**Backend Test Cases**

**for**

**Wellness Quest**

Team: GitHub Issues

Author: Ari Le

Capstone Project Spring, 2024

The University of West Florida

March 7th, 2024

CIS 4595: Capstone Project

Dr. Bernd Owsnicki-Klewe

# **Table of Contents**

[Table of Contents 3](#_Toc164810806)

[1 User Routes 7](#_Toc164810807)

[Test case 1.1: Successful User Registration 7](#_Toc164810808)

[Test case 1.2: Unsuccessful User Registration - Invalid email address for User Registration 7](#_Toc164810809)

[Test Case 1.3: Unsuccessful User Registration - Invalid password for User Registration 8](#_Toc164810810)

[Test case 1.4: Unsuccessful User Registration - No username for User Registration 9](#_Toc164810811)

[Test case 1.5: Unsuccessful User Registration - No email address for User Registration 10](#_Toc164810812)

[Test case 1.6: Unsuccessful User Registration - No password for User Registration 10](#_Toc164810813)

[Test case 1.7: Unsuccessful User Registration - No first name for User Registration 11](#_Toc164810814)

[Test case 1.8: Unsuccessful User Registration - No last name for User Registration 12](#_Toc164810815)

[Test case 1.9: Unsuccessful User Registration - Duplicate user - Same email address 12](#_Toc164810816)

[Test case 1.10: Unsuccessful User Registration - Duplicate user - Same username 13](#_Toc164810817)

[Test case 1.11: Unsuccessful User Registration - User is logged in 14](#_Toc164810818)

[Test case 1.12: Unsuccessful User Login - Incorrect email address 15](#_Toc164810819)

[Test case 1.13: Unsuccessful User Login - Incorrect password 16](#_Toc164810820)

[Test case 1.14: Successful User Login 16](#_Toc164810821)

[Test case 1.15: Successful User Logout 17](#_Toc164810822)

[Test case 1.16: Unsuccessful User Logout 17](#_Toc164810823)

[Test case 1.17: Successfully Get User’s Information 17](#_Toc164810824)

[Test case 1.18: Unsuccessfully Get User’s Information 18](#_Toc164810825)

[Test case 1.19: Unsuccessfully delete an account 19](#_Toc164810826)

[Test case 1.20: Successfully delete an account 19](#_Toc164810827)

[2 Challenge Routes 20](#_Toc164810828)

[General Setup: Register a Test Account 20](#_Toc164810829)

[Test case 2.1: Unsuccessfully Get All Challenges – User Not Logged In 20](#_Toc164810830)

[Test case 2.2: Unsuccessfully Get All User’s Challenges – User Not Logged In 21](#_Toc164810831)

[Test case 2.3: Unsuccessfully Get A User’s Challenge Using Its Challenge ID – User Not Logged In 21](#_Toc164810832)

[Test case 2.4: Unsuccessfully Add Challenge To User’s Profile – User Not Logged In 22](#_Toc164810833)

[Test case 2.5: Unsuccessfully Get A Challenge Information Using Its Challenge ID – User Not Logged In 22](#_Toc164810834)

[Test case 2.6: Unsuccessfully Get All User’s Finished Challenges – User Not Logged In 23](#_Toc164810835)

[Test case 2.7: Unsuccessfully Delete Challenge From User’s Profile – User Not Logged In 23](#_Toc164810836)

[General Setup: Login Test Account 24](#_Toc164810837)

[Test case 2.8: Successfully Get All Challenges 24](#_Toc164810838)

[Test case 2.9: Unsuccessfully Add Challenge To User’s Profile – Missing Challenge ID 25](#_Toc164810839)

[Test case 2.10: Successfully Add Challenge To User’s Profile 25](#_Toc164810840)

[Test case 2.11: Unsuccessfully Add Challenge To User’s Profile – Adding already added challenge 26](#_Toc164810841)

[General Setup: Add 2 More Challenges To The Test Account 26](#_Toc164810842)

[Test case 2.12: Successfully Get All User’s Challenges 27](#_Toc164810843)

[Test case 2.13: Successfully Get A User’s Challenge Using Its Challenge ID 28](#_Toc164810844)

