Sprint 4 Retrospective - Project Jimmy

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

Dates of the Sprint:

4th November 2024 to 17th November 2024

Information about team members and contributions:

Team Member	Contribution	Tasks
Kuan-Ru Huang	13%	• Integration of Chat Backend with Frontend UI
Wei-Chien Cheng	14%	• Custom Alert Boxes for Action Confirmations • Fix Background Image Visibility for Tall Screens • Hide Navbar Button on Login Page for Mobile View
Yash Phatak	15%	• Bug Fixes for Profile Matching Page
Mrunmay Deshmukh	15%	• Fitness Profile Management Bug Fixes • Enhanced Dashboard Design for User Landing Page

Kushal Lahoti	15%	• User Matching and Interaction Options • Fix User Profile Loading Issues Due to JavaScript and Cache Errors
Barry Liu	15%	• Chat Functionality Backend and Database Setup
ChuanHsin Wang	13%	• Fix Notification Modal Box Bugs and Improve UI for Read/Unread Notifications

Each team member's total solved user story points to date:

Name	Total points solved by the individual	Total points solved by the entire team
Kuan-Ru Huang	15.5	108
Wei-Chien Cheng	15.5	108
Yash Phatak	15.5	108
Mrunmay Deshmukh	15	108
Kushal Lahoti	15.5	108
Barry Liu	15.5	108
ChuanHsin Wang	15.5	108

Sprint Goal:

Goals for the Current Sprint

- Bug Fixes
    - Address multiple bugs from the previous sprint to enhance application stability.
    - These fixes are critical as the application approaches the final testing phases.
- Chat Feature Implementation
    - Develop and integrate a chat feature to facilitate user communication.
    - Full integration is targeted by the end of this sprint.
- Beta User Testing Initiation
    - Begin beta user testing next Wednesday to gather real-world feedback.
    - Insights will drive final adjustments to improve user experience.
- Total User Stories: 11 user stories added.
- Key Features: Chat functionality, user matching, profile management fixes, enhanced UI/UX, and background optimizations.

User Story

Chat Feature Implementation: As a user, I want a reliable chat feature to communicate with workout partners, enabling seamless connection within my fitness community.

- Backend Setup: Develop APIs for sending, receiving, and updating messages in real time.
- Database Configuration: Create tables for storing messages, timestamps, and chat session details securely.
- Functionality Testing: Validate smooth, error-free message delivery and retrieval.
- Testing Frameworks: Write RSpec and Cucumber tests to ensure backend reliability.
- Frontend Integration: Connect the backend to a responsive UI for real-time chat display.

User Matching and Interaction: As a user, I want to view and interact with matched users based on preferences, making it easier to connect with suitable partners.

- View Matched Users: Display users matching preferences with details (e.g., name, interests).
- Search Users: Filter matches using keywords.
- Profile Viewing: Display complete profiles of selected users.
- User Blocking: Allow blocking matched users with a confirmation modal.
- Initiate Chat: Open a chat window directly from the matched user's profile.

Fitness Profile Management Fixes

- Fix Loading Issues: Ensure consistent loading of input boxes.

- Dropdown Behavior: Close dropdown menus when clicking outside.

Enhanced Dashboard Design: As a user, I want a visually appealing, functional dashboard to easily access features while enjoying a clean interface.

- Layout Optimization: Create a structured layout with clear sections.
- Modern Design: Use appealing color schemes and legible typography.
- Mobile-Responsive Design: Adapt layouts and widgets for mobile devices.

Bug Fixes and UI Improvements:

- Profile Matching Page:

  - No Matches Message: Display a clear message if no matches exist.

  - Prevent Profile Reappearance: Ensure unmatched profiles don’t reappear.

  - Profile Completion: Redirect users without complete profiles.

- Notifications:

  - Remove Redundant Buttons: Eliminate "Mark as Unread" for clarity.

  - Unread Notification Priority: Highlight unread messages prominently.

- Navbar:

  - Hide on Login Page: Simplify login UI by hiding unnecessary elements in mobile view.

- Background Image:

  - Stretch to Fit: Ensure background images dynamically adjust to fit all screen sizes.

Testing and Validation

- RSpec and Cucumber: Comprehensive tests for backend API endpoints, database, and user interfaces.
- Beta Feedback: Real-world user input to guide final optimizations.

