STIRLING EDUCATION

IT 303 – Software Verification, Validation and Testing

TESTING DOCUMENTATION

Github Automation Testing

Prepared by:
Faris Muhović

IBU | International Burch University

Proposed to:
Samed Jukić, Assist. Prof. Dr.
Adnan Miljković, Teaching Assistant
Benjamin Peljto, Lecture Assistant

5th of January 2025

TABLE OF CONTENTS

Contents

1. Introduction

1.1. About the Project

This project involves testing the **GitHub web application**, focusing on both core and advanced functionalities using Selenium and JUnit in Java. GitHub serves as a platform for version control, collaboration, and repository management, offering tools to manage code, track issues, and collaborate effectively. As part of this project, a comprehensive QA automation suite was implemented to evaluate GitHub's performance and reliability. The suite includes automated tests for user authentication, form submissions, search functionality, navigation features, and more. Selenium WebDriver was used for UI interactions, while JUnit was employed for organizing and running the tests.

Website: GitHub (www.github.com)

1.2. Project Functionalities and Screenshots

Main Features of GitHub tested include:

- User Authentication: Login, registration, and session handling.
- Repository Management: Creation, deletion, and updates.
- Search and Filters: Searching for users, repositories, and topics, with sorting and filtering capabilities.
- Pagination: Verifying functionality and accuracy of paginated content.
- Navigation and Links: Ensuring seamless functionality for sidebar, header, and footer links.
- User Profile Customization: Updating profiles and visibility settings for repositories.
- HTTPS Enforcement: Verifying secure connections across the application.
- Accessibility and Performance Testing: Ensuring the platform meets accessibility standards and performs reliably under load.
- Security Testing: Verifying secure data handling during authentication, session management, and repository access.

Screenshots:

