Sprint 2 Retrospective - Project Jimmy

Links to our GitHub Repo, Pivotal Tracker, and Slack workspace:

- **Github Repo -** https://github.com/tamu-edu-students/jimmy-gym-buddy-finder
- **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:

7th October 2024 to 20th October 2024

Information about team member and contributions:

Team Member	**Contribution**	**Tasks**
Kuan-Ru Huang	15%	● UI Template for Matching with Gym Buddies ● Implement a mock testing framework for Authentication Modules
Wei-Chien Cheng	15%	● User Fitness Profile Creation and Management ● Test Integration for Dashboard and User Profile Management
Yash Phatak	15%	● Revamped UI Design for Buddy Finding App ● Fix User Profile Picture Visibility Bug in Production
Mrunmay Deshmukh	13%	● Mobile Responsive User Interface

Kushal Lahoti	15%	● Revamped UI Design for Buddy Finding App ● Fix User Profile Picture Visibility Bug in Production
Barry Liu	15%	● UI Template for Matching with Gym Buddies ● Implement a mock testing framework for Authentication Modules
ChuanHsin Wang	12%	● User Fitness Profile Creation and Management ● Test Integration for Dashboard and User Profile Management

Sprint Goal:

In this sprint, our goal is to complete seven user stories, which encompass tasks such as UI development for key features, bug fixes, and the creation of testing modules. So far, we have successfully implemented the authentication functionality and user profile setup. Following that, we conducted a demo with the client to present the current progress of the application. The client provided feedback and additional requirements, particularly related to the UI's styling and responsiveness.

We also discussed the client's expectations for the core functionality of the buddy matching feature during the sprint demo call, especially in terms of its design and user experience. Based on this feedback, we will be making the necessary adjustments and adding new features. In this sprint, we aim to revamp the UI design and ensure it follows a mobile-friendly structure. Additionally, we plan to integrate a fitness profile for users, capturing details like activity preferences, location, and gender. We will also work on developing the UI template for the buddy matching feature.

During the last deployment, we identified a bug where user-uploaded profile pictures were not visible on the Heroku production environment. This issue arises because Heroku uses ephemeral file storage, which means uploaded data is erased whenever the server (dyno) restarts. To address this, we will need to implement persistent file storage, either on Heroku or through an external platform like AWS, and integrate it with our current setup.

Finally, there were some tasks left in the previous sprint, creating a small backlog. We plan to address those items in this sprint and ensure that all outstanding tasks are completed by the sprint's end.

User stories: A total of 7 user-stories have been added to the sprint plan.

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: 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: 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 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.

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: 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: Implement of 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.

Sprint Achievements:

This sprint has made significant progress, with a total of 7 user stories added and key features implemented across various areas.

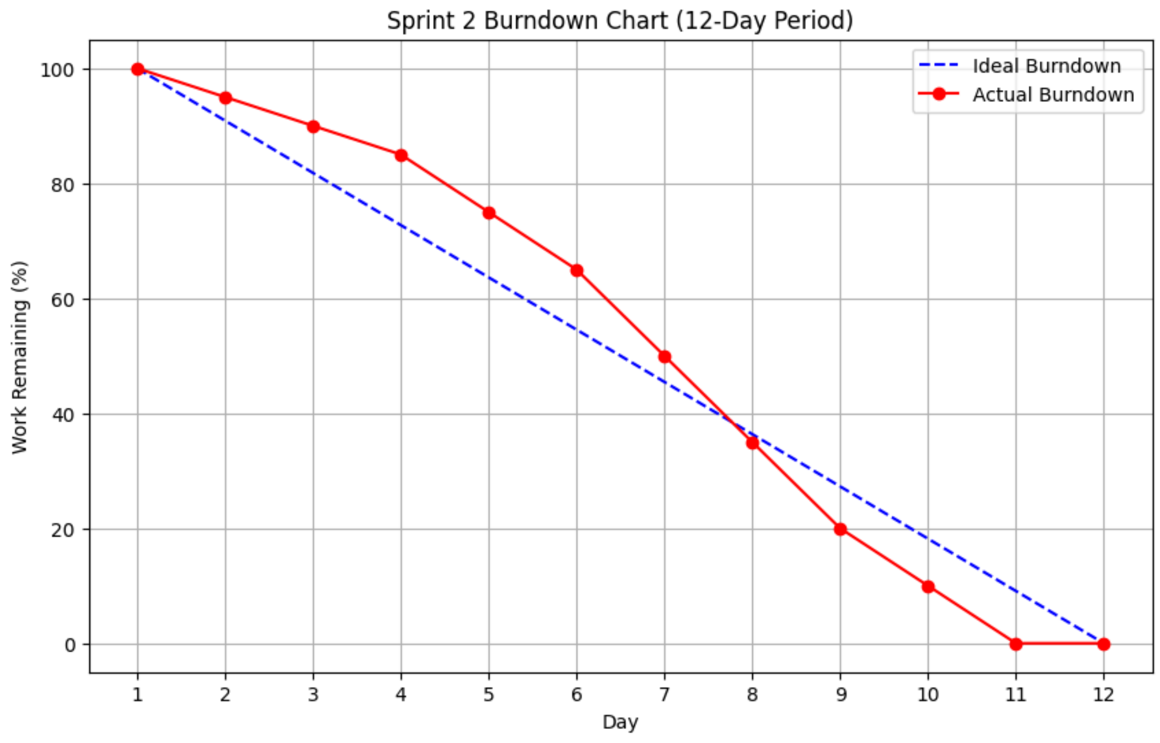
1. **Mobile Responsive User Interface**
   - The UI has been updated to be mobile-friendly, ensuring optimal user experience on smaller screens. Scenarios covered include responsive layout adjustments and alignment with the client's design requirements for mobile devices.
2. **UI Template for matching with Gym Buddies**
   - Introduced a new UI template for finding gym buddies, allowing users to search profiles based on workout type, location, and experience level. The implementation includes enabling users to swipe to match or unmatch with potential gym partners.
3. **User Fitness Profile Management**
   - Developed functionality for creating and managing fitness profiles, including specifying personal preferences, reviewing, and editing fitness-related information. This feature aims to enhance user connections with workout partners.
4. **Revamped Buddy-Finding App UI**
   - A fresh and engaging UI design has been implemented, featuring new styling elements and a vibrant color scheme. User feedback is being gathered to identify further improvements.
5. **Mock Testing Framework for Authentication**
   - Set up a mock testing framework for authentication modules, covering scenarios like simulating third-party authentication, handling failed authentication, and testing session persistence for authenticated users.
6. **Dashboard and User Profile Management Testing**

    - Integrated mock testing for dashboard and profile management, allowing validation of user access and functionality without relying on live authentication. Scenarios include profile updates, navigation, and access denial for unauthenticated users.
7. **Bug Fix for Profile Picture Visibility in Production**
    - Addressed the issue of profile picture disappearance in the deployed version, implementing a persistent image storage solution to ensure reliable display across dyno restarts. Configuration verification for the external storage service has been completed.

