Assignment02

SQE

22F-3738/22F-3661

Test-Ops UI Automation Testing Framework

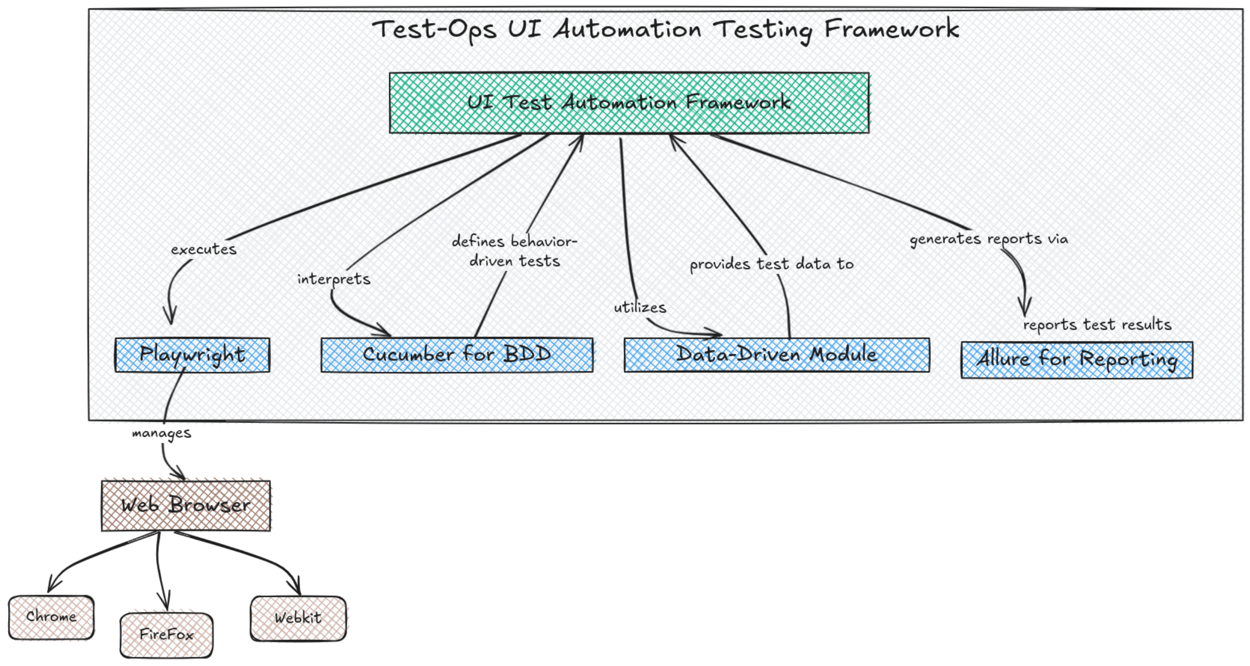
Introduction

The Test-Ops UI Automation Testing Framework is a comprehensive, robust solution designed to automate user interface testing for web applications. This document provides a detailed overview of the framework, its components, how it operates, the tools integrated within it, and the dependencies it relies on. The framework aims to streamline the testing process, improve accuracy, and reduce the time and effort required for testing web interfaces.

Framework Overview

This framework utilizes modern tools and technologies to provide a seamless experience in automating UI tests. It supports behavior-driven development (BDD) approaches, integrates with continuous integration (CI) pipelines, and offers detailed reporting features.

This screenshot is showing the architectural layout of the framework, illustrating how different components interact:



Core Components

1. Test Runner:

The primary component that orchestrates the execution of tests, managing the sequence in which tests are run, handling setup and teardown processes, and collecting results.