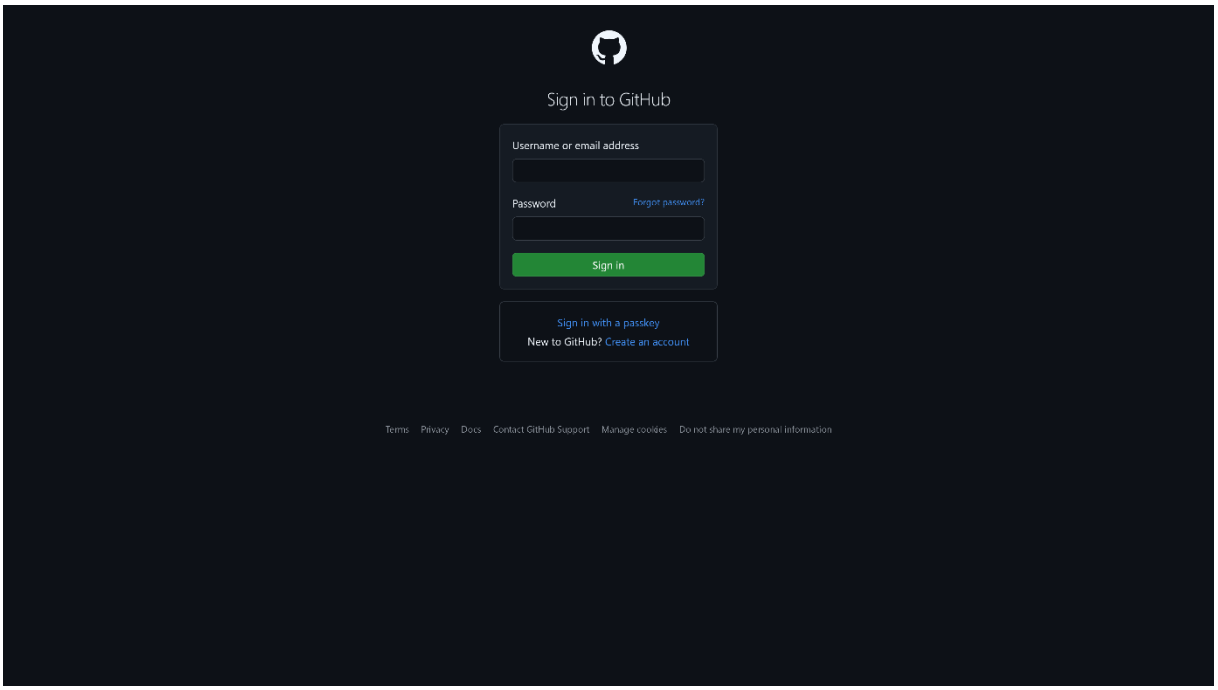

Figure 1 Login page

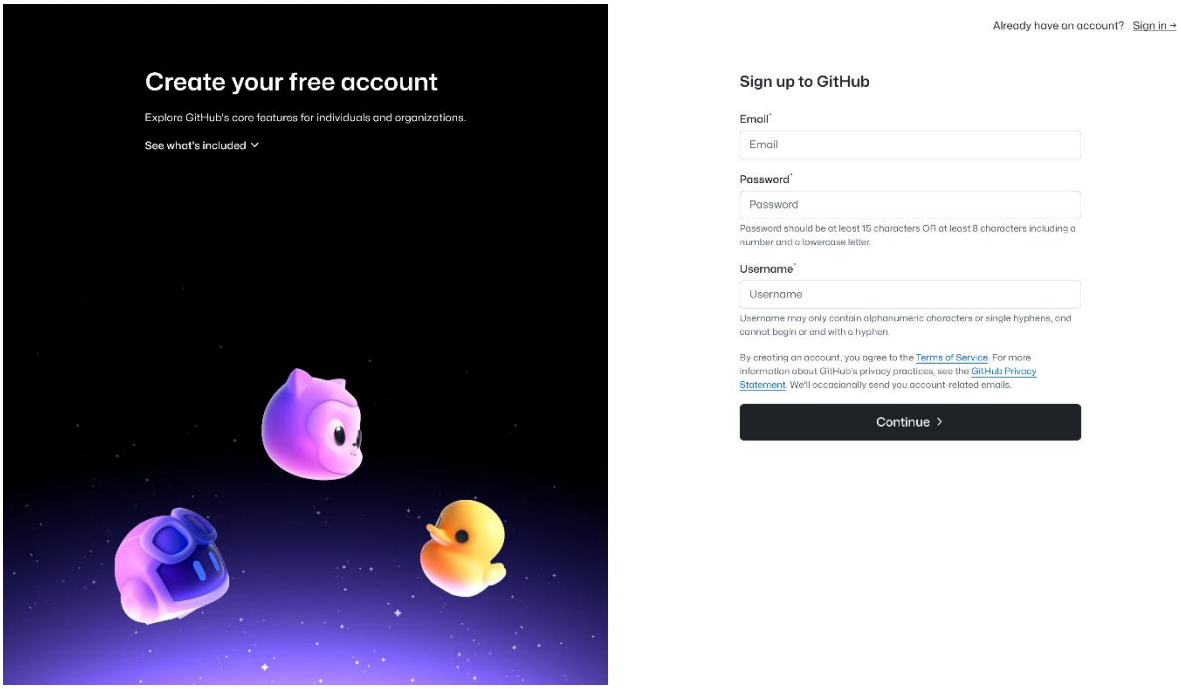

Figure 2 Signup Page

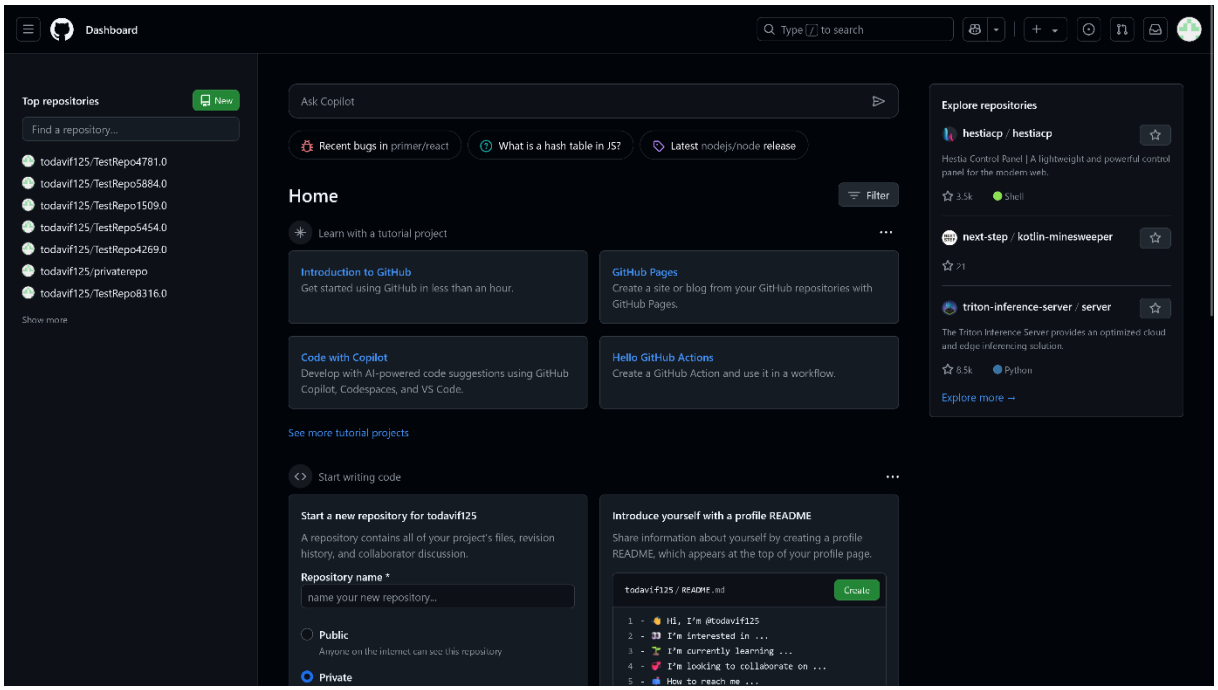

Figure 3 Dashboard Page

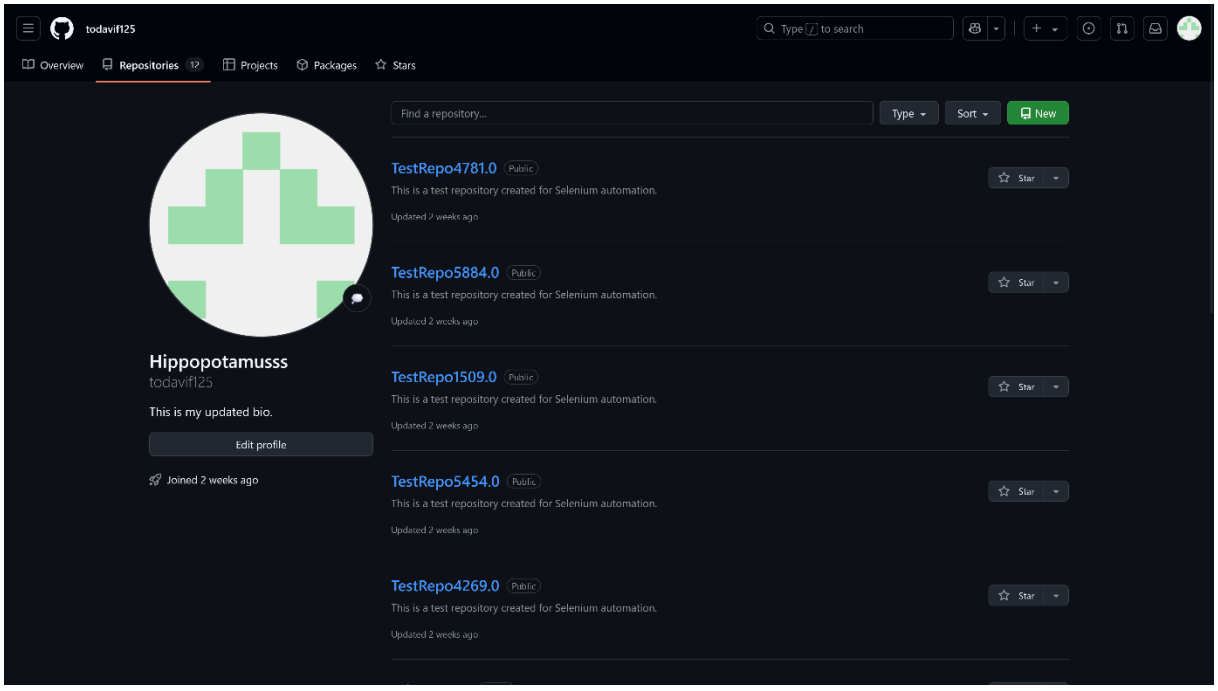

Figure 4 Repositories Page

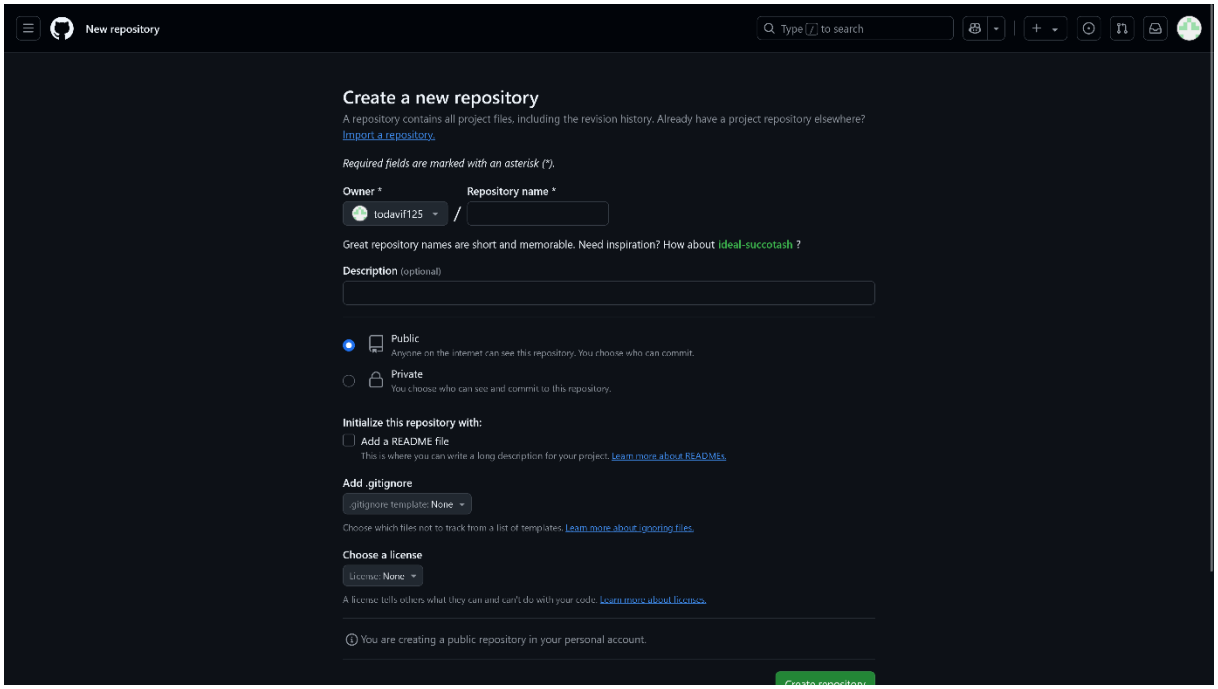

Figure 5 New Repository Page

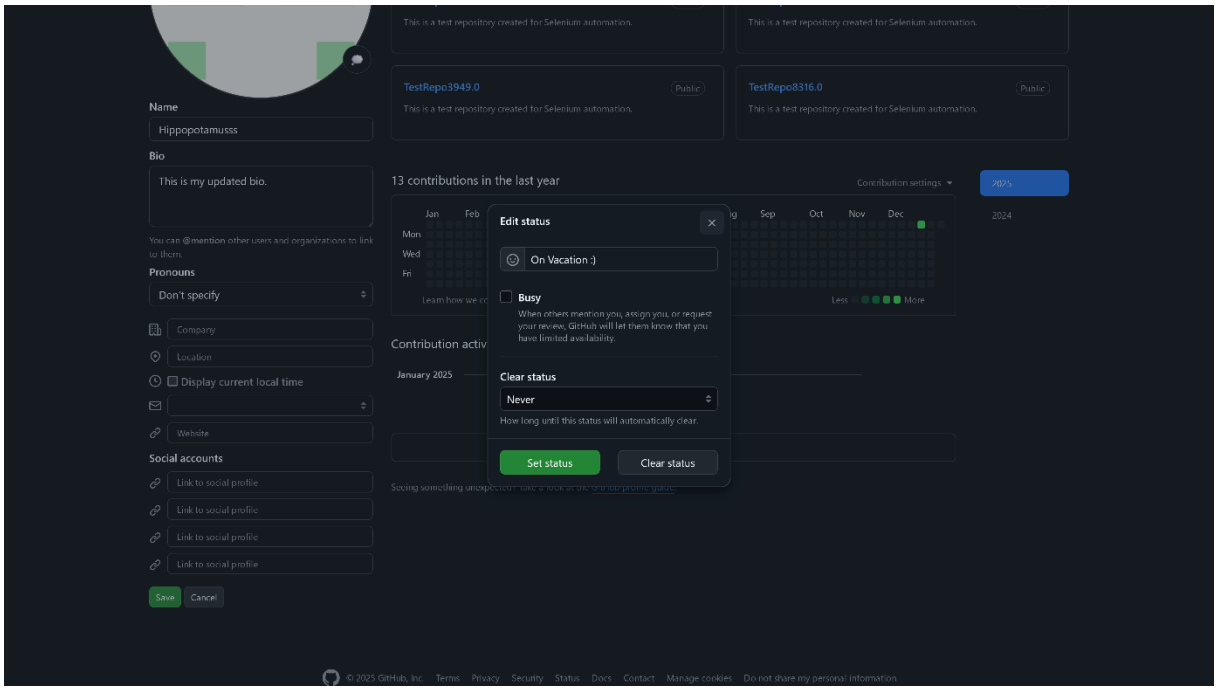

Figure 6 Profile Edit Page

Figure 7 Signup Page

2. Test Plan

2.1. Scope

This testing project focuses on the **core functionalities** of the GitHub web application, ensuring its reliability, security, and usability for end users. Specific areas of focus include:

- User Authentication: Comprehensive testing of login, registration, logout workflows, and secure session handling.
- Search and Filters: Validating the accuracy and functionality of search results for users, repositories, and topics, including sorting and filtering capabilities.
- Repository Management: Testing CRUD (Create, Read, Update, Delete) operations for repositories and error handling for invalid actions.
- Navigation and Links: Ensuring seamless navigation through sidebar, header, and footer links, and verifying their accessibility.
- Account Settings: Evaluating user profile customization, password updates, and repository visibility management.
- Pagination Functionality: Verifying usability and accuracy of paginated content in search results and repository views.
- HTTPS Enforcement: Ensuring all user interactions occur over secure HTTPS connections.
- Accessibility Testing: Testing compliance with WCAG guidelines to ensure the platform is accessible to all users, including those with disabilities.
- Performance Testing: Assessing the responsiveness and reliability of the platform under different load conditions.
- Security Testing: Ensuring secure handling of user data, session management, and protection against vulnerabilities such as XSS, CSRF, and unauthorized access.

