Final Report and Documentation - Team Jimmy

Project Summary:

Jimmy - The Gym Buddy Finder App is designed to address the challenges college students face in finding motivated and compatible workout partners. Many students struggle to maintain a consistent fitness routine due to a lack of accountability and companionship. Jimmy connects students who share similar workout interests and fitness goals, making it easier to find gym partners, join sports teams, and participate in group activities. By providing a user-friendly platform where students can create profiles, indicate their workout preferences, and browse potential partners within the campus community, Jimmy fosters a supportive fitness environment.

The app ensures a trustworthy and exclusive environment by requiring users to sign up with a university email. Key functionalities include profile creation, fitness preference matching, messaging capabilities, and notification systems to coordinate workouts and interactions. Jimmy not only facilitates individual connections but also supports sports teams and fitness clubs in recruiting members, thereby enhancing student engagement and promoting overall well-being on campus. Throughout the development process, we focused on creating a secure, intuitive, and engaging platform that meets the needs of college students seeking workout partners. By considering user feedback and continuously refining features, Jimmy aims to foster a supportive fitness community within college campuses.

Description of all user stories:

1. **Authentication and User Onboarding**

- Google Authentication Integration:

  Implemented secure login using Google Authentication to simplify the sign-in process for users. Ensured that users can log in using their university email, maintaining an exclusive community.

- User Registration and Profile Setup:

  Created a streamlined process for new users to set up their profiles upon first login, capturing essential personal information and fitness preferences.

Feature: Front-End Login Page
As a user intending to log in to the application,
I want to see a button that allows me to sign in with my Google account,
So that I can easily initiate the login process using Google Authentication.
Scenario 1: Enter the login page
Given: I am on the login page
When: I enter the login page
Then: I should see the button labeled "Sign in with Google"

Scenario 2: Successfully Login to a well-configured account
Given: I am on the login page
When: I press the "Sign in with Google" button
Then: I should be directed to Google Authentication
And: I should be redirected to the Dashboard page.

Scenario 3: Login Failed
Given: I am on the login page
When: I press the "Sign in with Google" button
Then: I should be directed to Google Authentication
And: If the login fails, I should see a message "Login failed, please try again"
And: I should be redirected back to the login page

Scenario 4: Successfully login to a non-configured account
Given: I am on the login page
When: I press the "Sign in with Google" button
Then: I should be directed to Google Authentication
And: I should be redirected to the Initial Profile Setup page.

Feature: Integration with 3rd-Party Authentication
As a developer,
I want to integrate Google Authentication with the login system,
So that users can securely log in using their Google account credentials.

Scenario 1: Redirect to Google Authentication.
Given: The user presses "Sign in with Google".
When: The request is sent to Google Authentication.
Then: The user should be redirected to the Google Authentication page.

Scenario 2: Login Failed.
Given: The user is on the Google Authentication page.
When: The authentication fails.
Then: The user should be redirected back to the login page with a failure message.

2. **Dashboard and Navigation**

- Dashboard Interface:

  Developed a user-friendly dashboard that introduces key features and allows easy navigation to profile management and other functionalities.

- Mobile Responsive Design:

  Ensured that the user interface was mobile-responsive, providing an optimal experience across various devices.

Feature: Dashboard Navigation
As a user
So I can understand the available features and manage my profile
I want to see feature introductions and navigate to the user profile management page

Scenario: View feature introductions and navigate to User Profile Management
Given: I am on the dashboard page,
When: I enter the dashboard for the first time,
Then: I should see introductions for each feature displayed on the screen,
When: I click the "User Profile" icon,
Then: I should be navigated to the User Profile Management page.

Feature: Enhanced Dashboard Design for User Landing Page
As a user,
I want an aesthetically pleasing and functional dashboard,
So that I can easily access key features while enjoying a visually appealing interface.

Scenario 1: Implement a Visually Appealing Dashboard Layout
Given: The user logs into the application and lands on the dashboard.
When: The dashboard loads with an optimized layout.
Then: The dashboard should have a clean, organized design with distinct sections for recent activity, profile highlights, and navigation, enhancing readability and accessibility.

Scenario 2: Use Modern Color Scheme and Typography
Given: The user views the dashboard after login.
When: The design includes a new color scheme and typography.
Then: The colors should be visually appealing yet easy on the eyes, with a consistent font style that is legible on both desktop and mobile devices.

Scenario 3: Make the Dashboard Mobile-Responsive
Given: The user opens the dashboard on a mobile device.
When: The dashboard loads in a mobile-friendly format.
Then: The layout, buttons, and widgets should adapt to mobile screens, keeping functionality and aesthetics consistent across all devices.

3. **User Profile Management**

- Personal Information Management:

  Allowed users to update personal information such as profile photo, name, gender, and age.

- Fitness Profile Creation:

  Enabled users to create and manage their fitness profiles, including activity preferences, workout days, location, and desired workout partner characteristics.

Feature: User Profile Management
As a user
So I can update my personal information
I want to be able to change my photo, modify my name, select my gender, and set my age on the profile management page

Scenario: Edit user profile details
Given: I am on the User Profile Management page,
When: I want to update my profile,
Then: I should be able to upload and change my profile photo,
And: I should be able to modify my name,
And: I should be able to select my gender from a dropdown menu,
And: I should be able to set or update my age using a date picker,
And: I should be able to save these changes,
And: I should see a confirmation message when the updates are successfully saved.

Feature: Set Up a Database for Storing User Profiles.
As a software developer,
So as to store user details,
I want to design and set up the database schema for user profiles.

Scenario: Create tables for user profile
Given: the need to store user profiles,
When: setting up the database,
Then: the system should create a user profile table to store user-related information.

Feature: Integrate Frontend with Backend for User Registration
As a software developer,
So as to allow the frontend to send user registration details,
I want to expose functions for user registration that the frontend can call.

Scenario: Frontend sends user registration data to the backend
Given: a user fills in their registration details on the frontend,
When: the user submits the form,
Then: the frontend should call the backend function,
And: the backend should store the user information in the database.

Feature: User Fitness Profile Creation and Management
As a user,
I want to create a fitness profile that includes my fitness-related information and preferences,
So that I can connect with workout partners and find activities that suit my interests and availability.

Scenario 1: Create a Fitness Profile
Given: The user is logged into the application.
When: The user navigates to the fitness profile creation page.
Then: The user should be able to enter personal information such as age, fitness goals, and experience level.

Scenario 2: Specify Preferences for different activities, partner age groups, locations, timings, etc.
Given: The user is creating their fitness profile.
When: The user selects their preferred activities (e.g., running, cycling, yoga), location, timings, specific age group of partners, etc.
Then: The selected preferences should be saved as part of their fitness profile.

Scenario 3: Review and Edit Fitness Profile
Given: The user has created their fitness profile.
When: The user navigates to their profile page.
Then: The user should be able to view their fitness information and preferences, and have the option to edit them as needed.

Feature: Fitness Profile Activity Preferences
As a user,
I want to manage my fitness profile by adding my activity preferences,
So that I can find workout partners that match my specific fitness goals and interests.

Scenario 1: Add Preferred Activities to Fitness Profile
Given: The user is on the fitness profile settings page.
When: The user selects preferred workout activities such as weightlifting, running, yoga, etc.
Then: The preferences should be saved and reflected in the user's fitness profile.

Scenario 2: Set Fitness Buddy Preferences
Given: The user wants to specify their preferences for workout partners.
When: The user selects options such as desired workout partner experience level, location, availability times, or preferred workout intensity.
Then: These preferences should be saved and used to match with potential gym buddies.

Scenario 3: Update or Remove Activity Preferences
Given: The user wants to update or remove existing activity preferences.
When: The user modifies their choices for fitness activities or buddy preferences.
Then: The updated preferences should replace the old ones and reflect on the profile accordingly.

Feature: Fitness Profile Management Bug Fixes
As a user,
I want a smooth experience on the fitness profile management page,
So that I can update my profile details without issues or interruptions.

Scenario 1: Fix Intermittent Loading Issue of Workout Days Input Box
Given: The user is on the fitness profile management page.
When: The page loads and the workout days input box appears.
Then: The input box should load consistently every time the page is accessed, without any intermittent failures.
Scenario 2: Close Dropdown Options When Clicking Outside
Given: The user has opened a dropdown on the fitness profile management page.
When: The user clicks anywhere outside the dropdown area.
Then: The dropdown should close automatically to provide a seamless and intuitive user experience.

4. **Profile Matching and Buddy Finder**

- Profile Matching Algorithm:

  Implemented a backend algorithm to match users based on fitness preferences, activity types, location, and experience levels.

- Profile Matching UI:

  Developed an intuitive interface for users to browse and interact with potential gym buddies, including filters and sorting options.

- Swipe Functionality:

  Introduced swipe gestures to allow users to express interest in or skip other profiles, similar to popular social matching apps.

Feature: UI Template for Matching with Gym Buddies
As a user,
I want to browse profiles based on workout type, location, and experience level,
So that I can find a gym buddy who fits my preferences.

Scenario 1: Display Search Filters
Given: The user is on the gym buddy search page,
When: The page loads,
Then: The user should see filters for workout type, location, and experience level to refine their search.
Scenario 2: Swipe to Match/UnMatch
Given: The user is viewing gym buddy profiles,
When: The user swipes on a profile,
Then: The system should record their interest in that profile for potential matching or unmatch it based on swipe direction.

Feature: Profile Matching Algorithm and Database Integration
As a developer,
I want to implement a backend that connects to the actual database and maintains profile matching data for specific users,
So that the system can display relevant user profiles based on specific matching, sorting, and filtering criteria for each user.

Scenario 1: Implement Profile Matching Algorithm
Given: Each user has fitness preferences and a history of interactions (matched, skipped, or blocked profiles).
When: The user browses the profiles.
Then: The backend should implement an algorithm that:

- Sorts and filters profiles based on activity preferences, location, workout timings, and experience level.
- Excludes profiles that have been blocked or already matched.
- Prioritizes profiles that are a closer match to the user's criteria.

Scenario 2: Maintain User Interaction Records
Given: The system needs to track the user's interactions with other profiles.
When: The user skips, blocks, or matches with a profile.
Then: The backend should store this interaction data in the database, ensuring that the same profiles are not shown again unnecessarily in the future.

Scenario 3: Query the Database for Relevant Profiles
Given: The backend needs to fetch profiles from the database.
When: The system queries the database for relevant user profiles.
Then: The profiles should be filtered and sorted based on user preferences and past interactions using a dynamic algorithm that adapts to user behavior.

Scenario 4: Handle Skipped, Blocked, and Matched Profiles
Given: The backend needs to manage and respect user choices for skipped, blocked, and matched profiles.
When: The user performs any of these actions on profiles.
Then: The corresponding records should be updated, and the backend will exclude these profiles from future queries unless preferences change.

Feature: Profile Matching UI with User Preferences and Filters
As a developer,
I want to connect the mock UI for the profile matching feature to the implemented backend and add functionality for users to select preferences and filters,
So that the system can display relevant user profiles based on dynamic criteria and preferences.