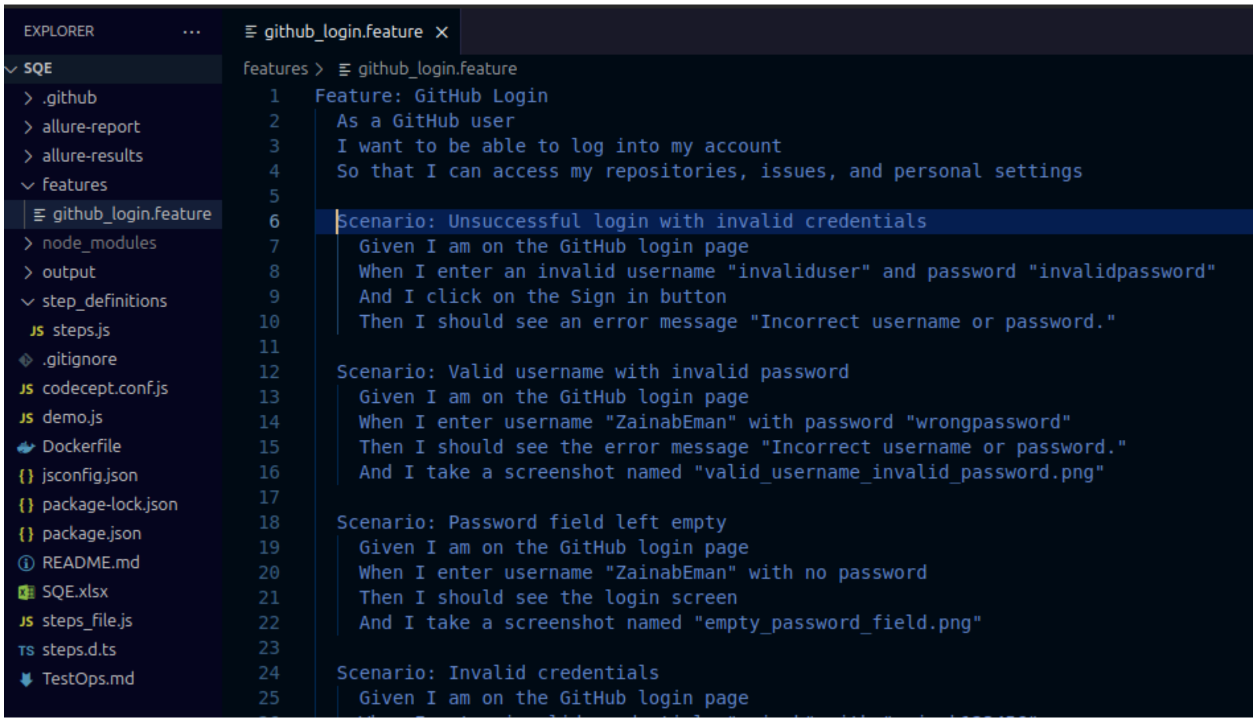
2. Page Object Model (POM):

A design pattern used to enhance test maintenance and reduce code duplication. Each page within the application is represented by a class that encapsulates all functionalities of the page.

3. Data-Driven Testing Module:

Enables tests to be executed with various sets of data, facilitating extensive coverage through multiple scenarios.

Here is the look how code is arranged



```
EXPLORER
SQE
 > .github
 > allure-report
 > allure-results
 v features
    github_login.feature
 > node_modules
 > output
 v step_definitions
    steps.js
 .gitignore
 codecept.conf.js
 demo.js
 Dockerfile
 jsconfig.json
 package-lock.json
 package.json
 README.md
 SQE.xlsx
 steps_file.js
 steps.d.ts
 TestOps.md
```

```
features > github_login.feature
Feature: GitHub Login
  As a GitHub user
  I want to be able to log into my account
  So that I can access my repositories, issues, and personal settings

  Scenario: Unsuccessful login with invalid credentials
    Given I am on the GitHub login page
    When I enter an invalid username "invaliduser" and password "invalidpassword"
    And I click on the Sign in button
    Then I should see an error message "Incorrect username or password."

  Scenario: Valid username with invalid password
    Given I am on the GitHub login page
    When I enter username "ZainabEman" with password "wrongpassword"
    Then I should see the error message "Incorrect username or password."
    And I take a screenshot named "valid_username_invalid_password.png"

  Scenario: Password field left empty
    Given I am on the GitHub login page
    When I enter username "ZainabEman" with no password
    Then I should see the login screen
    And I take a screenshot named "empty_password_field.png"

  Scenario: Invalid credentials
    Given I am on the GitHub login page
```

Tools Integrated

1. **Playwright** :Automates browser interactions, directly controlling browsers as a user would. It supports all major browsers and platforms.
2. **Cucumber**: Facilitates BDD by allowing test specifications to be written in plain English, which are then executed by the framework.
3. **Allure**: Generates rich, interactive reports that provide insights into test execution and results.
4. **Docker:** Used to containerize the environment, ensuring consistent, reproducible setups across different systems and CI/CD pipelines.

