# Note-Bridge Software Testing Report

Group 1

Rebah Ozkoc 3363775

Anil Sen 3367606

Ilgin Simay Ozcan 3362779

Enescu Alexandru-Cristian - 3055779

Simeon Nikolov - 3163989

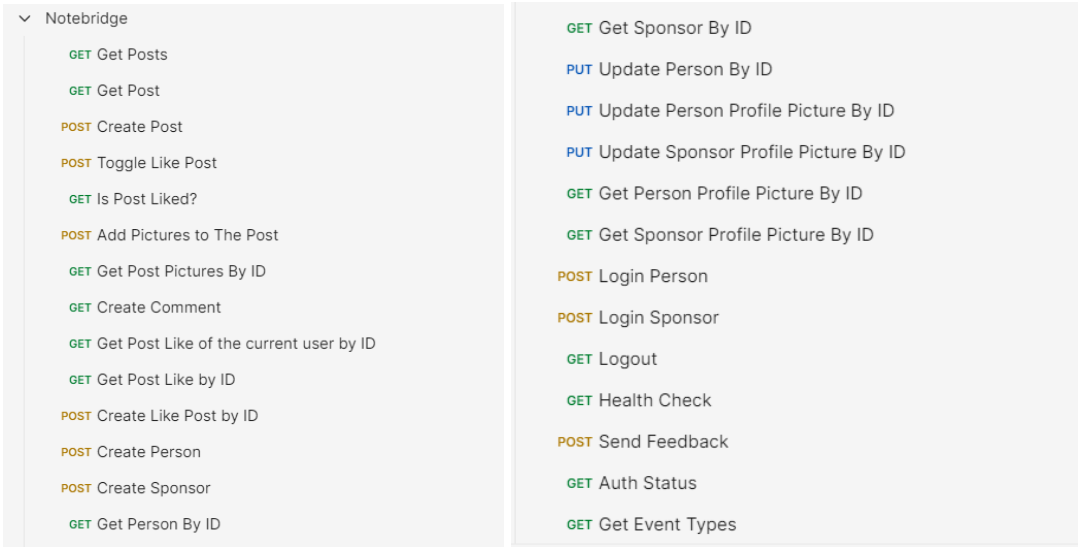
Adham Ehab Magdy Selim - 2948486

21/06/2024

*Content Table*

# Software Testing

During the development of our project, we continuously tested our functionalities with the Postman application by sending different requests to the backend endpoints. This allows us to see which functionalities are working properly. Here is the list of the endpoints that we use regularly.




Currently, we are trying to use the test-driven development approach to reduce unnecessary coding and reduce the number of bugs. At the start of the project, we did not follow this approach but after we realized the benefits of this method we started to use this approach. Now we create the unit tests first and then develop the actual functionality

For example here the update post method is not ready. However, we have a unit test for this method which expects the changed post after calling this method.

```
public Post update(Post updated) {   Rebah Özkoç *
    // TODO implement this method

    return updated;
}
```

```
@Test   Rebah Özkoç
@Order(5)
public void stage5_testUpdatePost() {
    post.setTitle("Updated Post");
    post.setDescription("Updated Description");
    Post updatedPost = PostDao.INSTANCE.update(post);
    assertEquals(updatedPost.getTitle(), post.getTitle(),  message: "Post title should be updated");
    assertEquals(updatedPost.getDescription(), post.getDescription(),  message: "Post description should be updated");
}
```

# Unit Testing With JUnit

We implemented unit tests with JUnit library for the core methods to ensure that the application works correctly all the time. For this testing, we created a new schema named under our database which has the exact tables with the deployment schema. This schema does not have any data in it and is used for testing.

Before running the tests we connect to this database with the setUpAll method. This method also cleans the related tables for that class. For example, in the code below, we set up the environment for the Base User unit tests.

```
@BeforeAll  Rebah Özkoç
public static void setUpAll() {
    try {
        DatabaseConnection.INSTANCE.load( deploymentMode: false);

        System.out.println("Working Directory = " + System.getProperty("user.dir"));
    } catch (Exception e) {
        System.err.println("Error while loading data.");
        e.printStackTrace();
    }
    System.out.println("Notebridge TEST initialized.");

    // Clean the database
    BaseUserDao.INSTANCE.deleteAll();
}
```

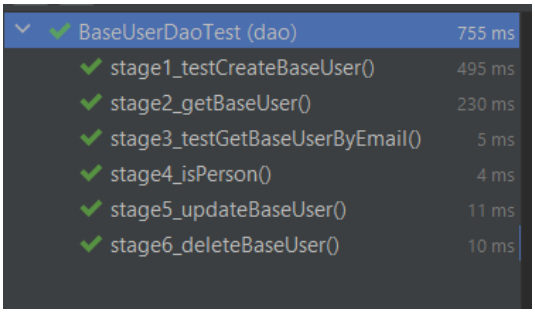
After that, the tests are run for the class. For example, the test for like functionality is given below. It checks if the post is liked before calling the function and after calling the function and controls the result with assert statements.