Exclusions:

Certain GitHub features were **excluded** from the scope due to their advanced nature or minimal impact on core workflows:

- **Gists**: As they represent a niche use case compared to repositories.
- **Organizations**: Collaboration-specific features beyond personal account use cases.
- **GitHub Actions**: Workflow automation and CI/CD pipelines.
- **Advanced API Testing**: Focus remains on UI functionalities, not backend integrations.

2.2. Testing Environment and Tools

Environment:

- Operating System: Windows 11 Pro / Version 10.0.22631 Build 22631
- Browser: Chrome (Version 117+)

Tools and Frameworks:

- Programming Language: Java.
- Code Editor: IntelliJ IDEA community edition.
- Testing Framework: Selenium Framework with Junit library.
- Browser Driver Management: WebDriverManager (Chrome)

2.3. Project Architecture

This project adopts the **Page Object Model** (POM) design pattern to enhance modularity, reusability, and maintainability of the test suite.

Key Elements of the POM Architecture in This Project:

- Page Classes: Each webpage has a corresponding class that encapsulates:
  - **Locators** (WebElements) for all interactive elements on the page.
  - Methods for interacting with these elements, such as clicking buttons, entering text, or retrieving data.
- Test Classes: Organized by functionality (e.g., LoginTests, SearchTests), each class focuses on validating a single test scenario.
- Utility Classes: Provide shared resources, including configuration files.
- **Base Class**: Manages WebDriver initialization, setup, and teardown processes. It also includes shared methods for test execution, such as the Dev Tools from selenium.

3. Test Execution

3.1. Login Functionality

Testing various aspects of the login functionality, including valid/invalid login attempts, UI behaviors, and redirection features to ensure the system handles user authentication accurately and securely.

Test Name: Test Valid Login				
Description: Validate that a user can successfully log in with valid credentials.				
Pre-condition(s): Test user account exists.				
Test Steps: 1. Navigate to the login page. 2. Enter valid email and password. 3. Click the "Sign In" button. 4. Verify the presence of "Dashboard" on the page.	**Test Data:** Email: todavif125@evusd.com Password: (E17xdbE|D*%=z=N	**Expected Result:** User successfully logs in and sees the dashboard.	**Actual Result:** The dashboard appeared as expected after login.	**Status:** PASS
Notes: The test account has been made just for testing purposes, it's not leaked or someone's else.				

```java
private void login(String email, String password) {
    LoginPage loginPage = new LoginPage(driver);
    loginPage.enterEmail(email);
    loginPage.enterPassword(password);
    loginPage.clickSignIn();
}
```

```java
@Test
public void testValidLogin() {
    driver.get(LOGIN_URL);
    login(VALID_EMAIL, VALID_PASSWORD);
    assertTrue(driver.getPageSource().contains("Dashboard"), "Login failed!");
}
```

Test Name: Test Invalid Login				
Description: Validate that the system shows an error message for incorrect login credentials.				
Pre-condition(s):				
Test Steps: 1. Navigate to the login page. 2. Enter an invalid email and password. 3. Click the "Sign In" button. 4. Verify error message appears on the page.	**Test Data:** Email: invalid_user@gmail.com Password: wrong_password	**Expected Result:** Error message "Incorrect username or password." is displayed.	**Actual Result:** Error message displayed as expected.	**Status:** PASS
Notes:				

```java
@Test
public void testInvalidLogin() {
    driver.get(LOGIN_URL);
    login(INVALID_EMAIL, INVALID_PASSWORD);
    assertTrue(driver.getPageSource().contains("Incorrect username or password."),
            "Expected error message 'Incorrect username or password.' was not found on the page.");
}
```

Test Name: Test Password Field Masking				
Description: Verify that the password field input is properly masked (type = "password").				
Pre-condition(s):				
Test Steps: 1. Navigate to the login page. 2. Inspect the password field element. 3. Verify that the input type is "password."	**Test Data:** None	**Expected Result:** Password field input is masked correctly.	**Actual Result:** Password field masking verified as correct.	**Status:** PASS
Notes:				

```java
@Test
public void testPasswordFieldMasking() {
    driver.get(LOGIN_URL);
    LoginPage loginPage = new LoginPage(driver);
    WebElement passwordField = driver.findElement(loginPage.getPasswordField());
    String inputType = passwordField.getDomAttribute("type");
    assertEquals("password", inputType, "Password field should mask the input.");
}

```

Test Name: Test Forgot Password Redirect				
Description: Validate redirection to the "Forgot Password" page.				
Pre-condition(s):				
Test Steps: 1. Navigate to the login page. 2. Click the "Forgot Password" link. 3. Verify that the redirection URL matches the expected "Forgot Password" URL.	**Test Data:** None	**Expected Result:** User is redirected to the correct "Forgot Password" page.	**Actual Result:** Redirection occurred as expected.	**Status:** PASS
Notes:				

```java
@Test
public void testForgotPasswordRedirection() {
    driver.get(LOGIN_URL);
    LoginPage loginPage = new LoginPage(driver);
    loginPage.clickForgotPasswordLink();
    String currentUrl = driver.getCurrentUrl();
    assertEquals(FORGOT_PASSWORD_URL, currentUrl, "The Forgot Password redirection failed.");
}
```

Test Name: Test Create Account Redirect				
Description: Validate redirection to the "Create Account" page.				
Pre-condition(s):				
Test Steps: 1. Navigate to the login page. 2. Click the "Create Account" link. 3. Verify the redirection URL matches the expected "Create Account" URL.	**Test Data:** None	**Expected Result:** User is redirected to the correct "Create Account" page.	**Actual Result:** Redirection occurred as expected.	**Status:** PASS
Notes:				

```java
@Test
public void testCreateAccountRedirection() {
    driver.get(LOGIN_URL);
    LoginPage loginPage = new LoginPage(driver);
    loginPage.clickCreateAccountLink();

    WebDriverWait waitForRedirect = new WebDriverWait(driver, Duration.ofSeconds(10));
    waitForRedirect.until(ExpectedConditions.urlContains(CREATE_ACCOUNT_URL));

    String currentUrl = driver.getCurrentUrl();

    assertEquals(CREATE_ACCOUNT_URL, currentUrl, "The Create Account redirection failed.");
}
```

Test Name: Test Email Not Registered				
Description: Validate the system handles login attempts with a non-registered email.				
Pre-condition(s):				
Test Steps: 1. Navigate to the login page. 2. Enter a non-registered email and valid password. 3. Click the "Sign In" button. 4. Verify the error message. "Dashboard" on the page.	**Test Data:** Email: nonexistent@domainf.fdcom Password: sldkalsdad	**Expected Result:** Error message "Incorrect username or password." is displayed.	**Actual Result:** Error message displayed as expected.	**Status:** PASS
Notes:				

```java
@Test
public void testEmailNotRegistered() {
    driver.get(LOGIN_URL);
    login("nonexistent@domainf.fdcom", VALID_PASSWORD);
    assertTrue(driver.getPageSource().contains("Incorrect username or password."), "Error message not shown for unregistered email.");
}
```

Test Name: Test Logout				
Description: Validate that a user is logged out successfully and redirected to the login page.				
Pre-condition(s): User must be logged in successfully.				
Test Steps: 1. Log in with valid credentials. 2. Click on the profile button. 3. Select "Logout" from the dropdown. 4. Verify redirection to the login page and absence of dashboard content.	**Test Data:** Email: todavif125@evusd.com Password: (E17xdbE|D*%=z=N	**Expected Result:** User is logged out successfully and redirected to the login page.	**Actual Result:** User is logged out successfully, and the dashboard was no longer accessible.	**Status:** PASS
Notes:				

```java
@Test
public void testLogout() {
    driver.get(LOGIN_URL);
    login(VALID_EMAIL, VALID_PASSWORD);

    WebDriverWait wait = new WebDriverWait(driver, Duration.ofSeconds(10));

    WebElement profileButton = wait.until(ExpectedConditions.elementToBeClickable(By.xpath("//button[@aria-label='Open user navigation menu']")));
    profileButton.click();

    WebElement logoutButton = wait.until(ExpectedConditions.elementToBeClickable(By.xpath("//a[@href='/logout']")));
    logoutButton.click();

    WebElement signOutFromAccount = wait.until(ExpectedConditions.elementToBeClickable(By.name("commit")));
    signOutFromAccount.click();

    assertFalse(driver.getPageSource().contains("Dashboard"), "Logout failed. User not redirected to login page");
}
```

3.2. User Registration Functionality

Testing the registration functionality to ensure users can create an account with valid data, while handling errors for invalid input, weak passwords, and unavailable usernames.

