NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

AY 2023/24 SEMESTER 1

SC2006 Software Engineering

Project Title: SportSync

Tutorial Class: SDAA

Group Number: 1

Group Name: Techies

Date of Submission: 12 November 2023

Name	**Matriculation Number**
Asher Lim Guojun	U2220846H
Brandon Jang Jin Tian	U2220936G
Chai Jie En	U2221398J
Cheam Zhong Sheng Andrew	U2221140J
Chew Zhaoding	U2220243J
Dennis Chen Chee Hao	U2220058A

Table of Contents

Revision History

Name	Date	Reason For Changes	Version
Dennis Chen	18/08/23	Starting of SRS Documentation	1.0
Brandon Jang	08/09/23	Lab 1 Submission	1.1
Andrew Cheam	22/09/23	Lab 2 Submission	2.0
Asher Lim	27/10/23	Lab 3 Submission	3.0
Chia Jia En	08/11/23	Full Report Completion	4.0
Chew Zhaoding	10/11/23	Feedback after presentation	4.1
Dennis Chen	12/11/23	Finalised Submission	4.2

1 Requirement

1.1 Introduction

1.1.1 Purpose

The Software Requirements and Specifications (SRS) document serves as a comprehensive and meticulous elucidation of the SportSync website, an all-encompassing web-based platform designed to facilitate the coordination and management of training sessions for athletes and coaches participating in various sports activities across Singapore.

Within the SRS, one can find a comprehensive explanation of the web application's underlying objectives and its myriad of features, aimed at streamlining the process of scheduling and organising training sessions. Furthermore, it delves into the intricacies of the application's interfaces, the intricacies of its implementation, and the myriad of restrictions and prerequisites that must be adhered to during its development.

This document is not only a roadmap but also a blueprint, encapsulating the essential requirements, analytical insights, and design specifications necessary for the successful execution of the SportSync website project.

1.1.2 Document Conventions

H1 - font: Arial, size: 20, style: bold
H2 - font: Arial, size: 16, style: normal
H3 - font: Arial, size 14, style: normal
H4 - font: Arial, size 13, style: normal
H5 - font: Arial, size 12, style: normal
Text - font: Arial, size: 11, style: normal
Line Height - 1.15

1.1.3 Intended Audience and Reading Suggestions

The document is intended for developers, project managers, investors, testers, and general users. It is advisable for developers and project managers to go through the entire document, while investors, testers, and general users are encouraged to focus on the overall description and system features related to the SportSync application to grasp its functionality. For an optimal comprehension, it is recommended to follow the content page sequence when reviewing this SRS document.

1.1.4 Product Scope

SportSync is a web-based platform that empowers users to live a healthy lifestyle through exercise and sports activities. The application allows athletes to book listed activities while coaches create activities to list. Furthermore, SportSync allows for a post-activity review of the coach, live chat functionality to clarify details of the activity between athletes and the coach, and information on nearby taxis should a user require taxi booking services to easily commute to and from the activities.

Our Core Objectives:

1. **Simplify Training Session Booking:** We aspire to become the go-to resource for sporty individuals in Singapore. Users can effortlessly browse and book training sessions, making it convenient to pursue their fitness goals and interests. Our platform will host a wide range of training options to cater to diverse preferences and skill levels.

2. **Community Building:** Beyond just facilitating green space discovery and training session bookings, we aim to foster a sense of community among users. Our platform will allow individuals to connect with like-minded individuals, arrange joint training sessions, and share their fitness journeys, further encouraging active and social lifestyles.

3. **Seamless Transportation Integration:** To enhance the overall user experience, we are committed to integrating with the Taxi API. This integration will allow users to locate alternative transportation options quickly, making it easier to reach their chosen training venues.

1.2 Overall Description

1.2.1 Product Perspective

With the SportSync web application, sports enthusiasts can now access and book sports coaching services. Our platform is dedicated exclusively to sports coaching services, making it a one-stop destination for athletes and aspiring players to find experienced coaches in various sports disciplines. After athletes book the coaching services, they are able to communicate with the coaches with our in-house chat to ensure clarity of the services they are providing.

What sets our application apart is the seamless integration of a taxi locator and taxi booking API, which enables users not only to discover and book coaching sessions but also to easily plan their transportation to the coaching location. This comprehensive approach to sports coaching and travel ensures a convenient and hassle-free experience for our users, combining two essential aspects of their sporting journey into one cohesive platform.

1.2.2 Product Functions

Major Functions:

- User Registration
- User Login
- User Create Listing
- Marketplace
    - Search Listing
    - User Book Listing
    - User Save Listing
- Taxi Availability
- Athlete - Coach Group Chat

Minor Functions:

- User Edit Profile
- User Edit Listing
- User Change Password

1.2.3 User Classes and Characteristics

Registered users - Users of SportSync are predominantly young and middle-aged adults ranging from 18 to 45 years old, with a balanced gender distribution and diverse income background.

In general, the demographic for our application should be users who reside in urbanised Singapore, who are minimally tech-savvy and familiar with using the internet.

1.2.4 Operating Environment

SportSync can be operated on any modern web browser including desktop browsers and mobile browsers.

For development purposes, the application is developed on:

- React 18.2.0
- Vite 4.4.11
- Flask 2.3.3

Application is tested on:

- Insomnia
- Localhost

React:
React is a popular JavaScript library for building user interfaces. It was developed by Facebook and is commonly used for creating interactive and dynamic web applications. React allows developers to build reusable UI components and efficiently update the user interface when data changes. It's known for its virtual DOM (Document Object Model) which optimises the rendering of components, making web applications fast and responsive.

Vite:
Vite is a build tool and development server designed for modern web development. It was created by Evan You, the same person who developed Vue.js. Vite focuses on providing a fast and efficient development environment for JavaScript frameworks like React, Vue, and others. It leverages ES modules and on-demand compilation to achieve rapid build and reload times during development. Vite's quick development server and optimised build process make it a popular choice for web application development.

Flask:
Flask is a lightweight and micro web framework for Python. It is designed to make web development in Python simple and flexible. Flask provides the essential tools for building web applications, allowing developers to add components and extensions as needed. It's often used to create RESTful APIs, web services, and small to medium-sized web applications. Flask is known for its simplicity and ease of use, making it a great choice for developers who want to get started quickly with web development in Python.

For testing the application locally on localhost, one have to ensure that they have the relevant flask modules installed by running the pip install command on the requirements.txt file and then “flask run”
Also, one needs to make sure that they have the NPM & Node.JS installed on the computer and run the command “npm install” on the relevant directory to automatically install all the packages required. “npm run dev” will run a server on localhost, which you would be able to access on your browser.

Insomnia:
Insomnia is an open-source cross-platform REST API client that simplifies the process of testing and debugging APIs. It provides a user-friendly interface for making HTTP requests, inspecting responses, and organising API endpoints. Insomnia is particularly valuable during the development and testing phases of web applications. It allows developers to interact with APIs, review responses, and ensure that the integration of the front-end (built with React) and the back-end (implemented with Flask) in the SportSync application is smooth and error-free. This tool helps in verifying that data is correctly exchanged between the client and the server components of the application, making the development process more efficient and reliable.

To use insomnia, download the application through a browser of your choice. Import the insomnia file under the directory

- backend
    - Insomnia_2023-10-10

1.2.5 Design and Implementation Constraints

Capacity of the user is capped at 10 for Chat Engine API and users are unable to send, receive messages thereafter. As pick-up locations are retrieved using real-time government taxi availability, it requires a relatively good internet connection to be able to see the available taxis load.

1.2.6 User Documentation

A README file, as well as the code base, can be found on the official GitHub directory of the application: https://github.com/DennisDCCH/Techies. For any other queries, the team will provide the client with further help in terms of troubleshooting future errors.

1.2.7 Assumptions and Dependencies

Assumptions:
SportSync requires a good, stable internet connection and an internet browser to work. Hence, the system assumes that all the requirements mentioned are met. If the user is not connected to the internet or does not have an internet browser, the web application would not work.

Dependencies:
SportSync relies on the Google Maps API to embed a map that accurately reflects the user's current location. In the absence of this API, the web application will not be able to display this information on the map, preventing users from visualising their current location.

Additionally, SportSync's Taxi Availability feature depends on the API provided by data.gov.sg.

If this API does not provide the latest information on taxis, we will not be able to retrieve it.
Our web application may face deployment and programming errors due to framework and library updates. To comply with the industry standards, we chose React as our primary framework, and JavaScript as our primary frontend language. For backend development, we chose Flask with Python as the primary language.

1.3 External User Interface Requirements

1.3.1 User Interfaces

Upon signing in, the web application must display the navigation bar that contains the features of ‘Profile’, ‘Activities’, ‘Map’, ‘Chat’.

1.3.2 Hardware Interfaces

The application is expected to run on any modern hardware devices such as smartphones, laptops and desktops that support a web browser with an internet connection.

1.3.3 Software Interfaces

- ReactJS v18.2.0 Javascript Framework is used to handle the overall navigation, responsiveness and reusable UI components.
- The system must have web browsers installed to support the application.
- Users must have NPM 9.8.1 installed to manage dependencies for Vite packages.
- data.gov.sg taxis dataset - To retrieve Taxis Dataset.
- Software requires React Google Maps API to be able to depict the user’s current location and nearby available taxis.

1.3.4 Communication Interfaces

In order to interface and load the website, SportSync relies on web protocols like HTTP that are sent from the client/users. Additionally, to display the map and inputted locations, the web application requires an internet connection to load React Google Maps API. Without an internet connection or access to this API, SportSync will not be able to display the map.

1.4 Functional Requirements

1.4.1 User Profile and Authentication

1.4.1.1 User Registration

Description and Priority:
○ The web application must allow the user to register for an account if they do not have an account yet (i.e no account linked to username)
○ Priority: High
● Stimulus/Response Sequences:
    ○ SEQ-1: User clicks on the ‘Register’ navigation button and enters valid input.
    ○ SEQ-2: User clicks on the ‘Register’ to register for an account.
● Functional Requirements:
    ○ REQ-1: Users must be able to enter information in the following fields:
        - First Name
        - Last Name
        - Email
        - Username
        - Password
        - Re-enter Password
        - Gender
        - Age
    ○ REQ-2.1 The system must check if the username is valid. A valid username is one that is not already registered in the system.
    ○ REQ-2.2 The system must check if the password is valid. A valid password is at least 8 characters long and a combination of uppercase letters, lowercase letters, numbers, and symbols.
    ○ REQ-3.1: If valid information is keyed in, the system will save the details in the database and redirect the user to the Login Page.
    ○ REQ-3.2: If invalid information is keyed in, the system will notify the user of the invalid details that have been entered.

1.4.1.2 User Login

Description and Priority:

○ The web application must allow the user to login to their account using a registered username and associated password

○ Priority: High

● Stimulus/Response Sequences:

  ○ SEQ-1: User clicks on ‘Login’ navigation button and enters valid input.

  ○ SEQ-2: User clicks on the Login button to log into the account.

● Functional Requirements:

  ○ REQ-1: Users must be able to enter information in the following fields:

    - Username
    - Password

  ○ REQ-2.1: If valid information is keyed in, the system will log the user in and redirect the user to the Main Menu.

  ○ REQ-2.2: If invalid information is keyed in, the system will notify the user of the invalid details that have been entered and prompt the user to try again.

1.4.1.3 User Profile

Description and Priority:

○ The web application must allow the user to view their profile when they are logged into their account. Users can edit their profile or log out of their account.

○ Priority: High

● Stimulus/Response Sequences:

  ○ SEQ-1: User clicks on the profile navigation icon to view their own user profile.

  ○ SEQ-2.1: If a user chooses to edit their profile information, user clicks on the ‘Edit Profile’ button and enter valid input.

  ○ SEQ-2.2: If the user chooses to log out, user clicks on the ‘Logout’ button.

● Functional Requirements:

  ○ REQ-1: System must display profile navigation icon in the home page which navigates users to their profile page.

  ○ REQ-2: System must display all user profile information in profile page.

  ○ REQ-3: User must be able to edit the information in the following fields:

    - Date of Birth
    - Email
    - Username
    - Profile picture
    - Bio

  ○ REQ-4: System must be able to log the user out of their account and redirect user to the Main Menu.

1.4.2 Listing on marketplace

1.4.2.1 Listing of coaching services

Description and Priority:

○ The web application must allow the users to list their coaching services. The listings must be able to be edited after being put onto the marketplace.

○ Priority: High

● Stimulus/Response Sequences:

  ○ SEQ-1: User clicks on the ‘List coaching service’ button and enters valid information.

  ○ SEQ-1.1: System displays listed coaching services and information on the marketplace

  ○ SEQ-2.1: If a user chooses to edit their listing information, user clicks on the ‘Edit Listing’ button and enters valid input.

  ○ SEQ-2.2: If a user chooses to delete their listing, the user clicks on the ‘Delete Listing’ button.

● Functional Requirements:

  ○ REQ-1: System must be able to navigate users to the page to list their coaching services.

  ○ REQ-2: User must be able to enter the information in the following fields regarding the coaching session:

    - Price
    - Location
    - Time / Date
    - Sport
    - Proficiency Level
    - Description

  ○ REQ-3: User must be able to edit the information in the following fields regarding the coaching session:

    - Location
    - Time
    - Description

1.4.2.2 Display of services on marketplace

Description and Priority:

○ The web application must display all coaching services that are listed onto the marketplace.

○ Priority: High

● Stimulus/Response Sequences:

○ SEQ-1: User clicks on marketplace to view all available coaching sessions.

○ SEQ-1.1: System displays listed service and information on the marketplace.

○ SEQ-2.1: User clicks on session to view more information about the listing.

○ SEQ-2.2: If a user chooses to save the listing, it is saved into their personal list.

○ SEQ-2.3: If user chooses to chat with the coach, a chat page with the coach will be opened up.

● Functional Requirements:

○ REQ-1: System must display the following information regarding the coaching session on the marketplace:

- Price
- Sport
- Listing's cover image

○ REQ-2.1: System must allow users to click the listing to view the following details of the coaching session:

- Price
- Location
- Time
- Sport
- Proficiency Level
- Profile of coach
- Description

○ REQ-2.1: User must be able to save the listing to view at a later time.

○ REQ-2.2: User must be able to commence chatting with the person who listed the coaching session.

1.4.3 Reviews

Description and Priority:

○ Allow users to rate the services of the coaches

○ Priority: Medium

● Stimulus/Response Sequences:

○ SEQ-1: User clicks on the ‘Review’ button and give their desired stars out of five.

○ SEQ-2: User clicks on the ‘Comment’ button to write their desired message about the coach.

○ SEQ-3: User clicks on the ‘Submit’ button to upload their review about the coach and clicks on the ‘Edit’ icon button.

○ SEQ-4: User clicks on the ‘Stars’ icon to view their review and view review histories about the coach.

● Functional Requirements:

○ REQ-1: System must be able to enable users to write and submit reviews for products, services, or other relevant components of the application.

○ REQ-2: System must be able to implement a rating system (e.g., star ratings) to allow users to rate items being reviewed.

○ REQ-3: System must be able to allow users to provide written feedback along with their ratings, giving them the opportunity to explain their opinions.

○ REQ-4: System must be able to support basic text formatting (e.g., bold, italics) to enhance the expressiveness of messages.

○ REQ-5: System must be able to allow users to edit or delete their own reviews, providing them with control over their feedback.

1.4.4 Search based on sports & coach profile

Description and Priority:

○ Allow users to search for their desired sports and coach

○ Priority: High

● Stimulus/Response Sequences:

○ SEQ-1: User clicks on the ‘Search’ button and enters a valid input.

○ SEQ-1.1: If a user clicks on the ‘Filter’ button, multiple filters will be reflected.

○ SEQ-1.1.1: User clicks on the ‘Apply’ icon button to search for their desired input.

● Functional Requirements:

○ REQ-1: System must be able to navigate the user to the Filter page with no hiccups or friction.

○ REQ-2: System must be able to provide a default set of filters to help users get started with a meaningful set of results.

○ REQ-3: System must be able to filter for multiple requirements regarding:

- Proficiency Level
- Type of Sport
- Age Group

with no hiccups or friction.

○ REQ-4: System must be able to handle cases where users input incorrect or incompatible filter values, providing meaningful error messages.

1.4.5 Chat Option

Description and Priority:

○ Allow users to chat with potential coaches they are interested in.

○ Priority: High

● Stimulus/Response Sequences:

  ○ SEQ-1: User clicks on the ‘Chat’ button and enters a valid message.

  ○ SEQ-2: User clicks on the ‘Send’ icon button to deliver their desired message to the coaches

  ○ SEQ-2.1: If user chooses to unsend their message, user press and hold on their message and clicks on the ‘Unsend’ icon button.

  ○ SEQ-2.2: If user chooses to edit their message, user press and hold on their message and clicks on the ‘Edit’ icon button.

● Functional Requirements:

  ○ REQ-1: System must be able to enable real-time communication between users, providing instant message delivery and reception.

  ○ REQ-2: System must be able to indicate whether users are online, offline, or inactive to help others gauge their availability for chat.

  ○ REQ-3: System must be able to support basic text formatting (e.g., bold, italics) to enhance the expressiveness of messages.

  ○ REQ-4: System must be able to allow users to edit or delete their own messages to correct errors.

  ○ REQ-5: System must be able to allow users to view past messages, ensuring continuity in conversations.

  ○ REQ-6: System must be able to provide the option to block or mute other users to prevent unwanted messages.

1.4.6 Taxi Availability

Description and Priority:

○ Allow users to search for nearby taxis and book a taxi

○ Priority: Medium

● Stimulus/Response Sequences:

  ○ SEQ-1: User clicks on the ‘Taxis’ navigation button and enters a valid input.

  ○ SEQ-2: System retrieves user’s current device location and displays all available nearby taxis.

  ○ SEQ-3: User inputs a location to show the available taxis nearby the input location.

  ○ SEQ-4: User clicks the 'Book a Taxi’ button to book a nearby taxi.

● Functional Requirements:

  ○ REQ-1: System must be able to navigate the user to the Taxis page with no hiccups or friction and provide validation when user input is invalid.

  ○ REQ-2: System must be able to retrieve user’s current location.

  ○ REQ-3: System must be able to show all nearby taxis on the map.

  ○ REQ-4: System must be able to show that a taxi is booked when the book taxi button is clicked.

1.5 Non-Functional Requirements

1.5.1 Performance Requirements

- The system must be able to return Flask query results to the user within 3-8 seconds. The system must be able to support 10,000 concurrent user visits at the same time while maintaining optimal performance.
- The Google map should be able to render all locations within Singapore.
- During the initial loading of the webpage, the system shall load the page in less than 3 seconds over a 56 Kbps modem to ensure a fast page load and a good SEO score.
- The application will run on all web browsers.
- The system must be able to accurately detect the live location of the user with a maximum error of 50 metres.
- The system must be able to load up the messages between users within 10 seconds.

1.5.2 Safety Requirements

- The system must be able to ensure data integrity within the system. This can be achieved by implementing measures such as error handling, data validation, and correction.

1.5.3 Security Requirements

- Users are required to log in with their username and password.
- Users are to keep their password safe and not share it with any other people, applications, or websites under any circumstances.
- The system must remind users not to leak their personal information such as addresses over the web application.
- The system must ensure that sensitive or personal data can only be accessed by authorised users.
- The system must ensure that data is protected in accordance with PDPA.
- The system must lock an account after numerous failed login's attempts in order to protect a user's information.
- The system must hash the user's credential information using Secure Hash Algorithm (SHA) before storing them on the database, without displaying them explicitly to the database administrators.

1.5.4 Reliability Requirements

- Within 3 minutes of a system reboot, the entire system must be restored to its full functionality.
- The system should not experience critical failure under normal usage in 95% of use cases.
- The system uptime must be 99% or higher with no more than 3 hours of planned downtime per year.
- The system must provide up-to-date information on the car park availability and activity availability.

1.5.5 Usability Requirements

- The user must be able to navigate smoothly between pages
- The user can easily navigate the interface by easily understanding how the webpage organises its content.
- The user can easily determine what each feature of the website does.

1.5.6 Localisation Requirements

- The system must be able to display date format in DD/MM/YYYY HH:MM format
- The system must be able to read addresses and postal codes in Singapore.

1.5.7 Software Quality Attributes

Security
Whenever a new user is logged in, a token will be generated as part of the account log in process and this token is stored as a cookie in the browser. This token is essentially a piece of information that proves the user's identity and authorisation to access certain parts of the web application. This token is utilised for every REST API on our backend to ensure that only authorised users are able to access it.

Low Coupling
We focus on reducing the interdependencies between different components or modules within a system. The goal is to make each component as independent as possible, so changes made to one component have minimal or no impact on others. We have defined clear and minimalistic interfaces for communication between components. This reduces the number of dependencies and makes it easier to replace or update components. Instead of hard-coding dependencies, we use dependency injection to provide required dependencies to components. This promotes flexibility and testability. In addition, in a multi-layered architecture, we keep communication between layers loosely coupled. For example, the presentation layer, business logic layer, and data access layer are separated with well-defined interfaces.

High Cohesive
We encourage grouping related functions and responsibilities within a module or component. We incorporated a single, well-defined purpose and should perform its tasks without unnecessary overlap. Our functionality is concentrated and clear such that our website is easy to test and maintain. Highly cohesive components are more likely to be reusable in different parts of our website.

Resource Optimisation
Within our system, we employ a variety of techniques to enhance the efficient use of resources, ultimately resulting in latency reduction and improved system performance. One key method is the strategic utilisation of UseEffect hooks, which allow us to initiate function calls selectively, only when specific states undergo changes. This approach significantly reduces the overall volume of API calls made. The advantage lies in the fact that API calls are avoided during routine page re-renders, resulting in a conservation of valuable system resources and, consequently, a more resource-efficient and responsive system.

1.5.8 Benefits of our Technology Stack

Our technology stack encompasses widely adopted web development frameworks, providing our clients with the convenience of future maintenance and upgrades without the need for specialised developers. Moreover, we harness the capabilities of React.js, which empower us to craft modular and reusable user interface components, exemplified by a Navigation bar that seamlessly integrates across all webpages, eliminating repetitive code duplication.

Furthermore, our Flask backend embodies fundamental design principles that underpin the efficiency and maintainability of our web application:

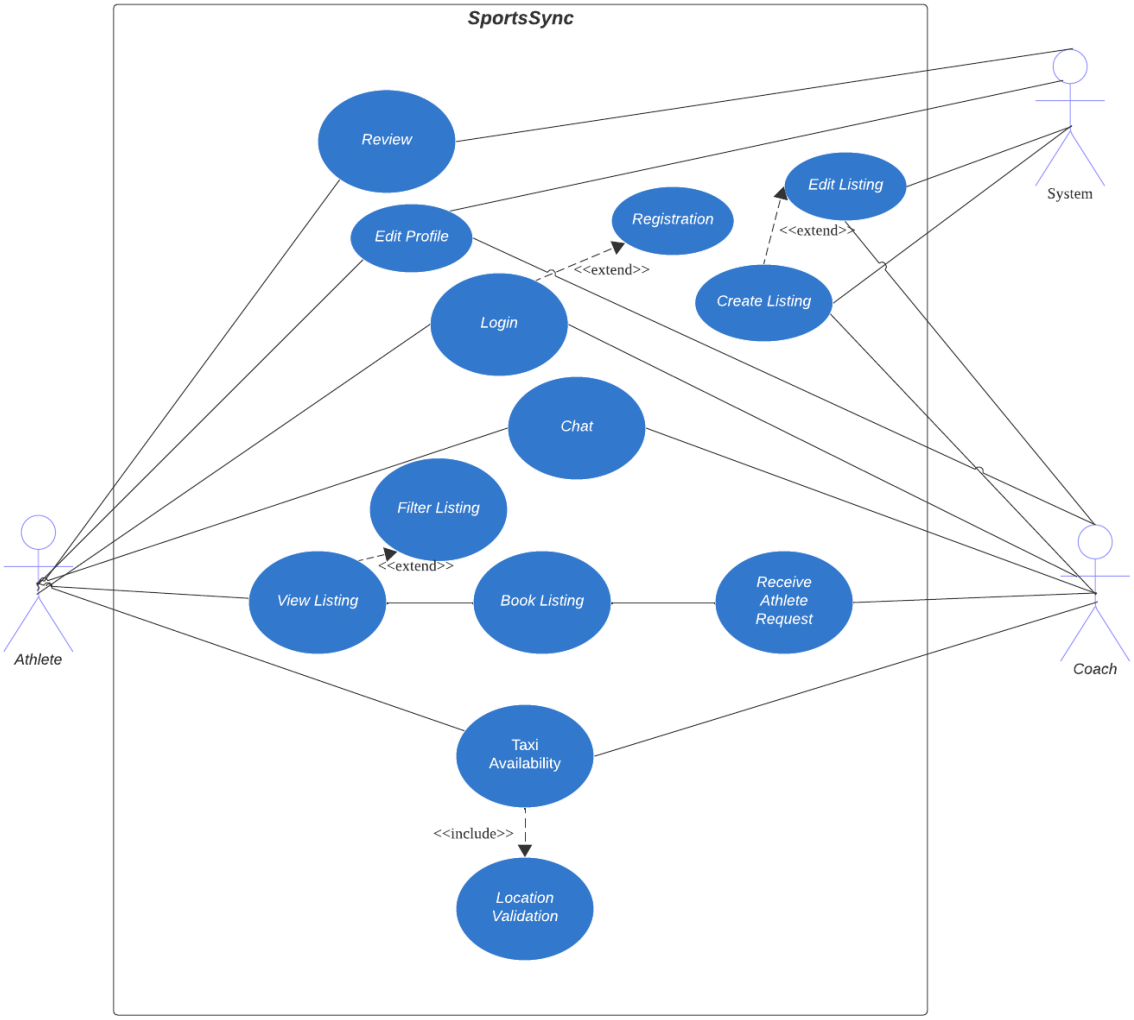
> **Don't Repeat Yourself (DRY)**: Flask endorses the DRY principle, facilitating the definition of routes and views through decorators while emphasising concise code. This approach curbs redundancy, fosters code reusability, and guarantees that modifications manifest uniformly throughout the application.
>
> **Separation of Concerns**: Flask, while affording flexibility, promotes the segmentation of various aspects within the application. This involves isolating routes, views, and data models into separate modules or packages, fostering a clean and easily maintainable codebase. Such segregation not only streamlines development but also simplifies future adjustments.
>
> **RESTful Design**: Flask excels in constructing RESTful APIs, aligning with a set of design principles geared toward creating scalable, consistent, and maintainable web services. This approach enhances communication between different components and facilitates integration with external services and platforms, elevating the application's versatility.

In tandem with these design principles, Vite serves as our JavaScript application build tool, amplifying development efficiency with rapid code bundling using modern ES modules. Vite's array of features, including hot module replacement and a versatile plugin system, endow our developers with a dynamic environment, enabling real-time code adjustments without full application reloads and enabling tailored build processes, ultimately expediting compilation times. This technology stack assures that our web application not only brims with features but also adheres to robust design principles, ensuring ease of maintenance and extensibility.