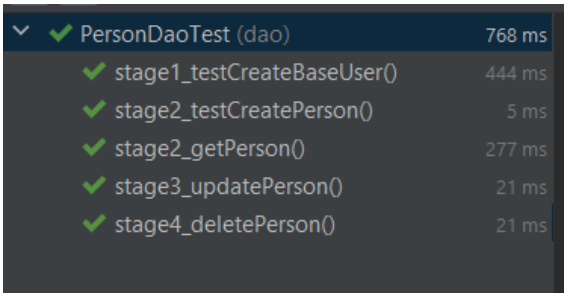
```
@Test  Rebah Özkoç
@Order(5)
public void stage5_testIsLikedAfterLike(){

    Like like = new Like();
    like.setPersonId(person.getId());
    like.setPostId(post.getId());
    Boolean isLiked = LikeDao.INSTANCE.isLiked(post.getId(), person.getId());

    assertEquals(isLiked,  actual: false,  message: "Post should not be liked");

    Like createdLike = PostDao.INSTANCE.toggleLike(like);
    assertEquals(createdLike.getPersonId(), person.getId(),  message: "Person id should be the same");
    assertEquals(createdLike.getPostId(), post.getId(),  message: "Post id should be the same");

    isLiked = LikeDao.INSTANCE.isLiked(post.getId(), person.getId());
    assertEquals(isLiked,  actual: true,  message: "Post should be liked");
}
```

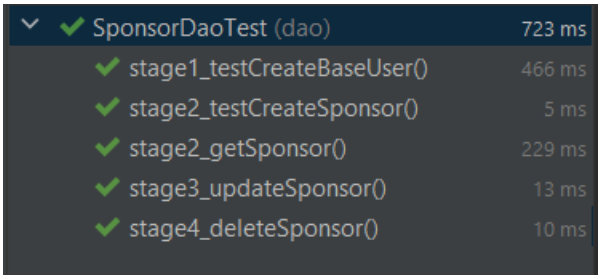
After all of the tests, tearDownAll method is called which closes the database connection and removes all the data from the respective tables.

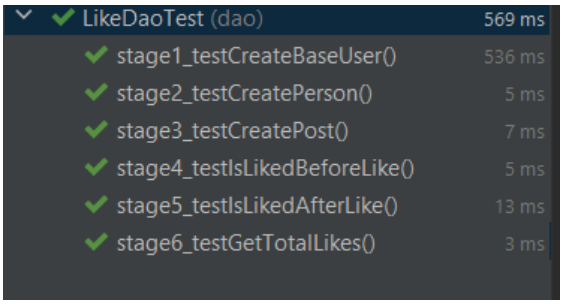
```java
@AfterAll  Rebah Özkoç
public static void tearDownAll() {

    BaseUserDao.INSTANCE.deleteAll();

    try {
        DatabaseConnection.INSTANCE.getConnection().close();
    } catch (Exception e) {
        System.err.println("Error while closing the database connection.");
        e.printStackTrace();
    }
    System.out.println("Notebridge TEST shutdown.");
}
```

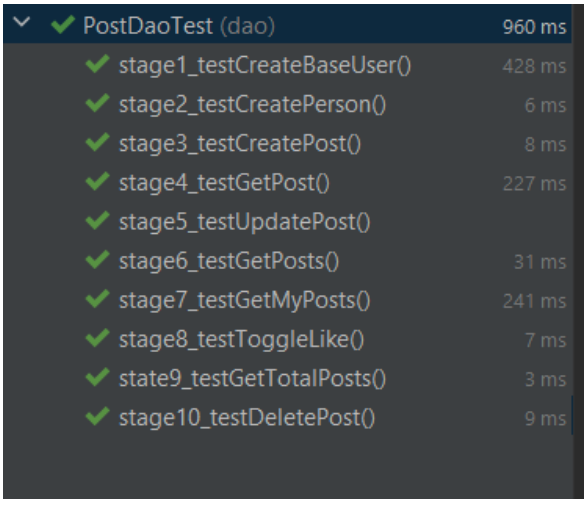
Here are some unit tests that we are using to test the application.



We are using the “Code Coverage for Java” plugin of JetBrains to see how much of the code is covered with these unit tests.

## Benefits Of Testing

Testing allows us to discover the bugs in our software and solve them as soon as possible. There are a couple of examples where we discovered bugs thanks to our unit tests.