Test Name: Test Valid Registration				
Description: Validate that a user can complete the registration form with valid data.				
Pre-condition(s): CAPTCHA must be manually solved.				
Test Steps: 1. Navigate to the registration page. 2. Enter valid email, username, and password. 3. Click "Continue." 4. Verify presence of CAPTCHA page and Terms of Service.	**Test Data:** **Email**: Unique & valid **Username**: Unique & valid **Password**: Valid	**Expected Result:** User can proceed to CAPTCHA after entering valid details.	**Actual Result:** CAPTCHA page appeared as expected.	**Status:** PASS
Notes: CAPTCHA prevents automated testing.				

```java
private void register(String email, String username, String password) {
    RegisterPage registerPage = new RegisterPage(driver);
    registerPage.enterEmail(email);
    registerPage.enterUsername(username);
    registerPage.enterPassword(password);
    registerPage.clickContinue();
}

@Test
public void testValidRegistration() {
    driver.get(REGISTER_URL);
    register(VALID_EMAIL, VALID_USERNAME, VALID_PASSWORD);
    RegisterPage registerPage = new RegisterPage(driver);
    WebElement captcha = driver.findElement(registerPage.getCaptchaPage());
    // CAPTCHA will prevent the form from being submitted, so we stop before it.
    assertNotNull(captcha, "Terms of Service text not displayed.");
}
```

Test Name: Test Invalid Email Format				
Description: Validate that the system shows an error for an incorrectly formatted email.				
Pre-condition(s):				
Test Steps: 1. Navigate to the registration page. 2. Enter an invalid email format. 3. Verify the appearance of an email error message.	**Test Data:** **Email**: invalidformat	**Expected Result:** Error message appears for invalid email format.	**Actual Result:** Email error message displayed correctly.	**Status:** PASS
Notes:				

```java
@Test
public void testInvalidEmail() {
    driver.get(REGISTER_URL);
    RegisterPage registerPage = new RegisterPage(driver);
    registerPage.enterEmail(INVALID_EMAIL_FORMAT);

    WebDriverWait wait = new WebDriverWait(driver, Duration.ofSeconds(10));
    WebElement emailError = wait.until(ExpectedConditions.visibilityOfElementLocated(registerPage.getEmailError()));

    assertNotNull(emailError, "Email error not displayed.");
}
```

Test Name: Test Weak Password				
Description: Validate that the system shows an error for a password that does not meet strength criteria.				
Pre-condition(s):				
Test Steps: 1. Navigate to the registration page. 2. Enter a weak password. 3. Verify the appearance of a password strength error message.	**Test Data:** **Password**: 123	**Expected Result:** Error message appears for weak password.	**Actual Result:** Password error message displayed correctly.	**Status:** PASS
Notes:				

```java
@Test
public void testWeakPassword() {
    driver.get(REGISTER_URL);
    RegisterPage registerPage = new RegisterPage(driver);
    registerPage.enterPassword(WEAK_PASSWORD);

    WebDriverWait wait = new WebDriverWait(driver, Duration.ofSeconds(10));
    WebElement passwordError = wait.until(ExpectedConditions.visibilityOfElementLocated(registerPage.getPasswordError()));

    assertNotNull(passwordError, "Error message for weak password not shown.");
}
```

Test Name: Test Username Not Available				
Description: Validate that the system shows an error for an already taken username.				
Pre-condition(s): Username is already registered.				
Test Steps: 1. Navigate to the registration page. 2. Enter an already taken username. 3. Verify the appearance of a username availability error message.	**Test Data:** **Username**: hippo	**Expected Result:** Error message appears for unavailable username.	**Actual Result:** Username error message displayed correctly.	**Status:** PASS
Notes:				

```java
@Test
public void testUsernameNotAvailable() {
    driver.get(REGISTER_URL);
    RegisterPage registerPage = new RegisterPage(driver);
    registerPage.enterUsername(INVALID_USERNAME);

    WebDriverWait wait = new WebDriverWait(driver, Duration.ofSeconds(10));
    WebElement usernameError = wait.until(ExpectedConditions.visibilityOfElementLocated(registerPage.getUsernameError()));

    assertNotNull(usernameError, "Error message for unavailable username not shown.");
}
```

3.3. Repository Management Functionality

Testing the repository management features include creating a new repository, ensuring private repositories are inaccessible to unauthorized users, and deleting a repository.

Test Name: Test Create New Repository				
Description: Validate that a user can create a new repository successfully.				
Pre-condition(s): User must be logged in.				
Test Steps: 1. Login to the application. 2. Navigate to the "New Repository" page. 3. Enter a unique repository name and description. 4. Click "Create Repository." 5. Verify URL contains repo name.	**Test Data:** **Repo Name:** Randomly generated **Description:** "Test repository for Selenium."	**Expected Result:** A new repository is created, and the user is redirected to the repository page.	**Actual Result:** Repository created successfully, and URL contains the repository name.	**Status:** PASS
Notes:				

```java
@Test
@Order(1)
public void testCreateNewRepository() {
    loginAndNavigate(LOGIN_URL);

    RepositoryPage repositoryPage = new RepositoryPage(driver);
    repositoryPage.clickNewRepoButton();

    String repoName = generateRepoName();
    repositoryPage.enterRepoName(repoName);
    repositoryPage.enterRepoDescription("This is a test repository created for Selenium automation.");

    waitForElementVisibility(By.id("RepoNameInput-is-available"));
    repositoryPage.clickCreateRepoButton();

    waitForUrlContains(repoName);
    assertTrue(driver.getCurrentUrl().contains(repoName), "Failed to create the repository!");
}
```

Test Name: Test Private Repo Visibility				
Description: Ensure that private repositories are inaccessible to unauthorized users.				
Pre-condition(s): Access a private repository without login.				
Test Steps: 1. Navigate to a private repository URL without login credentials. 2. Verify the "404" or "Not Found" message is displayed.	**Test Data:** None	**Expected Result:** New repository is created, and the user is redirected to the repository page.	**Actual Result:** Unauthorized users cannot access private repositories, and a "404" or equivalent message is shown.	**Status:** PASS
Notes:				

```java
@Test
@Order(3)
public void testPrivateRepoVisibility() {
    driver.get(PRIVATE_REPO_URL);

    boolean isPrivateRepo = new WebDriverWait(driver, Duration.ofSeconds(10))
            .until(driver -> driver.getPageSource().contains("404")
                    || driver.getPageSource().contains("This is not the web page you are looking for"));

    assertTrue(isPrivateRepo, "Private repository should not be visible to unauthorized users.");
}
```

Test Name: Test Delete Repository				
Description: Validate that a user can delete an existing repository and confirm deletion via message.				
Pre-condition(s): User must be logged in and have a repository.				
Test Steps: 1. Login to the application. 2. Navigate to "Your Repositories" page. 3. Click on a repository. 4. Navigate to repository settings. 5. Initiate delete, confirm by typing repo name. 6. Verify deletion success message.	**Test Data:** **- Repo Name:** Name of the repository to delete	**Expected Result:** The repository is deleted successfully, and a confirmation message is shown.	**Actual Result:** Repository deleted successfully, and confirmation message is displayed.	**Status:** PASS
Notes:				

```java
@Test
@Order(2)
public void testDeleteRepository() throws InterruptedException {
    loginAndNavigate(LOGIN_URL);

    driver.get(REPOS_URL);

    WebElement firstRepo = waitForElementToBeClickable(By.cssSelector("#user-repositories-list > ul > li:first-child a"));
    firstRepo.click();

    WebElement settingsTab = waitForElementToBeClickable(By.id("settings-tab"));
    settingsTab.click();

    clickAndWait(By.id("dialog-show-repo-delete-menu-dialog"));
    clickAndWait(By.id("repo-delete-proceed-button"));
    clickAndWait(By.id("repo-delete-proceed-button"));

    WebElement confirmationLabel = waitForElementVisibility(By.xpath("//label[contains(text(), 'To confirm, type')]"));
    String repoName = extractRepoNameFromLabel(confirmationLabel);

    WebElement inputField = waitForElementToBeClickable(By.id("verification_field"));
    inputField.sendKeys(repoName);

    clickAndWait(By.id("repo-delete-proceed-button"));

    WebElement flashMessage = waitForElementVisibility(By.cssSelector(".flash-notice .js-flash-alert"));
    String messageText = flashMessage.getText();

    assertTrue(messageText.contains("was successfully deleted"),
            "Expected successful deletion message, but got: " + messageText);
}
```

3.4. Search and Filters Functionality

Testing the search and filter functionalities, including searching for repositories, filtering by programming language, and sorting repositories by star count.

Test Name: Test Search for Public Repository				
Description: Validate that searching for a public repository returns the correct results.				
Pre-condition(s): None				
Test Steps: 1. Navigate to the search page. 2. Enter the search term "Selenium". 3. Verify that the results contain "Selenium".	**Test Data:** **- Search term:** "Selenium"	**Expected Result:** Repositories related to "Selenium" are displayed in search results.	**Actual Result:** Search results correctly display repositories related to "Selenium".	**Status:** PASS
Notes:				

```java
@Test
@Order(1)
public void testSearchForPublicRepository() {
    driver.get(SEARCH_URL);

    SearchAndFiltersPage searchAndFiltersPage = new SearchAndFiltersPage(driver);
    searchAndFiltersPage.enterSearch("Selenium");

    WebDriverWait wait = new WebDriverWait(driver, Duration.ofSeconds(10));
    wait.until(ExpectedConditions.presenceOfElementLocated(By.cssSelector(".Box-sc-g0xbh4-0 .gZKkEq > div:first-child")));

    WebElement repos = driver.findElement(By.cssSelector(".Box-sc-g0xbh4-0 .gZKkEq > div:first-child"));
    String elementText = repos.getText();
    assertTrue(elementText.contains("Selenium"), "The text does not contain 'Selenium'.");
}
```

