Automated Testing for MakeMyTrip International Trip Planning Feature

Developed By: Sneha CK (snehajayaprakash2017@gmail.com)

Introduction

The problem scenario involves testing a new feature introduced by MakeMyTrip for planning international trips from India. The goal is to ensure this feature functions correctly and meets user expectations by creating comprehensive test cases and automating the testing process.

The task requires writing test cases for the feature and automating the user flow. The flow includes:

1. Selecting a departure city
2. Choosing an international destination
3. Setting travel dates and duration
4. Searching for flights
5. Verifying available flights

The implementation was done considering the following key points: (adhering to software development best practices)

1. Code should be in a maintainable, modular and precise format.
2. Ensure the automation framework is robust and scalable.
3. Follow coding standards and best practices.
4. Facilitate easy setup and execution with clear instructions.
5. Ensure tests are reusable for similar functionalities.

The following are the objectives of the provided solution:

1. Write test automation for the given specific flow described in the previous section.
2. Identify, document and write additional smoke test cases.
3. Ensuring that the functionality works correctly and meets the specified requirements.
4. Identify any errors in the flow if any.
5. Perform extensive documentation of the activity.
6. Provide detailed and clear test reports for easy analysis of test results.

Problem Analysis

The following are the key aspects to address:

Feature Navigation and Interaction:

- Users must be able to navigate to the "Flights" page from the main interface.
- The feature requires users to select a departure city, choose an international destination, set travel dates and duration, and then initiate a search for flights.
- Ensure that all interactive elements (dropdowns, buttons, etc.) work as intended.

Test Case Development:

- Functional Test Cases:
    - Verify that selecting "From" as "Bangalore" and "To" as "Dubai" functions correctly.
    - Check the accuracy of the Dates and Duration selection and ensure the search functionality returns results.

- Smoke Test Cases:
    - Prioritize basic scenarios that cover the core functionality of the feature to ensure it is working before detailed testing.
    - Examples include verifying that the search returns results and that the filtering options are operational.

Automation Flow:

- Navigation and Selection:
    - Automate the process of navigating to the "Flights" page, selecting "Bangalore" as the departure city, and "Dubai" as the destination.
    - Set travel dates and adjust the duration as specified.

- Flight Search and Validation:
    - Initiate the search and retrieve dates where the flight price is below the median price for the month.
    - Select a weekend date or the date with the lowest price if a weekend is unavailable.
    - Verify that at least one flight is available for the selected date.

Additional Considerations:

- Edge Cases:
    - Handle scenarios such as no flights available for the selected dates or destination.
    - Validate how the system behaves with unexpected inputs or selections.

- Performance:
    - Ensure the automation scripts execute efficiently and within a reasonable timeframe.

- Documentation and Reporting:
    - Setup Instructions:
        - Provide clear documentation for setting up the test environment and executing the automated tests.
    - Test Reports:
        - Include detailed test reports that capture the results of test executions, including any issues or failures encountered.

