

Dissertation Title:
Characteristic of Good Test Automation Framework
& its Design

Course No: SS ZG628T
Course Title: Dissertation

Dissertation Work Done by:

Student Name: SAQUIB

BITS ID: 2021MT12266

Degree Program: Master's in software system

Research Area: Test Automation Framework

Dissertation Work carried out at:

Chegg India Pvt Limited, Delhi



**BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE,
PILANI
VIDYA VIHAR, PILANI, RAJASTHAN - 333031.
March 2023**

Dissertation Title:
Characteristic of Good Test Automation
Framework & its Design

Course No: SS ZG628T
Course Title: Dissertation

Dissertation Work Done by:

Student Name: SAQUIB

BITS ID: 2021MT12266

Degree Program: Master's in software system

Research Area: Test Automation Framework

Dissertation Work carried out at:

Chegg India Pvt Limited, Delhi



**BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE,
PILANI**

VIDYA VIHAR, PILANI, RAJASTHAN - 333031.

March 2023



Contents

1. Abstract.....	3
2. Background Research & Literature Review	4
3. Tool & Technology selected for Automation Framework.....	6
4. Introduction to Software Test Automation Framework (STAF)	9
5. Software Test Automation Low Level Design	10
6. Overview of the Architecture Design	12
7. Proposed Project Design	13
8. Use Case for Test Automation Framework	16
9. Detailed Steps for Framework Implementation (Coding)	17
10. Test Automation Framework Output Result & Metric	26
11. Plan of Pending Work and Timeline.....	28

Abstract

Test Automation framework design is always challenging and complex task, As the software industry growing steadily day by, each service require end to end testing either front end backend or integration functionalities, in turn the testing of these area also become laborious task. As demand grows, rapid development and advancement of tool and technology evolve in the market to fulfill automation framework demand based on the business and requirement needs, most of the readily available framework are not appropriate to use for innumerable business and project needs because some only provide record and playback, other provide only inbuilt keywords to use, rest provide support for only one level of testing. A good automation framework should be modular, reusable, consistent, logging, reporting facility. But most of the open-source tools and libraries available do not provide all the features outside of the box.

Therefore, we have decided to design in-house framework which fulfill all the company requirement needs and have good reporting and logging features along with the basic framework functionality. In this dissertation we are going to propose a framework design called Software Test Automation Framework (STAF). Framework design incorporating best practices, hybrid design page object model implementations to automate the software services different feature. Framework's goal is to lower the overall efforts of the manual testing by automating most of the test cases process and increase the testing coverage.

	
SAQUIB(2021MT2266)	Srihari Naidu (Supervisor)
Date: 20 / 03 / 2023	Date: 20 / 03 / 2023

Background Research and Literature Review.

Software testing is a crucial component of the software development process that guarantees the dependability and quality of software applications. Software testing techniques known as automation testing use scripts and automation tools to run test cases. Many benefits of automation testing include quicker testing, increased accuracy, and lower testing expenses. The creation of automation testing scripts, however, might take some time and involves knowledge of a variety of programming languages and automation technologies. Hence, a scalable and flexible automation testing framework is needed for software developers and testers to make it easier to create and maintain automation testing scripts.

There are multiple approach one can choose to implement the testing framework based on organization need and budget constraints. Generally, in literature there are several types of frameworks used in testing. The Automation Framework are organised and structured automated test, which increases their effectiveness and makes maintenance easier. Frameworks for automation come in a variety of forms, each with a unique set of benefits and drawbacks.

Among the most popular categories of automation frameworks are the following:

- **Linear Automation Framework:** This simple type of framework, test script are written and executed one after other there is no complex logic and dependent file structure used. Although it is very easy to create this type of framework but difficult to maintain and debug the fail script.
- **Modular Automation Framework:** In this type of framework reusable function and methods are created which can be used in multiple test script, it's easier to maintain and good for mid and small size project.
- **Data Driven Automation Framework:** This framework especially designed to read test data from csv, excel or text file and run test case with different data sets. Hence "data set" and the "test case" themselves are separated by a "data driven framework" (code). Since the test case and data set are independent, it is simple to update the test case for a certain functionality without changing the code. This help in reduce the number of test case written to test the same feature repeatedly with multiple sets of data, so improve maintainability.
- **Keyword Driven Automation Framework:** This type of framework defines keyword and actions to call and reused to test general functionality and these keyword used in the test script without writing all the end to end script just write core logic and used the already created keyword and action, In other words the Keyword-Driven framework is a method for externalising the script's keywords and actions into a different Object Repository, which has the advantages of greater code reuse, less frequent script updating, and greater portability.it is easier to maintain but requires significant effort to write the keyword and action library.
- **Hybrid Automation Framework:** This framework combines the features of multiple frameworks like keyword driven, data driven, having lots of functionality like flexibility, reusability, and maintainability. It provides flexible and scalable approach to automate the test scripts easily.
- **Behaviour Driven Development Automation Framework:** A software development methodology called Behaviour Driven Development (BDD) allows the tester or business analyst to write test cases in plain text (English).Even non-technical team members may understand the scenarios' straightforward language and how it applies to the software project. Communication between technical and non-technical teams, managers, and stakeholders is aided and improved as a result, It encourage the collaboration between technical and non-technical members.
- **Test Driven Development Automation Framework:** is a methodology for developing software that uses test cases to define and confirm what the code will accomplish. Simply put, test cases are developed and tested for each capability first, and if the test fails, new code is produced to pass the test and provide simple, bug-free code.

There are numerous frameworks available in market with different programming language, If we compare all the marketing language, python is easy to learn and write code for automation script. In the company some of the script already written for automation in python, In testing the company using python language, Hence we decided to develop python selenium based Hybrid Automation framework.

Python is a popular programming language used in software development and automated testing. Python offers several benefits, including the support for numerous libraries and frameworks and the ability to construct test scripts in a straightforward and basic language. Selenium is a well-liked automation tool for testing that supports a variety of platforms and browsers. Robot Framework, PyTest, and Behave are just a few of the hybrid Selenium and Python frameworks that have been created. These frameworks offer several benefits, including making it simple to write and maintain automation testing scripts and supporting a variety of platforms and browsers However lack in ability to integrate with in house tool, not able to handle complex scenario, interfacing with other programming frameworks not possible. It is therefore required to create a customized hybrid framework based on Python and Selenium that addresses these restrictions.

The goal of this project is to create a hybrid framework using Python and Selenium to increase the speed and efficacy of automation testing. The framework intends to overcome some of the drawbacks of current automation practices, such as the lack of support for many browsers, good reporting, cumbersome debugging or the complexity of updating and maintaining test scripts.

We will implement an automation testing framework that combines the benefits of the Python programming language, and the Selenium automation tool is known as a STAF (Software Test Automation Framework) hybrid framework design based on Python and Selenium. Developers and testers can simply construct and manage automation testing scripts because to the framework's flexible and scalable method to testing.

The main objective of Dissertation project is to achieve the following results:

- Design page Object Mode (POM) based selenium Python automation framework
- Automate few basic scenarios like Login, Logout, HomePageTitleVerification.
- Framework should be able to run Data Driven Test Cases (eg: Login with multiple data sets.)
- Framework print logs to each action or important steps also save the log file log folder.
- Framework support screenshot capture in case of failure and save the screenshot in screenshot folder.
- Support reporting tool to generate summary of all test cases pass/fail after the test suite run.
- Integrate with build tool to run the automated and schedule job based on merge or check in trigger.
- Fine tune the framework flexibility to integrate multiple features and create reusable keywords.

Results:

- Generate Pytest and Allure report indicating pass and fail status of the test and feature.
- Integrate with CI/CD git lab pipeline, run the pipeline and attach the console output.
- Develop the allure report with rich detailed graphical visualization of test suite run.
- Git Lab yml file developed for scheduling and running pipeline and test suite.

Tools and Technology chosen for the Automation framework.

Python is very popular language frequently used in testing and automation support wider range of libraries to integrate with and allow to integrate with 3rd party tools, The following are the some benefit of choosing python over other programming language in test automation framework.

