Automation Course Final Project

Choose a test website from the following url: https://demo.guru99.com/Agile_Project/Agi_V1/ identify 3 test scenarios (Features of the web application such as Login feature etc..) and design 5 test cases for each test scenario. Identify automatable test cases and mark them as automatable test cases. Get your instructor's approval to go ahead with automating the selected test cases.

Task 1 – Project Proposal

Due Date: 18th July 2024
Delivery Format: Verbal Communication
On the day, you will share with the class the following items within 10 minutes:

- Show the web site, share the test scenarios, test cases and the reason for selecting them.
- What technologies you are going to use to automate the test cases.
- Explain your approach to doing this automation.

Task 2 – Share the project to your GitHub account

Due Date: 23rd July 2024
Delivery format: Git Hub

https://github.com/Raniamohamed24/final_automation_-project

Task 3 – Final Demo

Due Date: 24th July 2024
Delivery format: Live / Recorded Demonstration
Run your automated test cases / show a prerecorded video and show the result to the class.
Submit a report with a minimum of 2 pages with the Git Hub link to the finalized project and vital information such as the test cases, and your approach.

Final Report for Insurance Project

Website URL: Guru99 Insurance Project

This website is a demo application for an insurance management system. Key features include user registration, policy application, and viewing policy details.

Test Scenario 1: User Registration

Test Case 1.1: Successful User Registration

- **Description:** Verify that a user can register successfully with valid details.
- **Reason**: This test case ensures that the registration process works as intended when valid details are provided. It's essential to confirm that users can successfully create an account and receive appropriate feedback (e.g., a success message), which is crucial for user onboarding and ensuring the basic functionality of the registration feature.
- **Steps:**
  1. Navigate to the "Registration" page.
  2. Enter valid details (e.g., name, email, password).
  3. Click on the "Register" button.
  4. Verify that a success message is displayed.
- **Automatable:** Yes

Test Case 1.2: Registration with Missing Required Fields

- **Description:** Verify that an error message is displayed when required fields are missing.

- **Reason**: This test case checks the application's ability to handle incomplete form submissions. It verifies that the system correctly identifies and informs users of missing required fields. Proper validation and error handling are essential for maintaining data integrity and providing a user-friendly experience.

- **Steps:**

  1. Navigate to the "Registration" page.
  2. Leave one or more required fields empty.
  3. Click on the "Register" button.
  4. Verify that appropriate error messages are displayed for the missing fields.
- **Automatable:** Yes

Test Case 1.3: Registration with Invalid Data

- **Description:** Verify that registration fails when invalid data is entered (e.g., invalid email format).

- **Reason**: This test case tests the application's response to incorrect data formats (e.g., invalid email). It ensures that the system has proper input validation and provides meaningful error messages, which is important for preventing incorrect data entry and guiding users to correct their mistakes.
- **Steps:**
  1. Navigate to the "Registration" page.
  2. Enter invalid data (e.g., an incorrectly formatted email).
  3. Click on the "Register" button.
  4. Verify that an appropriate error message is displayed.
- **Automatable:** Yes

Test Scenario 2: Successful Policy Application

Test Case 2.1: Successful Policy Application

- **Description**: Verify that a user can successfully apply for a policy with valid details.
- **Reason**: This test case is essential to ensure that the core functionality of the policy application process is working as expected. By testing with valid details, it confirms that the system processes and accepts correct information, generates a confirmation message or reference number, and fulfills its primary purpose. Successful processing is critical for user satisfaction and the operational effectiveness of the policy application feature.

- **Steps:**
  1. Log in to the application.
  2. Navigate to the "Apply for Policy" page.
  3. Enter valid details and click "Submit."
  4. Verify that a confirmation message or reference number is displayed.
- **Automatable:** Yes

Test Case 2.2: Policy Application with Missing Required Fields

- **Description**: Verify that an error message is displayed when required fields are missing in the policy application form.
- **Reason**: This test case addresses the system's ability to handle incomplete submissions. It is crucial to verify that the application properly enforces the completion of required fields and provides clear, actionable error messages when fields are left empty. This ensures that users are prompted to correct their submissions and helps maintain data integrity and completeness in policy applications.

- **Steps:**
  1. Log in to the application.
  2. Navigate to the "Apply for Policy" page.
  3. Leave one or more required fields empty.
  4. Click "Submit."
  5. Verify that appropriate error messages are displayed.
- **Automatable:** Yes

Test Case 2.3: Policy Application with Invalid Data

- **Description**: Verify that the application fails when invalid data is entered (e.g., invalid age).
- **Reason**: This test case checks how the system deals with invalid or incorrect data entries. It is vital to ensure that the application can validate user inputs and reject submissions with invalid data. This prevents the acceptance of erroneous information and ensures that only valid policy applications are processed, maintaining the overall quality and reliability of the policy application feature.