Scenario 1: Connect Mock UI to Backend for Displaying Profiles
Given: The user interacts with the profile-matching UI (e.g., browsing/swiping profiles).
When: The user opens the matching interface.
Then: The system should query the backend database and display actual user profiles that match the user's preferences and filters.

Scenario 2: Enable User to Select Preferences for Matching
Given: The user wants to refine the profiles they see based on specific fitness preferences.
When: The user selects filters such as workout type, location, experience level, and available times.
Then: The system should dynamically adjust the displayed profiles, only showing profiles that match the selected criteria.

Scenario 3: Save User's Preferences and Apply Filters in Backend Queries
Given: The user has set specific preferences and filters.
When: The backend queries the database for user profiles.
Then: The backend should apply these preferences and filters, ensuring that only the most relevant profiles are returned for display.

Scenario 4: Update Displayed Profiles Based on User Interactions
Given: The user interacts with displayed profiles by skipping, blocking, or matching.
When: The user takes an action on a profile.
Then: The backend should update the user's interaction history, and the displayed profiles should reflect this, excluding skipped or blocked users in future matches.
Scenario 5: Implement Real-Time Filtering of Profiles
Given: The user wants to update their preferences in real time.
When: The user modifies filters or preferences (such as changing location or workout type).
Then: The system should re-query the backend and immediately update the list of displayed profiles without requiring a page refresh.

Feature: User Matching and Interaction Options
As a user,
I want to view and interact with matched users,
So that I can connect with workout partners who align with my preferences and easily manage my interactions with them.

Scenario 1: View All Matched Users
Given: The user has set their fitness preferences.
When: The user navigates to the "Matched Users" page.
Then: A list of all users who match their preferences should be displayed, with relevant details such as name, shared interests, and location.

Scenario 2: Search for a Specific User
Given: The user wants to find a specific workout partner from their matched users.
When: The user types the name or a keyword related to the user's profile in the search bar on the "Matched Users" page.
Then: The matched user list should filter to show only users that match the search query.

Scenario 3: View Complete Profile of a Matched User
Given: The user is viewing the list of matched users.
When: The user clicks on a matched user's profile link.
Then: The complete profile of the selected user should be displayed, showing detailed information such as activity preferences, experience level, and availability.

Scenario 4: Block a Matched User
Given: The user wants to manage their interactions with a matched user.
When: The user clicks the "Block" button on a matched user's profile.
Then: The selected user should be blocked, removing them from the matched users list and preventing future interaction.

Scenario 5: Open Chat with a Matched User
Given: The user wants to initiate a conversation with a matched workout partner.
When: The user clicks the "Chat" button on the matched user's profile.
Then: A chat window should open, allowing the user to start a conversation with the matched user in real-time.

Feature: Bug Fixes for Profile Matching Page
As a user,
I want a consistent and error-free experience on the profile matching page,
So that I can view and interact with matched profiles without interruptions or issues.
Scenario 1: Display “No Available Profiles” Message When No Matches Exist
Given: There are no profiles that match the user’s preferences.
When: The user opens the profile matching page.
Then: A message should appear saying, “No available profiles,” clearly indicating that no matches are found.

Scenario 2: Prevent Reappearance of Last Matched Profile
Given: The user has matched with the last available profile.
When: The last profile is processed and there are no more matches.
Then: The last profile should not reappear, and the message “No available profiles” should be displayed instead.

Scenario 3: Restrict Profile Matching Before Completing Fitness Profile
Given: The user has not completed their fitness profile.
When: The user attempts to access the profile matching page.
Then: They should be redirected to complete their fitness profile first, preventing access to profile matching until it is finished.

Scenario 4: Add a Confirmation Modal Box for Blocking Users
Given: The user wants to block another user from their matches.
When: The user clicks the “Block” button on a matched profile.
Then: A custom modal box should appear, asking for confirmation before blocking, ensuring that users don’t accidentally block someone.

Scenario 5: Display User Photos in Profile Matching Cards
Given: The user is viewing matched profiles on the profile matching page.
When: The user photos should load for each matched profile card.
Then: The profile cards should display each user's photo, making the profiles more visually informative.

Scenario 6: Fix Top Navbar Padding for Mobile View
Given: The user is accessing the profile matching page on a mobile device.
When: The top navbar loads in mobile view.
Then: The navbar padding should be consistent with other screens, improving the visual consistency of the application on mobile.

5. **Communication Features**

- Notification System:

  Created a notification system to alert users when they have a new match, with the ability to view and manage notifications within the app.

- Private Chat Feature:

  Integrated a real-time chat functionality that allows matched users to communicate directly, coordinating workouts and fostering connections.

Feature: Notification Triggering on Match
As a developer,
I want to implement notification-triggering logic in the backend,
So that whenever two users match with each other, both users are notified of the match.

Scenario 1: Trigger Notification When Users Match
Given: Two users swipe right or express interest in each other.
When: Both users' preferences align, resulting in a match.
Then: The backend should trigger notifications to both users, informing them of the successful match.

Scenario 2: Store Notification Details in Database
Given: A match occurs between two users.
When: The notification is triggered.
Then: The notification details (e.g., match date, user IDs) should be stored in the database for future reference and tracking.

Scenario 3: Ensure Notifications Are Sent to Both Users
Given: A match is confirmed between two users.
When: The backend processes the match.
Then: Notifications should be sent to both users simultaneously, ensuring that both are informed of the match at the same time.

Feature: Notification Dialog Box in UI for Match Notifications
As a user,
I want to see all my match notifications in a dialog box,
So that I can easily track new matches and mark notifications as read.

Scenario 1: Display Notification Dialog Box
Given: The user has match notifications.
When: The user clicks on the notification icon or opens the notifications menu.
Then: A dialog box should appear, displaying all notifications, with unread notifications clearly marked as new.

Scenario 2: Mark Notification as Read
Given: The user has unread notifications.
When: The user clicks or interacts with an individual notification.
Then: The notification should be marked as read, updating its status in the backend and UI so it no longer appears as new.

Scenario 3: Segregate Read and Unread Notifications
Given: The user has both read and unread notifications.
When: The user opens the notification dialog.
Then: Unread notifications should appear at the top of the list and be visually distinguished (e.g., bold or highlighted), while read notifications are listed below or in a separate section.

Scenario 4: Store Read/Unread Status in the Backend
Given: The user marks a notification as read.
When: The action is taken.
Then: The read/unread status should be updated and stored in the backend, so the notification's state persists across sessions.

Scenario 5: Handle Empty Notification State
Given: The user has no notifications.
When: The user opens the notification dialog box.
Then: The dialog should display a message like "No new notifications" to indicate there are no available match notifications.

Feature: Fix Notification Modal Box Bugs and Improve UI for Read/Unread Notifications
As a user,
I want a clear and organized notifications modal,
So that I can easily distinguish between read and unread notifications and avoid unnecessary options.

Scenario 1: Remove "Mark as Unread" Button
Given: The user opens the notifications modal box.
When: The user views the available actions for notifications.
Then: The "Mark as Unread" button should be removed, leaving only relevant options to simplify the interface.

Scenario 2: Display Unread Notifications at the Top
Given: The user has both read and unread notifications.
When: The notifications modal box is opened.
Then: Unread notifications should appear at the top of the list, followed by read notifications at the bottom, making it easy to identify new information.

Scenario 3: Apply Bright Design to New (Unread) Notifications
Given: The user opens the notifications modal and views unread notifications.
When: Unread notifications are displayed.
Then: Unread notifications should have a bright design (e.g., bold text, highlighted background) to visually distinguish them as new, ensuring they capture the user's attention.

Feature: Private Chat UI Interface
As a user,
I want to have a private chat interface with another user,
So that I can communicate with my matched gym buddies directly through the app.

Scenario 1: Display Chat Interface Between Two Users
Given: The user has matched with another user.
When: The user clicks on the "Chat" button for the match.
Then: A private chat window or section should open, displaying an empty message field and previous conversation history (if any).

Scenario 2: Send and Display Messages in the Mock UI
Given: The user types a message into the chat input field.
When: The user presses "Send."
Then: The message should be displayed in the chat window on the user's side, and a corresponding "received" message box should appear on the other side (mocked for UI purposes).

Scenario 3: Display Chat History (Mocked Data)
Given: The user opens the chat interface with a matched user.
When: There are previously sent or received messages.
Then: The chat window should display the message history in a scrollable format, with the sender and receiver's messages aligned on opposite sides of the window.

Scenario 4: Handle Empty Chat State
Given: The user opens the chat window for the first time.
When: There is no message history between the users.
Then: The chat window should display a placeholder message, such as "Start the conversation!" or a blank space for the chat input.

Scenario 5: Responsive Design for Chat UI
Given: The user is accessing the app on different devices.
When: The user opens the chat interface on a mobile or tablet.
Then: The chat UI should adapt to the screen size, ensuring the input field and chat window are easy to use on both mobile and desktop devices.

Feature: Chat Functionality Backend and Database Setup
As a user,
I want a reliable chat feature that allows me to communicate with workout partners,
So that I can easily coordinate and stay connected with my fitness community.

Scenario 1: Create Chat Feature Backend
Given: The development team is setting up the backend for the chat feature.
When: They implement the necessary API endpoints and service logic to support message sending, receiving, and real-time updates.
Then: The chat feature should facilitate smooth, secure, and responsive communication between users.

Scenario 2: Setup Database Tables for Chat Feature
Given: The development team is configuring the database for chat functionality.
When: They create tables for storing messages, user IDs, timestamps, and chat session details.
Then: The database should securely store chat data and allow for efficient retrieval and querying to support real-time messaging.

Scenario 3: Ensure Chat Feature Works Seamlessly
Given: The chat feature has been implemented and configured with database support.
When: The development team tests message sending, receiving, and real-time delivery between users.
Then: The chat functionality should work without errors, supporting consistent and instant communication.

Scenario 4: Write Cucumber Scenarios and RSpec Tests for Chat Feature Backend
Given: The chat backend and database setup is complete.
When: The development team writer Cucumber scenarios to test chat functionality from a user perspective and RSpec tests to verify API endpoints and database interactions.
Then: The tests should validate that messages can be sent, received, and stored correctly, and all aspects of the chat feature should function as expected.

Feature: Integration of Chat Backend with Frontend UI
As a user,
I want to use a fully functional chat feature with a smooth interface,
So that I can seamlessly communicate with my workout partners without any disruptions.

Scenario 1: Connect Backend with Chat UI
Given: The backend for chat functionality has been developed, and a chat UI exists on the frontend.
When: The development team integrates the backend endpoints with the chat UI for sending and receiving messages.
Then: The chat UI should display messages in real-time, reflecting the backend data accurately.

Scenario 2: Display Real-Time Messages in Chat UI
Given: The user is in a chat session with a workout partner.
When: Messages are sent or received through the chat backend.
Then: The messages should appear instantly in the chat UI, updating in real-time without the need for manual refresh.

Feature: Proof of Concept (POC) for Persistent Chat Feature
As a developer,
I want to carry out a Proof of Concept (POC) to implement a **persistent chat feature** in the Ruby on Rails application,
So that I can evaluate the feasibility of storing and retrieving chat messages between users for long-term use.

