Test Strategy and Report

This document explains the testing strategy used for the UserController to ensure its functionality and security are working as expected.

1. Testing Goal

The primary goal of these tests is to verify the behavior of the UserController at the API level. We want to confirm two main things:

1. **Security:** Are the endpoint permissions enforced correctly? (e.g., Can a USER access ADMIN endpoints?)
2. **Functionality:** Do the endpoints return the correct HTTP status codes and content when accessed with the proper permissions?

2. Testing Approach & Tools

We are using an **integration testing** approach. This means we load the entire Spring application context, including our security configuration, to create a test environment that is as close to the real application as possible.

- **@SpringBootTest**: This annotation is the key. It boots up our whole application for the test.
- **@AutoConfigureMockMvc**: This configures a MockMvc object, which is a powerful tool for simulating HTTP requests (like GET and POST) to our API without needing to start a real web server.
- **@MockBean**: While we load the whole application, we don't want our tests to depend on an actual database. @MockBean allows us to replace the real UserRepository with a "mock" or "fake" version that we can control directly in our tests.

3. Test Case Breakdown

Our test class, UserControllerTest, contains several test cases, each verifying a specific scenario.

Public and Authenticated Access Tests

- **whenGetPublicEndpoint_thenSucceed()**
  - **Scenario:** A request is made to GET /public.
  - **Setup:** No user authentication is needed.
  - **Verification:** Checks that the server responds with 200 OK and the correct text.
- **whenGetUserEndpointAsUser_thenSucceed()**

    - **Scenario:** A USER tries to access the GET /user endpoint.
    - **Setup:** @WithMockUser(roles = "USER") simulates a logged-in user with the USER role.
    - **Verification:** Checks for a 200 OK status.
- **whenGetAdminEndpointAsAdmin_thenSucceed()**
    - **Scenario:** An ADMIN tries to access the GET /admin endpoint.
    - **Setup:** @WithMockUser(roles = "ADMIN") simulates a logged-in admin.
    - **Verification:** Checks for a 200 OK status.

Security and Authorization Failure Tests

- **whenGetAdminEndpointAsUser_thenIsForbidden()**
    - **Scenario:** A regular USER attempts to access the admin-only endpoint GET /admin.
    - **Setup:** @WithMockUser(roles = "USER").
    - **Verification:** This is a crucial security test. It asserts that the server correctly responds with 403 Forbidden.
- **whenGetAdminEndpointUnauthenticated_thenIsUnauthorized()**
    - **Scenario:** An unauthenticated (not logged in) user tries to access GET /admin.
    - **Verification:** Asserts that the server responds with 401 Unauthorized.

User Creation Tests (POST /users)

- **whenCreateUserAsAdmin_thenSucceed()**
    - **Scenario:** An ADMIN creates a new user.
    - **Setup:**
        - @WithMockUser(roles = "ADMIN") simulates the admin user.
        - We use when(userRepository.save(...)).thenReturn(...) to tell our mock repository what to do when the save method is called.
    - **Verification:** Asserts that the server responds with 201 Created, indicating success.
- **whenCreateUserAsUser_thenIsForbidden()**
    - **Scenario:** A regular USER attempts to create another user, which should not be allowed.
    - **Setup:** @WithMockUser(roles = "USER").
    - **Verification:** Asserts that the server correctly denies the request with a 403 Forbidden status.