- **Easy to use:** Python is a beginner-friendly language that is simple to learn. It's a great option for beginning programmers thanks to its straightforward syntax and readability. It is easy to grasp syntax and fundamentals of the language since it allows working professional to quickly pick up the language's principles and begin developing frameworks for testing automation.
- **Huge Developer Community:** Python has a sizable developer community that supports its growth and upkeep. This community makes it simple for students to acquire support and advice when learning the language by giving them access to a plethora of resources, such as documentation, lessons, and online discussion boards.
- **Support for a wide range of libraries:** Python has a huge collection of modules and packages, including those designed specifically for testing automation, including Pytest, Selenium, and Robot Framework. With just a little bit of coding, these libraries make it simple for students to create frameworks for testing automation.
- **Cross-Platform Compatibility:** Python may be used on many operating systems, such as Windows, macOS, and Linux, thanks to its cross-platform compatibility. With this function, students can work on any platform they feel comfortable with, which improves their learning.
- **Integration with Other Tools:** To build reliable testing automation frameworks, Python is easily integrated with other tools like Selenium WebDriver, pytest, and Robot Framework.

In conclusion, Python is a popular choice for testing automation frameworks due to its simplicity, robust libraries, and cross-platform compatibility. It is a flexible language for automated testing due to its compatibility with other tools and support for different testing kinds.

Along with the above reason most of the company product used python backend developed in python code which used python libraries and modules it would be easy to integrate the testing framework in the same language, we can also use some inbuilt library there for some complex project feature automation.

Pytest: The testing framework for Python known as Pytest is strong, well-liked, and frequently used in the testing world for test automation. Some of the explanations for Pytest's suitability as an automation framework and its widespread use in the IT sector include the following:

- **Simple and readable Syntax:** Pytest is suitable option for test automation because of its clear and intuitive syntax, which is simple to learn and interpret. Test cases are easier to maintain since the syntax is simple to construct, even for individuals who are new to programming. It is also simple to read and understand.
- **Extensive Feature Support:** A robust fixture framework in Pytest makes it possible for developers to define and manage test fixtures. This feature helps make test cases more dependable and robust by setting up preconditions for testing and cleaning them up afterward.
- **Supports Test Parametrization:** Pytest includes a parametrization feature that enables tester to create test cases that are more adaptable and cover a wider range of test circumstances. The amount of code required to cover a variety of test cases is decreased, like running test case on data with multiple sets of input and corresponding output.

- **Integration with other Tools:** Docker, Jenkins, Selenium WebDriver, and other Python tools and frameworks are just a few that Pytest can readily interact with. With the use of this capability, test automation engineers may create elaborate, reliable, and simple-to-maintain test automation systems.
- **Huge Community Support:** A sizable developer community supports the creation and upkeep of Pytest. Support, information, and several plugins are offered by this community to increase Pytest's capability. When needed, this feature makes it simple for test automation engineers to obtain resources and assistance.
- **Pytest includes extensive test reporting** that offers comprehensive data about test runs, including test durations, failed tests, and test coverage reports. This feature facilitates rapid issue diagnosis and resolution for developers, resulting in quicker feedback cycles.

In conclusion, Pytest is a best option for test automation because of its easy-to-read syntax, wide fixture support, test parametrization, integration with other tools, big community support, and sophisticated test reporting. Because to its prevalence within the sector, it also has a sizable and active development community, making it simpler to locate tools and support when required.

Selenium: Python and Selenium together offer various benefits for test automation in the industry. Selenium is a well-liked open-source automation testing solution for online applications. In the business world, utilizing Python and Selenium has a number of benefits, including the following:

- **Cross Platform Compatibility:** Selenium is cross-platform compatible, meaning it can test web applications on operating systems like Windows, Linux, and Mac OS. Python is a robust and adaptable tool for cross-browser testing since it is a cross-platform language that makes it simple to execute Selenium tests on several systems.
- **Simple to Learn and Use:** Selenium provide number of keyword to interact with browser a very easy integrated API hence to use with Python and Selenium also offers a user-friendly API for automating web applications, making it simpler to create and maintain test cases.
- **Supports a Variety of Browsers:** Selenium with Python allows for the automation of tests on a variety of browsers, including Chrome, Firefox, Safari, and Internet Explorer. This function makes sure that web applications are tested across a variety of browsers, enhancing compatibility and dependability.
- **Huge and Active Developer Community:** The Selenium with Python framework has a sizable and active developer community that offers support, documentation, and a number of plugins to expand its capability. When needed, this feature makes it simple for test automation engineers to locate resources and assistance.
- **Integration with Other Tools:** Python-based Selenium may be readily integrated with other programmes like Robot Framework, Jenkins, and pytest, among others. Engineers that specialise in test automation can create simple-to-manage end-to-end test automation systems using this functionality.
- **Cost-effectiveness:** Because Selenium with Python is an open-source tool, it is free to use and share. Moreover, Python is a cost-effective option for test automation thanks to its ecosystem's abundance of open-source and free modules that may be utilised to create frameworks.

In conclusion, Selenium with Python is a well-liked option for test automation in the market due to its cross-platform compatibility, ease of learning and use, support for many browsers, significant community support, integration with other tools, and cost-effectiveness.

Pytest HTML and Allure Report: Along with the simplicity and reusability of Pytest testing framework it offers strong reporting features and lets testers create test cases with an easy-to-use syntax. The Pytest framework's Pytest allure report is an addition that creates thorough and approachable HTML reports. The following are some benefits and applications of Pytest allure report in the test automation framework:

- **User Friendly Report:** Pytest Allure Report provides reports that are simple to read and that give a thorough summary of the test results. The report presents test execution time, test failures, and test summary in an intuitive and comprehensible graphical format.
- **Customizable Report:** Pytest's alluring report lets users add their own unique information, labels, and descriptions to the report output. With the use of this functionality, testers can provide extra context for the test cases, enhancing the reports' informational value.
- **Integration with Test Automation Framework:** Pytest Allure Report is readily integrated with various test automation frameworks, such as Python with Selenium. With the help of this feature, test automation engineers may create thorough reports and learn more about test outcomes, which makes it simpler to spot problems and increase test coverage overall.
- **Continuous integration (CI) tool integration:** The Pytest Allure report may be integrated with CI gitlab tools like Jenkins and Travis CI, among others. With the help of this functionality, testers can see test results as they happen and act quickly if a test fails.
- **Historical Reports:** Pytest allure report gives historical reports that let testers to track test outcomes over time. With the use of this capability, testers can find trends and patterns in test results and decide on test coverage and test strategy with greater knowledge.

In conclusion, Pytest Allure Report has a number of benefits, including configurable reports, reports that are easy to use, reports that are integrated with CI systems and test automation frameworks, and historical reports. With the help of these features, Pytest allure report is a potent tool for test automation engineers to create thorough reports and get insight into test results, making it simpler to spot problems and increase test coverage overall.

Hence overall we are using open source tool and libraries to develop our test automation framework, along with out of box functionality tool like selenium, Lots of company using selenium libraries and keywords for automation which make this tool popular and 1st choice among automation tool. Usage of pytest and support community growing day by day dure vast variety of use of python in multiple domains and project.

At last company need to be fulfilled with less investment and time, so our best option to go for these stacks and design the framework based on selenium, Python, Pytest and Allure.

In the next chapter we will start into the design and implementation of framework that we are proposing for this dissertation.

Introduction to Software Test Automation Framework (STAF)

The term "software test automation" refers to the use of software to control the execution of tests, the comparison of actual results to predicted results, the creation of test pre-conditions, and other test control and test reporting activities. Test automation framework used for testing. It provides several essential features, such as logging and reporting, and it enables the expansion of its testing capabilities by including new test libraries. The environment in which automated tests will be produced, designed, and implemented may be broadly defined as an automated test framework, which also includes a collection of abstract concepts, methods, and procedures. It also includes the physical structures utilized for test formulation and implementation, as well as the logical connections between those components. An automated test framework is structured like a software program. A test automation framework, which acts like an application, defines common features like processing external files, GUI interface, and provides templates for test structures, so creating an automated solution is very similar to creating software applications.

The framework we are going to design here for the test automation can be used to test any type of web-based application or backend application(future-scope) just we need to generate the following data set and components.