Scenario 1: Setting up Basic Chat Functionality
Given: The developer has set up models for users and chat messages,
When: Two users interact with each other through the chat interface,
Then: The system should allow messages to be sent and stored in a database for future retrieval.

Scenario 2: Evaluating Database Schema for Persistent Storage
Given: The chat messages need to be stored for persistence,
When: The developer designs the database schema for storing messages (e.g., sender, receiver, message content, timestamp),
Then: The schema should allow efficient retrieval and querying of past conversations.

Scenario 3: Testing Real-Time Chat Capabilities
Given: The POC involves exploring real-time communication,
When: The developer integrates ActionCable or WebSockets for live updates,
Then: The chat feature should support real-time message exchange between users.

Scenario 4: Exploring Data Retention and Archiving
Given: Users may want to retain chat history for future reference,
When: The POC considers data retention policies and archiving strategies,
Then: The system should offer options for retaining or archiving old conversations.

6. **Testing and Quality Assurance**

- Mock Testing Framework:

  Established a testing framework for authentication modules and other key functionalities to ensure robustness without relying on external services during testing.

- CI/CD Pipeline Implementation:

  Set up a continuous integration and continuous deployment pipeline to automate building and testing of code on every pull request, ensuring code quality before merging.

Feature: Implement a mock testing framework for Authentication Modules
As a developer,
I want to set up tests for third-party authentication and user session management,
So that I can validate authentication flows without relying on external services during testing.

Scenario 1: Mock Third-Party Authentication for Testing
Given: The user triggers a third-party authentication process (e.g., "Login with Google").
When: The authentication system is in test mode.
Then: A mock response should be used to simulate successful authentication.

Scenario 2: Validate User Authentication in Tests
Given: The application uses an authentication framework (e.g., Devise) for user sessions.
When: Unit tests are executed for actions requiring user login.
Then: A test helper should simulate a logged-in user without invoking the actual authentication process.

Scenario 3: Handle Failed Authentication in Tests
Given: A third-party authentication process is initiated.
When: A mock failure response is returned (e.g., invalid credentials).
Then: The application should correctly handle the failure and provide appropriate feedback in the test environment.

Scenario 4: Test User Session Persistence for Authenticated Users
Given: A user is authenticated via a third-party service or internal authentication.
When: The user navigates to restricted areas of the application.
Then: The user’s session should persist, allowing access to those areas.

Feature: Test Integration for Dashboard and User Profile Management
As a developer,
I want to integrate the mock testing framework for dashboard and profile management pages,
So that I can validate user access and functionality in these areas without depending on live authentication during testing.

Scenario 1: Test Access to Dashboard and User profile management page for Authenticated Users
Given: A user is authenticated via a third-party service or internal authentication.
When: The user tries to access the dashboard and the User profile management page.
Then: The mock testing framework should validate that the user has access and display the dashboard and the user profile management page content.

Scenario 2: Test Access Denial for Unauthenticated Users on Dashboard and User profile management page
Given: A user is not logged in or has failed authentication.
When: The user tries to access the dashboard and User profile management page.
Then: The application should prevent access and redirect the user to the login page, as simulated in the test environment.

Scenario 3: Test Profile Update Functionality
Given: A user is authenticated and accessing the profile management page.
When: The user updates profile information.
Then: The mock testing framework should simulate a successful update of the user's profile and reflect the changes.

Scenario 4: Test Profile Button Navigation from Dashboard to Profile Management
Given: A user is authenticated and viewing the dashboard.
When: The user clicks the profile button on the dashboard.
Then: The mock testing framework should simulate a successful redirection to the profile management page.

Scenario 5: Test Profile Information Display on Profile Management Page
Given: A user is authenticated and navigates to the profile management page.
When: The user views their profile on the profile management page.
Then: The mock testing framework should verify that the profile management page displays the user's name, age, and other relevant information.

Feature: Create PR-Build CI/CD Pipeline for Automated Building and Testing of Code
As an software engineer,
So as to ensure code quality before merging into the main branch,
I want to set up a CI/CD pipeline that automatically builds and tests code on every pull request.

Scenario: Set up PR-triggered CI/CD Pipeline
Given a developer has created a pull request on the repository,
When the pull request is opened or updated,
Then the CI/CD pipeline should automatically trigger a build,
And the pipeline should run all tests,
And the build and test results should be displayed in the pull request status,
And the pull request should not be mergeable if the build or tests fail.

7. **Bug Fixes and UI Enhancements**

- UI and CSS Improvements:

  Addressed various UI issues based on client feedback, including font colors, button styling, and background adjustments to enhance visual appeal and user experience.

- Responsive Design Fixes:

  Resolved CSS bugs and improved mobile responsiveness across different pages, ensuring consistent functionality and appearance on all devices.

Feature: Mobile Responsive User Interface
As a front-end developer,
I want to modify the UI to make it mobile responsive,
So that users can have an optimal viewing and interaction experience across various device sizes, as per the client's requirements.

Scenario 1: Ensure Responsive Layout on Mobile Devices with optimized navigation
Given: The user accesses the application on a mobile device.
When: The screen size is reduced (e.g., below 768px width).
Then: The layout should automatically adjust to a mobile-friendly design (e.g., stacked content).

Scenario 2: Align Client's Requirements for Mobile UI
Given: The client has specific design requirements for the mobile version.
When: The developer implements responsive changes.
Then: The design should meet the client's criteria for look and feel, such as specific color schemes, spacing, and font styles for mobile devices.

Feature: Revamped UI Design for Buddy Finding App
As a UI/UX designer,
I want to create a fresh and engaging UI for the buddy-finding app,
So that users have an enjoyable and intuitive experience while connecting with workout partners.
Scenario 1: Design new UI elements with refined styling and vibrant color scheme.
Given: The app's UI is being updated,
When: A new color scheme is implemented with refined navigation and styling elements,
Then: The UI should be visually appealing, responsive and user-friendly

Scenario 2: Gather User Feedback
Given: The new UI has been implemented,
When: Users interact with the app,
Then: Feedback should be collected to identify areas for improvement.

Feature: CSS Bug Fixes and Mobile Responsiveness for Profile Matching
As a developer,
I want to resolve CSS-related bugs on the profile matching page and improve its appearance on mobile-sized interfaces,
So that the page looks visually appealing and functions properly across different screen sizes, especially mobile devices.

Scenario 1: Fix CSS Bugs on Profile Matching Page
Given: There are existing CSS issues that affect the layout or appearance of the profile-matching page.
When: The developer inspects the page for broken styles or misaligned elements.
Then: The CSS bugs should be fixed to ensure a clean and functional design on desktop and mobile screens.

Scenario 2: Ensure Proper Layout on Mobile Devices
Given: The user views the profile matching page on a mobile device.
When: The page is rendered on smaller screen sizes (e.g., mobile phones).
Then: The layout should automatically adjust to fit the mobile interface without cutting off content or misplacing elements.

Scenario 3: Implement Responsive Design for Profile Cards and Buttons
Given: Profile cards and action buttons (such as swiping or matching) need to adapt to mobile screens.
When: The user views these elements on mobile devices.
Then: The profile cards and buttons should resize and align correctly to ensure easy interaction on smaller screens.

Feature: UI and CSS Enhancements as per client feedback
As a developer,
I want to resolve minor UI and CSS issues based on the client’s feedback,
So that the application looks more visually appealing and meets the client’s design expectations, including improvements to font color, buttons, and background styling.

Scenario 1: Update Font Color as per Client's Feedback
Given: The client has provided feedback regarding the font color on certain pages.
When: The developer reviews and adjusts the font color to match the client's preferences.
Then: The updated font color should enhance readability and align with the overall design aesthetic.

Scenario 2: Adjust Button Styles for Consistency
Given: The client has requested changes to button styles (e.g., size, color, hover effects).
When: The developer modifies the button CSS to reflect the feedback.
Then: The buttons across the application should have a consistent style that improves the user experience and aligns with the design theme.

Scenario 3: Improve Background Styling Based on Client Feedback
Given: The client has requested changes to the background (e.g., color, pattern, or texture).
When: The developer updates the background styling to match the requested changes.
Then: The background should complement the overall UI design and enhance the visual experience for users.

Scenario 4: Ensure Visual Consistency Across All Pages
Given: Multiple pages require UI consistency following the client's feedback.
When: The developer makes adjustments to fonts, buttons, and background elements.
Then: The UI should maintain a cohesive design across all pages, ensuring a seamless user experience.

Feature: Custom Alert Boxes for Action Confirmations
As a user,
I want custom alert boxes for confirmation actions,
So that I can have a clear, visually distinct confirmation prompt before taking important actions.

Scenario 1: Display Custom Confirmation Modal for Sensitive Actions
Given: The user is about to perform an action that requires confirmation, such as deleting a profile or blocking a user.
When: The user clicks the action button (e.g., "Delete" or "Block").
Then: A custom alert box modal should appear, providing a clear and visually distinct

confirmation prompt with “Confirm” and “Cancel” options.

Scenario 2: Include Descriptive Text and Actionable Buttons
Given: The user opens a custom confirmation alert box.
When: The alert box is displayed for actions like deletion or blocking.
Then: The modal should display a descriptive message (e.g., “Are you sure you want to delete this item?”) and two clear buttons: “Confirm” (styled in a distinct color) and “Cancel” (styled in a neutral color), to guide the user’s choice.

Scenario 3: Prevent Background Interactions When Modal is Active
Given: The user opens a custom confirmation alert box.
When: The modal appears on the screen.
Then: The background should be dimmed and interactions with background elements should be disabled until the user makes a choice (either confirming or canceling the action), focusing their attention on the alert box.

Scenario 4: Use Custom Styles for the Alert Box
Given: The user interacts with a confirmation modal.
When: The custom alert box appears on screen.
Then: The alert box should use a visually consistent design with soft shadows, rounded corners, and a color scheme that matches the application’s theme, creating an aesthetically pleasing experience.

Feature: Fix Background Image Visibility for Tall Screens
As a user,
I want the background image to display consistently across all screens,
So that I have a seamless visual experience without empty areas on taller pages.

Scenario 1: Ensure Background Image Stretches to Fit Tall Screens
Given: The user is on a page with content that exceeds the screen height.
When: The page loads or is scrolled past the initial screen height.
Then: The background image should stretch or repeat seamlessly to cover the full height of the page, preventing any areas without the background.

Scenario 2: Set Background Image to Cover Entire Page Height Dynamically
Given: The page content height exceeds the default screen size.
When: The background image is applied to the page.
Then: The CSS should dynamically adjust the background image to use background-size: cover or background-size: contain, with options like

background-repeat: no-repeat or background-repeat: repeat (depending on the design), so the image covers the entire page consistently.

Feature: Hide Navbar Button on Login Page for Mobile View
As a user,
I want a simplified interface on the login page in mobile view,
So that unnecessary elements like the navbar button are hidden when they are not needed.

Scenario 1: Remove Navbar Button on Login Page for Mobile Devices
Given: The user is on the login page using a mobile device.
When: The page loads in mobile view.
Then: The navbar button should be hidden, as the login page does not have additional options, providing a cleaner and less distracting interface.