While testing our strength against stored XSS, we noticed that we forgot to sanitize email during the creation of users. By also sanitizing email, we prevented any potential stored XSS attack, since we frequently show users’ email on several pages of our app.

Thanks to that, we made sure that whatever is inputted by the user is not the same as the one returned by the database after a successful sanitization technique, which is explained in our “Security Analysis Report”.

```
@Test  new *
@Order(7)
public void stage1_testCreateMaliciousBaseUser() {
    BaseUser maliciousBaseUser1 = BaseUserDaoTest.maliciousBaseUser;

    BaseUser userStoredInDatabase=BaseUserDao.INSTANCE.getUser(BaseUserDao.INSTANCE.create(maliciousBaseU

    System.out.println(userStoredInDatabase.getUsername());
    System.out.println(userStoredInDatabase.getEmail());

    assertTrue(!userStoredInDatabase.getUsername().equals(maliciousBaseUser1.getUsername()));
    assertTrue(!userStoredInDatabase.getEmail().equals(maliciousBaseUser1.getEmail()));

}
```

Another example is that we realized a bug in the personLikesPost table which prevented us from deleting posts if the post had any likes with the help of unit tests. We fixed this error by changing the database schema.

## User story testing

1. As a user, I want to create posts, so that I can indicate my desires for a specific event.
2. As a user, I want to browse posts, so that I can find events I would like to participate in and connect with other musicians.
3. As a user, I want to see an extended description of each card, so that I can find out more information.
4. As a user, I want to create an account, so that I can easily access all features.
5. As a user, I want to log in, so that I can use my account.
6. As a user, I want to like posts, so that I can show my interest and interact with the community.
7. As a user, I want to provide feedback, so that I can express my opinions and suggestions.

8. As a user, I want to edit posts, so that I can change any details or add more information.
9. As a user, I want to delete posts, so that the information I posted is removed if I want.
10. As a user, I want to be able to indicate that I am interested in a certain post/event.

# Selenium Testing

Each page of the web application has been properly tested using Selenium. The tests can be found under the Test folder located in the Project folder.

## User Story #1 - Create a post

We have tested the form used to create posts in the following way: when users submit it, they are redirected to the cards page. Then, we checked that the most recently created card has the same title as the text which was added to the 'title' field of the form. In this way, we know that the post has been successfully created.

```
driver.get("http://localhost:8080/notebridge/create-card.html");
WebElement title = driver.findElement(By.id("title"));
WebElement description = driver.findElement(By.id("description"));
WebElement location = driver.findElement(By.id("location"));
WebElement createPostBtn = driver.findElement(By.id("create-post-btn"));

title.sendKeys( ...keysToSend: "New post title");
description.sendKeys( ...keysToSend: "New post description");
location.sendKeys( ...keysToSend: "New post location");
createPostBtn.click();

WebDriverWait wait = new WebDriverWait(driver, Duration.ofSeconds(5));
wait.until(ExpectedConditions.urlContains( fraction: "http://localhost:8080/notebridge/cards.html"));

String expectedPageUrl = "http://localhost:8080/notebridge/cards.html";
String actualPageUrl = driver.getCurrentUrl();
assertEquals(expectedPageUrl, actualPageUrl);

WebElement cardTitle = driver.findElement(By.cssSelector("#displayed-card > div > h5"));
assertEquals(title.getText(), cardTitle.getText());
```

## User Story #2 - Browsing Posts

This was tested by creating a selenium test class that ensures that the main forum page contains elements (posts) of a certain type (cards). Furthermore, the "load more" button was tested to make sure that clicking the button would load more posts into the page. This was a simple check that compared the size of the "cards" div before and after loading more cards into the page. If loaded correctly, the height of the div expands to allow for the new cards, which is what we test for below.

```java
adham
@Test
public void loadCardsTest() throws InterruptedException {
    driver.get("http://localhost:8080/notebridge/cards.html");
    TimeUnit.SECONDS.sleep( timeout: 3);

    WebElement load = driver.findElement(By.id("load-more-btn"));
    WebElement cardsContainer = driver.findElement(By.id("cards"));
    Dimension len1 = cardsContainer.getSize();

    new Actions(driver).moveToElement(load).click().perform();
    TimeUnit.SECONDS.sleep( timeout: 5);

    Dimension len2 = cardsContainer.getSize();

    assertTrue( condition: len2.height > len1.height);

}
```

## User Story #3 - Expanding Posts

Using selenium we are able to create a test class that navigates the cards page and clicks on one of the posts. After clicking the post we ensure that we are redirected to the correct card-details URL

```java
adham
@Test
public void openCardTest() throws InterruptedException {
    driver.get("http://localhost:8080/notebridge/cards.html");
    TimeUnit.SECONDS.sleep( timeout: 3);

    WebElement card = driver.findElement(By.id("displayed-card"));
    new Actions(driver).moveToElement(card).click().perform();
    TimeUnit.SECONDS.sleep( timeout: 3);

    String cardurl = driver.getCurrentUrl().split( regex: "=")[0];


    assertEquals( expected: "http://localhost:8080/notebridge/card-details.html?id", cardurl);
}
```

.