Test script: Steps to be performed on the application contain expected and actual results.

Test data: Set of data like expected message, queries, invalid input need to test the functionality.

Locators: Identify the application object like button, search box, input filed, alerts ,links etc

Software testing is a crucial component of the software development process that guarantees the dependability and quality of software applications. Software testing techniques known as automation testing use scripts and automation tools to run test cases. Many benefits of automation testing include quicker testing, increased accuracy, and lower testing expenses.

The creation of automation testing scripts, however, might take some time and involves knowledge of a variety of programming languages and automation technologies. Hence, a scalable and flexible automation testing framework is needed for software developers and testers to make it easier to create and maintain automation testing scripts.

Along with that we can schedule the testing before each release, this framework helps in smooth transition from development to deployment stage of the product. Stakeholder can see the product health and gain confidence in the release cycle. Visibility of the project increase when Developer, stakeholder , business analyst and tester all are on the same page and understand the risk and challenge by just looking into Automation report dashboard page and decide there next plan related to product deployment.

Software Test Automation Low Level Design

This framework will have folder structures such as Locator's folder which contain the identification of each page object locators could be (xpath, id, css, class, tagName, linkText, partilaLinkText, name used for identification of button, text, input, search box, link and images etc) and used by test case script for writing the action and method functions. Test Data folder contains excel, csv or text file contain the static data or production like data which will be used for creating the precondition for the test cases (e.g. valid and invalid login credentials credit card info and other different test data).Test Case file files basically contains the action methods and keywords that will be used frequently across the test script (e.g. Login to the application, Log out functionality, Input text and Click on Search etc) as shown in Fig 1.

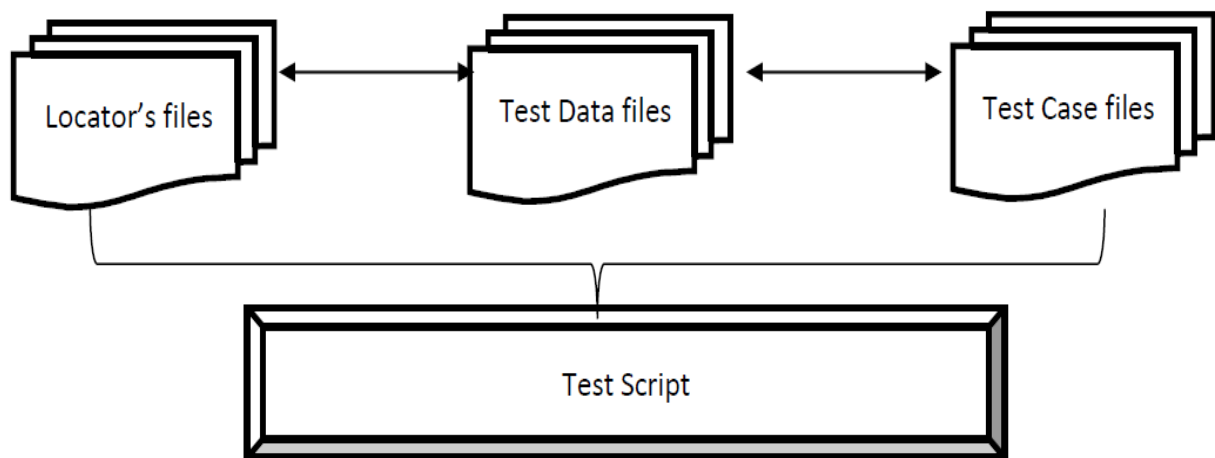


Fig 1. Proposed Low level Design.

Fig 2, Here we are using Page Object Design pattern to develop our framework. The main advantage of choosing POM is because of flexibility and reusable automation framework for multiple services functionalities. Not much technical expertise requires anyone to use this function with little programming knowledge.

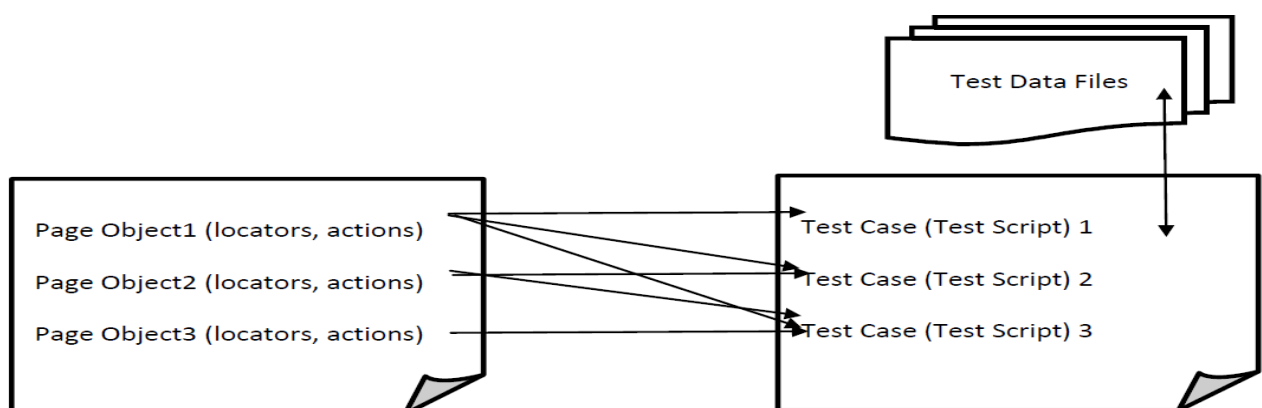


Fig 2. Page Object Model Design

Fig 3, All the Screen pages created as separate python file and we define actions class , object and methods inside that page only , when we need to write the test case for corresponding functionality just import the page model it will provide access of all the available function , In this way script looks clean and modular no need to write duplicate code and use the same code again and again hence reusable and modular framework design.