Scenario 2: Ensure Navbar Button Reappears on Other Pages in Mobile View
Given: The user navigates to other pages in the mobile view.
When: The user leaves the login page and accesses other sections of the app.
Then: The navbar button should reappear to provide navigation options, maintaining usability across other pages.

Feature: Fix User Profile Loading Issues Due to JavaScript and Cache Errors
As a user,
I want my profile to load reliably without interruptions,
So that I can access and update my information without facing loading errors.

Scenario 1: Fix JavaScript Errors Causing Profile Load Failures
Given: The user navigates to their profile page.
When: The profile page loads with JavaScript functions running.
Then: Any JavaScript errors that prevent the page from loading should be identified and resolved, ensuring the profile loads fully and reliably.

8. **Persistent Data Storage**

- User Profile Data Storage:

  Designed and set up a database schema for storing user profiles and fitness preferences securely.

- Persistent File Storage for Profile Pictures:

  Implemented a solution for persistent storage of user-uploaded profile pictures to address issues with ephemeral storage on the hosting platform.

Feature: Fix User Profile Picture Visibility Bug in Production
As a developer,
I want to fix the issue causing the user profile picture to disappear in the deployed version on Heroku,
So that users can consistently see their profile picture without it going missing over time.

Scenario 1: Display Profile Picture Correctly in Production
Given: The user has uploaded a profile picture.
When: The user accesses their profile in the deployed version of the application.
Then: The profile picture should be displayed correctly on the user's profile page without disappearing.

Scenario 2: Implement a Persistent Image Storage Solution
Given: The application is deployed on Heroku with an ephemeral filesystem.
When: The developer looks for solutions to store images.
Then: A reliable external image storage solution (such as Amazon S3, Google Cloud Storage, or Cloudinary) should be selected and implemented to ensure profile pictures persist across dyno restarts.

Scenario 3: Verify Configuration for External Storage
Given: An external storage solution is implemented.
When: The application handles profile picture uploads.
Then: The images should be correctly stored in the external service, and their links should be properly accessible in the application.

9. **Beta Testing and Feedback Integration**

- Beta User Testing Initiation:

  Began beta testing with real users to gather feedback on application features, usability, and overall experience.

- Feedback-Driven Improvements:

  Incorporated user and client feedback into ongoing development, focusing on enhancing core functionalities and user satisfaction.

10. **Additional Features and Considerations**

- Custom Alert Dialogs:

  Implemented custom confirmation dialogs for sensitive actions such as blocking users or deleting profiles to prevent accidental actions.

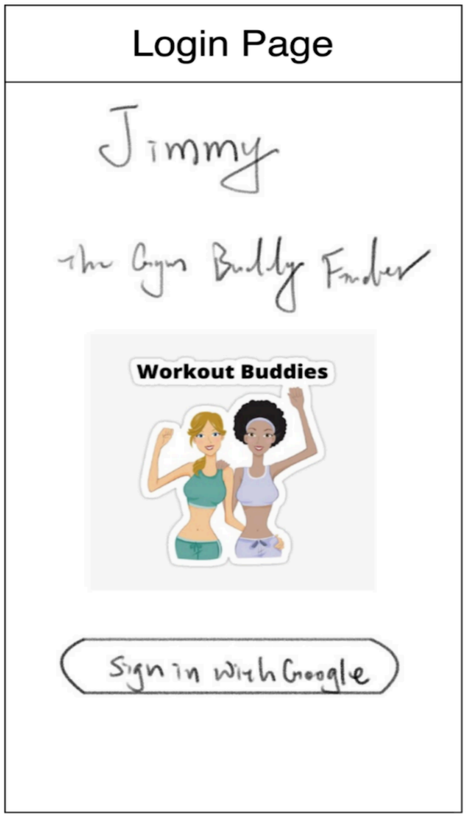
- Enhanced Dashboard Design:

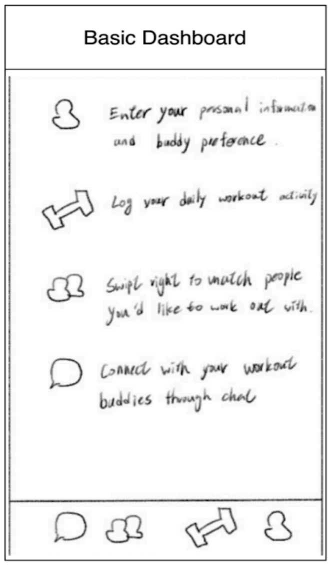
  Redesigned the dashboard to provide a more engaging and informative landing page for users upon login.

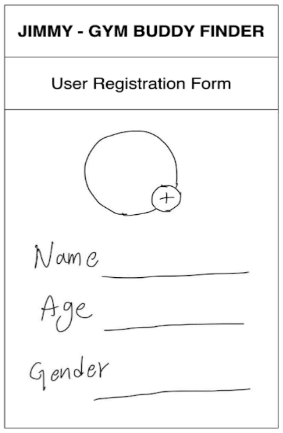
- Accessibility and Error Handling:

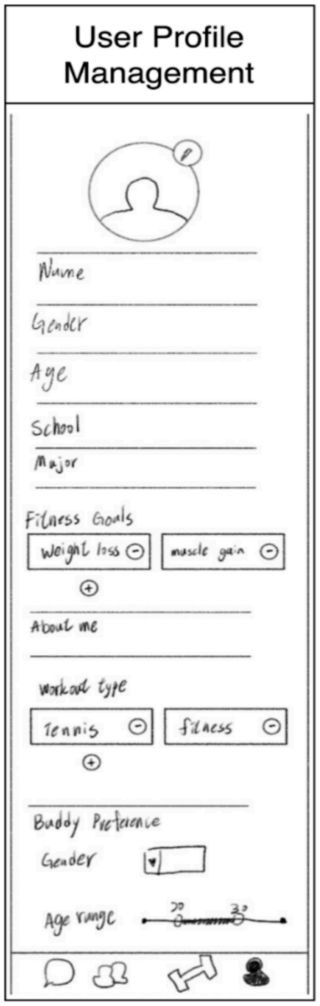
  Ensured that users are guided appropriately if they attempt to access features without completing necessary prerequisites, such as filling out their fitness profile.

User interface: Lo-fi UI mockups and storyboards

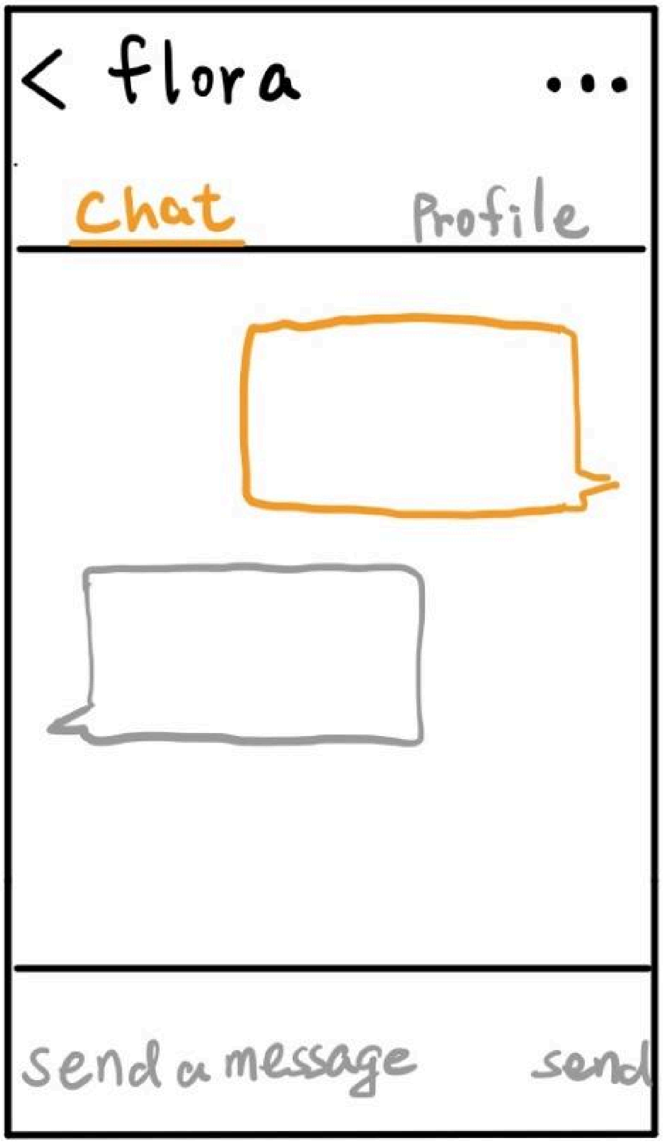



Basic Dashboard
Enter your personal information and buddy preference.
Log your daily workout activity
Swipe right to match people you'd like to work out with.
Connect with your workout buddies through chat