## User Story #4 - Create an account

We performed multiple tests on the sign-up form on the Register page. Firstly, we tested the positive scenario that a user completes correctly all the required fields and successfully creates an account. For this, we checked that when the required fields are properly filled, the user is redirected to the Login page, and an alert is shown on the screen with a corresponding message which confirms the creation of a new account.

```
firstName.sendKeys( ...keysToSend: "Test first name");
lastName.sendKeys( ...keysToSend: "Test last name");
username.sendKeys( ...keysToSend: "Test username 5");
phoneNumber.sendKeys( ...keysToSend: "123456789");
emailField.sendKeys( ...keysToSend: "email105@example.com");
passwordField.sendKeys( ...keysToSend: "Superpassword.123");
repeatPasswordField.sendKeys( ...keysToSend: "Superpassword.123");
registerBtn.click();

WebDriverWait wait = new WebDriverWait(driver, Duration.ofSeconds(5));
wait.until(ExpectedConditions.alertIsPresent());
Alert alert = driver.switchTo().alert();
String alertText = alert.getText();
alert.accept();

if (alertText != null) {
    System.out.println("Alert is shown on the screen, its content is: " + alertText);
} else {
    System.out.println("The alert did not appear.");
}

String expectedPageUrl = "http://localhost:8080/notebridge/login.html";
String actualPageUrl = driver.getCurrentUrl();
assertEquals(expectedPageUrl, actualPageUrl);
```

Then, we check if the application responds appropriately if the user does not enter a unique username. In this scenario, a warning with a specific message must be shown on the screen. Another test is performed to check what happens if a user enters an email which is not unique. Similarly, a warning with a specific message for this case is displayed.
We also test the situation in which users do not fill the fields 'password' and 'repeat password' with the same content. The users are informed about their errors in a similar way. In all three cases, the form is not submitted and their data is not sent to the server.

```
firstName.sendKeys( ...keysToSend: "Test first name");
lastName.sendKeys( ...keysToSend: "Test last name");
username.sendKeys( ...keysToSend: "username105");
phoneNumber.sendKeys( ...keysToSend: "123456789");
emailField.sendKeys( ...keysToSend: "alex@example.com");
passwordField.sendKeys( ...keysToSend: "Superpassword.123");
repeatPasswordField.sendKeys( ...keysToSend: "NotMatchingPassword");
registerBtn.click();

WebDriverWait wait = new WebDriverWait(driver, Duration.ofSeconds(5));
wait.until(ExpectedConditions.visibilityOf(warningMessage));

assertTrue(warningMessage.isDisplayed());

if(warningMessage.isDisplayed()) {
    System.out.println("Warning message is displayed, its content is: " + warningMessage.getText());
} else {
    System.out.println("Warning message is not displayed");
}
```

## User Story #5 - Login

Firstly, when the page is opened, it must display the correct title. For the Login page, different tests have been carried out. We checked if the form for logging in works correctly in the following way: when users enter the correct credentials, the web application redirects them to the Home page, where the 'Log out', 'Messenger', and 'Profile' buttons are displayed in the navigation bar. In case users do not enter the correct credentials, a warning message must be shown under the password field. The web application has successfully passed these tests.

```java
public void testLoginFormCorrectCredentials() {
    WebElement emailField = driver.findElement(By.id("email"));
    WebElement passwordField = driver.findElement(By.id("password"));
    WebElement loginBtn = driver.findElement(By.id("login-button"));

    emailField.sendKeys( ...keysToSend: "alex@example.com");
    passwordField.sendKeys( ...keysToSend: "Superpassword.123");
    loginBtn.click();

    WebDriverWait wait = new WebDriverWait(driver, Duration.ofSeconds(5));
    wait.until(ExpectedConditions.urlContains( fraction: "http://localhost:8080/notebridge/home.html"));

    String expectedPageUrl = "http://localhost:8080/notebridge/home.html";
    String actualPageUrl = driver.getCurrentUrl();
    assertEquals(expectedPageUrl, actualPageUrl);

    WebElement logoutBtn = driver.findElement(By.id("log-out-btn"));
    WebElement messengerBtn = driver.findElement(By.id("messenger-btn"));
    WebElement profileBtn = driver.findElement(By.id("profile-btn"));

    assertTrue(logoutBtn.isDisplayed());
    assertTrue(messengerBtn.isDisplayed());
    assertTrue(profileBtn.isDisplayed());
}
```