[Test case 2.14: Unsuccessfully Get A User’s Challenge – Using Invalid Challenge ID 28](#_Toc164810845)

[Test case 2.15: Successfully Get A Challenge Using Its Challenge ID 29](#_Toc164810846)

[Test case 2.16: Unsuccessfully Get A Challenge – Using Invalid Challenge ID 30](#_Toc164810847)

[Test case 2.17: Successfully Get All User’s Finished Challenges 30](#_Toc164810848)

[Test case 2.18: Successfully Updating The User’s Location While Doing The Started Challenge 31](#_Toc164810849)

[Test case 2.19: Successfully Switching Challenge Status to “Finished” When Challenge Progress Exceeds Challenge Goal 32](#_Toc164810850)

[Test case 2.20: Unsuccessfully Updating The User’s Location 33](#_Toc164810851)

[Test case 2.21: Successfully Delete Challenge from User’s Profile 34](#_Toc164810852)

[Test case 2.22: Unsuccessfully Delete Challenge from User’s Profile 35](#_Toc164810853)

# **1 User Routes**

## **Test case 1.1: Successful User Registration**

1. Objective: To verify that users can successfully register for an account.
2. Input:

* Username: arilalale
* Email: nbl8@students.uwf.edu
* Password: Password123\*\*
* First name: Ari
* Last name: Le

1. Test Steps:

* Send a POST request to the user registration endpoint with the provided input data.
* Verify that the returned status code is 200.
* Verify that the response body contains the message “User arilalale created”.

1. Expected Result: The user registered successfully.

## **Test case 1.2: Unsuccessful User Registration - Invalid email address for User Registration**

1. Objective: To verify that the application handles invalid input gracefully.
2. Input:

* Username: arilalale
* Email: fakeAddr
* Password: Password123\*\*
* First name: Ari
* Last name: Le

1. Test Steps:

* Send a POST request to the user registration endpoint with the provided input data.
* Verify that the returned status code is 400.
* Verify that the response body contains the message “Not a valid email address”.

1. Expected Result: The user registered unsuccessfully. The user receives an error message about the invalid email address.

## **Test Case 1.3: Unsuccessful User Registration - Invalid password for User Registration**

1. Objective: To verify that the application handles invalid input gracefully.
2. Input:

* Username: arilalale
* Email: fakeEmail@fake.com
* Password: password
* First name: Ari
* Last name: Le

1. Test Steps:

* Send a POST request to the user registration endpoint with the provided input data.
* Verify that the returned status code is 400.
* Verify that the response body contains the message “Password needs at 8 characters with at least 1 uppercase, 1 lowercase, 1 digit, and 1 special character”.

1. Expected Result: The user registered unsuccessfully. The user receives an error message about how a password must be.

## **Test case 1.4: Unsuccessful User Registration - No username for User Registration**

1. Objective: To verify that the application handles no input entered gracefully.
2. Input:

* Username:
* Email: fakeEmail@fake.com
* Password: Password123\*\*
* First name: Ari
* Last name: Le

1. Test Steps:

* Send a POST request to the user registration endpoint with the provided input data.
* Verify that the returned status code is 400.
* Verify that the response body contains the message “Username not provided”.

1. Expected Result: The user registered unsuccessfully. The user receives an error message about not providing a username.

## **Test case 1.5: Unsuccessful User Registration - No email address for User Registration**

1. Objective: To verify that the application handles no input entered gracefully.
2. Input:

* Username: arilalale
* Email:
* Password: Password123\*\*
* First name: Ari
* Last name: Le

1. Test Steps:

* Send a POST request to the user registration endpoint with the provided input data.
* Verify that the returned status code is 400.
* Verify that the response body contains the message “Email not provided”.

1. Expected Result: The user registered unsuccessfully. The user receives an error message about not providing an email address.

## **Test case 1.6: Unsuccessful User Registration - No password for User Registration**

1. Objective: To verify that the application handles no input entered gracefully.
2. Input:

* Username: arilalale
* Email: fakeEmail@fake.com
* Password:
* First name: Ari
* Last name: Le

1. Test Steps:

* Send a POST request to the user registration endpoint with the provided input data.
* Verify that the returned status code is 400.
* Verify that the response body contains the message “Password not provided”.