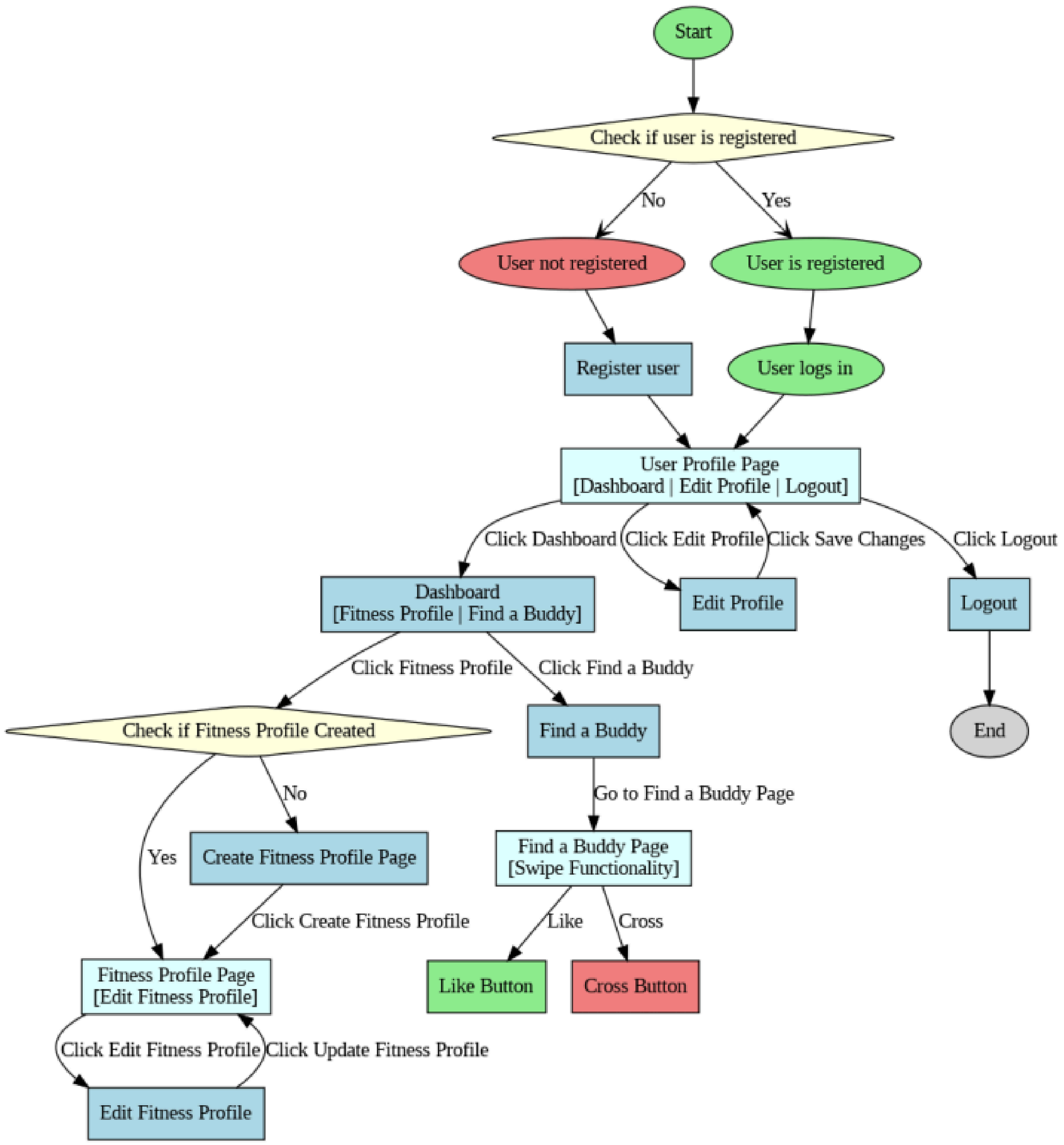
Sprint Backlog Items and Status:

The team successfully addressed the test coverage backlog from the last sprint, improving the overall quality of the test suite. For the current sprint, a new backlog item has been added to fix a UI-related bug in the ProfileSwipe feature. This issue will be prioritized and resolved as part of this sprint to ensure a seamless user experience. The team is on track to complete all planned tasks while accommodating the additional bug fix.

Burndown:



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, stands at 96.92%, 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, **1 code smell** was 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.

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 1 MVP:

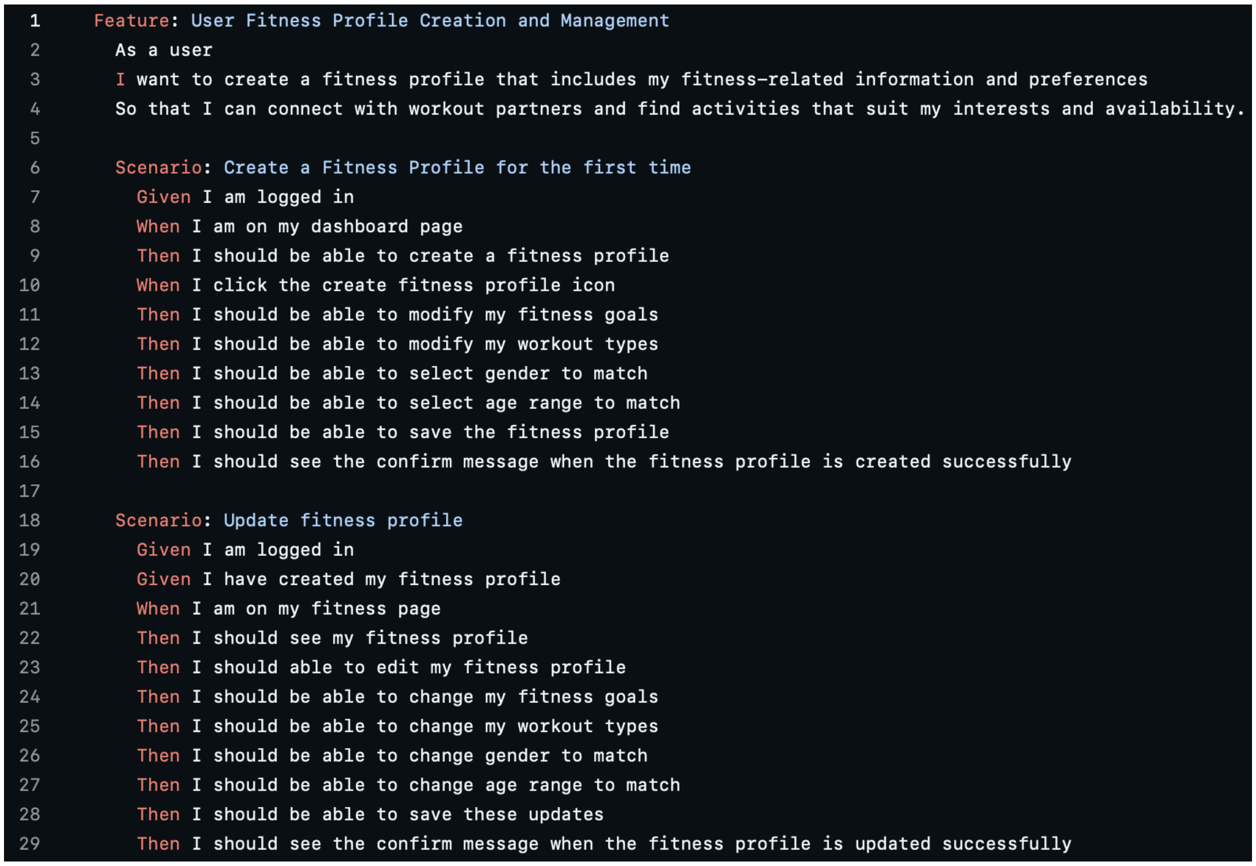
Date: 23rd October, 2024
Time: 10 am - 10.30 am
Place: Zoom Call

In the client meeting, we showcased the user interface (UI) and walked through key features such as the profile swipe and fitness profile functionalities. The client provided feedback and suggested incorporating a "block" feature to allow users to manage unwanted interactions more effectively.