1.5.9 Business Rules

1. Developer
    a. Developers can remove users that do not abide by the web application guidelines
    b. Developers can remove listing the breaches the web application guidelines
2. User
    a. User can make adjustment to their profile details, such as username, bio, profile picture
    b. User can make adjustment to their listing details, such as description, pricing, and activity picture
    c. User can manage their listing
        i. Delete their own listing
        ii. Delete their booked listing
        iii. Delete their saved listing

1.6 Use Case Diagram



1.7 Use Cases Descriptions

Use Case ID:	1.0		
Use Case Name:	Registration		
Created By:	Andrew Cheam	Last Updated By:	Andrew Cheam
Date Created:	31/08/2023	Date Last Updated:	08/11/2023

Actor:	User
Description:	User registers for an account in the systems database. A valid username and password is required for registration.
Preconditions:	1. User has not created an account
Postconditions:	1. New User account is created and added to database
Priority:	High
Frequency of Use:	Once for each user
Flow of Events:	1. User clicks on the register button 2. Users are prompted to input the following details - First Name, Last Name, Email, Username, Password, Re-enter Password, Gender, Date of Birth. 3. System checks if there is existing email and username in the database 4. System checks if the password is valid. A valid password is at least 8 characters long and a combination of uppercase letters, lowercase letters, numbers, and symbols. 5. User clicks on the “Register New Account” button 6. System creates a User account and adds information entered by the User into the database and ChatAPI database 7. User goes into the login page
Alternative Flows:	**AF-3: Username entered is not unique** 1. The UI will display an alert stating “A user with that username already exists.” 2. UI returns to Step 1 **AF-3: Entered Email is invalid** 1. The UI displays the message “Please include an “@” in the email address.” 2. UI returns to Step 1 **AF-4: Entered Password does not meet requirements** 1. The UI displays the following messages: a. “Password is not the same” b. “Password must at least 8 characters long” c. “Password must have at least 1 number” d. “Password must have at least 1 upper case” e. “Password must have at least 1 special character” 2. UI returns to Step 1
Exceptions:	-

Includes:	-
Special Requirements:	-
Assumptions:	-
Notes and Issues:	-

Use Case ID:	2.0		
Use Case Name:	Login		
Created By:	Brandon Jang	Last Updated By:	Brandon Jang
Date Created:	31/08/2023	Date Last Updated:	08/11/23

Actor:	User
Description:	Registered User logs in to their account. The username and password must match the records in the database for a user to login
Preconditions:	1. User has created an account 2. Username and Password entered correctly
Postconditions:	1. User account is moved into their account home page
Priority:	High
Frequency of Use:	Medium
Flow of Events:	1. User clicks on the login button 2. User is prompted to input Username and Password 3. System verifies that the Username and Password matches the records in the database 4. User is logged in to the account and sees the home page
Alternative Flows:	**AF-1: User has not created an account and password** 1. The UI will display an alert stating “Invalid Credentials” 2. UI returns to Step 1 3. User clicks on register button and goes to Use Case 1
Exceptions:	-
Includes:	-
Special Requirements:	-
Assumptions:	-
Notes and Issues:	-

Use Case ID:	3.0		
Use Case Name:	Create Listing		
Created By:	Andrew Cheam	Last Updated By:	Andrew Cheam
Date Created:	31/08/2023	Date Last Updated:	08/11/2023

Actor:	User (Coach)
Description:	Registered User creates a listing on the marketplace and edits listing
Preconditions:	1. User has created an account 2. User has logged in to their account
Postconditions:	1. A listing has been added to the database
Priority:	High
Frequency of Use:	Medium
Flow of Events:	1. User clicks on the “Create New Listing” button 2. User is prompted to input the following details - Price, Location, Date and Time, Number of Slots, Sport, Proficiency Level, Description, Add a Photo 3. User clicks on the “Create Listing” button 4. System adds the information of listing into the database and group chat adds to ChatAPI database 5. Owner of listing clicks into listing to view the information 6. Owner of listing has option to delete listing
Alternative Flows:	**AF-2: User does not input all required fields for listing** 1. UI displays the message “Please fill up this field.” 2. UI returns to Step 1
Exceptions:	-
Includes:	-
Special Requirements:	-
Assumptions:	-
Notes and Issues:	-

Use Case ID:	4.0		
Use Case Name:	Edit Listing		
Created By:	Brandon Jang	Last Updated By:	Brandon Jang
Date Created:	31/08/2023	Date Last Updated:	08/11/2023

Actor:	User (Coach)
Description:	Registered User creates a listing on the marketplace and edits listing
Preconditions:	1. User has created an account 2. User has logged in to their account
Postconditions:	1. A listing has been added to the database
Priority:	High
Frequency of Use:	Medium
Flow of Events:	1. User clicks on the “Edit” button 2. User is prompted to input the following details - Price, Location, Date and Time, Number of Slots, Sport, Proficiency Level, Description, Add a Photo 3. System edits the information of listing into the database 4. Owner of listing clicks into listing to view the information 5. Owner of listing has option to delete listing
Alternative Flows:	**AF-2: User does not input all required fields for listing** 1. UI displays the message “Please fill up this field.” 2. UI returns to Step 1
Exceptions:	-
Includes:	-
Special Requirements:	-
Assumptions:	-
Notes and Issues:	-

Use Case ID:	5.0		
Use Case Name:	View Listing		
Created By:	Andrew Cheam	Last Updated By:	Andrew Cheam
Date Created:	31/08/2023	Date Last Updated:	08/11/2023

Actor:	User (Athlete)
Description:	Registered User views listings on the marketplace
Preconditions:	1. User has created an account 2. User has logged in to their account
Postconditions:	3. User has a session booked under their account in the database
Priority:	High
Frequency of Use:	Medium
Flow of Events:	1. User clicks on the 'Activities' button 2. System displays all listings. The following details are shown - coach profile, sport, and activity cover image 3. User clicks on the listing to view all available information
Alternative Flows:	**AF-1: User makes use of filter to search** 1. User clicks on the "Filter" button after searching 2. System displays multiple filters regarding the following - Proficiency Level, Sport, Price 3. User selects options for various filters and searches the marketplace 4. System displays the results that match the filters that are in place 5. UI returns to Step 2
Exceptions:	-
Includes:	-
Special Requirements:	-
Assumptions:	-
Notes and Issues:	-

Use Case ID:	6.0		
Use Case Name:	Book Listing		
Created By:	Brandon Jang	Last Updated By:	Brandon Jang
Date Created:	31/08/2023	Date Last Updated:	08/11/2023

Actor:	User (Athlete)
Description:	Registered User searches a listing on the marketplace and selects the listing
Preconditions:	1. User has created an account 2. User has logged in to their account 3. User has selected a specific listing
Postconditions:	1. User has a session booked under their account in the database 2. Available slots in listing decreases by 1
Priority:	High
Frequency of Use:	Medium
Flow of Events:	1. User clicks the “Book Listing” button and the listing is added to their booked sessions in the database 2. User is added as member of listing’s group chat in the ChatAPI database 3. The athlete’s information is saved into the listing’s database
Alternative Flows:	**AF-1: User saves a listing** 1. User clicks the “Save Listing” button and the listing is added to their personal list in the database 2. UI displays the message “You have successfully saved this listing.” 3. UI returns to specific listing page **AF-1: User booked the service again** 1. UI displays the message “You have already booked this service.” 2. UI returns to specific listing page
Exceptions:	-
Includes:	-
Special Requirements:	-
Assumptions:	-
Notes and Issues:	-

Use Case ID:	7.0		
Use Case Name:	Review		
Created By:	Andrew Cheam	Last Updated By:	Andrew Cheam
Date Created:	31/08/2023	Date Last Updated:	08/11/2023

Actor:	Athletes
Description:	To allow the Athletes to rate the services of coaches that they seek services from. This will help to eradicate unethical or uncommitted coaches
Preconditions:	1. User has created an account 2. User has logged in to their account 3. User has accepted the services from the coach's listing
Postconditions:	1. All reviews sent are immediately stored in the database and displayed to the other users in the website
Priority:	Medium
Frequency of Use:	Medium
Flow of Events:	1. Service from the Coach is accepted by the User 2. User clicks on the 'Review' button and give their desired stars out of five. 3. User clicks on the 'Description' input to write their desired message about the coach. 4. User clicks on the "Rating" bar to give the number of ratings to the coaching service. 5. User clicks on the 'add review' button to upload their review about the coach and click on the 'Edit' icon button. 6. Review is saved in the database and updated to all other Users in the website 7. User clicks on the 'Stars' icon to view their review and view review histories about the coach.
Alternative Flows:	**AF-1: User has reviewed once** 1. UI displays the message "You have reviewed the coach."
Exceptions:	-
Includes:	-
Special Requirements:	-
Assumptions:	-
Notes and Issues:	-

Use Case ID:	8.0		
Use Case Name:	Location Validation		
Created By:	Brandon Jang	Last Updated By:	Brandon Jang
Date Created:	31/08/2023	Date Last Updated:	08/11/2023

Actor:	System
Description:	validateLocation takes in a string input containing a location name. It calls the google maps validate API to validate the input
Preconditions:	1. Called from: a. Use Case 10.0: Taxi Availability
Postconditions:	1. GeoCode containing the coordinates in the form {lat, lng} is returned to the calling function if the validation is successful 2. An alert is displayed on the screen and nothing is returned to the calling function if the validation is unsuccessful
Priority:	High
Frequency of Use:	High
Flow of Events:	1. validateLocation(location) use case is called from another use case 2. It makes a request to the Google Maps Validate API, passing in the location 3. If the input is a valid location, it returns a GeoCode containing the coordinates for the calling function to process. 4. If the input is an invalid location, it displays an alert asking the user to reenter a valid location
Alternative Flows:	-
Exceptions:	-
Includes:	-
Special Requirements:	-
Assumptions:	-
Notes and Issues:	Requires around 100ms for the entire validation call

Use Case ID:	9.0		
Use Case Name:	Taxi Availability		
Created By:	Andrew Cheam	Last Updated By:	Andrew Cheam
Date Created:	31/08/2023	Date Last Updated:	08/11/2023

Actor:	Initiated by User
Description:	User enters a location into the search bar, and the system will display all nearby taxis on the map
Preconditions:	1. User has an account 2. User has successfully logged into account 3. User enters a valid search location
Postconditions:	1. Location Markers are placed on the map indicating the locations of the taxis 2. The markers will remain on the map until a. User searches for a different location b. User clicks away from the taxis availability tab
Priority:	Medium
Frequency of Use:	Medium
Flow of Events:	1. User clicks on the “Taxi” Button in the navigation bar 2. The “Taxi UI” is overlaid onto the Google map behind 3. The user enters a location into the search bar 4. The system returns the real-time location of all taxis within 3km 5. Once returned, the user can click on the “Show taxis nearby” button which drops markers on each taxi’s current location on the map
Alternative Flows:	**AF-3: User enters an invalid location** - validateLocation( ) returns an alert “Please enter a valid location” - No taxis are returned - UI returns to Step 2 **AF-5: No available taxis nearby** - No taxis are displayed to the user - UI returns to Step 2
Exceptions:	-
Includes:	validateLocation(location) where location is a string containing the location entered by the user
Special Requirements:	-
Assumptions:	-
Notes and Issues:	Slight latency of around 200ms due to API calls to validate location and get all taxis

Use Case ID:	10.0		
Use Case Name:	Chat		
Created By:	Brandon Jang	Last Updated By:	Brandon Jang
Date Created:	31/08/2023	Date Last Updated:	08/11/2023

Actor:	User
Description:	To allow the Athletes to communicate with the Coach . This will help facilitate booking of service.
Preconditions:	1. User has created an account 2. User has logged in to their account
Postconditions:	1. All messages sent is immediately displayed to the other users in the chat
Priority:	High
Frequency of Use:	High
Flow of Events:	1. User clicks on “Chat” button in homepage, which loads all prior chatting history 2. User inputs text and clicks on the send icon to send the message 3. Message is updated to Coach in the chat
Alternative Flows:	-
Exceptions:	-
Includes:	-
Special Requirements:	-
Assumptions:	-
Notes and Issues:	-

1.8 Data Dictionary

Term	Definition
User	An individual who uses the application / website.
Athlete	A User who has the intent to book coaching services.
Coach	A User who has the intent to provide coaching services.
Coaching services	A specific instance of a service that a coach would want to provide. This service will have to include price, location, time, sports and other relevant details for athletes to see before they determine whether they are interested in taking up this service.
Marketplace	A platform that showcases all relevant information of all the coaching services available and whereby these coaching services can be booked by athletes.
Account	A combination of a username, password and other user information that allows the application to identify the user. Users must be logged into an account to book or provide coaching services on the platform.
Sign Up	The process whereby a user without an existing account creates one by entering their desired username, password and other information into the website.
Sign In	The process whereby a user enters their account's username and password into the application in order to access its features, given that they already have an existing account.
Username	A string of characters to uniquely identify a user.
Password	A string of characters that is at least 8 characters long with a combination of uppercase letters, lowercase letters, numbers and symbols.
Database	An online spreadsheet which contains all information of each user and coaching services.
Reviews	A scaling system implemented for athletes to rate and give comments on the services provided by the coach.
Chat Function	A messaging platform for athletes and coaches to discuss among themselves about further details of the coaching service that is not stated on the platform.
Map	A function that allows users to view the geography of Singapore.
Taxi Availability	A function that allows users to view nearby available taxis on the map.
Car Park Availability	A function that allows users to view nearby car parks with

	available slots in the area on the map.
Date-Time	Exact Date and Time of a ride in format (DD/MM/YYYY HH:MM).

1.9 UI Mockups

Fig. 1.1: Home Page

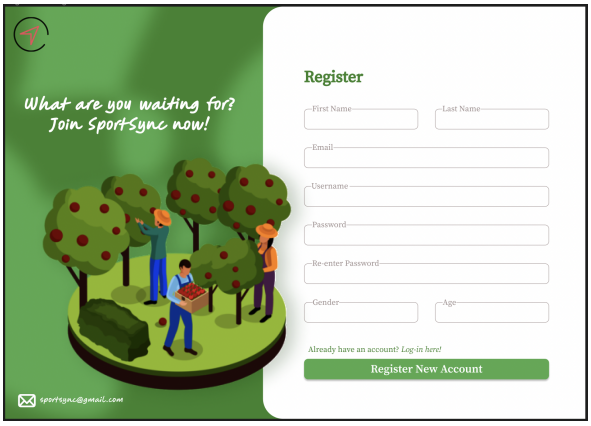


Fig. 1.2: Registration Page

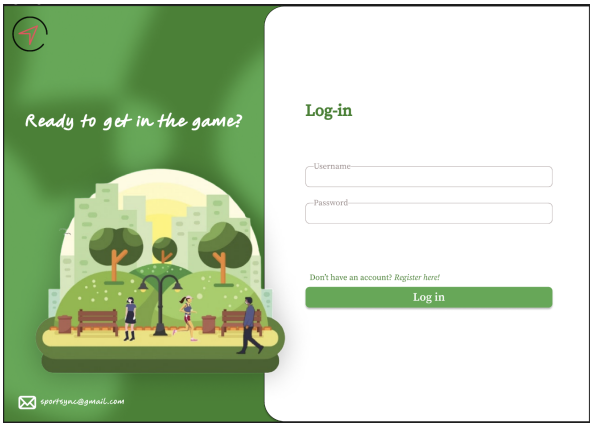


Fig. 1.3: Log-in Page

The figure above shows the user interface for MainMenu, Registration, and Log-in. The design of the screen is kept simple and only essential information is displayed on the screen. This will enable the user to have a seamless onboarding experience with GSL.

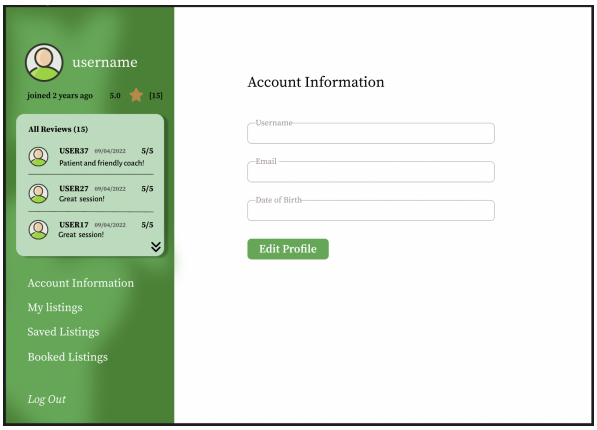


Fig. 1.4: Profile Page

Fig. 1.5: Search & Select Listing through Filter

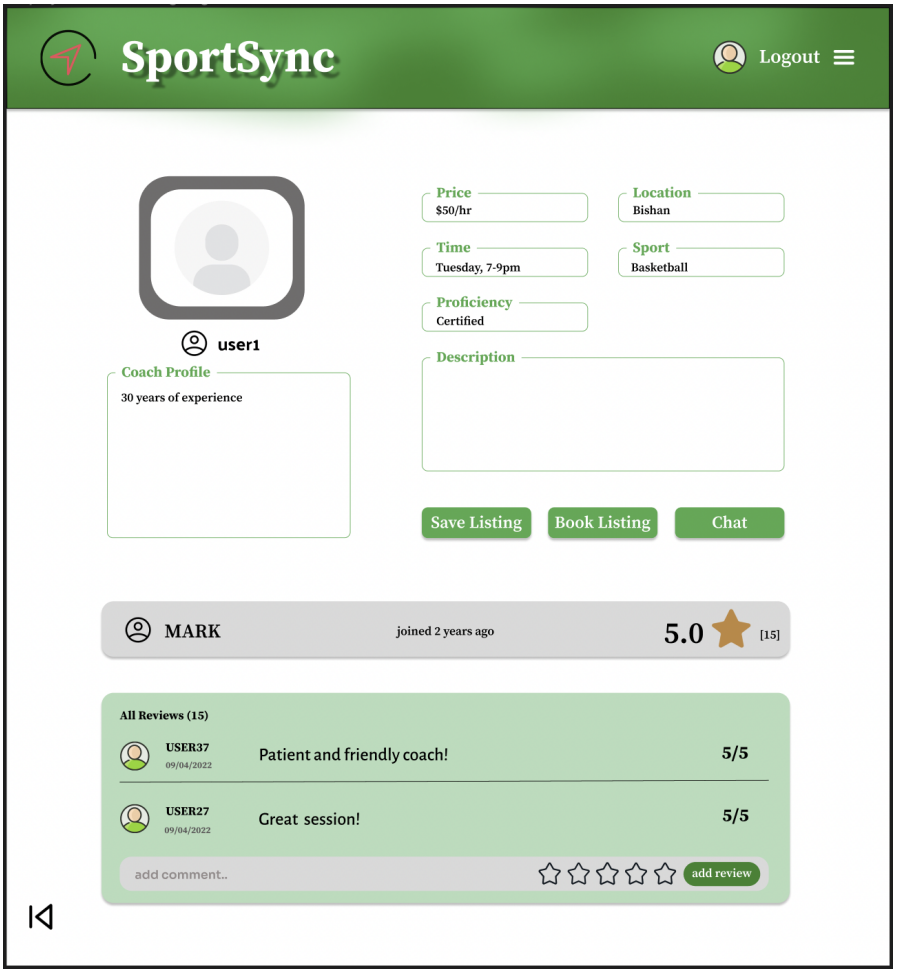


Fig. 1.6: Display of Selected Listing

Users are able to rate and post reviews about their coaches and training sessions. They are also able to view all reviews of the coaches.

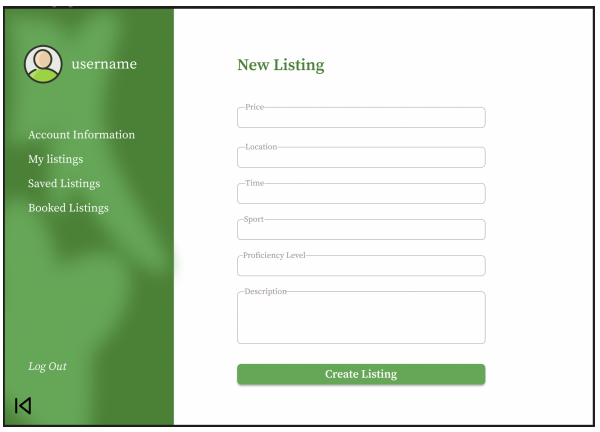


Fig. 1.7: Create Listing

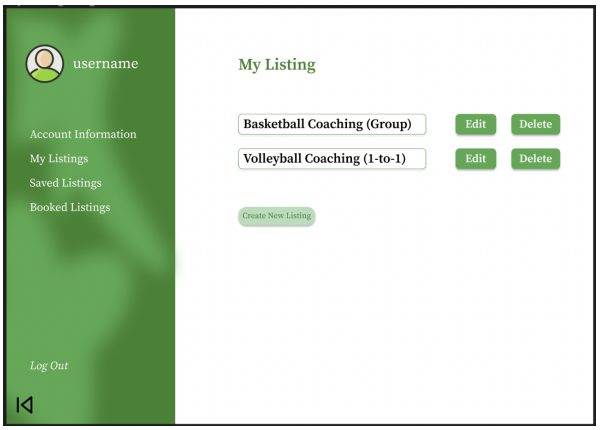


Fig. 1.9: My Listing Page

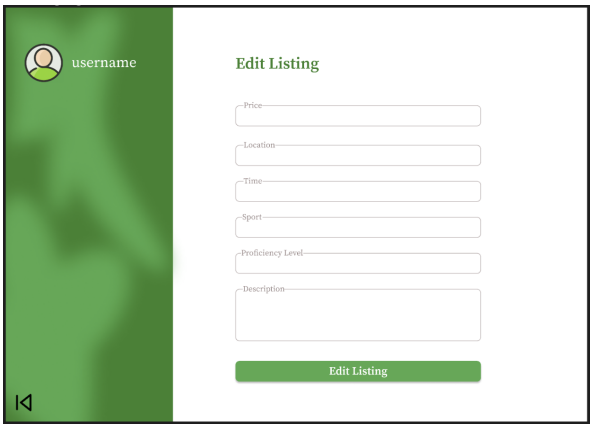


Fig. 1.10: Edit Listing Page

The user is able to view, create and edit their listing details.

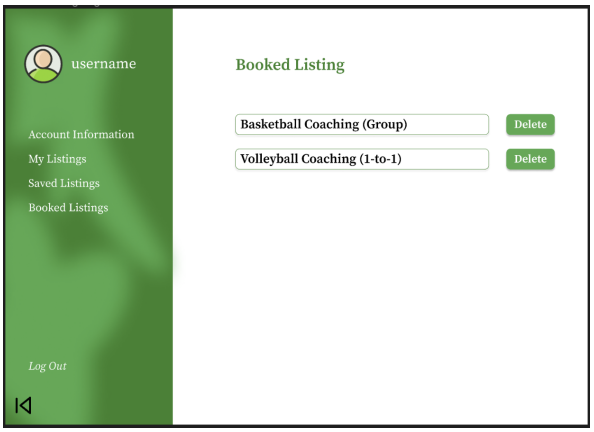


Fig. 1.11: Booked Listing Page

Fig. 1.12: Saved Listing Page

Fig. 1.13 Chat Page

Users are able to chat with the potential coaches that they are interested in.

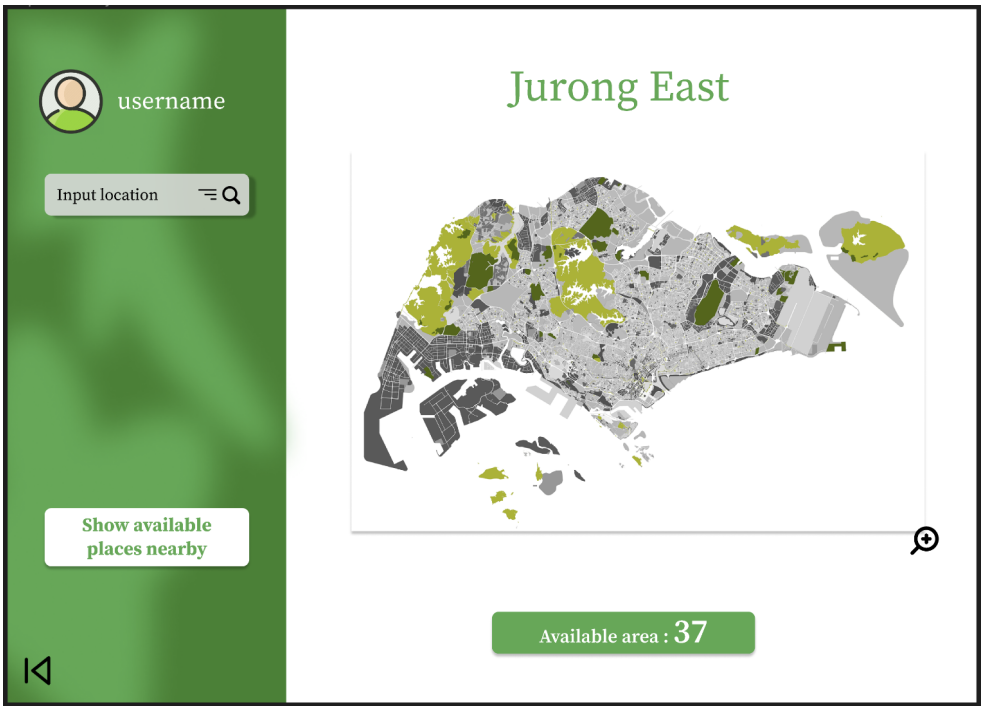


Fig. 1.14 Map Availability Page

Users are able to check out the available sports facilities nearby..

(KIV)

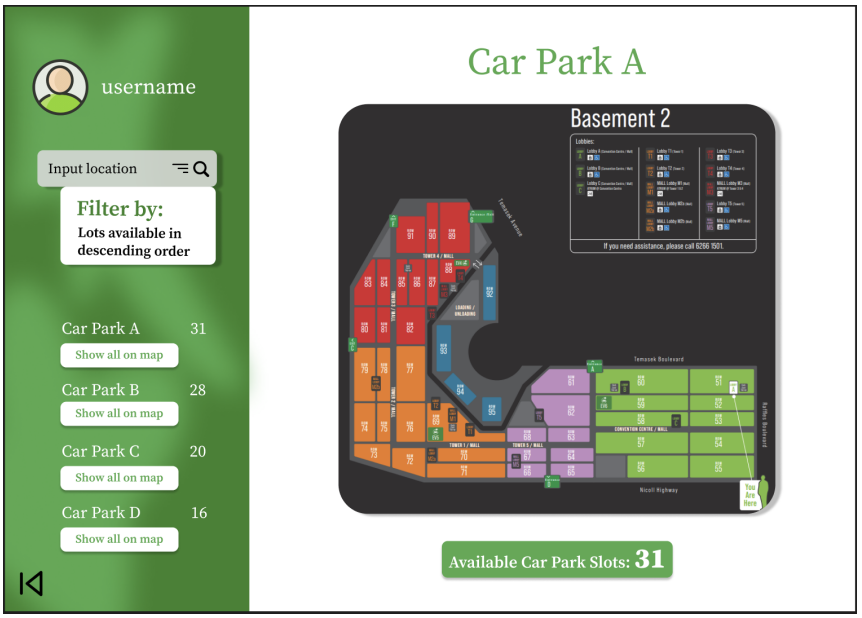


FIg. 1.15: Car Park Availability Page

The Carparks Availability Page shows the list of car parks that are based on the input location. Users will enter the location that they are looking for parking at and click on the search icon. The list will show the relevant car parks near the location indicated and the details of the parking lot such as Name of the Parking Lot, Lots Available at the Parking Lot.

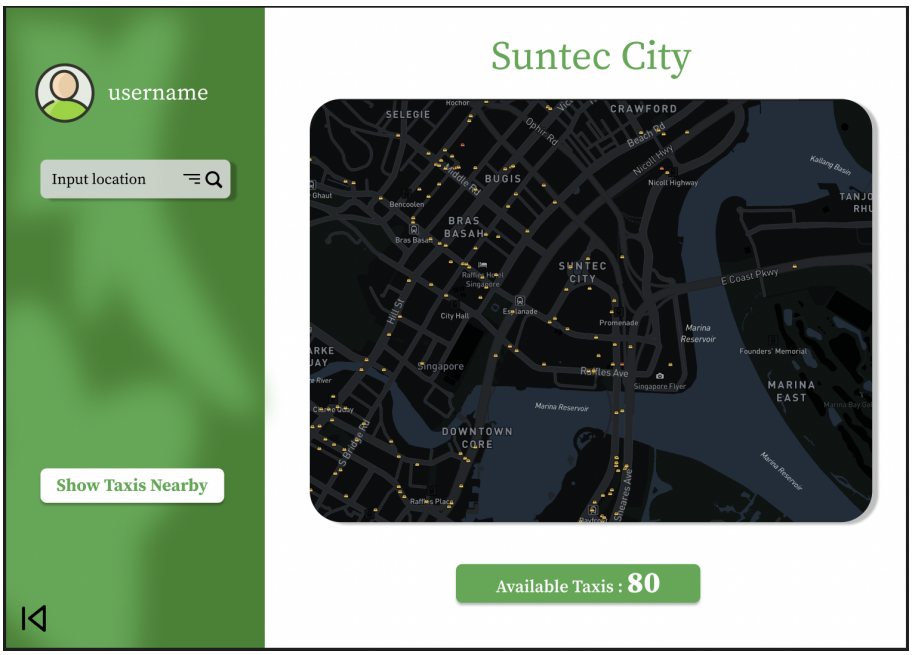


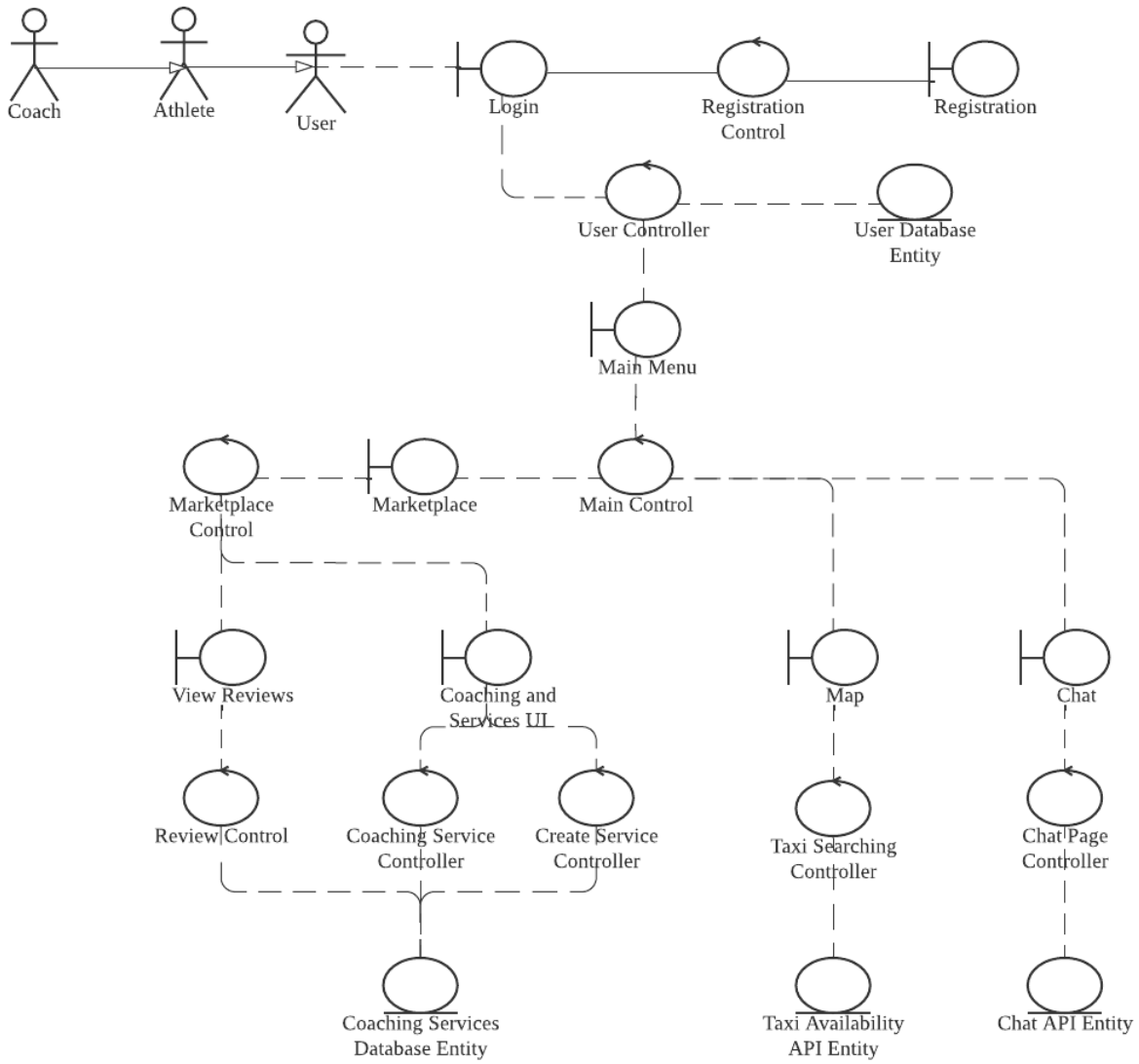
FIg. 1.16: Taxi Availability Page

Users can view the taxis that are nearby the location that they have searched for. The user will input the pick-up location that they want to be picked up at, and the UI will depict the taxis that are near the user's input location.

