UI Testing for GitHub Using Selenium

Introduction

Automated UI testing is essential for ensuring the reliability and functionality of web applications. This project focuses on automating the UI testing of GitHub's core user flows, including login, account creation, and password reset functionalities. By leveraging Selenium WebDriver and Python, the goal is to identify potential issues and validate error handling and success paths on GitHub's interface.

Setup and Installation

- Downloading ChromeDriver

To begin testing, ChromeDriver must be installed to interact with the Chrome browser. It is crucial to download the version of ChromeDriver that matches the installed version of Chrome. The steps are as follows:

. Visit ChromeDriver download page https://developer.chrome.com/docs/chromedriver/downloads.

. Check your Chrome version by navigating to chrome://settings/help.

. Download the corresponding ChromeDriver version.

. Extract the file and add the path to your system environment variables.

- Installing Selenium

Selenium can be installed using pip by running the following command:

```
pip install selenium
```

After installation, verify by running a simple script to open a browser window using Selenium.

```
from selenium import webdriver

driver = webdriver.Chrome()
driver.get("https://www.google.com")
```

Planning the Test Cases

Before writing the automation scripts, test cases were carefully planned based on critical user flows on GitHub. The following six test cases were selected:

. Access GitHub Login Page

. Unsuccessful Login (Wrong Credentials)

. Successful Login

. Account Creation (Invalid Inputs)

. Forgot Password Flow

. Account Creation with Existing Email

Code Structure and Libraries

-Main Libraries Used

. Selenium – Automating web interactions

. WebDriverWait – Explicit waits for dynamic elements

. By – Locating elements on the page

. time – Adding delays when necessary for smooth execution

- Key Functions

. driver.get(url) – Opens the specified URL

. send_keys() – Simulates typing into input fields

. click() – Simulates button clicks

. save_screenshot() – Captures screenshots at each step

. WebDriverWait() – Waits for elements to load before interacting

-Common Code Elements

. Explicit Waits – Ensuring elements load fully before interacting

. Error Handling – Capturing unexpected errors with try-except blocks

. Screenshots – Documenting each stage of testing to verify results

Detailed Breakdown of Test Cases

-Test Case 1: Access GitHub Login Page

Purpose: Verify the GitHub login page loads successfully.Key Code Snippet:

```
driver.get("https://github.com/login")
driver.save_screenshot("github_login_page.png")
```

Result: The page loaded successfully, and a screenshot was taken.

-Test Case 2: Unsuccessful Login (Wrong Credentials)

Purpose: Test the error message displayed for incorrect login details.Key Code Snippet:

```
driver.find_element(By.ID, "login_field").send_keys("wrong_username")
driver.find_element(By.ID, "password").send_keys("wrong_password")
driver.find_element(By.NAME, "commit").click()
driver.save_screenshot("failed_login.png")
```

Result: The system displayed an 'Incorrect username or password' message.

-Test Case 3: Successful Login

Purpose: Ensure login works with valid credentials.
Key Code Snippet:

```
driver.find_element(By.ID, "login_field").send_keys("valid_username")
driver.find_element(By.ID, "password").send_keys("valid_password")
driver.find_element(By.NAME, "commit").click()
driver.save_screenshot("successful_login.png")
```

Result: Login was successful, and the dashboard loaded.

-Test Case 4: Account Creation (Invalid Inputs)

Purpose: Test account creation with invalid email, password, and username.Key Code Snippet:

```
driver.get("https://github.com/signup")
driver.find_element(By.ID, "email").send_keys("invalid_email@example")
driver.find_element(By.ID, "password").send_keys("123")
driver.find_element(By.ID, "login").send_keys("test_user")
driver.save_screenshot("failed_acc_creation.png")
```

Result: Error messages were triggered for all invalid inputs.

-Test Case 5: Forgot Password Flow

Purpose: Verify the password reset function.Key Code Snippet:

```
driver.get("https://github.com/password_reset")
driver.find_element(By.ID, "email_field").send_keys("valid_email@example.com")
driver.save_screenshot("forgot_password.png")
```

Result: Password reset email was successfully requested.

-Test Case 6: Account Creation with Existing Email

Purpose: Ensure GitHub prevents duplicate accounts with existing emails.Key Code Snippet:

```
driver.get("https://github.com/signup")
driver.find_element(By.ID, "email").send_keys("existing_email@example.com")
driver.save_screenshot("existing_email.png")
```

Result: Error appeared indicating the email was already associated with an account.

Challenges and Solutions

Dynamic Elements: Used explicit waits to ensure elements loaded before interacting.

Timing Issues: Applied WebDriverWait to avoid timing-related failures.

Screenshots: Implemented screenshots at each step to verify and document results.

Conclusion

Automating GitHub's UI using Selenium provided valuable insights into the platform's user flows. The test cases verified error handling, login processes, and account management. All six tests were completed successfully, highlighting GitHub's robust error validation mechanisms. Future improvements could include automating additional flows and integrating the tests into a CI/CD pipeline for continuous validation.