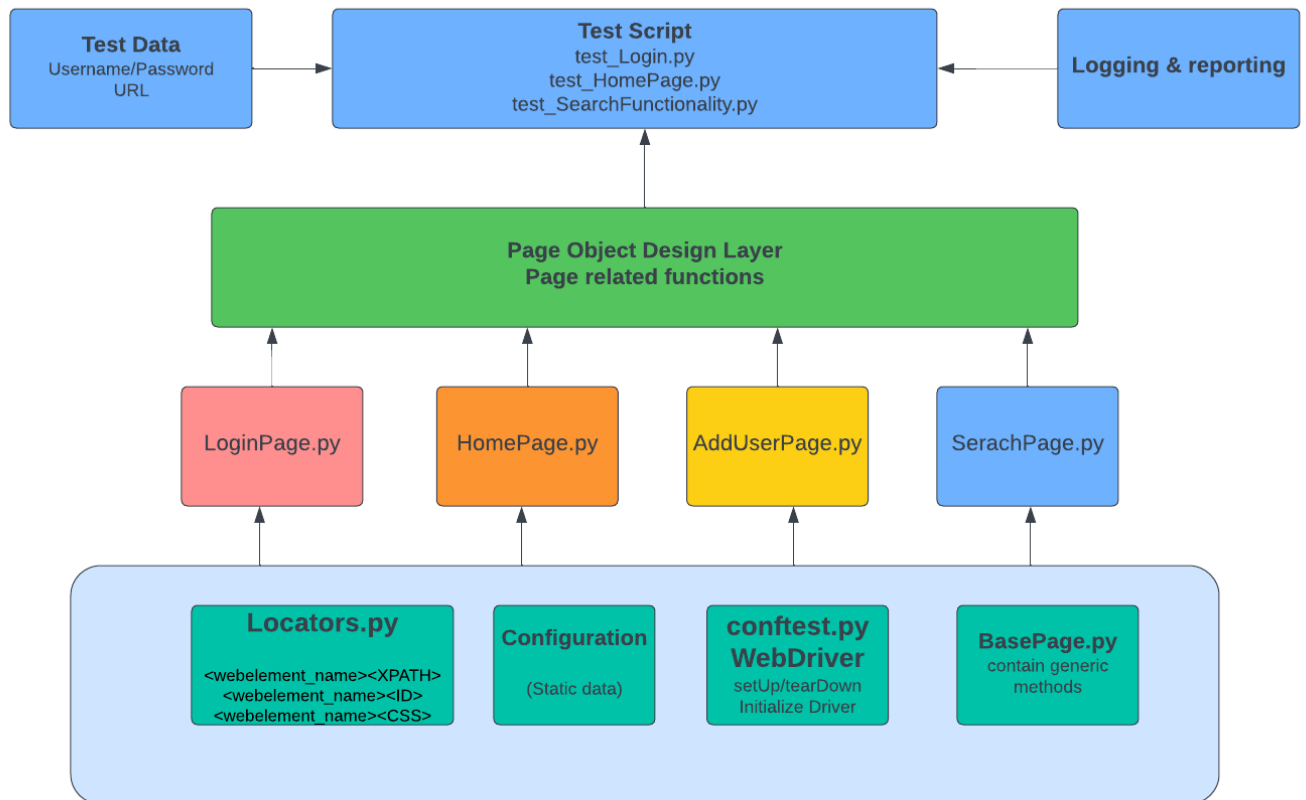


Fig 3. Page Object Model Detailed Design

Overview and Architecture Design

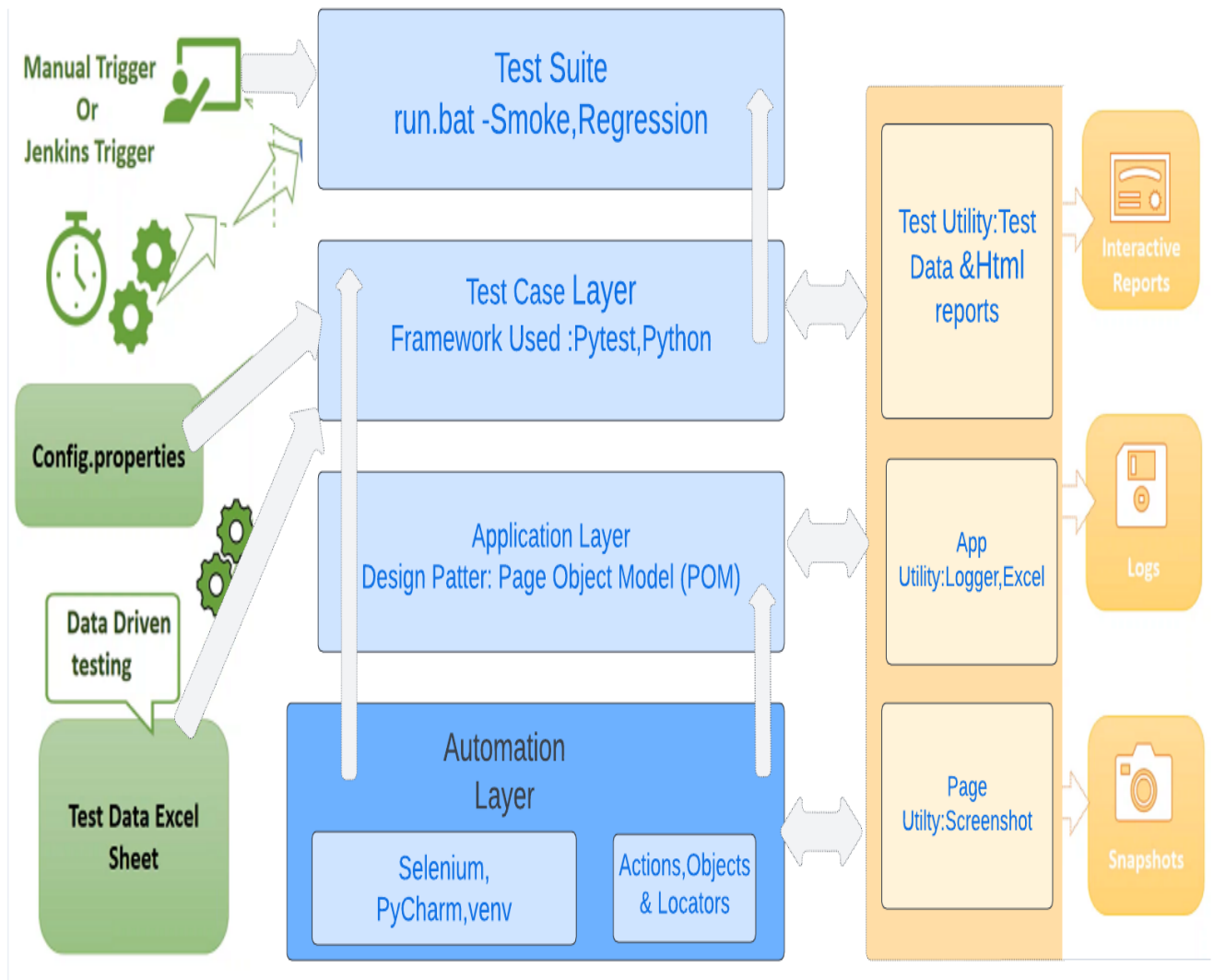


Fig 4. Test Automation Framework Architecture Diagram

Tools and Libraries Used in Framework Design

This framework will use open-source components like selenium library and python scripting, We will use Pycharm editor tool to create and manage all code and execute throughout the dissertation project.

Create a new Project in Pycharm (STAF Framework Design) and install the dependent library.

- *Selenium: Selenium Libraies*
- *Pyest: Python UnitTest Framework*
- *Pytest-html: Pyest Html Report*
- *Openpyxl: Run test parallel.*
- *Allure-pytest: to generate allure reports*

To begin with install PyCharm tool, create virtual environment, install all dependency using pip, create folder structure for (STAF framework Design) to build logical relation among the directory structure. Scope for this dissertation purpose we will use demo website for test cases automation and design complete framework for open-source service. Later will use this design to automate a company related application.

Create the folders structure in Pycharm IDE follow similar hierarchy.

Proposed Project Design

STAF-Python-Selenium

- | ***Configuration (Folder)***
- | ***Locators (Package)***
- | ***Logs (Folder)***
- | ***pageObjects (Package)***
- | ***Report (Folder)***
- | ***Screenshot (Folder)***
- | ***testCases (Package)***
- | ***testData (Folder)***
- | ***Utility (Package)***

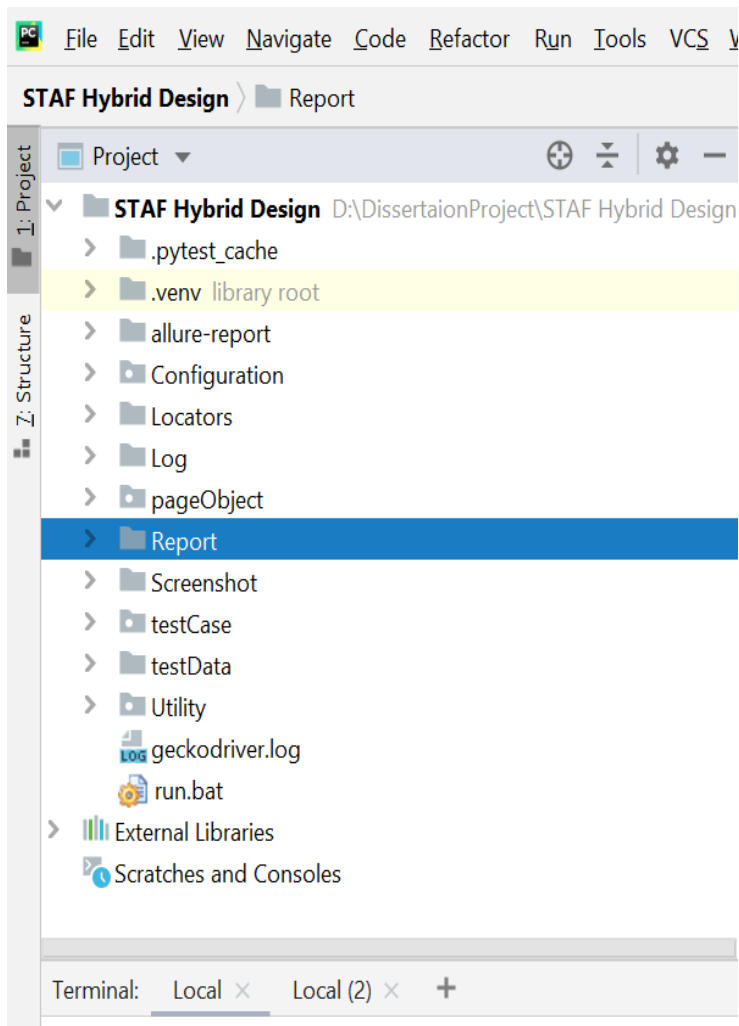


Fig 5. PyCharm IDE Folder Structure Hierarchy

Configuration:

For any test automation framework it is required that the static data used for test script should be placed in appropriate and easily accessible location, Data could be Application Launch URL, valid login credential, Login Page title, Dashboard Page title, Selenium driver executable path for different browser like chrome, Firefox, edge etc.