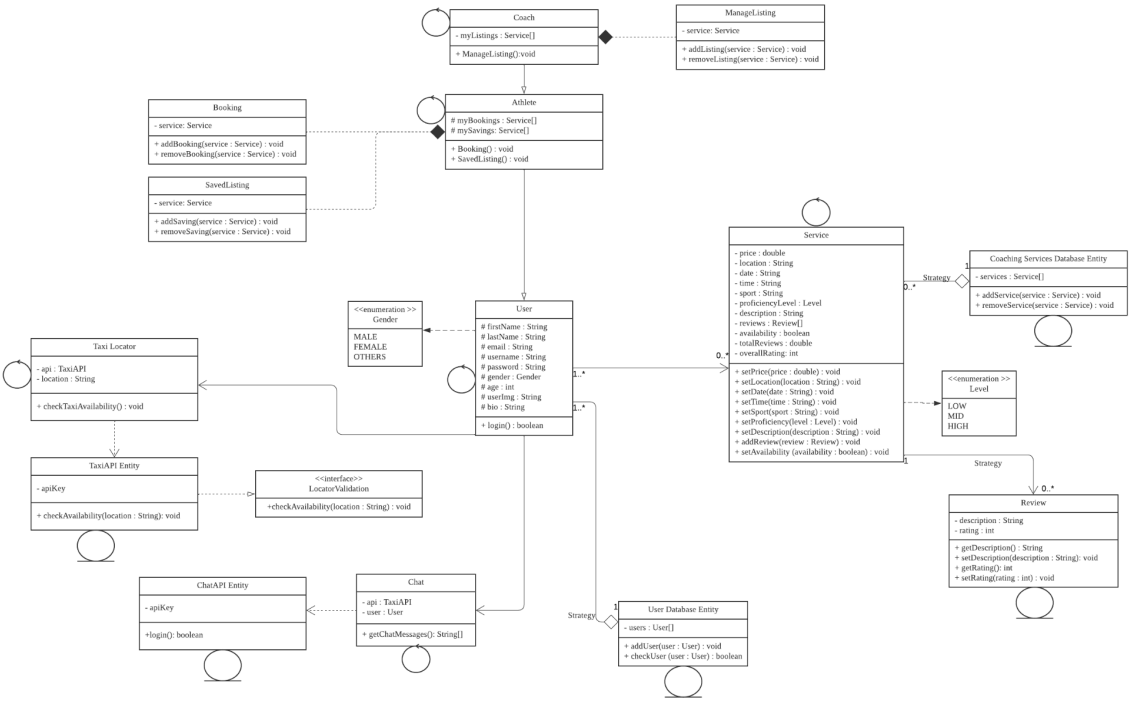
2. Design

2.1 Class Diagram

2.1.1 Stereotype

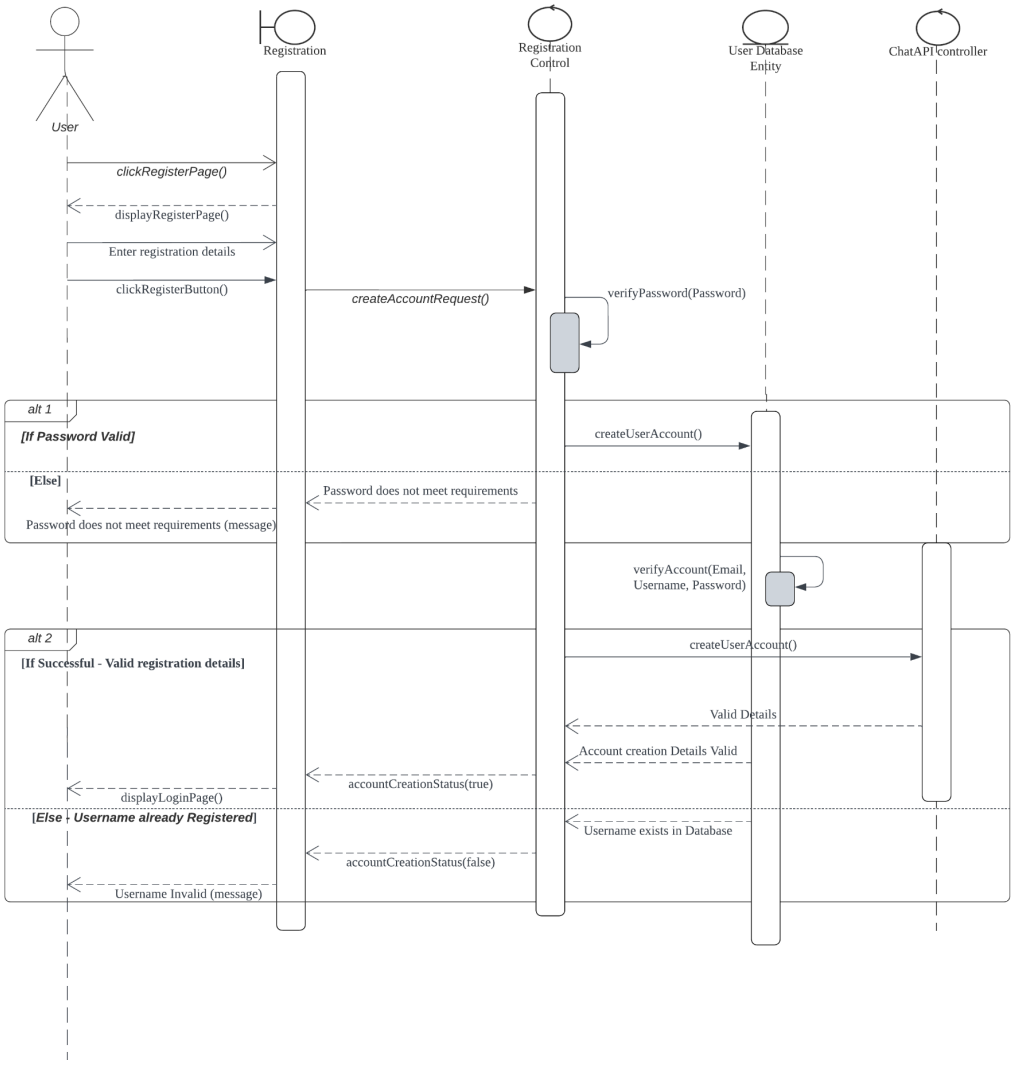


2.1.2 Entity Class Diagram

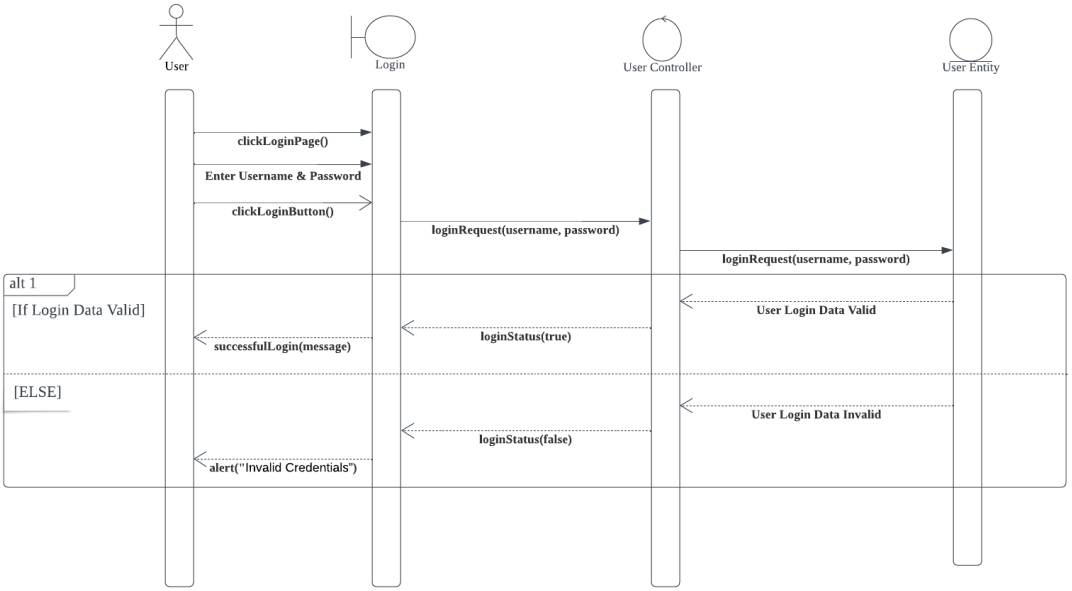


2.2 Sequence Diagram

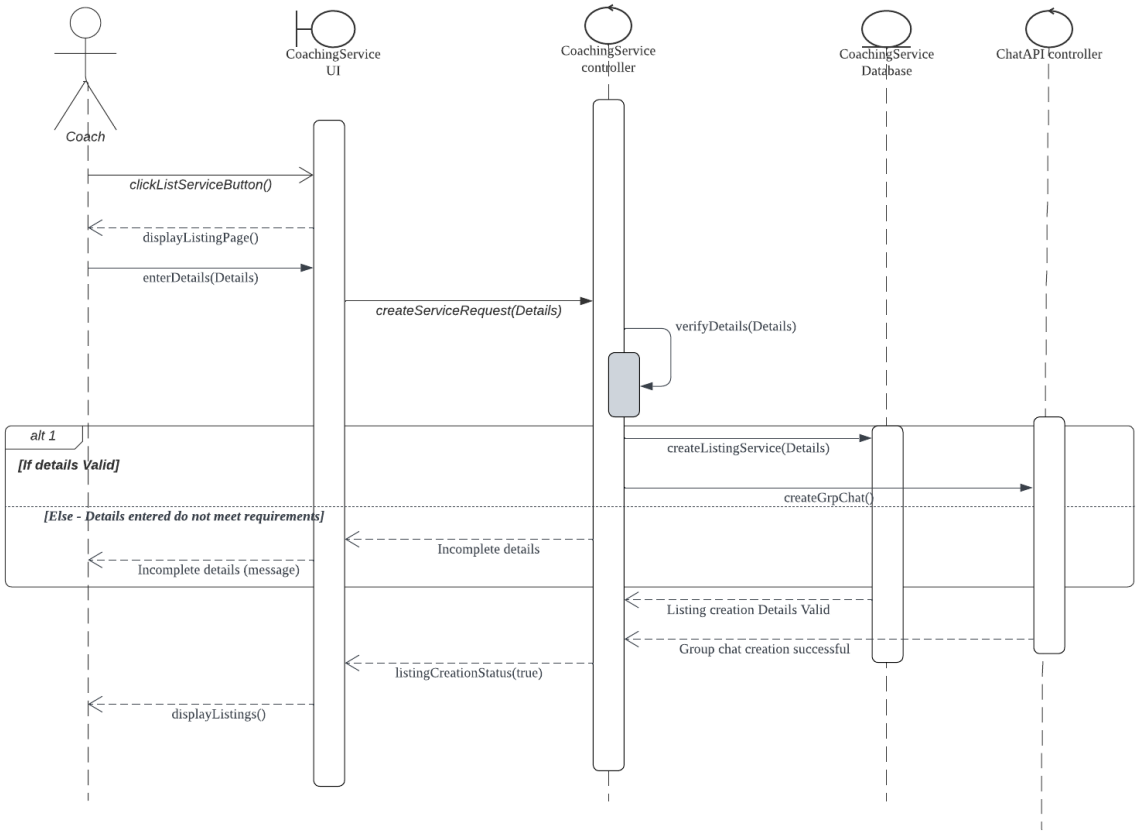
2.2.1 Use Case 1.0 (Registration)



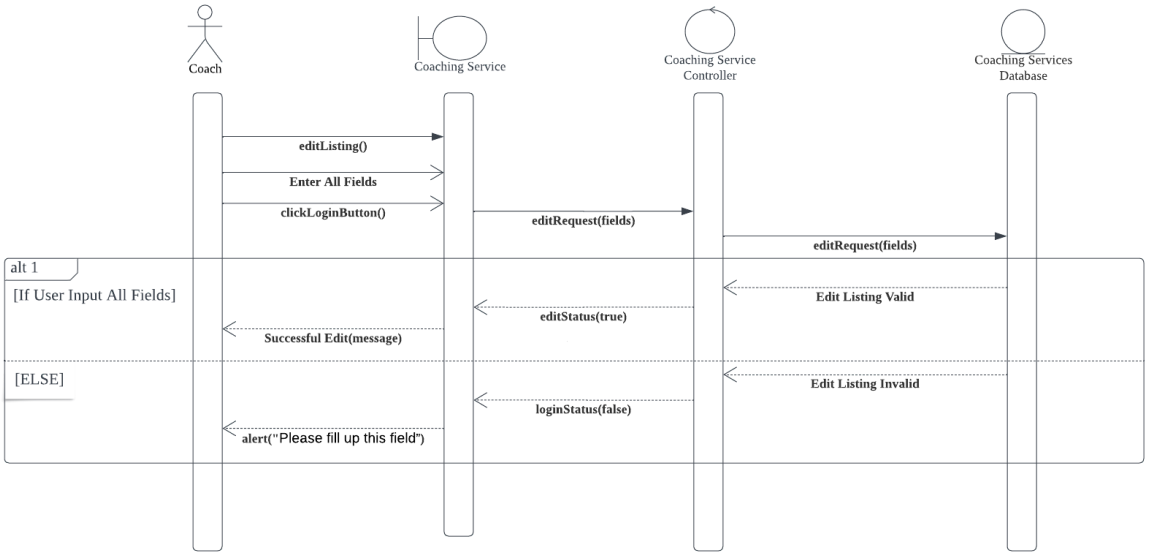
2.2.2 Use Case 2.0 (Login)



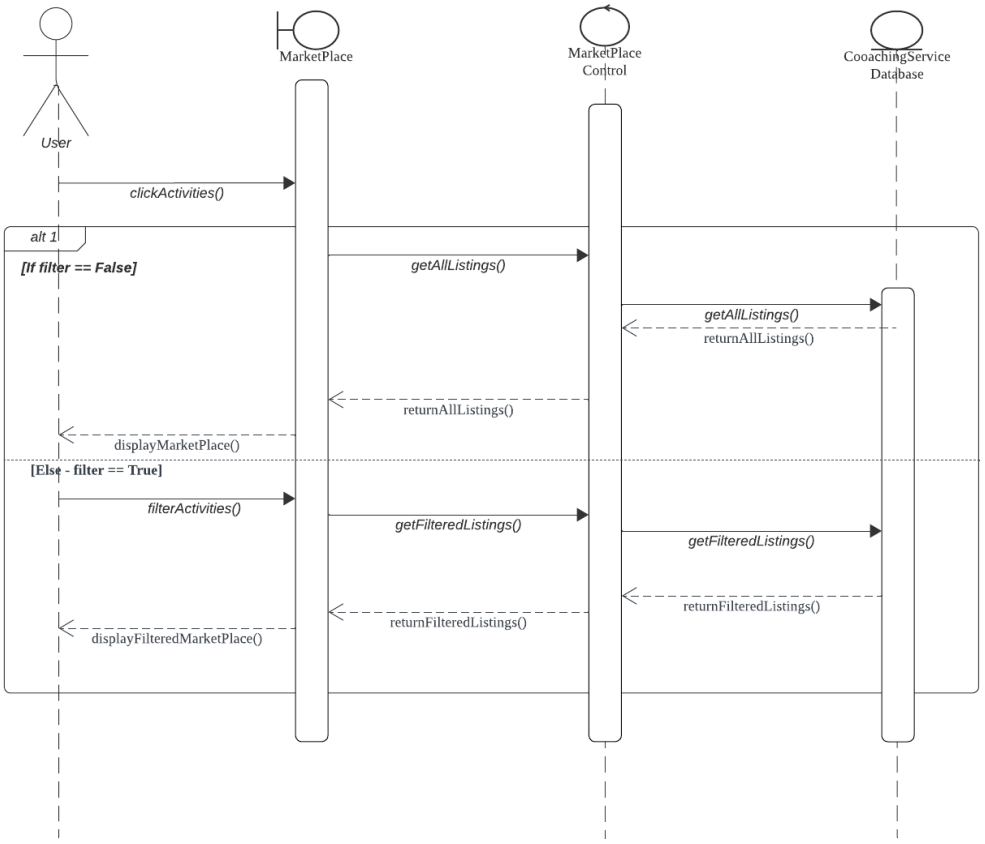
2.2.3 Use Case 3.0 (Create Listing)



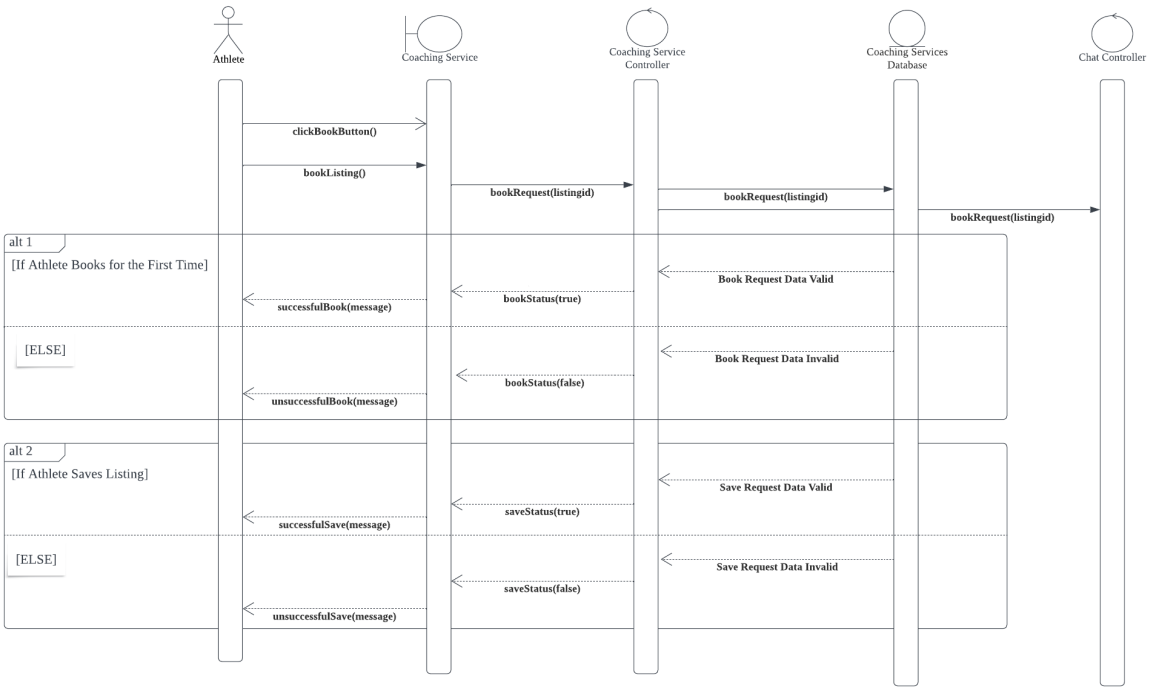
2.2.4 Use Case 4.0 (Edit Listing)



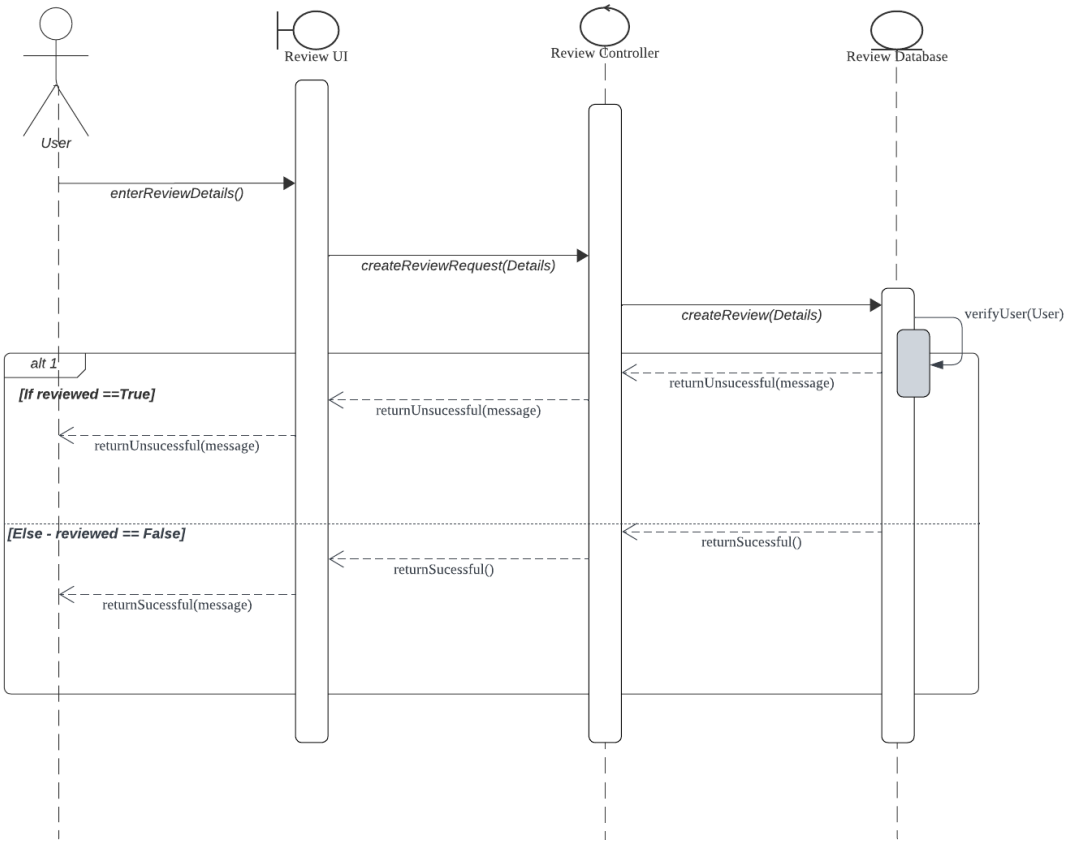
2.2.5 Use Case 5.0 (View Listing)



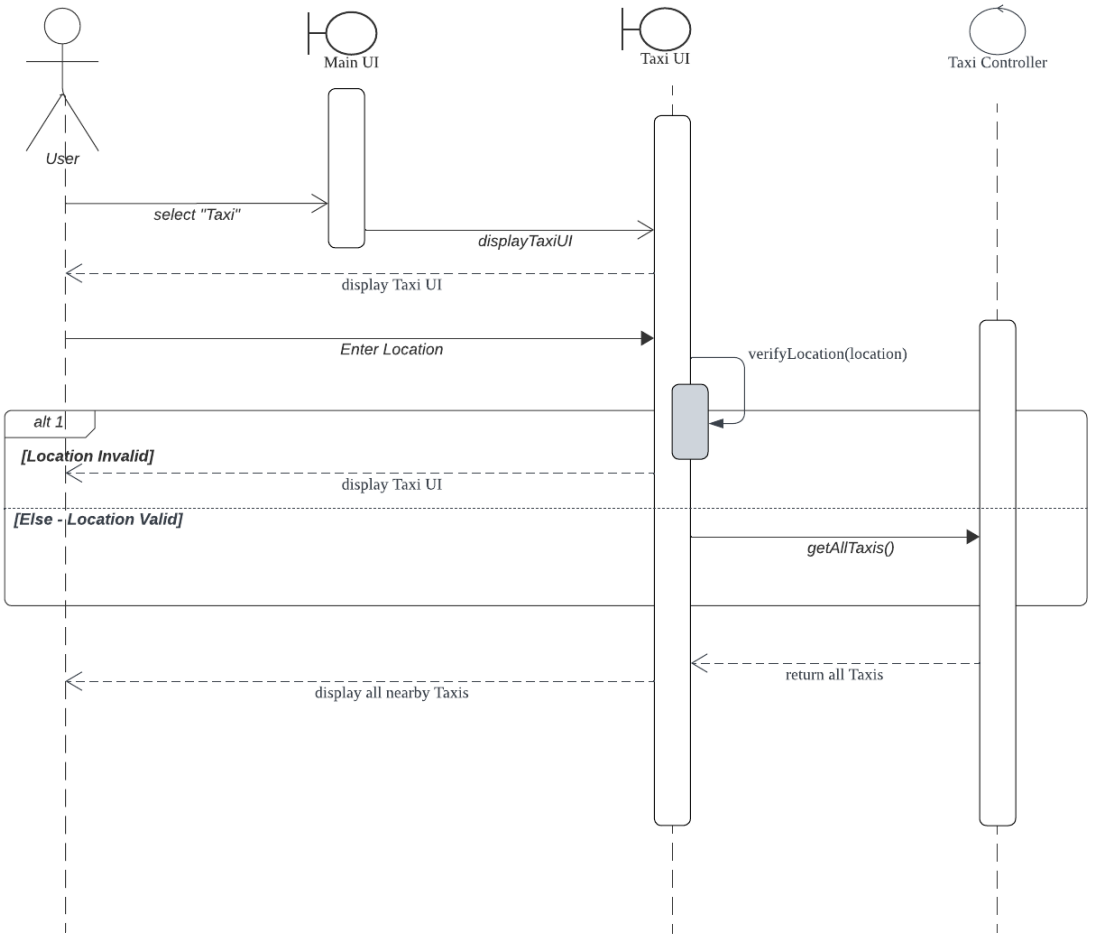
2.2.6 Use Case 6.0 (Book Listing)



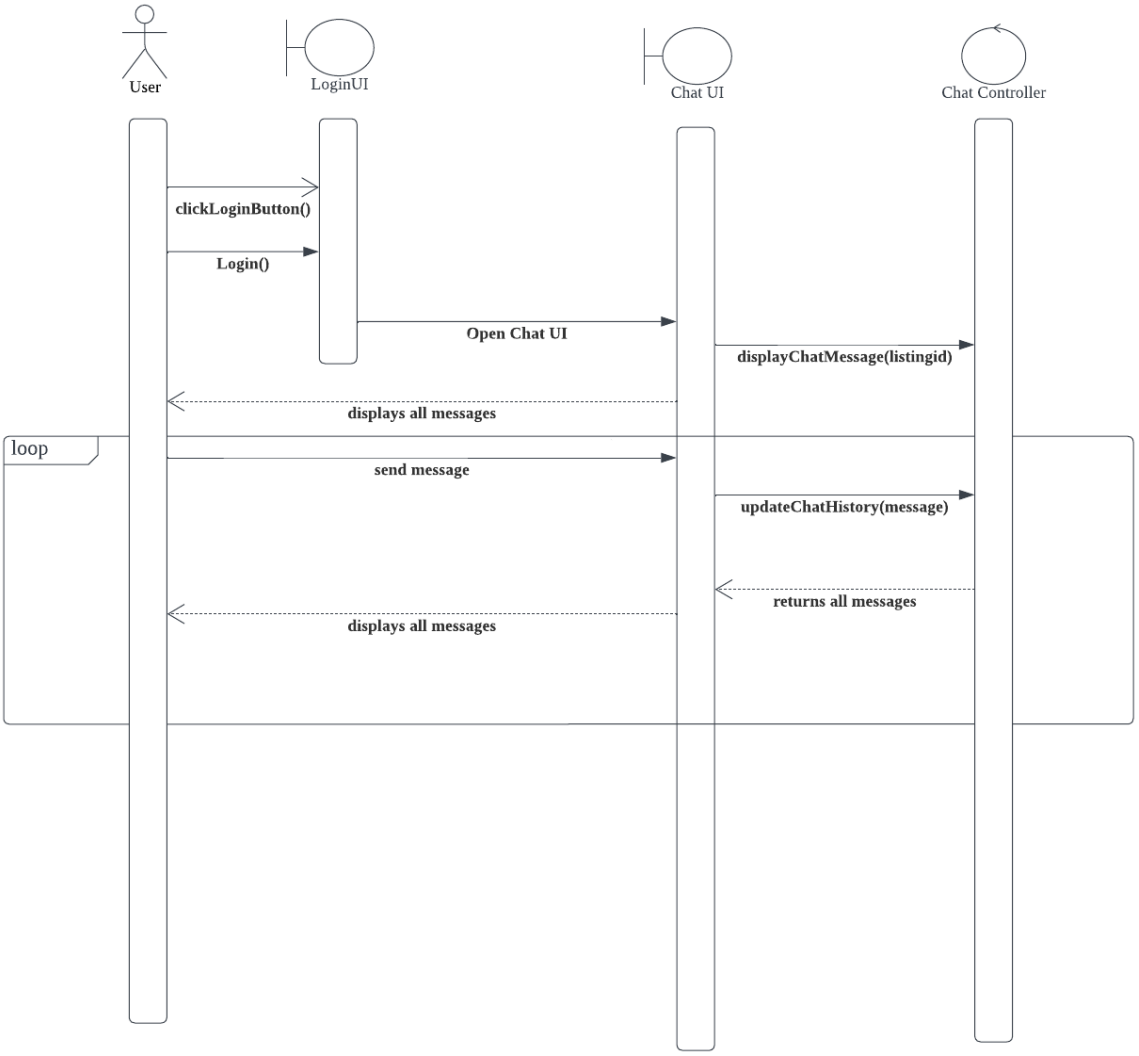
2.2.7 Use Case 7.0 (Review)



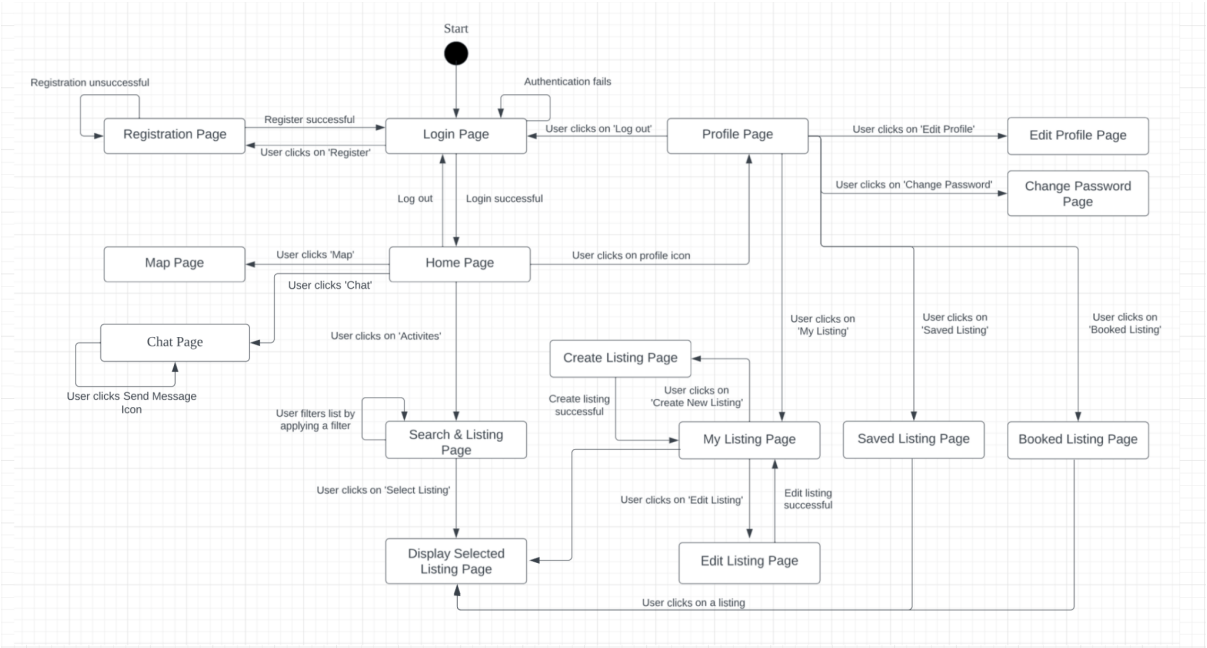
2.2.8 Use Case 9.0 (Taxi Availability)



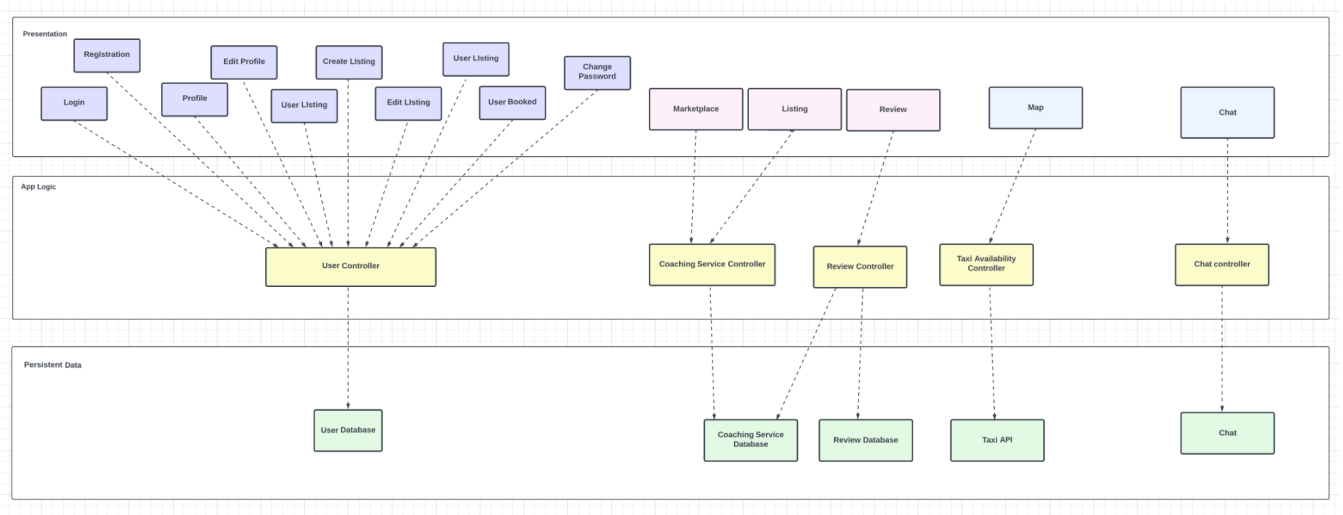
2.2.9 Use case 10.0 (Chat)