Sprint Achievements:

This sprint made remarkable progress in which the team delivered a total of 11 user stories. Key features were implemented across various domains focusing on bug fixes, real-time chat integration, and user interaction enhancements. Key updates included implementing the Chat feature, bug fixes of profile management, improved search and matching, and streamlined UI design.

1. **Chat Feature Backend**

- Built backend API endpoints for message handling and real-time updates.
- Set up database tables for storing messages, user details, and timestamps.
- Completed testing to ensure seamless chat functionality.

2. **Chat Feature Frontend Integration**

- Connected backend API to the chat UI for sending and receiving messages.
- Ensured real-time message updates in the chat interface.

3. **User Matching and Interaction**

- Displayed matched users based on preferences.
- Enabled search functionality to find specific users.
- Allowed users to view complete profiles of matched users.
- Provided an option to block users and open chat windows.

4. **Fitness Profile Bug Fixes**

- The intermittent loading issue with the workout days input box was resolved to ensure consistent functionality.
- Dropdown menus were enhanced to close automatically when users click outside.

5. **Bug Fixes for Profile Matching Page**

- Added a “No Available Profiles” message when no matches exist.
- Prevented reappearance of the last matched profile after processing.
- Restricted access to profile matching until fitness profile completion.
- Introduced a confirmation modal for blocking users.
- Ensured user photos were displayed correctly on profile cards.
- Fixed navbar padding issues for mobile view.

6. **Enhanced Dashboard Design**

- Implemented a clean and organized layout for the user dashboard.
- Applied a modern color scheme and typography.
- Made the dashboard fully mobile-responsive.

7. **Custom Alert Boxes**

- Created visually distinct confirmation modals for sensitive actions like blocking and deleting.
- Prevented background interactions when alert modals are active.

8. **Fix Background Image Visibility**

- Ensured the background image adapts dynamically to cover tall screens.

9. **Navbar Adjustments on Login Page**

- Removed the navbar button on the login page in mobile view for a cleaner interface.
- Restored the navbar button on other pages for mobile users.

10. **Improved Notifications Modal**

- Removed the "Mark as Unread" option for better clarity.
- Displayed unread notifications at the top for better visibility.

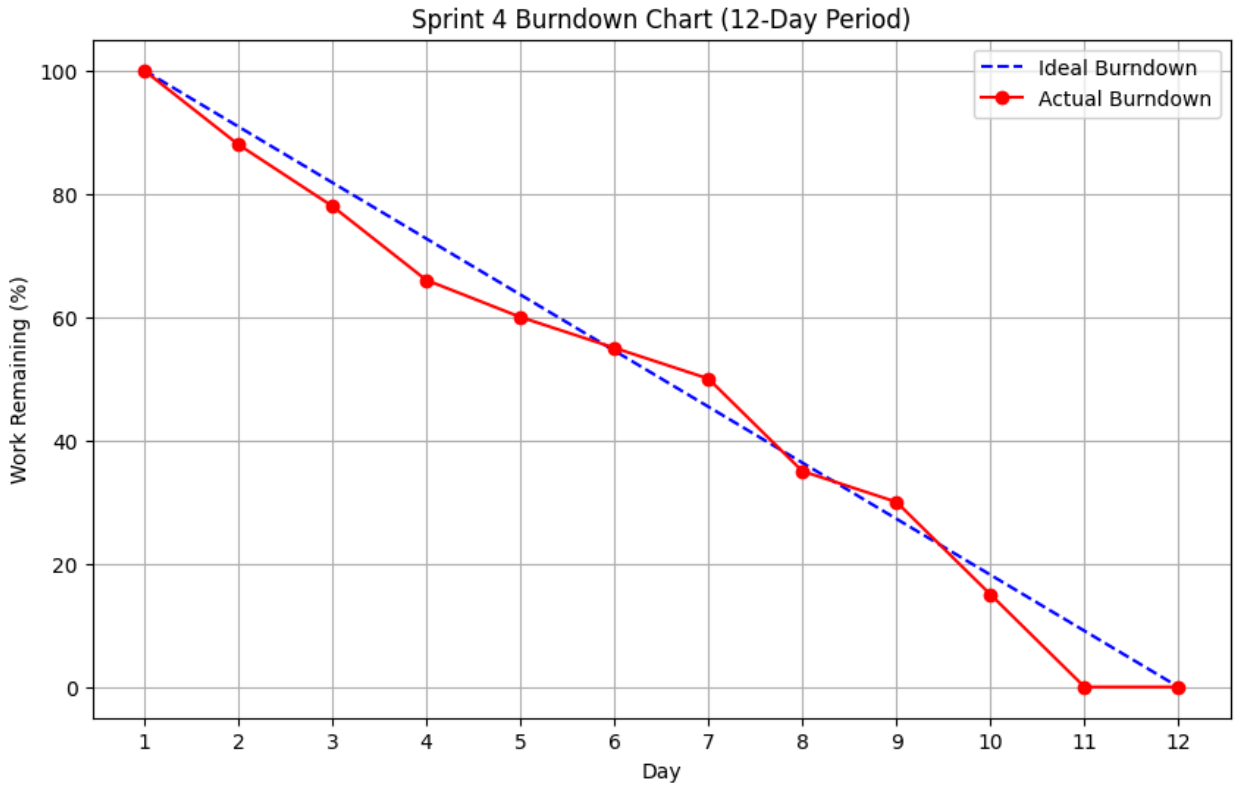
11. **User Profile Loading Issues Due to JavaScript and Cache Errors**