Bdd and Tdd:

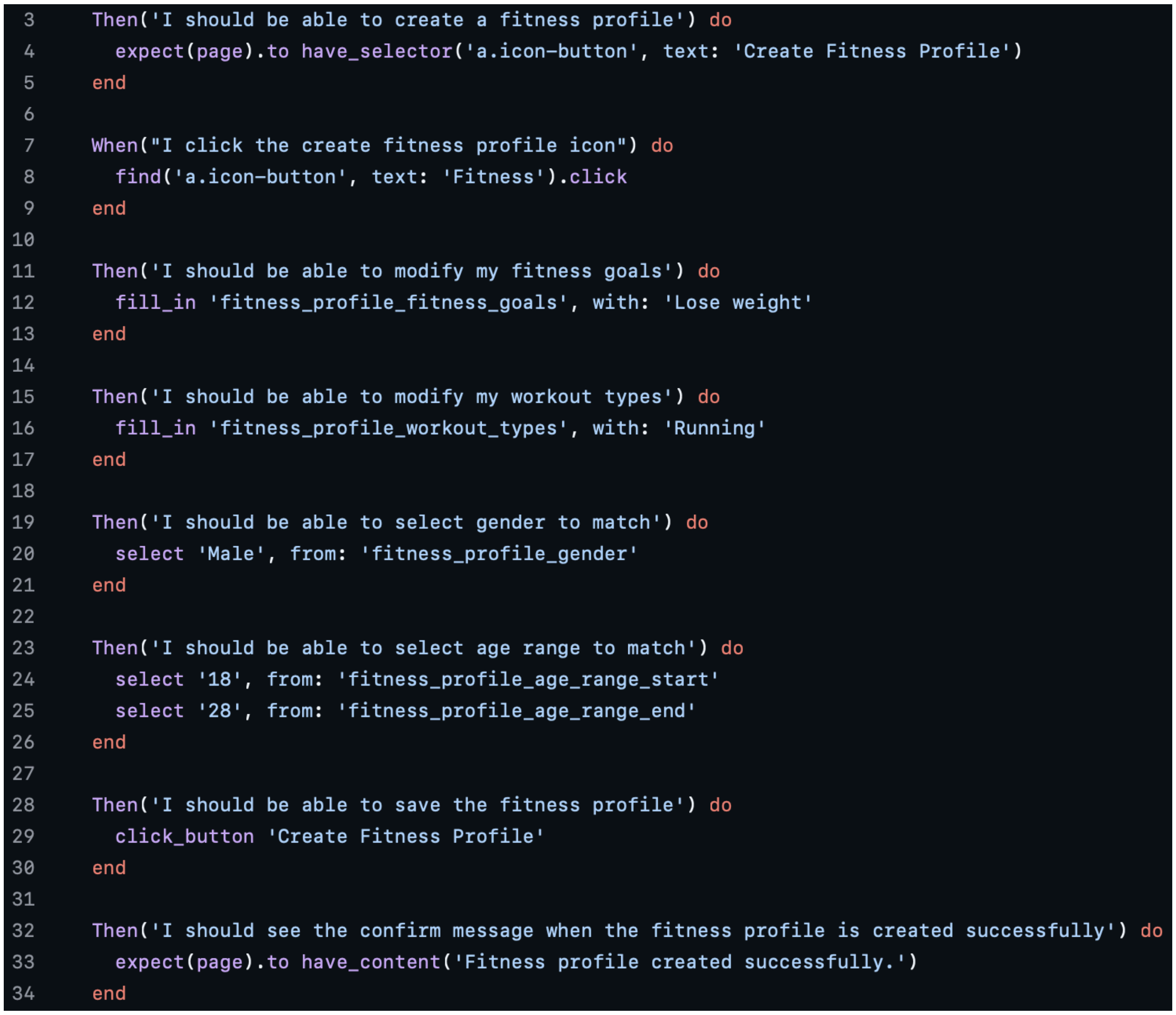
Bdd:

fitness_profile_management.feature

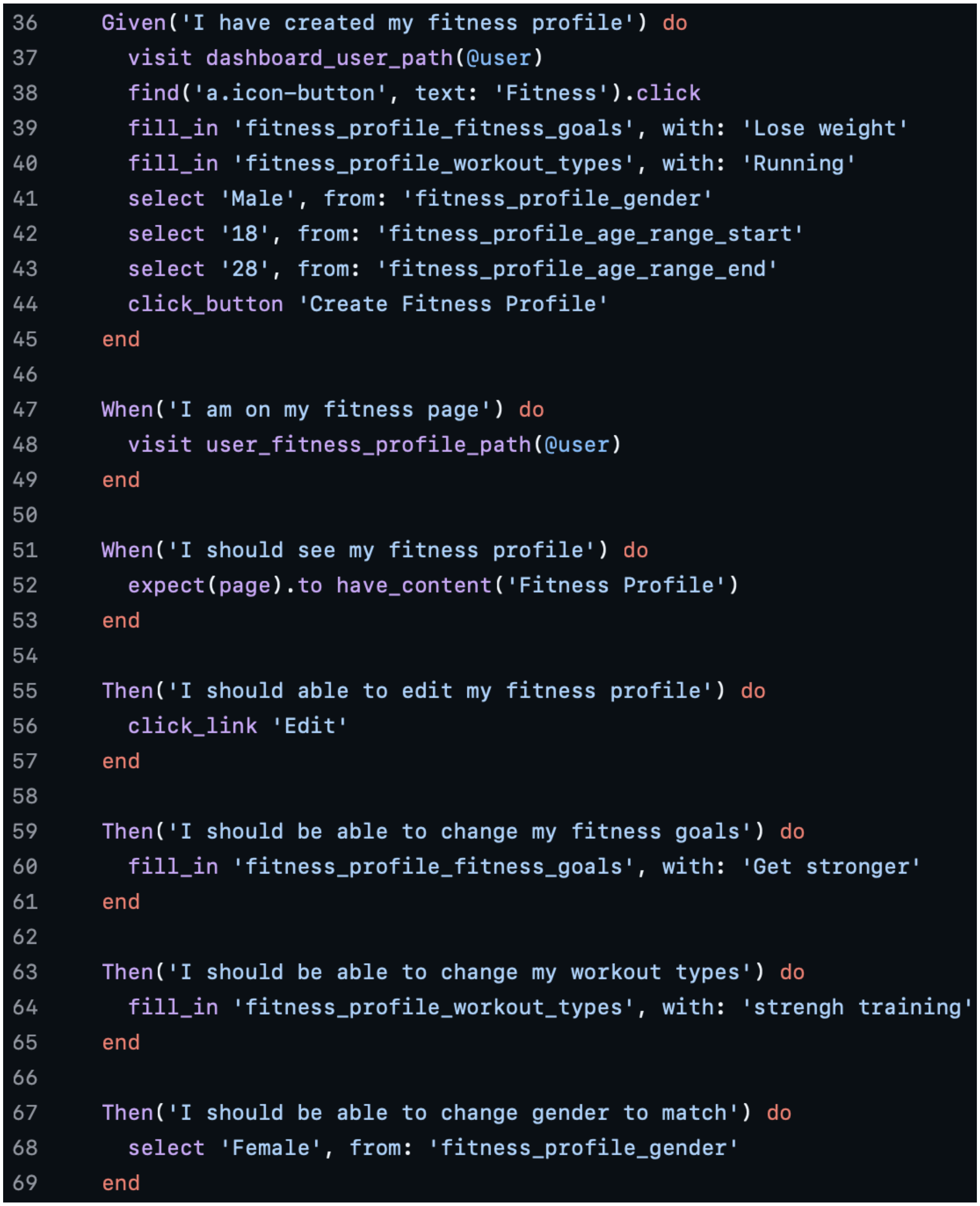


```
Feature: User Fitness Profile Creation and Management
  As a user
  I want to create a fitness profile that includes my fitness-related information and preferences
  So that I can connect with workout partners and find activities that suit my interests and availability.

  Scenario: Create a Fitness Profile for the first time
    Given I am logged in
    When I am on my dashboard page
    Then I should be able to create a fitness profile
    When I click the create fitness profile icon
    Then I should be able to modify my fitness goals
    Then I should be able to modify my workout types
    Then I should be able to select gender to match
    Then I should be able to select age range to match
    Then I should be able to save the fitness profile
    Then I should see the confirm message when the fitness profile is created successfully

  Scenario: Update fitness profile
    Given I am logged in
    Given I have created my fitness profile
    When I am on my fitness page
    Then I should see my fitness profile
    Then I should able to edit my fitness profile
    Then I should be able to change my fitness goals
    Then I should be able to change my workout types
    Then I should be able to change gender to match
    Then I should be able to change age range to match
    Then I should be able to save these updates
    Then I should see the confirm message when the fitness profile is updated successfully
```