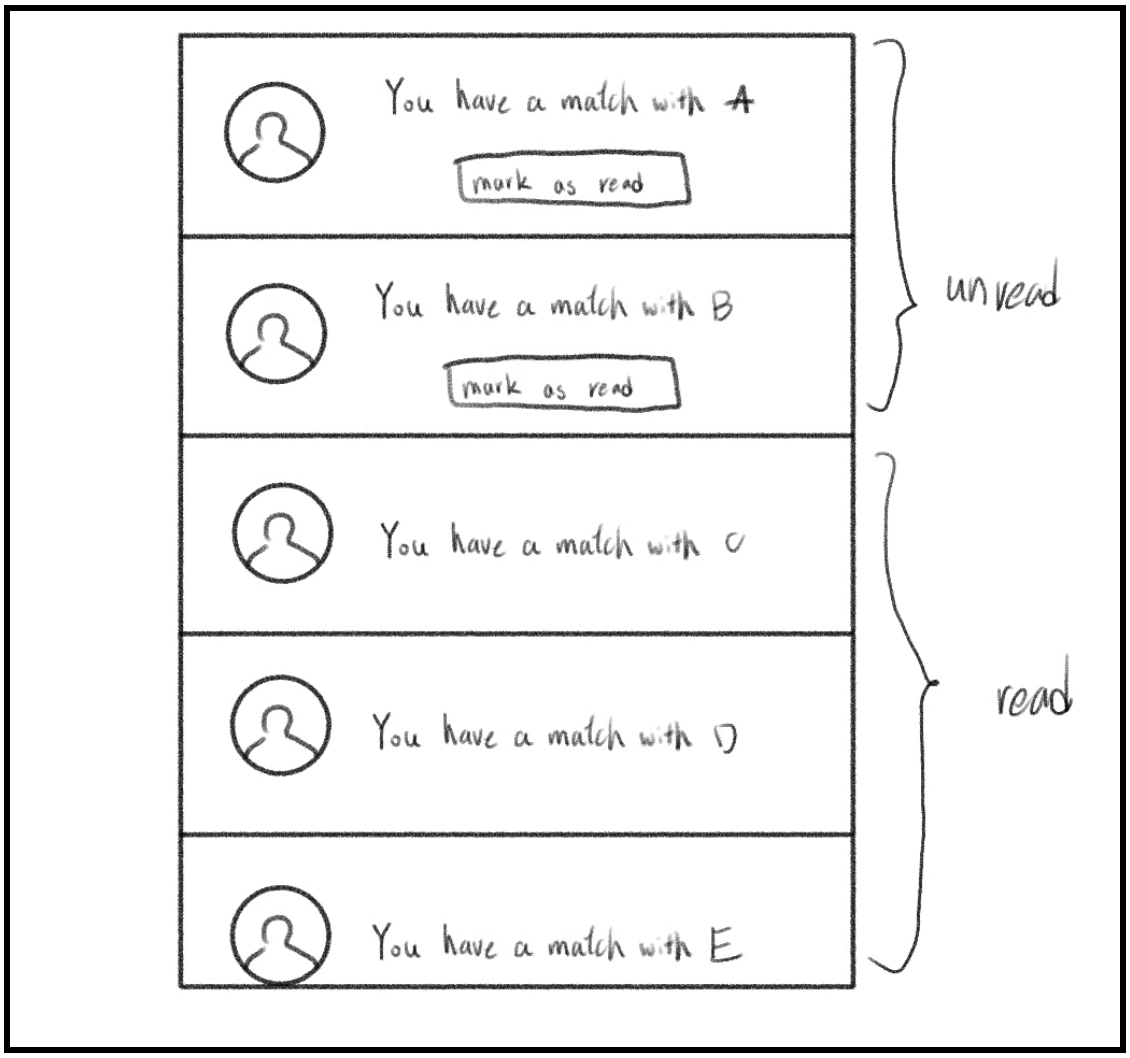




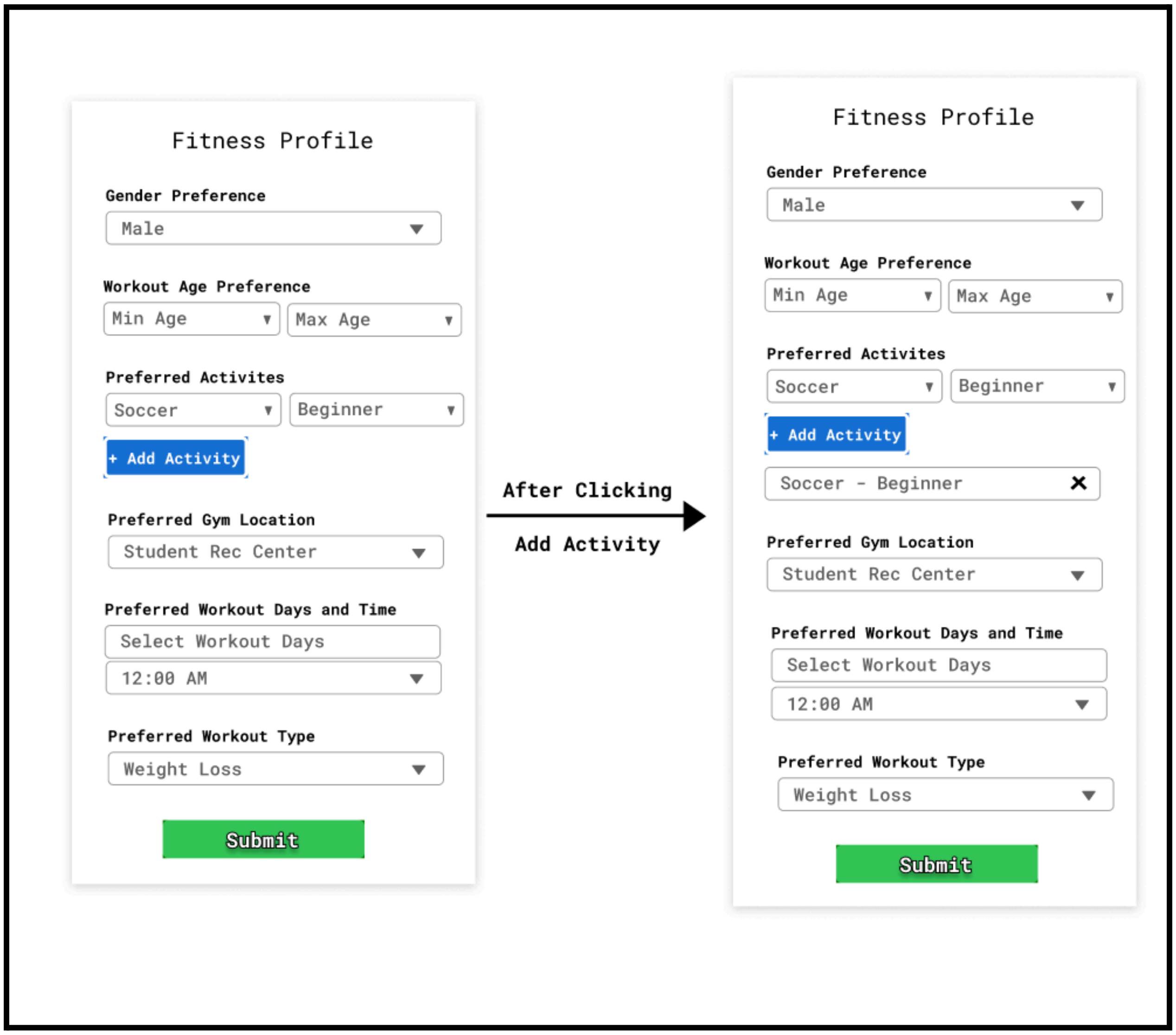
Chat Room UI:



Notifications UI:



Fitness Profile UI:



Fitness Profile

Gender Preference

Male

Workout Age Preference

Min Age | Max Age

Preferred Activites

Soccer | Beginner

+ Add Activity

Preferred Gym Location

Student Rec Center

Preferred Workout Days and Time

Select Workout Days

12:00 AM

Preferred Workout Type

Weight Loss

Submit

After Clicking Add Activity

Fitness Profile

Gender Preference

Male

Workout Age Preference

Min Age | Max Age

Preferred Activites

Soccer | Beginner

+ Add Activity

Soccer - Beginner ✕

Preferred Gym Location

Student Rec Center

Preferred Workout Days and Time

Select Workout Days

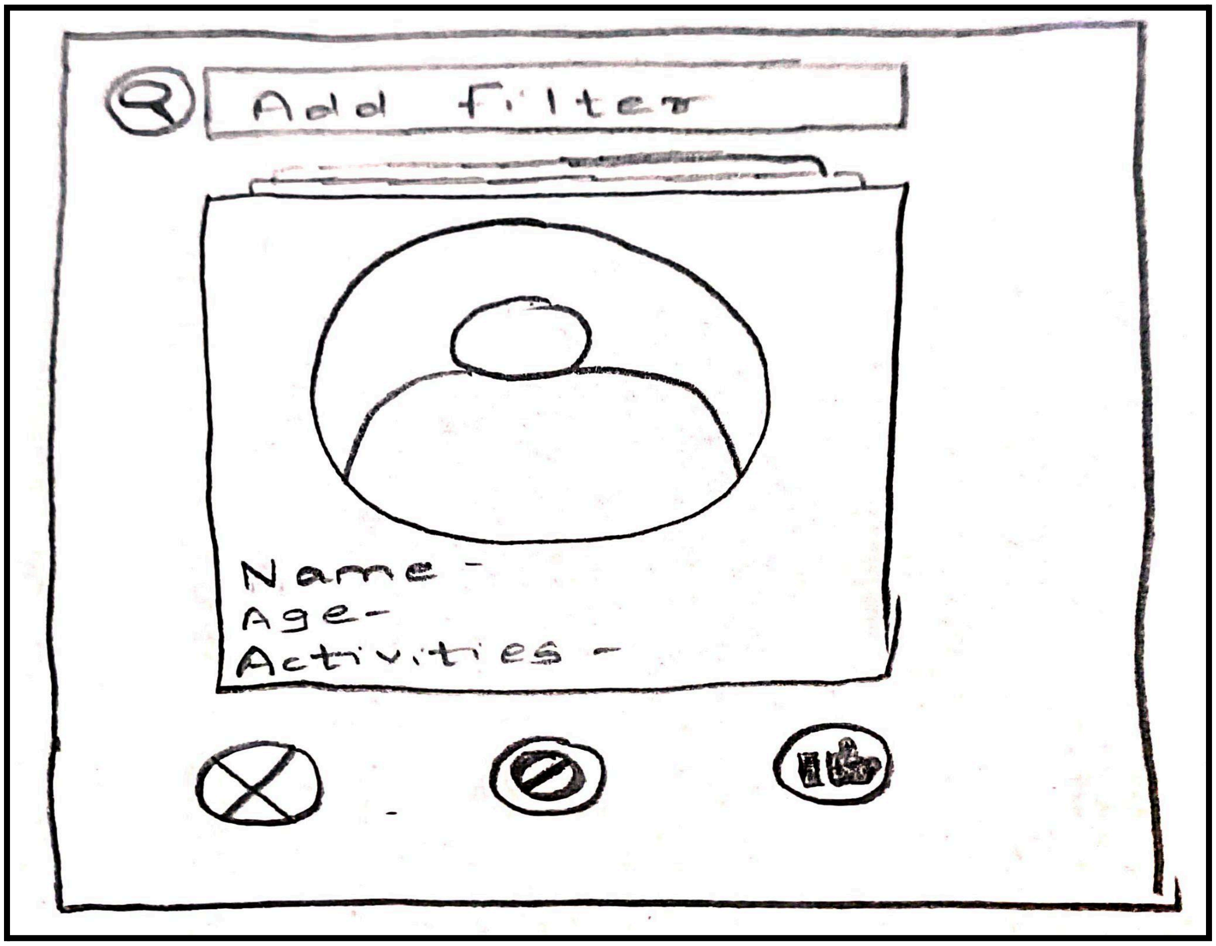
12:00 AM

Preferred Workout Type

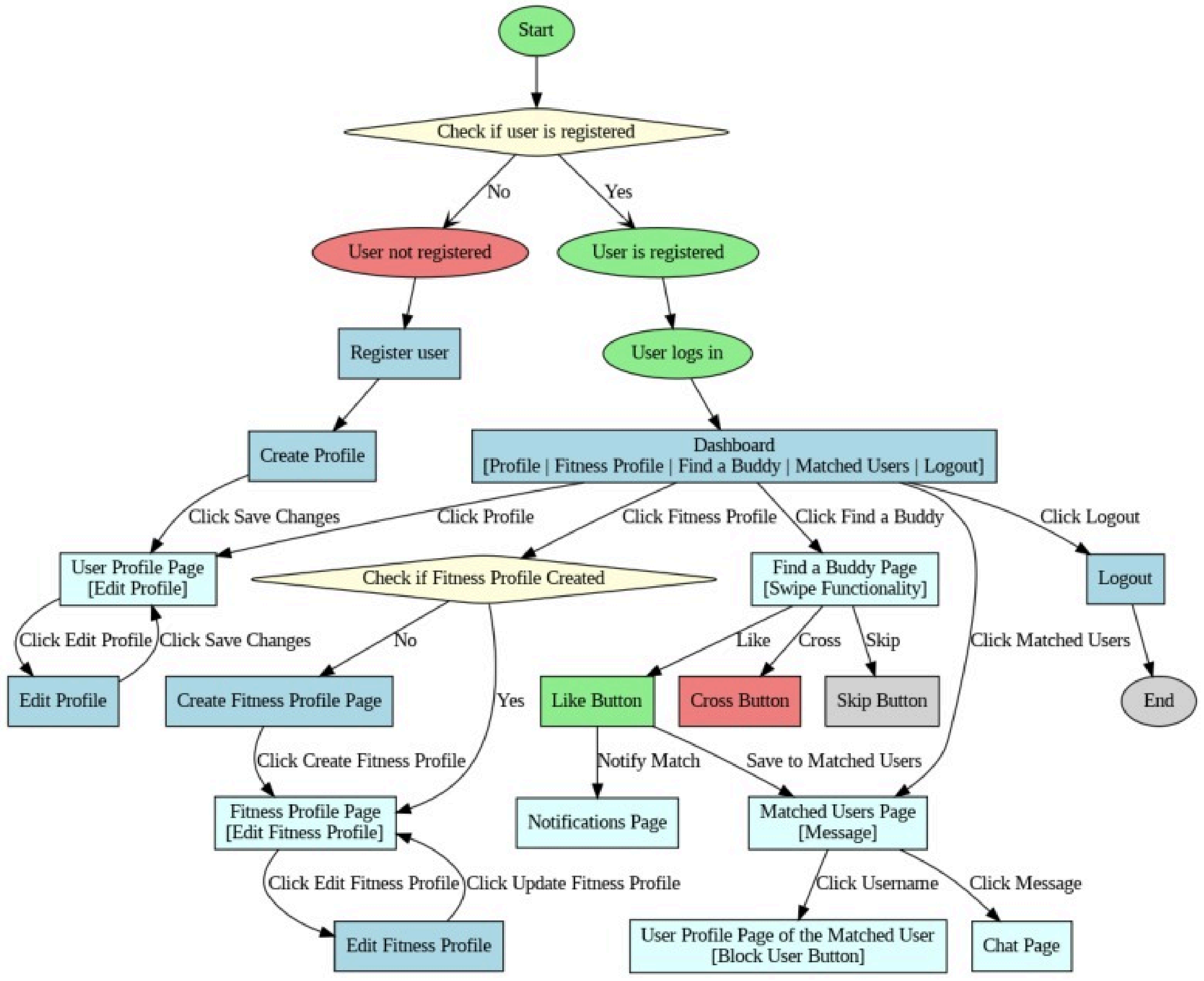
Weight Loss

Submit

Profile Matching UI:

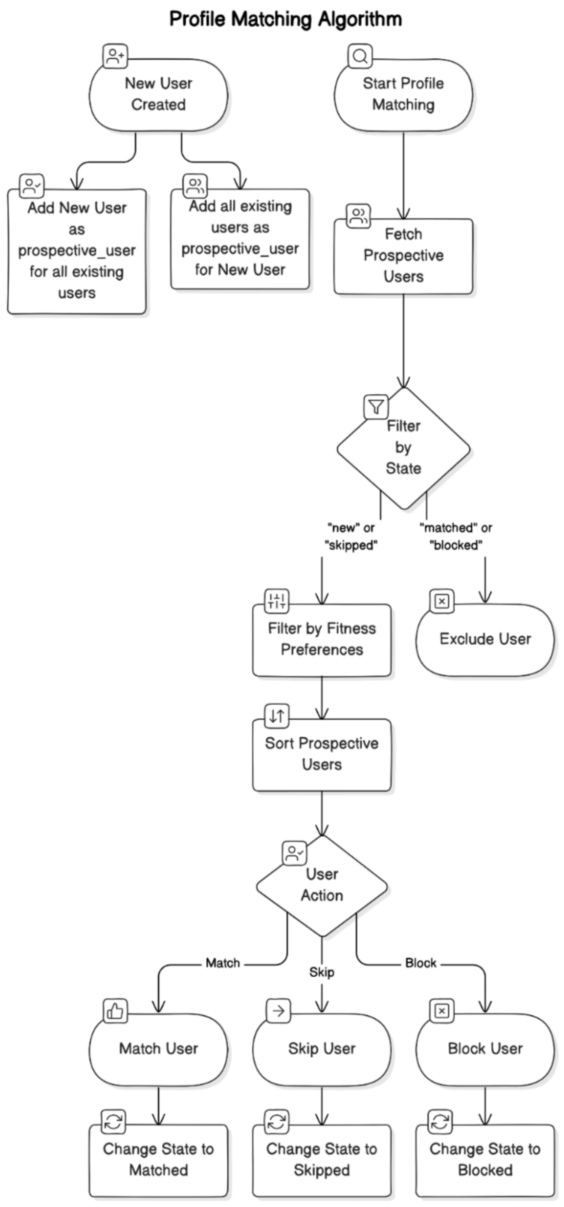


Design Diagram:



Profile Matching Algorithm Design Diagram:

Profile Matching Algorithm Design Diagram:



Team Roles in each iteration:

Iteration 1:

ChuanHsin Wang - Scrum Master
Kushal Lahoti - Product Owner
Yash Phatak - Developer
Wei-Chien Cheng - Developer
Kuan-Ru Huang - Developer
Barry Liu - Developer
Mrunmay Deshmukh - Developer

Iteration 2:

Kushal Lahoti - Scrum Master
Yash Phatak - Product Owner
ChuanHsin Wang - Developer
Wei-Chien Cheng - Developer
Kuan-Ru Huang - Developer
Barry Liu - Developer
Mrunmay Deshmukh - Developer

Iteration 3:

Barry Liu - Scrum Master
Kuan-Ru Huang - Product Owner
Yash Phatak - Developer
Kushal Lahoti - Developer
ChuanHsin Wang - Developer
Wei-Chien Cheng - Developer
Mrunmay Deshmukh - Developer

Iteration 4:

Mrunmay Deshmukh - Scrum Master
Wei-Chien Cheng - Product Owner
Yash Phatak - Developer
Kushal Lahoti - Developer
ChuanHsin Wang - Developer
Kuan-Ru Huang - Developer
Barry Liu - Developer

The team remained fully committed to their roles throughout the sprint, without changing responsibility. Everyone stayed focused on their assigned tasks and contributed consistently.

Summary of each scrum iteration:

Iteration 1:

Summary:

In this sprint, we successfully set up the foundational structure of the app and implemented user registration with third-party login options. We began by developing a basic login page that allows users to sign in using services such as Google. Additionally, we enabled users to create a profile with essential details as requested by the client. We then built a dashboard to display key user information and a profile page for users to view their details. Finally, we set up the backend and database to store user information securely.

1. **Front-End Login Page**
   - A user-friendly login page was developed, featuring a "Sign in with Google" button that allows users to initiate the login process via Google Authentication.
   - Depending on their account configuration, users are either redirected to the dashboard or the profile setup page.
   - The implemented scenarios include displaying the "Sign in with Google" button, a successful login redirecting the user to the dashboard, a failed login showing an error message and returning the user to the login page, and non-configured users being redirected to the profile setup page after login.
2. **Integration with 3rd-Party Authentication**
   - Google Authentication was integrated into the app's login system to provide a secure login experience using Google credentials.
   - If authentication fails, the user is redirected to the login page with an appropriate error message.
   - The implemented scenarios include redirection to Google Authentication and handling failed login attempts.

3. **Dashboard Navigation**
    - A dashboard was developed to introduce the key features of the app and provide easy navigation for users.
    - From the dashboard, users can also access their profile management page. The implemented scenario includes the dashboard displaying feature introductions and enabling navigation to the user profile management page.
4. **User Profile Management**
    - A profile management page was implemented, allowing users to edit their personal information, such as uploading a profile picture, modifying their name, selecting their gender, and updating their age.
    - The implemented scenario enables users to update and save changes to their profile, with a confirmation provided upon successful updates.
5. **Database for Storing User Profiles**
    - A database schema was designed and implemented to securely store and manage user profile information. The scenario involved creating tables in the database specifically for storing user profile details.
6. **Integration of Frontend with Backend for User Registration**
    - A user registration page was created, enabling users to sign up for the application by providing basic information. The page includes form validations to ensure users submit accurate and complete details before registration.
    - Additionally, functions were developed to store user details in the database upon successful registration. The scenario implemented ensures that the frontend sends registration data to the backend, where it is securely stored in the database.
7. **PR-Build CI/CD Pipeline for Automated Building and Testing of Code**
    - A CI/CD pipeline was established to automatically build and test code whenever a pull request is created or updated, ensuring that code quality is maintained before merging into the main branch.
    - The implemented scenario triggers the pipeline upon pull requests and displays the build and test results, preventing the merging of pull requests if tests or builds fail.

Total Points Completed:

Stories	**Points**	**Total Points Completed in the Sprint**
Front-End Login Page	3	21
Integration with 3rd-Party Authentication	3	21
Dashboard Navigation	3	21
User Profile Management	3	21
Integrate Frontend with Backend for User Registration	3	21
Set Up a Database for Storing User Profiles	3	21
Create PR-Build CI/CD Pipeline for Automated Building and Testing of Code	3	21

Iteration 2:

Summary

This sprint made significant strides, with the addition of 7 user stories and the implementation of key features across multiple areas.

1. **Mobile Responsive User Interface**
   We optimized the UI for mobile devices, ensuring a seamless user experience on smaller screens. This included adjusting the layout and aligning the design with the client's specifications for mobile responsiveness.
2. **UI Template for Matching with Gym Buddies**
   We introduced an innovative UI template for users to find gym buddies, allowing them to filter profiles based on workout type, location, and experience level. Users can now swipe to match or unmatch with potential partners, enhancing the social interaction within the app.
3. **User Fitness Profile Management**
   We built a system for users to create and manage their fitness profiles, providing options to input personal preferences, review, and update fitness-related details. This feature strengthens user connections by helping them find compatible workout partners.

4. **Revamped Buddy-Finding App UI**
   We launched a redesigned and visually appealing UI, featuring new styling elements and a vibrant color scheme. We gathered user feedback to refine and enhance the design further.
5. **Mock Testing Framework for Authentication**
   We implemented a mock testing framework for the authentication modules, simulating third-party authentication, handling failed logins, and validating session persistence for authenticated users.
6. **Dashboard and User Profile Management Testing**
   We integrated mock testing for the dashboard and profile management features, validating user access and functionality without relying on live authentication. We tested scenarios like profile updates, navigation, and access restrictions for unauthenticated users.
7. **Bug Fix for Profile Picture Visibility in Production**
   We resolved the issue of disappearing profile pictures in the production environment by implementing a robust image storage solution, ensuring consistent display even during dyno restarts. We also completed configuration checks for the external storage service to ensure reliability.

Total Points Completed:

Stories	**Points**	**Total Points Completed in the Sprint**
Mobile Responsive User Interface	3	24
UI Template for Matching with Gym Buddies	4	24
Fix User Profile Picture Visibility Bug in Production	3	24
User Fitness Profile Creation and Management	4	24
Revamped UI Design for Buddy Finding App	4	24
Implement of mock testing framework for Authentication Modules	3	24
Test Integration for Dashboard and User Profile Management	3	24

Iteration 3:

Summary

This sprint achieved impressive progress, adding 9 user stories and implementing key features across multiple areas, focusing on improving fitness profiles, profile matching, notifications, and chat functionality.