- Fixed JavaScript Errors Causing Profile Load Failures

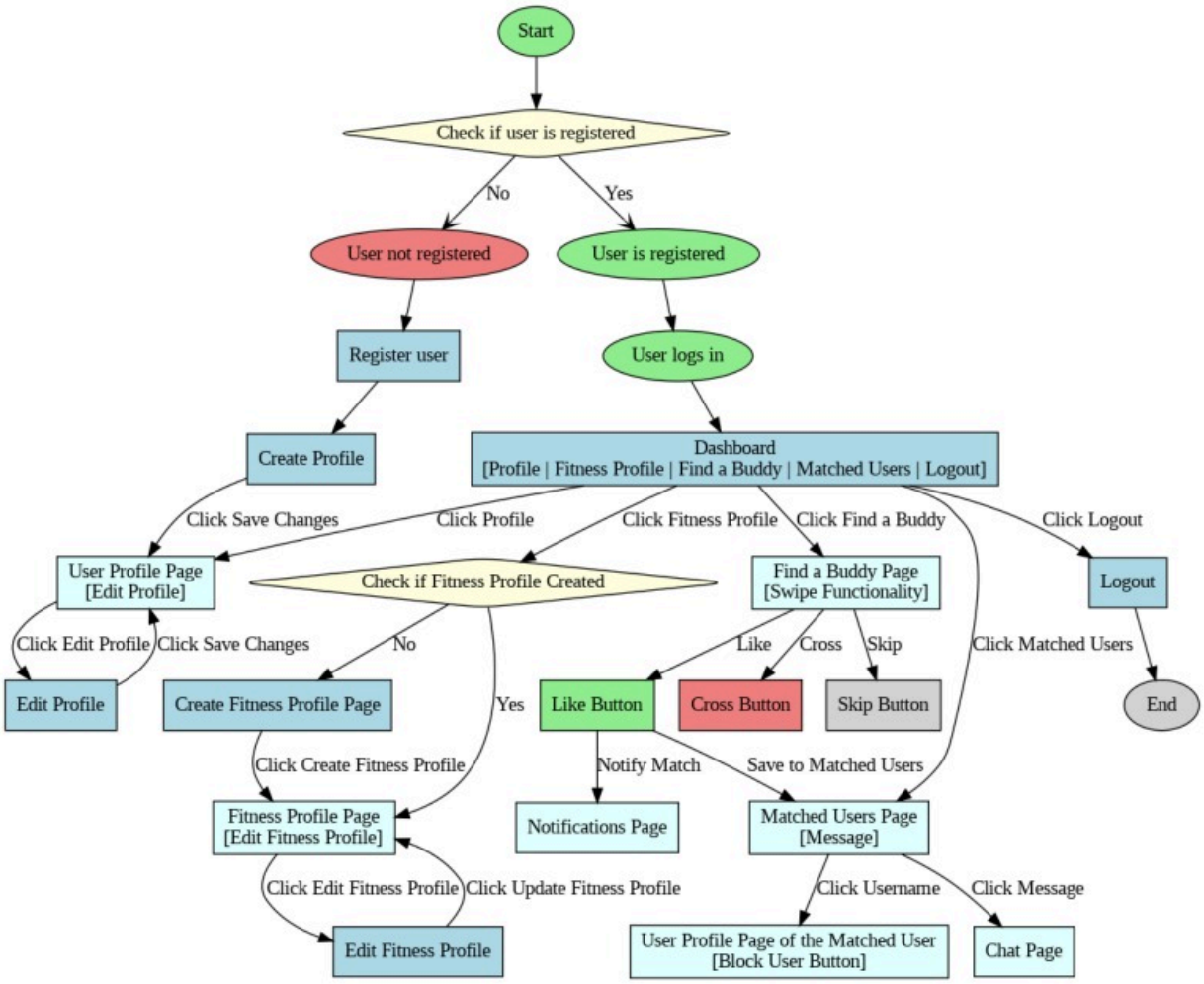
Sprint Backlog Items and Status:

The current sprint has no remaining backlog items. All sprint goals have been completed. The team has achieved everything planned for this sprint.

Burndown Chart:

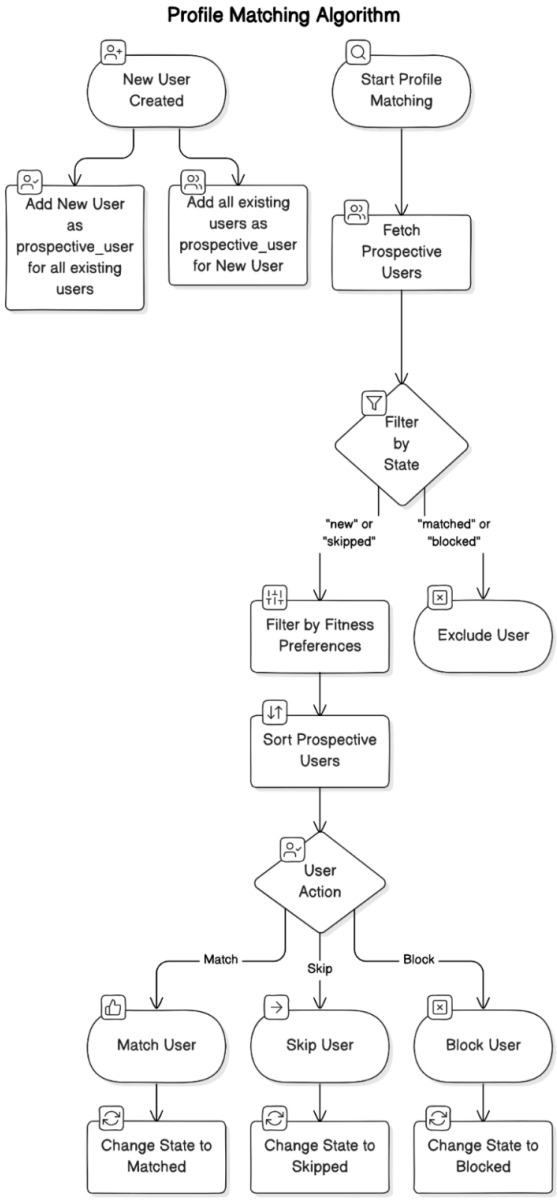


Design Diagram:



Profile Matching Algorithm Design Diagram:

Profile Matching Algorithm Design Diagram:



Documentation of Changes:

We did not incorporate any changes, and everything was implemented as per the plan.

Evaluation of Code and Test Quality:

Our project's overall quality has been rated A, reflecting a solid adherence to coding standards and best practices. We used **SimpleCov** and **CodeClimate** to evaluate the quality of both the code and the tests, focusing on aspects such as coverage, code smells, and style.

- **SimpleCov Score**: Currently, our test coverage, as measured by SimpleCov, **93.40%** for rspec and **92.91%** for cucumber scenarios. The coverage stands at **95.60%** combined for rspec and cucumber scenarios, which indicates that most of the code is well-tested. Our team initially wrote individual test cases and Cucumber scenarios for each feature, and all tests passed successfully when team members tested their respective features.
- **Code Smells**: During the analysis, **2 code smells** were detected. Addressing this will improve the overall readability and maintainability of the code. Even though the number of code smells is minimal, resolving it is essential to maintain a high-quality codebase and ensure long-term sustainability. Our project's overall quality has been rated **A**, reflecting a solid adherence to coding standards and best practices. We are committed to addressing the remaining code smell and increasing the test coverage in future iterations to ensure continued improvement in the project's quality.