Test Name: Test Filter by Programming Language				
Description: Ensure that filtering by programming language works correctly.				
Pre-condition(s): None				
Test Steps: 1. Go to the URL with the Python filter for Selenium repositories. 2. Verify that the results contain "Python".	**Test Data:** **- Search term:** "Selenium" **- Programming language**: "Python"	**Expected Result:** Only repositories related to "Python" are displayed in search results.	**Actual Result:** Results correctly display repositories related to "Selenium" and "Python".	**Status:** PASS
Notes:				

```java
@Test
@Order(2)
public void testFilterByProgrammingLanguage() {
    // here we can automatically search for selenium + python language,
    // because the languages in github filters are anchor tags with these properties.

    driver.get("https://github.com/search?q=selenium+language:Python&type=repositories");

    WebDriverWait wait = new WebDriverWait(driver, Duration.ofSeconds(10));
    wait.until(ExpectedConditions.presenceOfElementLocated(By.cssSelector(".Box-sc-g0xbh4-0 .gZKkEq > div:first-child")));

    WebElement repos = driver.findElement(By.cssSelector(".Box-sc-g0xbh4-0 .gZKkEq > div:first-child"));
    String elementText = repos.getText();
    assertTrue(elementText.contains("Python"), "The text does not contain 'Python'.");
}
```

Test Name: Test Sort by Stars				
Description: Ensure that sorting by stars displays repositories in descending order of star count.				
Pre-condition(s): None				
Test Steps: 1. Go to the sorted URL for Selenium Python repositories. 2. Extract and compare the star counts to ensure they are sorted in descending order.	**Test Data:** **- Search term:** "Selenium" **- Programming language:** "Python" **- Sort by:** "Stars (Descending)"	**Expected Result:** Repositories should be sorted in descending order of star count.	**Actual Result:** The repositories' star counts are correctly sorted in descending order.	**Status:** PASS
Notes: Sorting algorithm validation successful.				

```java
@Test
@Order(3)
public void testSortByStars() {
    // Same like test 2
    driver.get("https://github.com/search?q=selenium+language:Python&type=repositories&s=stars&o=desc");
    WebDriverWait wait = new WebDriverWait(driver, Duration.ofSeconds(10));
    wait.until(ExpectedConditions.presenceOfElementLocated(By.cssSelector(".Box-sc-g0xbh4-0 .gZKkEq > div:first-child")));

    List<WebElement> repos = driver.findElements(By.cssSelector(".Box-sc-g0xbh4-0 .gZKkEq > div"));
    List<Double> starCounts = new ArrayList<>();

    for (WebElement repo : repos) {
        WebElement starsElement = repo.findElement(By.cssSelector("a[aria-label*='stars'] span"));
        String starsCountText = starsElement.getText();

        double starsCount = parseStarsCount(starsCountText);
        starCounts.add(starsCount);
    }

    List<Double> sortedStarCounts = new ArrayList<>(starCounts);
    Collections.sort(sortedStarCounts, Collections.reverseOrder());

    // Assert that the actual star counts are in descending order
    assertTrue(starCounts.equals(sortedStarCounts), "Star counts are not in descending order.");
}
```

3.5. Session Handling

Testing session persistence after page reload and session expiration after logout to ensure proper session management.

Test Name: Test Session Persistence				
Description: Verify that the session persists after a page reload.				
Pre-condition(s): User is logged in successfully.				
Test Steps: 1. Navigate to the login page. 2. Enter valid credentials and sign in. 3. Refresh the page. 4. Check if the "Dashboard" is still displayed, indicating session persistence.	**Test Data:** - Valid email: from config - Valid password: from config	**Expected Result:** User remains logged in after a page refresh and the "Dashboard" is visible.	**Actual Result:** Session persists successfully and the "Dashboard" is visible after refresh.	**Status:** PASS
Notes:				

```java
@Test
public void testSessionPersistence() {
    driver.get(LOGIN_URL);

    LoginPage loginPage = new LoginPage(driver);
    loginPage.enterEmail(VALID_EMAIL);
    loginPage.enterPassword(VALID_PASSWORD);
    loginPage.clickSignIn();

    driver.navigate().refresh();

    assertTrue(driver.getPageSource().contains("Dashboard"), "Session is not persistent after page reload.");
}
```

Test Name: Session Expiration After Logout				
Description: Verify that the session expires, and the user is redirected to the login page after logging out.				
Pre-condition(s): User is logged in successfully.				
Test Steps: 1. Navigate to the login page. 2. Enter valid credentials and sign in. 3. Click on the profile button and select logout. 4. Navigate to a restricted page. 5. Verify redirection to the login page.	**Test Data:** - **Valid email**: from config - **Valid password**: from config	**Expected Result:** User is redirected to the login page after logging out and trying to access a restricted page.	**Actual Result:** After logout, the user is successfully redirected to the login page when trying to access a restricted page.	**Status:** PASS
Notes: Logout and session expiration works as expected.				

```java
@Test
public void testSessionExpirationAfterLogout() {
    driver.get(LOGIN_URL);

    LoginPage loginPage = new LoginPage(driver);
    loginPage.enterEmail(VALID_EMAIL);
    loginPage.enterPassword(VALID_PASSWORD);
    loginPage.clickSignIn();

    WebDriverWait wait = new WebDriverWait(driver, Duration.ofSeconds(10));

    WebElement profileButton = wait.until(ExpectedConditions.elementToBeClickable(By.xpath("//button[@aria-label='Open user navigation menu']")));
    profileButton.click();

    WebElement logoutButton = wait.until(ExpectedConditions.elementToBeClickable(By.xpath("//a[@href='/logout']")));
    logoutButton.click();

    WebElement signOutFromAccount = wait.until(ExpectedConditions.elementToBeClickable(By.name("commit")));
    signOutFromAccount.click();

    driver.get("https://github.com/dashboard");

    assertTrue(driver.getCurrentUrl().contains("https://github.com/login"), "User should be redirected to login page after logout");
}
```

3.6. HTTPS Enforcement

Testing that the application forces HTTPS redirection, ensures secure pages are served over HTTPS, and that the SSL certificate is valid.

Test Name: Test Redirect to HTTPS				
Description: Verify that the application redirects from HTTP to HTTPS.				
Pre-condition(s):				
Test Steps: 1. Navigate to "http://github.com". 2. Check if the URL starts with "https://".	**Test Data:** None	**Expected Result:** Application should be redirected to "https://".	**Actual Result:** URL redirects to "https://" successfully.	**Status:** PASS
Notes:				

```java
@Test
public void testRedirectToHttps() {
    driver.get("http://github.com");
    String currentUrl = driver.getCurrentUrl();
    assertTrue(currentUrl.startsWith("https://"), "Application does not redirect to HTTPS");
}
```

Test Name: Test Secure Page Served Over HTTPS				
Description: Verify that the login page is served securely over HTTPS.				
Pre-condition(s):				
Test Steps: 1. Navigate to the login page (login URL). 2. Check if the URL starts with "https://".	**Test Data:** None	**Expected Result:** Login page should be served over HTTPS.	**Actual Result:** The login page is served over HTTPS as expected.	**Status:** PASS
Notes:				

```java
@Test
public void testSecurePageHttps() {
    driver.get(LOGIN_URL);
    String currentUrl = driver.getCurrentUrl();
    assertTrue(currentUrl.startsWith("https://"), "Login page is not served over HTTPS");
}
```

Test Name: Test Valid SSL Certificate				
Description: Verify that the SSL certificate is valid for the domain.				
Pre-condition(s):				
Test Steps: 1. Navigate to "https://www.ssllabs.com/ssltest/analyze.html?d=github.com". 2. Check for SSL certification results on the page.	**Test Data:** None	**Expected Result:** The SSL certificate should be valid, and there should be evidence of "TLS", "HSTS", and "CAA" on the page.	**Actual Result:** SSL certificate is valid, with evidence of "TLS", "HSTS", and "CAA".	**Status:** PASS
Notes: Sometimes, this site might be down, so make sure to try again later.				

```java
@Test
public void testValidSslCertificate() {
    driver.get("https://www.ssllabs.com/ssltest/analyze.html?d=github.com");
    WebDriverWait wait = new WebDriverWait(driver, Duration.ofSeconds(10));
    boolean isValid = wait.until(
            driver -> driver.getPageSource().contains("This server supports TLS")
                    || driver.getPageSource().contains("HTTP Strict Transport Security (HSTS) with long duration deployed on this server")
                    || driver.getPageSource().contains("DNS Certification Authority Authorization (CAA) Policy found for this domain.")
    );

    assertTrue(isValid, "SSL certificate is not valid.");
}
```

3.7. Accessibility Testing

Ensuring that the login page complies with accessibility standards, including descriptive link texts, proper form labels, and functional keyboard navigation.

