|  |  |
| --- | --- |
| Department of Software Engineering  National University of Computer and Emerging Sciences,  Chiniot-Faisalabad Campus | A logo with blue letters and a star  Description automatically generated |

**SOFTWARE QUALITY ENGINEERING**

**Project Phase-02**

**INSTRUCTOR:** Mr. Fraz Zaheer Ghorsi

**Group Members:**

|  |  |  |
| --- | --- | --- |
| Name | Section | Roll No |
| Ibrahim Faisa­l | BSE-5B | 22F-3702 |
| Furqan Asghar | BSE-5B | 22F-3693 |

# UI Test Automation Framework Documentation

Table of Contents

[**Overview** : 3](#_Toc179668348)

[**System Requirements** : 4](#_Toc179668349)

[**Installation and Setup** : 4](#_Toc179668350)

[1. Environment Setup : 5](#_Toc179668351)

[2. Project Setup : 5](#_Toc179668353)

[3. Dependency Management : 5](#_Toc179668354)

[**Framework Structure :** 5](#_Toc179668355)

[Directory Layout 6](#_Toc179668356)

[Key Components 6](#_Toc179668357)

[**Writing Tests** : 7](#_Toc179668358)

[Feature Files 7](#_Toc179668359)

[Step Definitions 7](#_Toc179668360)

[**Executing Tests** : 7](#_Toc179668361)

[**Data Integration** : 8](#_Toc179668362)

[1. Excel 8](#_Toc179668363)

[2. Databases 8](#_Toc179668364)

[3. Redis 8](#_Toc179668365)

[**Reporting :** 8](#_Toc179668366)

[Allure Reports 8](#_Toc179668367)

[**Version Control and Code Management :** 8](#_Toc179668368)

[***GitHub Usage*** 9](#_Toc179668369)

[**Best Practices** : 9](#_Toc179668370)

[**Conclusion** : 10](#_Toc179668371)

**Overview:**

This documentation outlines the setup, usage, and management of a **UI Test Automation Framework** tailored for web applications. The framework integrates **Selenium WebDriver** for browser automation, **Cucumber** for behavior-driven development (BDD), **Allure** for reporting, and supports multiple data sources such as **Excel**, **Databases**, and **Redis** for data-driven testing.

The framework is designed to be scalable, maintainable, and easy to use, making it adaptable to various web applications and test cases.

## System Requirements:

To ensure the framework runs smoothly, the following software and tools are required:

* Java JDK: Version 11 or higher.
* Maven: Version 3.6 or higher for managing dependencies and building the project.
* Git: For source control management.
* IDE: A development environment such as IntelliJ IDEA or Eclipse with support for Java and Maven projects.
* Web Browsers: Chrome, Firefox, Edge, etc., along with their respective WebDriver binaries.
* Allure Command-line: For generating and viewing HTML-based test execution reports.

## Installation and Setup:

### 1. **Environment Setup:**

### Follow these steps to configure your environment for the UI Test Automation Framework:

* **Install Java JDK**: Download and install from [Oracle's official website](https://www.oracle.com/java/technologies/javase-downloads.html).
* **Install Maven**: Visit the [Apache Maven site](https://maven.apache.org/install.html) and follow the installation guide.
* **Install Git**: Download Git from [Git SCM](https://git-scm.com/downloads).
* **Setup IDE**: Install and configure your preferred IDE (e.g., **IntelliJ IDEA** or **Eclipse**) to recognize Maven and Java projects.

### 2. **Project Setup:**

* Clone the Repository: Use Git to clone the framework repository from GitHub.
* git clone https://github.com/ibrahim3702/SQE\_Project.git
* Import the Project into IDE: Open your IDE and import the project as a Maven project to ensure that all dependencies are managed.

### 3. **Dependency Management:**

To download and install all necessary dependencies specified in the pom.xml file, run the following Maven command in the project root directory: mvn clean install. This command will clean the project, compile the code, and download all required libraries.

## Framework Structure:

### **Directory Layout**

The framework follows a structured directory layout to ensure clarity and maintainability:

* src/main/java: Contains core framework classes, such as utilities, driver management, and helper classes.
* src/test/java: Houses all test-related files, including step definitions, test hooks, and test runners.
* src/test/resources: Contains resources such as Cucumber feature files, test data, and configuration files.

### **Key Components**

#### **Driver-Factory**

The **Driver-Factory** is responsible for managing WebDriver instances for different browsers like Chrome, Firefox, and Edge. It ensures thread safety for parallel test execution and manages WebDriver lifecycle events like initialization and teardown.

