Ömer Faruk AKGEYİK

omer.akgeyik@outlook.com
+90 530 422 52 75

11 December 2024

INTRODUCTION

This document outlines the solutions implemented for the QA Engineer Task provided by OpenPayd. The tasks were designed to evaluate technical abilities in various domains, including test automation, web crawling, and API testing. The solutions were developed using Java, Selenium, and relevant tools, adhering to the specified requirements. Below, you will find detailed instructions on how to execute the code and review the results for each task.

This document consists of the following sections:

1. Project Setup and Execution
2. Technical Details and Technologies Used
3. Project Access and File Structure
4. Additional Information

Project Setup and Execution

To develop this project, I used IntelliJ IDEA to create a Maven project. The necessary dependencies were added to the pom.xml file, which has been included in the GitHub repository along with the config.properties file.

For execution, I utilized JUnit and implemented a Runner class. To ensure each feature file can be executed individually, I added appropriate annotations to the Cucumber files.

To run the project:

1. Pull the repository from GitHub into your local environment.
2. Open it as a Maven project in your IDE.
3. Reload the Maven dependencies using the pom.xml file.
4. Execute the tests using the Runner class.

This setup ensures that the project is modular, configurable, and easy to execute.

Technical Details and Technologies Used

This project was developed using the following tools and technologies:

- **Java (JDK 21.0.5)**: The primary programming language used for implementing the test scenarios and functionalities.
- **Selenium (4.25.0)**: Utilized for automating browser interactions to perform web-based tests.
- **Cucumber-Java (7.20.1)**: Used for behavior-driven development (BDD), allowing the test cases to be written in a human-readable format and linked to automated step definitions.
- **JUnit (5.11.3)**: Employed for running the tests and generating reports, providing a structured framework for assertions and test management.
- **Rest-Assured (5.5.0)**: Used for API testing, ensuring proper validation of RESTful API endpoints.
- **Maven (3.9.9)**: Used for project build management, dependency handling, and ensuring the environment is consistent and modular.
- **Assertions**: Integrated to validate the expected outcomes in test scenarios.
- **Properties File**: A config.properties file was used to manage environment configurations and provide flexibility.

Technology Versions

Tool/Library	Version
Java (JDK)	21.0.5
Selenium	4.25.0
Cucumber-Java	7.20.1
JUnit	5.11.3
Rest-Assured	5.5.0
Maven	3.9.9

By combining these tools and technologies, the project achieves a robust and maintainable structure for test automation. Selenium handles the browser-level actions, while Cucumber and JUnit provide a clear and organized workflow for executing and reporting test results. Maven simplifies dependency management, ensuring compatibility and ease of setup.

Project Access and File Structure

Accessing the Project

The project is hosted on a GitHub repository. To access and run the project, clone the repository using the provided link and open it in an IDE that supports Maven projects, such as IntelliJ IDEA. The pom.xml file is included in the repository to manage dependencies, which should be reloaded using Maven before running the project.

File Structure

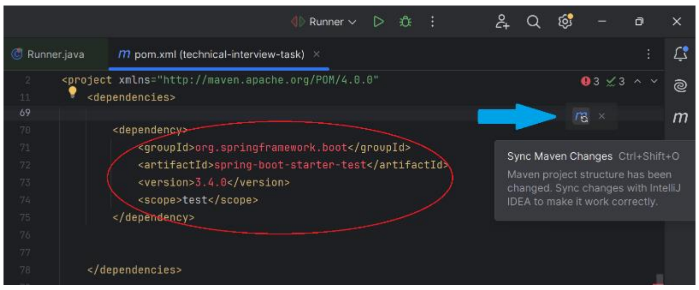
The project follows a standard Maven directory structure with additional custom packages and files for test automation:

- **pom.xml**: Contains project dependencies, plugin configurations, and build specifications.
- **src Folder**: The main source folder, structured as follows:
    - **src/test/java**:
        - **pages Package**: Contains classes that store the locators for web elements on the Amazon website.
        - **runners Package**: Contains the Runner class, used to execute the test scenarios.
        - **stepdefinitions Package**: Implements the methods corresponding to the steps defined in the feature files.
        - **utilities Package**: Contains utility classes:
            - Driver: Manages WebDriver instances.
            - ConfigReader: Reads properties from the config.properties file.
            - ReusableMethods: Provides commonly used helper methods.
    - **src/test/resources**:
        - **features Folder**: Stores the .feature files used by Cucumber to define test scenarios.
- **target Folder**: Contains generated test artifacts, including JUnit HTML reports, after the tests are executed.
- **config.properties File**: Stores configuration data such as base URLs, credentials, or environment-specific settings.

This modular file structure ensures better organization, reusability, and maintainability of the project. It separates concerns across packages and files, making it easier to navigate and manage.

Additional Information

1- If you encounter issues where the dependencies added to the pom.xml file are highlighted in red and errors are displayed, as shown in the image below, you can resolve this by syncing the Maven changes. Simply click the button indicated by the blue arrow in the image. This will reload the Maven configuration and ensure that the required dependencies are correctly added to your project



```
<project xmlns="http://maven.apache.org/POM/4.0.0"
    <dependencies>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <version>3.4.0</version>
            <scope>test</scope>
        </dependency>


    </dependencies>
```

Sync Maven Changes Ctrl+Shift+O

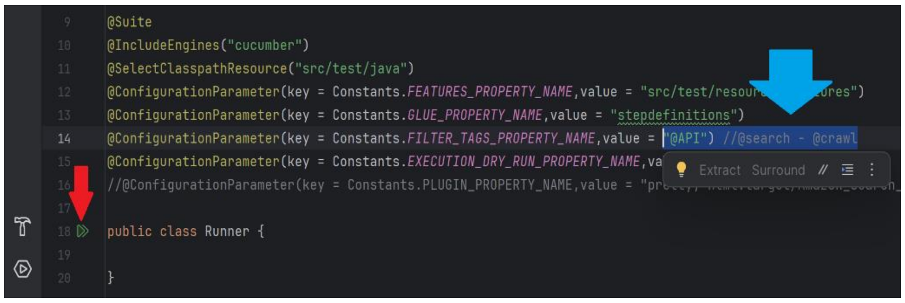
Maven project structure has been changed. Sync changes with IntelliJ IDEA to make it work correctly.

2- Running Tests with the Runner Class

As shown in the image below, tests can be executed by clicking on the red arrow. This triggers the execution of the Runner class.

To ensure that each feature file runs individually rather than executing tests in parallel, you can specify the desired feature's tag in the location marked with a blue arrow. Replace the tag with the one corresponding to the feature you wish to execute. For example, you can use @API or other relevant tags defined in your .feature files.

This approach allows for more precise control over the test execution, ensuring that only the intended scenarios are run at a time.



```
@Suite
@IncludeEngines("cucumber")
@SelectClasspathResource("src/test/java")
@ConfigurationParameter(key = Constants.FEATURES_PROPERTY_NAME,value = "src/test/resour     ures")
@ConfigurationParameter(key = Constants.GLUE_PROPERTY_NAME,value = "stepdefinitions")
@ConfigurationParameter(key = Constants.FILTER_TAGS_PROPERTY_NAME,value = "@API") //@search - @crawl
@ConfigurationParameter(key = Constants.EXECUTION_DRY_RUN_PROPERTY_NAME,va
//@ConfigurationParameter(key = Constants.PLUGIN_PROPERTY_NAME,value = "pr

public class Runner {

}
```