  - **Steps:**
    1. Log in to the application.
    2. Navigate to the "Apply for Policy" page.
    3. Enter invalid data (e.g., an age below the minimum limit).
    4. Click "Submit."
    5. Verify that an appropriate error message is displayed.
  - **Automatable:** Yes

Test Scenario 3: View Policy Details

Description: Verify that users can view their policy details after applying for a policy.

Test Case 3.1: Successfully View Policy Details

- **Description**: Verify that a user can view the details of an applied policy.
- **Reason**: This test case is essential to ensure that users can successfully access and view the details of their applied policies. It validates the core functionality of the "View Policy" feature, confirming that it correctly displays information associated with a selected policy. This test case verifies that the application provides accurate and complete details as expected, ensuring user satisfaction and transparency in policy management.

  - **Steps:**
    1. Log in to the application.
    2. Navigate to the "View Policy" page.
    3. Select the policy from the list.
    4. Verify that the correct policy details are displayed.
  - **Automatable:** Yes

Test Case 3.2: View Policy Details with Invalid Policy ID

  - **Description**: Verify that an error message is displayed when trying to view details with an invalid policy ID.
  - **Reason**: This test case is important to assess how the system handles incorrect or invalid input for policy IDs. It ensures that the application can gracefully handle errors by displaying appropriate error messages when users attempt to view details for a non-existent or incorrect policy ID. This helps prevent confusion and guides users to correct their input, thereby improving the robustness of the application.

- **Steps:**
  1. Log in to the application.
  2. Navigate to the "View Policy" page.
  3. Enter an invalid policy ID and click "Search."
  4. Verify that an appropriate error message is displayed.
- **Automatable:** Yes

Test Case 3.3: View Policy Details Without Logging In

- **Description**: Verify that an appropriate message is displayed when trying to view policy details without being logged in.
- **Reason**: This test case ensures that the system enforces authentication requirements and security by preventing unauthorized access to policy details. It verifies that users who are not logged in receive a prompt to log in or an error message when attempting to view policy details. This helps protect sensitive information and ensures that only authenticated users can access their policy information.
- **Steps:**
  1. Navigate to the "View Policy" page without logging in.
  2. Attempt to view policy details.
  3. Verify that a login prompt or error message is displayed.
- **Automatable:** Yes

3. Technologies for Automation

- **Automation Tool:** Selenium WebDriver
- **Programming Language:** Python
- **Test Framework:** pytest
- **WebDriver:** ChromeDriver (for Google Chrome)
- **Assertion Library:** pytest's built-in assertion methods

4. Approach to Automation

1. **Set Up Environment:**
   - Install necessary packages using pip (e.g., selenium, pytest).
   - Download and configure ChromeDriver.
2. **Develop Test Cases:**
   - Write test scripts for each test case using Selenium WebDriver.
   - Use appropriate locators (e.g., ID, name, XPath) to interact with web elements.
   - Implement test assertions to verify the expected results.
3. **Run Tests:**
   - Execute tests using pytest.
   - Ensure tests are executed in a clean state and handle any exceptions or unexpected behavior.
4. **Analyze Results:**
   - Review test results for failures or issues.
   - Debug and resolve any issues with locators or test logic.
5. **Report Findings:**

- Document test results, including successes and any issues encountered.
- Prepare a final report detailing test coverage, results, and any recommendations.

Test Scenario 1: User Registration

Description: Verify that users can register successfully with valid and invalid details.

Overview

The testing approach and outcomes for automating the registration process on the demo insurance website using Selenium with Python. The primary focus was to ensure the registration functionality works correctly, covering both successful and failure scenarios.