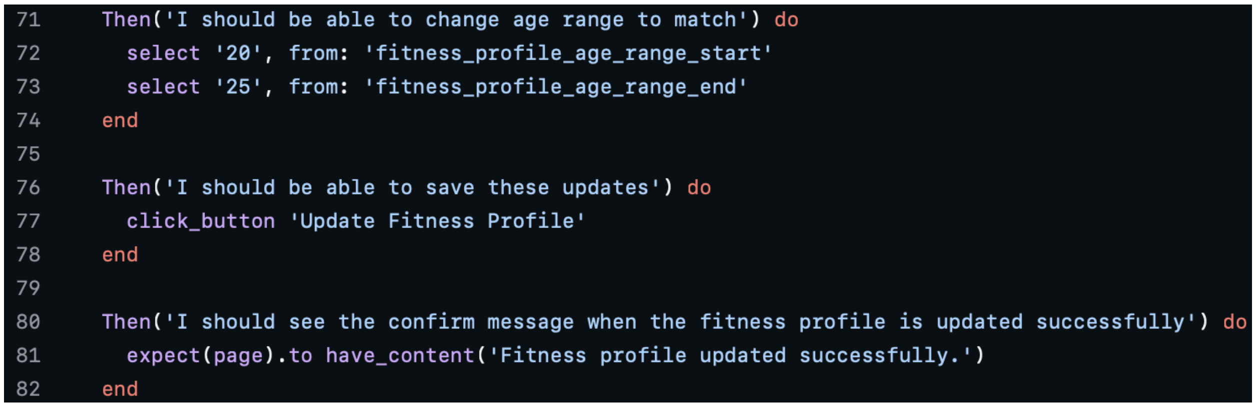
fitness_profile_management_steps



```ruby
Then('I should be able to create a fitness profile') do
  expect(page).to have_selector('a.icon-button', text: 'Create Fitness Profile')
end

When("I click the create fitness profile icon") do
  find('a.icon-button', text: 'Fitness').click
end

Then('I should be able to modify my fitness goals') do
  fill_in 'fitness_profile_fitness_goals', with: 'Lose weight'
end

Then('I should be able to modify my workout types') do
  fill_in 'fitness_profile_workout_types', with: 'Running'
end

Then('I should be able to select gender to match') do
  select 'Male', from: 'fitness_profile_gender'
end

Then('I should be able to select age range to match') do
  select '18', from: 'fitness_profile_age_range_start'
  select '28', from: 'fitness_profile_age_range_end'
end

Then('I should be able to save the fitness profile') do
  click_button 'Create Fitness Profile'
end

Then('I should see the confirm message when the fitness profile is created successfully') do
  expect(page).to have_content('Fitness profile created successfully.')
end
```



```ruby
Given('I have created my fitness profile') do
  visit dashboard_user_path(@user)
  find('a.icon-button', text: 'Fitness').click
  fill_in 'fitness_profile_fitness_goals', with: 'Lose weight'
  fill_in 'fitness_profile_workout_types', with: 'Running'
  select 'Male', from: 'fitness_profile_gender'
  select '18', from: 'fitness_profile_age_range_start'
  select '28', from: 'fitness_profile_age_range_end'
  click_button 'Create Fitness Profile'
end

When('I am on my fitness page') do
  visit user_fitness_profile_path(@user)
end

When('I should see my fitness profile') do
  expect(page).to have_content('Fitness Profile')
end

Then('I should able to edit my fitness profile') do
  click_link 'Edit'
end

Then('I should be able to change my fitness goals') do
  fill_in 'fitness_profile_fitness_goals', with: 'Get stronger'
end

Then('I should be able to change my workout types') do
  fill_in 'fitness_profile_workout_types', with: 'strengh training'
end

Then('I should be able to change gender to match') do
  select 'Female', from: 'fitness_profile_gender'
end
```



```ruby
Then('I should be able to change age range to match') do
  select '20', from: 'fitness_profile_age_range_start'
  select '25', from: 'fitness_profile_age_range_end'
end

Then('I should be able to save these updates') do
  click_button 'Update Fitness Profile'
end

Then('I should see the confirm message when the fitness profile is updated successfully') do
  expect(page).to have_content('Fitness profile updated successfully.')
end
```

The purpose of this feature is to allow users to create, view, and update their fitness profiles, which include their fitness goals, workout preferences, gender, and age range. This functionality enables users to tailor their profiles, making it easier to connect with suitable workout partners and activities based on their interests.

Scenarios Covered:

Create a Fitness Profile for the First Time:

This scenario walks the user through the process of creating a new fitness profile, including modifying fitness goals, selecting workout types, gender, and age range, and then saving the profile.

After successfully creating the profile, the user should see a confirmation message.

Update an Existing Fitness Profile:

This scenario allows a user who has already created a fitness profile to view and edit their existing information.

Users can change fitness goals, workout types, gender, and age range, and upon saving, they should see a confirmation message indicating that the profile has been updated successfully.

Purpose of Step Definitions:

The provided step definitions define the behavior-driven tests (BDD) for the fitness profile feature. These steps ensure that the user interface functions correctly in allowing users to create and update their fitness profiles.

Step Breakdown:

Fitness Profile Creation Steps:

Checks for the presence of the "Create Fitness Profile" button.

Simulates clicking the icon to create a profile and filling out fitness goals, workout types, gender, and age range.

Simulates saving the profile and confirms success via a confirmation message.

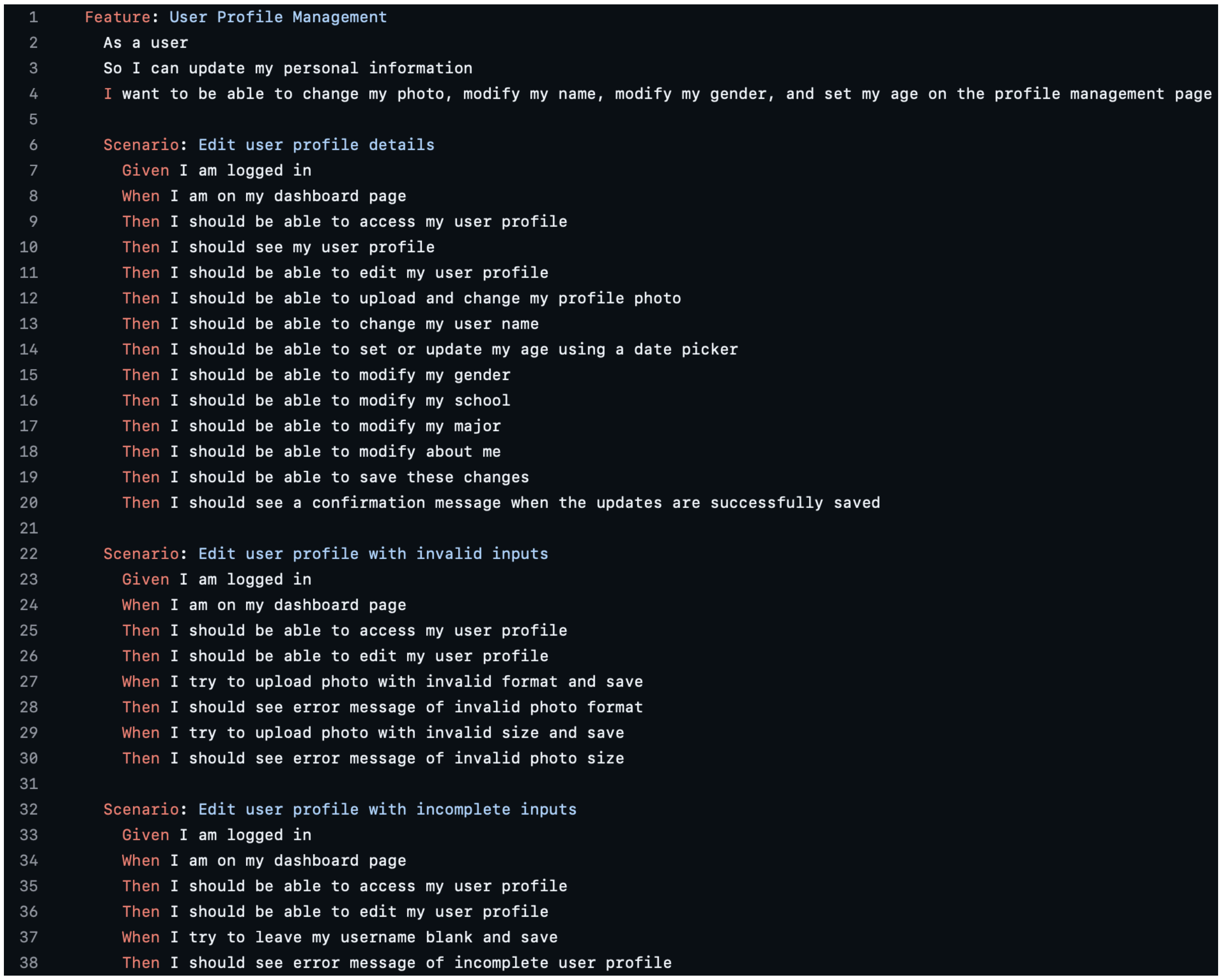
Fitness Profile Update Steps:

Prepares the test environment by simulating a user having already created a fitness profile.

