: Template Detail

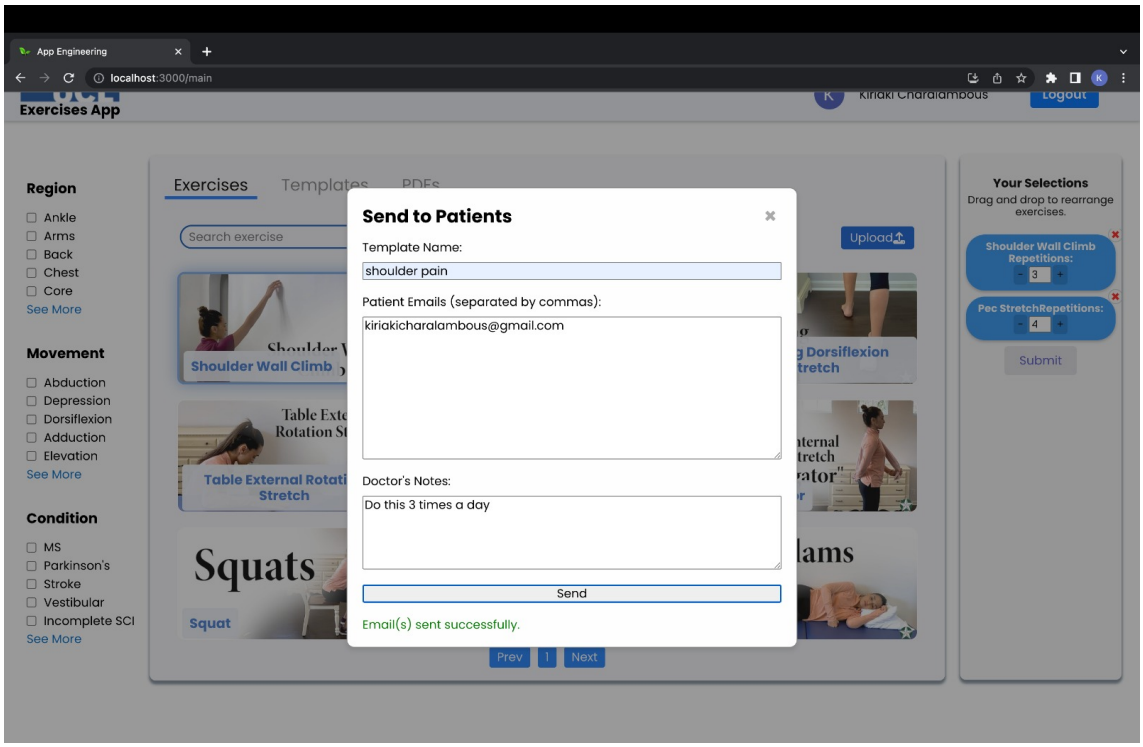

Figure A.14: Template Detail

A.4.2.6 Uploading Exercise Json

In this part, you will upload the related information of Exercise, including the title, description, YouTube link, tags, and JSON file.

1. Title: Enter a unique title. If the title is already taken, the system will prompt you to change the title.

2. Action description: Provide a brief description of the action features.

3. YouTube link: Provide a YouTube video link related to the action.

4. Tags: Select tags related to the action, making it easier for other users to find your content. You can choose based on categories such as joints/regions, movements, and conditions.

5. Upload JSON file: Drag and drop the JSON file related to the action into the designated area or click on the area to select the file.

After completing all required fields, click the ”Submit” button. If successfully submitted, the system will prompt ”Data submitted successfully”.