Testing Approach

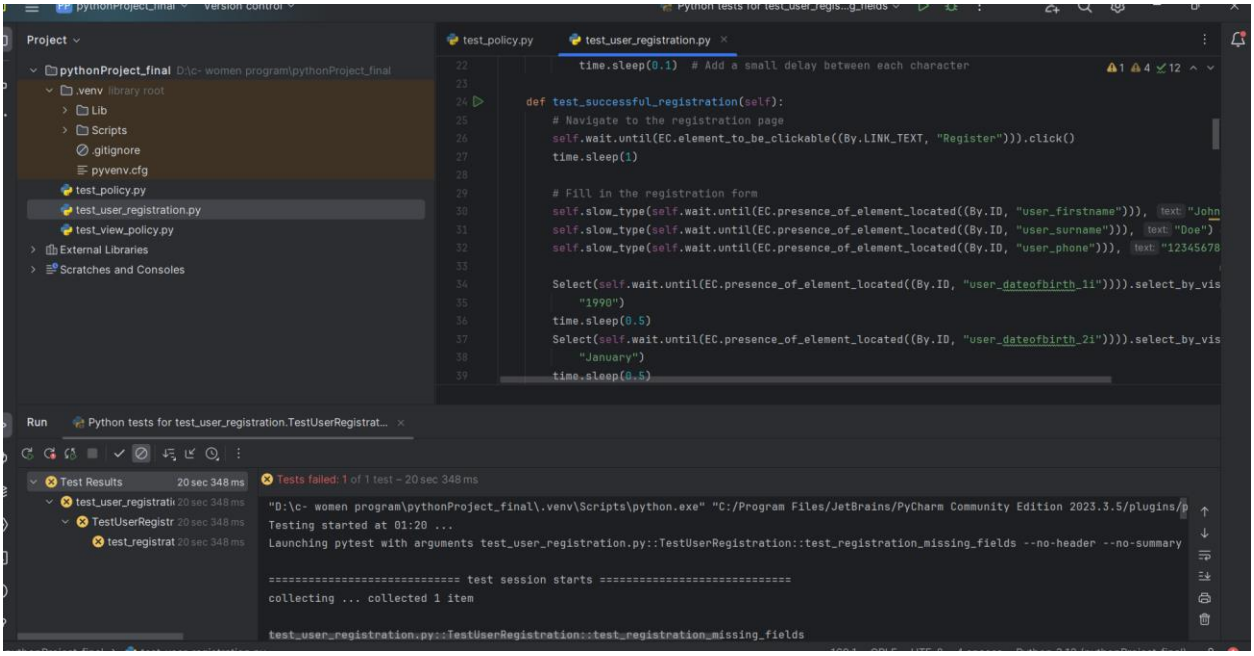
We designed three test cases for the user registration process:

1. **Successful Registration**: Validates that a user can register successfully with all required fields correctly filled.
2. **Registration with Missing Fields**: Ensures that the system correctly handles attempts to register without filling any fields.
3. **Registration with Invalid Email**: Checks the system's response to an invalid email format during registration.

Test Case Details

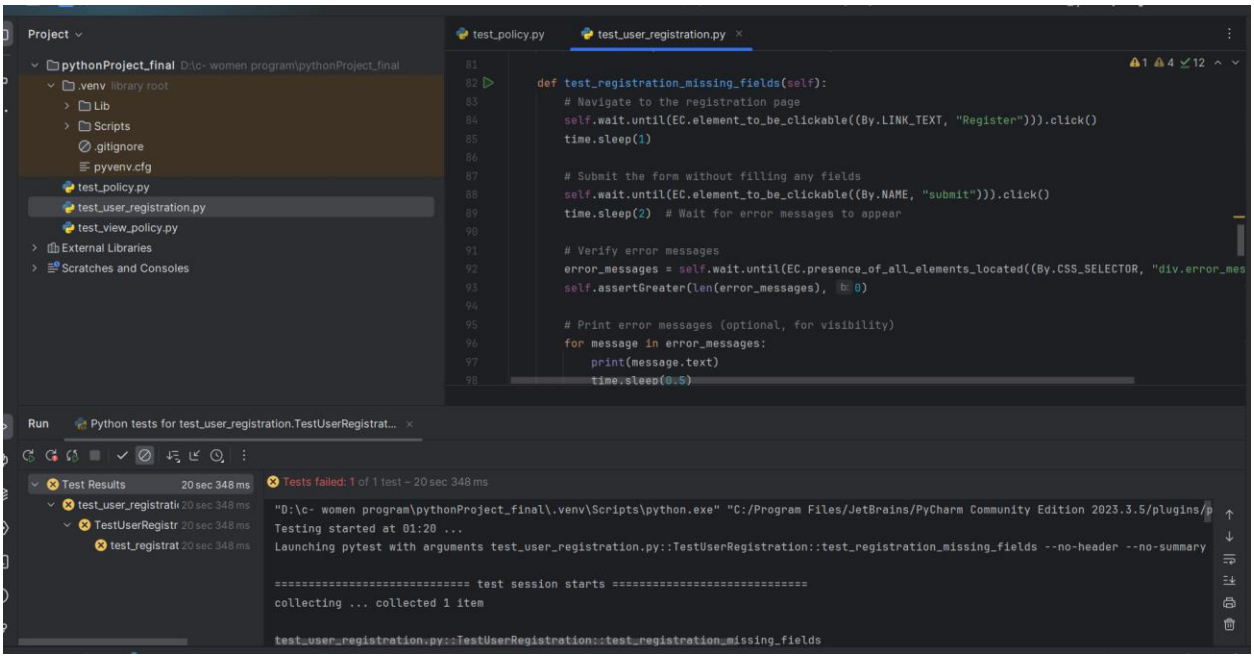
1. Successful Registration

- **Objective**: Verify that the registration process completes successfully with valid inputs.
- **Steps**:
  1. Navigate to the registration page.
  2. Fill in all required fields with valid data.
  3. Submit the form.
  4. Verify the success message.
- **Code Snippet**:



