Test Plan

The test plan initially includes unit tests for respective features in our Django application. Next, an overview of the functionality testing is mentioned which includes external user validation.

Test Plan for User Authentication

1. Introduction

This test plan outlines the testing strategy for the user authentication system of the EcoGo web application. The goal is to verify that users can successfully log in, log out, and that authentication-based access control functions correctly.

2. Scope

In-Scope Features:

- User Login with Valid Credentials
- User Login with Invalid Credentials
- User Logout
- Homepage Unauthorised Access Control

Out-of-Scope Features:

- Password reset and recovery mechanisms.
- Multi-factor authentication (MFA).

3. Test Objectives

- Verify that a registered user can log in successfully.
- Ensure that login fails when incorrect credentials are used.
- Confirm that an authenticated user can log out properly.
- Ensure that unauthenticated users are restricted from accessing the homepage.

4. Test Approach

The authentication system will be tested using automated unit tests with Django's built-in TestCase framework.

- Testing Methodology: Automated backend testing using Django's Test Client.
- Test Execution: The test cases will be run using Django's unittest framework.

5. Test Environment

Framework: Django

Testing Tool: Django TestCase

Database: Django default SQLite (test database)

Test Data: A predefined test user created in setUp().

6. Test Cases

Test Case 1: User Login with Valid Credentials

Objective: Ensure that a registered user can log in successfully.

Test Steps:

1. Send a POST request to self.login_url with valid credentials (testuser, testpass123).
2. Verify that the response redirects to the homepage (self.homepage_url).
3. Confirm that the session key is set, indicating successful authentication.

Expected Outcome:

- The response should redirect to the homepage after login.
- The session key should exist, confirming the user is authenticated.

Test Case 2: User Login with Invalid Credentials

Objective: Ensure that login fails when incorrect credentials are used.

Test Steps:

1. Send a POST request to self.login_url with an invalid password.
2. Verify that the response re-renders the login page (user/login.html) with errors.
3. Check that the response status code is 200, indicating login failure.
4. Confirm that the user is not authenticated (session key should not exist).

Expected Outcome:

- The login page is re-rendered with an error message.
- The response returns a 200 status code.
- The session key should not exist, confirming authentication failure.

Test Case 3: User Logout

Objective: Verify that an authenticated user can log out successfully.

Test Steps:

1. Log in the test user using a POST request with valid credentials.
2. Verify that the session contains the authenticated user.
3. Send a GET request to self.logout_url to log out.
4. Verify that the response redirects to the landing page (self.landing_url).
5. Confirm that the session is cleared, ensuring the user is logged out.

Expected Outcome:

- The user is logged out successfully.
- The response redirects to the landing page.
- The session key should be removed, confirming that the user is no longer authenticated.

Test Case 4: Homepage Access Requires Authentication (Login)

Objective: Verify that an unauthenticated user trying to access the homepage is redirected to the login page.

Test Steps:

1. Send a GET request to self.homepage_url without logging in.
2. Verify that the response returns a 302 status code (redirect).
3. Ensure that the redirect URL points to the login page (self.login_url).

Expected Outcome:

- The response should return status code 302 (indicating a redirect).
- The user is redirected to the login page, confirming that authentication is required.

7. Test Execution

- The tests will be executed using Django's TestCase framework.
- Results will be recorded, indicating pass/fail status.
- Any failed tests will be analyzed for errors, and necessary fixes will be implemented.

CardsTest

Test Case: test_create_card

- Ensures that cards can be created with and without an image.
    - Create a card named “uno” without an image.
    - Create a card named “Grunty” with an image “Test.jpg”.
    - Attempt to create a card named “dos” with a non-existing image “Missing”.

- Expected Results:
    - The card “uno” should be created with a default image path.
    - The card “Grunty” should be created with the specified image path
    - A *FileNotFoundError* should be raised for the card “dos”

Test Case: test_change_image

- Ensure that the card’s image can be changed to an existing file.

- Expected results:
    - The image of “card0” should be updated to the new image path.
    - A *FileNotFoundError* should be raised for a non-existent file.

User Inventory Tests

Setup:
Before each test is run, the following steps are followed:
- A user is created in the User Model
- A card with the name coal-imp is created with a value of 10 and saved

Test Cases

test_user_inventory_index_redirect

- **Objective**:
    - Tests that the "user/inventory" endpoint correctly redirects to the login page when a non-authenticated user attempts to access.

- **Expected Output:**
    - The server responds with 302 Redirect Request code.

test_user_inventory_sell_invalid

- **Objective**:

    - Tests that the /user/inventory endpoint correctly handles invalid input when attempting to sell a card.
    - It covers three scenarios: a malformed request, a generally non-existent card, and selling a card the user doesn't own.

- **Expected Output:**
    - The server responds with 400 Bad Request codes for each case.
    - User inventory and point balance remain unchanged after invalid requests.

test_user_inventory_sell_valid

- **Objectives**:
    - Verifies that a valid sell request successfully removes one quantity of the specified card from the user's inventory.
    - Confirms that the updated template is rendered and the user receives points.

- **Expected Output:**
    - The response status code is 200 OK.
    - The card quantity is reduced by only one unit.
    - An updated inventory view is displayed.