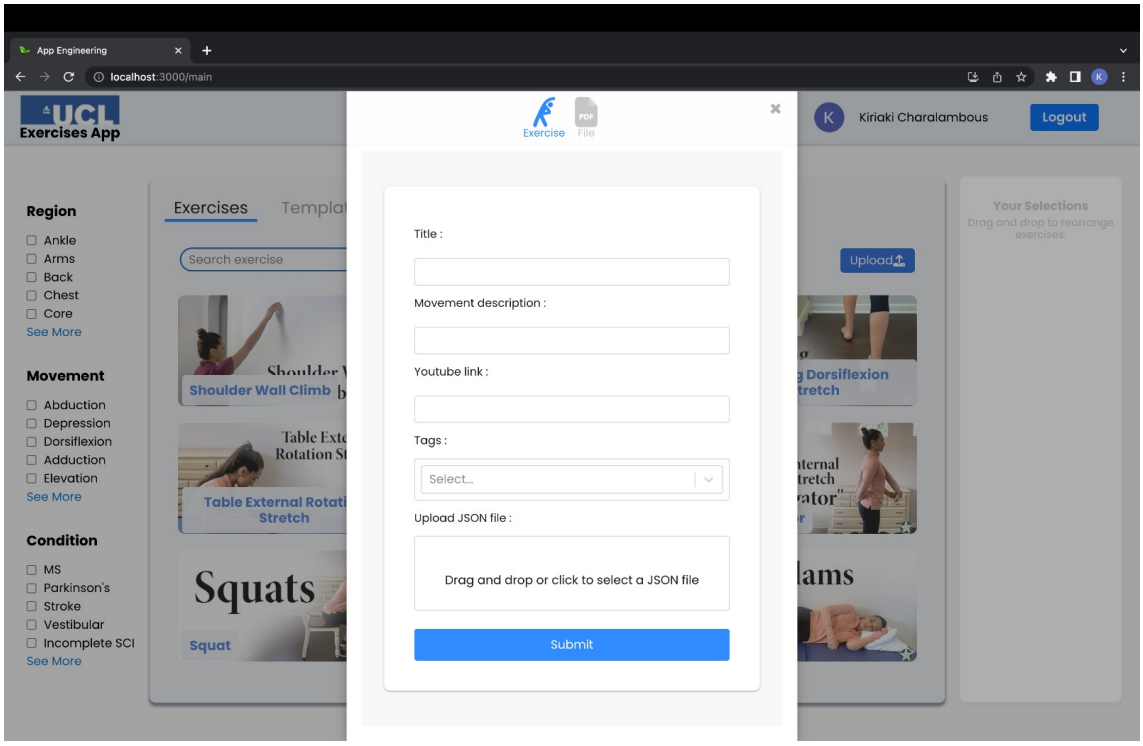

Figure A.15: Upload Exercise

A.4.2.7 Uploading PDF Files

In this part, you will upload PDF files and their related information, including the title, author, institution, location, and description.

1. Title: Enter a unique title. If the title is already taken, the system will prompt you to change the title.
2. Author: Enter the author of the PDF file.
3. Institution: Enter the institution or team related to the PDF file.
4. Location: Enter the publication location of the file.
5. Description: Provide a brief description of the content of the PDF file.
6. Upload PDF file: Drag and drop the PDF file into the designated area or click on the area to select the file.

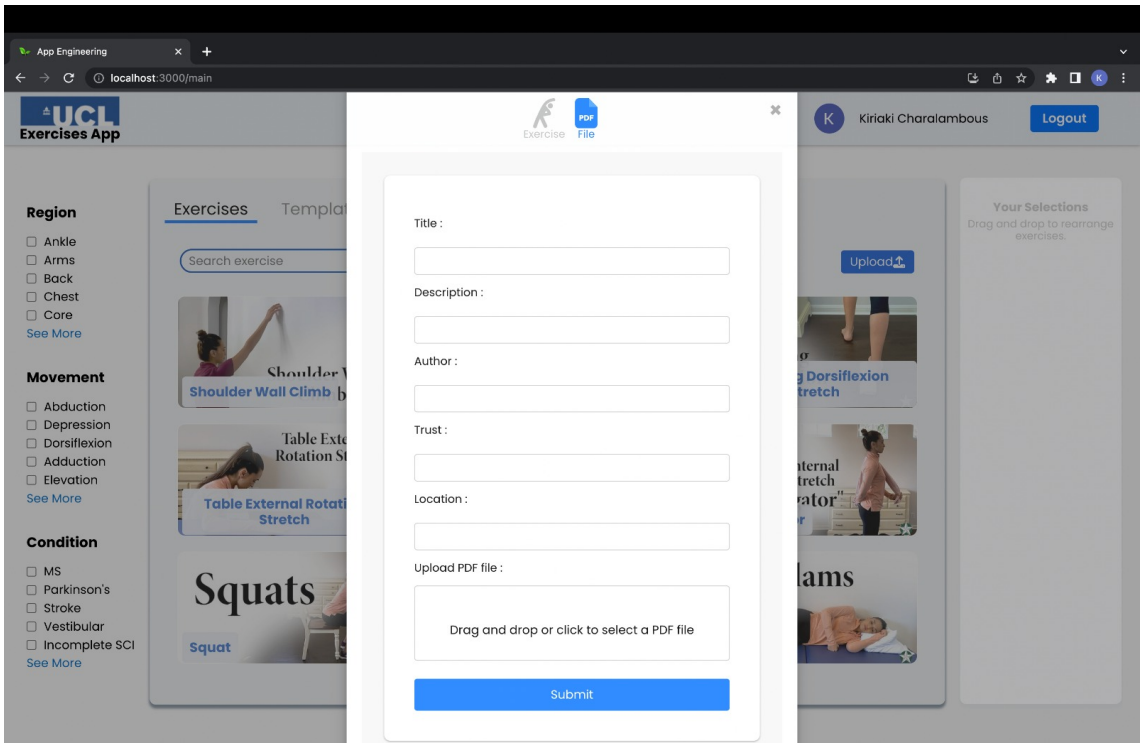

Figure A.16: Upload PDF

After completing all required fields, click the ”Submit” button. If successfully submitted, the system will prompt ”Data submitted successfully”.