1. Expected Result: The user registered unsuccessfully. The user receives an error message about not providing a password.

## **Test case 1.7: Unsuccessful User Registration - No first name for User Registration**

1. Objective: To verify that the application handles no input entered gracefully.
2. Input:

* Username: arilalale
* Email: fakeEmail@fake.com
* Password: Password123\*\*
* First name:
* Last name: Le

1. Test Steps:

* Send a POST request to the user registration endpoint with the provided input data.
* Verify that the returned status code is 400.
* Verify that the response body contains the message “First name not provided”.

1. Expected Result: The user registered unsuccessfully. The user receives an error message about not providing a first name.

## **Test case 1.8: Unsuccessful User Registration - No last name for User Registration**

1. Objective: To verify that the application handles no input entered gracefully.
2. Input:

* Username: arilalale
* Email: fakeEmail@fake.com
* Password: Password123\*\*
* First name: Ari
* Last name:

1. Test Steps:

* Send a POST request to the user registration endpoint with the provided input data.
* Verify that the returned status code is 400.
* Verify that the response body contains the message “Last name not provided”.

1. Expected Result: The user registered unsuccessfully. The user receives an error message about not providing a last name.

## **Test case 1.9: Unsuccessful User Registration - Duplicate user - Same email address**

1. Objective: To verify that the application prevents user registration when attempting to register with a username or email address already in the system.
2. Input:

* Username: testuser
* Email: nbl8@students.uwf.edu
* Password: Password123\*\*
* First name: Ari
* Last name: Le

1. Test Steps:

* Send a POST request to the user registration endpoint with the provided input data.
* Verify that the returned status code is 409.
* Verify that the response body contains the message “User already exists”.

1. Expected Result: The user registration attempt fails due to attempting to register with a username or email address that already exists in the system. The application returns a 409 status code along with the message "User already exists".

## **Test case 1.10: Unsuccessful User Registration - Duplicate user - Same username**

1. Objective: To verify that the application prevents user registration when attempting to register with a username or email address already in the system.
2. Input:

* Username: arilalale
* Email: fakeEmail@fake.com
* Password: Password123\*\*
* First name: Ari
* Last name: Le

1. Test Steps:

* Send a POST request to the user registration endpoint with the provided input data.
* Verify that the returned status code is 409.
* Verify that the response body contains the message “User already exists”.

1. Expected Result: The user registration attempt fails due to attempting to register with a username or email address that already exists in the system. The application returns a 409 status code along with the message "User already exists".

## **Test case 1.11: Unsuccessful User Registration - User is logged in**

1. Objective: To verify that the application prevents user registration when the same user is already logged in.
2. Setup Steps:

* Send a POST request to the user login endpoint with the below input data to successfully log in a user.
* Input data:
  + Email: nbl8@students.uwf.edu
  + Password: Password123\*\*

1. Test Steps:

* Send a POST request to the user registration endpoint with the below input data after the setup step.
* Input data:
  + Username: arilalale
  + Email: nbl8@students.uwf.edu
  + Password: Password123\*\*
  + First name: Ari
  + Last name: Le
* Verify that the returned status code is 406.
* Verify that the response body contains the message “User is already logged in”.

1. Cleanup Steps:

* Send a POST request to the user logout endpoint to log out the user.

1. Expected Result: The user registration attempt fails due to a user being already logged in. The application returns a 406 status code along with the message "User is already logged in".

## **Test case 1.12: Unsuccessful User Login - Incorrect email address**

1. Objective: To verify that the application handles unsuccessful user login attempts when an incorrect email address is provided.
2. Input:

* Email: fakeEmail@fake.com
* Password: Password123\*\*

1. Test steps:

* Send a POST request to the user login endpoint with the provided input data.
* Verify that the returned status code is 404.
* Verify that the response body contains the message “User not found”.

1. Expected Result: The user login attempt fails due to an incorrect email address, and the application returns a 404 status code along with the message "User not found".