2.3 Dialog Map



2.4 System Architecture Diagram



3. Implementation

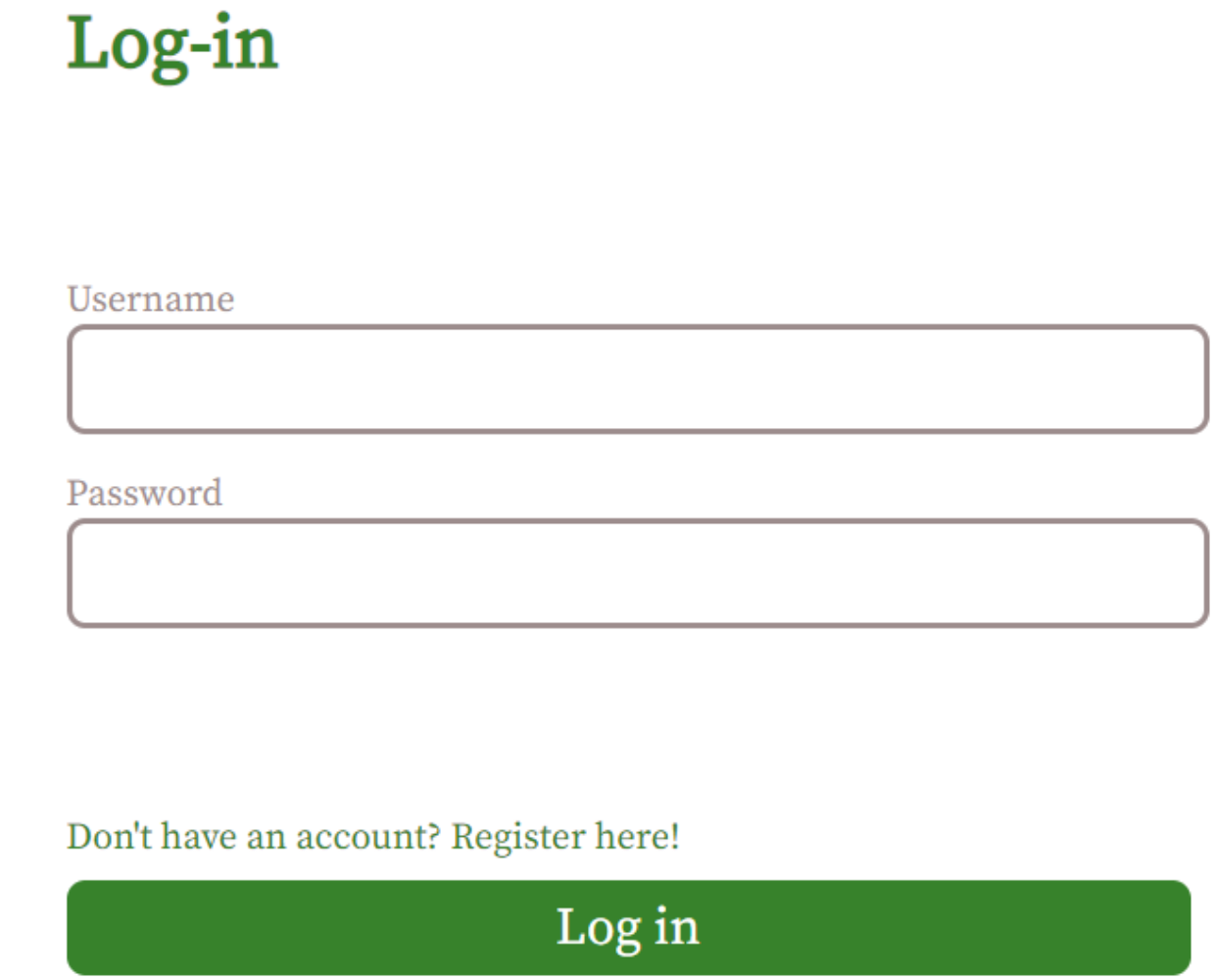
3.1 Application Skeleton

3.1.1 Boundary Classes

3.1.1.1 Login

```jsx
export default function loginUI() {
    const data = useActionData()
    return (
        <div className = "login-background">
            <LoginSidebar />
            <div className = "loginUI">
                <Form className = "login-form" method = "post" action = "/">
                    <h1 className = "login-form-text">Log-in</h1>
                    <div className = "login-form-username">
                        <label className = "login-form-username-text">Username</label>
                        <input
                            className = "login-form-username-box"
                            type = "text"
                            name = "username"
                            required/>
                    </div>
                    <div className = "login-form-password">
                        <label className = "login-form-password-text">Password</label>
                        <input
                            className = "login-form-password-box"
                            type = "password"
                            name = "password"
                            required
                        />
                    </div>
                    <Link className = "login-to-register" to = "/registration">Don't have an account? Register here!</Link>
                    <button className = "login-button" type = "submit">Log in</button>

                    {data && data.error && <p className = "login-error">{data.error}</p>}

                </Form>
            </div>
        </div>
    )
}
```

The Login Page will have the LoginUI Component. The LoginUI will accept a username and password which will be passed to a loginAction (3.1.1.1.1) to authenthicate.



The above image is how the loginUI component looks like in the final implementation of the website.

3.1.1.1.1 Login Action

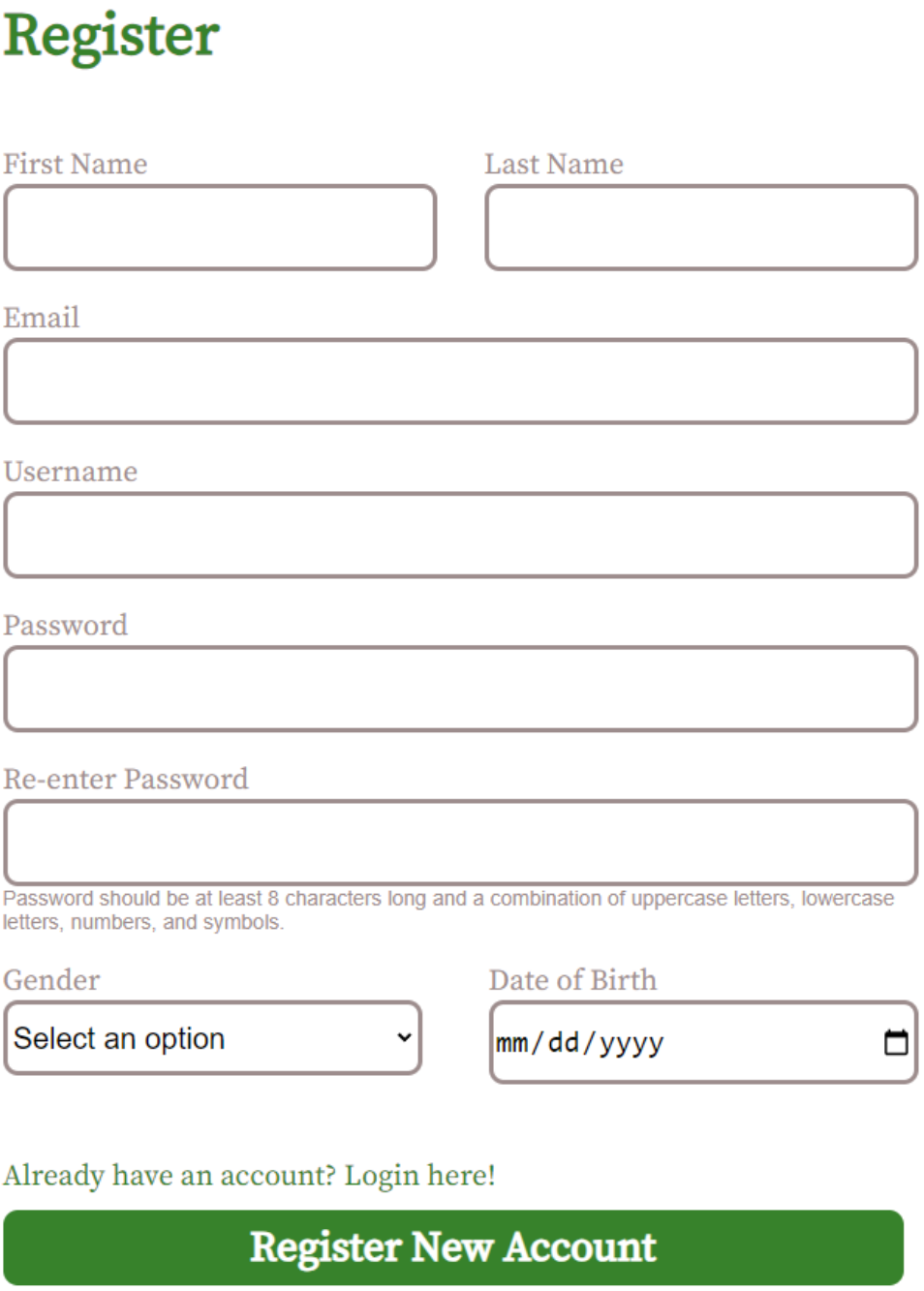
```
export const loginAction = async ({ request }) => {
    const data = await request.formData()

    const submission = {
        username: data.get("username"),
        password: data.get("password")
    }

    console.log(submission)

    try {
        const response = await axios.post(LOGIN_URL, submission);

        if (response.status === 200) {

            console.log(response.data.access_token)
            return redirect("/homepage");
        } else {
            // Handle other response statuses as needed
            return { error: "Login failed" };
        }
    } catch (error) {
        // Handle any Axios request error, such as network issues or server unavailability
        console.log(error)
        const errorMessage = error.response.data.message
        return { error: errorMessage };
    }
}
```

Login action takes in the form information that was submitted and passes it to the user control class (9.2.1). If the information is within the database, it will return back and redirect the user to the homepage. If information is not within the database, it will pass an error message back to loginUI to display.

3.1.1.2 Registration

```
export default function RegisterUI() {
    const data = useActionData()

    return (
        <div className = "register-background">
            <RegisterSidebar />
            <div className = "registerUI">
                <Form className = "register-form"  method = "post" action = "/registration">
                    <h1 className = "register-form-text">Register</h1>
                    <div className = "register-form-name">
                        <div className = "register-form-firstname">
                                <label className = "register-form-firstname-text">First Name</label>
                                <input className = "register-form-firstname-box" type = "text" name = "firstname" required/>
                        </div>
                        <div className = "register-form-lastname">
                                <label className = "register-form-lastname-text">Last Name</label>
                                <input className = "register-form-lastname-box" type = "text" name = "lastname" required/>
                        </div>
                    </div>
                    <div className = "register-form-email">
                        <label className = "register-form-email-text">Email</label>
                        <input className = "register-form-email-box" type = "email" name = "email" required/>
                    </div>
                    <div className = "register-form-username">
                        <label className = "register-form-username-text">Username</label>
                        <input className = "register-form-username-box" type = "text" name = "username" required/>
                    </div>
                    <div className = "register-form-password">
                        <label className = "register-form-password-text">Password</label>
                        <input className = "register-form-password-box" type = "password" name = "password" required/>
                    </div>
                    <div className = "register-form-password2">
                        <label className = "register-form-password2-text">Re-enter Password</label>
                        <input className = "register-form-password2-box" type = "password" name = "password2" required/>
                        <span className = "msg">Password should be at least 8 characters long and a combination of uppercase letters, lowercase <br />letters, numbers, and symbols.</span>
                    </div>

                    <div className = "register-form-miscellaneous">
                        <div className = "register-form-gender">
                            <label className = "register-form-gender-text">Gender</label>
                            <select className = "register-form-gender-option" name = "gender" required>
                                <option value="" disabled selected>Select an option</option>
                                <option value = "Male">Male</option>
                                <option value = "Female">Female</option>
                                <option value = "Others">Others</option>
                            </select>
                        </div>
                        <div className = "register-form-age">
                            <label className = "register-form-age-text" htmlFor = "dob">Date of Birth</label>
                            <input className = "register-form-age-box" type = "date" name = "dob" required/>
                        </div>
                    </div>
                    <Link className = "register-to-login" to = "/">Already have an account? Login here!</Link>
                    <button className = "register-button" type = "submit">Register New Account</button>

                    {data && data.error && <p className = "register-error">{data.error}</p>}
                </Form>
            </div>
        </div>

    )
}
```

The Registration Page will have the RegistrationUI Component. The RegistrationUI will accept first name, last name, username, email, gender, age and password which will be passed to a registrationAction (3.1.1.2.1) to process.



The above image is how the registerUI component looks like in the final implementation of the website.

3.1.1.2.1 Registration Action

```
export const registerAction = async ({ request }) => {
    const data = await request.formData()

    const password2 = data.get("password2")
    const submission = {
        firstname: data.get("firstname"),
        lastname: data.get("lastname"),
        email: data.get("email"),
        username: data.get("username"),
        password: data.get("password"),
        gender: data.get("gender"),
        dob: data.get("dob"),
        userImg: "",
        bio: "",
    }
    // password checker function
    if (submission.password != password2)
        return {error: "Password is not the same"};
    if (submission.password.length < 8)
        return {error: "Password must at least 8 characters long"};
    if (!(/\d/.test(submission.password)) )
        return {error: "Password must have at least 1 number"};
    if (!(/[A-Z]/.test(submission.password)))
        return {error: "Password must have at least 1 upper case"};
    if (!(/[a-z]/.test(submission.password)))
        return {error: "Password must have at least 1 lower case"};
    if (!(/[!@#$%^&*()_+\-=\[\]{};':"\\|,.<>\/?]+/.test(submission.password)))
        return {error: "Password must have at least 1 special character"};

    try {
        const response = await axios.post(REGISTER_URL, submission);

        // Check the response and handle it as needed
        if (response.status === 200) {
            // Registration was successful; you can perform any necessary actions here
            console.log("success")
            return redirect("/");
        } else {
            // Handle other response statuses as needed
            return { error: "Registration failed" };
        }
    } catch (error) {
        // Handle any Axios request error, such as network issues or server unavailability
        console.log(error)
        const errorMessage = error.response.data.message
        return { error: errorMessage };
    }
```

Registration action takes in the form information that was submitted and first checks that the password that was inputted meets the requirements.
Afterwards, it will submit it to the user control class (3.1.2.1). If registration is successful, it will return back and redirect the user to the login page. If registration is unsuccessful, it will pass back the error message to registerUI to display.

3.1.1.3 Homepage

```
export default function Homepage() {

    const [notification, setNotification] = useState(true)

    useEffect(() => {
        axios.get("/homepage")
            .then((response) => {
                console.log(response.data)
                console.log(response.data.overallNotification)
                setNotification(response.data.overallNotification)
            })
            .catch((error) => {
                console.error("Error fetching data:", error)
            })
    })

    return (
        <div>
            <Navbar
                notification = {notification}
            />
            <HomepageContent />
            <Footer />
        </div>
    )
}
```

Homepage consists of 3 components: a navigation bar (3.1.1.3.1), homepage content (3.1.1.3.2) and a footer. It will also call upon the user control class to see if there are any new changes made to the user's account.

3.1.1.3.1 Navbar Component

```
export default function Navbar(props) {
    if (props.notification == true) {
        var notificationMessage = "NEW!"
    }

    return (
        <nav>
            <div className="nav-left">
                <Link className = "nav-home" to = "/homepage">
                    <img src={SPORTSYNCLOGO} alt = "sportsync-logo" width = "141px" />
                </Link>
                <h3 className="nav-logo_name">SportSync</h3>
            </div>
            <div className="nav-right">
                <p className = "error">{notificationMessage}</p>
                <Link className = "nav-profile" to = "/profile">
                    <img src={HUMANLOGO} alt = "human-logo" width = "98px" height = "97px" />
                </Link>
                <button type="button" className="nav-button-logout">
                    <Link className = "nav-button-logout-link" to = "/">Logout</Link>
                </button>
                <div className="dropdown-container">
                    <img className = "nav-right-dropdown-icon" src = {DROPDOWNMENULOGO} alt = "dropdown-menu-logo"/>
                    <ul className="dropdown-menu-nav">
                        <li><Link to = "/marketplace">Actvities</Link></li>
                        <li><Link to = "/map">Map</Link></li>
                        <li><Link to = "/chat">Chat</Link></li>
                        <li><Link to = "#">FAQ</Link></li>
                    </ul>
                </div>
            </div>
        </nav>
    )
}
```

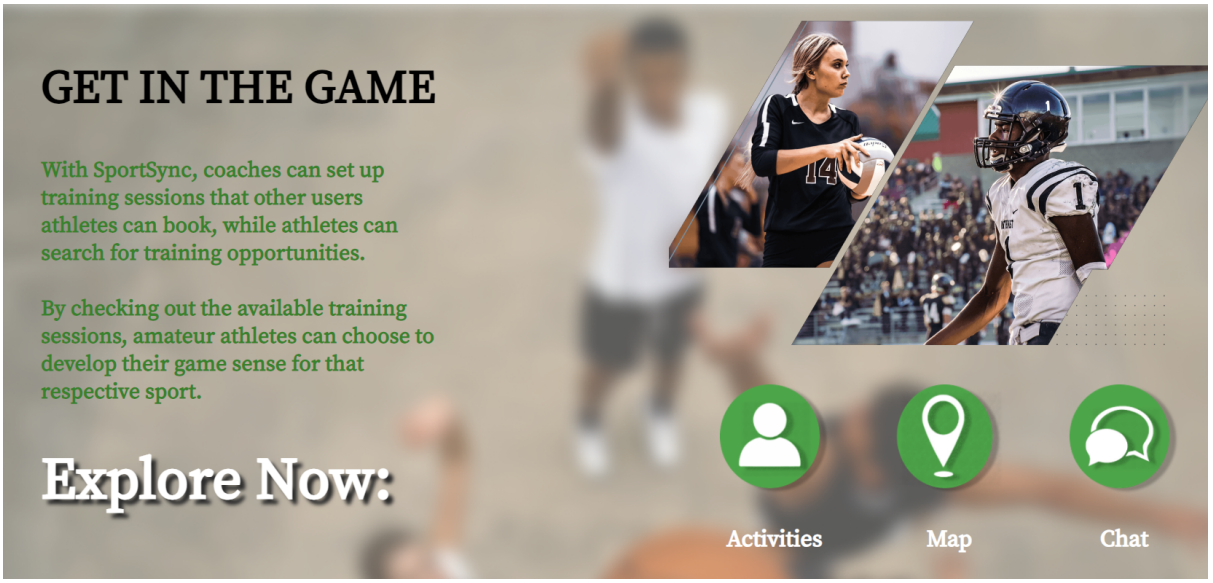
The Navbar component brings the user to different pages of the website. It is able to bring the user to the homepage (3..1.3), profile page (3..1.4), logout of their account and have a dropdown menu which brings them to the marketplace (3.1.1.13), map page (3.1.1.14) and chat page (3.1.1.16). This navbar component will be included in many different pages for ease of use for the user. This navbar component will also alert the user whenever another user books his listing.

The above image is how the navbar component looks like in the final implementation of the website.

3.1.1.3.2 Homepage Content Component

```jsx
export default function HomepageContent() {
    return (
        <div className = "main">
            <div className = "main-left">
                <h2 className = "main-title">GET IN THE GAME</h2>
                <h3 className = "main-description">
                    With SportSync, coaches can set up
                    training sessions that other users
                    athletes can book, while athletes can
                    search for training opportunities.<br /><br />

                    By checking out the available training
                    sessions, amateur athletes can choose
                    to develop their game sense for that
                    respective sport.
                </h3>
                <h1 className = "main-explore">Explore Now:</h1>
            </div>
            <div className = "main-right">
                <div className = "main-right-top">
                    <img src = {SPORTS} width = "751px" height = "445px" />
                </div>
                <div className = "main-right-bottom">
                    <div className = "main-right-bottom-coach">
                        <Link to = "/marketplace">
                            <img src = {COACHLOGO} width = "150px" height = "150px" />
                        </Link>
                        <p>Activities</p>
                    </div>
                    <div className = "main-right-bottom-map">
                        <Link to = "/map">
                            <img src = {MAPLOGO} width = "150px" height = "150px" />
                        </Link>
                        <p>Map</p>
                    </div>
                    <div className = "main-right-bottom-chat">
                        <Link to = "/chat">
                            <img src = {CHATLOGO} width = "150px" height = "150px" />
                        </Link>
                        <p>Chat</p>
                    </div>
                </div>
            </div>
        </div>
```

Homepage Content Component tells the user a brief description of what SportSync is. It is also able to bring the user to the marketplace (3.1.1.13), map page (3.1.1.14) and chat page (3.1.1.16).

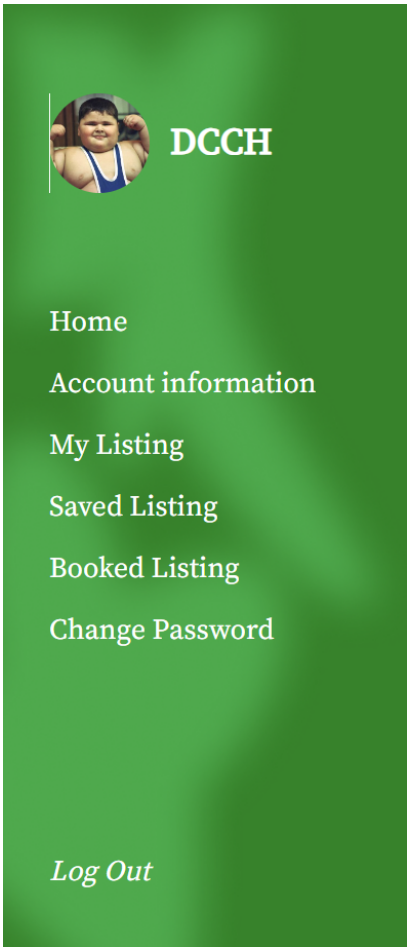


The above image is how the homepageContent component looks like in the final implementation of the website.

3.1.1.4 Sidebar Menu Component

```
export default function Sidebar(props) {
    const userImgSrc = props.item.userImg !== "" ? `/images/${props.item.userImg}` : "/images/human-logo.png";

    return (
        <div className = "sidebar-background">
            <div className = "sidebar-user-info">
                <img className = "sidebar-user-info-pic" src= {userImgSrc} alt="Logo" />
                <h1 className = "sidebar-user-info-username">{props.item.username}</h1>
            </div>
            <div className = "sidebar-listing">
                <Link className = "sidebar-home" to = "/homepage">Home</Link>
                <Link className = "sidebar-account" to = "/profile">Account information</Link>
                <Link className = "sidebar-mylisting" to = "/mylisting">My Listing</Link>
                <Link className = "sidebar-savedlisting" to = "/savedlisting">Saved Listing</Link>
                <Link className = "sidebar-bookedlisting" to = "/bookedlisting">Booked Listing</Link>
                <Link className = "sidebar-changepw" to = "/change-password">Change Password</Link>
            </div>
            <div className = "sidebar-logout-button">
                <Link className = "sidebar-logout" to = "/"><i>Log Out</i></Link>
            </div>
        </div>
    )
}
```

The sidebar menu component is a general menu that brings the user to different pages, which includes homepage (3..1.3), profile (3.1.1.5), user listing (3.1.1.7), user saved listing (3.1.1.10), user booked listing (3.1.1.11), change password (3.1.1.12) and a logout button. It will also accept information that is passed into it to display the user's profile picture and username. This sidebar menu will be included in many of the pages for ease of use for the user.



The above image is how the sidebar component looks like in the final implementation of the website.

3.1.1.5 Profile

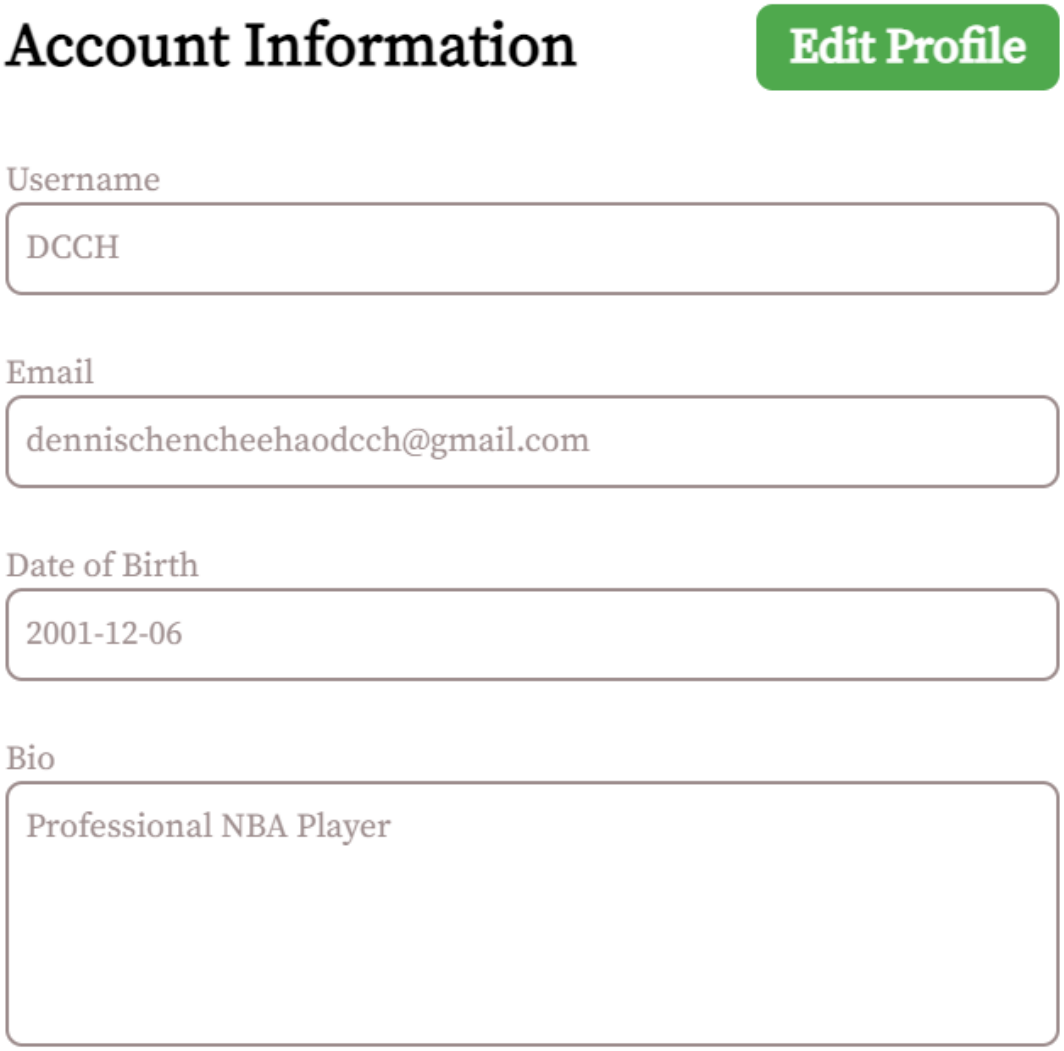
```
export default function Profile() {
    const [userData, setUserData] = useState([]);

    useEffect(() => {
        axios.get("/user")
        .then((response) => {
            setUserData(response.data);
        })
        .catch((error) => {
            console.error("Error fetching user data:", error);
        });
    }, []);

    return (
        <div className = "profile-container">
            <Sidebar
                key = {`sidebar_${userData.id}`}
                item = {userData}
            />
            <AccountInfo
                key = {`account_info_${userData.id}`}
                item = {userData}
            />
        </div>
    )
}
```

Profile page will call upon our user control class (3.1.2.1) to get the user information. It will pass this information to the sidebar menu component (3.1.1.4) and account information component (3.1.1.5.1) to be displayed.

3.1.1.5.1 Account Information Component

```
export default function AccountInfo(props) {
    return (
        <div className = "accountinfo">
            <div className = "accountinfo-head">
                <h1 className = "accountinfo-title">Account Information</h1>
                <button className = "accountinfo-editprofile-button">
                    <Link className = "accountinfo-editprofile-link" to = "/edit-profile">Edit Profile</Link>
                </button>
            </div>
            <div className = "accountinfo-username">
                <span>Username</span>
                <span className = "accountinfo-user-username">{props.item.username}</span>
            </div>
            <div className = "accountinfo-email">
                <span>Email</span>
                <span className = "accountinfo-user-email">{props.item.email}</span>
            </div>
            <div className = "accountinfo-dob">
                <span>Date of Birth</span>
                <span className = "accountinfo-user-dob">{props.item.dob}</span>
            </div>
            <div className = "accountinfo-bio">
                <span>Bio</span>
                <span className = "accountinfo-user-bio">{props.item.bio}</span>
            </div>
        </div>
    )
}
```

The account information component will display the user's information. It will also contain a button which brings the user to the edit profile page (3.1.1.6).



The above image is how the accountInfo component looks like in the final implementation of the website.

3.1.1.6 Profile - Edit Profile

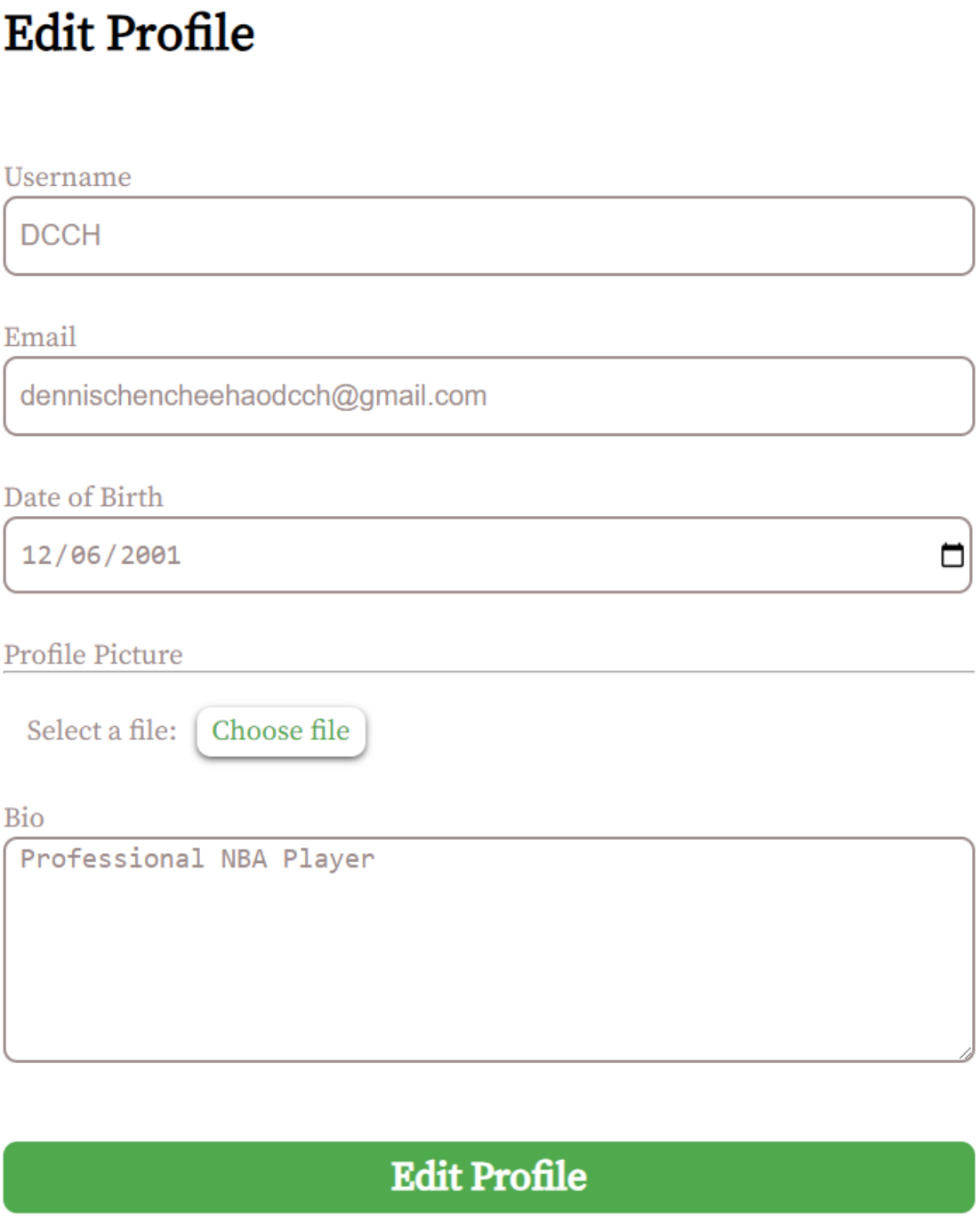
```
export default function EditProfile() {
    const [userData, setUserData] = useState([]);

    useEffect(() => {
        axios.get("/user")
        .then((response) => {
            setUserData(response.data);
        })
        .catch((error) => {
            console.error("Error fetching user data:", error);
        });
    }, []);

    return (
        <div className = "editprofile-container">
            <Sidebar
                key = {`sidebar_${userData.id}`}
                item = {userData}
            />
            <div className = "editprofile-form-container">
                <ProfileInfo
                    key = {`profile_info_${userData.id}`}
                    item = {userData}
                />
            </div>
        </div>
    )
}
```

The edit profile page will call upon our user control class (3.1.2.1) to get the user's information. It will pass this information to the sidebar menu component (3.1.1.4) and profile information component (3.1.1.6.1) to be displayed.

