Sprint 3 Plan - Project Jimmy

Team roles:

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

Post sprint client meeting date/time/place:

Date: 16th and 23rd October, 2024 (Every Wednesday)
Time: 10:30am - 11:00am
Place: Zoom Call

Client Meeting Summary:

In our recent client meeting, we provided a demo of the gym buddy finder application, showcasing the features and functionality implemented so far. The demo was well-received, and the clients expressed satisfaction with the overall progress and pace of the project.

During the demo, we received specific **UI/UX and CSS styling feedback**, particularly related to font styling and the appearance of certain elements. We have acknowledged these points and will be addressing them in the current sprint to ensure the application aligns with the client's design expectations.

Additionally, the client plans to conduct **user testing** using the deployed URL and will provide feedback regarding any issues or modification requests. This feedback will be shared with us in the form of a document for further review and implementation in upcoming sprints.

Sprint goal:

The primary goal of this sprint is to enhance the core user experience of the gym buddy finder application by improving user interaction and communication features while addressing key functional and design issues and conducting a Proof of Concept (POC) for new functionality.

1. **Feature Implementation:** The sprint focuses on implementing the following features – the profile matching backend, a notification system for matched users, and a private chat UI.

    a. The **profile matching feature** will be fully connected to the backend, ensuring that users can view actual profiles based on their preferences, with relevant filters and sorting algorithms in place.
    b. A **notification system**, where both the notification backend and the notification UI will be implemented, allowing users to receive match notifications and mark them as read.
    c. A **mock private chat UI** will be implemented, allowing users to initiate communication with their matched gym buddies.

2. **UI/UX and CSS Styling Improvements**: A significant portion of the sprint will focus on resolving **CSS-related bugs**, especially on the **profile matching page**, ensuring that the interface is fully responsive and visually appealing across devices. The application will be adjusted to improve the mobile experience, including proper alignment, font scaling, and consistent button styling. Additionally, we will address minor UI issues raised by client feedback, such as **font colors, button styling, and background adjustments**.

3. **POC for Persistent Chat Feature**: A **Proof of Concept (POC)** will be conducted to explore how a **persistent chat feature** can be implemented in the Ruby on Rails application. The POC will determine the best approach for the future implementation of a robust chat feature.

By the end of this sprint, the application will have improved profile matching functionality, better design, a notification system, and a solid foundation for future chat integration, providing an overall richer and more responsive user experience.

User stories: A total of 9 user stories have been added to the sprint plan.

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: 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: 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: 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: 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: 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),

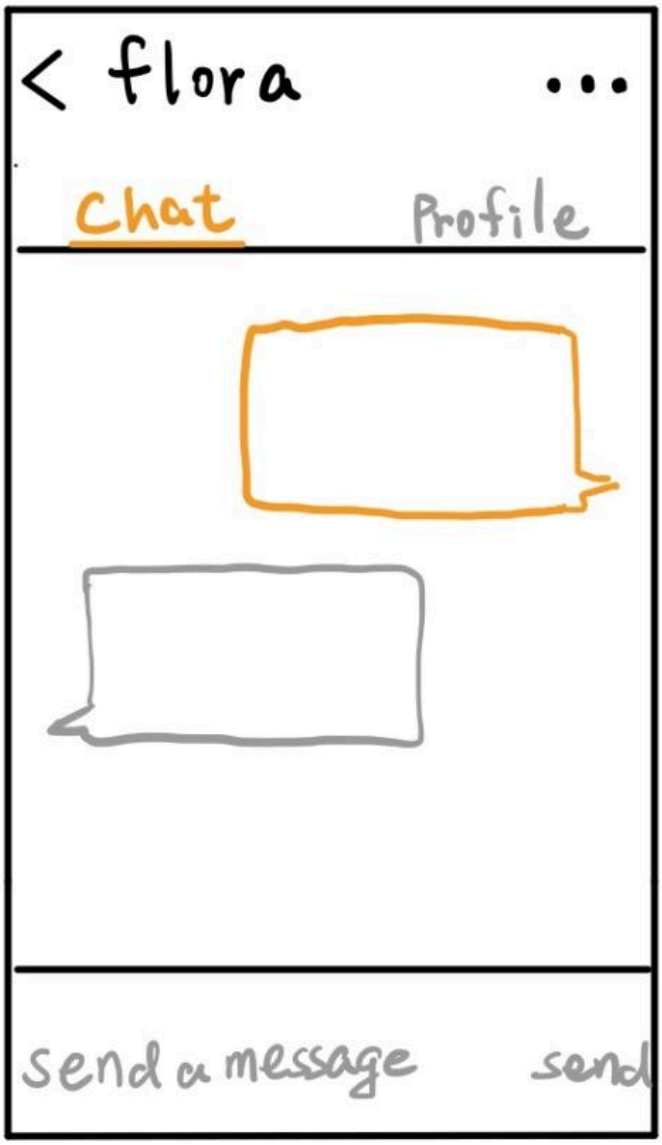
Then: The schema should allow efficient retrieval and querying of past conversations.