## **Test case 1.13: Unsuccessful User Login - Incorrect password**

1. Objective: To verify that the application handles unsuccessful user login attempts when an incorrect password is provided.
2. Input:

* Email: nbl8@students.uwf.edu
* Password: Pass123456\*\*

1. Test steps:

* Send a POST request to the user login endpoint with the provided input data.
* Verify that the returned status code is 401.
* Verify that the response body contains the message “Incorrect password”.

1. Expected Result: The user login attempt fails due to an incorrect password. The application returns a 401 status code (Unauthorized) along with the message "Incorrect password".

## **Test case 1.14: Successful User Login**

1. Objective: To verify that users can successfully log in to their accounts using valid credentials.
2. Input:

* Email: nbl8@students.uwf.edu
* Password: Password123\*\*

1. Test steps:

* Send a POST request to the user login endpoint with the provided input data.
* Verify that the returned status code is 200.
* Verify that the response body contains the message “User logged in”.

1. Expected Result: The user successfully logs in to their account using valid credentials. The application returns a 200 status code (OK) along with the message "User logged in".

## **Test case 1.15: Successful User Logout**

1. Objective: To verify that users can successfully log out from their accounts.
2. Precondition: User is logged in.
3. Test steps:

* Send a POST request to the user logout endpoint.
* Verify that the returned status code is 200.
* Verify that the response body contains the message “User logged out”.

1. Expected Result: The user successfully logs out from their account. The application returns a 200 status code (OK) along with the message "User logged out".

## **Test case 1.16: Unsuccessful User Logout**

1. Objective: To verify that users cannot log out when not logged in.
2. Test steps:

* Send a POST request to the user logout endpoint.
* Verify that the returned status code is 200.
* Verify that the response body contains the message “User not logged in, cannot logout”.

1. Expected Result: The user cannot log out when not logged in. The application returns a 200 status code (OK) along with the message "User not logged in, cannot logout".

## **Test case 1.17: Successfully Get User’s Information**

1. Objective: To verify that users can retrieve their information after logging in.
2. Setup:

* Send a POST request to the user login endpoint with the below input data.
* Input data:
  + Email: nbl8@students.uwf.edu
  + Password: Password123\*\*

1. Test steps:

* Send a GET request to the user information endpoint.
* Verify that the returned status code is 200.
* Verify that the response body contains the user's information, including username, email, first name, last name, level, XP, XP to reach the next level and login status.

1. Expected Result: The user successfully retrieves their information after logging in. The application returns a 200 status code (OK) along with the user's information, confirming that the operation was successful.

## **Test case 1.18: Unsuccessfully Get User’s Information**

1. Objective: To verify that users cannot retrieve their information without logging in.
2. Setup:

* Send a POST request to the user logout endpoint.

1. Test Steps:

* Send a GET request to the user information endpoint.
* Verify that the returned status code is 200.
* Verify that the response body contains the message "logged\_in: false".

1. Expected Result: The user cannot retrieve their information without logging in. The application returns a 200 status code (OK) along with the message "logged\_in: false", indicating that the operation was unsuccessful.

## **Test case 1.19: Unsuccessfully delete an account**

1. Objective: To verify that users cannot delete their accounts without logging in.
2. Test Steps:

* Send a DELETE request to the delete account endpoint.
* Verify that the response status code is 401 (Unauthorized).

1. Expected Result: The user cannot delete their account without logging in. The application returns a 401 status code (Unauthorized) indicating that the operation was unsuccessful.

## **Test case 1.20: Successfully delete an account**

1. Objective: To verify that users can successfully delete their accounts.
2. Setup:

* Send a POST request to the user login endpoint with the below input data.
* Input data:
  + Email: nbl8@students.uwf.edu
  + Password: Password123\*\*

1. Test Steps:

* Send a DELETE request to the delete account endpoint.
* Verify that the response status code is 204.

1. Expected Result: The user successfully deletes their account. The application returns a 204 status code (No Content), indicating that the operation was successful. Additionally, the user account should no longer exist in the database.