3.1.1.6.1 Profile Info Component

```jsx
export default function ProfileInfo(props) {
    const data = useActionData()

    return (
        <div>
            <h1 className = "profileinfo-title">Edit Profile</h1>
            <Form className = "profileinfo-form" method = "post" action = "/edit-profile">
                <div className = "profileinfo-form-username">
                    <label className = "profileinfo-form-username-label">Username</label>
                    <input className = "profileinfo-form-username-input" type = "text" name = "username" defaultValue = {props.item.username} />
                </div>
                <div className = "profileinfo-form-email">
                    <label className = "profileinfo-form-email-label">Email</label>
                    <input className = "profileinfo-form-email-input" type = "email" name = "email" defaultValue = {props.item.email} />
                </div>
                <div className = "profileinfo-form-dob">
                    <label className = "profileinfo-form-dob-label">Date of Birth</label>
                    <input className = "profileinfo-form-dob-input" type = "date" name = "dob" defaultValue = {props.item.dob} />
                </div>
                <p className = "profileinfo-form-text">Profile Picture</p>
                <hr/>
                <div className = "profileinfo-form-file">
                    <span className = "profileinfo-form-file-text">Select a file:</span>
                    <label className = "profileinfo-form-file-label">
                        <span>Choose file</span>
                        <input type = "file" name = "file" accept = "image/*" />
                    </label>
                </div>
                <div className = "profileinfo-form-bio">
                    <label className = "profileinfo-form-bio-label">Bio</label>
                    <textarea className = "profileinfo-form-bio-input" type = "text" name = "bio" defaultValue = {props.item.bio} />
                </div>
                <button className = "profileinfo-form-button" type = "submit">Edit Profile</button>

                {data && data.error && <p className = "profileinfo-error">{data.error}</p>}

            </Form>
        </div>
    )
}
```

The profile information component consists of a form for the user to fill in to edit and add any new information to update their account. It will take in the user's information and put a default value to each of the inputs. It will accept username, email, date of birth, a file and a description of the user's bio. Afterwards, all of this information will be passed to edit profile action (3.1.1.6.2) to be processed



The above image is how the profileInfo component looks like in the final implementation of the website.

3.1.1.6.2 Edit Profile Action

```
export const editProfileAction = async ({ request }) => {
    const data = await request.formData()

    const submission = {
        email: data.get("email"),
        username: data.get("username"),
        dob: data.get("dob"),
        userImg: data.get("file"),
        bio: data.get("bio")
    }

    console.log(submission)

    try {
        const response = await axios.put("/user", submission);

        if(response.status === 201) {
            return redirect("/profile")
        } else {
            return { error: "Profile update failed"}
        }
    } catch (error) {
        console.log(error)
        const errorMessage = error.response.data.message
        return { error: errorMessage };
    }
}
```

Edit profile action takes in the form information that was submitted and submit it to the user control class (3.1.2.1). If editing of profile is successful, it will return back and redirect the user back to the profile page (3.1.1.5). At the profile page, it will show the updated information that the user has submitted. If editing of profile is unsuccessful, it will pass back the error message for profile info component to display.

3.1.1.7 User Listing

```
export default function MyListing() {
    const [userData, setUserData] = useState([]);
    const [userListing, setUserListing] = useState([]);
    const [message, setMessage] = useState(null);

    useEffect(() => {
        axios.get("/user")
        .then((response) => {
            setUserData(response.data);
        })
        .catch((error) => {
            console.error("Error fetching user data:", error);
        });
    }, []);

    const fetchUserListing = () => {
        axios.get("/user/listings")
        .then((response) => {
            if(Array.isArray(response.data)) {
                setUserListing(response.data);
            } else {
                setUserListing([])
                setMessage(response.data.message);
            }
        })
        .catch((error) => {
            console.error("Error fetching user data:", error);
        });
    }

    useEffect(() => {
        fetchUserListing()
    }, []);

    let content

    if(Array.isArray(userListing) && userListing.length > 0) {
        content = userListing.map((item) => {
            return (
                <Card2
                    key = {item.id}
                    item = {item}
                    fetchUserListing = {fetchUserListing}
                />
            )
        })
    } else {
        content = <p>You have not created any listing</p>;
    }


    return (
        <div className = "mylisting-container">
            <Sidebar
                key = {`sidebar_${userData.id}`}
                item = {userData}
            />
            <div className = "mylisting">
                <h1 className = "mylisting-title">My Listing</h1>
                <section className = "mylisting-list">
                    {content}
                </section>
                <button className = "mylisting-newlist-button">
                    <Link className = "mylisting-newlist-link" to = "/create">Create New Listing</Link>
                </button>
            </div>
        </div>

    )
}
```

The user listing page will call upon our user control class (3.1.2.1) to get the user's information and the user listing's information. It will pass this information to the sidebar menu component (3.1.1.4) and card 2 component (3.1.1.7.1) to be displayed. It will also contain a button which will bring the user to a create listing page (3.1.1.8).

3.1.1.7.1 Card 2 Component

```
export default function Card2(props) {
    if (props.item.haveNotification == true) {
        var notificationMessage = "NEW!"
    }

    const handleDelete = async () => {
        try {
            //Choosing which endpoint
            const deleteURL = `/services/${props.item.id}`

            console.log(deleteURL)
            // Send the DELETE request
            await axios.delete(deleteURL);
            console.log(`Item with id ${props.item.id} deleted successfully`);
            props.fetchUserListing()
        } catch (error) {
            console.error('Error deleting item:', error);
        }
    }

    return (
        <div className = "card2">
            <h1 className = "card2-title">{props.item.sport}<span>{notificationMessage}</span></h1>
            <button className = "card2-select-button">
                <Link className = "card2-select-link" to = {`/listing/${props.item.id}`}>Select</Link>
            </button>
            <button className = "card2-edit-button">
                <Link className = "card2-edit-link" to = {`/edit-listing/${props.item.id}`}>Edit</Link>
            </button>
            <button className = "card2-delete-button" onClick = {handleDelete}>Delete</button>
        </div>
    )
}
```

The card 2 component takes in the user listing's information. It will also contain 3 buttons. Furthermore, it will also notify the user which specific listing has been booked by another user.

Delete button, this will send a request to our coaching service control class (3.1.2.2) to tell them that our user wants to delete this listing from the database. If this request is successful, it will use the parent component (User Listing (3.1.1.7)) method to update the information displayed.

Edit button, this will bring the user to the edit listing page (3.1.1.9) of this specific listing.

Select button, this will bring the user to the specific listing page (3.1.1.15).

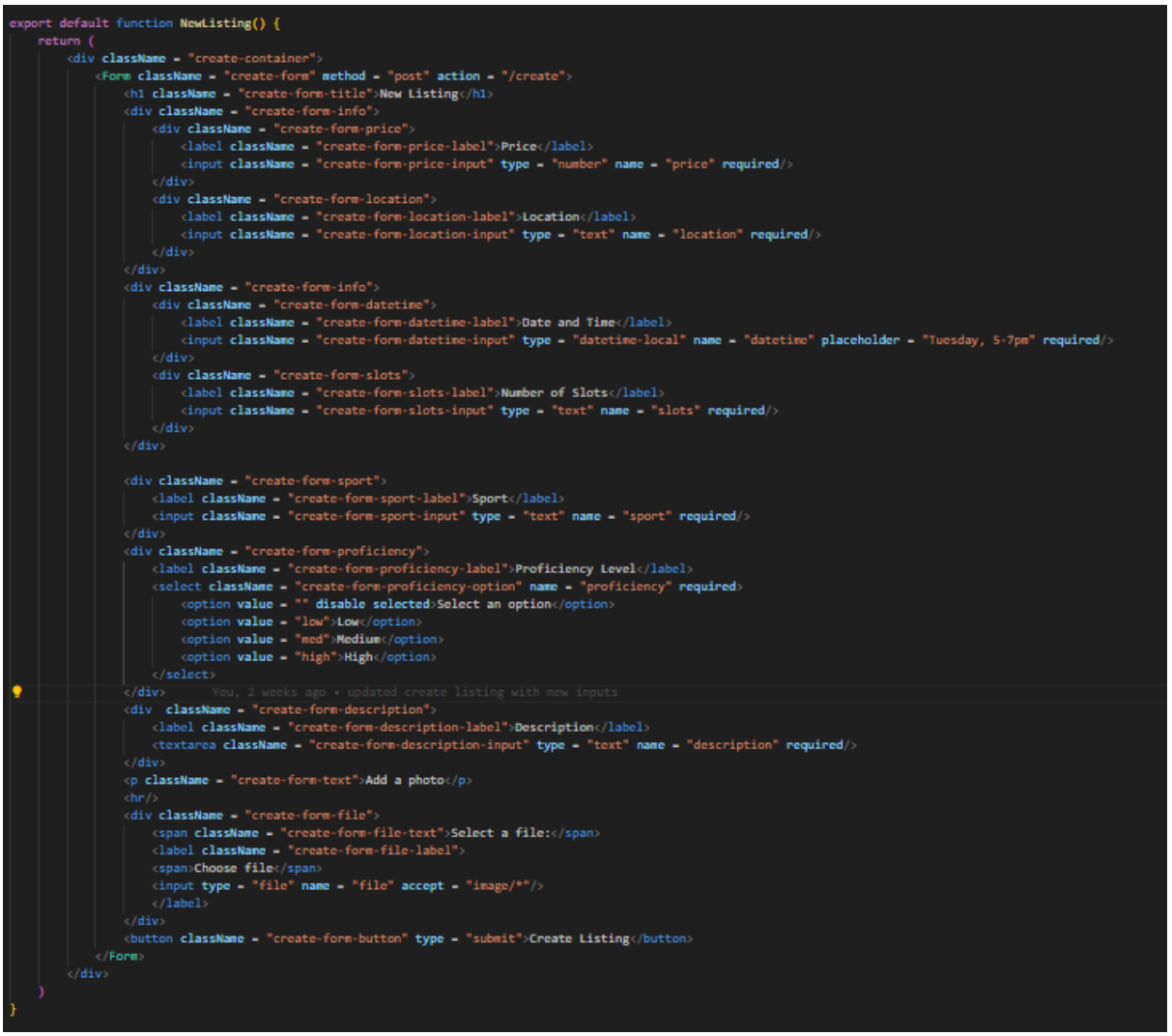
The above image is how the card2 component looks like in the final implementation of the website.

3.1.1.8 Create Listing

```
export default function CreateListing() {
    const [userData, setUserData] = useState([]);

    useEffect(() => {
        axios.get("/user")
        .then((response) => {
            setUserData(response.data);
        })
        .catch((error) => {
            console.error("Error fetching user data:", error);
        });
    }, []);

    return (
        <div className = "create-container">
            <Sidebar
                key = {`sidebar_${userData.id}`}
                item = {userData}
            />
            <div className = "create-form-container">
                <NewListing />
            </div>
        </div>

    )
}
```

The create listing page will call upon our user control class (3.1.2.1) to get the user information.It will pass this information to the sidebar menu component (3.1.1.4) to be displayed. It will also contain the new listing component (3.1.1.8.1).

3.1.1.8.1 New Listing Component



```
export default function NewListing() {
    return (
        <div className = "create-container">
            <Form className = "create-form" method = "post" action = "/create">
                <h1 className = "create-form-title">New Listing</h1>
                <div className = "create-form-info">
                    <div className = "create-form-price">
                        <label className = "create-form-price-label">Price</label>
                        <input className = "create-form-price-input" type = "number" name = "price" required/>
                    </div>
                    <div className = "create-form-location">
                        <label className = "create-form-location-label">Location</label>
                        <input className = "create-form-location-input" type = "text" name = "location" required/>
                    </div>
                </div>
                <div className = "create-form-info">
                    <div className = "create-form-datetime">
                        <label className = "create-form-datetime-label">Date and Time</label>
                        <input className = "create-form-datetime-input" type = "datetime-local" name = "datetime" placeholder = "Tuesday, 5-7pm" required/>
                    </div>
                    <div className = "create-form-slots">
                        <label className = "create-form-slots-label">Number of Slots</label>
                        <input className = "create-form-slots-input" type = "text" name = "slots" required/>
                    </div>
                </div>

                <div className = "create-form-sport">
                    <label className = "create-form-sport-label">Sport</label>
                    <input className = "create-form-sport-input" type = "text" name = "sport" required/>
                </div>
                <div className = "create-form-proficiency">
                    <label className = "create-form-proficiency-label">Proficiency Level</label>
                    <select className = "create-form-proficiency-option" name = "proficiency" required>
                        <option value = "" disable selected>Select an option</option>
                        <option value = "low">Low</option>
                        <option value = "med">Medium</option>
                        <option value = "high">High</option>
                    </select>
                </div>       You, 2 weeks ago • updated create listing with new inputs
                <div  className = "create-form-description">
                    <label className = "create-form-description-label">Description</label>
                    <textarea className = "create-form-description-input" type = "text" name = "description" required/>
                </div>
                <p className = "create-form-text">Add a photo</p>
                <hr/>
                <div className = "create-form-file">
                    <span className = "create-form-file-text">Select a file:</span>
                    <label className = "create-form-file-label">
                    <span>Choose file</span>
                    <input type = "file" name = "file" accept = "image/*"/>
                    </label>
                </div>
                <button className = "create-form-button" type = "submit">Create Listing</button>
            </Form>
        </div>
    )
}
```

The new listing component will consist of a form for the user to fill in order for him to create a new listing. This form will accept the price, location, date and time, number of slots, the type of sport, proficiency level, description and file which displays the image of the sport. This information will be passed to a create listing action (3.1.1.8.2) to be processed.

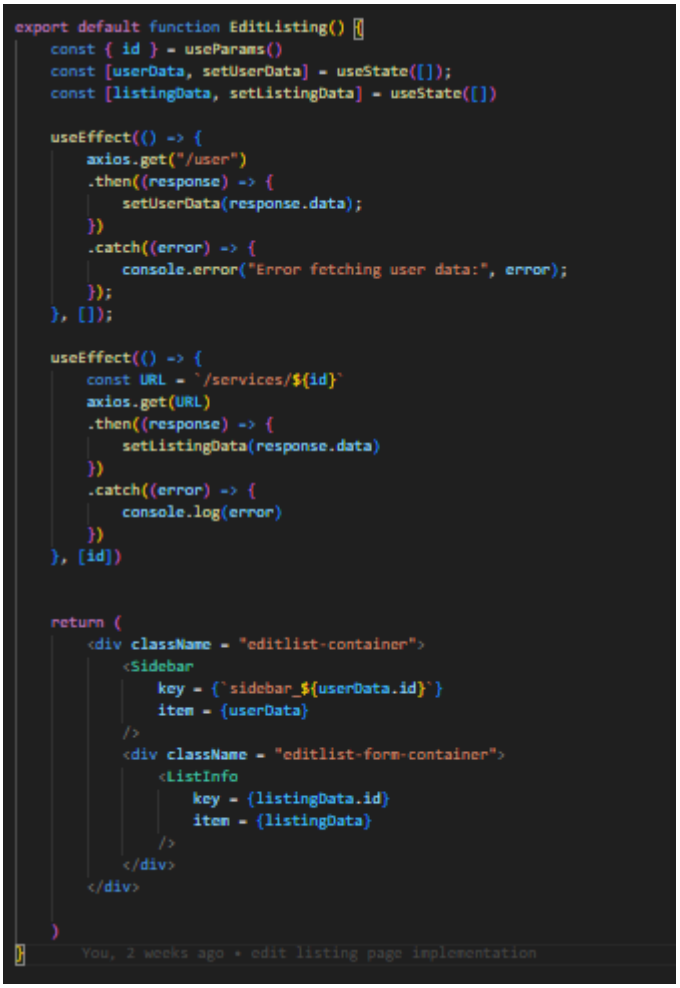
The above image is how the newListing component looks like in the final implementation of the website.

3.1.1.8.2 Create Listing Action

```
const CREATE_SERVICE_URL = '/coaching_services'
export const createListingAction = async ({ request }) => {
    const data = await request.formData()

    const submission = {
        price: parseFloat(data.get("price")),
        location: data.get("location"),
        datetime: data.get("datetime"),
        sport: data.get("sport"),
        proficiency: data.get("proficiency"),
        description: data.get("description"),
        coverImg: data.get("file"),
        maximum: parseInt(data.get("slots"))
    }

    //submit new listing to backend
    console.log(submission)
    try {
        const response = await axios.post(CREATE_SERVICE_URL, submission);

        if(response.status === 201) {
            return redirect("/mylisting")
        } else {
            // Handle any problems
            return { error: "Creation of service failed"}
        }
    } catch (error) {
        console.log(error.response.data.message)
        const errorMessage = error.response.data.message
        return { error: errorMessage}
    }
}       You, 2 weeks ago • added create listing
```

Create Listing action takes in the form information that was submitted and passes it to the coaching service control class (3.1.2.2). If creation of the listing is successful, it will return back and redirect the user to his listing page. If registration is unsuccessful, it will pass back the error message to display.

3.1.1.9 Edit Listing



```
export default function EditListing() {
    const { id } = useParams()
    const [userData, setUserData] = useState([]);
    const [listingData, setListingData] = useState([])

    useEffect(() => {
        axios.get("/user")
        .then((response) => {
            setUserData(response.data);
        })
        .catch((error) => {
            console.error("Error fetching user data:", error);
        });
    }, []);

    useEffect(() => {
        const URL = `/services/${id}`
        axios.get(URL)
        .then((response) => {
            setListingData(response.data)
        })
        .catch((error) => {
            console.log(error)
        })
    }, [id])


    return (
        <div className = "editlist-container">
            <Sidebar
                key = {`sidebar_${userData.id}`}
                item = {userData}
            />
            <div className = "editlist-form-container">
                <ListInfo
                    key = {listingData.id}
                    item = {listingData}
                />
            </div>
        </div>

    )
}
      You, 2 weeks ago • edit listing page implementation
```

The edit listing page will call upon our user control class (3.1.2.1) to get the user information and the coaching service control class (3.1.2.2) to get the specific listing information. It will pass this information to the sidebar menu component (3.1.1.4) and the list info component (3.1.1.9.1) to be displayed.

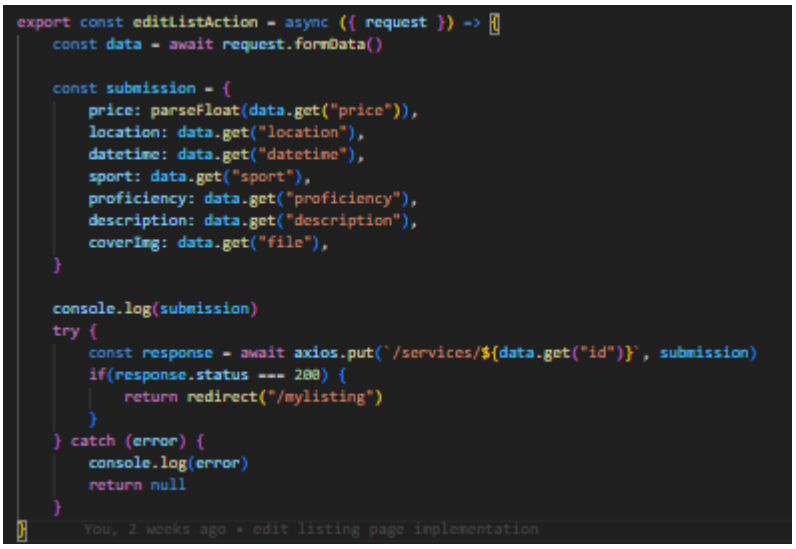
3.1.1.9.1 List Info Component

```jsx
export default function ListInfo(props) {
    const { id } = useParams();
    const editActionURL = `/edit-listing/${id}`;
    return (
        <div className = "listinfo-container">
            <Form className = "listinfo-form" method = "post" action = {editActionURL}>
                <h1 className = "listinfo-form-title">Edit Listing</h1>
                <input type="hidden" name="id" value={props.item.id} />
                <div className = "listinfo-form-info">
                    <div className = "listinfo-form-price">
                        <label className = "listinfo-form-price-label">Price</label>
                        <input className = "listinfo-form-price-input" type = "number" name = "price" defaultValue = {props.item.price}/>
                    </div>
                    <div className = "listinfo-form-location">
                        <label className = "listinfo-form-location-label">Location</label>
                        <input className = "listinfo-form-location-input" type = "text" name = "location" defaultValue = {props.item.location}/>
                    </div>
                </div>
                <div className = "listinfo-form-datetime">
                    <label className = "listinfo-form-datetime-label">Date and Time</label>
                    <input className = "listinfo-form-datetime-input" type = "datetime-local" name = "datetime" defaultValue = {props.item.datetime}/>
                </div>
                <div className = "listinfo-form-sport">
                    <label className = "listinfo-form-sport-label">Sport</label>
                    <input className = "listinfo-form-sport-input" type = "text" name = "sport" defaultValue = {props.item.sport}/>
                </div>
                <div className = "listinfo-form-proficiency">
                    <label className = "listinfo-form-proficiency-label">Proficiency Level</label>
                    <select className = "listinfo-form-proficiency-option" name = "proficiency" defaultValue={props.item.proficiency}>
                        <option value = "Low">Low</option>
                        <option value = "Med">Medium</option>
                        <option value = "High">High</option>
                    </select>
                </div>
                <div  className = "listinfo-form-description">
                    <label className = "listinfo-form-description-label">Description</label>
                    <textarea className = "listinfo-form-description-input" type = "text" name = "description" defaultValue = {props.item.description}/>
                </div>
                <p className = "listinfo-form-text">Add a photo</p>
                <hr/>
                <div className = "listinfo-form-file">
                    <span className = "listinfo-form-file-text">Select a file:</span>
                    <label className = "listinfo-form-file-label">
                    <span>Choose file</span>
                    <input type = "file" name = "file" accept = "image/*"/>
                    </label>
                </div>
                <button className = "listinfo-form-button" type = "submit">Edit Listing</button>
            </Form>
        </div>
    )
}
      You, 2 weeks ago • edit listing page implementation
```

The list info component consists of a form for the user to fill in to edit and add any new information to update their listing. It will take in the user listing's information and put a default value to each of the inputs. It will accept price, location, date and time, sport, proficiency level, description, and a file of the sport. Afterwards, all of this information will be passed to edit listing action (3.1.1.9.2) to be processed.

The above image is how the listInfo component looks like in the final implementation of the website.

3.1.1.9.2 Edit Listing Action



```
export const editListAction = async ({ request }) => {
    const data = await request.formData()

    const submission = {
        price: parseFloat(data.get("price")),
        location: data.get("location"),
        datetime: data.get("datetime"),
        sport: data.get("sport"),
        proficiency: data.get("proficiency"),
        description: data.get("description"),
        coverImg: data.get("file"),
    }

    console.log(submission)
    try {
        const response = await axios.put(`/services/${data.get("id")}`, submission)
        if(response.status === 200) {
            return redirect("/mylisting")
        }
    } catch (error) {
        console.log(error)
        return null
    }
}       You, 2 weeks ago • edit listing page implementation
```

Edit listing action takes in the form information that was submitted and submit it to the coaching service control class (3.1.2.2). If editing of the listing is successful, it will return back and redirect the user back to his listing page (3.1.1.5). At his listing page, it will show the updated information that the user has submitted.

3.1.1.10 User Saved Listing

```
export default function SavedListing() {
    const [userData, setUserData] = useState([]);
    const [userSaved, setUserSaved] = useState([]);
    const [message, setMessage] = useState(null);

    useEffect(() => {
        axios.get("/user")
        .then((response) => {
            setUserData(response.data);
        })
        .catch((error) => {
            console.error("Error fetching user data:", error);
        });
    }, []);

    const userSavedListing = () => {
        axios.get("/user/saved")
        .then((response) => {
            if(Array.isArray(response.data)) {
                setUserSaved(response.data);
            } else {
                setUserSaved([])
                setMessage(response.data.message);
            }
        })
        .catch((error) => {
            console.error("Error fetching user data:", error);
        });
    }

    useEffect(() => {
        userSavedListing()
    }, []);

    let content

    if(Array.isArray(userSaved) && userSaved.length > 0) {
        content = userSaved.map((item) => {
            return (
                <Card3
                    key = {item.id}
                    item = {item}
                    fetchUserListing = {userSavedListing}
                />
            )
        })
    } else {
        content = <p>{message}</p>;
    }

    return (
        <div className = "savedlisting-container">
            <Sidebar
                key = {`sidebar_${userData.id}`}
                item = {userData}
            />
            <div className = "savedlisting">
                <h1 className = "savedlisting-title">Saved Listing</h1>
                <section className = "savedlisting-list">
                    {content}
                </section>
            </div>
        </div>

    )
}
```

The user saved listing page will call upon our user control class (3.1.2.1) to get the user information and the user saved listing information. It will pass this information to the sidebar menu component (3.1.1.4) and card 3 component (3.1.1.10.1) to be displayed.

3.1.1.11 User Booked Listing

```
export default function BookedListing() {
    const [userData, setUserData] = useState([]);
    const [userBooked, setUserBooked] = useState([]);
    const [message, setMessage] = useState(null);

    useEffect(() => {
        axios.get("/user")
        .then((response) => {
            setUserData(response.data);
        })
        .catch((error) => {
            console.error("Error fetching user data:", error);
        });
    }, []);

    const fetchUserBooked = () => {
        axios.get("/user/booked")
        .then((response) => {
            if(Array.isArray(response.data)) {
                setUserBooked(response.data);
            } else {
                setUserBooked([])
                setMessage(response.data.message);
            }
        })
        .catch((error) => {
            console.error("Error fetching user data:", error);
        });
    }

    useEffect(() => {
        fetchUserBooked()
    }, []);

    let content

    if(Array.isArray(userBooked) && userBooked.length > 0) {
        content = userBooked.map((item) => {
            return (
                <Card3
                    key = {item.id}
                    item = {item}
                    fetchUserListing = {fetchUserBooked}
                />
            )
        })
    } else {
        content = <p>{message}</p>;
    }

    return (
        <div className = "bookedlisting-container">
            <Sidebar
                key = {`sidebar_${userData.id}`}
                item = {userData}
            />
            <div className = "bookedlisting">
                <h1 className = "bookedlisting-title">Booked Listing</h1>
                <section className = "bookedlisting-list">
                    {content}
                </section>
            </div>
        </div>

    )
}
    You, 2 weeks ago • added more pages and linking them ...
```

The user booked listing page will call upon our user control class (3.1.2.1) to get the user information and the user booked listing information. It will pass this information to the sidebar menu component (3.1.1.4) and card 3 component (3.1.1.11.1) to be displayed.

3.1.1.10.1 / 3.1.1.11.1 Card 3 Component

```jsx
export default function Card3(props) {
    const location = useLocation();
    const currentURL = location.pathname;

    const handleDelete = async () => {
        try {
            //Choosing which endpoint
            const deleteURL = currentURL === '/bookedlisting' ? `/book/${props.item.id}` : `/save/${props.item.id}`;

            console.log(deleteURL)
            // Send the DELETE request
            await axios.delete(deleteURL);
            props.fetchUserListing()
            console.log(`Item with id ${props.item.id} deleted successfully`);
        } catch (error) {
            console.error('Error deleting item:', error);
        }
      }

    return (
        <div className = "card3">
            <h1 className = "card3-title">{props.item.sport}</h1>
            <button className = "card3-select-button">
                <Link className = "card3-select-link" to = {`/listing/${props.item.id}`}>Select</Link>
            </button>
            <button className = "card3-delete-button" onClick={handleDelete}>Delete</button>
        </div>
    )
}
    You, 2 weeks ago • implement new listing component, implemented into…
```

The card 3 component takes in the user saved / booked listing's information. It will also contain 2 buttons.

Delete button, this will send a request to our coaching service control class (3.1.2.2) to tell them that our user wants to delete this saved / booked listing from the database. If this request is successful, it will use the parent component method to update the information displayed.

Select button, this will bring the user to the listing page (3.1.1.15) of this specific listing.

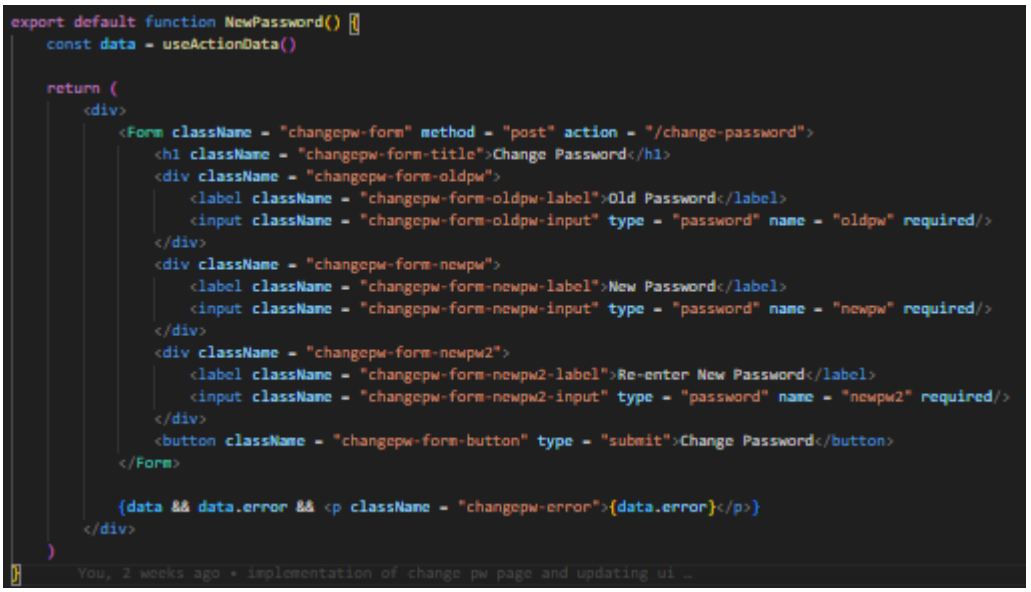
The above image is how the card3 component looks like in the final implementation of the website.

