Testing Report

1. Accounts Test Cases and Results

Authenticated User Access Tests

Test Case 1: Authenticated User Redirect

- Description: Verifies that an authenticated user attempting to access the home page is redirected correctly.
- Expected Outcome: HTTP 302 (Redirect) to /home
- Actual Outcome: Passed

Test Case 2: Unauthenticated User Redirect

- Description: Ensures that an unauthenticated user accessing the home page is redirected to the signup page.
- Expected Outcome: HTTP 302 (Redirect) to /accounts/register/
- Actual Outcome: Passed

API Authentication Tests

Test Case 3: User Registration

- Description: Checks if a new user can be registered successfully via API.
- Expected Outcome: HTTP 201 (Created), User appears in the database.
- Actual Outcome: Passed

Test Case 4: User Gets One Pack on Registration

- Description: Ensures that a newly registered user receives one pack automatically.
- Expected Outcome: pack_count = 1 for the newly created user.
- Actual Outcome: Passed

Test Case 5: User Login

- Description: Verifies that a registered user can log in and receive an authentication token.
- Expected Outcome: HTTP 200 (OK), response contains a valid token.
- Actual Outcome: Passed

Test Case 6: Fetch User Profile

- Description: Ensures that a logged-in user can retrieve their profile information.
- Expected Outcome: HTTP 200 (OK), profile contains correct user information.
- Actual Outcome: Passed

Test Case 7: Update Profile Information

- Description: Checks if a user can update profile attributes such as wrapper_count and pack_count.
- Expected Outcome: HTTP 200 (OK), profile values updated in the database.
- Actual Outcome: Passed

Form Validation Tests

Test Case 8: Custom User Creation Form

- Description: Ensures that the form validation for user creation works correctly.
- Expected Outcome: Form should be valid when correct data is provided.
- Actual Outcome: Passed

Privacy Page Tests

Test Case 9: Privacy Page Rendering

- Description: Checks if the privacy page loads correctly.
- Expected Outcome: HTTP 200 (OK), correct template used (accounts/privacy.html)
- Actual Outcome: Passed

2. Backend Test Cases and Results

Achievement Testing

Test Case 1: Pack Achievement Award Test

Description: Verifies that a pack opening achievement is properly awarded when the user reaches the required threshold.

Steps:

1. Set the user's packs_opened count to 10 (matching the threshold)
2. Call the achievement service to check for PACKS achievements
3. Verify the achievement was awarded

Expected Results:

- Achievement service should return True indicating an award
- User's pack_count should increase by 1
- A PlayerAchievements record should exist for "Bronze Pack Opener"

Actual Results: All expectations met

Test Case 2: Pack Reward Test

Description: Verifies that the user receives the correct reward (an additional pack) upon earning an achievement.

Steps:

1. Set the user's packs_opened count to 10
2. Record initial pack_count
3. Call the achievement service
4. Verify the user's pack_count increased

Expected Results:

- Achievement service should return True
- User's pack_count should increase by exactly 1

Actual Results: All expectations met

Test Case 3: Duplicate Award Prevention Test

Description: Ensures that users cannot receive the same achievement (and rewards) multiple times.

Steps:

1. Set the user's packs_opened count to 10
2. Award the achievement once
3. Attempt to award the achievement again
4. Verify no additional rewards were given

Expected Results:

- First check should return True (achievement awarded)
- Second check should return False (no new achievements)
- Pack count should only increase once

Actual Results: All expectations met

Test Case 4. Invalid Stat Type Handling Test

Description: Tests the system's response to invalid achievement types.

Steps:

1. Call the achievement service with an invalid stat type
2. Verify appropriate error handling

Expected Results:

- A ValueError should be raised

Actual Results: Error properly raised

Test Case 5: Below Threshold Test

Description: Confirms that achievements are not awarded when the user hasn't reached the required threshold.

Steps:

1. Set the user's packs_opened count to 9 (just below threshold of 10)
2. Call the achievement service
3. Verify no achievement was awarded

Expected Results:

- Achievement service should return False
- No PlayerAchievements record should be created

Actual Results: All expectations met

Authentication and Access Control Testing

Test Case 1: Pack Page Access Tests

Description: Verifies access control for the packs page.