# **2 Challenge Routes**

## **General Setup: Register a Test Account**

1. Objective: To register a test account for use in subsequent test cases.
2. Input:

* Username: arilalale
* Email: nbl8@students.uwf.edu
* Password: Password123\*\*
* First name: Ari
* Last name: Le

1. Setup Steps:

* Send a POST request to the user registration endpoint with the provided input data.
* Verify that the response status code is 200.

1. Expected Result: The test account is successfully registered, and the application returns a 200 status code.

## **Test case 2.1: Unsuccessfully Get All Challenges – User Not Logged In**

1. Objective: To verify that a user cannot retrieve all challenges when not logged in.
2. Test Steps:

* Send a GET request to the endpoint to fetch all challenges.
* Verify that the returned status code is 403.
* Verify that the response body contains an error message “User is not logged in”.

1. Expected Result: The challenges are not retrieved, and the application returns a 403 status code along with the error message “User is not logged in”.

## **Test case 2.2: Unsuccessfully Get All User’s Challenges – User Not Logged In**

1. Objective: To verify that a user cannot retrieve their challenges when not logged in.
2. Test Steps:

* Send a GET request to the endpoint to fetch the user’s challenges.
* Verify that the response status code is 403.
* Verify that the response body contains an error message “User is not logged in”.

1. Expected Result: The user’s challenges are not retrieved, and the application returns a 403 status code along with the error message “User is not logged in”.

## **Test case 2.3: Unsuccessfully Get A User’s Challenge Using Its Challenge ID – User Not Logged In**

1. Objective: To verify that a user cannot retrieve a specifiv challenge without logging in.
2. Input:

* Challenge\_id: 1

1. Test Steps:

* Send a GET request to the endpoint with the specific challenge ID to fetch the user’s challenge.
* Verify that the response status code is 403.
* Verify that the response body contains an error message “User is not logged in”.

1. Expected Result: The user’s challenge is not retrieved, and the application returns a 403 status code along with the error message “User is not logged in”.

## **Test case 2.4: Unsuccessfully Add Challenge To User’s Profile – User Not Logged In**

1. Objective: To verify that a challenge cannot be added to a user’s profile without logging in.
2. Input:

* Challenge\_id: 1

1. Test Steps:

* Send a POST request to add a challenge to the user’s profile without logging in.
* Verify that the response status code is 403.
* Verify that the response body contains an error message “User is not logged in”.

1. Expected Result: The challenge is not added to the user’s profile, and the application returns a 403 status code along with the error message “User is not logged in”.

## **Test case 2.5: Unsuccessfully Get A Challenge Information Using Its Challenge ID – User Not Logged In**

1. Objective: To verify that a challenge’s information cannot be retrieved without logging in.
2. Input:

* Challenge\_id: 1

1. Test Steps:

* Send a GET request to the endpoint to retrieve the challenge’s information without logging in.
* Verify that the response status code is 403.
* Verify that the response body contains an error message “User is not logged in”.

1. Expected Result: The challenge’s information is not retrieved, and the application returns a 403 status code along with the error message “User is not logged in”.

## **Test case 2.6: Unsuccessfully Get All User’s Finished Challenges – User Not Logged In**

1. Objective: To verify that a user’s finished challenges cannot be retrieved when the user is not logged in.
2. Test Steps:

* Send a GET request to the endpoint to fetch all user’s finished challenges.
* Verify that the response status code is 403.
* Verify that the response body contains the message “User is not logged in”.

1. Expected Result: The user’s finished challenges are not retrieved, and the application returns a 403 status code along with the message “User is not logged in”.

## **Test case 2.7: Unsuccessfully Delete Challenge From User’s Profile – User Not Logged In**

1. Objective: To verify that a challenge cannot be deleted from a user’s profile without logging in.
2. Input:

* Challenge\_id: 1

1. Test Steps:

* Send a DELETE request to the endpoint to delete the challenge from the user’s profile without logging in.
* Verify that the response status code is 403.
* Verify that the response body contains an error message “User is not logged in”.

1. Expected Result: The challenge is not deleted from the user’s profile, and the application returns a 403 status code along with the error message “User is not logged in”.