Simulates navigating to the fitness page, editing the profile, and changing fitness-related information.

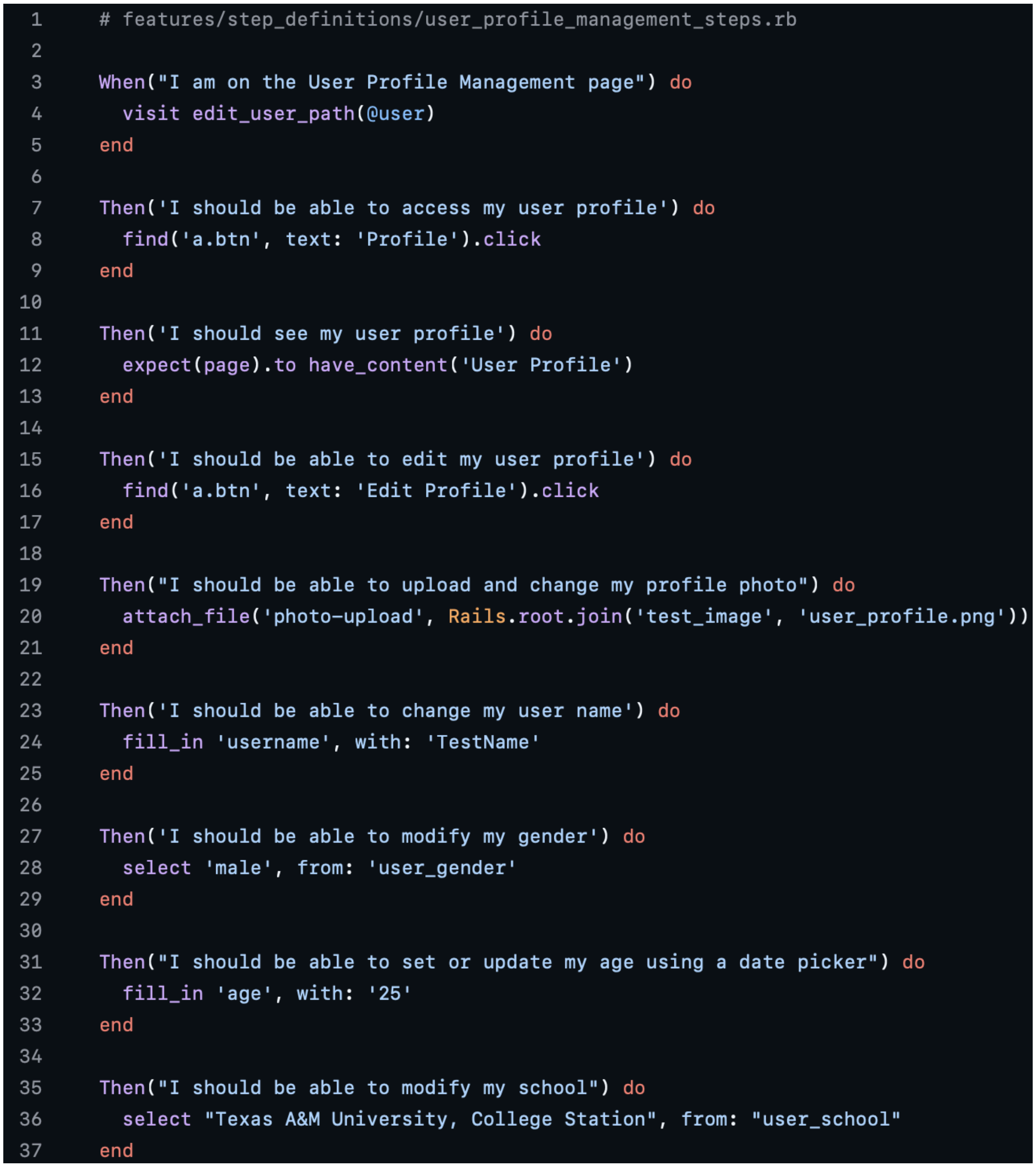
Confirms that the updated profile is saved successfully, and the user receives an appropriate confirmation message.

user_page_management.feature

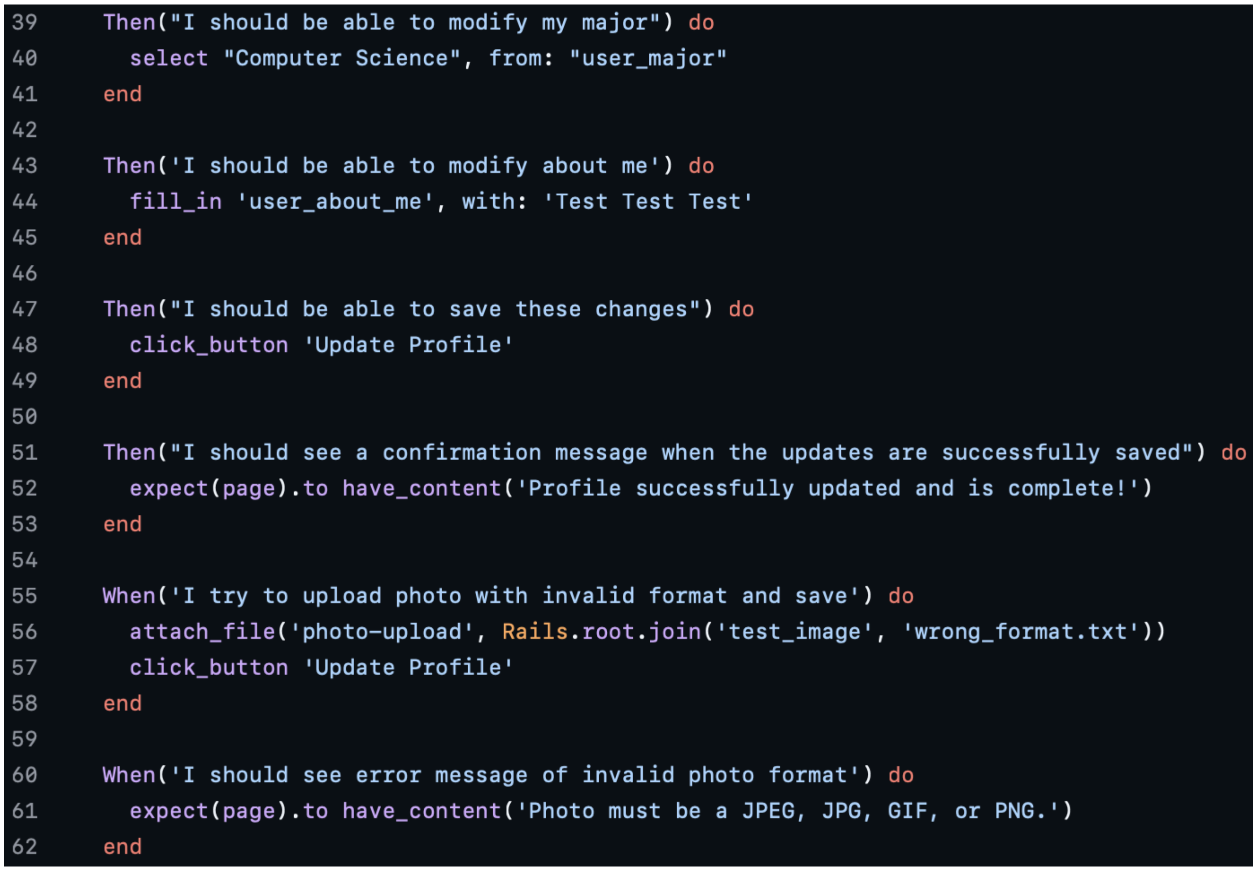


```
Feature: User Profile Management
  As a user
  So I can update my personal information
  I want to be able to change my photo, modify my name, modify my gender, and set my age on the profile management page

  Scenario: Edit user profile details
    Given I am logged in
    When I am on my dashboard page
    Then I should be able to access my user profile
    Then I should see my user profile
    Then I should be able to edit my user profile
    Then I should be able to upload and change my profile photo
    Then I should be able to change my user name
    Then I should be able to set or update my age using a date picker
    Then I should be able to modify my gender
    Then I should be able to modify my school
    Then I should be able to modify my major
    Then I should be able to modify about me
    Then I should be able to save these changes
    Then I should see a confirmation message when the updates are successfully saved

  Scenario: Edit user profile with invalid inputs
    Given I am logged in
    When I am on my dashboard page
    Then I should be able to access my user profile
    Then I should be able to edit my user profile
    When I try to upload photo with invalid format and save
    Then I should see error message of invalid photo format
    When I try to upload photo with invalid size and save
    Then I should see error message of invalid photo size

  Scenario: Edit user profile with incomplete inputs
    Given I am logged in
    When I am on my dashboard page
    Then I should be able to access my user profile
    Then I should be able to edit my user profile
    When I try to leave my username blank and save
    Then I should see error message of incomplete user profile
```