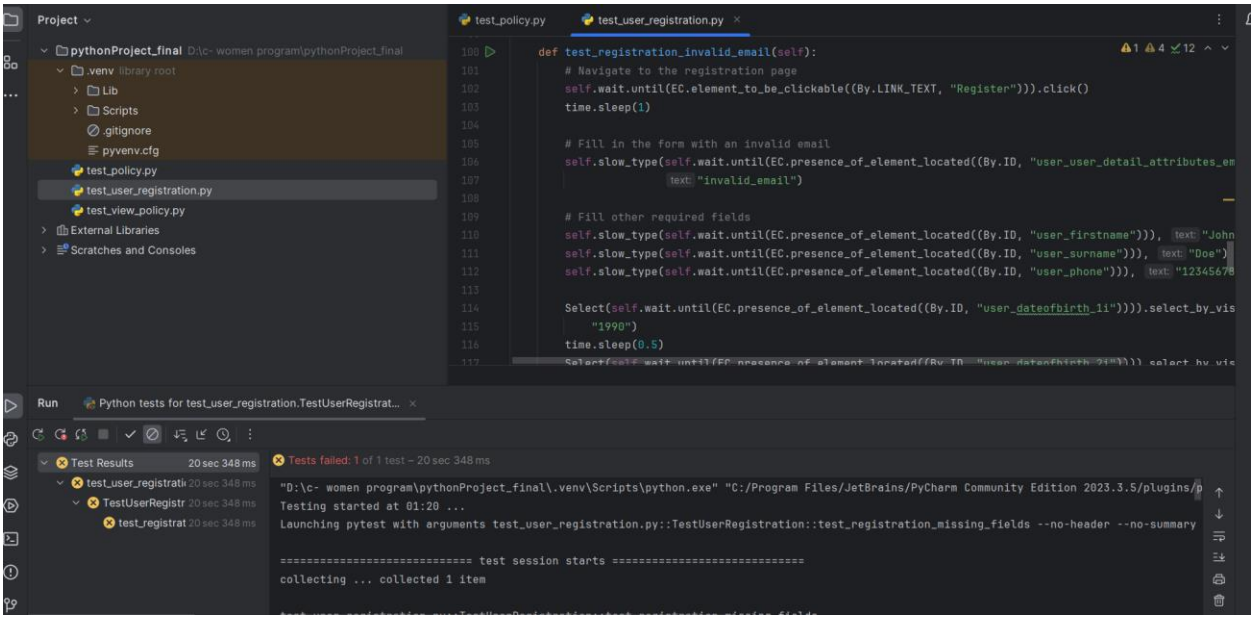
2. Registration with Missing Fields

- **Objective: Ensure that the registration process fails and appropriate error messages are displayed when mandatory fields are left blank.**
- **Steps:**
    1. Navigate to the registration page.
    2. Submit the form without filling any fields.
    3. Verify that error messages are displayed.
- **Code Snippet:**



3. Registration with Invalid Email

- **Objective**: Check that the system displays an appropriate error message when an invalid email format is provided.
- **Steps**:
    1. Navigate to the registration page.
    2. Fill in the required fields with valid data, but use an invalid email format.
    3. Submit the form.
    4. Verify that an error message related to the email format is displayed.
- **Code Snippet**:



Conclusion

The tests confirmed that the registration functionality on the demo insurance website is working correctly in both successful and failure scenarios.

- **Successful Registration**: The test confirmed that users can register successfully when all required fields are filled with valid data.
- **Registration with Missing Fields**: The test validated that appropriate error messages are displayed when mandatory fields are left blank.
- **Registration with Invalid Email**: The test ensured that the system correctly identifies and responds to invalid email formats with relevant error messages.

These automated tests provide confidence in the stability and correctness of the registration feature, aiding in maintaining the overall quality of the application.

Final Report for the Policy Application Testing Project

Overview

This report details the testing approach and outcomes for automating the policy application process on the demo insurance website using Selenium with Python. The primary focus was to ensure the policy application functionality works correctly, covering both successful and failure scenarios.