Customer Meeting - Demo for Sprint 4 MVP:

Date: 13th November, 2024 & 20th November, 2024
Time: 10 am - 10.30 am
Place: Zoom Call

Client Meeting Summary:

In our recent client meeting, we presented the progress made during this sprint, focusing on the newly implemented chat feature, critical bug fixes, and enhancements to the user interface and experience. We also shared insights from the beta user testing, which began last Wednesday and provided valuable feedback for refining the application.

The client was particularly impressed with the chat feature, which is now fully integrated and allows seamless communication between matched users. They appreciated the message persistence functionality and the intuitive design, which enhances user engagement.

Additionally, they noted the improvements in application stability resulting from addressing key bugs, particularly those affecting profile management and user matching.

The refined UI/UX, including further optimizations for mobile responsiveness and consistent styling, was well-received. The client commended the improvements made to the navigation flow and visual consistency, which have significantly enhanced the app's usability.

Overall, the client was satisfied with the sprint outcomes, including the implementation of eleven user stories that contributed to critical improvements in communication features, profile management, and UI/UX. They expressed confidence in the app's readiness as it approaches the final testing phases and are eager to see the continued progress in the next sprint.

Bdd and Tdd:

Bdd:

conversation.feature

```
Feature: Conversations and Messages
  As a user
  I want to have a conversation with a matched user
  So that I can send and receive messages in real-time

  Background:
    Given I am logged in as a user
    And I have a matched user
    And I start a conversation with the matched user

   @javascript

  @javascript
  Scenario: Sending a message in a conversation
    Given I am on the conversation page
    When I create a new message with content "Hello, Gym Buddy!"
    Then the message should be saved to the database
    And I should see the message in the conversation
```

conversation_steps

```
Given("I have a matched user") do
  @matched_user = FactoryBot.create(:user, :complete_profile)
  @match = UserMatch.create!(
    user_id: @user.id,
    prospective_user_id: @matched_user.id,
    status: "matched"
  )
  UserMatch.create!(
    user_id: @matched_user.id,
    prospective_user_id: @user.id,
    status: "matched"
  )
end

Given("I start a conversation with the matched user") do
  @conversation = Conversation.create!(
    user1: @user,
    user2: @matched_user
  )
end

Given("I am on the conversation page") do
  visit conversation_path(@conversation, user_id: @user.id)

  # Debug output
  puts "\nPage details:"
  puts "Current path: #{current_path}"
  puts "Conversation ID in test: #{@conversation.id}"
  puts "Form action: #{find('#new-message-form')['action']}"
end
```

```ruby
When("I create a new message with content {string}") do |content|
  # Debug current state
  puts "\nCreating message:"
  puts "User ID: #{@user.id}"
  puts "Conversation ID: #{@conversation.id}"

  # Fill in the form
  within('#new-message-form') do
    fill_in 'message[content]', with: content
  end

  # Submit using JavaScript to handle Turbo
  page.execute_script(<<~JS)
    const form = document.getElementById('new-message-form');
    const formData = new FormData(form);

    fetch(form.action, {
      method: 'POST',
      body: formData,
      headers: {
        'Accept': 'application/json',
        'X-Requested-With': 'XMLHttpRequest'
      },
      credentials: 'same-origin'
    }).then(response => response.json())
    .then(data => {
      console.log('Success:', data);
    })
    .catch((error) => {
      console.error('Error:', error);
    });
  JS

  # Wait for AJAX completion
  sleep(2)
end
```

The purpose of this feature is to allow users to seamlessly engage in real-time conversations with their matched partners within the application. This functionality enhances the interactive and social experience for users, fostering better connections and collaboration between individuals who share fitness interests.

The implementation ensures that:

1. Matched Users: Users who have mutually expressed interest in each other can initiate a conversation.
2. Real-Time Messaging: Messages are sent and received instantaneously, providing a dynamic interaction.
3. Persistence: All messages are saved in the database, ensuring conversations are not lost.
4. User Experience: A clean and accessible interface displays the messages, making it easy for users to participate in conversations.