1. **Fitness Profile Activity Preferences**
   We empowered users to manage their fitness activity preferences, allowing them to set workout buddy criteria and adjust or remove preferences as needed. These changes were reflected in their fitness profile to enhance partner matching.
2. **Profile Matching Algorithm and Database Integration**
   We developed a backend profile-matching algorithm that sorts and filters profiles based on user preferences. We also tracked interaction history (matched, skipped, blocked) to provide personalized profile recommendations.
3. **Profile Matching UI with User Preferences and Filters**
   We connected the UI to the backend, enabling real-time display of matching profiles. Users can now refine matches with filters like activity type and location, with profiles updating in real time as preferences change.
4. **CSS Bug Fixes and Mobile Responsiveness**
   We resolved CSS issues on the profile matching page and optimized the layout for mobile devices, ensuring a smooth user experience on smaller screens. Responsive design was applied to profile cards and buttons for better interaction.
5. **UI and CSS Enhancements (Client Feedback)**
   We updated font colors, button styles, and backgrounds based on client feedback, ensuring a consistent and visually appealing UI across all pages to improve the overall user experience.
6. **Notification Triggering on Match**
   We implemented notification logic to trigger real-time alerts when users match, sending notifications to both parties. We stored notification details in the database for tracking and future reference.
7. **Notification Dialog Box in UI**
   We created a notification dialog box to display unread and read match notifications. Users can mark notifications as read, and the system persists the status in the backend, including handling an empty state with appropriate messaging.
8. **Private Chat UI Interface**
   We developed a private chat interface for matched users, enabling them to send messages with mock displays in the UI. The chat interface is designed to be responsive, ensuring compatibility on both mobile and desktop devices.

9. **Proof of Concept (POC) for Persistent Chat Feature**
   We implemented basic chat functionality, storing messages in the database for long-term retention. We tested the database schema to ensure scalability and explored real-time messaging with ActionCable, considering options for chat history retention.

<u>Total Points Completed:</u>

Stories	**Points**	**Total Points Completed in the Sprint**
Fitness Profile Activity Preferences	4	35
Profile Matching Algorithm and Database Integration	5	35
Profile Matching UI with User Preferences and Filters	3	35
CSS Bug Fixes and Mobile Responsiveness for Profile Matching	2	35
UI and CSS Enhancements as per client feedback	1	35
Notification Triggering on Match	5	35
Notification Dialog Box in UI for Match Notifications	5	35
Private Chat UI Interface	5	35
Proof of Concept (POC) for Persistent Chat Feature	5	35

Iteration 4:

This sprint made significant progress with the completion of 11 user stories. The team successfully delivered key features across various domains, focusing on bug fixes, real-time chat integration, and user interaction enhancements. Notable updates include implementing the Chat feature, bug fixes for profile management, improved search and matching functionality, and a streamlined UI design.

1. **Chat Feature Backend**
   We developed backend API endpoints to handle messages and provide real-time updates. Database tables were set up to store messages, user details, and timestamps, followed by testing to ensure seamless chat functionality.
2. **Chat Feature Frontend Integration**
   We connected the backend API to the chat UI, enabling users to send and receive messages in real-time, ensuring the chat interface was fully functional.
3. **User Matching and Interaction**
   We displayed matched users based on user preferences and added search functionality to allow users to find specific profiles. Users can now view complete profiles of matches, block users, and open chat windows.
4. **Fitness Profile Bug Fixes**
   We resolved the intermittent loading issue with the workout days input box, ensuring consistent functionality. Dropdown menus were also improved to automatically close when users click outside them.
5. **Bug Fixes for Profile Matching Page**
   We added a "No Available Profiles" message when no matches exist and prevented the reappearance of the last matched profile after processing. We restricted access to the profile matching page until the fitness profile was completed and introduced a confirmation modal for blocking users. Additionally, we ensured that user photos were displayed correctly on profile cards and fixed navbar padding issues for mobile view.
6. **Enhanced Dashboard Design**
   We implemented a clean and organized layout for the user dashboard, applying a modern color scheme and typography. The dashboard was also made fully mobile-responsive for improved usability.
7. **Custom Alert Boxes**
   We created visually distinct confirmation modals for sensitive actions like blocking and deleting, ensuring background interactions were blocked when alert modals were active.
8. **Fix Background Image Visibility**
   We resolved issues with the background image, ensuring it adapts dynamically to cover tall screens for a consistent visual experience.

9. **Navbar Adjustments on Login Page**
   We removed the navbar button on the login page in mobile view for a cleaner interface while restoring it on other pages to ensure mobile users had access.
10. **Improved Notifications Modal**
    We simplified the notifications modal by removing the "Mark as Unread" option, making it clearer. Unread notifications are now displayed at the top for better visibility.
11. **User Profile Loading Issues Due to JavaScript and Cache Errors**
    We fixed JavaScript errors that were causing profile load failures, ensuring a smoother user experience.

Total Points Completed:

Stories	**Points**	**Total Points Completed in the Sprint**
Chat Functionality Backend and Database Setup	4	28
Integration of Chat Backend with Frontend UI	4	28
User Matching and Interaction Options	3	28
Fitness Profile Management Bug Fixes	2	28
Bug Fixes for Profile Matching Page	4	28
Enhanced Dashboard Design for User Landing Page	2	28
Custom Alert Boxes for Action Confirmations	2	28
Fix Background Image Visibility for Tall Screens	1	28
Hide Navbar Button on Login Page for Mobile View	1	28
Fix Notification Modal Box Bugs and Improve UI for Read/Unread Notifications	4	28
Fix User Profile Loading Issues Due to JavaScript and Cache Errors	1	28

Team Member Contribution:

Overall:

Name	Total Stories completed	Total points solved
Kuan-Ru Huang	4	15.5
Wei-Chien Cheng	6	15.5
Yash Phatak	5	15.5
Mrunmay Deshmukh	6	15
Kushal Lahoti	5	15.5
Barry Liu	4	15.5
ChuanHsin Wang	4	15.5

Iteration 1:

Name	Total Stories completed	Total points solved
Kuan-Ru Huang	1	3
Wei-Chien Cheng	1	3
Yash Phatak	1	3
Mrunmay Deshmukh	1	3
Kushal Lahoti	1	3
Barry Liu	1	3
ChuanHsin Wang	1	3

Iteration 2:

Name	Total Stories completed	Total points solved
Kuan-Ru Huang	1	3.5
Wei-Chien Cheng	1	3.5
Yash Phatak	1	3.5
Mrunmay Deshmukh	1	3
Kushal Lahoti	1	3.5
Barry Liu	1	3.5
ChuanHsin Wang	1	3.5

Iteration 3:

Name	Total Stories completed	Total points solved
Kuan-Ru Huang	1	5
Wei-Chien Cheng	1	5
Yash Phatak	2	5
Mrunmay Deshmukh	2	5
Kushal Lahoti	1	5
Barry Liu	1	5
ChuanHsin Wang	1	5

Iteration 4:

Name	Total Stories completed	Total points solved
Kuan-Ru Huang	1	4
Wei-Chien Cheng	3	4
Yash Phatak	1	4
Mrunmay Deshmukh	2	4
Kushal Lahoti	2	4
Barry Liu	1	4
ChuanHsin Wang	1	4

Customer Meeting Dates and Description:

Sprint 1:

- **Date:** Oct 9, 2024
- **Time:** 10:30 AM - 11:00 AM
- **Place:** Zoom Call
- **Description:**
    - Demonstrated the **login page using Google** third-party authentication.
    - Showcased **dashboard navigation** allowing users to access and mange their profiles.
    - Highlighted the integration of frontend and backend for seamless user registration.
- **Feedback:** As this was the initial meeting with the client, the overall direction and goals of the application were discussed.

Sprint 2:

- **Date:** Oct 16, 2024 and Oct 23, 2024
- **Time:** 10:00 AM - 10:30 AM
- **Place:** Zoom Call
- **Description:**
    - Presented the mobile-responsive UI, ensuring an optimized user experience on all devices
    - Showcased **fitness profile** creation, allowing users to specify workout preferences and availability.
    - Introduced the **profile matching** feature, enabling users to swipe and match with potential gym buddies.
    - Discussed enhancements to the app’s visual appeal and usability based on client feedback.
- **Feedback:**
    - Implement a block feature for user interactions
    - Adjust the profile matching sequence to display unmatched users first.

Sprint 3:

- **Date:** Oct 23, 2024 and Nov 6, 2024
- **Time:** 10:00 AM - 10:30 AM
- **Place:** Zoom Call
- **Description:**
    - Demonstrated a **Proof of Concept (POC)** for a persistent chat feature to enable real-time communication.
    - Highlighted the implementation of a **real-time notification system** for user matches.
    - Showcased UI/UX improvements, focusing on mobile responsiveness and consistency.
    - Presented advanced **profile-matching algorithms** that incorporate user preferences.
- **Feedback:**
    - Enhance the mobile experience further.
    - Refine the private chat feature with user feedback.

Sprint 4:

- **Date:** Nov 13, 2024 and Nov 20, 2024
- **Time:** 10:00 AM - 10:30 AM
- **Place:** Zoom Call
- **Description:**
  - Showcased the fully implemented **chat feature** with real-time messaging and message persistence.
  - Highlighted critical bug fixes, particularly user matching and fitness profile management.
  - Shared insights from beta user testing, emphasizing improvements in app stability and user feedback integration.
  - Presented UI/UX enhancements, including optimized mobile responsiveness and streamlined navigation.
- **Feedback:**
  - The client expressed satisfaction with the chat feature and UI refinements, emphasizing readiness for the final testing phases.

BDD/TDD Processes:

To ensure a solid application, we adopted a combined Behavior-Driven Development (BDD) and Test-Driven Development (TDD) approach.

- BDD - It focused on end-user interactions and was implemented using Cucumber with detailed feature files providing a simple language to bridge the gap between non-technical stakeholders and developers.
- TDD - It focused on backend reliability and was implemented using RSpec with _spec.rb files to test models and controllers.

BDD Process:

1. Write Scenarios: Created high-level scenarios based on user stories to define the expected behavior of the application clearly.
2. Implement Step Definitions: Step files map each scenario to application logic using precise validations.
3. Run Tests: Cucumber tests verified that the application matched the expected behavior.
4. Introduction of every features:
    a. fitness_profile_management.feature ensured users could add, update, and delete fitness preferences.
    b. profile_swipe.feature validated that users could swipe to match or block potential gym buddies.
    c. conversations.feature enables real-time communication between matched users for seamless interaction.
    d. dashboard_navigation.feature provides users with an overview of available features and seamless access to their profile management page.
    e. notification.feature keeps users informed about mutual matches through notifications, which can be marked as read or unread for better management.
    f. session.feature enables secure access to the platform using Google OAuth, providing tailored user experiences for first-time and returning users, while ensuring non-configured users complete their profiles and offering clear feedback for login or logout actions.
    g. user_matches.feature allows users to explore, manage, and interact with potential matches through actions like matching, skipping, and blocking, while also providing safeguards against invalid operations and ensuring mutual interactions are notified appropriately.