## User Story #5 - Accessing & Updating Account Details

In these tests, for setting up the test, the user logs in to their account and is navigated back to the home page, which is either Person or Sponsor.

```java
@BeforeClass  Anil *
public static void setUpAndLogIn() {
    try {
        System.setProperty("webdriver.chrome.driver", Utils.readFromProperties( key: "SELENIUM_DRIVER_PATH"));
        driver = new ChromeDriver();
        driver.get("http://localhost:8080/notebridge/login.html");
        WebDriverWait wait = new WebDriverWait(driver, Duration.ofSeconds(15));

        WebElement email = wait.until(ExpectedConditions.elementToBeClickable(By.id("email")));
        WebElement pass = wait.until(ExpectedConditions.elementToBeClickable(By.id("password")));
        WebElement login = wait.until(ExpectedConditions.elementToBeClickable(By.cssSelector("#login-form but

        email.sendKeys( ...keysToSend: "crazyuserboy2@gmail.com");
        pass.sendKeys( ...keysToSend: "superpasspword");
        login.click();

    } catch (Exception e) {
        System.err.println("Exception caught: " + e.getMessage());
        e.printStackTrace();
    }
}
```

After that, the user navigates to the profile page to check if they are able to see their account's details on the profile page. We focused on the element showing the username of the account and if it is set to "username", which is the default value showing up whenever it

is intended to be accessed by unauthorized users, then assert fails. In case assert does not fail, we ensure that the account data is fetched successfully and the logged in user is able to see it.

```
@Test  Anil *
@Order(1)
public void accessProfilePage(){

    driver.get("http://localhost:8080/notebridge/profile.html");
    WebDriverWait wait = new WebDriverWait(driver, Duration.ofSeconds(15));


    WebElement profileUsername = wait.until(ExpectedConditions.elementToBeClickable(By.cssSelector("#userna

    //If user does not log in successfully, then the profile page prints out default username
    //Being not equal to this, means the user has logged in successfully
    assertTrue( condition: profileUsername.getText()!="@username");


}
```

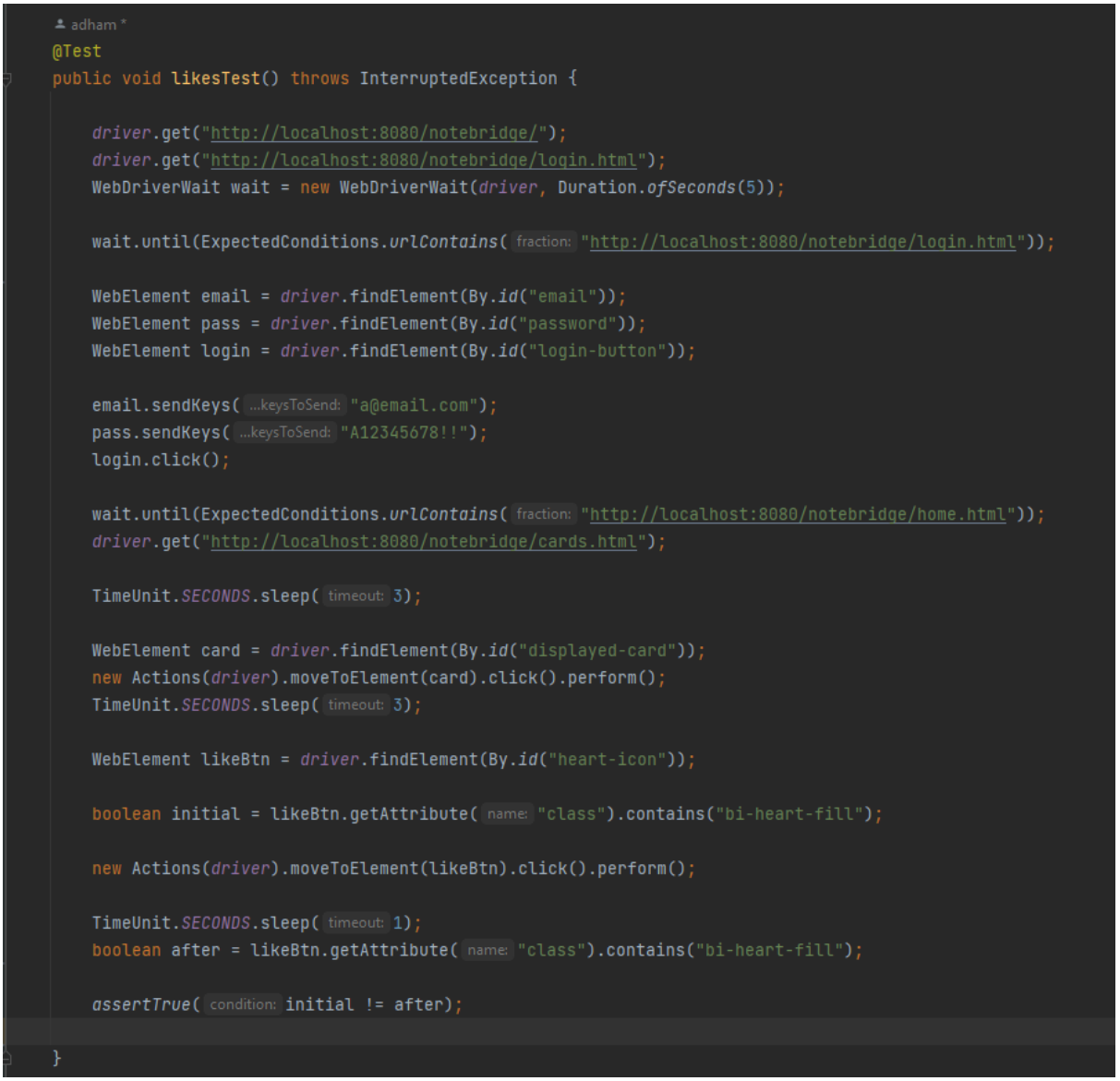
As a second step, we try to further check if the logged-in user is able to update their account details such as “Name-Lastname” for Person and “CompanyName-WebsiteURL” for Sponsor.