Scenario Covered: Real-Time Messaging in Conversations

Feature: Conversations and Messages

- Objective: As a user, I want to have a conversation with a matched user so that I can send and receive messages in real-time.
- Scenario: Sending a message in a conversation ensures that the message is saved in the database and immediately visible in the conversation thread.

dashboard_navigation.feature

```
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

  Background:
  Given the database is reset
  And a user exists with the following details:
    | field      | value              |
    | first_name | TestUser           |
    | last_name  | TestLastName       |
    | age        | 25                 |
    | gender     | female             |
    | email      | test@gmail.com     |
    | password   | dummy              |

  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 the text "Welcome to Jimmy"
```

dashboard_navigation_steps

```
Given("the database is reset") do
  User.destroy_all
end

Given("a user exists with the following details:") do |table|
  attributes = table.rows_hash
  @user = User.new(
    first_name: attributes["first_name"],
    last_name: attributes["last_name"],
    age: attributes["age"],
    gender: attributes["gender"],
    email: attributes["email"],
    uid: SecureRandom.hex(10),
    provider: "google"
  )
  if @user.save
    puts "User successfully created!"
  else
    puts "There were errors while saving the user: #{@user.errors.full_messages}"
  end
end

  Given("I am on the dashboard page") do
    @user = User.find_by(first_name: "TestUser")
    if @user.nil?
      raise "User not found"
    end
    visit dashboard_user_path(@user)
  end
```

```ruby
When("I enter the dashboard for the first time") do
  # First time?
end

Then("I should see the text {string}") do |expected_text|
  within("h1.display-3") do
    expect(page).to have_content(expected_text)
  end
end

When("I click the Profile icon") do
  click_link "Profile"
end

Then("I should be navigated to the User Profile Management page") do
  expect(current_path).to eq(profile_user_path(@user))
end
```

The purpose of this feature is to provide users with a seamless and intuitive dashboard experience where they can:

1. Understand the Application's Features:
   The dashboard displays a welcoming introduction, highlighting the application's capabilities and features, ensuring users feel informed and engaged upon logging in.
2. Navigate to Profile Management:
   A dedicated navigation option allows users to quickly access and update their personal information via the profile management page.

Scenario Covered: View Feature Introductions and Navigate to User Profile Management

This scenario ensures that:

1. Users entering the dashboard for the first time see an introductory message ("Welcome to Jimmy") clearly displayed.
2. Users can easily navigate to the profile management page by clicking the "Profile" icon, ensuring a smooth user experience.

notification.feature

```
Feature: Match Notification
  As a user
  I want to receive a notification when a mutual match occurs
  So that I am informed when another user has also matched with me

  Scenario: Notification creation after a mutual match
    Given I am logged in as a user
    And there is a prospective user
    And the prospective user has already matched with me
    When I match with the prospective user
    Then I should see a success message
    And a notification should be created for the user who matched

  Scenario: Mark a notification as read
    Given I am logged in as a user
    And I have an unread notification
    When I mark the notification as read
    Then the notification should be marked as read
    And I should see a JSON response confirming the change

  Scenario: Mark a notification as unread
    Given I am logged in as a user
    And I have a read notification
    When I mark the notification as unread
    Then the notification should be marked as unread
```

notification_steps

```
Given("the prospective user has already matched with me") do
    UserMatch.create!(user_id: @prospective_user.id, prospective_user_id: @user.id, status: "matched")
  end

  Then("a notification should be created for the user who matched") do
    notification_for_user = Notification.find_by(user: @user, matched_user: @prospective_user)

    expect(notification_for_user).to be_present
    expect(notification_for_user.read).to eq(false)
  end

Given("I have an unread notification") do
    @notification = FactoryBot.create(:notification, user: @user, read: false)
  end

  When("I mark the notification as read") do
    page.driver.post "/users/#{@user.id}/notifications/#{@notification.id}/mark_as_read.json"
  end

  Then("the notification should be marked as read") do
    @notification.reload
    expect(@notification.read).to be true
  end
```

```
  Then("I should see a JSON response confirming the change") do
    response_body = JSON.parse(page.body)
    expect(response_body["id"]).to eq(@notification.id)
    expect(response_body["read"]).to eq(true)
  end

Given("I have a read notification") do
    @notification = FactoryBot.create(:notification, user: @user, read: true)
  end

  When("I mark the notification as unread") do
    page.driver.post "/users/#{@user.id}/notifications/#{@notification.id}/mark_as_unread.json"
  end

  Then("the notification should be marked as unread") do
    @notification.reload
    expect(@notification.read).to be false
  end
```

The purpose of this feature is to ensure that users are promptly informed of mutual matches and can easily manage their match notifications. This feature aims to:

1. Notify Users of Mutual Matches:
   Automatically create a notification when two users mutually match, ensuring both are aware of the connection.
2. Enhance User Engagement:
   Keep users engaged by making them aware of new connections in real-time.
3. Enable Notification Management:
   Allow users to manage their notifications by marking them as read or unread, ensuring clarity and organization in the notification system.