Thank you for choosing the UCL Exercise APP Platform. We hope this user manual has been helpful, and we wish you a successful rehabilitation journey!

Appendix B

Deployment Manual

B.1 Current Deployment Information

B.1.1 Estimated System Cost

The deployment of our web application involves various costs, including server hosting, database storage, and data transfer. In this document, we provide an estimated cost breakdown for the deployment of our web application, including the server types and their respective costs for the project and per day.

Service Type	**Cost for project**	**Cost per day**
Storage	$5.53	$0.11
Cloud Database (e.g., MySQL)	$44.28	$0.90
Bandwidth	$0.03	$0.00
Virtual Machines	$27.21	$0.56
Virtual Network	$4.50	$0.09
Total	**$81.56**	**$1.66**

Table B.1: Estimated Cost Breakdown

B.2 Server and Database Connection Information

In Azure create a resource group , we will later add an Azure VM for SQL database and a VM within this.

B.2.1 Database

Platform: Azure DataBase for MySQL server

Configuration: Basic, 1 vCore(s), 5 GB

`Credentials`

Server Name: exercise-app-db.mysql.database.azure.com

Username: ExercisesApp

Password: YunFu420

B.2.2 Virtual Machine

Platform: Azure VM

Configuration: Standard B1ms (1 vcpus, 2 GiB memory)

Storage: 2 GiB (Standard HDD)

`Credentials`

Host Name: uclexercisesapp.uksouth.cloudapp.azure.com

Username: ExercisesApp

Password: YunFu4201999

B.2.3 Deployment Guide for the Web Application

B.2.3.1 Configure the database

1. Create an Azure Virtual Machine for MySQL Database.Remember the credentials in creation(host name, username, and password).

2. Make sure your IP is accepted for connecting to your database.

3. In MySQL workbench connect to the deployed database using your credentials and create the db structure.

4. Use this SQL code to create the DB:

5.

```sql
CREATE DATABASE `exercisesapp` /*!40100 DEFAULT CHARACTER SET latin1 */;
CREATE TABLE `comments` (
  `NewCommentId` int(11) NOT NULL AUTO_INCREMENT,
  `CommentId` int(11) NOT NULL,
  `Text` varchar(7000) NOT NULL,
  `UserType` varchar(200) NOT NULL,
  PRIMARY KEY (`NewCommentId`,`CommentId`)
) ENGINE=InnoDB AUTO_INCREMENT=22 DEFAULT CHARSET=latin1;

CREATE TABLE `exercises` (
  `Title` varchar(400) NOT NULL,
  `Description` varchar(10000) NOT NULL,
  `Link` varchar(300) NOT NULL,
  `JSON` varchar(50000) NOT NULL,
  `CreatedAt` datetime NOT NULL DEFAULT CURRENT_TIMESTAMP,
  PRIMARY KEY (`Title`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

CREATE TABLE `likes` (
  `CommentId` int(11) NOT NULL,
  `Likes` int(11) DEFAULT NULL,
  PRIMARY KEY (`CommentId`),
  CONSTRAINT `CommentId` FOREIGN KEY (`CommentId`) REFERENCES `reviews` (`CommentId`) ON DELETE NO ACTION ON UPDATE NO ACTION
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

CREATE TABLE `pdfs` (
  `PDFTitle` varchar(400) NOT NULL,
  `Author` varchar(300) NOT NULL,
  `Trust` varchar(300) NOT NULL,
  `Location` varchar(300) NOT NULL,
  `PDFDescription` varchar(10000) NOT NULL,
  `PDF` longblob NOT NULL,
  `PDFCreatedAt` datetime NOT NULL DEFAULT CURRENT_TIMESTAMP,
  PRIMARY KEY (`PDFTitle`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Figure B.1