Test Name: Test Links Have Descriptive Text				
Description: Verify that all links on the page have descriptive text.				
Pre-condition(s):				
Test Steps: 1. Navigate to the login page. 2. Check if all links have non-empty descriptive text. 3. Print a message if any link is missing text.	**Test Data:** None	**Expected Result:** All links should have non-empty descriptive text.	**Actual Result:** Some links did not have descriptive text.	**Status:** FAIL
Notes: Links such as the homepage logo did not have descriptive text. This will need to be fixed for accessibility compliance.				

```java
@Test
public void testLinksHaveDescriptiveText() {
    driver.get(LOGIN_URL);

    WebDriverWait wait = new WebDriverWait(driver, Duration.ofSeconds(10));
    wait.until(ExpectedConditions.presenceOfElementLocated(By.tagName("a")));

    for (WebElement link : driver.findElements(By.tagName("a"))) {
        String linkText = link.getText();

        if (linkText == null || linkText.trim().isEmpty()) {
            System.out.println("Link without descriptive text: " + link.getAttribute("outerHTML"));
        }

        assertTrue(linkText != null && !linkText.trim().isEmpty(), "Link don't have descriptive text");
    }
}
```

Test Name: Test Form Labels on Login Page				
Description: Ensure that the login form fields have proper labels for accessibility.				
Pre-condition(s):				
Test Steps: 1. Navigate to the login page. 2. Check if the form fields have proper labels.	**Test Data:** None	**Expected Result:** All form fields should have descriptive labels.	**Actual Result:** All form fields had descriptive labels.	**Status:** PASS
Notes:				

```java
@Test
public void testFormLabelsOnLoginPage() {
    driver.get(LOGIN_URL);

    WebDriverWait wait = new WebDriverWait(driver, Duration.ofSeconds(10));
    wait.until(ExpectedConditions.presenceOfElementLocated(By.id("login_field")));

    WebElement usernameLabel = driver.findElement(By.xpath("//label[@for='login_field']"));
    WebElement passwordLabel = driver.findElement(By.xpath("//label[@for='password']"));

    assertTrue(usernameLabel != null && passwordLabel != null, "Form fields don't have proper labels");
}
```

Test Name: Test Keyboard Navigation on Login Page				
Description: Ensure keyboard navigation works properly on the login page.				
Pre-condition(s):				
Test Steps: 1. Navigate to the login page. 2. Focus on the username field using the keyboard. 3. Use the "Tab" key to navigate to the password field.	**Test Data:** None	**Expected Result:** The focus should move correctly between the fields.	**Actual Result:** Keyboard navigation worked correctly, and focus moved from the username field to the password field.	**Status:** PASS
Notes:				

```java
@Test
public void testKeyboardNavigationOnLoginPage() {
    driver.get(LOGIN_URL);

    WebDriverWait wait = new WebDriverWait(driver, Duration.ofSeconds(10));
    wait.until(ExpectedConditions.presenceOfElementLocated(By.id("login_field")));

    WebElement usernameField = driver.findElement(By.id("login_field"));
    usernameField.click();
    usernameField.sendKeys("testUser");

    driver.findElement(By.id("password")).click();

    WebElement passwordField = driver.findElement(By.id("password"));
    assertTrue(passwordField.isDisplayed(), "Focus should be on the password field after tabbing");
}
```

3.8. Performance Testing

Performance testing ensures that the application meets performance benchmarks such as fast page load times and quick API response times.

Test Name: Test Page Load Time				
Description: Ensure that the page loads within an acceptable time frame (under 3 seconds).				
Pre-condition(s):				
Test Steps: 1. Navigate to the base URL of the application. 2. Measure the page load time using the performance.timing API. 3. Verify that the load time is under 3000 milliseconds (3 seconds).	**Test Data:** None	**Expected Result:** The page should be loaded within 3 seconds.	**Actual Result:** The page loaded in less than 3 seconds.	**Status:** PASS
Notes: If the load time exceeds 3000 milliseconds, this indicates a performance issue.				

```java
@Test
public void testPageLoadTime() {
    driver.get(BASE_URL);

    JavascriptExecutor js = (JavascriptExecutor) driver;
    long loadTime = (Long) js.executeScript(
            "return performance.timing.loadEventEnd - performance.timing.navigationStart;");
    assertTrue(loadTime < 3000, "Page didn't load within 3 seconds");
    System.out.println("Page load time test passed, It is: " + loadTime + "ms");
}
```

Test Name: Test API Response Time				
Description: Ensure that API calls complete within an acceptable time frame (under 2 seconds).				
Pre-condition(s):				
Test Steps: 1. Navigate to the base URL of the application. 2. Measure the API response time using the performance.getEntriesByType('resource') API. 3. Verify that the total API response time is under 2000 milliseconds (2 seconds).	**Test Data:** None	**Expected Result:** The API response time should be under 2 seconds.	**Actual Result:** The API response time was under 3 seconds.	**Status:** PASS
Notes:				

```java
@Test
public void testApiResponseTime() {
    driver.get(BASE_URL);
    JavascriptExecutor js = (JavascriptExecutor) driver;

    double apiResponseTime = (double) js.executeScript(
            "let entries = performance.getEntriesByType('resource');" +
                    "let apiCalls = entries.filter(e => e.initiatorType === 'fetch');" +
                    "return apiCalls.reduce((sum, e) => sum + e.duration, 0);"
    );
    System.out.println("API response time: " + apiResponseTime + "ms");
    assertTrue(apiResponseTime < 2000, "API calls should complete within 2 seconds");
}
```

3.9. Security Testing

The Security Tests ensure that the application implements proper security measures, including secure HTTP headers, protection against Cross-Site Request Forgery (CSRF) attacks, and prevention of SQL injection vulnerabilities.

Test Name: Test Secure Headers				
Description: Ensure that the application sends appropriate security headers, such as Content-Security-Policy and Strict-Transport-Security.				
Pre-condition(s):				
Test Steps: 1. Enable network interception using DevTools. 2. Capture the response headers when the login page is loaded. 3. Verify that the response includes Content-Security-Policy and Strict-Transport-Security headers.	**Test Data:** None	**Expected Result:** The response should include both Content-Security-Policy and Strict-Transport-Security headers.	**Actual Result:** The response included both Content-Security-Policy and Strict-Transport-Security headers.	**Status:** PASS
Notes:				

```java
@Test
public void testSecureHeaders() {
    devTools = ((ChromeDriver) driver).getDevTools();
    devTools.createSession();
    devTools.send(Network.enable(Optional.empty(), Optional.empty(), Optional.empty()));

    devTools.addListener(Network.responseReceived(), response -> {
        var headers = response.getResponse().getHeaders();

        assertTrue(headers.containsKey("content-security-policy"),
                "Content-Security-Policy header should be present");

        assertTrue(headers.containsKey("strict-transport-security"),
                "Strict-Transport-Security header should be present");
    });

    driver.get(LOGIN_URL);
}
```

Test Name: Test CSRF Token Presence				
Description: Ensure that a CSRF token is present in the login form to protect against cross-site request forgery attacks.				
Pre-condition(s):				
Test Steps: 1. Navigate to the login page. 2. Locate the CSRF token in the login form by inspecting the authenticity_token element. 3. Verify that the CSRF token is not null.	**Test Data:** None	**Expected Result:** The CSRF token should be present in the login form.	**Actual Result:** The CSRF token is present in the login form.	**Status:** PASS
Notes: A missing CSRF token could allow CSRF attacks.				

```java
@Test
public void testCsrfTokenPresence() {
    driver.get(LOGIN_URL);

    WebElement csrfTokenElement = driver.findElement(By.name("authenticity_token"));
    String csrfToken = csrfTokenElement.getAttribute("value");

    assertNotNull(csrfToken, "CSRF token should be present in the login form");
}
```

Test Name: Test SQL Injection Handling				
Description: Ensure the application properly handles SQL injection attempts and prevents unauthorized access.				
Pre-condition(s):				
Test Steps: 1. Navigate to the login page. 2. Input a SQL injection payload (' OR 1=1 --) in the login field. 3. Enter an invalid password and click login. 4. Verify that an error message is displayed indicating the login attempt failed.	**Test Data:** SQL Injection Payload: ' OR 1=1 --	**Expected Result:** An error message should be displayed indicating that the login attempt failed.	**Actual Result:** Error message displayed: Validation error	**Status:** PASS
Notes:				

```java
@Test
public void testSqlInjectionHandling() {
    driver.get(LOGIN_URL);

    driver.findElement(By.name("login")).sendKeys("' OR 1=1 --");
    driver.findElement(By.name("password")).sendKeys("invalidPassword");
    driver.findElement(By.name("commit")).click();

    WebElement errorMessage = driver.findElement(By.cssSelector("div.flash-error"));
    assertTrue(errorMessage.isDisplayed(), "Error message should be displayed for SQL injection attempts");
}
```

3.10. User Profile Customization Tests

The User Profile Customization Tests verify that users can successfully update their profile information and status. These tests ensure the application allows users to edit and save changes to their personal details and user status in their profile.

Test Name: Test Update Profile Information				
Description: Verify that the user can update their name and bio in their profile and the changes are saved successfully.				
Pre-condition(s): User is logged in with valid credentials.				
Test Steps: 1. Navigate to the login page. 2. Enter valid email and password, click "Sign In". 3. Navigate to the profile page. 4. Wait for the "Edit profile" button to appear. 5. Click on "Edit profile" and update the name and bio. 6. Save the changes and verify that the updated name and bio are displayed.	**Test Data:** - Valid email: from config - Valid password: from config - Updated name: "Hippopotamus" - Updated bio: "This is my updated bio."	**Expected Result:** - User is signed in and the profile page loads successfully. - "Edit profile" option is available. - Updated name and bio are saved and displayed correctly on the profile page.	**Actual Result:** Name and bio were updated successfully and displayed correctly.	**Status:** PASS
Notes:				

```java
@Test
public void testUpdateProfileInformation() {
    loginAndNavigateToProfile();

    ProfilePage profilePage = new ProfilePage(driver);
    waitForPageContains("Edit profile");

    profilePage.editProfile();

    String updatedName = "Hippopotamus";
    String updatedBio = "This is my new bio.";

    profilePage.updateName(updatedName);
    profilePage.updateBio(updatedBio);
    profilePage.saveChanges();

    waitForPageContains("Edit profile");

    assertTrue(isTextPresentOnPage(updatedName), "Name is not present.");
    assertTrue(isTextPresentOnPage(updatedBio), "Bio is not present.");
}
```