There are multiple ways we can manage this in Pytest framework, either we save the data in **configuration.ini** file and write a code to parse this file or just create **config.py** python file create a class in the file and add all the static data as a class variable and import it in test script whenever requires.

Hence, we have implemented both ways in our framework due to flexibility.

Locators:

Locators used to identify the web element to select HTML DOM to perform any action on the page element. The quality of test script depends on the accuracy of uniquely identified test element to interact with. Selenium offers multiple types of locators – ID, XPATH, Name, CSS, Class, PartialLinkText, LinkText.

We have created Locator folder which contains all the web element, the syntax used with naming convention in this way we are easily distinguish the page section by following naming convention.

Suppose (button_login_xpath= '(By.XPATH, "//*[@type='submit']")') means that this is login page button web element identified by xpath locators. Similarly in this way we have provided all the web element locators in the same file and it is used by the actions and methods wherever required by just importing it.

pageObject:

In test automation, the Page Object Model (POM) design pattern is frequently used to provide an object repository for web UI elements. The model's benefit is that it lessens code duplication and enhance test maintenance.

Under this model, a Page Class should exist for each web page in the application. This page's Web elements will be identified by this page class, which also has page methods that operate on those Web elements. The names of these techniques should reflect the function they provide.

To test the web application, we have to create one class per page for each screen, suppose we have to test Login page so create a pageObject class like LoginPage.py which contain all the locators, actions and keywords related to login screen defined in this class. Similarly, this page object class inherit Base class object in the parent class so that they can inherit basic method by default without re-writing it again and just we need to write new functions and methods that are specific to that page only.

Utility:

In test Automation Framework some time test scripts require to read data from excel or csv file, furthermore it might be necessary to define logging configuration and use it in the test script, In addition to that it also require to generate random data and used in the script, these all use cases can be handled by creating utility functions and methods, hence we created separate .py in the utility folder to read the csv or excel data from and used in the test script, also created customLogger.py to define the logger formatting and file handling properties.

Therefore, Utility folder is plays an important role, where we can write additional methods used in test script and it is reusable and modular, anytime we can add new method based on our need.

Logs:

No one can deny the logging and debugging importance while troubleshooting failed test cases. In good test Automation framework, it is must to have feature. The Automation Log gives comprehensive details on every test that was conducted, including all script routines, keyword tests, low-level functions, and so forth. It includes the outcomes of each test's activities as results. We have created Log folder which contains log file having naming convention current_date_time.log for each run of test cases. We are generating log file for each time when we execute the test suite irrespective of fail and pass status.

Screenshots:

There is one folder created in STAF framework to save screenshot of the application screens. It is very helpful in debugging the failed test cases, sometimes test case fail but it is difficult to identify at which point it failed because there are number of screen present in single test case flow. If somehow, we capture the failed screen, it will be very easy to debug the failure and script can be rectified easily. In addition to that help in quickly troubleshoot and modifying the script. Screenshot folder contains the screenshot for failed test cases only. Screenshot require significant storage space on disk to limit the usage we are just capturing screenshot in case of test case got failed, no need to take screenshot for pass and working scenario as of now.

testData:

We have created one folder called testData which will contain all the static data related to testing used in tests script, Tester required numerous raw data for testing in automation script, like valid login credentials, in valid login credentials, to register or sign up all the input form details like (Name, email, address, phone number etc)

Hence, we need to provide this information before hand, thus we created testData folder which contain excel, csv or text file which have these test data available to use in the script. Although this data cannot be used directly we need to write parser code to read csv or excel data and then used in the script. Sometimes JSON or xml data also required as a payload for backend API request.

Report:

Report folder is created to store the html unit test report for each run. Pytest generate report in html and json format, we are generating report in html format here, Test report contain the count of pass and fail test cases and if the test case fail it will display the failure reason, and debugging related information, Report also show the environment details like on which browser test case was run either chrome or firefox. This report is just one html file can be easily sharable across the team on we generate one link and provide access to the concerned stakeholders.

Another way to implement report is to use allure-pytest library this will generate nice intuitive report lot of descriptive details on allure server . For this first we have to run a command to generate allure-report folder having contains sets of files for allure report, then use this folder path to generate allure report by running allure server and provide this path to the server directory. This could be installed on centralized location and anyone able to access the report whenever needed. We will implement method for this test framework project.

Use Case for test Automation Framework.

In this section we will implement the actual framework using the Pycharm tool and write the code using selenium library.

For the dissertation purpose we have identified few experiments to begin with this implementation:

Experiment 1: Implement the validation of Login Functionality of Web Application

Experiment 2: Implement the validation of Data Driven Functionality

Experiment 3: Implement the validation of functionality to verify Home Page Title of Web Application

Experiment 4: Implement the validation of Add User functionality of Web Application

Experiment 5: Implement the validation of Register User functionality of Web Application

Experiment 6: Implement the validation of Search User By name functionality of Web Application

Experiment 5: Implement the validation of Search User By email of Web Application

Experiment 5: Implement the validation of Invalid login functionality of Web Application

Internal Architecture of Selenium Python Automation Framework



Fig 6. Internal Architecture of Pytest Selenium Based Automation Framework.

Detailed Steps for Framework Implementation (Coding)

Step 1: Automating Login Use Case.

- Create LoginPage.py Object Class under “pageObject” folder.
- Create tes_Login.py test under “testCase”.
- Create conftest.py under “testcase” for setup and fixture initialization.

Step 2: Capture Screenshot on Failure.

- Write selenium script to capture the screenshot on failure.
- Create one screenshot folder to save all the images.
- Use this script test_Login.py to capture screenshots during failure.

Step 3: Read common values from .ini file

- Add config.ini file in “Configuration” folder .
- Create “readProperties.py” utility file under utilities package to read common data.
- Replace URL , static data hard coded value from script to parameterized variable.
- Create Locators.py to and used the variable locator name in the script from this file.

Step 4: Add logging to the test case script.

- Add customLogger.py file under utility package it contains the file handler and stream handler logger implementations.
- Call the logger object in test cases to add logs in the test script.

Step 5: Run test on desired browser/ Cross Browser and parallel.

- Update the conftest.py with the required fixture which will accept the command line argument from browser or directly provide browser list in the (params=['chrome', 'Firefox']) fixture file.
- Pass argument name in the browser itself or define in the fixture file to run for multiple browser.
- To run test case in parallel install pytest-xdist library in the framework.

To run test on desired browser

```
>pytest -v -s testCase\test_Login.py --browser chrome  
>pytest -v -s testCase\test_Login.py --browser firefox
```

To run test parallel

```
>pytest -v -s -n=2 testCase\test_Login.py
```

Step 6: Generate Pytest HTML Report.

- To Generate html report, install pytest-html report library upgrade conftest.py with html hook

To generate html report run the below command.

```
>pytest -v --html=Report\report.html testCase\test_Login.py
```

Step 7: Automating Data Driven Test Cases.

- Prepare the test data in excel sheet, Place the excel sheet inside “testData” folder
- Create XLUtility.py utility class under utility package
- Create test_LoginDataDriven.py under “testCase” package.
- Run the test case to verify data driven functionality of framework.

Step 8: Adding New Test Cases.

- Add test cases related to other functionality verify home page title.
- Add test cases related to negative scenarios like invalid login functionality.