TDD Process:

1. Write Unit Tests: The RSpec test ensured that individual components worked correctly before implementation.
2. Run Tests: Each feature was built incrementally, ensuring the tests were passed before moving on to the next step.
3. Refactoring: Having tests in place allowed safe refactoring of code. For example, the existing tests of notifications_controller_spec ensured no regression issues were reintroduced.
4. Introduction of every test:
    - channels
        - connection_spec ensures that the WebSocket connection functionality in the application is established successfully and operates as expected.
    - Controllers
        - application_controller_spec ensures that the require_login before_action is functioning correctly by redirecting unauthenticated users to the welcome page and providing an appropriate flash alert message when access to a protected action is attempted.
        - conversations_controller_spec ensures that the ConversationsController correctly manages conversations between matched users, including user assignments, conversation retrieval or creation, message loading, and initializing a new message for the form.
        - dashboard_controller_spec ensures that the DashboardController responds with a redirect when attempting to access a non-existent user's dashboard.
        - fitness_profiles_controller_spec ensures that the FitnessProfilesController handles all key actions, including creating, viewing, editing, and updating fitness profiles, while validating user session setup and redirecting appropriately upon successful operations.
        - matching_controller_spec ensures the MatchingController correctly assigns user-related variables, interacts with UserMatchesController to retrieve prospective matches, renders the appropriate view, and verifies the OmniAuth authentication process.
        - messages_controller_spec ensures that the MessagesController handles message creation properly, verifying successful creation with valid parameters and returning appropriate error responses with invalid parameters, all within the context of a mutual match conversation.
        - notifications_controller_spec verifies that the NotificationsController handles notification management correctly, including fetching notifications, marking them as read or unread, and ensuring proper

behavior for both logged-in and non-logged-in users, with appropriate HTTP responses and redirects.
    - profiles_controller_spec evaluates the ProfilesController's show action to ensure it handles profile retrieval robustly, covering cases of valid and invalid user profiles, with checks to confirm that no errors are raised during the process.
    - sessions_controller_spec ensures the SessionsController correctly handles user authentication with Google OAuth, including scenarios for successful login, logout, failed authentication, and handling users with incomplete profiles.
    - user_matches_controller_spec validates the core functionalities of the UserMatchesController, ensuring that prospective users are retrieved correctly, and user interactions like matching, skipping, and blocking are handled appropriately, including error scenarios and notifications for reciprocal matches.
    - users_controller_spec ensures the UsersController correctly handles user profile updates, validating scenarios like successful updates, incomplete or invalid inputs, and file upload constraints (size and format), while providing appropriate feedback and maintaining the application's stability.
    - welcome_controller_spec checks whether the GET #index action in the WelcomeController correctly handles a scenario where a logged-in user does not receive a welcome message.
- jobs
    - user_match_job_spec validates that the UserMatchJob creates new and reciprocal matches correctly, prevents self-matches, and avoids duplicating existing matches.
- models
    - fitness_profile_spec validates the FitnessProfile model's core functionality, ensuring attributes and custom methods work as expected, such as activities, schedule, workout types, gym locations, and gender preferences.
    - user_spec confirms the validity of the User model under specific scenarios, ensuring required attributes like email and username are present, particularly during profile updates.

Benefits:

- High Test Coverage - Achieved a combined coverage of 95.6%, reflecting a well-tested code.
- Fewer Bugs - Early detection and resolution reduced issues during the integration phases.
- Maintainability - The tests provided a safety net for refactoring without fear of breaking functionality.

Challenges:

- Time-Consuming - Writing detailed tests and scenarios increased initial development time. However, this investment paid off during later stages with reduced debugging time.
- Mocking Complexity - Services like OmniAuth for Google Authentication required extensive mocking, complicating the test process. We were successfully able to simulate the OmniAuth tests using FactoryBot.
- UI Testing Complexity - Writing test cases for interactive web pages was challenging. We used a selenium web driver to solve this.

Configuration Management Approach:

- The configuration management strategy focused on maintaining a streamlined workflow with robust version control and clear branching practices. We utilized a Git-based approach, ensuring that all changes were tracked, reviewed, and tested before merging into the main branch.
- Each developer worked on their individual development branches to implement specific features or user stories. Once completed, these branches were merged into an integration branch, where features were tested collectively to ensure seamless interaction.
- After successful integration testing, the changes were merged into the main branch for deployment.
- A CI/CD pipeline was implemented to automatically test and build code at each stage, maintaining high code quality and reducing integration risks. Releases were organized sprint-wise, ensuring completed features were deployed consistently at the end of each sprint.

Spikes Conducted:

- We performed spikes to explore and address specific challenges, such as implementing a mock testing framework for authentication modules, ensuring profile picture persistence across production environments, and researching the implementation of a profile-matching algorithm.

- The spike for the profile matching algorithm involved investigating different techniques to enable matching gym buddies based on workout preferences, location, and experience level, ensuring it was both efficient and user-friendly.

Branches and Releases:

- The branching strategy involved individual development branches for each developer, which were merged into an integration branch for collective testing and then into the main branch for production.
- A total of **48 branches** were created throughout the project to implement various features and user stories.
- The project followed a structured release process with **4 sprint-wise releases**, ensuring that completed features were deployed consistently and effectively throughout the sprint cycle.

Issues in the Production Release Process on Heroku:

While deploying the app on Heroku, no issues were found. The process was completed smoothly without any errors or disruptions.

Tools/Gems used:

GitHub:
GitHub was used for version control and collaboration. It made it easy to track code changes, create branches, and merge them, which was essential for keeping the codebase clean and for teamwork. The main benefits were easy pull requests, code reviews, and integration with CI/CD pipelines. However, managing merge conflicts during feature development was sometimes challenging and required careful coordination.

CodeClimate:
CodeClimate helped monitor code quality by checking for issues like maintainability, complexity, and duplication. It was useful for keeping the code healthy and identifying areas for improvement. While it provided continuous feedback, it occasionally gave false positives, meaning some issues flagged by the tool needed manual verification.

SimpleCov:
SimpleCov was used to track test coverage, ensuring that the code was well-tested across different areas. It generated reports to show which parts of the code were covered by tests, helping us find areas that needed more testing. A challenge was getting all developers to write tests for new features, as missing coverage could impact the quality of the reports.

Pivotal Tracker:
We used Pivotal Tracker to manage tasks and track progress throughout the sprint. It helped break down larger features into smaller tasks, assign complexity points, and monitor progress in real-time. This tool improved team collaboration and made sprint planning easier. However, new users had a learning curve, and some customization limitations were noted. Despite these, it was an effective tool for managing and tracking project progress.

Gems used:

Below is a summary of key gems included in the project, highlighting their functionality and relevance:

Core Framework and Utilities

- Rails (~> 7.2.1): The core framework for building the application, providing MVC architecture, routing, and other essential features.
- Puma (>= 5.0): A fast, multi-threaded web server optimized for production environments.
- Importmap-Rails: Enables JavaScript module imports without needing Webpack or other bundlers.
- Turbo-Rails: Enhances user experience by enabling SPA-like navigation and faster page loads.
- Stimulus-Rails: Provides lightweight JavaScript framework for front-end interactivity.
- Jbuilder: Simplifies building JSON APIs in Rails.

Authentication and Authorization

- Devise: A flexible authentication solution for Rails applications.
- OmniAuth: A framework for multi-provider authentication, extended with:
    - OmniAuth-Google-OAuth2: For Google login integration.
    - OmniAuth-Rails-CSRF-Protection: Mitigates CSRF vulnerabilities in OmniAuth.

File and Asset Management

- CarrierWave (~> 2.0): Handles file uploads and processing.
- Sassc-Rails: Provides SCSS compilation for styling.
- Bootstrap (~> 5.3.0): Implements responsive, professional front-end design.

Database and Redis

- SQLite3 (>= 1.4): The default database for development and testing environments.
- Redis (>= 4.0.1) and Redis-Client: Utilized for caching and Action Cable support.
- Pg: PostgreSQL adapter, essential for production (e.g., Heroku deployments).

Testing and Development Tools

- RSpec-Rails: Popular testing framework tailored for Rails.
- Capybara: Facilitates system and acceptance testing.
- SimpleCov: Provides test coverage analysis.
- Cucumber-Rails: Supports BDD (Behavior-Driven Development).
- Database Cleaner: Ensures clean state for database between tests.
- FactoryBot-Rails: Simplifies creation of test data.
- Brakeman: Static analysis tool for security vulnerabilities.
- Rubocop (and extensions): Ensures adherence to Ruby and Rails style guides.

Production-Specific

- AWS-SDK-S3: For integrating Amazon S3 storage.
- Tzinfo-Data: Provides timezone information for Windows platforms.

Code Management:

All code is pushed to the main branch, which holds the most up-to-date version of the project. At the end of each sprint, a tag is created to mark the code for that sprint, making it easy to reference and track progress.

Github Repo Information:

Repository Name: Jimmy Gym Buddy Finder
GitHub Link: https://github.com/tamu-edu-students/jimmy-gym-buddy-finder

The Repository is structured as follows:

- /app - It contains the core application logic:
    - Models - It defines the relationship between the application components.
    - Controllers - It manages user interactions and business logic.
    - Views - It contains the templates for rendering the user interface.
    - Javascript contains the .js files used to enable user interaction and enhance user experience.
    - Assets - It contains the static files specific to the application, such as:
        - Stylesheets containing the CSS files.
        - Images containing the images used for the application.
- /config - It contains the application configuration files, including:
    - .yml files for various configurations.
    - routes.rb to define application routes and endpoints.

- /db - It handles database management:
    - Migration - It contains the scripts for updating the database schema.
    - schema.rb contains the auto-generated current database schema.
- /documentation - It contains project-related documentation.
- /features - It contains test cases for Behaviour-driven Development (BDD) using cucumber.
- /spec - It contains test cases for Test-driven Development (TDD) using RSpec.
- Additional files:
    - Gemfile specifies the Ruby gem dependencies for the application.
    - README.md contains the information needed to setup and deploy the application.
    - Dockerfile is used for containerizing the application.
    - Procfile contains the configuration for deploying the application on Heroku.

Deployment Process and Scripts:

- The deployment involves setting up Heroku, Heroku Postgres, Heroku Redis, Google OAuth and Amazon AWS S3 Bucket.
- The detailed steps for configuring the above components have been added to our README.md file of the application repository and can be found out here - https://github.com/tamu-edu-students/jimmy-gym-buddy-finder/blob/main/README.md

Links to our GitHub Repo, Heroku Deployment, Pivotal Tracker, Slack workspace, and Code Climate:

- Github Repo - https://github.com/tamu-edu-students/jimmy-gym-buddy-finder
- Deployment URL - https://jimmy-buddy-finder-f97708d96ef8.herokuapp.com/
- Pivotal Tracker - https://www.pivotaltracker.com/n/projects/2721606
- Slack Workspace - https://app.slack.com/client/T07P2NT2ZM1/C07P00FFRGD
- Code Climate - https://codeclimate.com/github/tamu-edu-students/jimmy-gym-buddy-finder

Links to Presentation and Demo Video:

- Presentation & Demo Video - https://www.youtube.com/watch?v=802vG74ZODY