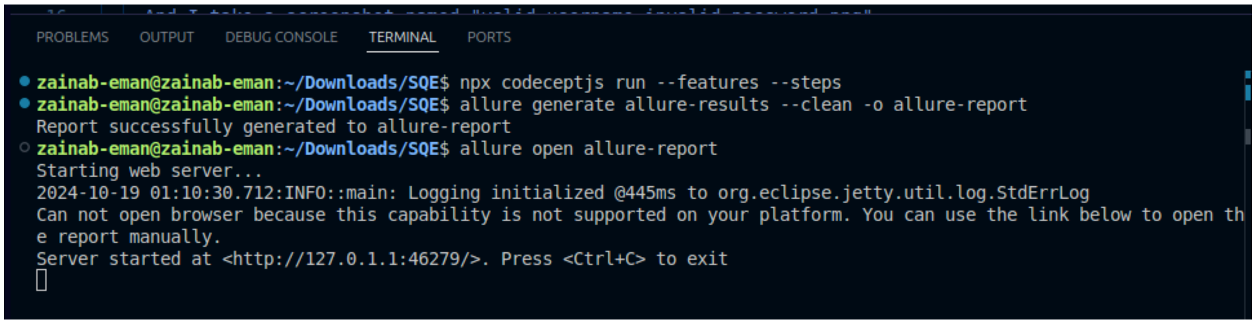
How It Works

Test Preparation

Test Cases Writing: Utilizing the BDD approach, testers write scenarios in Gherkin syntax that describe the expected behavior of the application.
Page Objects Creation: Developers write page objects for each page that needs to be tested, implementing the functionalities as methods that can be easily called by the test scripts.