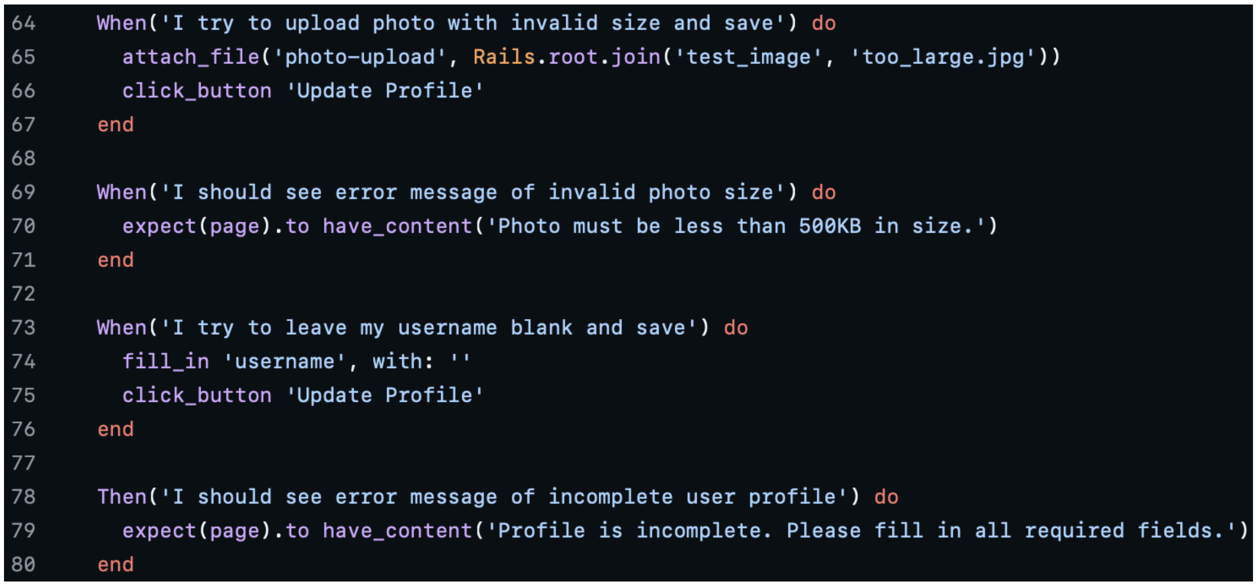
user_profile_management_steps



```ruby
# features/step_definitions/user_profile_management_steps.rb

When("I am on the User Profile Management page") do
  visit edit_user_path(@user)
end

Then('I should be able to access my user profile') do
  find('a.btn', text: 'Profile').click
end

Then('I should see my user profile') do
  expect(page).to have_content('User Profile')
end

Then('I should be able to edit my user profile') do
  find('a.btn', text: 'Edit Profile').click
end

Then("I should be able to upload and change my profile photo") do
  attach_file('photo-upload', Rails.root.join('test_image', 'user_profile.png'))
end

Then('I should be able to change my user name') do
  fill_in 'username', with: 'TestName'
end

Then('I should be able to modify my gender') do
  select 'male', from: 'user_gender'
end

Then("I should be able to set or update my age using a date picker") do
  fill_in 'age', with: '25'
end

Then("I should be able to modify my school") do
  select "Texas A&M University, College Station", from: "user_school"
end
```



```ruby
  Then("I should be able to modify my major") do
    select "Computer Science", from: "user_major"
  end

  Then('I should be able to modify about me') do
    fill_in 'user_about_me', with: 'Test Test Test'
  end

  Then("I should be able to save these changes") do
    click_button 'Update Profile'
  end

  Then("I should see a confirmation message when the updates are successfully saved") do
    expect(page).to have_content('Profile successfully updated and is complete!')
  end

  When('I try to upload photo with invalid format and save') do
    attach_file('photo-upload', Rails.root.join('test_image', 'wrong_format.txt'))
    click_button 'Update Profile'
  end

  When('I should see error message of invalid photo format') do
    expect(page).to have_content('Photo must be a JPEG, JPG, GIF, or PNG.')
  end
```



```ruby
  When('I try to upload photo with invalid size and save') do
    attach_file('photo-upload', Rails.root.join('test_image', 'too_large.jpg'))
    click_button 'Update Profile'
  end

  When('I should see error message of invalid photo size') do
    expect(page).to have_content('Photo must be less than 500KB in size.')
  end

  When('I try to leave my username blank and save') do
    fill_in 'username', with: ''
    click_button 'Update Profile'
  end

  Then('I should see error message of incomplete user profile') do
    expect(page).to have_content('Profile is incomplete. Please fill in all required fields.')
  end
```

The **User Profile Management** feature allows users to update their personal information, such as their profile photo, name, gender, age, school, major, and about me section. It includes validation checks to ensure the correctness of inputs, such as photo format, size, and mandatory fields.

Scenarios Covered:

Edit User Profile Details:

This scenario walks through updating various parts of a user's profile, including changing their profile photo, username, age, gender, school, major, and personal description.

After making updates, the user should see a confirmation message indicating successful changes.

Edit User Profile with Invalid Inputs:

This scenario checks for errors when the user attempts to upload a profile photo with an invalid format (e.g., text file) or an invalid size (e.g., larger than allowed).

Upon encountering these invalid inputs, the user should see specific error messages that describe the issue.

Edit User Profile with Incomplete Inputs:

This scenario ensures that required fields (such as username) cannot be left blank, and that the user receives an error message indicating that their profile is incomplete if they attempt to save with missing information.

Purpose of Step Definitions:

The provided step definitions outline the behavior-driven tests (BDD) for the user profile management feature. These steps help ensure that the profile management functionality is working correctly, covering cases of valid, invalid, and incomplete inputs.

Step Breakdown:

Profile Access and Editing:

Navigates to the user's profile page, verifies that the profile details are visible, and allows the user to enter edit mode.

Profile Updates:

Simulates uploading a new profile photo, changing the username, updating age using a date picker, and modifying other fields like gender, school, major, and the "about me" section.

After saving, the test confirms that the updates were successful by checking for a confirmation message.