Test Case 1.1: Authenticated User Access

- Steps:
    1. Log in as the test user
    2. Attempt to access "/packs" endpoint
- Expected Result: User can access the page (HTTP 200 OK)
- Actual Result: Access granted with status code 200

Test Case 1.2: Unauthenticated User Access

- Steps:
    1. Without logging in, attempt to access "/packs" endpoint
- Expected Result: User is redirected to login page
- Actual Result: Redirected to "/accounts/login/?next=/packs" with status code 302

Test Case 2: Home Page Access Tests

Description: Verifies access control for the home page.

Test Case 2.1: Authenticated User Access

- Steps:
    1. Log in as the test user
    2. Attempt to access "/home" endpoint
- Expected Result: User can access the page (HTTP 200 OK)
- Actual Result: Access granted with status code 200

Test Case 2.2: Unauthenticated User Access

- Steps:
    1. Without logging in, attempt to access "/home" endpoint
- Expected Result: User is redirected to login page
- Actual Result: Redirected to "/accounts/login/?next=/home" with status code 302

Test Case 3: Profile Page Access Tests

Description: Verifies access control for the profile page.

Test Case 3.1: Authenticated User Access

- Steps:
    1. Log in as the test user
    2. Attempt to access "/profile" endpoint
- Expected Result: User can access the page (HTTP 200 OK)
- Actual Result: Access granted with status code 200

Test Case 3.2: Unauthenticated User Access

- Steps:
    1. Without logging in, attempt to access "/profile" endpoint
- Expected Result: User is redirected to login page
- Actual Result: Redirected to "/accounts/login/?next=/profile" with status code 302

4. Scanner Page Access Tests

Description: Verifies access control for the scanner page.

Test Case 4.1: Authenticated User Access

- Steps:
    1. Log in as the test user
    2. Attempt to access "/scanner/" endpoint
- Expected Result: User can access the page (HTTP 200 OK)
- Actual Result: Access granted with status code 200

Test Case 4.2: Unauthenticated User Access

- Steps:
    1. Without logging in, attempt to access "/scanner/" endpoint
- Expected Result: User is redirected to login page
- Actual Result: Redirected to "/accounts/login/?next=/scanner/" with status code 302

Test Case 5: Gym Battle Page Access Tests

Description: Verifies access control for the gym battle page.

Test Case 5.1: Authenticated User Access

- Steps:
    1. Log in as the test user
    2. Attempt to access "/gym-battle/1/" endpoint
- Expected Result: User can access the page (HTTP 200 OK)
- Actual Result: Access granted with status code 200

Test Case 5.2: Unauthenticated User Access

- Steps:
    1. Without logging in, attempt to access "/gym-battle/1/" endpoint
- Expected Result: User is redirected to login page
- Actual Result: Redirected to "/accounts/login/?next=/gym-battle/1/" with status code 302

Test Case 6: Gym Battle Completed Page Access Tests

Description: Verifies access control for the gym battle completed page.

Test Case 6.1: Authenticated User Access

- Steps:
    1. Log in as the test user
    2. Attempt to access "/gym-battle-completed/" with win and gym ID parameters
- Expected Result: User can access the page (HTTP 200 OK)
- Actual Result: Access granted with status code 200

Test Case 6.2: Unauthenticated User Access

- Steps:
    1. Without logging in, attempt to access "/gym-battle-completed" with parameters
- Expected Result: User is redirected to login page
- Actual Result: Returns 200 OK (possibly intentional for this endpoint)

Test Case 7: Change Deck Page Access Tests

Description: Verifies access control for the change deck page.

Test Case 7.1: Authenticated User Access

- Steps:
    1. Log in as the test user
    2. Attempt to access "/change_deck" endpoint
- Expected Result: User can access the page (HTTP 200 OK)
- Actual Result: Access granted with status code 200

Test Case 7.2: Unauthenticated User Access

- Steps:
    1. Without logging in, attempt to access "/change_deck" endpoint
- Expected Result: User is redirected to login page
- Actual Result: Redirected to "/accounts/login/?next=/change_deck" with status code 302

Battle System Testing

Test Case 1: View Gym Page Tests

Description: Verifies that the view-gym page renders correctly with proper context data.