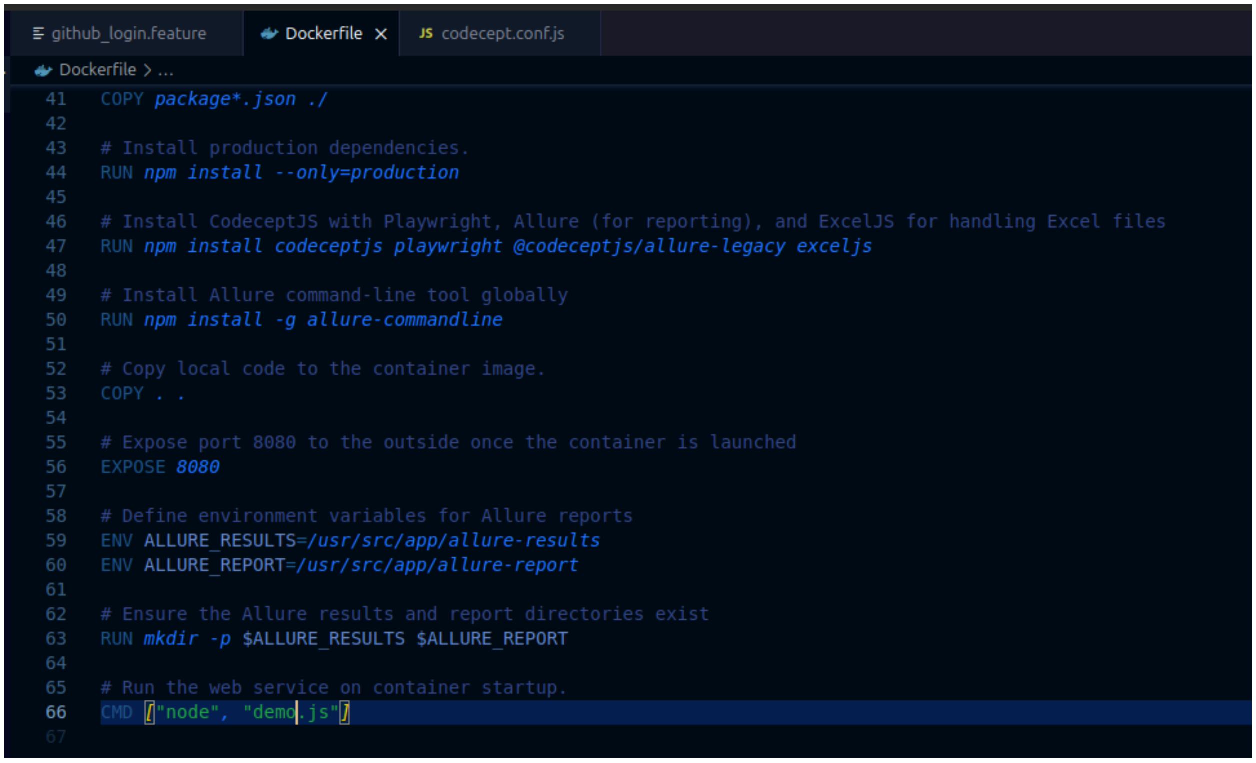
Execution Flow

Environment Setup: Docker containers are set up with all the necessary dependencies and configurations.
Test Execution: The test runner picks up the feature files, interprets them, and executes corresponding Selenium commands against the browser.
Result Collection: Results are gathered and processed to generate reports.



```
zainab-eman@zainab-eman:~/Downloads/SQE$ npx codeceptjs run --features --steps
zainab-eman@zainab-eman:~/Downloads/SQE$ allure generate allure-results --clean -o allure-report
Report successfully generated to allure-report
zainab-eman@zainab-eman:~/Downloads/SQE$ allure open allure-report
Starting web server...
2024-10-19 01:10:30.712:INFO::main: Logging initialized @445ms to org.eclipse.jetty.util.log.StdErrLog
Can not open browser because this capability is not supported on your platform. You can use the link below to open th
e report manually.
Server started at <http://127.0.1.1:46279/>. Press <Ctrl+C> to exit
```