Step 9: Group the test cases and run the test cases based on set of groups.

- Pytest framework provides the feature to group the test cases based on markers.
@pytest.mark.sanity
@pytest.mark.regression

- To use the markers on pytest framework add markers entry in pytest.ini file
- **[pytest]**
markers=
 sanity
 regression

- Selecting groups of test cases at run time pytest provide -m switch.

```
-m "sanity"  
-m "regression"  
-m "sanity or regression"
```

Run Command

```
>pytest -v -m "regression" --html=Report\report.html testcase
```

Step 10: Run the test cases through Command Prompt using run.bat file.

- Create run.bat file and paste this command inside the file.
>pytest -v -m "regression or sanity" --html=Report\report.html testcase
- To run test cases directly from cmd, Open command prompt as Administrator and then run.bat file

Step 11: Push the code to Version Control System (VCS) Git lab repository

- Initialize the folder repo with git init command, stage the project changes, then push it to local repo after that commit the changes to remote repository.
- Update the project file and repeat the above steps to push the modified changes create the MR request.
- Once the MR approved merge the changes into the main branch.

Step 12: Integrate the code with CI/CD pipeline Git Lab Runner

- We can use git lab runner to execute the pipeline, Git lab runner provide facility to run the CI/CD pipeline based on trigger and schedule.
- To run CI/CD pipeline create gitlab-ci.yml file , define the image, stages, services, script and artifact parameters
- After validation of the gitlab-ci.yml file we can execute the test case pipeline inside the git lab runner.

Step 13: Configure allure report, install allure docker service, and run the allure report generation pipeline.

- Allure report is more detailed and descriptive version of test results. To obtain the allure report through git lab runner pipeline execution, we need to add some more stages in yaml file.
- To generate allure report download the artifact in the git lab repository using curl command
- Once the gitlab-ci.yml file validated successfully, run the pipeline in through the git lab runner, next stage to generate report use allure-docker-service image copy the allure-report data to public directory.
- After the test case pipeline succeeded, allure report generation pipeline will run we can see allure report inside git lab pages.

Implemented Code Screenshot and short description.

Page Object Model Coding Implementation in the framework Structure for different web screens.

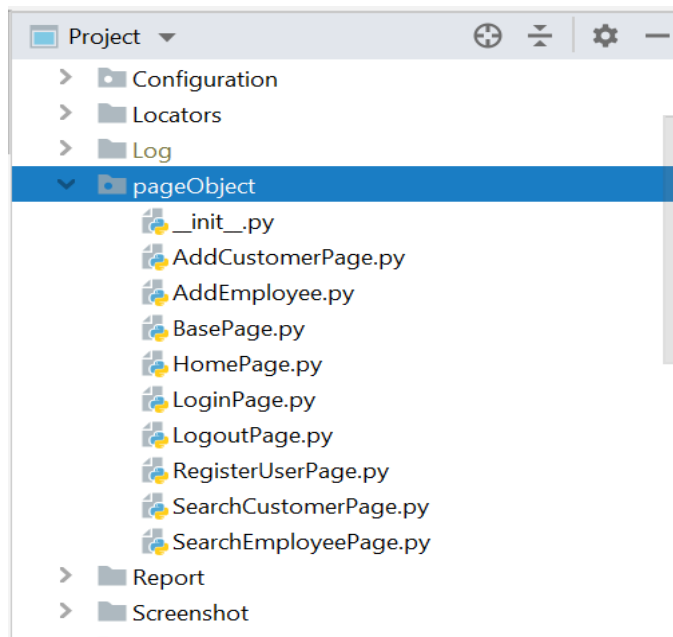


Fig 7. Page Object Package

Under the testcase package we have written test cases related to multiple web feature functionalities.

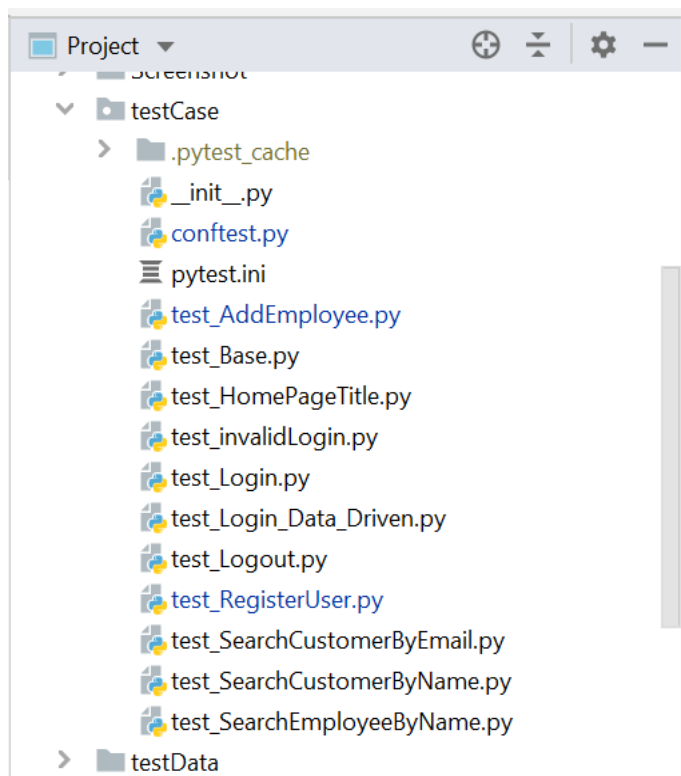


Fig 8. Test Case Package

Inside Conftest.py file, implemented setup and teardown using Pytest fixture for cross browser testing, to run the test cases on multiple browser chrome, safari, Firefox etc.

Conftest file also allow to pass the command line argument in the runner command and decide to select value based on env and browser while executing the script.

```

3  from webdriver_manager.firefox import GeckoDriverManager
4  from selenium.webdriver.chrome.service import Service
5  import pytest
6  from selenium.webdriver import Remote
7
8  @pytest.fixture(params=["chrome", "firefox"], scope='class')
9  def setup_driver(request):
10     if request.param == "chrome":
11         options = webdriver.ChromeOptions()
12         options.add_argument('--no-sandbox')
13         options.add_argument('--headless')
14         options.add_argument('--disable-gpu')
15         options.add_argument('--disable-dev-shm-usage')
16         options.add_argument("start-maximized")
17         options.add_argument("disable-infobars")
18         options.add_argument("--disable-extensions")
19         web_driver = webdriver.Chrome(service=Service(ChromeDriverManager().install()))
20
21     if request.param=="firefox":
22         web_driver = webdriver.Firefox(service=Service(GeckoDriverManager().install()))
23
24     if request.param=="edge":
25         web_driver = webdriver.Edge(service=Service(ChromeDriverManager().install()))
26
27     request.cls.driver = web_driver
28     yield
29     web_driver.close()

```

Fig 9. Conftest.py

Configuration folder contain information related to launch URL, login, and Password for initial account role. Chrome Web driver, browser executable path, also the details related to home page title and Login Page title etc. This value should not be hard coded in the script when required read it from the configuration by just calling the config.py class Test Data.

```

1  class TestData:
2      CHROME_EXECUTABLE_PATH=""
3      FIRFOX_EXECUTABLE_PATH = ""
4      BASE_URL = "https://opensource-demo.orangehrmlive.com/web/index.php/auth/login"
5      InvalidUsername="admin"
6      InvalidPassword = "admin"
7      USER_NAME = "Admin"
8      PASSWORD = "admin123"
9      ACCOUNT_NAME = "admin123"
10     LOGIN_PAGE_TITLE = "OrangeHRM"
11     Login_Page_Headig = "Login"
12     login_page_url = "https://opensource-demo.orangehrmlive.com/web/index.php/auth/login"
13     home_page_url = "https://opensource-demo.orangehrmlive.com/web/index.php/dashboard/index"
14     Home_page_heading = "Dashboard"
15     HOME_PAGE_TITLE = "OrangeHRM"

```

Fig 10. Config.py

Test data folder contain the csv or excel or .txt file for using the parameterize data in the script. To run the data driven testing input and output data stored in csv or excel file that can be used from this folder.