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.

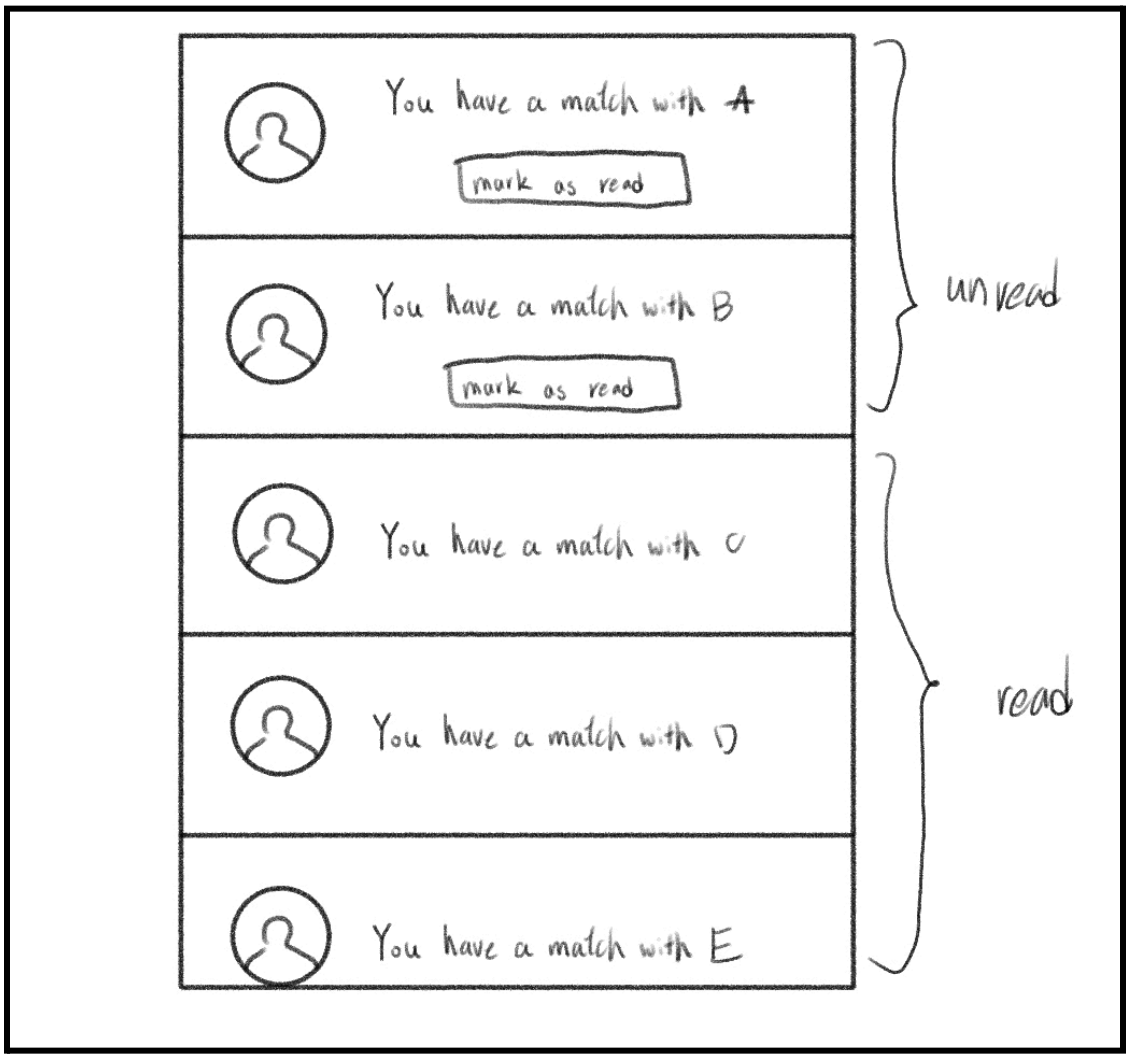
Scenario 4: Exploring Data Retention and Archiving
Given: Users may want to retain chat history for future reference,
When: The POC considers data retention policies and archiving strategies,
Then: The system should offer options for retaining or archiving old conversations.