Test Name: Test Update User Status				
Description: Verify that the user can set and save a custom status in their profile.				
Pre-condition(s): User is logged in with valid credentials.				
Test Steps: 1. Navigate to the login page. 2. Enter valid email and password, click "Sign In". 3. Navigate to the profile page. 4. Wait for the "Edit profile" button to appear. 5. Set a custom status (e.g., "On Vacation :)") and save. 6. Verify that the custom status is visible and saved.message is displayed indicating the login attempt failed.	**Test Data:** - Valid email: from config - Valid password: from config - Status: "On Vacation :)"	**Expected Result:** - User is signed in and the profile page loads successfully. - "Edit profile" option is available. - Custom status is saved and displayed correctly on the profile page.	**Actual Result:** Status was saved successfully and displayed correctly.	**Status:** PASS
Notes:				

```java
@Test
public void testUpdateUserStatus() {
    loginAndNavigateToProfile();

    ProfilePage profilePage = new ProfilePage(driver);
    waitForPageContains("Edit profile");

    String status = "On Vacation :)";
    profilePage.setStatus(status);
    profilePage.saveStatus();

    profilePage.seeStatus();

    assertTrue(isTextPresentOnPage(status), "Status didn't save.");
}
```

3.11. User Profile Customization Tests

The Repository Error Handling Tests ensure that proper error handling mechanisms are in place when users attempt to create repositories with duplicate names or when they fail to provide required information in the form.

Test Name: Test Repository Name Duplication				
Description: Verify that an error message is displayed when attempting to create a repository with a name that already exists.				
Pre-condition(s): User is logged in with valid credentials.				
Test Steps: 1. Navigate to the login page. 2. Enter valid email and password, click "Sign In". 3. Navigate to the repository creation page. 4. Enter a repository name that already exists. 5. Click the "Create Repository" button. 6. Verify that an error message for the duplicate repository name is displayed.	**Test Data:** - Valid email: from config - Valid password: from config - Repository name: "super-octo-adventure" - Repository description: "This is a test repository."	**Expected Result:** - User is signed in and the repository creation page loads. - An error message indicating that the repository name already exists is displayed.	**Actual Result:** Error message for duplicate repository name was displayed successfully.	**Status:** PASS
Notes:				

```java
@Test
public void testRepositoryNameDuplication() {
    driver.get(LOGIN_URL);
    LoginPage loginPage = new LoginPage(driver);
    loginPage.enterEmail(VALID_EMAIL);
    loginPage.enterPassword(VALID_PASSWORD);
    loginPage.clickSignIn();

    RepositoryPage repositoryPage = new RepositoryPage(driver);

    repositoryPage.clickNewRepoButton();
    repositoryPage.enterRepoName(REPO_NAME);
    repositoryPage.enterRepoDescription(REPO_DESCRIPTION);
    repositoryPage.clickCreateRepoButton();

    WebDriverWait wait = new WebDriverWait(driver, Duration.ofSeconds(10));
    wait.until(ExpectedConditions.visibilityOfElementLocated(By.id("RepoNameInput-message")));

    boolean errorMessageExists = driver.getPageSource().contains("already exists on this account");
    assertTrue(errorMessageExists, "Error message for duplicate repository name is not displayed.");
}
```

Test Name: Test Form Validation Errors				
Description: Verify that an error message is displayed when attempting to create a repository without providing a repository name.				
Pre-condition(s): User is logged in with valid credentials.				
Test Steps: 1. Navigate to the login page. 2. Enter valid email and password, click "Sign In". 3. Navigate to the repository creation page. 4. Leave the repository name field empty. 5. Click the "Create Repository" button. 6. Verify that an error message for the empty repository name is displayed.	**Test Data:** - Valid email: from config - Valid password: from config - Repository name: empty - Repository description: "This is a test repository."	**Expected Result:** - User is signed in and the repository creation page loads. - An error message indicating that the repository name must not be blank is displayed.	**Actual Result:** Error message for empty repository name was displayed successfully.	**Status:** PASS
Notes:				

```java
@Test
public void testFormValidationErrors() {
    driver.get(LOGIN_URL);
    LoginPage loginPage = new LoginPage(driver);
    loginPage.enterEmail(VALID_EMAIL);
    loginPage.enterPassword(VALID_PASSWORD);
    loginPage.clickSignIn();

    RepositoryPage repositoryPage = new RepositoryPage(driver);

    repositoryPage.clickNewRepoButton();
    repositoryPage.clickCreateRepoButton();

    WebDriverWait wait = new WebDriverWait(driver, Duration.ofSeconds(10));
    wait.until(ExpectedConditions.visibilityOfElementLocated(By.id("RepoNameInput-message")));

    boolean errorMessageExists = driver.getPageSource().contains("repository name must not be blank");
    assertTrue(errorMessageExists, "Error message for empty repository name is not displayed.");
}
```

3.12. Search Result Pagination Tests

The Search Result Pagination Tests validate the proper functionality of pagination in search results, including navigation between pages and direct page access.

Test Name: Test Navigate to Second Page				
Description: Ensure that the second page of search results loads correctly when navigating to it.				
Pre-condition(s):				
Test Steps: 1. Navigate to the search results page. 2. Navigate to the second page of results by adding &p=2 to the URL. 3. Verify that the URL contains p=2.	**Test Data:** - Search URL: https://github.com/search?q=selenium&type=repositories - Page number: 2	**Expected Result:** The second page of search results should be loaded, and the URL should contain p=2.	**Actual Result:** The second page of search results loaded correctly.	**Status:** PASS
Notes:				

```java
@Test
public void testNavigateToSecondPage() {
    driver.get(BASE_SEARCH_URL);

    driver.get(BASE_SEARCH_URL + "&p=2");

    assert driver.getCurrentUrl().contains("p=2") : "Second page of results not loaded correctly";
}
```

Test Name: Test Navigate Back to First Page				
Description: Ensure that navigation from the second page back to the first page works as expected.				
Pre-condition(s):				
Test Steps: 1. Navigate to the search results page. 2. Navigate to the second page of results by adding &p=2 to the URL. 3. Navigate back to the first page. 4. Verify that the URL matches the base search URL.	**Test Data:** - Search URL: https://github.com/search?q=seleniu m&type=repositor ies - Page number: 2	**Expected Result:** The first page of search results should load, and the URL should return to the base search URL without &p=2.	**Actual Result:** Navigation back to the first page worked as expected.	**Status:** PASS
Notes:				

```java
@Test
public void testNavigateBackToFirstPage() {
    driver.get(BASE_SEARCH_URL);

    driver.get(BASE_SEARCH_URL + "&p=2");

    assert driver.getCurrentUrl().contains("p=2") : "Second page of results not loaded correctly";

    driver.get(BASE_SEARCH_URL);

    assert driver.getCurrentUrl().equals(BASE_SEARCH_URL) : "Failed to return to the first page";
}
```

Test Name: Test Direct Page Navigation				
Description: Verify that navigating directly to a specific page number works as expected.				
Pre-condition(s):				
Test Steps: 1. Navigate to the search results page. 2. Directly navigate to the third page of results by adding &p=3 to the URL. 3. Verify that the URL contains p=3.	**Test Data:** - Search URL: https://github.com/search?q=selenium&type=repositories - Page number: 3	**Expected Result:** The third page of search results should load, and the URL should contain p=3.	**Actual Result:** The third page of search results loaded correctly.	**Status:** PASS
Notes:				

```java
@Test
public void testDirectPageNavigation() {
    driver.get(BASE_SEARCH_URL);
    driver.get(BASE_SEARCH_URL + "&p=3");

    assert driver.getCurrentUrl().contains("p=3") : "Page 3 of results not loaded correctly";
}
```

3.13. Navigation Menu Tests

The Navigation Menu Tests verify that the navigation menu buttons correctly redirect users to the intended pages, ensuring seamless navigation.

Test Name: Test Navigation to Signup				
Description: Verify that clicking the "Sign up" button on the home page navigates the user to the Sign up page.				
Pre-condition(s): User is on the home page.				
Test Steps: 1. Set the browser window size. 2. Navigate to the home page. 3. Click the "Sign up" button. 4. Verify that the URL contains /signup.	**Test Data:** None	**Expected Result:** The user should be redirected to the sign-up page, and the URL should contain /signup.	**Actual Result:** The navigation to the Sign up page worked as expected.	**Status:** PASS
Notes:				

```java
@Test
public void testNavigationToSignup() {
    driver.manage().window().setSize(new Dimension(1920, 1080));
    driver.get(BASE_URL);
    HomePage homePage = new HomePage(driver);
    homePage.clickSignup();
    assertTrue(driver.getCurrentUrl().contains("/signup"), "Failed to navigate to Sign up page");
}
```

Test Name: Test Navigation to Login				
Description: Verify that clicking the "Sign in" button on the home page navigates the user to the Login page.				
Pre-condition(s): User is on the home page.				
Test Steps: 1. Set the browser window size. 2. Navigate to the home page. 3. Click the "Login" button. 4. Verify that the URL contains /login.	**Test Data:** None	**Expected Result:** The user should be redirected to the login page, and the URL should contain /login.	**Actual Result:** The navigation to the Login page worked as expected.	**Status:** PASS
Notes:				

```java
@Test
public void testNavigationToLogin() {
    driver.manage().window().setSize(new Dimension(1920, 1080));
    driver.get(BASE_URL);
    HomePage homePage = new HomePage(driver);

    homePage.clickLogin();

    assertTrue(driver.getCurrentUrl().contains("/login"), "Failed to navigate to Login page");
}
```