Test Case 1.1: Authenticated User Access

- Steps:
  - Create a test user with profile
  - Create a test gym with assigned cards
  - Create player cards for the user
  - Log in as the test user
  - Request the view-gym page for the created gym
- Expected Results:
  - Response status code is 200 OK
  - Correct template is used ('backend/battles/view_gym.html')
  - Context contains gym data (ID, name, fact, coordinates, cooldown)
  - Context contains player's deck card data
- Actual Results: All expectations met
  - Response status code is 200
  - Correct template is used
  - Context contains all expected gym data
  - Context contains player's deck card slots (all null in this test)

Test Case 1.2: Unauthenticated User Access

- Steps:
  - Without logging in, attempt to access the view-gym page
- Expected Results:
  - User is redirected to login page
- Actual Results: Redirected to login page with correct "next" parameter

Test Case 2: Gym Battle Page Tests

Description: Verifies that the gym-battle page renders correctly with proper context data for battle gameplay.

Test Case 2.1: Authenticated User Access

- Steps:
  - Create a test user with profile
  - Create a test gym with assigned cards
  - Create player cards for the user
  - Assign cards to the user's deck
  - Log in as the test user
  - Request the gym-battle page for the created gym
- Expected Results:
  - Response status code is 200 OK
  - Correct template is used ('backend/battles/gym_battle.html')

    - Context contains player's deck card data (serialized as JSON)
    - Context contains gym's card data (serialized as JSON)
    - Context contains battle metadata (username, team names, gym ID)
- Actual Results: All expectations met
    - Response status code is 200
    - Correct template is used
    - Context contains all expected player deck card data
    - Context contains all expected gym card data
    - Context contains all expected battle metadata

Test Case 2.2: Unauthenticated User Access

- Steps:
    - Without logging in, attempt to access the gym-battle page
- Expected Results:
    - User is redirected to login page
- Actual Results: Redirected to login page with correct "next" parameter

Deck Views Tests

Test Case 1: Change Deck View Test

Description: Verifies that users can access the change deck page and that it contains the correct context data.

Specific Checks:

- GET request returns 200 status code
- Correct template is used ('backend/profile/changeDeck.html')
- Context contains necessary variables (cards, player_cards, deck cards)
- HTTP referrer is properly stored in session

Result: PASS

Test Case 2: Update Deck View Test

Description: Tests that users can update their deck by selecting cards.

Specific Checks:

- POST request with valid card selection redirects properly
- Redirect goes to the expected URL
- Profile is updated with selected cards in correct order
- Database reflects the changes to the user's deck

Result: PASS

Test Case 3: Clear Deck Test

Description: Verifies that submitting an empty card selection clears the user's deck.

Specific Checks:

- POST request with empty card selection processes successfully
- User profile is updated to have null values for all deck slots
- Database reflects the cleared deck state

Result: PASS

Test Case 4: Change Deck View Context Test

Description: Detailed verification of all context variables provided by the change deck view.

Specific Checks:

- Response status code is 200
- Correct template is used
- All required context variables are present:
    - 'cards' (available cards)
    - 'player_cards' (cards owned by player)
    - 'user' (current user)
    - 'deck_card_1', 'deck_card_2', 'deck_card_3' (current deck)
    - 'previous_page' (referrer information)

Result: PASS

Pack Functionality Tests

Test Case 1: Open Pack with Correct Settings

Description: Verifies that users can access the pack opening page when they have a pack available and no wrappers.

Specific Checks:

- User has 1 pack and 0 wrappers
- GET request to '/opening_pack' returns 200 status code
- Correct template is used ('backend/packs/opening_pack.html')

Result: PASS

Test Case 2: Open Pack with No Packs

Description: Tests the system's behavior when a user attempts to open a pack without having any available.

Specific Checks:

- User has 0 packs and 0 wrappers
- GET request to '/opening_pack' returns 302 status code (redirect)
- User is redirected to '/packs'
- Redirected page returns 200 status code
- Correct template is used ('backend/packs/nopacks.html')

Result: PASS

Test Case 3: Open Pack with Full Bin

Description: Verifies that users with a full wrapper bin (10 wrappers) cannot open packs even if they have packs available.

Specific Checks:

- User has 1 pack and 10 wrappers
- GET request to '/opening_pack' returns 302 status code (redirect)
- User is redirected to '/packs'
- Redirected page returns 200 status code
- Correct template is used ('backend/packs/bin_full.html')

Result: PASS

Test Case 4: Open Pack with No Pack and Empty Bin

Description: Tests the system's response when a user has neither packs nor wrappers.

Specific Checks:

- User has 0 packs and 0 wrappers
- GET request to '/opening_pack' returns 302 status code (redirect)
- User is redirected to '/packs'
- Redirected page returns 200 status code
- Correct template is used ('backend/packs/nopacks.html')

Result: PASS

Player Leaderboard View Tests

Test Case 1: Authentication Requirement Test

Description: Verifies that unauthenticated users are redirected to the login page.