## **General Setup: Login Test Account**

1. Objective: To log in using the test account created for testing purposes.
2. Input:

* Email: nbl8@students.uwf.edu
* Password: Password123\*\*

1. Setup Steps:

* Send a POST request to the login endpoint with the test account credentials.
* Verify that the response status code is 200.

1. Expected Result: The test account is successfully logged in, and the application returns a 200 status code.

## **Test case 2.8: Successfully Get All Challenges**

1. Objective: To verify that all challenges can be retrieved successfully.
2. Test Steps:

* Send a GET request to the endpoint to fetch all challenges.
* Verify that the response status code is 200.
* Verify that the response body contains the list of all challenges and their information including challenge ID, challenge name, challenge description, challenge type, challenge goal, and challenge XP.

1. Expected Result: All challenges are retrieved successfully, and the application returns a 200 status code along with the list of challenges. Each challenge in the list includes challenge ID, challenge name, challenge description, challenge type, challenge goal, and challenge XP.

## **Test case 2.9: Unsuccessfully Add Challenge To User’s Profile – Missing Challenge ID**

1. Objective: To verify that a challenge cannot be added to a user’s profile without the challenge ID input.
2. Test Steps:

* Send a POST request to add a challenge to the user’s profile wihout providing the challenge ID.
* Verify that the response status code is 400.
* Verify that the response body contains an error message “Missing challenge\_id”.

1. Expected Result: The challenge is not added to the user’s profile, and the application returns a 400 status code due to the missing challenge ID input. The returned error message contains “Missing challenge\_id”.

## **Test case 2.10: Successfully Add Challenge To User’s Profile**

1. Objective: To verify that a challenge can be added successfully to a user’s profile.
2. Input:

* Challenge\_id: 1

1. Test Steps:

* Send a POST request to add a challenge to the user’s profile.
* Verify that the response status code is 200.
* Verify that the response body contains the added challenge’s information including the challenge status, the challenge progress, the user ID, and the challenge ID.

1. Expected Result: The challenge is added successfully to the user’s profile. The application returns a 200 status code along with the challenge’s information (challenge status, challenge progress, user ID, and challenge ID).

## **Test case 2.11: Unsuccessfully Add Challenge To User’s Profile – Adding already added challenge**

1. Objective: To verify that a challenge cannot be added to a user’s profile if it is already added.
2. Setup: This test case must be executed after executing test case 2.9.
3. Input:

* Challenge\_id: 1

1. Test Steps:

* Send a POST request to add a challenge to the user’s profile.
* Verify that the response status code is 400.
* Verify that the response body contains an error message “Challenge already added”.

1. Expected Result: The challenge is not added to the user’s profile, and the application returns a 400 status code with an error message “Challenge already added”.

## **General Setup: Add 2 More Challenges To The Test Account**

1. Objective: To prepare the test environment by adding two more challenges to the test account.
2. Setup Steps:

* Send a POST request to add the challenge with challenge\_id is 3 to the test account.
* Verify that the response status code is 200.
* Send another POST request to add the challenge with challenge\_id is 4 to the test account.
* Verify that the response status code is 200.

1. Expected Result: Two additional challenges are successfully added to the test account.

## **Test case 2.12: Successfully Get All User’s Challenges**

1. Objective: To verify that all challenges associated with the logged-in user can be retrieved successfully.
2. Test Steps:

* Send a GET request to the endpoint to fetch all user’s challenges.
* Verify that the response status code is 200.
* Verify that the response body contains the list of challenges associated with the logged-in user. Each challenge contains the challenge ID, the challenge name, the challenge description, the challenge type, the challenge goal, the challenge XP, the challenge status, the challenge progress.

1. Expected Result: All challenges associated with the logged-in user are retrieved successfully, and the application returns a 200 status code along with the list of challenges. Each challenge in the list contains the challenge ID, the challenge name, the challenge description, the challenge type, the challenge goal, the challenge XP, the challenge status, the challenge progress.