Test Name: Test Navigation to Pricing				
Description: Verify that clicking the "Pricing" button on the home page navigates the user to the Pricing page.				
Pre-condition(s): User is on the home page.				
Test Steps: 1. Set the browser window size. 2. Navigate to the home page. 3. Click the "Pricing" button. 4. Verify that the URL contains /pricing.	**Test Data:** None	**Expected Result:** The user should be redirected to the pricing page, and the URL should contain / pricing.	**Actual Result:** The navigation to the pricing page worked as expected.	**Status:** PASS
Notes:				

```java
@Test
public void testNavigationToPricing() {
    driver.manage().window().setSize(new Dimension(1920, 1080));
    driver.get(BASE_URL);
    HomePage homePage = new HomePage(driver);

    homePage.clickPricing();

    assertTrue(driver.getCurrentUrl().contains("/pricing"), "Failed to navigate to Pricing page");
}
```

Test Name: Test Navigation to Copilot				
Description: Verify that clicking the " Copilot " button on the home page navigates the user to the Copilot page.				
Pre-condition(s): User is on the home page.				
Test Steps: 1. Set the browser window size. 2. Navigate to the home page. 3. Click the " Copilot " button. 4. Verify that the URL contains /copilot.	**Test Data:** None	**Expected Result:** The user should be redirected to the Copilot page, and the URL should contain /copilot.	**Actual Result:** The navigation to the Copilot page worked as expected.	**Status:** PASS
Notes:				

```java
@Test
public void testNavigationToCopilot() {
    driver.manage().window().setSize(new Dimension(1920, 1080));
    driver.get(BASE_URL);
    HomePage homePage = new HomePage(driver);

    homePage.clickCopilot();

    assertTrue(driver.getCurrentUrl().contains("/copilot"), "Failed to navigate to Copilot page");
}
```

3.14. Footer Links Tests

The Footer Links Tests verify that the footer navigation links correctly redirect users to the intended pages when clicked, ensuring accessibility to all footer sections.

Test Name: Test Navigation to Subscribe				
Description: Verify that clicking the "Subscribe" link in the footer navigates the user to the newsletter subscription page.				
Pre-condition(s): User is on the home page.				
Test Steps: 1. Set the browser window size. 2. Navigate to the home page. 3. Scroll to the bottom of the page. 4. Click the "Subscribe" link. 5. Verify that the URL contains /newsletter.	**Test Data:** None	**Expected Result:** The user should be redirected to the newsletter subscription page, and the URL should contain /newsletter.	**Actual Result:** Navigation to the Subscribe page worked as expected.	**Status:** PASS
Notes:				

```java
@Test
public void testNavigationToSubscribe() {
    navigateToFooterAndClick(() -> new HomePage(driver).clickSubscribe(), "/newsletter");
}
```

Test Name: Test Navigation to CLI				
Description: Verify that clicking the " CLI" link in the footer navigates the user to the newsletter subscription page.				
Pre-condition(s): User is on the home page.				
Test Steps: 1. Set the browser window size. 2. Navigate to the home page. 3. Scroll to the bottom of the page. 4. Click the " CLI" link. 5. Verify that the URL contains /cli.	**Test Data:** None	**Expected Result:** The user should be redirected to the CLI page, and the URL should contain /cli.	**Actual Result:** Navigation to the CLIpage worked as expected.	**Status:** PASS
Notes:				

```java
@Test
public void testNavigationToCli() {
    navigateToFooterAndClick(() → new HomePage(driver).clickCli(), "cli");
}
```

Test Name: Test Navigation to Desktop				
Description: Verify that clicking the " Desktop" link in the footer navigates the user to the newsletter subscription page.				
Pre-condition(s): User is on the home page.				
Test Steps: 1. Set the browser window size. 2. Navigate to the home page. 3. Scroll to the bottom of the page. 4. Click the " Desktop" link. 5. Verify that the URL contains /desktop.	**Test Data:** None	**Expected Result:** The user should be redirected to the desktop page, and the URL should contain /desktop.	**Actual Result:** Navigation to the Desktop page worked as expected.	**Status:** PASS
Notes:				

```java
@Test
public void testNavigationToDesktop() {
    navigateToFooterAndClick(() -> new HomePage(driver).clickDesktop(), "/desktop");
}
```

Test Name: Test Navigation to Mobile				
Description: Verify that clicking the " Mobile" link in the footer navigates the user to the newsletter subscription page.				
Pre-condition(s): User is on the home page.				
Test Steps: 1. Set the browser window size. 2. Navigate to the home page. 3. Scroll to the bottom of the page. 4. Click the " Mobile" link. 5. Verify that the URL contains /mobile.	**Test Data:** None	**Expected Result:** The user should be redirected to the Mobile page, and the URL should contain /mobile.	**Actual Result:** Navigation to the Mobile page worked as expected.	**Status:** PASS
Notes:				

```java
@Test
public void testNavigationToMobile() {
    navigateToFooterAndClick(() -> new HomePage(driver).clickMobile(), "/mobile");
}
```

3.15. Title Tests

The Title Tests verify that the page titles of critical GitHub pages are displayed correctly, ensuring proper identification of each page by the user.

Test Name: Test Login Page Title				
Description: Verify that the title of the GitHub login page contains the text "Sign in to GitHub".				
Pre-condition(s): User is on the login page.				
Test Steps: 1. Navigate to the login page URL: https://github.com/login . 2. Capture the page title. 3. Verify that the title contains "Sign in to GitHub".	**Test Data:** None	**Expected Result:** The login page title should contain "Sign in to GitHub".	**Actual Result:** The login page title was displayed as expected.	**Status:** PASS
Notes:				

```java
@Test
public void testLoginPageTitle() {
    driver.get("https://github.com/login");
    String pageTitle = driver.getTitle();
    assertTrue(pageTitle.contains("Sign in to GitHub"), "Login page title does not contain 'Sign in to GitHub'");
}
```

Test Name: Test Signup Page Title				
Description: Verify that the title of the GitHub Signup page contains the text "Sign up to GitHub".				
Pre-condition(s): User is on the signup page.				
Test Steps: 1. Navigate to the signup page URL: https://github.com/signup. 2. Capture the page title. 3. Verify that the title contains "Sign up to GitHub".	**Test Data:** None	**Expected Result:** The signup page title should contain "Sign up to GitHub".	**Actual Result:** The signup page title was displayed as expected.	**Status:** PASS
Notes:				

```java
@Test
public void testSignupPageTitle() {
    driver.get("https://github.com/signup");
    String pageTitle = driver.getTitle();
    assertTrue(pageTitle.contains("Sign up to GitHub"), "Signup page title does not contain 'Sign up to GitHub'");
}
```

Test Name: Test Pricing Page Title				
Description: Verify that the title of the GitHub pricing page contains the text "Pricing".				
Pre-condition(s): User is on the pricing page.				
Test Steps: 1. Navigate to the pricing page URL: https://github.com/pricing. 2. Capture the page title. 3. Verify that the title contains "Pricing".	**Test Data:** None	**Expected Result:** The pricing page title should contain "Pricing".	**Actual Result:** The pricing page title was displayed as expected.	**Status:** PASS
Notes:				

```java
@Test
public void testPricingPageTitle() {
    driver.get("https://github.com/pricing");
    String pageTitle = driver.getTitle();
    assertTrue(pageTitle.contains("Pricing"), "Pricing page title does not contain 'Pricing'");
}
```

Test Name: Test Copilot Page Title				
Description: Verify that the title of the GitHub Copilot page contains the text "GitHub Copilot".				
Pre-condition(s): User has access to the Copilot page.				
Test Steps: 1. Navigate to the Copilot page URL: https://github.com/features/copilot. 2. Capture the page title. 3. Verify that the title contains "GitHub Copilot".	**Test Data:** None	**Expected Result:** The Copilot page title should contain "GitHub Copilot".	**Actual Result:** The Copilot page title was displayed as expected.	**Status:** PASS
Notes:				

```java
@Test
public void testCopilotPageTitle() {
    driver.get("https://github.com/features/copilot");
    String pageTitle = driver.getTitle();
    assertTrue(pageTitle.contains("GitHub Copilot"), "GitHub Copilot page title does not contain 'GitHub Copilot'");
}
```

4. Conclusion

4.1. Testing Summary

Testing Tool	Total Tests	Passed Tests	Failed Tests
JUnit with Selenium	49	47 ~ 48	1 ~ 2

Failed Tests:

Accessibility Test: All links should contain text – this test failed due to missing text content in some links, which violates accessibility best practices.

SSL Certificate Test: This test is inconsistent as some pages intermittently fail to load, making validation of the SSL certificate unreliable.

4.2. Final Thoughts

Testing GitHub presented unique challenges due to the lack of proper id attributes for many elements, making element selection in tests more cumbersome. The absence of well-defined identifiers required relying on alternative locators, such as class, CSS selectors, or XPath, which were often less reliable and more prone to changes.

Additionally, GitHub's rate limiting posed a significant obstacle. To mitigate this, I had to implement random wait times of up to 5 seconds between requests to avoid triggering API limits. This approach, while effective, increased the overall test execution time and added complexity to the test suite.

Despite these challenges, the test cases successfully validated GitHub's core functionalities, such as navigation, user interactions, and error handling. These experiences underscore the importance of robust, test-friendly web design and highlight areas for improvement in GitHub's testability. Future efforts could benefit from enhanced support for automated testing and better documentation for UI element identification.