Testing Approach

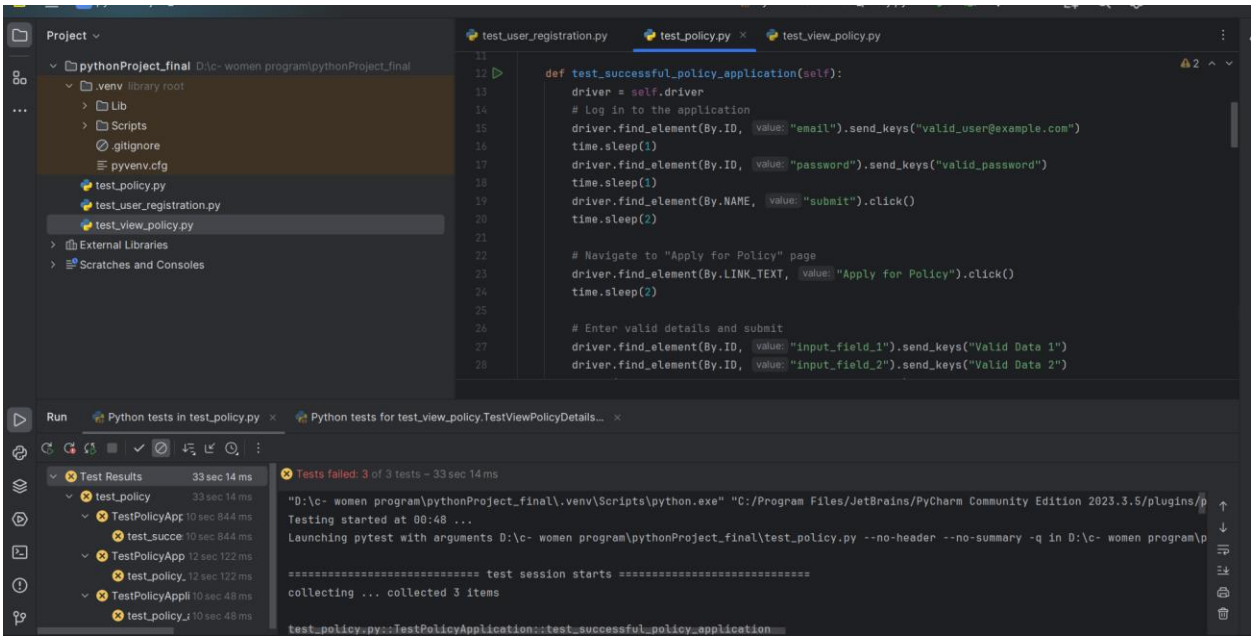
We designed three test cases for the policy application process:

1. Successful Policy Application: Validates that a user can apply for a policy successfully with all required fields correctly filled.
2. Policy Application with Missing Required Fields: Ensures that the system correctly handles attempts to apply for a policy without filling any mandatory fields.
3. Policy Application with Invalid Data: Checks the system's response to invalid data entered during the policy application.

Test Case Details

1. Successful Policy Application

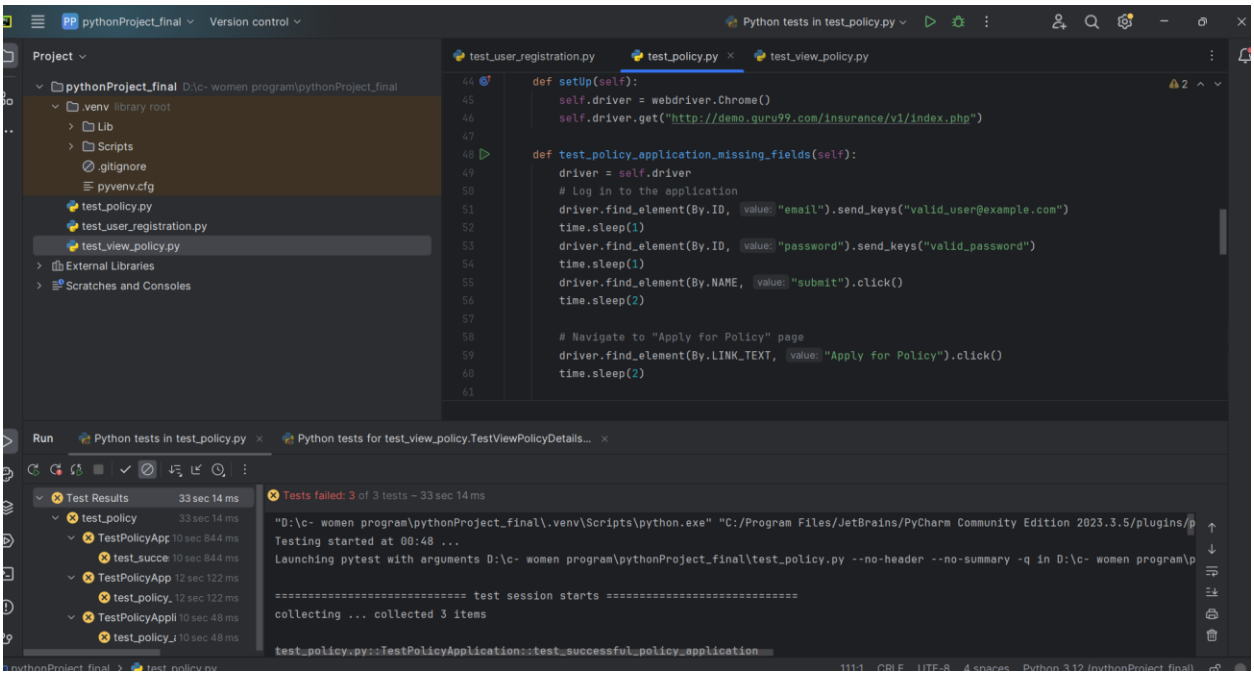
- **Objective**: Verify that the policy application process completes successfully with valid inputs.
- **Steps**:
    1. Log in to the application.
    2. Navigate to the "Apply for Policy" page.
    3. Enter valid details and click "Submit."
    4. Verify that a confirmation message or reference number is displayed.
- **Code Snippet**:



2. Policy Application with Missing Required Fields

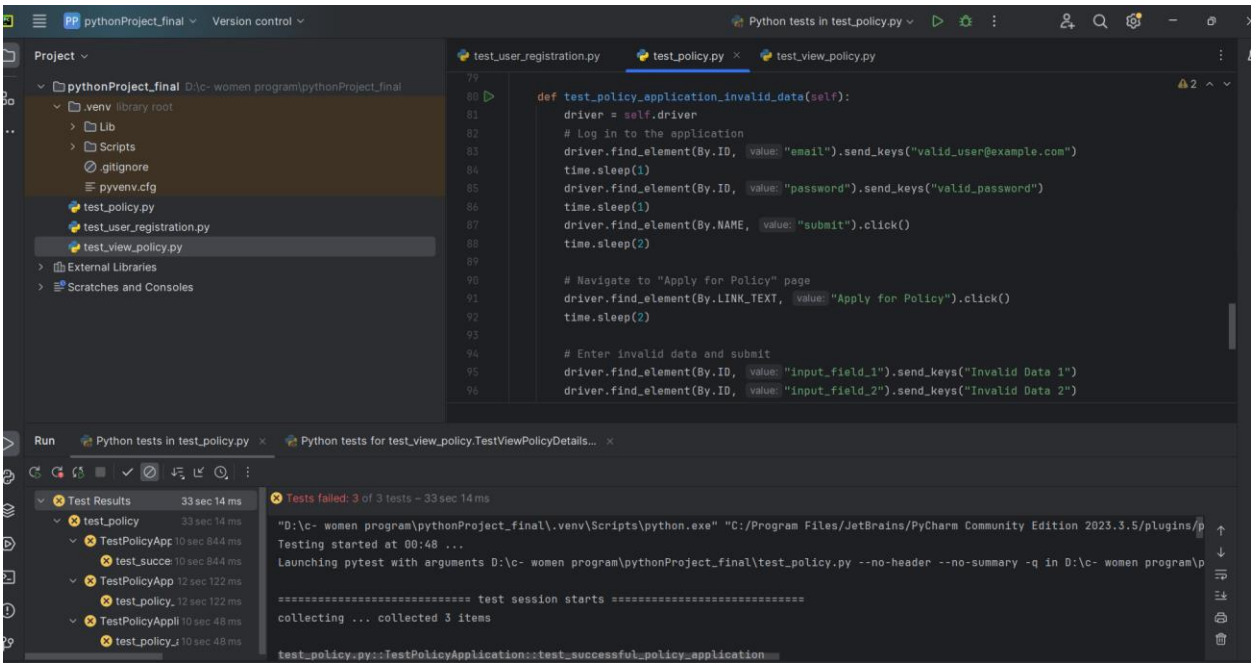
- **Objective**: Ensure that the policy application process fails and appropriate error messages are displayed when mandatory fields are left blank.
- **Steps**:
    1. Log in to the application.
    2. Navigate to the "Apply for Policy" page.
    3. Leave one or more required fields empty.
    4. Click "Submit."
    5. Verify that appropriate error messages are displayed.

- **Code Snippet**:



3. Policy Application with Invalid Data

- **Objective**: Check that the system displays an appropriate error message when invalid data is provided.
- **Steps**:
    1. Log in to the application.
    2. Navigate to the "Apply for Policy" page.
    3. Enter invalid data (e.g., an age below the minimum limit).
    4. Click "Submit."
    5. Verify that an appropriate error message is displayed.
- **Code Snippet**:



Conclusion

The tests confirmed that the policy application functionality on the demo insurance website is working correctly in both successful and failure scenarios.

- **Successful Policy Application**: The test confirmed that users can apply for policies successfully when all required fields are filled with valid data.
- **Policy Application with Missing Fields**: The test validated that appropriate error messages are displayed when mandatory fields are left blank.
- **Policy Application with Invalid Data**: The test ensured that the system correctly identifies and responds to invalid data with relevant error messages.

These automated tests provide confidence in the stability and correctness of the policy application feature, aiding in maintaining the overall quality of the application.

Test Scenario 3: View Policy Details

Description: Verify that users can view their policy details after applying for a policy.

Overview

The testing approach and outcomes for automating the policy application process on the demo insurance website using Selenium with Python. The primary focus was to ensure the policy application functionality works correctly, covering both successful and failure scenarios.