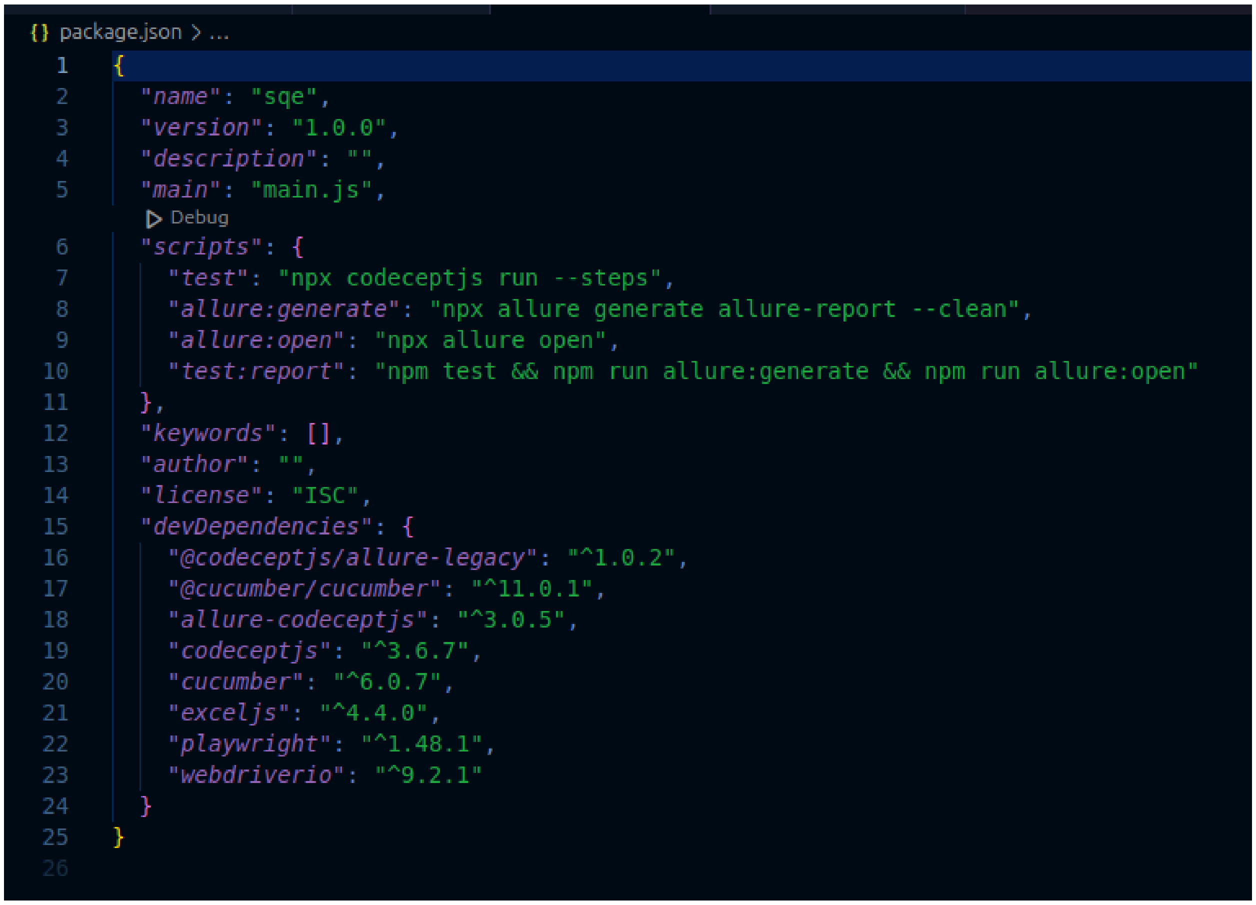
Docker Configuration: Screenshot of the Docker configuration files or commands used to set up the environment.



```
COPY package*.json ./

# Install production dependencies.
RUN npm install --only=production

# Install CodeceptJS with Playwright, Allure (for reporting), and ExcelJS for handling Excel files
RUN npm install codeceptjs playwright @codeceptjs/allure-legacy exceljs

# Install Allure command-line tool globally
RUN npm install -g allure-commandline

# Copy local code to the container image.
COPY . .

# Expose port 8080 to the outside once the container is launched
EXPOSE 8080

# Define environment variables for Allure reports
ENV ALLURE_RESULTS=/usr/src/app/allure-results
ENV ALLURE_REPORT=/usr/src/app/allure-report

# Ensure the Allure results and report directories exist
RUN mkdir -p $ALLURE_RESULTS $ALLURE_REPORT

# Run the web service on container startup.
CMD ["node", "demo.js"]
```

Dependencies

Node.js: Provides the runtime environment for running JavaScript outside the browser.
Java: Required for running Allure to create report based on web.
Browser Drivers: Playwright to run test cases on chrome , firefox and web kit



{} package.json > ...

```json
{
  "name": "sqe",
  "version": "1.0.0",
  "description": "",
  "main": "main.js",
  "scripts": {
    "test": "npx codeceptjs run --steps",
    "allure:generate": "npx allure generate allure-report --clean",
    "allure:open": "npx allure open",
    "test:report": "npm test && npm run allure:generate && npm run allure:open"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "devDependencies": {
    "@codeceptjs/allure-legacy": "^1.0.2",
    "@cucumber/cucumber": "^11.0.1",
    "allure-codeceptjs": "^3.0.5",
    "codeceptjs": "^3.6.7",
    "cucumber": "^6.0.7",
    "exceljs": "^4.4.0",
    "playwright": "^1.48.1",
    "webdriverio": "^9.2.1"
  }
}
```

Integration with CI/CD

The framework integrates seamlessly with CI/CD pipelines, enabling automatic triggering of tests upon code commits, merges, or as scheduled.

Jenkins: Popular tool for implementing CI/CD, which can orchestrate and manage the testing process.
GitHub Actions: An alternative for managing CI/CD directly from within GitHub repositories.

Conclusion

The Test-Ops UI Automation Testing Framework offers a powerful, flexible solution for automating UI testing across various platforms and environments. By leveraging leading tools and practices, it ensures thorough testing coverage, efficient error detection, and supports rapid application development cycles.