3.1.1.12 Change Password

```
export default function ChangePassword () {
    const [userData, setUserData] = useState([]);

    useEffect(() => {
        axios.get("/user")
        .then((response) => {
            setUserData(response.data);
        })
        .catch((error) => {
            console.error("Error fetching user data:", error);
        });
    }, []);

    return (
        <div className = "password-container">
            <Sidebar
                key = {`sidebar_${userData.id}`}
                item = {userData}
            />
            <div className = "password-form-container">
                <NewPassword />
            </div>
        </div>
    )
}       You, 2 weeks ago • implementation of change pw page and updating
```

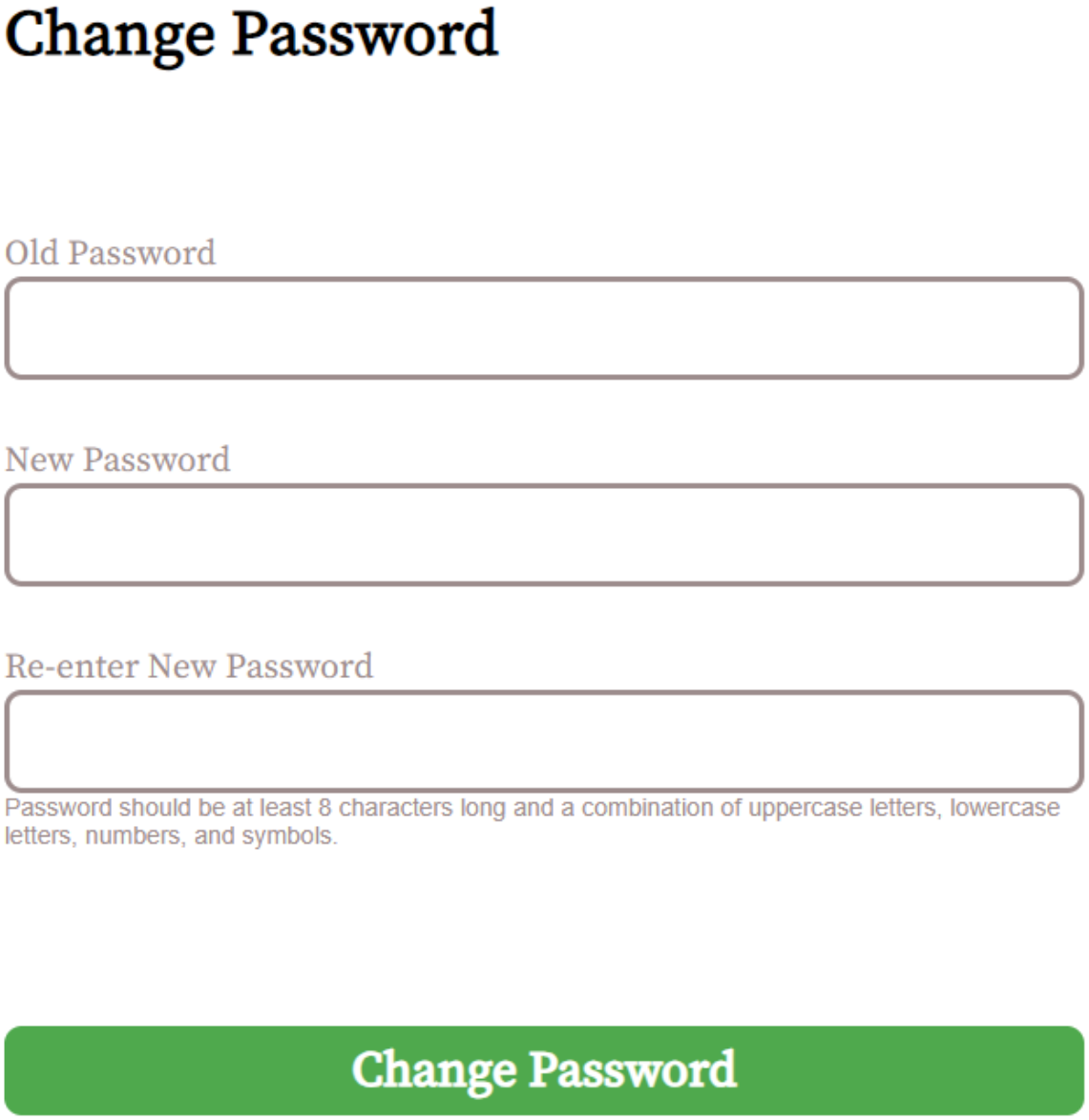
Change password page will call upon our user control class (3.1.2.1) to get the user information. It will pass this information to the sidebar menu component (3.1.1.4) to be displayed. It will also contain the new password component (3.1.1.12.1).

3.1.1.12.1 New Password Component



```
export default function NewPassword() {
    const data = useActionData()

    return (
        <div>
            <Form className = "changepw-form" method = "post" action = "/change-password">
                <h1 className = "changepw-form-title">Change Password</h1>
                <div className = "changepw-form-oldpw">
                    <label className = "changepw-form-oldpw-label">Old Password</label>
                    <input className = "changepw-form-oldpw-input" type = "password" name = "oldpw" required/>
                </div>
                <div className = "changepw-form-newpw">
                    <label className = "changepw-form-newpw-label">New Password</label>
                    <input className = "changepw-form-newpw-input" type = "password" name = "newpw" required/>
                </div>
                <div className = "changepw-form-newpw2">
                    <label className = "changepw-form-newpw2-label">Re-enter New Password</label>
                    <input className = "changepw-form-newpw2-input" type = "password" name = "newpw2" required/>
                </div>
                <button className = "changepw-form-button" type = "submit">Change Password</button>
            </Form>

            {data && data.error && <p className = "changepw-error">{data.error}</p>}
        </div>
    )
}
    You, 2 weeks ago • implementation of change pw page and updating ui …
```

The new password component consists of a form for the user to fill in to edit their existing password. It will require the user's old and new password. Afterwards, all of this information will be passed to change password action (3.1.1.12.2) to be processed.



The above image is how the newPassword component looks like in the final implementation of the website.

3.1.1.12.2 Change Password Action

```
export const changePwAction = async ({ request }) => {
    const data = await request.formData()

    const newpw2 = data.get("newpw2")

    const submission = {
        password: data.get("oldpw"),
        newpw: data.get("newpw"),
    }
    console.log(submission, newpw2)

    // password checker function
    if (submission.newpw != newpw2)
        return {error: "Password is not the same"};
    if (submission.newpw.length < 8)
        return {error: "Password must at least 8 characters long"};
    if (!(/\d/.test(submission.newpw)) )
        return {error: "Password must have at least 1 number"};
    if (!(/[A-Z]/.test(submission.newpw)))
        return {error: "Password must have at least 1 upper case"};
    if (!(/[a-z]/.test(submission.newpw)))
        return {error: "Password must have at least 1 lower case"};
    if (!(/[!@#$%^&*()_+\-=\[\]{};':"\\|,.<>\/?]+/.test(submission.newpw)))
        return {error: "Password must have at least 1 special character"};



    try {
        await axios.put(URL, submission);
        return redirect("/");
    } catch (error) {
        console.log(error)
        const errorMessage = error.response.data.message
        return { error: errorMessage };
    }

}
```

Change password action takes in the form information that was submitted and first checks that the new password that was inputted meets the requirements.
Afterwards, it will submit it to the user control class (3.1.2.1). If the change password is successful, it will return back and redirect the user to the login page. If the change password is unsuccessful, it will pass back the error message to the new password component to display.

3.1.1.13 Marketplace

```
export default function Marketplace() {

    const [coachingServices, setCoachingServices] = useState([]);

    useEffect(() => {
        // Make a GET request to "/coaching_services" using Axios
        axios.get("/coaching_services")
            .then((response) => {
                // Assuming the response data is an array, update the state with the data
                console.log("Received data:", response.data);
                setCoachingServices(response.data);
            })
            .catch((error) => {
                console.error("Error fetching data:", error);
            });
    }, []);

    let content

    if(coachingServices.length != 0) {
        content = coachingServices.map (item => {
            return (
                <Card
                    key = {item.id}
                    item = {item}
                />
            )
        })
    } else {
        content = <p>Currently there is no services</p>
    }



    return (
        <div>
            <Navbar />
            <Searchbar
                updateListing = {setCoachingServices}
            />
            <section className="cards-list">
                {content}
            </section>
        </div>
    )
}
```

The marketplace page will call upon our coaching service control class (3.1.2.2) to get all coaching services. It will pass this information to the card component (3.1.1.13.1) to display. It will also contain navbar (3.1.1.3.1) and search bar (3.1.1.13.2) components.

3.1.1.13.1 Card Component

```
export default function Card(props) {
    const userImgSrc = props.item.coach.userImg !== "" ? `/images/${props.item.coach.userImg}` : "/images/human-logo.png"
    const coverImgSrc = props.item.coverImg !== "" ? `/images/${props.item.coverImg}` : "/images/listing-coverimg.png"
    return (
        <div className="card">
            <div className = "card-coach">
                <img className = "card-coach-img" src = {userImgSrc} alt = "coach-pic" />
                <h1 className = "card-coach-username">{props.item.coach.username}</h1>
            </div>
            <img className = "card-coverImg" src = {coverImgSrc} alt = "sport-pic"/>
            <h1 className = "card-sport">{props.item.sport}</h1>
            <button type="button" className="card-select-button">
                    <Link className = "card-select-button-link" to = {`/listing/${props.item.id}`}>Select Listing</Link>
            </button>
        </div>
    )
}
    You, 2 weeks ago • added card component …
```

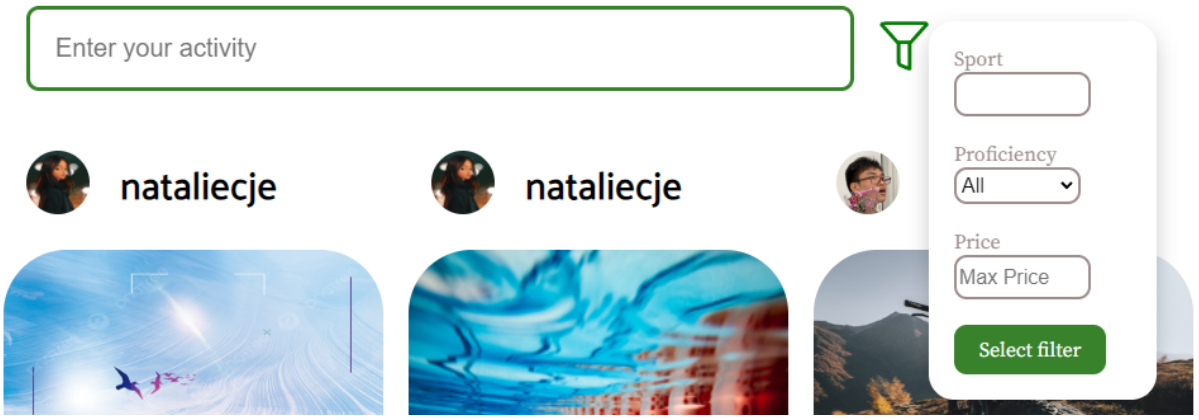
The card component takes in the user listing's information. It will also contain a select button which will bring the user to the listing page (3.1.1.15) of this specific listing.

The above image is how the card component looks like in the final implementation of the website.

3.1.1.13.2 SearchBar Component

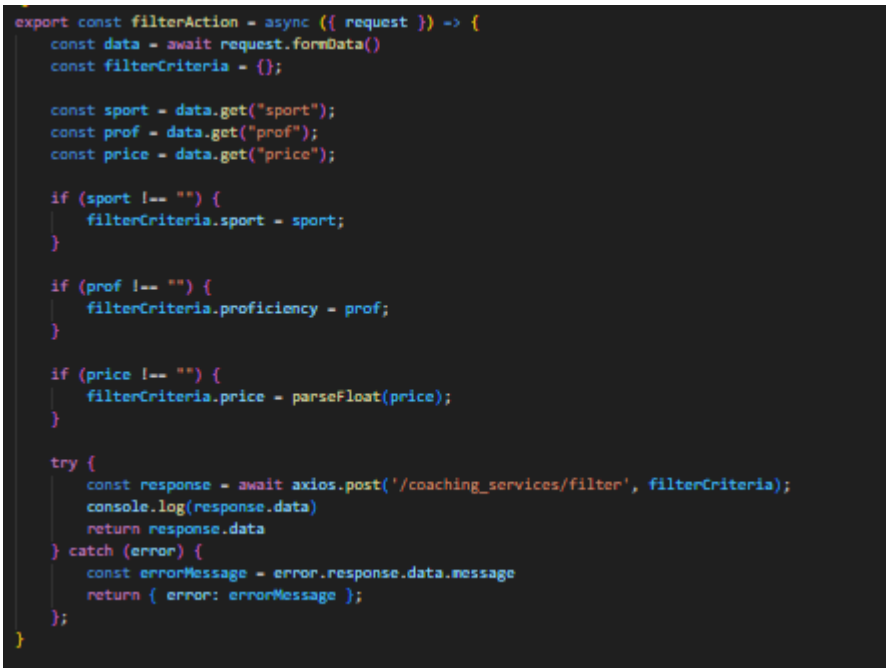
```jsx
export default function Searchbar(props) {
    const data = useActionData()

    useEffect(() => {
        if(data != undefined) {
            props.updateListing(data)
        }
    }, [data])

    return (
        <section>
            <div className="search">
                <input
                    className = "search-box"
                    type="text"
                    placeholder="Enter your activity"
                    /* do functions based on change */
                />
            </div>
            <div className="nav-right">
                <div className="dropdown">
                    <img className = "filter-icon" src = {FILTERLOGO} alt = "filter-logo"/>
                        <Form className = "filter-form" method = "post" action = "/marketplace">
                            <div className = "dropdown-menu-sport">
                                <label className = "dropdown-menu-sport-text">Sport</label>
                                <input className = "dropdown-menu-sport-box" type = "text" name = "sport"></input>
                            </div>
                            <div className = "dropdown-menu-prof">
                                <label className = "dropdown-menu-prof-text">Proficiency</label>
                                <select className = "dropdown-menu-prof-box" name = "prof">
                                    <option value = "">All</option>
                                    <option value = "low">Low</option>
                                    <option value = "med">Medium</option>
                                    <option value = "high">High</option>
                                </select>
                            </div>
                            <div className = "dropdown-menu-price">
                                <label className = "dropdown-menu-price-text">Price</label>
                                <input className = "dropdown-menu-price-box" type = "number" name = "price" placeholder="Max Price"></input>
                            </div>
                            <button className = "filter-button" type = "submit">Select filter</button>
                        </Form>
                </div>
            </div>
        </section>
    )
}
    Chai Jie En, 4 days ago • searchbar + filter trial
```

The search bar component consists of a form for the user to fill in in order to filter the kind of services that would like to see on the marketplace. It will accept sport, proficiency and price. Afterwards, all of this information will be passed to filter action (3.1.1.13.3) to be processed. When the filter goes through, it will return back a new set of services that is available which will be passed back to the marketplace (3.1.1.13).



The above image is how the searchbar component looks like in the final implementation of the website.

3.1.1.13.3 Filter Action



```
export const filterAction = async ({ request }) => {
    const data = await request.formData()
    const filterCriteria = {};

    const sport = data.get("sport");
    const prof = data.get("prof");
    const price = data.get("price");

    if (sport !== "") {
        filterCriteria.sport = sport;
    }

    if (prof !== "") {
        filterCriteria.proficiency = prof;
    }

    if (price !== "") {
        filterCriteria.price = parseFloat(price);
    }

    try {
        const response = await axios.post('/coaching_services/filter', filterCriteria);
        console.log(response.data)
        return response.data
    } catch (error) {
        const errorMessage = error.response.data.message
        return { error: errorMessage };
    };
}
```

Filter action takes in the form information that was submitted and submit it to the coaching service control class (3.1.2.2). If the filter is successful, it will return all the services that fit the criteria and return back to the search bar component (3.1.1.13.2). If the filter is unsuccessful, it will pass back the error message.

3.1.1.14 Map

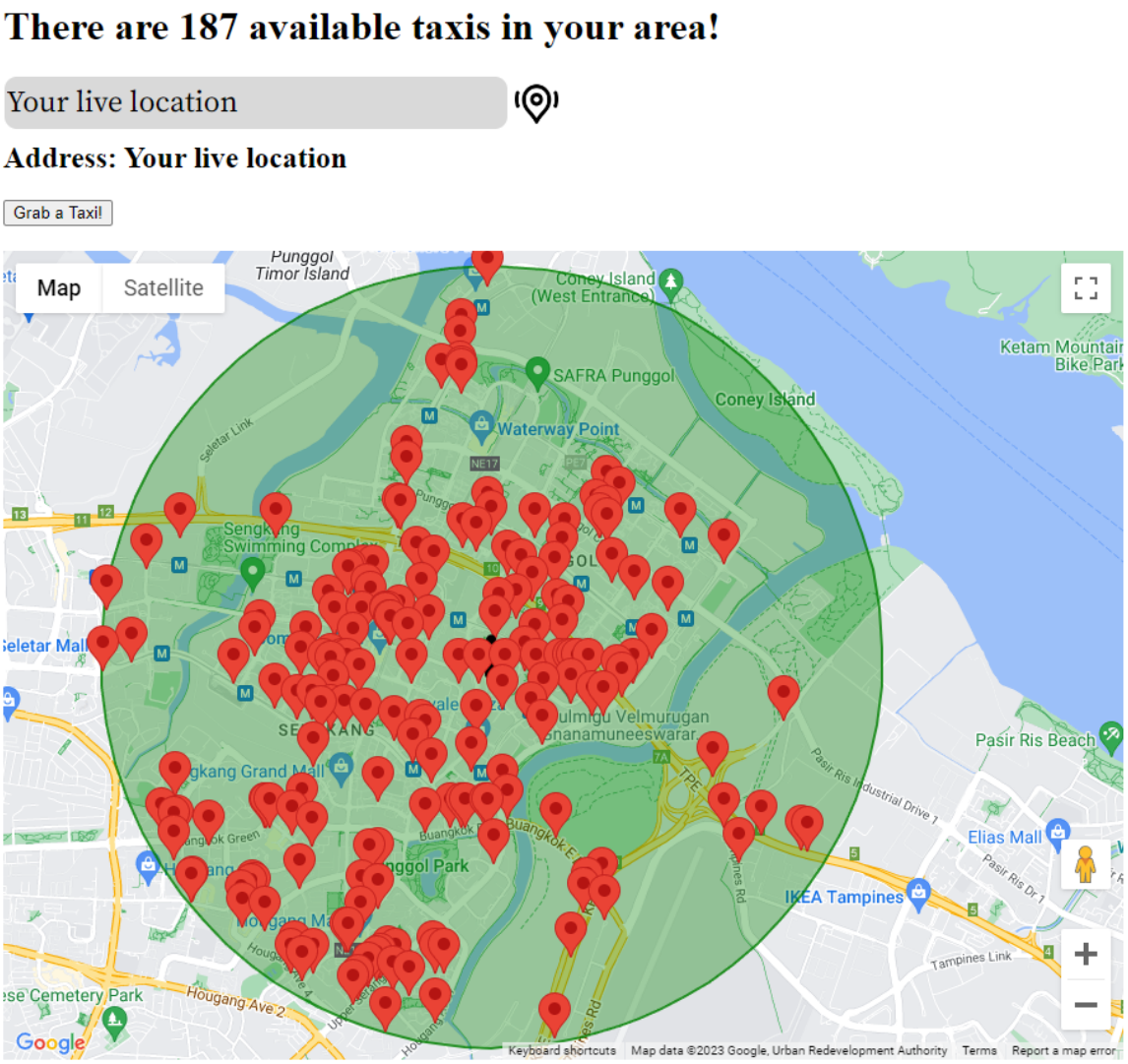
```
return (
    <div className = "map-page-container">
        <MapSidebar
            item = {userData}
        />
        <div className = "map-container">
          <h1>There are {totalTaxis} available taxis in your area!</h1>
          <MapSearchBar
            updateUserCentre={updateUserCentre}
          />
            <GoogleMap
                zoom = {15}
                center = {userCenter}
                mapContainerClassName="map-size"
            >
              {/* render the markers within the circle */}
              {filteredCoordinates &&
                filteredCoordinates.map((coord, index) => (
                  <Marker
                    key={index}
                    position={{ lat: coord[1], lng: coord[0] }}
                  />
                ))
              }
              <Marker icon={myLocation} position={userCenter} />
              <Circle center={userCenter}
                radius={radius}
                options={{
                fillColor: "green",
                fillOpacity: 0.35,
                strokeColor: "green",
                strokeOpacity: 0.8,
                strokeWeight: 2,
              }}
              />
            </GoogleMap>
        </div>
    </div>
)
```

Displays the map and the available taxis in a 3km radius. The map makes API calls to fetch live data to display the taxis as markers on the map that are within the circle shape. The circle shape can be changed in MapSearchBar (3.1.1.14.1).

3.1.1.14.1 MapSearchBar

```
return (
    <div>
    {/* <p>Your location is lat: {currLocation.lat}, lng: {currLocation.lng}</p> */}
        <div className="mapsidebar-input">
            <PlacesAutocomplete
                value={address}
                onChange={handleChange}
                onSelect={handleSelect}
                >
                {({ getInputProps, suggestions, getSuggestionItemProps, loading }) => (
                <div >
                    <div className = "mapsidebar-input-container">
                        <input
                        {...getInputProps({
                            placeholder: " Search Places ",
                            className: "mapsidebar-input-box",
                        })}
                        />
                        <button className= "mapsidebar-input-button" type="button" onClick={getLocation}>
                            <img className= "mapsidebar-input-button" src={MAGNIFYINGLOGO} alt="Search"/>
                        </button>
                    </div>
                    <div className="autocomplete-dropdown-container">
                        {loading && <div>Loading...</div>}
                        {suggestions.map((suggestion) => {
                        const className = suggestion.active
                        ? "suggestion-item--active"
                        : "suggestion-item";
                        const style = suggestion.active
                        ? { backgroundColor: "#fafafa", cursor: "pointer", position: "relative", zIndex: "100"}
                        : { backgroundColor: "grey", cursor: "pointer", position: "relative", zIndex: "100" };
                        return (
                            <div
                            {...getSuggestionItemProps(suggestion, {
                            className,
                            style,
                            })}
                            >
                            <span>{suggestion.description}</span>
                            </div>
                        );
                        })}
                    </div>
                </div>
            )}
            </PlacesAutocomplete>
        </div>
        <h2 className="address-para">Address: {address}</h2>
        <span className = "error">{content}</span>
    </div>
);
```

Allows the user to input their desired address to locate available taxis within a 3km radius. The PlacesAutocomplete component gives suggested places based on what the user inputs. Upon the change of address, the coordinates of that address will be sent to Map (3.1.1.14) and set as the new centre of the circle and the API data will be fetched and displayed as markers.



The above image is how the map page looks like in the final implementation of the website.

3.1.1.15 Listing

```
export default function Listing() {
    const { id } = useParams() //coaching services id

    // Linkage to backend
    const [specificListing, setSpecificListing] = useState(null);

    const fetchListingData = () => {
        axios.get(`/services/${id}`)
        .then((response) => {
            console.log("Received data:", response.data);
            setSpecificListing(response.data);
        })
        .catch((error) => {
            console.error("Error fetching data:", error);
        });
    };

    useEffect(() => {
        fetchListingData()
    }, [])


    return (
        <div>
            <Navbar />
            {specificListing ? (
                <>
                    <div className= "information-container">
                        <ListDisplay
                            key={`display_${specificListing.id}`}
                            item={specificListing}
                            fetchListingData = {fetchListingData}
                        />
                        <ListReview
                            key={`review_${specificListing.id}`}
                            item={specificListing}
                            fetchListingData = {fetchListingData}
                        />
                    </div>
                </>
            ) : (
                <p>Loading or not found...</p>
            )}
        </div>

    )
}
```

Listing page shows all the details of a specific coaching service. It pulls the information from the coaching service control class (3.1.2.2) and pass this information to the list display component (3.1.1.15.1) and list review component (3.1.1.15.2)

3.1.1.15.1 List Display Component

```
return (
    <>
        <div className = "listdisplay-container">
            <div className = "listdisplay-user">
                <div className = "listdisplay-user-container">
                    <img className = "listdisplay-service-picture" src = {coverImgSrc}/>
                    <span>{props.item.coach.username}</span>
                </div>
                <div className = "listdisplay-user-bio">
                    <label className = "listdisplay-user-bio-label">Coach Bio</label>
                    <span className = "listdisplay-user-bio-box">{props.item.coach.bio}</span>
                </div>
            </div>
            <div className = "listdisplay-service-container">
                <div className = "listdisplay-service-info">
                    <div className = "listdisplay-service-info-price">
                        <label>Price</label>
                        <span>{props.item.price}</span>
                    </div>
                    <div className = "listdisplay-service-info-location">
                        <label>Location</label>
                        <span>{props.item.location}</span>
                    </div>
                    <div className = "listdisplay-service-info-time">
                        <label>Time</label>
                        <span>{datetimeWithoutT}</span>
                    </div>
                    <div className = "listdisplay-service-info-sport">
                        <label>Sport</label>
                        <span>{props.item.sport}</span>
                    </div>
                    <div className = "listdisplay-service-info-proficiency">
                        <label>Proficiency</label>
                        <span>{props.item.proficiency}</span>
                    </div>
                    <div className = "listdisplay-service-info-availability">
                        <label>Availability</label>
                        <span>{props.item.available}</span>
                    </div>
                </div>
                <div className = "listdisplay-serivce-description">
                    <label>Description</label>
                    <span>{props.item.description}</span>
                </div>
                <div className = "listdisplay-serivce-buttons">
                    <button onClick = {handleSaving}>Save Listing</button>
                    <button onClick = {handleBooking}>Book Listing</button>
                </div>
            </div>
        </div>
        <span className = "alert">{content}</span>
    </>
)
```

The list display component will take in all the information regarding the service and display it on the listing page (3.1.1.15). Furthermore, it will have 2 buttons which the user can press to save and book this particular listing. Whenever the buttons are pressed, the control class will be called upon and a return message will be displayed. It will also make any updates to the existing listing information like the decrease in slots available.

nataliecje (Nat Chia)

Coach Bio

Avid Sports Person - I am a expert in all sports - Professional Triathlon Athlete

Price: 150

Location: Buona

Time: 2023-11-16 11:00

Sport: Swimming

Proficiency: med

Availability: 4

Description: Long Distance Swimming Coaching Session

Save Listing

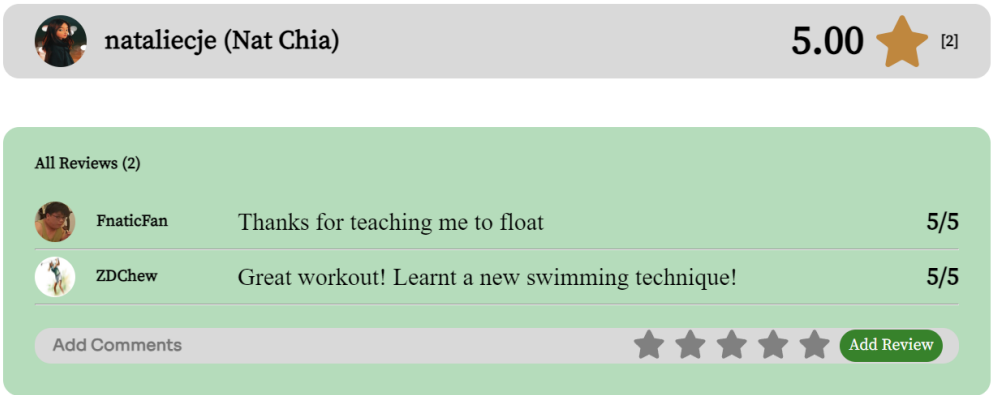
Book Listing

The above image is how the listDisplay component looks like in the final implementation of the website.

3.1.1.15.2 List Review Component

```
return (
    <div className = "listreview-container">
        <div className = "listreview-overall">
            <div className = "listreview-overall-user">
                <img className = "listreview-overall-userimg" src = {userImgSrc} alt = "user-profile-pic"/>
                <span className = "listreview-overall-username">{props.item.coach.username}</span>
            </div>
            <div className = "listreview-overall-stats">
                <span className = "listreview-overall-stats-rating">{props.item.overallRating.toFixed(2)}</span>
                <img className = "listreview-overall-stats-star" src = {STARLOGO} alt = "star-logo"/>
                <span className = "listreview-overall-stats-count">[{props.item.numReviews}]</span>
            </div>
        </div>
        <div className = "listreview-individual">
            <h1 className = "listreview-individual-title">All Reviews ({props.item.numReviews})</h1>
            <div className = "listreview-reviews-container">
                {reviews}
            </div>
            <Form className = "listreview-input" method = "post" action = {`/listing/${id}`}>
                <input type = "hidden" name = "id" value = {props.item.id} />
                <input className = "listreview-individual-input" type = "text" name = "reviewMsg" placeholder = "Add Comments" required/>
                <div className = "listreview-miscallenous">
                    {[...Array(5)].map((star, index) => {
                        const currentRate = index + 1
                        return (
                            <>

                                <label>
                                    <input type = "radio" name = "rate"
                                        value = {currentRate}
                                        onClick={() => setRating(currentRate === rating ? 0 : currentRate)}
                                    />
                                    <FaStar size = {40}
                                        color = { currentRate <= (rateColor || rating) ? "yellow" : "grey"}
                                    />
                                </label>
                            </>
                        )
                    })}
                    <button type = "submit" onClick={handleSubmit}>Add Review</button>
                </div>
            </Form>
            <span className = "error">{data}</span>
        </div>

    </div>
)
```

The list review component will take in all the review information regarding the service and display it on the listing page (3.1.1.15). Furthermore, it will have a form in which the user can input their review message and a rating. This form will be submitted to the review action (3.1.1.15.3) to be processed. After the review message has been sent, it will update the listing page with the new review.



The above image is how the listReview component looks like in the final implementation of the website.