Invalid Inputs Handling:

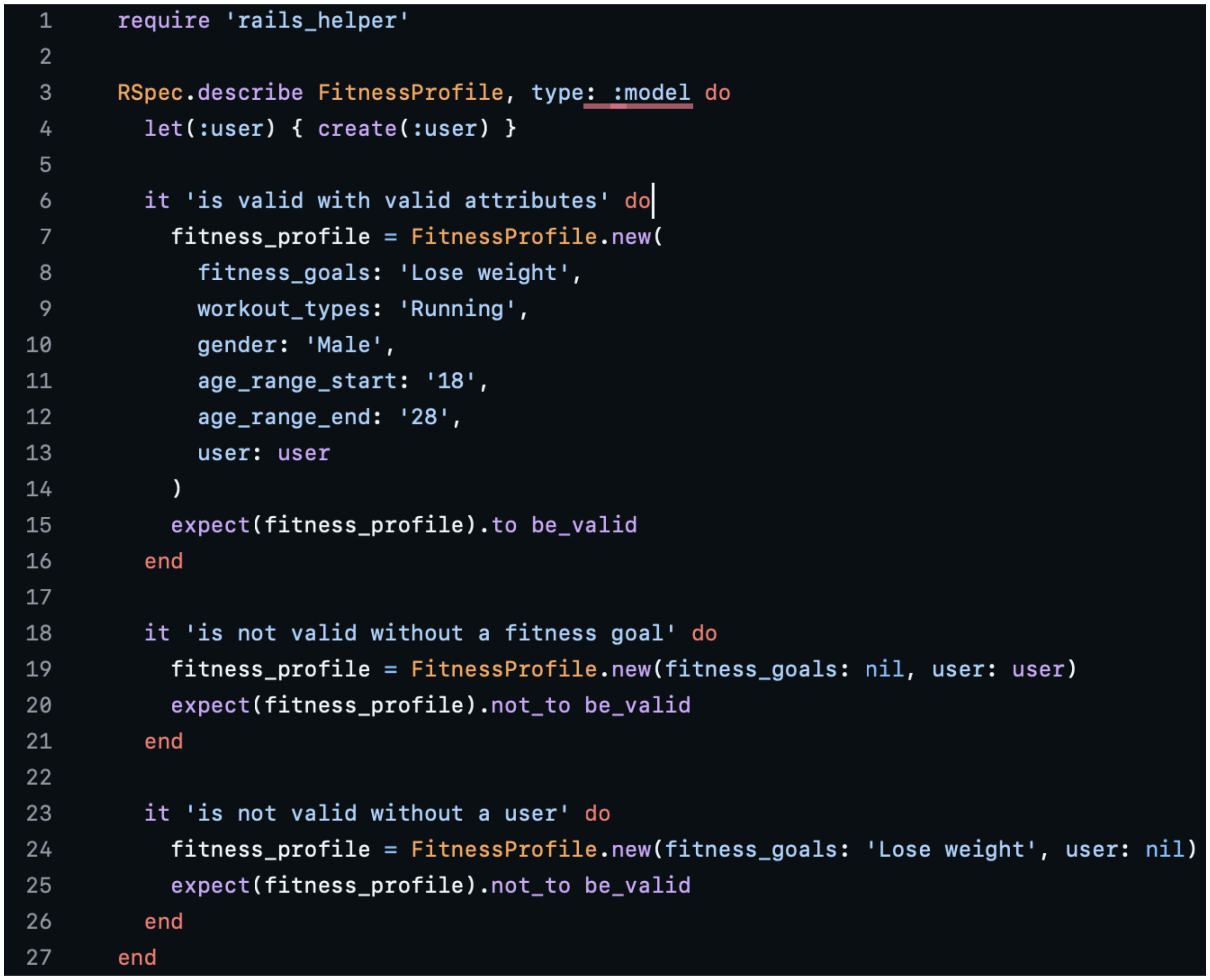
Handles attempts to upload invalid photo formats (like .txt files) or images that exceed the size limit, and verifies that appropriate error messages are displayed.

Incomplete Inputs Handling:

Tests the validation for required fields (like username), ensuring that users cannot save incomplete profiles, and shows error messages when fields are missing.

Tdd:

fitness_profile _spec



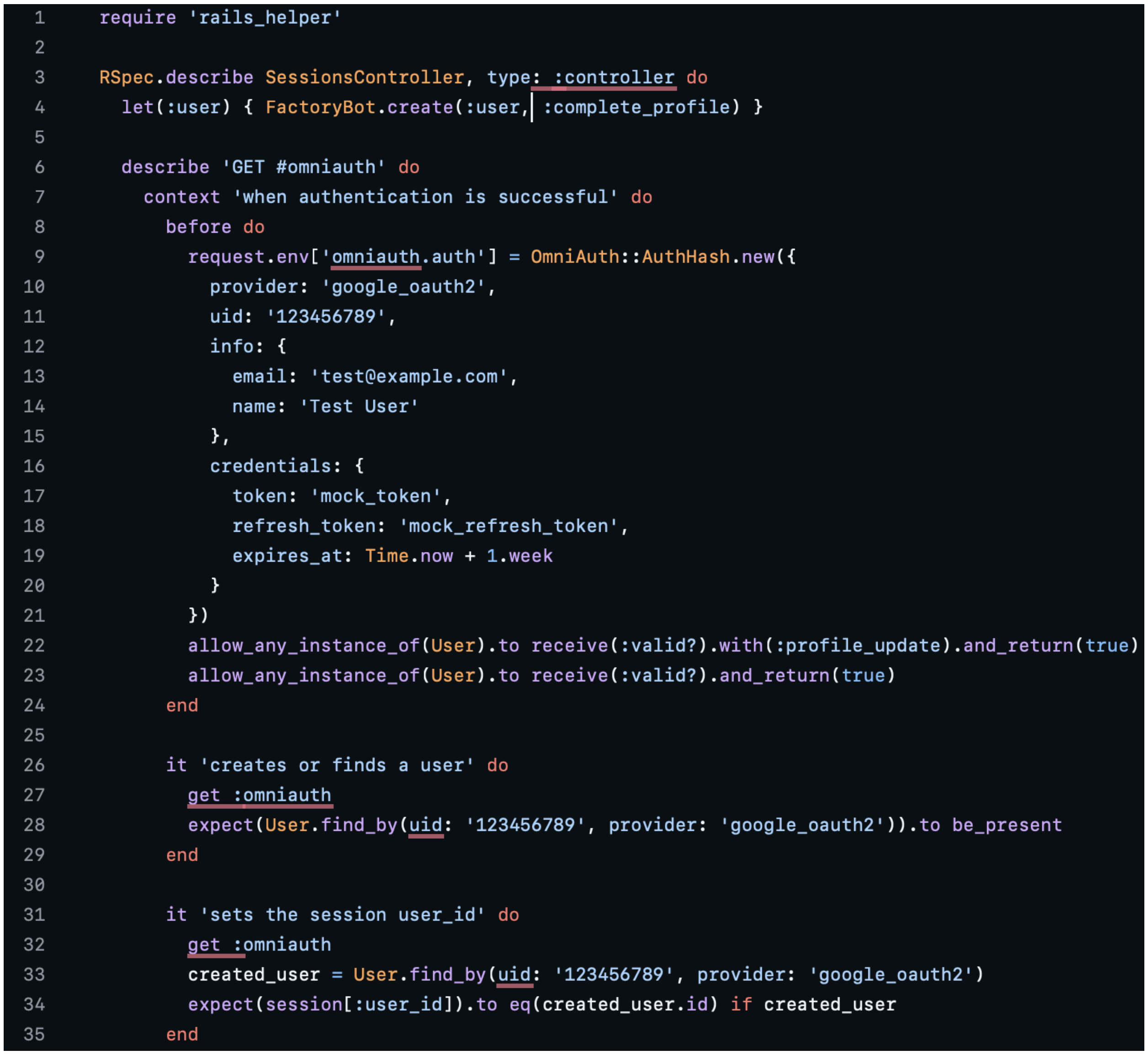
```ruby
require 'rails_helper'

RSpec.describe FitnessProfile, type: :model do
  let(:user) { create(:user) }

  it 'is valid with valid attributes' do
    fitness_profile = FitnessProfile.new(
      fitness_goals: 'Lose weight',
      workout_types: 'Running',
      gender: 'Male',
      age_range_start: '18',
      age_range_end: '28',
      user: user
    )
    expect(fitness_profile).to be_valid
  end

  it 'is not valid without a fitness goal' do
    fitness_profile = FitnessProfile.new(fitness_goals: nil, user: user)
    expect(fitness_profile).not_to be_valid
  end

  it 'is not valid without a user' do
    fitness_profile = FitnessProfile.new(fitness_goals: 'Lose weight', user: nil)
    expect(fitness_profile).not_to be_valid
  end
end
```

The purpose of these tests is to ensure that the FitnessProfile model behaves as expected when interacting with its associated attributes, such as fitness_goals, workout_types, gender, and user.