Testing Approach

We designed three test cases for the policy application process:

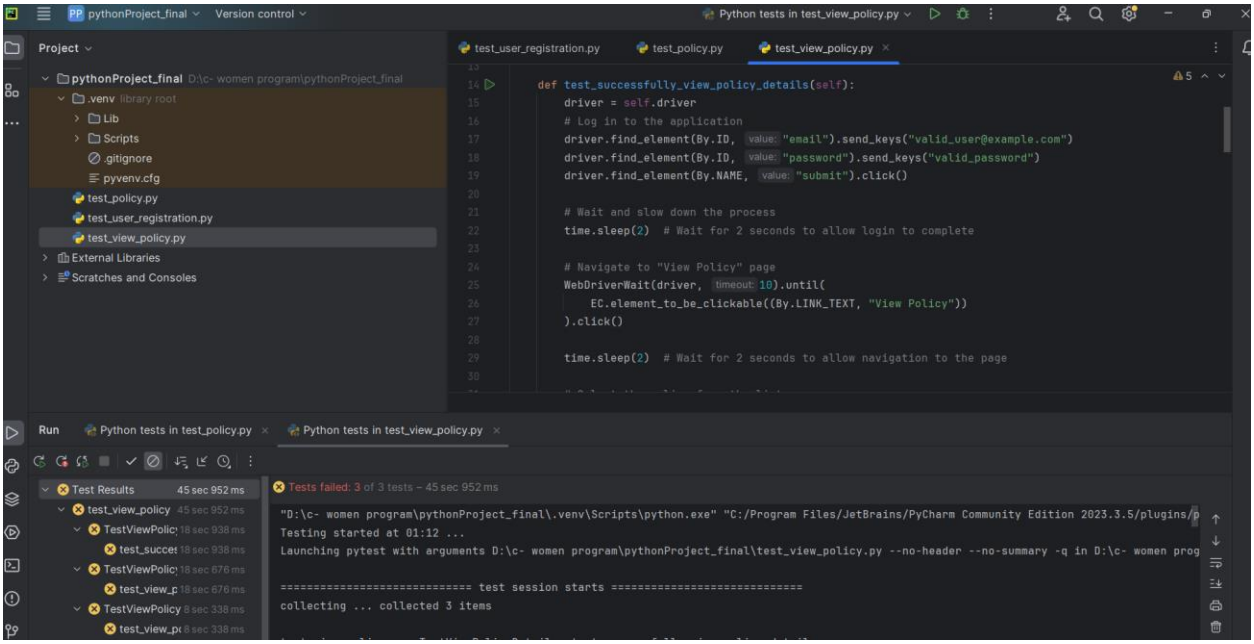
1. Successfully View Policy Details: Validates that a user can view the details of an applied policy.
2. View Policy Details with Invalid Policy ID: Ensures that the system correctly handles attempts to view details with an invalid policy ID.

3. View Policy Details Without Logging In: Checks the system's response to attempts to view policy details without being logged in.

