Selenium Hybrid Automation Framework - Project Guide

1. Project Overview

This document outlines a structured Selenium Hybrid Automation Framework designed for testing web applications.
It includes features such as object-oriented page classes, reusable common methods, HTML reporting, logging, and screenshot capture for failure tracking.

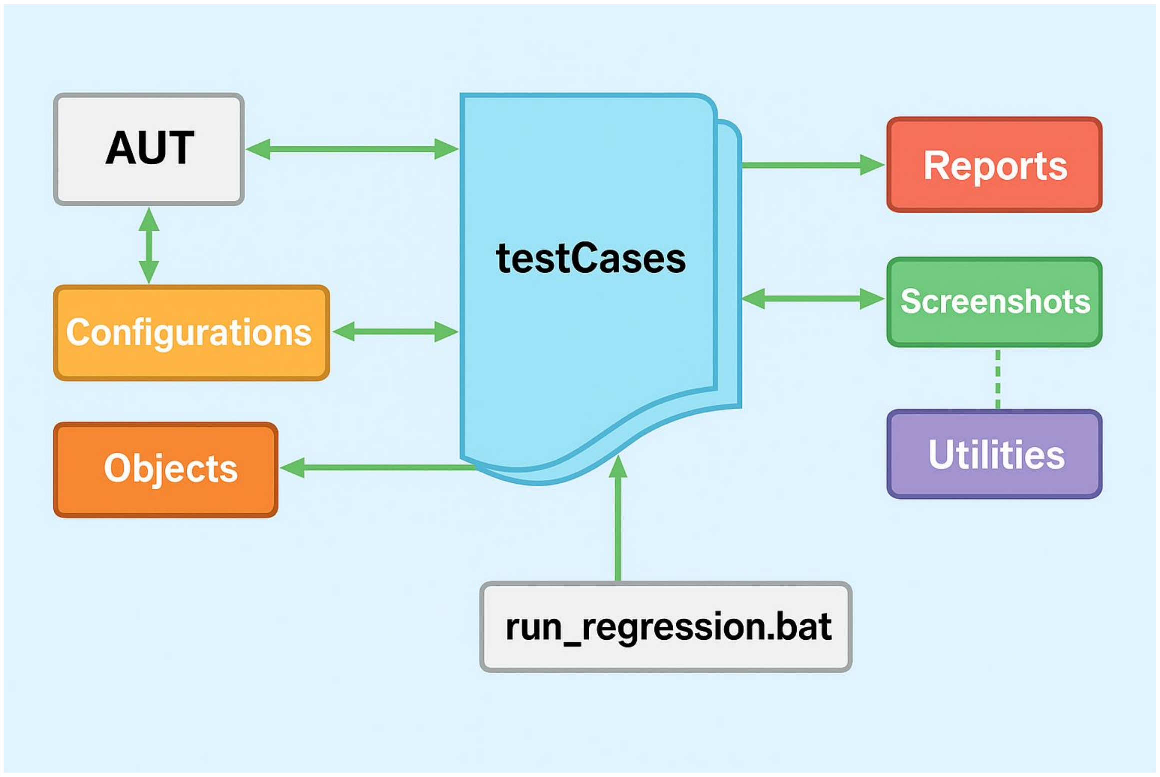
2. Folder Structure and Purpose

Below is the project structure:

```
Demo_Online_Shopping/
├── Configurations/
│   ├── customLogger.py
│   ├── settings.xml
├── Pages/
│   ├── universal.py
│   ├── demoshop.py
├── Objects/
│   ├── objectlocators.ini
├── testCases/
│   ├── conftest.py
│   ├── test_invalid_login.py
│   ├── test_valid_login.py
│
├── Reports/
│   ├── Detailed_Log/
│   ├── Summary_report/
├── Screenshots/
│   ├── test_valid_login.png
├── Utilities/- Future Implementations for sending email reporting
└── Run_regression.bat
```

3. Framework Flow Diagram

Below is a high-level diagram of the framework's flow:



Configurations

This folder contains the settings.xml file which holds critical configuration data such as <Environment>, <Release>, and <UAT> URL values. These are dynamically read during test execution to determine the environment and application under test.

Pages

This contains page-specific classes like demoshop.py, where business-level actions (e.g., login, search, wishlist) are defined. It also includes universal.py, which holds shared utility methods like launch_application(), init_driver(), and capture_screenshot() that are reused across multiple test scripts.

Objects

Includes objectlocators.ini, which stores the UI locators (XPaths, CSS selectors) grouped by logical page sections. These locators are dynamically retrieved by get_object_locator() method in universal.py and used in tests for robust and maintainable element access.

testCases

This folder houses all the individual test scripts (e.g., test_login.py, test_search.py). Each test imports reusable methods from the Pages module and is triggered using the run_regression.bat file. The conftest.py file inside this folder sets up test-level metadata, including the environment name, release version, and tester name, which are reflected in the final HTML report.

Reports

Contains two subfolders:

- Detailed_Log/: Saves log files for each executed test case, making it easy to trace failures and understand execution flow.
- Summary_report/: Contains the consolidated summary_report.html, which shows the total tests run, passed, and failed in a well-formatted HTML layout.

Screenshots

Screenshots of failed steps are automatically captured and saved here, organized by date and test script name, aiding in visual debugging.

Utilities

Currently reserved for enhancements like sending summary reports via email. It may later include modules like email_reporter.py to notify stakeholders post-execution.

Run_regression.bat

The run_regression.bat is a Windows batch file that serves as a single-click launcher. When executed, it runs all test scripts under the testCases/ directory using the pytest framework. The command includes HTML reporting options that automatically generate a detailed and a summary report upon completion.

4. How to Run Test Cases

To run all test cases: Run_regression.bat

It is windows executable bat file which contains below command to execute all testcases.py files present in testCases folder when we double click the bat file.

```
    pytest -v -s testCases/ --html=Reports/Summary/summary_report.html --self-contained-html --capture=tee-sys
```

To run a specific test case:

```
    pytest -v -s testCases/test_invalid_login.py --html=Reports/final_report.html --self-contained-html --capture=tee-sys
```

5. Running in Parallel

To execute tests in parallel using 3 processes:

```
pytest -n 3 -v -s testCases/ --html=Reports/final_report.html --self-contained-html --capture=tee-sys
```

This modular and layered structure ensures scalability, maintainability, and ease of debugging. Each folder communicates seamlessly with others—for instance, test scripts call reusable methods from Pages, fetch locators from Objects, and use configuration data from Configurations, with all outcomes logged in Reports and error snapshots saved in Screenshots.