To do that, we reach out to relevant edit buttons and insert our changes. Then we check if we get an alert indicating that our update is successful after clicking on the “Save Changes” button. If so, we conclude that the test is successful and both Person and Sponsor entities are able to update their data.

```java
@Test  Anil *
@Order(2)
public void tryEditingName(){

    driver.get("http://localhost:8080/notebridge/profile.html");
    WebDriverWait wait = new WebDriverWait(driver, Duration.ofSeconds(15));


    WebElement editBtn = wait.until(ExpectedConditions.elementToBeClickable(By.cssSelector(".profile-he
    editBtn.click();

    WebElement nameInput = wait.until(ExpectedConditions.elementToBeClickable(By.cssSelector("#nameInpu
    WebElement lastnameInput=wait.until(ExpectedConditions.elementToBeClickable(By.cssSelector("#lastna
    WebElement saveChangesBtn=wait.until(ExpectedConditions.elementToBeClickable(By.cssSelector("#editP


    // Clear and interact with nameInput
    nameInput.clear();
    nameInput.clear();
    lastnameInput.clear();
    nameInput.sendKeys( ...keysToSend: "CHANGED");
    lastnameInput.sendKeys( ...keysToSend: "Changed");
    saveChangesBtn.click();
    //alert showing up saying update successfull
    Alert alert = wait.until(ExpectedConditions.alertIsPresent());

    String alertText= alert.getText();
    driver.switchTo().alert().accept();
    assertTrue(alertText.contains("Update successful"));
```