Scenarios Covered:

1. Notification Creation After a Mutual Match

This scenario ensures that:

- A notification is generated when a mutual match occurs.
- Users receive a success message confirming the match.

2. Mark a Notification as Read

This scenario ensures that:

- Users can mark notifications as read, indicating that they have seen the notification.
- The change is confirmed with a JSON response.

3. Mark a Notification as Unread

This scenario ensures that:

- Users can revert a notification to an unread state for better tracking.

TDD:

conversation_controller_spec

```
require 'rails_helper'

RSpec.describe ConversationsController, type: :controller do
  let(:user) { FactoryBot.create(:user, :complete_profile) }
  let(:other_user) { FactoryBot.create(:user, :complete_profile) }

  before do
    # Simulate user login
    session[:user_id] = user.id
    allow(controller).to receive(:current_user).and_return(user)

    # Create mutual "matched" relationship if needed by app logic
    UserMatch.create!(user_id: user.id, prospective_user_id: other_user.id, status: "matched")
    UserMatch.create!(user_id: other_user.id, prospective_user_id: user.id, status: "matched")
  end

  describe "GET #show" do
    context "when viewing a conversation with a matched user" do
      before do
        get :show, params: { user_id: user.id, id: other_user.id }
      end

      it "assigns the current user and other user" do
        expect(assigns(:current_user)).to eq(user)
        expect(assigns(:other_user)).to eq(other_user)
      end
```

```
      it "finds or creates a conversation between the users" do
        conversation = Conversation.between(user.id, other_user.id).first
        expect(assigns(:conversation)).to eq(conversation)
      end

      it "assigns messages in ascending order of creation to @messages" do
        conversation = assigns(:conversation)
        message1 = FactoryBot.create(:message, conversation: conversation, user: user, content: "Hello!")
        message2 = FactoryBot.create(:message, conversation: conversation, user: other_user, content: "Hi!")

        get :show, params: { user_id: user.id, id: other_user.id }
        expect(assigns(:messages)).to eq([ message1, message2 ])
      end

      it "initializes a new message for the form" do
        expect(assigns(:message)).to be_a_new(Message)
      end
    end
  end
end
```

The purpose of this test suite is to validate the behavior and functionality of the ConversationsController for various scenarios involving conversation interactions. It ensures that users can view and interact with their conversations seamlessly, verifying the following key aspects:

1. User and Match Validation:
    - Confirms that only conversations with matched users are accessible.
2. Conversation Management:
    - Ensures the controller finds or creates the appropriate conversation between the current user and the matched user.
3. Message Assignment and Order:
    - Verifies that messages associated with the conversation are retrieved in ascending order of creation.
4. Message Initialization:
    - Check that a new Message object is initialized for use in the conversation form.

messages_controller_spec

```ruby
require 'rails_helper'

RSpec.describe MessagesController, type: :controller do
  let(:user) { FactoryBot.create(:user, :complete_profile) }
  let(:matched_user) { FactoryBot.create(:user, :complete_profile) }
  let(:conversation) { Conversation.between(user.id, matched_user.id).first_or_create!(user1: user, user2: matched_user) }

  before do
    # Simulate user login
    session[:user_id] = user.id
    allow(controller).to receive(:current_user).and_return(user)

    # Create mutual "matched" relationship
    UserMatch.create!(user_id: user.id, prospective_user_id: matched_user.id, status: "matched")
    UserMatch.create!(user_id: matched_user.id, prospective_user_id: user.id, status: "matched")
  end

  describe "POST #create" do
    let(:valid_message_params) { { conversation_id: conversation.id, message: { content: "Hello, Gym Buddy!" } } }
    let(:invalid_message_params) { { conversation_id: conversation.id, message: { content: "" } } }

    before do
      request.env["HTTP_ACCEPT"] = "application/json"
    end
```

```
    context "with valid params" do
      it "creates a new message in the conversation" do
        expect {
          post :create, params: valid_message_params, format: :json
        }.to change(conversation.messages, :count).by(1)
      end

      it "returns a successful JSON response" do
        post :create, params: valid_message_params, format: :json
        expect(response).to have_http_status(:created)
        expect(JSON.parse(response.body)).to eq("status" => "success")
      end
    end

    context "with invalid params" do
      it "does not create a new message" do
        expect {
          post :create, params: invalid_message_params, format: :json
        }.not_to change(conversation.messages, :count)
      end

      it "returns an error JSON response" do
        post :create, params: invalid_message_params, format: :json
        expect(response).to have_http_status(:unprocessable_entity)
        expect(JSON.parse(response.body)).to have_key("error")
      end
    end
  end
end
```