Specific Checks:

- GET request to '/player_leaderboard' without login returns a redirect response
- User is redirected to the login page with appropriate 'next' parameter

Result: PASS

Test Case 2: Template and Context Test

Description: Tests that the view renders the correct template and includes all necessary context data.

Specific Checks:

- Authenticated user receives 200 status code
- Correct template is used ('backend/leaderboard/player_leaderboard.html')
- Context contains 'players' key with expected data
- Player data includes correct username, team name, and statistics

Result: PASS

Test Case 3: Team Exclusion Test

Description: Verifies that players from the "Fossil Fuels" team are excluded from the leaderboard.

Specific Checks:

- Players from "Recycle" team appear in the leaderboard
- Players from "Fossil Fuels" team do not appear in the leaderboard

Result: PASS

Test Case 4: Ordering Test

Description: Tests that players are correctly ordered by the number of gyms they own.

Specific Checks:

- Three users with different gym counts (5, 2, 1) are created
- Players appear in descending order based on gym count
- Ordering matches expected sequence (user2, user1, user3)

Result: PASS

Test Case 5: Card Count Test

Description: Verifies that card counts (by type and total) are correctly calculated.

Specific Checks:

- Player with specific card distribution (1 plastic, 1 recycle, 1 plant) is created
- Card counts in player data match expected values
- Collection total correctly sums all cards

Result: PASS

Test Case: 6. Pagination Test

Description: Tests that the leaderboard is limited to the top 10 players even when more exist.

Specific Checks:

- 15 users with varying gym counts are created
- Leaderboard contains exactly 10 players
- First player has the highest gym count (14)

Result: PASS

Scanner Functionality Tests

Test Case 1: Scanner Template Rendering Test

Description: Verifies that an authenticated user can access the scanner page and that the correct template is rendered.

Specific Checks:

- Authenticated user receives 200 status code when accessing '/scanner/'
- Correct template is used ('backend/scanner/scanner.html')

Result: PASS

Test Case 2: Authentication Requirement Test

Description: Tests that unauthenticated users are redirected to the login page when attempting to access the scanner.

Specific Checks:

- Unauthenticated GET request to '/scanner/' returns a redirect response (302)
- User is redirected to the login page with appropriate 'next' parameter
- Redirect URL is correctly formed as '/accounts/login/?next=/scanner/'

Result: PASS

Team Leaderboard View Tests

Test Case 1: Authentication Requirement Test

Description: Verifies that unauthenticated users are redirected to the login page.

Specific Checks:

- GET request to '/team_leaderboard' without login returns a redirect response
- User is redirected to the login page with appropriate 'next' parameter

Result: PASS

Test Case 2: Template and Context Test

Description: Tests that the view renders the correct template and aggregates data properly for a team with a single gym.

Specific Checks:

- Authenticated user receives 200 status code
- Correct template is used ('backend/leaderboard/team_leaderboard.html')
- Context contains 'teams' key with expected data
- Only user-selectable teams are included
- Gym count is correct (1)
- Card type counts are correct (1 of each type)
- Team icon URL is correctly formatted

Result: PASS

Test Case 3: Multiple Gyms Test

Description: Verifies that the view correctly aggregates data for a team with multiple gyms and varied card combinations.

Specific Checks:

- Team with three gyms is correctly counted
- Cards from all gyms are properly aggregated:
  - 3 plastic cards (from first gym)
  - 3 recycle cards (from second gym)
  - 3 plant cards (from third gym)

Result: PASS

Test Case 4: Zero Gyms Test

Description: Tests that a team with no gyms still appears in the leaderboard with zero counts.

Specific Checks:

- Team with no gyms is included in context
- Gym count is correctly shown as 0
- All card type counts are correctly shown as 0

Result: PASS

Manual UX Testing: Frontend Development

For our frontend development, we conducted manual UX testing (User Acceptance Testing) rather than implementing automated testing, for example Selenium. While we researched front-end testing methods as potential automated testing solutions, however, this sort of testing is out of scope, due to the scale of the project. Instead, we performed comprehensive manual testing to verify that all elements loaded correctly across our Django-powered application. This manual approach allowed us to thoroughly examine the templates, confirming proper rendering and functionality of UI components, responsive behavior across different screen sizes, and consistent styling throughout the application. Our manual testing processes, though more time-intensive than an automated solution, proved effective in identifying and resolving frontend issues before deployment.