## **Test case 2.13: Successfully Get A User’s Challenge Using Its Challenge ID**

1. Objective: To verify that a user’s challenge can be retrieved successfully using a valid challenge ID.
2. Input:

* Challenge\_id: 3

1. Test Steps:

* Send a GET request to the endpoint to fetch the user’s challenge using the provided challenge ID.
* Verify that the response status code is 200.
* Verify that the response body contains the challenge information corresponding to the provided challenge ID. The challenge information must include the challenge ID, the challenge name, the challenge description, the challenge type, the challenge goal, the challenge XP, the challenge status, and the challenge progress.

1. Expected Result: The user’s challenge is retrieved successfully using the valid challenge ID, and the application returns a 200 status code along with the challenge information. The challenge information includes include the challenge ID, the challenge name, the challenge description, the challenge type, the challenge goal, the challenge XP, the challenge status, and the challenge progress.

## **Test case 2.14: Unsuccessfully Get A User’s Challenge – Using Invalid Challenge ID**

1. Objective: To verify that a user’s challenge cannot be retrieved using an invalid challenge ID.
2. Input:

* Challenge\_id: 2

1. Test Steps:

* Send a GET request to the endpoint to fetch the user’s challenge using the provided challenge ID.
* Verify that the response status code is 404.
* Verify that the response body contains the message “Not Found”.

1. Expected Result: The user’s challenge is not retrieved due to the use of an invalid challenge ID, and the application returns a 404 status code with an error message indicating that the challenge was not found.

## **Test case 2.15: Successfully Get A Challenge Using Its Challenge ID**

1. Objective: To verify that a challenge can be successfully retrieved using its challenge ID.
2. Input:

* Challenge\_id: 2

1. Test Steps:

* Send a GET request to the endpoint to fetch the challenge by its challenge ID.
* Verify that the response status code is 200.
* Verify that the response body contains the details of the retrieved challenge. The details must include the challenge ID, the challenge name, the challenge description, the challenge type, the challenge goal, the challenge XP.

1. Expected Result: The challenge is retrieved successfully using its challenge ID, and the application returns a 200 status code along with the details of the challenge. The details of the challenge includes the challenge ID, the challenge name, the challenge description, the challenge type, the challenge goal, the challenge XP.

## **Test case 2.16: Unsuccessfully Get A Challenge – Using Invalid Challenge ID**

1. Objective: To verify that a challenge cannot be retrieved using an invalid challenge ID.
2. Input:

* Challenge\_id: 35

1. Test Steps:

* Send a GET request to the endpoint to fetch the challenge using the provided challenge ID.
* Verify that the response status code is 404.
* Verify that the response body contains the message “Not Found”.

1. Expected Result: The challenge is not retrieved due to the use of an invalid challenge ID, and the application returns a 404 status code with an error message indicating that the challenge was not found.

## **Test case 2.17: Successfully Get All User’s Finished Challenges**

1. Objective: To verify that user’s finished challenges can be retrieved successfully.
2. Setup: Manually set a challenge 1’s status to ‘finished’.
3. Test Steps:

* Send a GET request to the endpoint to fetch all user’s finished challenges.
* Verify that the response status code is 200.
* Verify that the response body contains the list of all finished challenges associated with the logged-in user. Each finished challenge contains challenge ID, challenge name, challenge description, challenge type, challenge goal, challenge XP, challenge status, and challenge progress.

1. Expected Result: The user’s finished challenges are retrieved successfully, and the application returns a 200 status code along with the list of finished challenges associated with the logged-in user. Each finished challenge contains challenge ID, challenge name, challenge description, challenge type, challenge goal, challenge XP, challenge status, and challenge progress.

## **Test case 2.18: Successfully Updating The User’s Location While Doing The Started Challenge**

1. Objective: To verify that locations as well as the challenge progress can be successfully updated using the GPS route.
2. Input:

{

“locations”: [

{

“challenge\_id”: 3,

“location\_object”: {

“coords”: {

“latitude”: 30.453967,

“longitude”: -86.652502

}

}

},

{

“challenge\_id”: 3,

“location\_object”: {

“coords”: {

“latitude”: 30.454095,

“longitude”: -86.652369

}

}

}

]

}

1. Test Steps:

* Send a POST request to the GPS route endpoint with the provided input data.
* Verify that the response status code is 200.
* Verify that the response body contains the message “OK” indicating the locations as well as the challenge progress are updated.
* Verify that the challenge’s status is set to “started”.