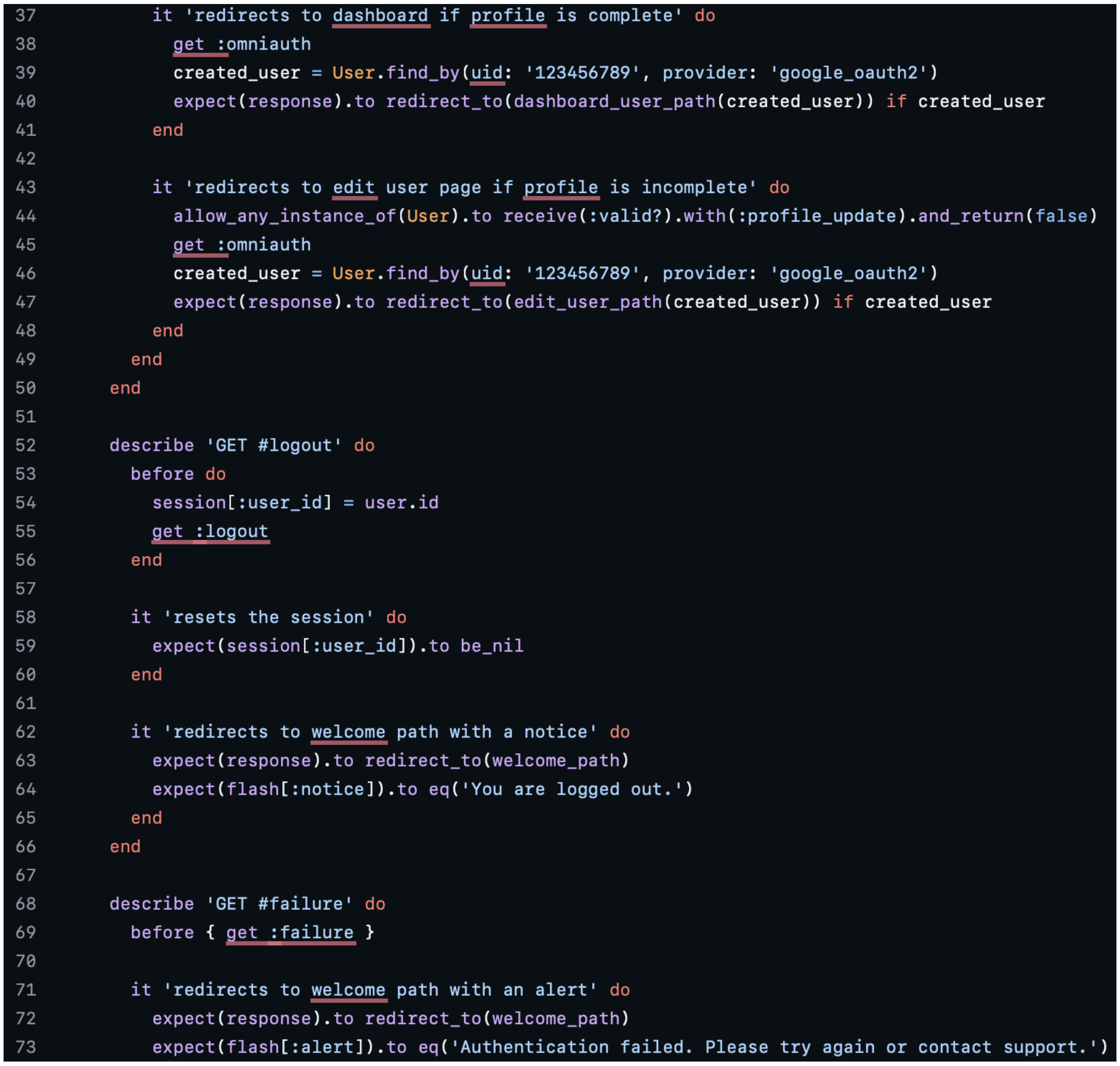
Specifically, the tests check:

1. That a FitnessProfile is valid when all necessary attributes are present and correctly assigned.
2. That the model is not valid if required attributes, like fitness_goals or user, are missing.

session_controller_spec



```ruby
require 'rails_helper'

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

  describe 'GET #omniauth' do
    context 'when authentication is successful' do
      before do
        request.env['omniauth.auth'] = OmniAuth::AuthHash.new({
          provider: 'google_oauth2',
          uid: '123456789',
          info: {
            email: 'test@example.com',
            name: 'Test User'
          },
          credentials: {
            token: 'mock_token',
            refresh_token: 'mock_refresh_token',
            expires_at: Time.now + 1.week
          }
        })
        allow_any_instance_of(User).to receive(:valid?).with(:profile_update).and_return(true)
        allow_any_instance_of(User).to receive(:valid?).and_return(true)
      end

      it 'creates or finds a user' do
        get :omniauth
        expect(User.find_by(uid: '123456789', provider: 'google_oauth2')).to be_present
      end

      it 'sets the session user_id' do
        get :omniauth
        created_user = User.find_by(uid: '123456789', provider: 'google_oauth2')
        expect(session[:user_id]).to eq(created_user.id) if created_user
      end
```



```ruby
      it 'redirects to dashboard if profile is complete' do
        get :omniauth
        created_user = User.find_by(uid: '123456789', provider: 'google_oauth2')
        expect(response).to redirect_to(dashboard_user_path(created_user)) if created_user
      end

      it 'redirects to edit user page if profile is incomplete' do
        allow_any_instance_of(User).to receive(:valid?).with(:profile_update).and_return(false)
        get :omniauth
        created_user = User.find_by(uid: '123456789', provider: 'google_oauth2')
        expect(response).to redirect_to(edit_user_path(created_user)) if created_user
      end
    end
  end

  describe 'GET #logout' do
    before do
      session[:user_id] = user.id
      get :logout
    end

    it 'resets the session' do
      expect(session[:user_id]).to be_nil
    end

    it 'redirects to welcome path with a notice' do
      expect(response).to redirect_to(welcome_path)
      expect(flash[:notice]).to eq('You are logged out.')
    end
  end

  describe 'GET #failure' do
    before { get :failure }

    it 'redirects to welcome path with an alert' do
      expect(response).to redirect_to(welcome_path)
      expect(flash[:alert]).to eq('Authentication failed. Please try again or contact support.')
```

These tests focus on validating the behavior of user authentication and session management, particularly when using OmniAuth for third-party authentication services such as Google. The tests aim to ensure that the controller handles different authentication scenarios appropriately, maintains session integrity, and redirects users to the correct paths based on their profile status. Additionally, the tests cover functionality for logging out users and handling authentication failures.

Purpose of the Tests:

1. **OmniAuth Authentication (GET #omniauth)**:

    - **Successful Authentication**: Ensures that users are correctly created or found, and their session is established. It also checks redirection logic based on whether the user has completed their profile.
    - **Incomplete Profile Handling**: Ensures users with incomplete profiles are redirected to complete their information.
2. **Logout Functionality (GET #logout)**:
    - Validates that user sessions are properly reset upon logging out and that the user is redirected with the appropriate notification.
3. **Failure Handling (GET #failure)**:
    - Confirms that when authentication fails, users are redirected to a welcome page with an alert message.