Cards Shop Tests

Setup:
Before each test is run, the following steps are followed:
- A user is created in the User Model
- A pack with a cost of 20 is created and saved

Test Cases

test_shop_buy_item_invalid

- **Objectives**:
    - Tests that an invalid input into 'user/shop' will result in the proper error code being sent.
    - These should include: An invalid request structure, When a pack does not exist, When a user does not have enough points.

- **Expected Output:**
    - The response status code is 400 Bad Request in each case above.

- The user’s points balance is unchanged if a bad request occurs.

test_shop_buy_item_valid

- **Objectives**:
  - Tests that if the input into 'user/shop/' is valid, that cards will be added and the template will be sent.
  - A user is created and given enough points to buy a card (5 cards will be bought)
  - Ensures cards are awarded and then points are deducted accordingly.

- **Expected Output:**
  - The user is given 5 cards in their inventory, including the duplicates.
  - The user’s points balance decreases by the amount the card costs multiplied by the number if cards that are bought.

Leaderboard Tests

Setup:
Before each test is run, the following steps are followed:
- All the users are removed from the User model
- All the use data is removed from the UserData model

Test Cases

test_leaderboard_only_5_users

- **Objectives**:
  - Creating 5 users in the User model
  - Ensuring that exactly 5 users are shown, even if 10 is the maximum number of users shown.

- **Expected Output:**
  - A 200 response code indicating there are no errors
  - The users are included in the context of the front-end template
  - The length of the context is 5, instead of 10 (the default maximum)

test_leaderboard_returns_top_10_users

- **Objectives**:
    - Creating 20 users with different levels
    - Ensuring that only 10 users are shown (the maximum) in the correct order of decreasing levels

- **Expected Output:**
    - A 200 response code indicating there are no errors
    - The length of the context is 20, instead of 20 (the total number of users)
    - The order of the users is correct, in decreasing levels

Card Trading

Setup:
Before each test is run, the following steps are followed:

- Two users are created inside of the trading room

Test Cases

test_join_room

- **Objectives**:
    - A TradingRoom instance is created with room_owner as the owner.
    - Verifies that the join_room method in TradingRoom correctly registers a new member and then triggers appropriate responses for both the room owner and the joining user.

- **Expected Output:**
    - First call to response_func is made for the room owner, confirming correct user and data.
    - Second call is made for the joining user (room_member), also with correct user and data.
    - A third call (if it happens) will cause the test to fail.
    - room_member is set correctly in the TradingRoom instance and then response_func is stored properly in the instance for future use.

test_N_to_D_transition_process

- **Objectives**:

- Tests that the transition from the Neutral (N) state to the Decision (D) state works as expected within a TradingRoom.
- Verifies that both the room owner and member receive the correct "D" state flag during the transition.

- **Expected Output:**
  - The mock response is called 4 times:
    - The first two calls happen during join_room (setup phase).
    - The third and fourth calls send "state_flag": "D" to the room owner and room member respectively.
  - The state transition behaves correctly and only sends the appropriate data to each participant.

test_member_disconnect

- **Objectives**:
  - Tests that the TradingRoom class properly handles a room member disconnecting.
  - Ensures that internal state is reset and the room owner is notified appropriately.
  - A TradingRoom is created with a room owner, then a member joins the room, then the member disconnects using the disconnect() function.

- **Expected Output:**
  - The response function is called 3 times:
    - The first two calls happen during join_room (setup phase).
    - The third and fourth calls send "state_flag": "D" to the room owner and room member respectively.
  - Room state is correctly reset:
    - room_member is None
    - Response_func is None
    - Trade_hash is None
    - End_room remains False
    - State is set to “W” which means waiting state

To Run The Tests:

Before running the tests, ensure you take a look at the README file.

To run the tests, use the following command:

To conduct the user, card shop, and leaderboard tests:

```
cd src
python manage.py test apps/user
```

To conduct the cards tests:

```
cd src
python manage.py test apps/cards
```

The same goes with all of the other apps, simply switch out the app you want to test.

Test Results

All tests currently work as intended above:

python manage.py test apps/user

Found 28 test(s).
Creating test database for alias 'default'...
Gamemaster group created!
view_user permission does not exist!
System check identified no issues (0 silenced).
.....

Ran 28 tests in 10.211s

OK

Functionality Testing

Functional testing was carried out in two phases. In the first phase, the development team manually validated each feature as a whole, ensuring that all components worked together as expected according to the functional requirements. This internal review helped catch early issues and confirm that individual features were implemented correctly from end to end.

In the second phase, the system was provided to external users who tested it in real-world scenarios. These users interacted with the application based on typical use cases, allowing us to validate the system's functionality, usability, and overall behavior from an end-user perspective. Their feedback was valuable in identifying edge cases and ensuring the system met the intended requirements in practical usage.

External User Feedback:

An overview of the user feedback included the following:

- A great use of the register and login pages, where users highlighted how they liked the register form validation which indicates which field did not allow them to register (in the case of an invalid field).

- The landing page also received positive results, where users mentioned that the project's scope was clear, and had an appealing layout.

- The users successfully scanned QR codes which in turn gave an individual card to their inventories. The card-selling feature within the inventory also worked as intended.

- Once we artificially gave users enough points to buy packs, the pack shop was used correctly and no issues were identified. The packs were bought correctly and 5 random cards were added to the user's inventory.

- Once users had enough cards, they tested the trading feature which received positive comments on the design of a room owner and member. However, they indicated that the member's side should indicate that the owner is making selections instead of no information being displayed previous to the trade proposal.