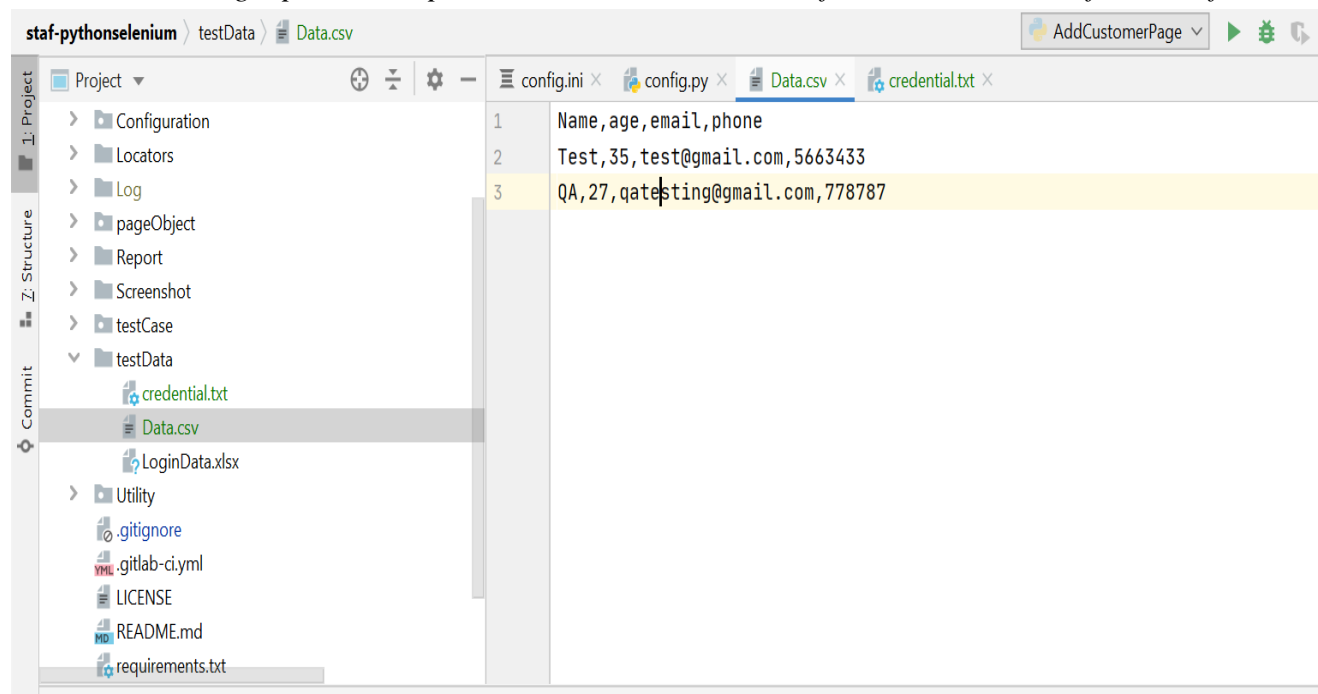


Fig 11. Test Data Folder

Pytest.ini file responsible to group the test cases based on tags, if we want to run test cases related to login feature, we can group these test cases using markers. This help in running test cases related to requirement like release, sanity, smoke or regression etc.

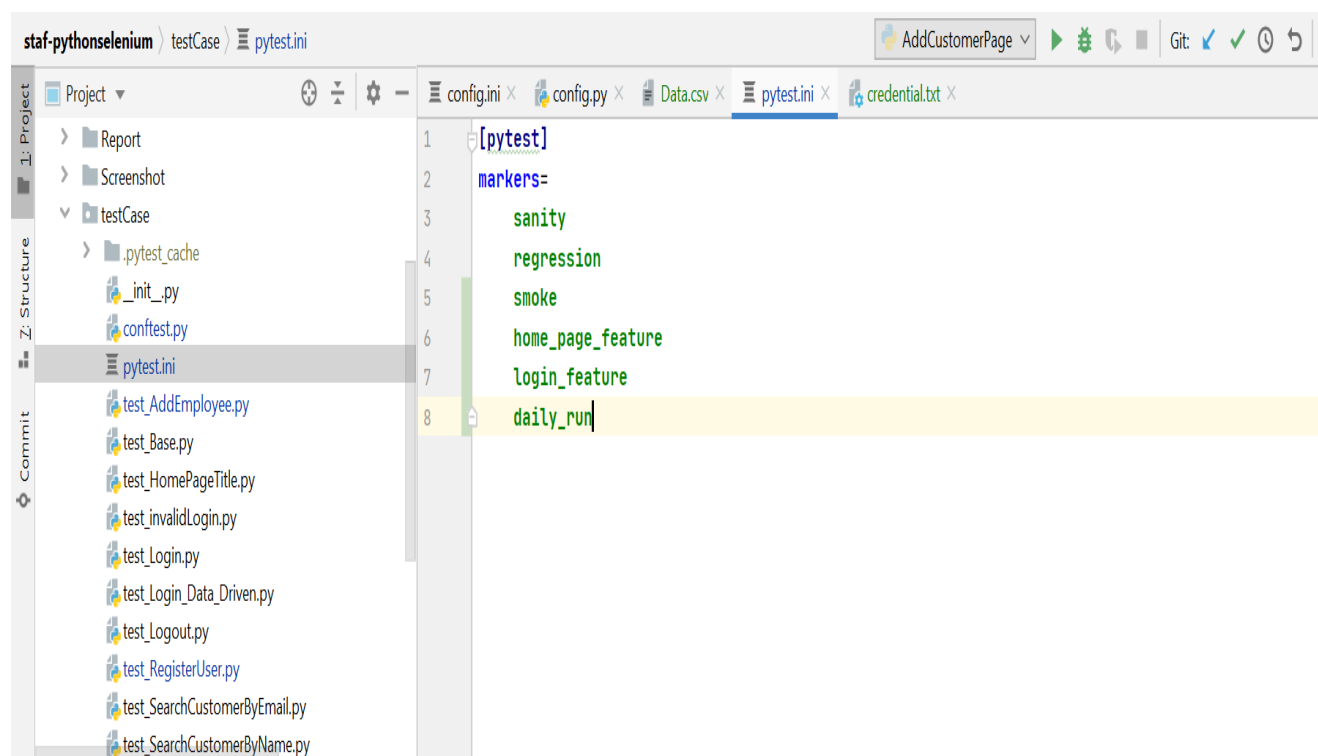


Fig 12. Pytest.ini

Utility folders contain the common methods and parser function which help to read or write the data from csv or excel file, these are general method, contains logger initialization and implementation to use log function in the test case script. These methods are created so that if anybody want to perform an operation related to file or parsing, they just need to call this function instead of writing separate parsing login script in their script.

```

1 import openpyxl
2
3 def getRowCount(file, sheetName):
4     workbook = openpyxl.load_workbook(file)
5     sheet = workbook[sheetName]
6     return(sheet.max_row)
7
8 def getColumnCount(file, sheetName):
9     workbook = openpyxl.load_workbook(file)
10    sheet = workbook[sheetName]
11    return(sheet.max_column)
12
13 def readData(file, sheetName, rownum, columnno):
14    workbook = openpyxl.load_workbook(file)
15    sheet = workbook[sheetName]

```

Fig 13. XLUtililty.py

Locator Folder contains details related to web object identifier, web element can be located using different ways like xpath, CSS, tagName, linkText, id etc. In this folder all the locator are placed and each script call the locators from this file instead of local initialization of locator. If any web element updated or changes we just need to change/update the locator only at one place instead of all the script location.

```

1 from selenium.webdriver.common.by import By
2
3
4 class Locator:
5     textbox_username_id = (By.NAME, "username")
6     textbox_password_id = (By.NAME, "password")
7     button_login_xpath = (By.XPATH, "//*[@type='submit']")
8     login_page_heading = (By.XPATH, "//*[@class='oxd-text oxd-text--h5 orangehrm-login-title']")
9     home_page_heading = (By.XPATH, "//*[@class='oxd-text oxd-text--h6 oxd-topbar-header-breadcrumbs']")
10    link_logout_linktext = (By.LINK_TEXT, "Logout")
11    logout_dropdown_xpath = (By.XPATH, "//*[@class='oxd-icon bi-caret-down-fill oxd-userdropdown-x"]
12
13    # Add User
14    adminUser_xpath = (By.XPATH, "//*[@href='/web/index.php/admin/viewAdminModule']")
15    adNewUser_xpath = (By.XPATH, "//*[@class='oxd-button oxd-button--medium oxd-button--secondary"]
16    user_role_dropdown = (By.XPATH, "//*[@text()='User Role']//following::div[@class='oxd-select-te"]
17    selectList_xpath = (By.XPATH, "//*[@role='listbox']")
18    optionBox_xpath = (By.XPATH, "//*[@class='oxfd-select-option']")
19    user_status_xpath = (By.XPATH, "//*[@text()='User Role']//following::div[@class='oxd-select-te"]
20    input_password_xpath = (By.XPATH, "//*[@text()='Password']//following::input[1]")
21    confirm_password_xpath = (By.XPATH, "//*[@text()='Confirm Password']//following::input[@class="
22    employee_name_xpath = (By.XPATH, "//*[@placeholder='Type for hints...']")