3.1.1.15.3 Review Action

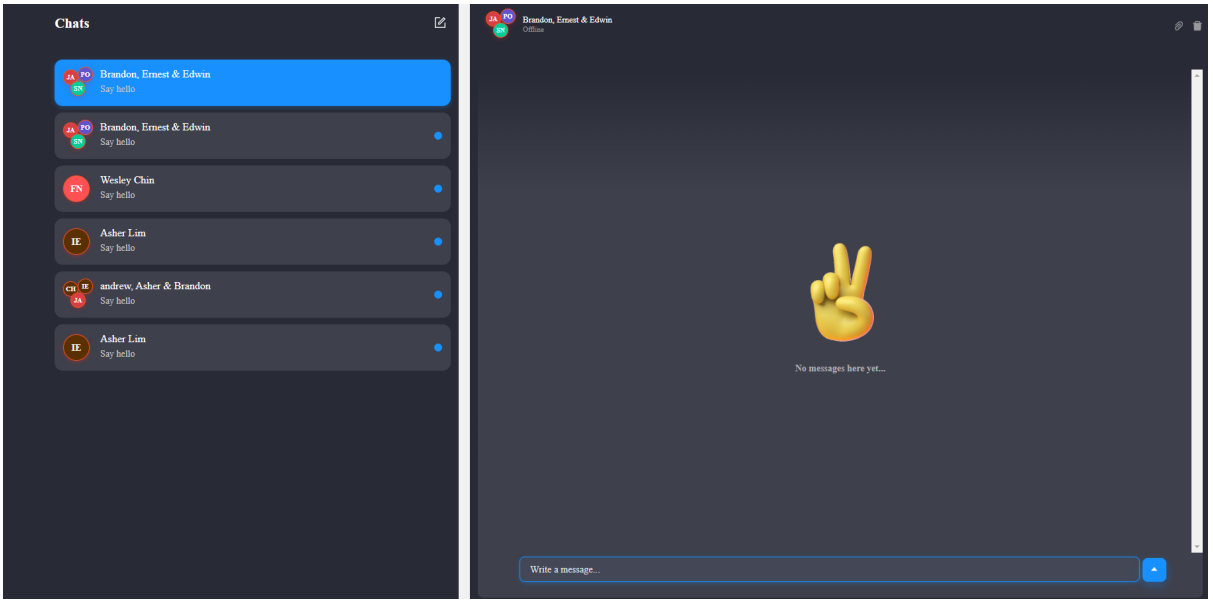
```
export const reviewAction = async ({ request }) => {
    const data = await request.formData()

    const submission = {
        reviewMsg: data.get("reviewMsg"),
        rating: data.get("rate") !== null ? parseInt(data.get("rate")) : 0,
    }

    console.log(submission)
    try {
        const URL = `/review/${data.get("id")}`
        await axios.post(URL, submission)
        return null
    } catch (error) {
        console.log("Enter catch block")
        const errorMessage = error.response.data.message
        return errorMessage
    }
}
```

Review action takes in the form information that was submitted and submit it to the review control class (3.1.2.3). If the review is uploaded successfully, it will return a signal to the list review component (3.1.1.15.2) to update. If the review is uploaded unsuccessfully, it will pass back the error message.

3.1.1.16 Chat

```
export default function Chat() {
    const [userChat, setUserData] = useState(null);
    useEffect(() => {
        axios.get("/mychats")
        .then((response) => {
            console.log(response.data)
            setUserData(response.data);
        })
        .catch((error) => {
            console.error("Error fetching user data:", error);
        });
    }, []);

    if(userChat == null)
      return <p>Loading...</p>

    return (
        <div style={{ height: "100vh", width: "100vw" }}>
          <PrettyChatWindow
            projectId={"8b57194b-974a-406b-aeb2-b042c3112d85"}
            username={userChat.username}
            secret={userChat.username}
            style={{ height: "100%" }}
          />
        </div>
      );
}
```

Chat page calls upon the user controller(3.1.2.1) to get the information of the user. It passes this information to an online chat engine which generates and stores all the chat messages between our different users. There is also a separate online database from the chat engine to store the messages.



The above image is how the chat page looks like in the final implementation of the website.

3.1.2 Controller Classes

3.1.2.1 User Controller

```python
@blp.route("/register")
class UserRegister(MethodView):
    @blp.arguments(UserSchema)
    def post(self, user_data):
        """ Registers a user given a JSON input of their details"""

        if UserModel.query.filter(UserModel.username == user_data["username"]).first():
            abort(409, message="A user with that username already exists.")

        user = UserModel(
            firstname = user_data["firstname"],
            lastname = user_data["lastname"],
            email = user_data["email"],
            username=user_data["username"],
            password=pbkdf2_sha256.hash(user_data["password"]),
            dob = user_data["dob"],
            gender = user_data["gender"],
            userImg = user_data["userImg"],
            bio = user_data["bio"]

        )
        db.session.add(user)
        db.session.commit()

        #call chatAPI to create user in their database
        response = ChatAPI.create_user(user_data)

        return {"message": "User created successfully.", "chat_api_response":response}, 200
```

This function is used to help users register for an account. A valid registration adds user details into the user entity, and creates a chat account through chatAPI.

```python
@blp.route("/login")
class UserLogin(MethodView):
    @blp.arguments(LoginSchema)
    def post(self, user_data):
        """ Authenticates a user via username and password with the database, grants them an access token and a refresh token"""

        user = UserModel.query.filter(
            UserModel.username == user_data["username"]
        ).first()

        # Havent Give user a refresh token
        if user and pbkdf2_sha256.verify(user_data["password"], user.password):
            access_token = create_access_token(identity=user.id)
            response = jsonify({"access_token": access_token})
            set_access_cookies(response, access_token)
            return response, 200

        abort(401, message="Invalid credentials.")
```

This function is used to log users in with their credentials, which are validated by the user entity. Upon successful login, browser cookie is set to a fresh JSON Web Token access token.

```python
@blp.route("/user")
class User(MethodView):

    @blp.response(200, UserSchema)
    @jwt_required()
    def get(self):
        """ Retrieve user id """
        user_id = get_jwt_identity()
        user = UserModel.query.get(user_id)
        return user

    @blp.arguments(UserUpdateSchema)
    @blp.response(200, UserSchema)
    @jwt_required()
    def put(self, user_data):
        """ Edit user profile """
        user_id = get_jwt_identity()
        user = UserModel.query.get(user_id)

        user.email = user_data["email"]
        user.dob = user_data["dob"]
        user.bio = user_data["bio"]
        user.username = user_data["username"]
        if user_data["userImg"] != "":
            user.userImg = user_data["userImg"]
        db.session.commit()

        return user, 201
```

This function allows authenticated users to get their user profile and also to update it. Valid request will update user details in user entity.

```python
@blp.route("/user/changepassword")
class ChangePassword(MethodView):
    @blp.arguments(UserChangePasswordSchema)
    @jwt_required()
    def put(self,user_data):
        """ Change Password """
        user_id = get_jwt_identity()
        user = UserModel.query.get(user_id)
        if pbkdf2_sha256.verify(user_data["password"], user.password) == False:
            abort(400, message="Old password does not match current password")

        if pbkdf2_sha256.verify(user_data["newpw"], user.password):
            abort(409, message="Password needs to be different from old password.")

        user.password = pbkdf2_sha256.hash(user_data["newpw"])
        db.session.commit()
        return jsonify({"message": "password changed"}), 201
```

This function is used to change a user's password. Valid request will update user password in user entity.

```python
@blp.route("/user/booked")
class ViewBooked(MethodView):
    @jwt_required()
    @blp.response(200, PlainCoachingServiceSchema(many = True))
    def get(self):
        """ Retrieve User's booking services """
        user_id = get_jwt_identity()
        user = UserModel.query.get(user_id)

        bookings = user.booked
        if bookings:
            bookings.reverse()
            return bookings
        else:
            return jsonify({"message": "You have not booked any services"}), 200
```

This function is used to access the services that the user has booked and returns the data to the UI.

```python
@blp.route("/user/saved")
class ViewSaved(MethodView):
    @jwt_required()
    @blp.response(200, PlainCoachingServiceSchema(many = True))
    def get(self):
        """ Retrieve user's saved coaching services """
        user_id = get_jwt_identity()
        user = UserModel.query.get(user_id)
        saved = user.saved
        if saved:
            saved.reverse()
            return saved
        else:
            return jsonify({"message": "You have not saved any services"}), 200
```

This function is used to access the services that the user has saved and returns the data to the UI.

```python
@blp.route("/user/listings")
class ViewListings(MethodView):
    @jwt_required()
    @blp.response(200, PlainCoachingServiceSchema(many = True))
    def get(self):
        """ Retrieve user's listed coaching services """
        user_id = get_jwt_identity()
        user = UserModel.query.get(user_id)
        listings = user.listings.all()
        if listings:
            listings.reverse()
            return listings
        else:
            return jsonify({"message": "You have not listed any services"}), 200
```

This function is used to access the services that the user has listed and returns the data to the UI.

3.1.2.2 CoachingService Controller

```python
@blp.route("/coaching_services/filter")
class CoachingServiceSearch(MethodView):

    @blp.arguments(CoachingServiceFilterSchema)
    @blp.response(201, CoachingServiceSchema(many = True))
    def post(self, filters):
        """ Gets a list of filtered coaching services """
        query = db.session.query(CoachingServiceModel).order_by(CoachingServiceModel.price.desc()).all()

        if "proficiency" in filters:
            query = [service for service in query if service.proficiency == filters["proficiency"]]

        if "price" in filters:
            query = [service for service in query if service.price <= filters["price"]]

        if "sport" in filters:
            query = [service for service in query if service.sport == filters["sport"]]

        return query
```

This function is used to filter a set of listings based on the values given to the backend. It returns the filtered listings to the UI.

```python
@blp.route("/coaching_services")
class CoachingServiceList(MethodView):
    @blp.response(200, CoachingServiceSchema(many=True))
    def get(self):
        """ Retrieve all listing services """

        # Order it from last service to first service
        return CoachingServiceModel.query.order_by(CoachingServiceModel.id.desc()).all()
```

This function is used to retrieve all listings in the marketplace from the coaching service entity.

```python
@blp.arguments(CoachingServiceSchema)
@blp.response(201, CoachingServiceSchema)
@jwt_required()
def post(self, coaching_service_data):
    """ List coaching service """
    user_id = get_jwt_identity()
    user = UserModel.query.get(user_id)

    coaching_service = CoachingServiceModel(**coaching_service_data, coach_id = user_id, numReviews = 0, overallRating = 0, available = coaching_service_data["maximum"])
    response, createdChat = ChatAPI.create_grp_chat(user, coaching_service_data["description"])
    coaching_service.chat_id = response["id"]
    try:
        db.session.add(coaching_service)
        db.session.commit()

    except SQLAlchemyError:
        abort(500, message="An error occurred creating the service.")

    return coaching_service
```

This function takes in details of a coaching service to create a listing in the coaching service entity. It also creates a group chat through chatAPI with the coach as the admin of the group.

```python
@blp.route("/services/<string:listing_id>")
class Services(MethodView):
    @blp.response(200, CoachingServiceSchema)
    def get(self, listing_id):
        """ Retrieve a specific listing given listing id """
        coaching_service = CoachingServiceModel.query.get_or_404(listing_id)
        coaching_service.reviews.reverse()
        return coaching_service
```

This function is used to retrieve a specific listing by its given ID.

```python
    @jwt_required()
    def delete(self, listing_id):
        """ Delete coaching services """
        user_id = get_jwt_identity()
        user = UserModel.query.get(user_id)
        coaching_service = CoachingServiceModel.query.get_or_404(listing_id)


        if coaching_service in user.listings:
            db.session.delete(coaching_service)
            db.session.commit()
            response, deleted = ChatAPI.delete_grp_chat(coaching_service)
            return {"message": "Item and Group Chat deleted.", "Group chat deletion status": deleted}
        else:
            return {"message": "You do not have the rights to delete this listing."}
```

This function is used to delete a specific listing by its given ID from the coaching service entity. It also deletes the group chat that corresponds to the service listed.

```python
    @jwt_required()
    @blp.arguments(CoachingServiceUpdateSchema)
    def put(self,coaching_service_data, listing_id):
        """ Edit coaching services listing """
        user_id = get_jwt_identity()
        user = UserModel.query.get(user_id)
        coaching_service = CoachingServiceModel.query.get_or_404(listing_id)

        coaching_service = CoachingServiceModel.query.get(listing_id)

        if coaching_service in user.listings:
            coaching_service.sport = coaching_service_data["sport"]
            coaching_service.location = coaching_service_data["location"]
            coaching_service.price = coaching_service_data["price"]
            coaching_service.description = coaching_service_data["description"]
            coaching_service.proficiency = coaching_service_data['proficiency']
            coaching_service.coverImg = coaching_service_data['coverImg']
            coaching_service.datetime = coaching_service_data['datetime']

            db.session.commit()
            return {"message": "Service Edited"}, 200
        else:
            return {"message": "You do not have the rights to edit this listing."}
```

This function allows the creator of a coaching service to delete the service from the coaching service entity.

```python
@blp.route("/book/<string:listing_id>")
class Booking(MethodView):

    @jwt_required()
    def post(self, listing_id):
        """ Booking of coaching services listing """
        user_id = get_jwt_identity()
        user = UserModel.query.get(user_id)
        coaching_service = CoachingServiceModel.query.get(listing_id)

        if user is None or coaching_service is None:
            return {'message': 'User or service not found'}, 404
        if coaching_service.coach_id == user_id:
            return {'message': 'You cannot book your own listing'}, 400
        if coaching_service in user.booked:
            return {'message': 'You have already booked this service'}, 400
        if coaching_service.available == 0:
            return {'message': 'There are no more available slots'}, 400

        user.booked.append(coaching_service)
        coaching_service.athletes.append(user)

        #add user into as chat member for coaching service grp chat
        joined = ChatAPI.join_grp_chat(user.username, coaching_service)

        coaching_service.available -= 1
        coaching_service.haveNotification = True
        db.session.commit()

        return {'message': 'Service booked successfully', "Joined group chat status": joined}, 200
```

This function allows a user to book a specific coaching service. The user is appended to the “athlete” list in the coaching service entity, and the coaching service is appended to the “booked” list in the user entity. The athlete service entity is updated to store the many to many relational mapping. The user is added to the service group chat.

```python
    @jwt_required()
    def delete(self, listing_id):
        """ Cancel a booking of coaching services"""
        user_id = get_jwt_identity()
        user = UserModel.query.get(user_id)
        coaching_service = CoachingServiceModel.query.get(listing_id)

        if user is None or coaching_service is None:
            return {'message': 'User or service not found'}, 404

        if coaching_service not in user.booked:
            return {'message': 'You have not booked this service yet'}, 400
        user.booked.remove(coaching_service)
        coaching_service.available += 1
        db.session.commit()
        left = ChatAPI.leave_grp_chat(user.username, coaching_service)

        return {'message': 'Service removed successfully from bookings', "Left group chat status": left}
```

This function is used to undo a booking that a user has done. The user will automatically leave the group chat that is linked to the service.

```python
@blp.route("/save/<string:listing_id>")
class Saving(MethodView):

    @jwt_required()
    def post(self, listing_id):
        """ Saving coaching services listing """
        user_id = get_jwt_identity()
        user = UserModel.query.get(user_id)
        coaching_service = CoachingServiceModel.query.get(listing_id)

        if user is None or coaching_service is None:
            return {'message': 'User or service not found'}, 404

        if coaching_service in user.saved:
            return {'message': 'You have already saved this service'}, 400
        user.saved.append(coaching_service)
        db.session.commit()

        return {'message': 'Service saved successfully'}, 200
```

This function allows a user to save a specific coaching service. The coaching service is appended to the “saved” list in the user entity. The athlete saved entity is updated to store the many to many relational mapping.

```python
    @jwt_required()
    def delete(self, listing_id):
        """ Delete the saved coaching service listing """
        user_id = get_jwt_identity()
        user = UserModel.query.get(user_id)
        coaching_service = CoachingServiceModel.query.get(listing_id)

        if user is None or coaching_service is None:
            return {'message': 'User or service not found'}, 404

        if coaching_service not in user.saved:
            return {'message': 'You have not booked this service yet'}, 400
        user.saved.remove(coaching_service)
        db.session.commit()

        return {'message': 'Service removed successfully from saved listings'}
```

This function is used to remove a saved listing from a user’s saved list.

3.1.2.3 Review Controller

```python
@blp.route("/review/<string:listing_id>")
class Review(MethodView):

    @blp.response(201, ReviewSchema(many = True))
    def get(self, listing_id):
        """ Retrieve coaching service reviews """
        coaching_service = CoachingServiceModel.query.get(listing_id)
        reviewList = coaching_service.reviews
        if reviewList:
            return reviewList
        else:
            return jsonify({"message": "This listing has no reviews"}), 200
```

This function is used to retrieve all reviews of a specific listing.

```python
@jwt_required()
@blp.arguments(ReviewSchema)
@blp.response(201, ReviewSchema)
def post(self, review_data, listing_id):
    """ Post a review for the coaching services """
    user_id = get_jwt_identity()
    user = UserModel.query.get(user_id)

    coaching_service = CoachingServiceModel.query.get(listing_id)
    reviewList = coaching_service.reviews

    if user not in coaching_service.athletes:
        response = make_response(jsonify({"message":"Not allowed to review as you did not book the service"}), 400)
        return response

    for review in reviewList:
        if user_id == review.reviewer_id:
            response = make_response(jsonify({"message":"Not allowed to review as you have already reviewed the service"}), 400)
            return response

    review = ReviewModel(**review_data, reviewer_id = user_id)
    reviewList.insert(0, review)

    coaching_service.numReviews += 1
    coaching_service.overallRating = ((coaching_service.overallRating * (coaching_service.numReviews - 1)+
    review.rating)/coaching_service.numReviews

    try:
        db.session.add(review)
        db.session.commit()
    except SQLAlchemyError:
        abort(500, message="An error occurred creating the review.")
    return review
```

This function is used to post a review of a specific listing. It updates the “review” list in the coaching service entity and creates a new row of a review object in the review entity.

3.1.2.4 ChatApi controller

```python
@staticmethod
def delete_grp_chat(coaching_service):
    coach = coaching_service.coach
    response = requests.delete(f'https://api.chatengine.io/chats/{coaching_service.chat_id}/',
                               headers={"Project-ID": ChatAPI.project_id, "User-Name": coach.username, "User-Secret": coach.username})

    if response.status_code == 200:
        return response.json(), True
    else:
        return "Error creating group"

@staticmethod
def join_grp_chat(username, coaching_service):
    coach_username = coaching_service.coach.username
    chat_id = coaching_service.chat_id
    response = requests.post(f'https://api.chatengine.io/chats/{chat_id}/people/', data={"username": username},
    headers={"Project-ID": ChatAPI.project_id, "User-Name": coach_username, "User-Secret": coach_username})
    if response.status_code == 201:
        return True
    else:
        return False

@staticmethod
def leave_grp_chat(username, coaching_service):
    coach_username = coaching_service.coach.username
    chat_id = coaching_service.chat_id
    response = requests.put(f'https://api.chatengine.io/chats/{chat_id}/people/', data={"username": username},
                            headers={"Project-ID": ChatAPI.project_id, "User-Name": coach_username, "User-Secret": coach_username})
    print(response.json())
    if response.status_code == 200:
        return True
    else:
        return False
```

```python
@staticmethod
def get_user(user):
    response = requests.get('https://api.chatengine.io/users/me/',headers={"Project-ID": ChatAPI.project_id,"User-Name": user.username,"User-Secret": user.username})
    return response.json()

@staticmethod
def create_user(user_data):
    response = requests.post('https://api.chatengine.io/users/',data={"username": user_data["username"],"secret": user_data["username"],"email": user_data["email"],
                             "first_name": user_data["firstname"],"last_name": user_data["lastname"],},
                             headers={"Private-Key": ChatAPI.private_key})
    return response.json()

@staticmethod
def get_all_users():
    response = requests.get('https://api.chatengine.io/users/',headers={"Private-Key": ChatAPI.private_key})
    users = [(user["username"], user["id"]) for user in response.json()]
    return users

@staticmethod
def delete_all_users():
    users = ChatAPI.get_all_users()
    for user in users:
        user_id = user[1]
        requests.delete(f'https://api.chatengine.io/users/{user_id}',
                        headers={"Private-Key": ChatAPI.private_key})
    return

@staticmethod
def create_grp_chat(user, title="Group Chat"):
    response = requests.post('https://api.chatengine.io/chats/', data={"title": title, "is_direct_chat": False},
                             headers={"Project-ID": ChatAPI.project_id, "User-Name": user.username, "User-Secret": user.username})

    if response.status_code == 201:
        return response.json(), True
    else:
        return "Error creating group"
```

This class has the functions that are required to create requests to the chatAPI. These functions are used in the endpoints.

3.1.3 Entity Classes

3.1.3.1 User Entity

```python
class UserModel(db.Model):
    __tablename__ = "user"

    id = db.Column(db.Integer, primary_key=True)
    firstname = db.Column(db.String(80), nullable=False)
    lastname = db.Column(db.String(80), nullable=False)
    email = db.Column(db.String(80), nullable=False)
    dob = db.Column(db.String(80), nullable=False)
    bio = db.Column(db.String(80), nullable=True)
    username = db.Column(db.String(80), unique=True, nullable=False)
    password = db.Column(db.String(255), nullable=False)
    gender = db.Column(db.String(80), nullable=False)
    userImg = db.Column(db.String(80), nullable=True)




    #many to many relationships
    booked = db.relationship("CoachingServiceModel", back_populates="athletes", secondary="athlete_service")

    saved = db.relationship("CoachingServiceModel", secondary="athlete_saved")

    #One to many relationships
    listings = db.relationship("CoachingServiceModel", back_populates="coach", lazy="dynamic", cascade="all, delete")

    reviews = db.relationship("ReviewModel", back_populates="reviewer", lazy="dynamic")
```

This class contains data for User Profiles and their relationships with Coaching Service Entity and Review Entity

3.1.3.2 Coaching Service Entity

```python
class CoachingServiceModel(db.Model):

    __tablename__ = "coaching_service"

    id = db.Column(db.Integer, primary_key=True)
    sport = db.Column(db.String(80), nullable=False)
    datetime = db.Column(db.String(80), nullable=False)
    location = db.Column(db.String(120), nullable=False)
    price = db.Column(db.Float(precision=2), nullable=False)
    description = db.Column(db.String(255))
    coverImg = db.Column(db.String(255))
    proficiency = db.Column(db.String(255))
    overallRating = db.Column(db.Float(precision=2))
    numReviews = db.Column(db.Integer)
    chat_id = db.Column(db.Integer, nullable=False)

    maximum = db.Column(db.Integer)
    available = db.Column(db.Integer)
    haveNotification = db.Column(db.Boolean, default = False, nullable = False)

    #many to many relation
    athletes = db.relationship("UserModel", back_populates="booked", secondary="athlete_service")

    #many to one relation
    coach_id = db.Column(db.Integer, db.ForeignKey('user.id'), nullable = False)
    coach = db.relationship("UserModel", back_populates="listings")

    #one to many relation
    reviews = db.relationship("ReviewModel", back_populates="service", cascade="all, delete")
```

This class contains data for Coaching Services and their relations to User Entity and Review Entity

3.1.3.3 Review Entity

```python
class ReviewModel(db.Model):

    __tablename__ = "review"

    id = db.Column(db.Integer, primary_key=True)
    rating = db.Column(db.Integer, nullable=False)
    reviewMsg = db.Column(db.String(255))
    picture = db.Column(db.String(255))

    #many to one relationships
    service_id = db.Column(db.Integer, db.ForeignKey('coaching_service.id'), nullable=False)
    service = db.relationship("CoachingServiceModel", back_populates="reviews")

    #many to one relationships
    reviewer_id = db.Column(db.Integer, db.ForeignKey("user.id"), unique=False, nullable=False)
    reviewer = db.relationship("UserModel", back_populates = "reviews")
```

This class contains data for Review and their relationships with Coaching Service Entity and User Entity

3.2 Source Code

For the source code, please refer to our github repository.
https://github.com/DennisDCCH/Techies

3.3 Live Demo Script

Actors: Andrew (main), Natalie (friend), Asher (coach)
Narrator: Zhaoding
PowerPoint presenter: Brandon
Presenter: Dennis

Scene	Description	Script	Use Cases
1	Friend asks the main character whether he wants to train, showing that they need a coach. See got new app, **jump to motivation of app (brandon ppt)**	Good afternoon TA and fellow classmates, We are group 1 Techies, now we will be starting our presentation for SportSync. Narrator: The soccer inter hall games is coming up in a month, and Andrew has the idea to train to prepare for the upcoming game Andrew: *Plays with ball by himself Natalie: Yo! What are you doing? Andrew: Training for IHG! Do you want to join? Natalie: Yea sure! Eh, why not get a coach instead of training by ourselves? Recently I saw this website called SportSync which allows us to book training sessions with a coach. That would be so much more fruitful. Actually, I have already booked one session next week! **Jump to brandon** **Dennis** Now I will be going through a brief rundown of the different functionalities of our website. So after we log in, we will reach the homepage of our website, which will contain a navigation bar as well as a main content segment. At the navigation bar, is where we can have our users access their profile and anything linked to their accounts and the services that they might provide or the services that they have previously booked or saved. So here we can see the general information of our user, and there is an edit profile button which allows the user to change or add new information to their account. Next, all of our users are also able to create their own listing and services which they will be able to see here on this page. Moving on, users can see their booked and saved services on the following 2 buttons, which they have the option of having 2 buttons to delete or select the services to show them this	nil

		specific service details. Lastly, our website also allows our users to change their password. Moving on to the main content segment, here we have the activities, map and chat buttons. The activities button brings you to a marketplace with all existing listings on our website which you can explore. This marketplace has a filter function which allows the user to filter the specific sports, the proficiency level they are looking for or the maximum price they are willing to pay. Next we have a map page which shows the user all existing taxis available near his current area. The default location when first entering this page will always be the current live location, but we can change it to anywhere we like, for instance somewhere in Sengkang. Lastly, we have a chat page which enables our users to communicate with one another and make any negotiation they want separately through this feature. With that, we will carry on with our live demonstration.	
2	Pre: Friend 1 created an account and booked a soccer listing. **(DONE)** Friend asks the main character to create an account and log in.	Natalie: Isn't the app great? Andrew! Go and create an account and book the same listing as me! Andrew: Ok sure! Seems like I need to give some information about myself! I wonder what happens if I use a username that natalie usually uses hehe. Narrator: Andrew tries to register for an account with Natalie's username. (invalid username) Natalie: Hello Andrew isn't that's my username, what are you trying to do… Andrew: Maybe we can have the same ***registers unsuccessfully with Potatbuttock*** OOPS wait let me change it up. Natalie: Yup, make sure that your password is at least 8 characters, with at least 1 upper case, 1 lower case, 1 number & 1 special case character. ***register successfully*** Andrew: Ok I registered! Let me log into my account now. Narrator: Andrew logs in to his account. ***logs in unsuccessfully*** Andrew: Oops, must be a typo of my password	Register, login AF: Register invalid, login invalid

		logs in successfully	
3	Main character want to edit his email/bio/picture/DOB	Natalie: Andrew, do you know that you can edit your profile and create your own bio? ***go to account info page*** Andrew: For real? Okay, let me update my bio real quick. I would like to let people know that I am an avid soccer player and also show them a picture of myself. ***insert selfie and bio "I am an avid soccer player!"*** Narrator: Andrew proceeds to edit his profile and add a brief bio of himself.	Edit Profile
4	Friend asks the main character to filter the sport of choice and book the soccer listing that friend has booked. Main character checks that it is in his booked listings.	Natalie: Ok now you can try to book the listing by filtering the listings. You can go to the marketplace and filter according to your choice. ***go to marketplace page*** Natalie: Since you wanna book the same soccer listing as me, you can just type "soccer" into the filter input. ***filter sport - "Soccer"*** Andrew: Okay, that sounds pretty straightforward. Eh, this is the correct listing right? Natalie: Yup, then you click on the 'select listing' button, and then book the listing. ***Enter the specific listing and book the listing*** Narrator: Andrew books the listing and checks that it is in his booked listings page ***Goes to booking page*** Andrew: Great, looks like I have the listing in my booked listings page.	Book Listing, Filter Listing
5	Pre: Friend 1 booked a mahjong listing. Availability of listing: 1 **(DONE)** Friend 1 asks main character to save mahjong listing	Natalie: Yo! Do you want to go swimming next week as well? Andrew: Huh!?? I can't even swim… what do you mean! Natalie: Oh, I meant dry swimming! Do you want to play mahjong? There's one more availability in this listing that I have booked. ***Goes to marketplace page***	Save Listing (links to show cannot when unavailable later on)

		Andrew: Hmmm, I don't mind, but I need to check my calendar first! Maybe I'll save the listing so that I can book it in the future. I hope that there are still available slots! ***Goes to specific listing page and save the listing*** Narrator: Andrew saves the mahjong listing and sees that it is in his saved listings page. ***Goes to saved listing page*** Andrew: Wonderful, its already in my saved listings page.	
6	Pre: Friend has sent a message to group chat "Hey coach! My friend is relatively new! Do guide him!" Main character goes to the chat page and sees that friend and coach are already in group chat. Main character sends a new message to introduce himself	Narrator: Prior to the day of training, Natalie has sent a message to group chat saying that Andrew is relatively new and will be joining in a while. Natalie: Yo Andrew, did you know that there is a group chat function? You are already in the groupchat as soon as you booked the listing ***Go to the chat page!*** Andrew: Oh damn, I'm so excited to go for the first training. Let me just introduce myself in the groupchat! ***Insert:** **Hey guys, I am Andrew!** **Excited to be here!***	Chat function (athlete)
7	Direct message coach	Natalie: Eh can we train bare-footed? Andrew: Hm, maybe i should direct message the coach to check on this! ***At chat page, dm Asher Lim and ask** **Hey coach, can we train bare-footed?*** Narrator: Andrew proceeds to find the coach and ask the coach if he can play barefoot	Chat function
8	Coach logs in to see a new notification of booking. Coach chats with athletes "Hey guys remember to bring your boots and be punctual!"	***switch to coach tab*** Narrator: The coach logs in to his account to check whether he has new messages or athletes that have joined his session ***logs in with ieatsighpies account*** Asher: Oh, it seems like I have a new notification. Let's check it out! Now the availability of the coaching session has decreased. Oh right, better remind the athletes to bring their training equipment and be punctual on the day itself.	Chat function (coach) Notification of new booking

		go to chat page and** **"Hey guys! Excited to meet yall soon! Remember to bring your gear and be punctual!" Narrator: The coach goes to the group chat and sends a reminder message to the athletes. Coach: Wah this athlete is really funny ah… ask me if he can train barefoot. Obviously cannot ah! Let me just text him back ***Cannot lah, for your safety please bring proper shoes*** **Jump to brandon** ***dennis faster log out and book mahjong** **Logs in with jangjangjang account book both mahjong activities** **Logs in back with cheamy account***	
8	*Day of the training:* Main Character and friend use the taxiAPI to book a taxi and arrive at the location. It was a very good session	Narrator: On the day of the training session, Andrew and Natalie decided to make use of the taxi API to search and book available taxis nearby. Natalie: Oh no it's raining, should we book a taxi to the training venue? Andrew: Yea sure! Lets use the taxi booking service on the app! Seems like all we need to do is click one button. ***Goes to map page, and press book taxi button*** Narrator: With a simple click on the screen, they successfully booked a taxi and arrived at the location as planned.	TaxiAPI
	Coach training session	Impromptu PT session by asher	funfun
9	Pre: Main character has booked the soccer listing and came back from training. Main character posts a review under the soccer listing "Fantastic training session, I have learnt a lot!", clicks on the 5-star.	Narrator: After a fruitful soccer session, Andrew finds the coach to be not just instructive but also highly encouraging, he is very satisfied with the training, prompting him to leave a 5-star review in the website as a token of appreciation ***Goes to specific listing page - give 5 star review** **Highly recommend Asher, very encouraging and instructive"** Andrew: Phew, that was a good workout! I'm gonna leave a 5-star review for the coach and his service! Hopefully this review will help other athletes to know that this coach is good!	Review
10	Pre: One more person has booked the listing.	Narrator: After leaving a review, Andrew browses through his saved listing and sees the mahjong listing he saved previously	Book Listing