## User Story #6 - Liking Posts

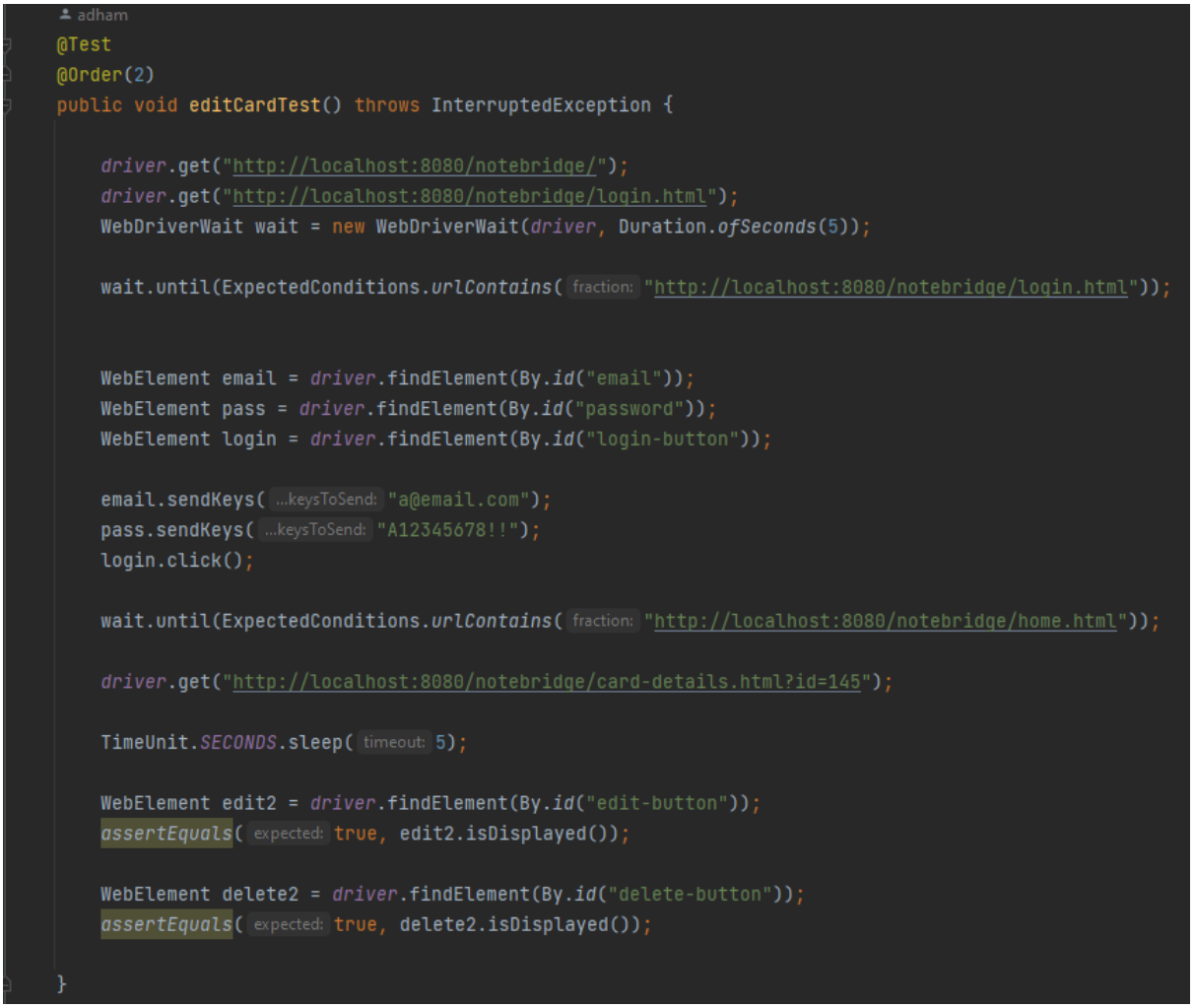
A large test class was created, starting off with signing in and navigating to a post. First, we store the initial state of the like button using its class attributes. After selenium “likes” the post, the state of the button is checked again and then compared to the initial state. The test ensures the two states are always opposite. Proper waiting and delays are essential during these tests to give the site enough time to load and process data and correctly test it after it has been loaded fully.



```
adham *
@Test
public void likesTest() throws InterruptedException {

    driver.get("http://localhost:8080/notebridge/");
    driver.get("http://localhost:8080/notebridge/login.html");
    WebDriverWait wait = new WebDriverWait(driver, Duration.ofSeconds(5));

    wait.until(ExpectedConditions.urlContains( fraction: "http://localhost:8080/notebridge/login.html"));

    WebElement email = driver.findElement(By.id("email"));
    WebElement pass = driver.findElement(By.id("password"));
    WebElement login = driver.findElement(By.id("login-button"));

    email.sendKeys( ...keysToSend: "a@email.com");
    pass.sendKeys( ...keysToSend: "A12345678!!");
    login.click();

    wait.until(ExpectedConditions.urlContains( fraction: "http://localhost:8080/notebridge/home.html"));
    driver.get("http://localhost:8080/notebridge/cards.html");

    TimeUnit.SECONDS.sleep( timeout: 3);

    WebElement card = driver.findElement(By.id("displayed-card"));
    new Actions(driver).moveToElement(card).click().perform();
    TimeUnit.SECONDS.sleep( timeout: 3);

    WebElement likeBtn = driver.findElement(By.id("heart-icon"));

    boolean initial = likeBtn.getAttribute( name: "class").contains("bi-heart-fill");

    new Actions(driver).moveToElement(likeBtn).click().perform();

    TimeUnit.SECONDS.sleep( timeout: 1);
    boolean after = likeBtn.getAttribute( name: "class").contains("bi-heart-fill");

    assertTrue( condition: initial != after);

}
```