Proposed Solution

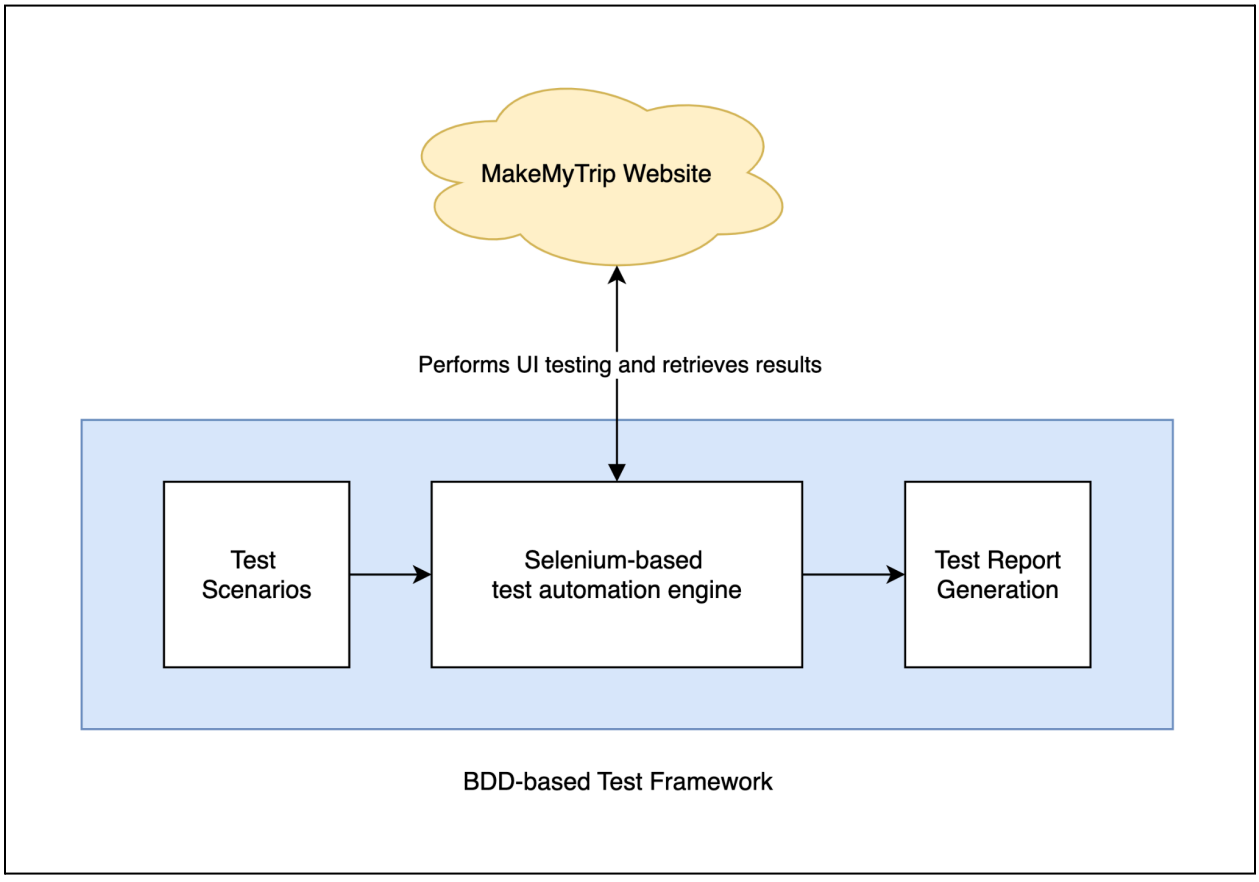
- **Framework Setup**
    - Cucumber BDD:
        - Utilize Cucumber to write test scenarios in a behavior-driven format.
        - This allows to define test cases in plain language using Gherkin syntax, which promotes clarity and collaboration with non-technical stakeholders.
    - Selenium:
        - Used Selenium for browser automation.
        - This tool is used for interacting with the web browser for BDD scenarios, such as filling out forms, clicking buttons, and validating content.
    - Pytest:
        - Integrate Pytest to manage and execute the test cases.
        - Pytest can be used for its rich features, including fixtures, parameterization, and its ability to handle both unit and integration tests.
        - It will manage the execution of BDD scenarios and provide detailed test reports.

- **Test Case Categorization**
    - **Smoke Testing**:
        - This includes essential tests that cover critical functionalities of the international trip planning feature.
        - The goal is to verify that the core functions (e.g., searching for flights, booking, payment) work as expected. These tests should run frequently, such as on every build or deployment, to catch any critical issues early.
    - **Regression Testing**:
        - These tests focus on verifying that new code changes haven't adversely affected existing functionality.

- This category is broader and includes various test scenarios covering all aspects of the international trip planning feature.
- They are run less frequently, such as during major releases or after significant code changes.

- **HTML Reports**
  - Configure Pytest to generate HTML reports that detail test results. This includes passing and failing test cases, execution time, and any errors or exceptions encountered with screenshots.
  - HTML reports offer a user-friendly way to review test results and share them with stakeholders.

Architecture Diagram



Implementation Plan

1. **Environment Setup:**

a. Choose a test framework (e.g., Pytest, Unittest).
b. Install necessary requirements - which includes the test framework, Selenium, Web Driver for Selenium.
c. Configuring the web driver to work with Selenium.
d. Creating a suitable directory structure for modular development.

2. **Defining Page Object Model (POM) and Interaction Methods**
    a. Define Page Classes.
    b. Define locators for all elements.
    c. Implement methods to interact with each element.
    d. Methods for clicking elements, entering text, verifying element presence, etc.

3. **Test Case Implementation**
    a. Write test cases to verify each aspect of the Dashboard functionality.
    b. Create a conftest.py file for Pytest to handle setup and teardown.
    c. Define fixtures for browser setup and teardown.
    d. Implement Test Cases using the methods defined previously.

4. **Running Tests and Reporting**
    a. Run the test cases using the chosen test framework and wait for the results.
    b. Use reporting plugins (e.g., Pytest HTML) to generate test reports.

Conclusion

An automated testing framework was developed for the feature. This was built using Selenium, Pytest and BDD framework. This included creation of 11 test cases in which 5 smoke test cases and 6 regression test cases. The tests were run and reports were generated.

- Reports are available in HTML format under the reports directory
- Report, architecture diagram and other relevant screenshots are available under the docs directory.

References

- https://www.selenium.dev/documentation/webdriver/elements/finders/