```sql
CREATE TABLE `reviews` (
  `CommentId` int(11) NOT NULL AUTO_INCREMENT,
  `ExerciseTitle` varchar(255) NOT NULL,
  `Rating` int(11) NOT NULL,
  `Comment` varchar(7000) NOT NULL,
  `CreatedTime` datetime NOT NULL DEFAULT CURRENT_TIMESTAMP,
  PRIMARY KEY (`CommentId`,`ExerciseTitle`),
  KEY `Title_idx` (`ExerciseTitle`)
) ENGINE=InnoDB AUTO_INCREMENT=11 DEFAULT CHARSET=latin1;

CREATE TABLE `tags` (
  `Title` varchar(400) NOT NULL,
  `Tag` varchar(400) NOT NULL,
  PRIMARY KEY (`Title`,`Tag`),
  CONSTRAINT `Title` FOREIGN KEY (`Title`) REFERENCES `exercises` (`Title`) ON DELETE NO ACTION ON UPDATE NO ACTION
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

CREATE TABLE `timelines` (
  `TimelineTitle` varchar(400) NOT NULL,
  `TimelineJSON` text NOT NULL,
  `Created_At` datetime NOT NULL DEFAULT CURRENT_TIMESTAMP,
  PRIMARY KEY (`TimelineTitle`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Figure B.2: SQL table creation

B.2.3.2 Deploy the web application

1. Deploy an Azure Virtual Machine,and remember the username and password you use. Once it's deployed take a note of the host IP address.

2. In your terminal, use SSH to connect to your VM instance from your local machine <username>@<hostIP>.

3. Once in SSH, install NodeJS,pm2 and npm by running the following commands:

   - cd ∼
   - sudo npm install pm2@latest -g
   - curl -sL https://deb.nodesource.com/setup 8.x -o nodesource setup.sh
   - sudo bash nodesource setup.sh
   - sudo apt install nodejs
   - sudo apt install build-essential

4. For mac, you can use CyberDuck to add files to your VM :
   - In your SSH, create a directory to keep the files e.g. mkdir uclexercises
   - Open CyberDuck and click open connection
   - Select SFTP (SSH File Transfer Protocol)
   - In the server fields write your IP , and then your username and password.
   - Once the connection, is made you will be able to see your uclexercises file.
   - In your editor run the command : npm run build
   - Copy the contents of the build folder into a folder called "views"
   - Now you can drag and drop your Backend folder into the CyberDuck uclexercises

5. Once the file is transferred, install and configure NGINX.

6. Also install SSL certificate

7. Now, navigate to the Backend folder in SSH, and write the command : sudo vim /etc/nginx/sites-available/default
   make sure that this part of the file looks like this:

```
server {

        # SSL configuration
        #
        # listen 443 ssl default_server;
        # listen [::]:443 ssl default_server;
        #
        # Note: You should disable gzip for SSL traffic.
        # See: https://bugs.debian.org/773332
        #
        # Read up on ssl_ciphers to ensure a secure configuration.
        # See: https://bugs.debian.org/765782
        #
        # Self signed certs generated by the ssl-cert package
        # Don't use them in a production server!
        #
        # include snippets/snakeoil.conf;

        root /var/www/html;

        # Add index.php to the list if you are using PHP
        index index.html index.htm index.nginx-debian.html;
    server_name uclexercisesapp.uksouth.cloudapp.azure.com; # managed by Certbot


        location / {
                root /home/ExercisesApp/uclexercises/Backend/views;
                try_files $uri /index.html;
                  }

        location /api {
                proxy_pass http://localhost:3001;
                proxy_http_version 1.1;
                proxy_set_header Upgrade $http_upgrade;
                proxy_set_header Connection 'upgrade';
                proxy_set_header Host $host;
                      }
```

Figure B.3: Configuration file

8. Makesure you are still in the Backend and write these commands now:

   - sudo pm2 start index.js

   sudo systemctl restart nginx

9. Your app should now be visible at your url.

Appendix C

Resources

Table C.1: Learning Resources

Language	Resource
React	https://www.youtube.com/watch?v=4UZrsTqkcW4&t=1s
Node.js	https://www.youtube.com/watch?v=LAUi8pPlcUM&list=PLC3y8-rFHvwh8shCMHFA5kWxD9PaPwxaY
MySQL	https://www.youtube.com/watch?v=5OdVJbNCSso
CSS	https://www.youtube.com/watch?v=OXGznpKZ_sA

Appendix D

Code Citation

Table D.1: Code Citation

No.	Function Name	Code File Name	Source
1	google	Login.js	`https://developers.google.com/identity/sign-in/web/sign-in`
2	ucl	Login.js	`https://uclapi.com/docs/`
3	thumbnail	App.js	`https://cloudinary.com/blog/guest_post/Generate-Simple-Youtube-Thumbnails`