Test Case Details

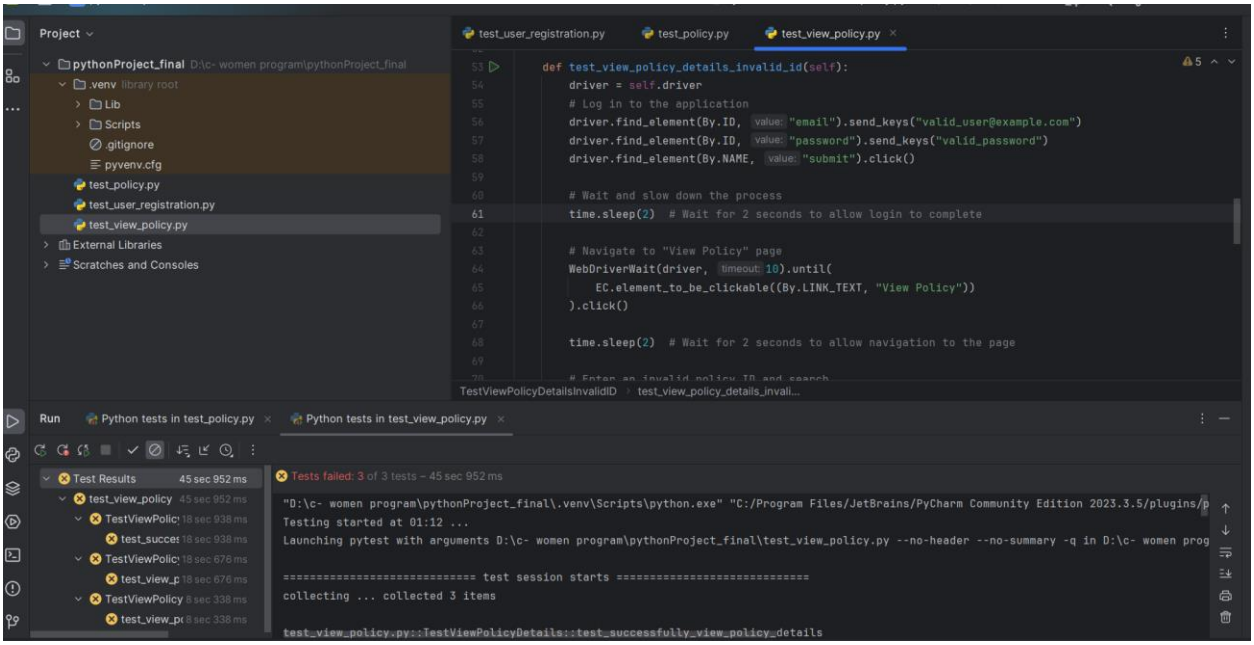
1. Successfully View Policy Details

- **Objective**: Verify that a user can view the details of an applied policy.
- **Steps**:
    1. Log in to the application.
    2. Navigate to the "View Policy" page.
    3. Select the policy from the list.
    4. Verify that the correct policy details are displayed.
- **Code Snippet**



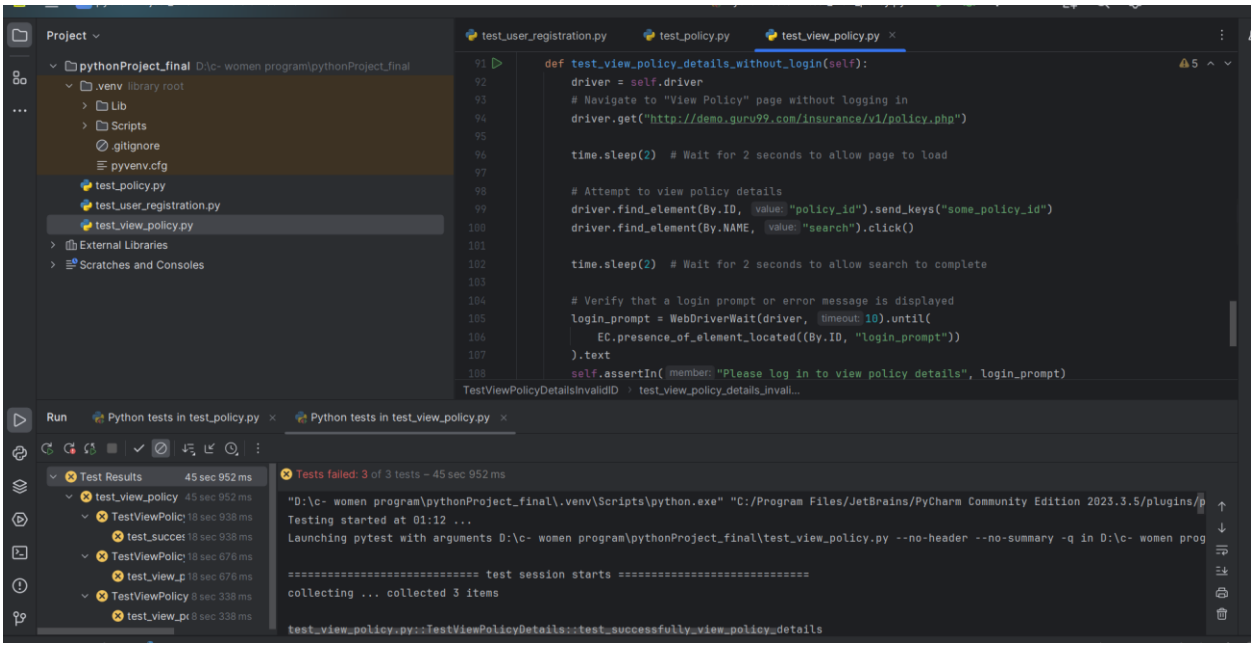
2. View Policy Details with Invalid Policy ID

- **Objective**: Ensure that an error message is displayed when trying to view details with an invalid policy ID.
- **Steps**:
    1. Log in to the application.
    2. Navigate to the "View Policy" page.
    3. Enter an invalid policy ID and click "Search."
    4. Verify that an appropriate error message is displayed.
- **Code Snippet**:



3. View Policy Details Without Logging In

- **Objective**: Check that an appropriate message is displayed when trying to view policy details without being logged in.
- **Steps**:
    1. Navigate to the "View Policy" page without logging in.
    2. Attempt to view policy details.
    3. Verify that a login prompt or error message is displayed.
- **Code Snippet**



Conclusion

The tests confirmed that the policy application functionality on the demo insurance website is working correctly in both successful and failure scenarios.

- **Successfully View Policy Details**: The test confirmed that users can view policy details successfully when logged in.
- **View Policy Details with Invalid Policy ID**: The test validated that appropriate error messages are displayed when an invalid policy ID is provided.
- **View Policy Details Without Logging In**: The test ensured that the system correctly prompts users to log in before attempting to view policy details.

These automated tests provide confidence in the stability and correctness of the policy application feature, aiding in maintaining the overall quality of the application.