The purpose of this test suite is to validate the behavior of the MessagesController, particularly the create action, which handles message creation within a conversation. This suite ensures that the controller adheres to expected behavior, processes valid and invalid inputs correctly, and provides appropriate responses in JSON format.

Key Aspects Tested:

1. Message Creation with Valid Parameters:

    - Verifies that a message is successfully created when the input parameters are valid.
    - Confirms that the appropriate JSON response is returned.
2. Message Creation with Invalid Parameters:
    - Ensures that invalid input does not result in a new message being created.
    - Verifies that the controller responds with an appropriate error status and message.

notification_controller_spec

```
  describe 'POST #mark_as_read' do
    it 'marks the notification as read' do
      post :mark_as_read, params: { user_id: user.id, id: notification.id }, format: :json
      expect(response).to have_http_status(:ok)
      expect(JSON.parse(response.body)['read']).to be true
    end

    it 'returns unprocessable_entity when update fails' do
      allow_any_instance_of(Notification).to receive(:update).and_return(false)
      post :mark_as_read, params: { user_id: user.id, id: notification.id }, format: :json
      expect(response).to have_http_status(:unprocessable_entity)
      expect(JSON.parse(response.body)['message']).to eq('Failed to mark notification as read.')
    end
  end

  describe 'POST #mark_as_unread' do
    it 'marks the notification as unread' do
      notification.update(read: true)
      post :mark_as_unread, params: { user_id: user.id, id: notification.id }, format: :json
      expect(response).to have_http_status(:ok)
      expect(JSON.parse(response.body)['read']).to be false
    end

    it 'returns unprocessable_entity when update fails' do
      allow_any_instance_of(Notification).to receive(:update).and_return(false)
      post :mark_as_unread, params: { user_id: user.id, id: notification.id }, format: :json
      expect(response).to have_http_status(:unprocessable_entity)
      expect(JSON.parse(response.body)['message']).to eq('Failed to mark notification as unread.')
    end
  end
end

context 'when user is not logged in' do
  before { session[:user_id] = nil }
```

```
    describe 'GET #index' do
      it 'redirects to the welcome page' do
        get :index, params: { user_id: user.id }
        expect(response).to have_http_status(:found)
        expect(response).to redirect_to('/welcome/index')
      end
    end

    describe 'POST #mark_as_read' do
      it 'redirects to the welcome page' do
        post :mark_as_read, params: { user_id: user.id, id: notification.id }, format: :json
        expect(response).to have_http_status(:found)
        expect(response).to redirect_to('/welcome/index')
      end
    end

    describe 'POST #mark_as_unread' do
      it 'redirects to the welcome page' do
        post :mark_as_unread, params: { user_id: user.id, id: notification.id }, format: :json
        expect(response).to have_http_status(:found)
        expect(response).to redirect_to('/welcome/index')
      end
    end
  end
end
```

The purpose of this test suite is to validate the functionality and behavior of the NotificationsController. The tests ensure that notifications are managed appropriately, covering scenarios for both logged-in and logged-out users. Key features tested include retrieving notifications, marking them as read/unread, and handling unauthorized access.

Key Features Tested:

1. Notification Retrieval

- Ensures that a logged-in user can retrieve their notifications in the correct order (most recent first).
- Confirms that unauthorized access redirects users to the welcome page.

2. Marking Notifications as Read

- Validates that a notification can be marked as read by a logged-in user.
- Checks for proper error handling when marking as read fails.

3. Marking Notifications as Unread

- Ensures that a read notification can be marked as unread.
- Verifies error handling when marking as unread fails.

4. Unauthorized Access

- Confirms that unauthenticated users are redirected to the welcome page for all actions.