## User Story #8 + 9 - Editing and Deleting your post(s)

Currently, the functionality to edit and delete a post has not been fully implemented. However, we have the front end of these buttons ready and tested. It is extremely important that these buttons only show up to the author of the card when they are logged in and that is it. To test this, we use Selenium to log in and go to a card created by that account, where it then checks for the existence of these buttons. Normally, the buttons do not exist on the HTML file unless the author of the card is the person visiting.



```
adham
@Test
@Order(2)
public void editCardTest() throws InterruptedException {

    driver.get("http://localhost:8080/notebridge/");
    driver.get("http://localhost:8080/notebridge/login.html");
    WebDriverWait wait = new WebDriverWait(driver, Duration.ofSeconds(5));

    wait.until(ExpectedConditions.urlContains( fraction: "http://localhost:8080/notebridge/login.html"));


    WebElement email = driver.findElement(By.id("email"));
    WebElement pass = driver.findElement(By.id("password"));
    WebElement login = driver.findElement(By.id("login-button"));

    email.sendKeys( ...keysToSend: "a@email.com");
    pass.sendKeys( ...keysToSend: "A12345678!!");
    login.click();

    wait.until(ExpectedConditions.urlContains( fraction: "http://localhost:8080/notebridge/home.html"));

    driver.get("http://localhost:8080/notebridge/card-details.html?id=145");

    TimeUnit.SECONDS.sleep( timeout: 5);

    WebElement edit2 = driver.findElement(By.id("edit-button"));
    assertEquals( expected: true, edit2.isDisplayed());

    WebElement delete2 = driver.findElement(By.id("delete-button"));
    assertEquals( expected: true, delete2.isDisplayed());

}
```

## User Story #8 + 9 - Interested in a post.

The “I’m interested” button is very simple but reacts differently in certain circumstances. Currently, the backend has not been fully implemented. The button is visible to everyone on every card, even when not logged in. The main difference is that, when a logged-out user attempts to use the button, they are instead redirected to the login page where they must log in first.

```java
adham
@Test
@Order(1)
public void InterestedTest() throws InterruptedException {
    driver.get("http://localhost:8080/notebridge/");
    driver.get("http://localhost:8080/notebridge/card-details.html?id=180");

    WebElement button = driver.findElement(By.id("interested-button"));
    button.click();

    TimeUnit.SECONDS.sleep( timeout: 2);
    assertEquals( expected: "http://localhost:8080/notebridge/login.html", driver.getCurrentUrl());
    TimeUnit.SECONDS.sleep( timeout: 2);
}
```

## Home page tests

On the Home page, the functionality of the buttons on the navigation bar has been tested. When clicked, the 'Browse Posts' and 'Login' buttons must redirect users to the appropriate pages. Also, the 'Contact Us' button must move the view to the 'Contact Us' form. Furthermore, when users are not logged in, they are not supposed to see buttons such as 'Messenger' or 'Profile'. We carried out tests for all these cases.

The 'Contact Us' form has also been successfully tested: when users fill in its fields, an alert is shown on the screen with a confirmation message.

```java
/**
 * Tests if the Login button redirects the user to the Login page.
 */
@Test  AlexEnescu
@Order(3)
public void testLoginBtn() {
    WebElement loginBtn = driver.findElement(By.id("login-btn"));
    loginBtn.click();
    String expectedPageUrl = "http://localhost:8080/notebridge/login.html";
    String actualPageUrl = driver.getCurrentUrl();
    assertEquals(expectedPageUrl, actualPageUrl);
}

/**
 * Tests if the Profile button is hidden when the user is logged out.
 */
@Test  AlexEnescu *
@Order(4)
public void testHiddenProfileBtn() {
    WebElement profileBtn = driver.findElement(By.id("profile-btn"));

    assertFalse(profileBtn.isDisplayed());

    if(profileBtn.isDisplayed()){
        System.out.println("Profile button is displayed.");
    } else {
        System.out.println("Profile button is not displayed.");
    }
```

```java
public void testContactUsForm() {
    driver.get("http://localhost:8080/notebridge/#feedback-form-container");
    WebElement emailField = driver.findElement(By.id("email"));
    WebElement messageField = driver.findElement(By.id("message"));
    WebElement sendMessageBtn = driver.findElement(By.id("send-message-btn"));

    emailField.sendKeys( ...keysToSend: "testEmailField@example.com");
    messageField.sendKeys( ...keysToSend: "testMessageField");
    sendMessageBtn.sendKeys(Keys.RETURN);

    WebDriverWait wait = new WebDriverWait(driver, Duration.ofSeconds(5));
    wait.until(ExpectedConditions.alertIsPresent());
    Alert alert = driver.switchTo().alert();
    String alertText = alert.getText();
    alert.accept();

    if (alertText != null) {
        System.out.println("Alert is shown on the screen, its content is: " + alertText);
    } else {
        System.out.println("The alert did not appear.");
    }
}
```