	Availability of mahjong listing: 0 (Done on the spot) Main character tries to book the mahjong listing that he saved, but fails because there is no more slot left.	and intends to book it. Andrew: Let's take a look at the mahjong session that I saved earlier. When I saved it, there was only one slot left. I hope that there are still slots for me to play! ***Goes to saved listing page and select the specific listing*** Oh man… there aren't anymore available slots! Let me just try booking to see what happens if the availability is 0. ***Press book button and sees error message*** Oh well, maybe next time then..	invalid
11	Main character creates his own soccer listing after attending sessions of training	Andrew: Hey Natalie, after going for so many training sessions, I really feel like I want to contribute back to the ecosystem! Should I create a listing to coach beginners to play soccer? Natalie: Why not! 🙂 Narrator: After multiple training sessions, Andrew's soccer skills have improved immensely. In a bid to spread his passion for soccer, Andrew took the initiative to create a beginner-friendly soccer listing for people who want to improve their skills. ***Goes to create listing page** **Price: 10** **Location: Bedok** **Description: Fun and easy beginer socer course!*** Andrew: I'm going to create an affordable listing with a low proficiency level to cater to beginners! I also want to have it near my own house, and put a cool profile picture so that people will be interested!	Create Listing
12	Main character edit listing to fix a typo.	Natalie: Eh there's a typo in your description! Why not try to edit your listing with the edit listing function! Andrew: Oh woops, totally did not see that mistake. Thankfully this function exists! ***Edit Listing button** **Description: Fun and easy beginner soccer course!*** Narrator: Andrew proceeds to edit his listing.	Edit Listing
13	Main character changes the password to make it more secure.	Natalie: Wait Andrew just a reminder, nowadays got many hackers maybe you should consider a more complicated password. Andrew: Ok I don't want people to steal my information. Let me just change my password to a longer one. Do you know how to change?	Change Password

		Natalie: Yup you can go into your profile, then click on change password at the sidebar. ***Goes to change password page and changes password*** Narrator: After Andrew changed his password, he logged out and tried logging in again to check. ***Logs back in with new password*** Andrew: Wah, this website is so useful! Whoever made this website did a fantastic job!! Deserve an A+! **Jumps to Brandon**	

3.4 Demo Video

For the video demonstrating our project, please refer to the following youtube video.
https://youtu.be/1rgdCgu9vcY

4 Testing

4.1 Black-Box Testing

4.1.1 Functionality: Registration

TestID	Test Input	Expected Output	Actual Output
1.a.1	Invalid email format	Return error message 'Please include an '@' in the email address' Or Return error message 'Please enter a part following '@'.'	Return error message 'Please include an '@' in the email address' Or Return error message 'Please enter a part following '@'.'
1.a.2	Password does not meet specification Requirements: 1 Uppercase, 1 lowercase, 1 Special character, at least 8 Characters	Return error message for password must have at least 1 Uppercase, 1 lowercase, 1 Special character, at least 8 Characters	Return error message for password must have at least 1 Uppercase, 1 lowercase, 1 Special character, at least 8 Characters
1.a.3	Invalid username (username already registered)	Return error message for username already exists	Return error message for username already exists
1.a.4	Re-enter password does not match	Return error message for the password is not the same	Return error message for the password is not the same

1.a.5	Submitting with invalid fields	Return error message ‘Please fill out this field’	Return error message ‘Please fill out this field’
1.a.6	Submitting with all fields valid	Redirected to Login Page	Redirected to Login Page

4.1.2 Functionality: Login

TestID	**Test Input**	**Expected Output**	**Actual Output**
2.1	Invalid username and password	Returns error message for invalid login credentials	Returns error message for invalid login credentials
2.2	Valid username and password	Redirects to Homepage	Redirects to Homepage

4.1.3 Functionality: Edit Profile

TestID	**Test Input**	**Expected Output**	**Actual Output**
3.1	Press ‘Edit Profile’ button	Redirects to Edit Profile Page	Redirects to Edit Profile Page
3.2	Invalid username (Username already registered)	Return error message for username already exists	Return error message for username already exists
3.3	Valid input	Redirects to Profile Page with updated profile information	Redirects to Profile Page with updated profile information

4.1.4 Functionality: Create Listing

TestID	Test Input	Expected Output	Actual Output
4.1	Press ‘Create new listing’ in My listing page	Redirects to Create Listing Page	Redirects to Create Listing Page
4.2	Submitting with invalid fields	Return error message ‘Please fill out this field’	Return error message ‘Please fill out this field’
4.3	Submitting with valid fields	Redirects to My Listing Page	Redirects to My Listing Page

4.1.5 Functionality: Edit listing

TestID	Test Input	Expected Output	Actual Output
5.1	Press ‘Edit’ button	Redirects to Edit Listing Page	Redirects to Edit Listing Page
5.1	No input of ‘Price’	Return ‘-1’ in the ‘Price’ field	Return ‘-1’ in the Price field
5.2	Submitting with valid fields	Edit Page showed updated inputs	Redirected to login page

4.1.6 Functionality: Save a listing

TestID	Test Input	Expected Output	Actual Output
6.1	Press ‘Select Listing’	Redirects to Listing Page	Redirects to Listing Page
6.2	Press ‘Save Listing’	Return message ‘Service saved successfully’	Return message ‘Service saved successfully’
6.3	Press ‘Save Listing’ repeatedly	Return message ‘You have already saved this service’	Return message ‘You have already saved this service’

4.1.7 Functionality: Book listing

TestID	Test Input	Expected Output	Actual Output
7.1	Press ‘VIew Listing’	Redirects to Listing Page	Redirects to Listing Page
7.2	Press ‘Book Listing’ button on your own listing	Return message ‘You cannot book your own listing’	Return message ‘You cannot book your own listing’
7.3	Press ‘Book Listing’ button on others’ listing	Return message ‘Service booked successfully’	Return message ‘Service booked successfully’

4.1.8 Functionality: Delete my / saved / booked listings

TestID	Test Input	Expected Output	Actual Output
8.1	Press ‘Profile’ button	Redirects to Profile Page	Redirects to Profile Page

8.2	Press ‘My / Saved / Booked listings”	Redirects to My / Saved / Booked Listing Page	Redirects to My / Saved / Booked Listing Page
8.3	Press ‘Delete’ button	Listing deleted	Listing deleted

4.1.9 Functionality: Change Password

TestID	**Test Input**	**Expected Output**	**Actual Output**
9.1	Invalid input	Return error message for password must have at least 1 Uppercase, 1 lowercase, 1 Special character, at least 8 Characters	Return error message for password must have at least 1 Uppercase, 1 lowercase, 1 Special character, at least 8 Characters
9.2	Same password as old password	Return error message ‘Password needs to be different from old password’.	Return error message ‘Password needs to be different from old password’.
9.3	Re-enter password does not match	Return error message ‘the password is not the same’	Return error message ‘the password is not the same’
9.4	Valid input	Redirects to Login Page	Redirects to Login Page

4.1.10 Functionality: Search & select listing

TestID	**Test Input**	**Expected Output**	**Actual Output**
10.1	Press ‘Select filter’ after entering inputs for ‘Sports’	Show all filtered listings	Show all filtered listings

	and/or ‘Proficiency’ and/or ‘Price’		
10.2	Press ‘Select filter’ without entering any inputs	Show all listings	Show all listings
10.4	Press ‘View Listing’	Redirects to Listing Page	Redirects to Listing Page

4.1.11 Functionality: Review

TestID	**Test Input**	**Expected Output**	**Actual Output**
11.1	Press ‘View Listing’	Redirects to Listing Page	Redirects to Listing Page
11.2	Post a review without booking	Return error message ‘Not allowed to review as you did not book the service’	Return error message ‘Not allowed to review as you did not book the service’
11.3	Post a review after ‘Book’ button pressed	Review successfully posted under the listing	Review successfully posted under the listing

4.1.12 Functionality: Taxi Availability

TestID	**Test Input**	**Expected Output**	**Actual Output**
13.1	Pressing search button with invalid location input	Return error message ‘Please enter a valid address’	Return error message ‘Please enter a valid address’

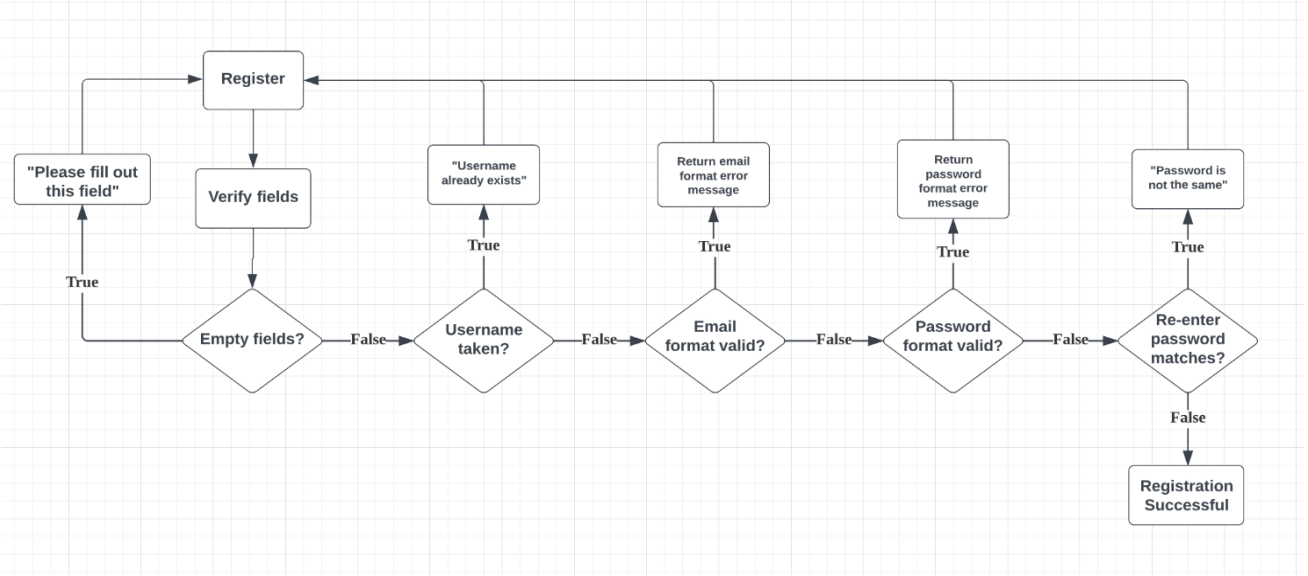
13.2	Pressing search button with valid location input	Return map with amount and location of available taxis near the input location.	Return map with amount and location of available taxis near the input location.
13.3	Press 'Live Location' button	Return map with amount and location of available taxis near the input location.	Return map with amount and location of available taxis near the input location.
13.4	Press 'Grab a Taxi!'	Return 'You have successfully booked a taxi'	Return 'You have successfully booked a taxi'
13.5	Press 'Grab a Taxi!' repeatedly	Return 'You have previously booked a taxi already'	Return 'You have previously booked a taxi already'

4.1.13 Functionality: Chat

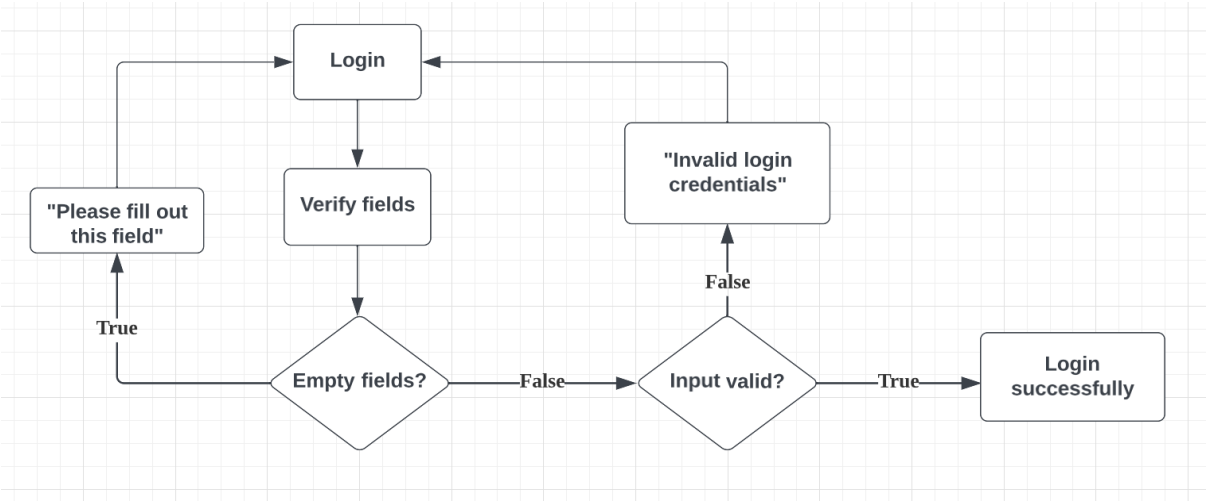
TestID	**Test Input**	**Expected Output**	**Actual Output**
14.1	Clicking 'Chat' button	Display conversation history for current chat	Display conversation history for current chat
14.2	Inputting message and pressing send	Sends message to all members of the chat Updates conversation history to show message	Sends message to all members of the chat Updates conversation history to show message

4.2 White-Box Testing

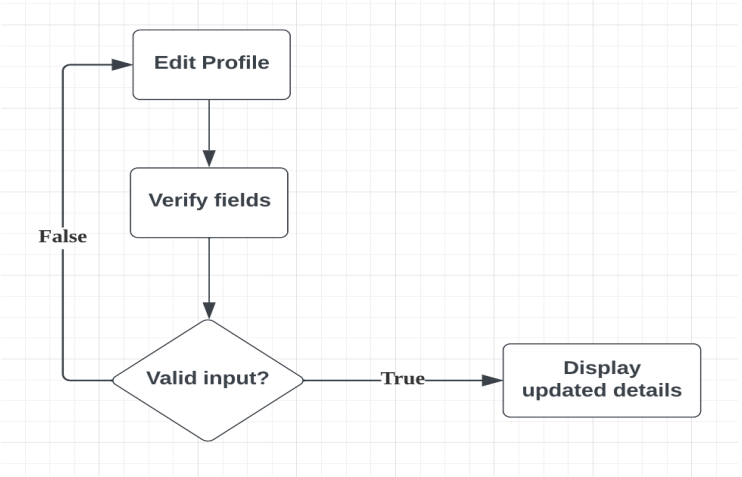
4.2.1 Functionality: Registration



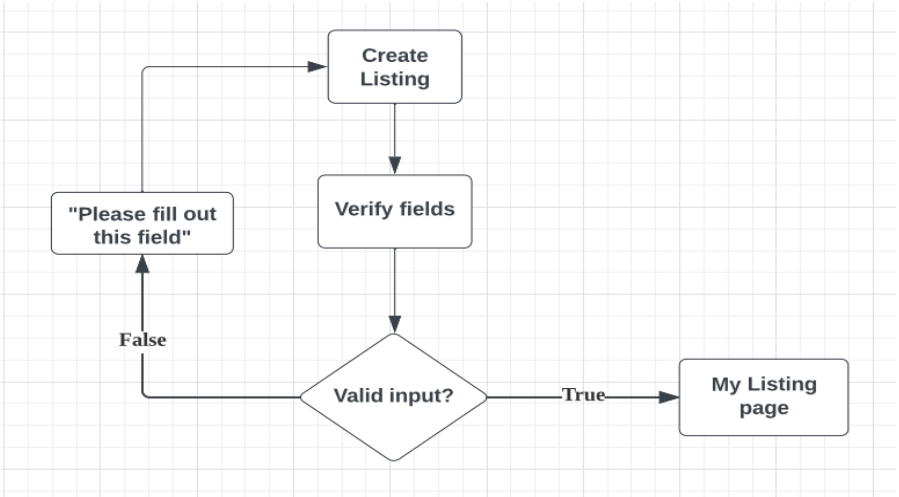
4.2.2 Functionality: Login



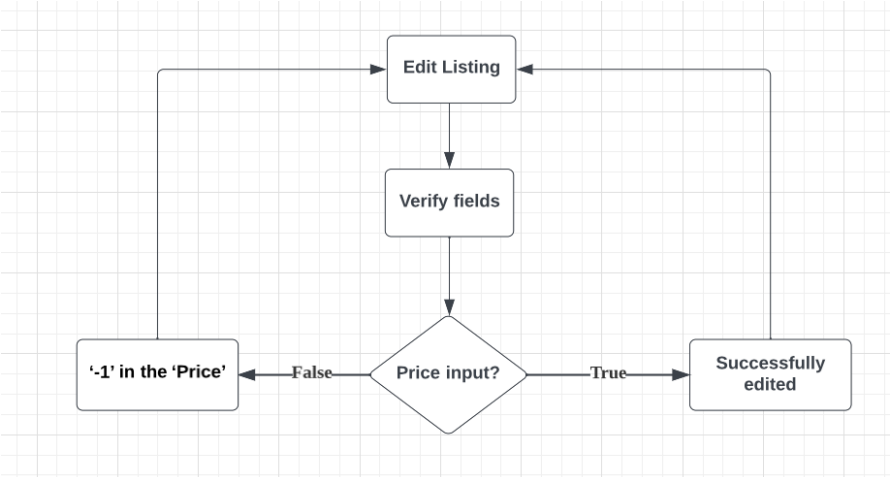
4.2.3 Functionality: Edit Profile



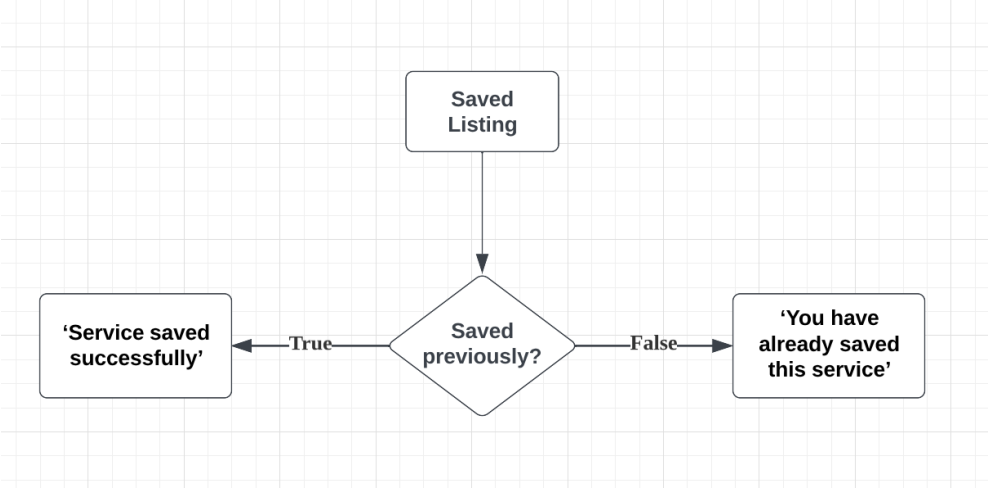
4.2.4 Functionality: Create Listing



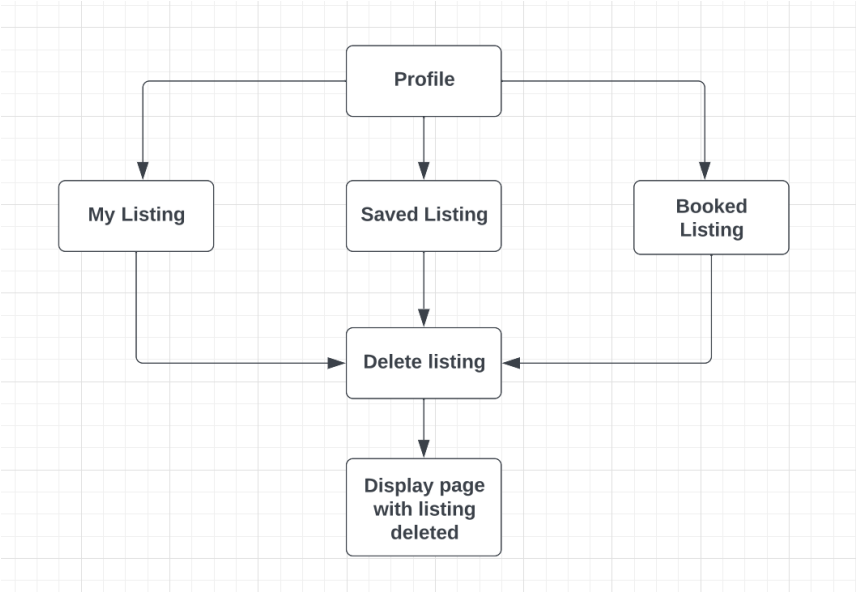
4.2.5 Functionality: Edit listing



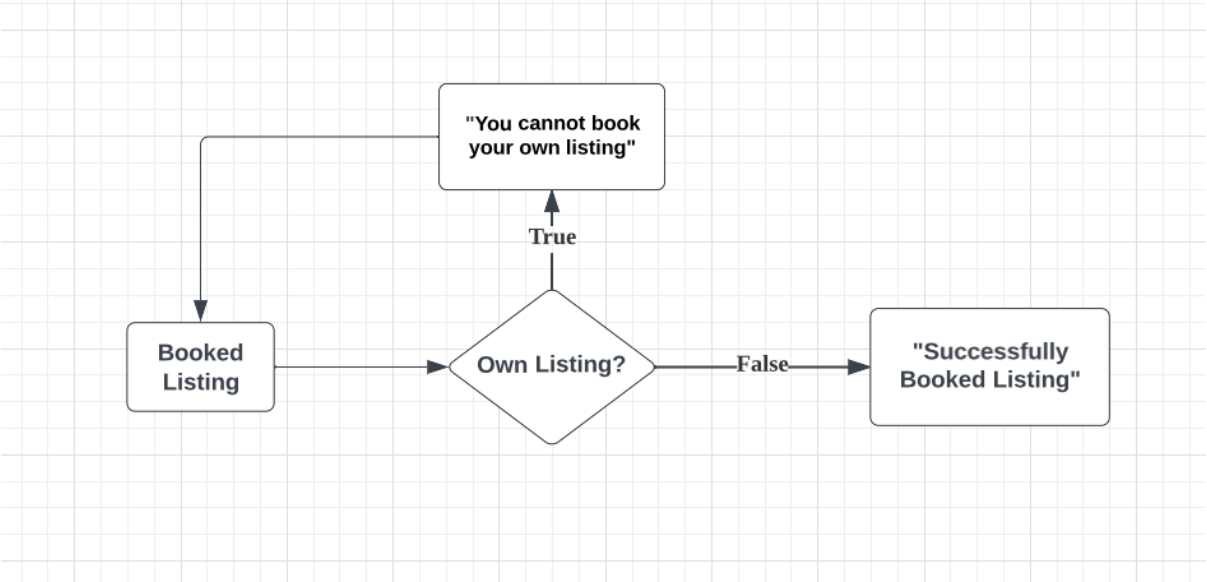
4.2.6 Functionality: Save a listing



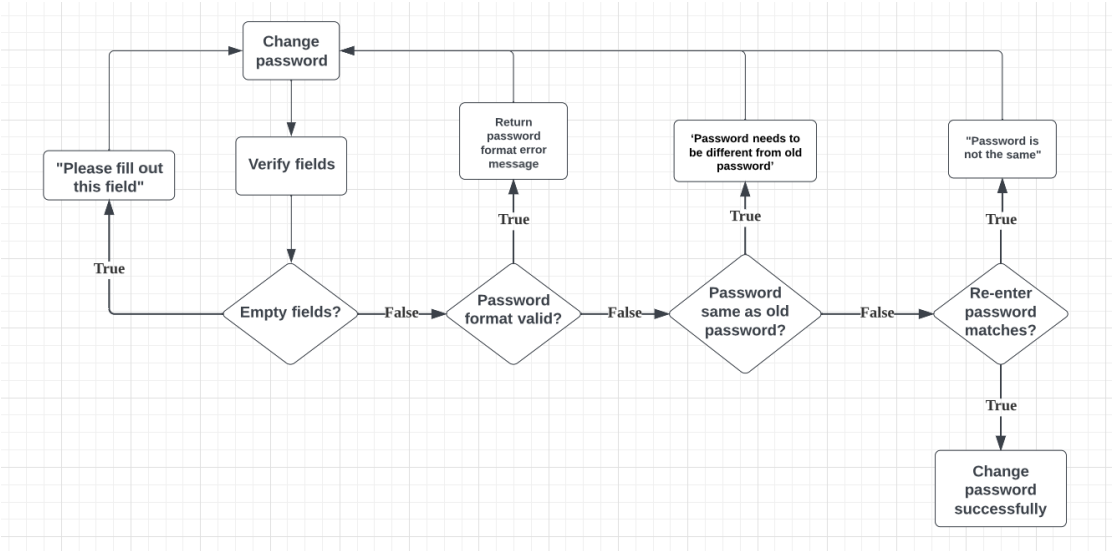
4.2.7 Functionality: Delete my / booked / saved listings



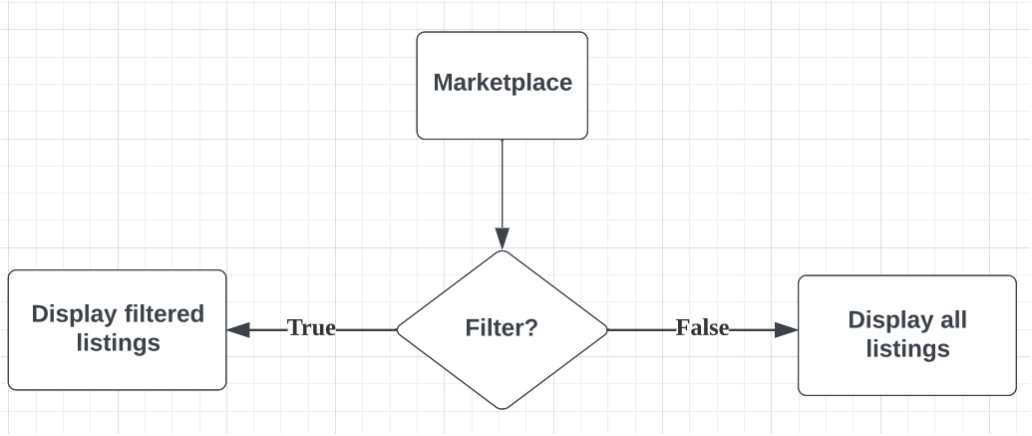
4.2.8 Functionality: Book listing



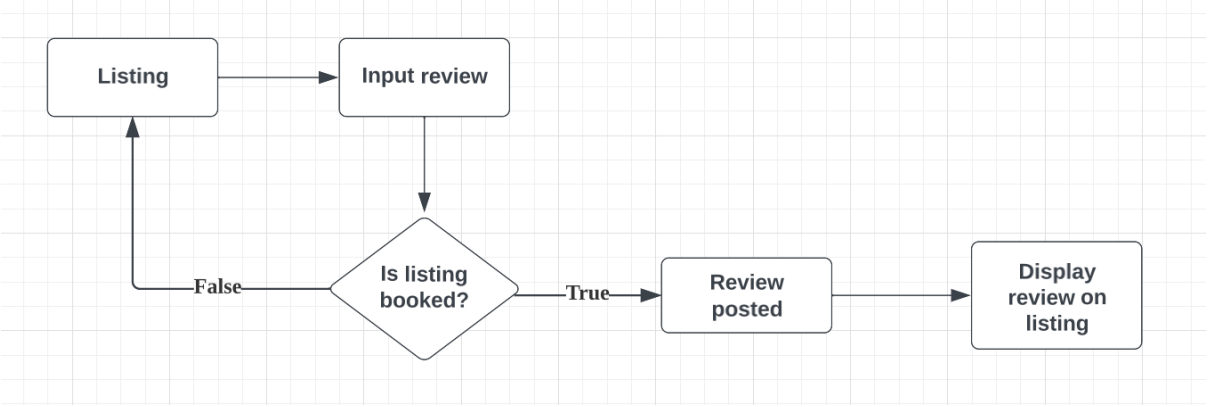
4.2.9 Functionality: Change Password



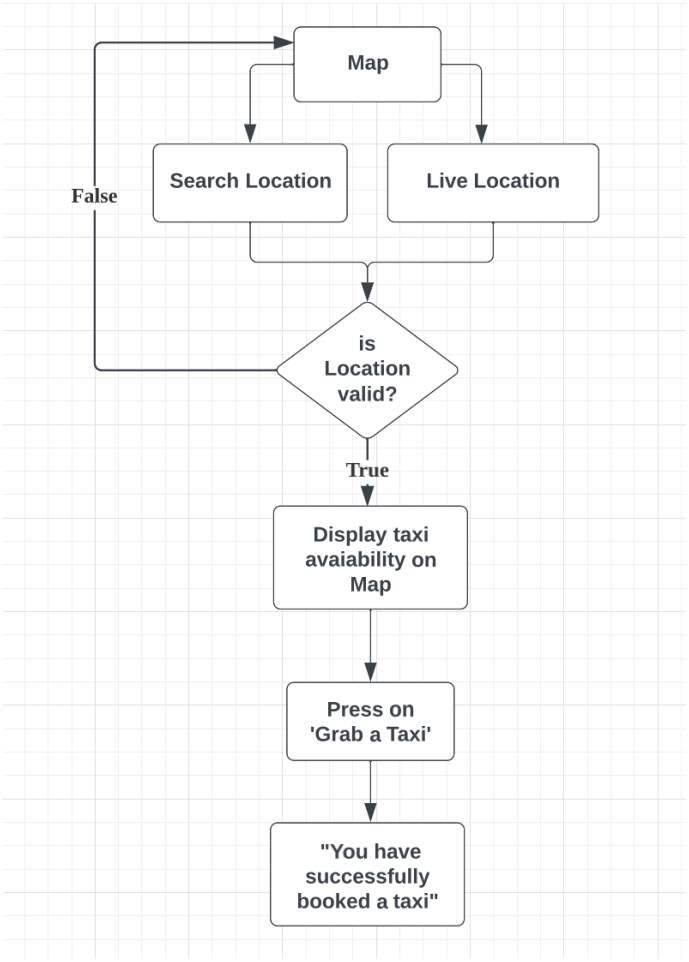
4.2.10 Functionality: Search & select listing



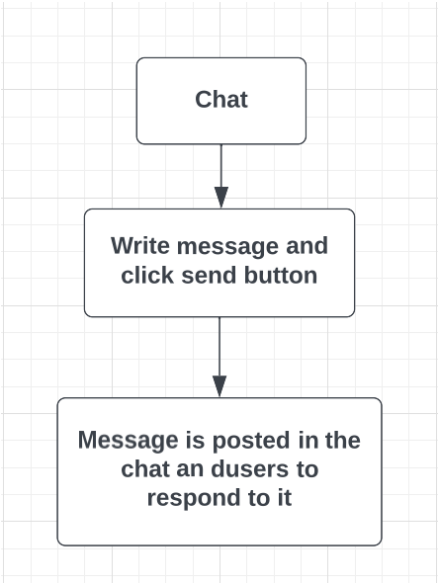
4.2.11 Functionality: Review



4.2.12 Functionality: Taxi Availability



4.2.13 Functionality: Chat



5 References

Bruegge, & Dutoit, A. H. (2010). Object-oriented software engineering : using UML, patterns, and Java (3rd ed.). Prentice Hall.

M. Fowler (2003) UML Distilled: A Brief Guide to the Standard Object Modeling Language 3rd Edition

Robert C. Martin (2009) Clean Code: A Handbook of Agile Software Craftsmanship

Sommerville, I. (2016). Software Engineering. 10th Edition, Pearson Education Limited, Boston.

S. Zhiqi, Prof. Liu Yang (2023) Nanyang Technological University SC2006 Software Engineering