User stories: User interface: Lo-fi UI mockups and storyboards

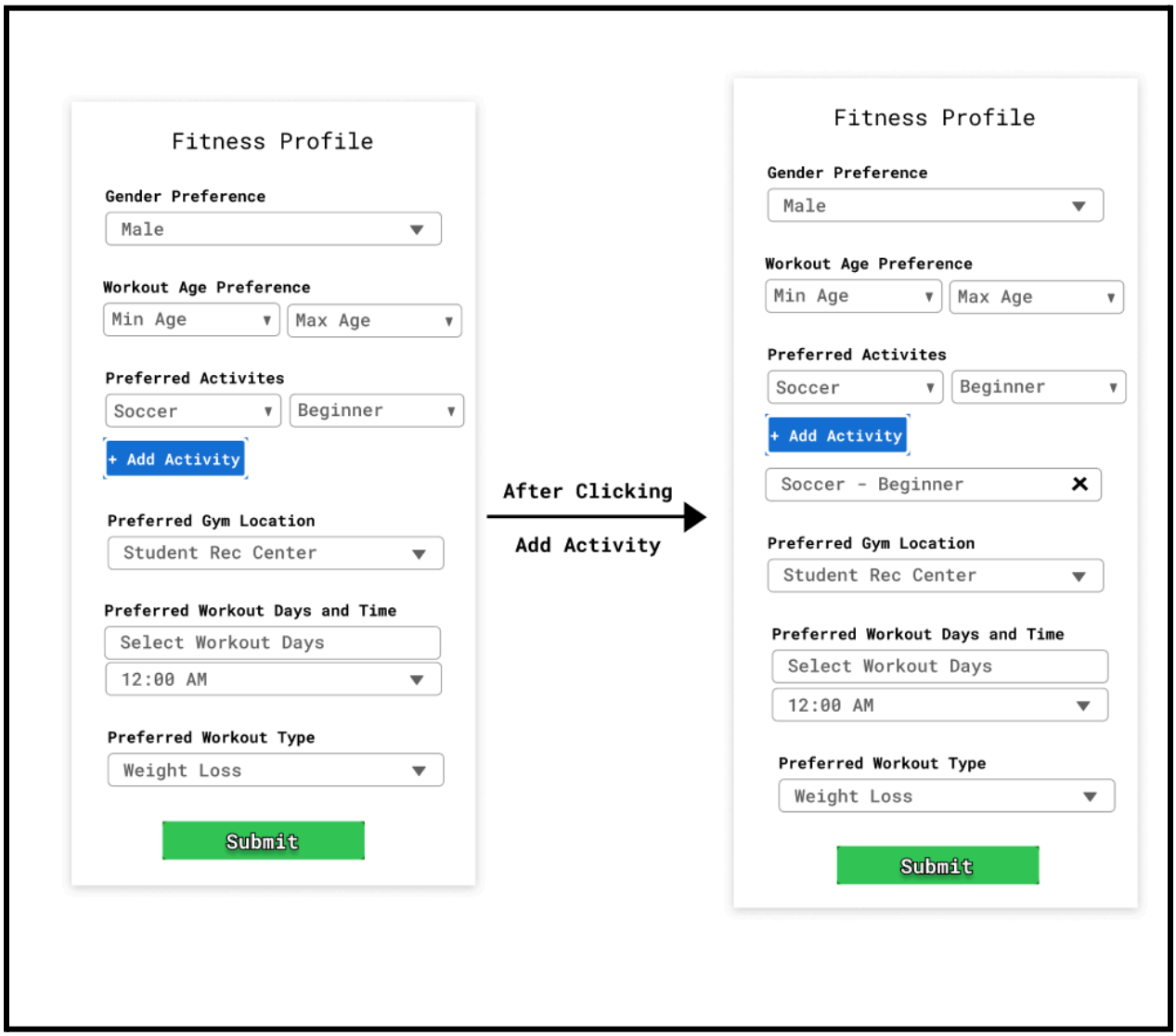
Chat Room UI:



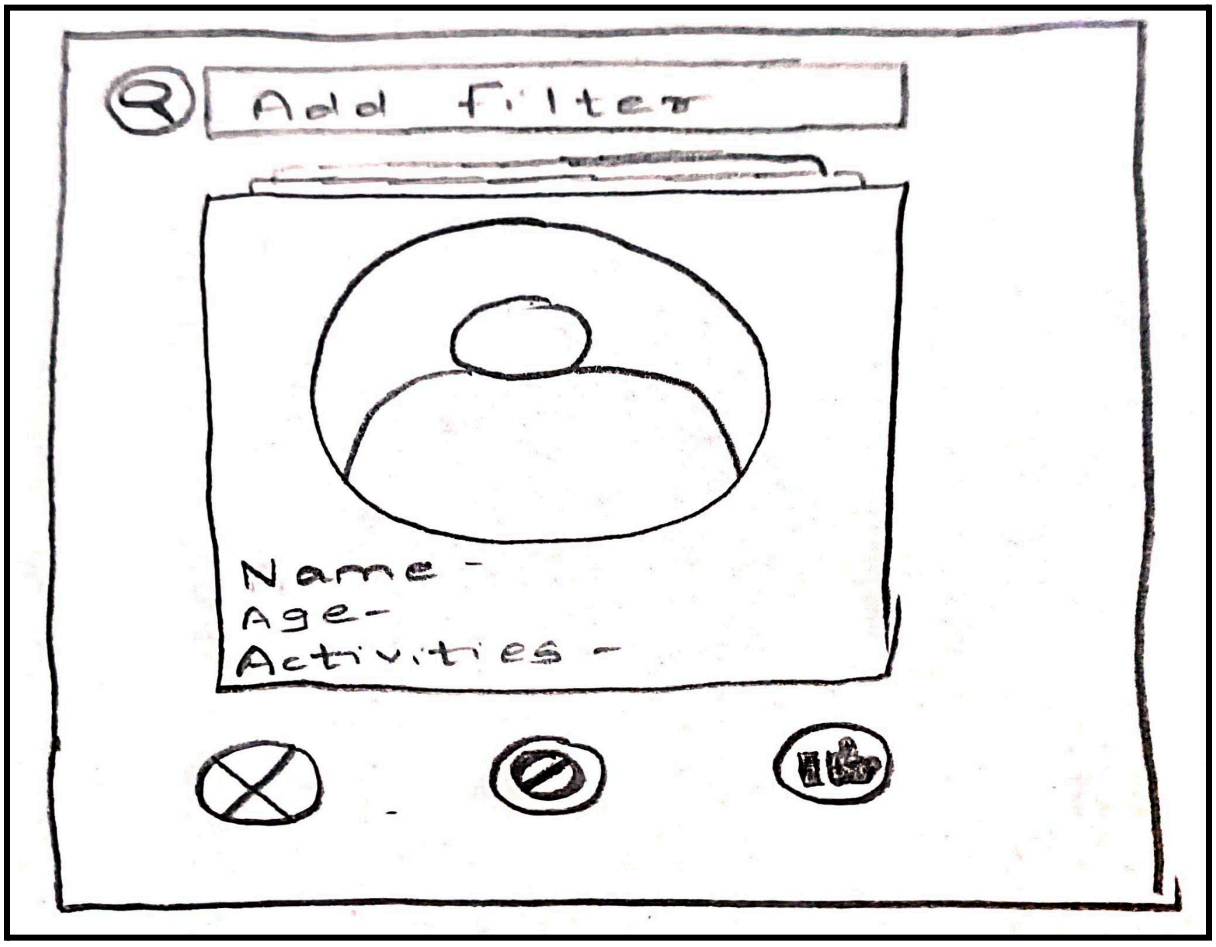
Notifications UI:



Fitness Profile UI:



Profile Matching UI:



Which stories were pulled into the Sprint? To whom is each story assigned?

Stories	Points	Assigned To
Fitness Profile Activity Preferences	4	Mrunmay Deshmukh
Profile Matching Algorithm and Database Integration	5	Kushal Lahoti
Profile Matching UI with User Preferences and Filters	3	Yash Phatak
CSS Bug Fixes and Mobile Responsiveness for Profile Matching	2	Yash Phatak
UI and CSS Enhancements as per client feedback	1	Mrunmay Deshmukh
Notification Triggering on Match	5	Wei-Chien Cheng
Notification Dialog Box in UI for Match Notifications	5	ChuanHsin Wang
Private Chat UI Interface	5	Barry Liu
Proof of Concept (POC) for Persistent Chat Feature	5	Kuan-Ru Huang

What are the tasks and their time estimates?

A. Fitness Profile Activity Preferences - 4 Days
B. Profile Matching Algorithm and Database Integration - 1 Week
C. Profile Matching UI with User Preferences and Filters - 3 Days
D. CSS Bug Fixes and Mobile Responsiveness for Profile Matching - 2 Days
E. UI and CSS Enhancements as per client feedback - 1 Day
F. Notification Triggering on Match - 1 Week
G. Notification Dialog Box in UI for Match Notifications - 1 Week
H. Private Chat UI Interface - 1 Week
I. Proof of Concept (POC) for Persistent Chat Feature - 1 Week

Links to our GitHub Repo, Heroku Deployment, Pivotal Tracker, and 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