#### **Page Objects**

The **Page Object Model (POM)** organizes web pages into reusable classes that encapsulate UI elements and methods for interacting with the web application.

#### **Step Definitions**

These classes implement the Gherkin steps defined in feature files, providing the glue between the **Gherkin** scenarios and the actual browser automation logic.

#### **Hooks**

Hooks contain setup and teardown methods for the tests, such as initializing the WebDriver and closing it after the test execution.

#### **Cucumber Runner**

The **Cucumber Runner** class configures the test execution environment and integrates **Allure** for generating reports.

## Writing Tests:

### **Feature Files**

* Located in src/test/resources/features.
* Written in Gherkin language, feature files describe the behavior of the application in a human-readable format.
* Example of a Gherkin scenario:

*Feature: User Login Scenario: Successful login with valid credentials Given the user is on the login page*

*When the user enters valid credentials*

*Then the user is redirected to the dashboard*

### **Step Definitions**

* Implement the Gherkin steps in Java and interact with web pages using Page Objects.
* Example of a step definition:

*@Given("the user is on the login page")*

*public void navigateToLoginPage() {*

*LoginPage loginPage = new LoginPage(driver);*

*loginPage.navigateTo();*

*}*

## ****Executing Tests:****

Tests can be executed using Maven. To run all tests, use the following command:

* + mvn test

This will trigger the Cucumber test execution process and generate reports.

## Data Integration:

### **1. Excel**

The framework integrates with **Apache POI** to read and write data from Excel files. This is particularly useful for data-driven tests where test data varies across different runs.

### **2. Databases**

Using **JDBC**, the framework can connect to various relational databases (e.g., MySQL, PostgreSQL) to fetch and manipulate test data.

### **3. Redis**

**Jedis** is used to interact with Redis, a fast in-memory data store. This is useful for scenarios where the application interacts with cached data.

## Reporting:

### **Allure Reports**

The framework integrates with **Allure** for generating visually appealing HTML-based reports. Allure captures detailed logs, test steps, and screenshots (if applicable) to enhance report visibility.

## Version Control and Code Management:

### ***GitHub Usage***

The framework is hosted on GitHub for version control. All development activities, including task tracking and code management, should adhere to the following guidelines:

### ***Repository***

https://github.com/ibrahim3702/SQE\_Project.git

#### **Branching Strategy**

* **Main Branch**: The primary, production-ready branch.
* **Feature Branches**: Each new feature or bug fix is developed in a separate branch.
* **Pull Requests**: Feature branches are merged back into main via pull requests, which should be reviewed by peers to ensure code quality.

#### **Issue Tracking**

* GitHub Issues are used to track tasks, enhancements, and bugs. Each task should have a corresponding issue, and developers should assign themselves issues they are working on.

#### **GitHub Workflow**

**Workflow Explanation**

* **Trigger**: This workflow is triggered on push or pull requests to the main branch.
* **Environment**: It runs on an Ubuntu latest environment.
* **Steps**:
  + **Checkout**: Checks out your repository under $GITHUB\_WORKSPACE, so your workflow can access it.
  + **Setup JDK 11**: Sets up a Java development environment with JDK 11 using the setup-java action.
  + **Cache Maven packages**: Caches the Maven dependencies to speed up future builds.
  + **Build with Maven**: Executes the Maven build command and packages your application.
  + **Run tests**: Executes the test suite using Maven.
  + **Deploy**: A placeholder for deployment commands or scripts.

**Extend and Customize**

* **Advanced Testing**: Include steps that run integration tests or UI tests if you have them.
* **Deployment**: If you have a deployment process (e.g., to a cloud provider or a testing environment), you can integrate those commands into the workflow.
* **Notifications**: Set up notifications (e.g., via email or Slack) if the build fails or succeeds.

## ****Best Practices:****

* **Code Reviews**: Mandatory for all pull requests to ensure that bugs are caught early and code quality is maintained.
* **Continuous Integration**: Integrate the framework with CI tools like **Jenkins** or **GitHub Actions** to automate the process of building, testing, and deploying the code.
* **Reusable Components**: Design page objects, utilities, and helpers for maximum reusability to keep the test code DRY (Don't Repeat Yourself).

## Conclusion:

This documentation provides a detailed guide for setting up, using, and managing the UI Test Automation Framework. The framework is designed to be modular, scalable, and maintainable, enabling seamless collaboration among team members. By following the best practices outlined in this document, the framework will continue to grow and evolve to support more complex test scenarios and different testing needs.