1. Expected Result: The locations as well as the challenge progress are updated successfully, and the application returns a 200 status code along with the message “OK”. Additionally, the challenge’s status is “started”.
2. Fail Test Report:
   1. Description of the Issue: The challenge’s status did not switch to “started” as expected after updating the user’s location.
   2. Observations: The response status code was 200, indicating a successful update of the user’s location. However, the challenge status remained unchanged.
   3. Expected Result vs. Actual Result: While the location update was successful, the challenge status did not reflect as “started” as expected.

## **Test case 2.19: Successfully Switching Challenge Status to “Finished” When Challenge Progress Exceeds Challenge Goal**

1. Objective: To verify that the challenge status switches to “finished” when the challenge progress exceeds the challenge goal.
2. Input:

{

“locations”: [

{

“challenge\_id”: 4,

“location\_object”: {

“coords”: {

“latitude”: 30.453967,

“longitude”: -86.652502

}

}

},

{

“challenge\_id”: 4,

“location\_object”: {

“coords”: {

“latitude”: 30.456167,

“longitude”: -86.650958

}

}

}

]

}

1. Test Steps:

* Send a POST request to the GPS route endpoint with the provided input data.
* Verify that the response status code is 200.
* Verify that the response body contains the message “OK” indicating the locations as well as the challenge progress are updated.
* Verify that the challenge’s progress is higher than or equal to the challenge’s goal.
* Verify that the challenge’s status is switched to “finished”.

1. Expected Result: The challenge’s status switches to “finished”.
2. Fail Test Report:
   1. Description of the Issue: The challenge’s status remains at “not started” even though the challenge progress exceeds the challenge goal.
   2. Observations: The challenge progress was successfully updated, surpassing the challenge goal. However, the challenge status did not transition to “finished” as expected.
   3. Expected Result vs. Actual Result: Despite the challenge progress exceeding the challenge goal, the challenge status remained unchanged at “not started”.

## **Test case 2.20: Unsuccessfully Updating The User’s Location**

1. Objective: To verify that locations as well as the challenge progress cannot be updated when the chosen challenge is not started or unavailable.
2. Precondition:

* The chosen challenge’s status must be either not started or unavailable (not added to the logged-in user’s profile).

1. Input:

{

“locations”: [

{

“challenge\_id”: 2,

“location\_object”: {

“coords”: {

“latitude”: 30.454095,

“longitude”: -86.652369

}

}

},

{

“challenge\_id”: 2,

“location\_object”: {

“coords”: {

“latitude”: 30.454136,

“longitude”: -86.652339

}

}

}

]

}

1. Test Steps:

* Send a POST request to the GPS route endpoint with the provided input data.
* Verify that the response status code is 400.
* Verify that the response body contains the message “Challenge not started or is unavailable”.

1. Expected Result: The locations as well as the challenge progress are not updated, and the application returns a 400 status code with the message indicating that the challenge is not started or is unavailable.

## **Test case 2.21: Successfully Delete Challenge from User’s Profile**

1. Objective: To verify that a challenge can be successfully deleted from a user’s profile.
2. Input:

* Challenge\_id: 4

1. Test Steps:

* Send a DELETE request to the endpoint to delete the challenge from the user’s profile.
* Verify that the response status code is 204.
* Verify that the challenge is no longer associated with the user’s profile.

1. Expected Result: The challenge is successfully deleted from the user’s profile, and the application returns a 204 status code indicating successful deletion.

## **Test case 2.22: Unsuccessfully Delete Challenge from User’s Profile**

1. Objective: To verify that a challenge cannot be deleted
2. Test Steps:

* Send a DELETE request to the endpoint without a challenge ID.
* Verify that the response status code is 500.
* Verify that the response body contains the error message “Unable to delete challenge”.

1. Expected Result: No challenge is deleted from the user’s profile. The application returns a 500 status code and an error message “Unable to delete challenge”.