```

Fig 14. Locators.py

Log Folder contains the step-by-step execution flow of the test case, this helps in verifying the feature details and navigation also keep the record of each test case run, log file name generated with date and timestamp at the execution of test cases.

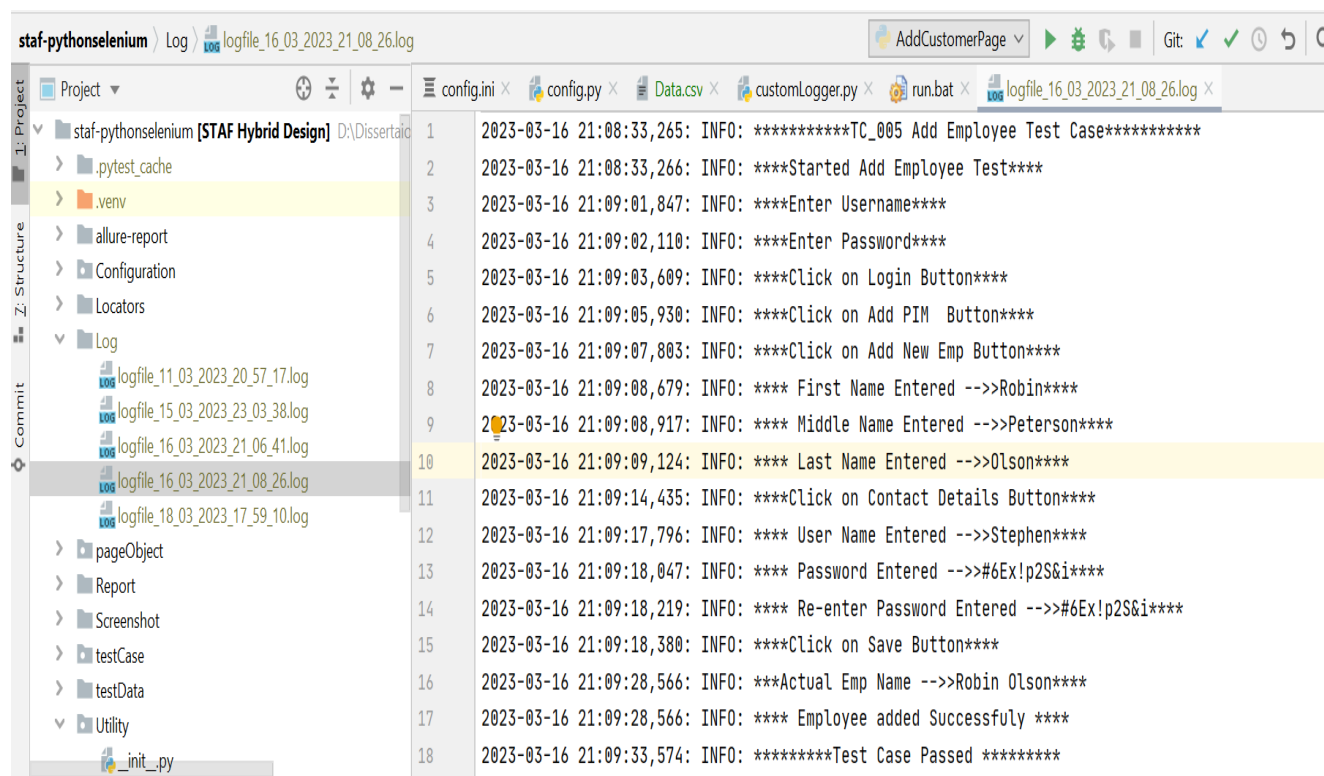


Fig 15. Logs Folder

Screenshot folders contain the captured screens of failed test case at the time of executing the test case. It helps in troubleshooting and debugging the failed test cases. This functionality aid in easily tracking the issue on the web page when test cases run on schedule pipeline.

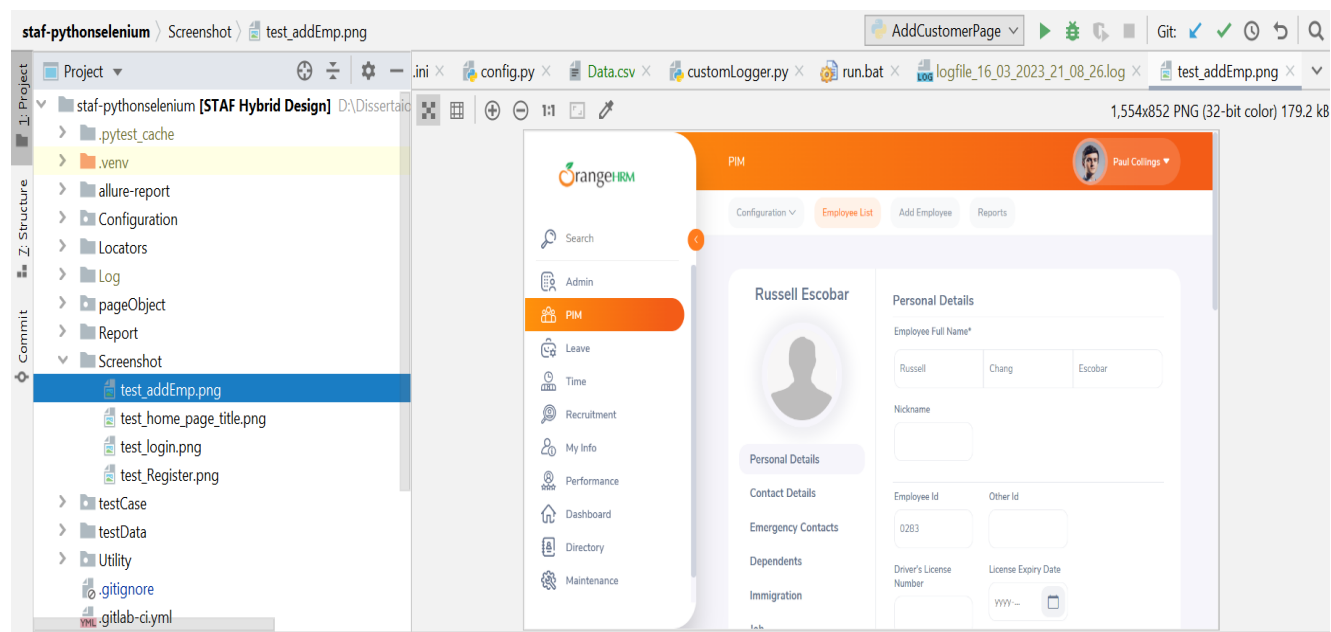


Fig 16. Screenshot Folder

Test Automation Framework Output Results and Metric.

Python integrated with pytests generate out of box html report but just providing one tag command at the time of execution of test cases. These reports contains summary of test execution and Result of Pass and failed test cases.

Test case execution Pass Report

report.html

Report generated on 09-Mar-2023 at 16:46:04 by pytest-html v3.2.0

Environment

Packages	["pluggy": "1.0.0", "pytest": "7.2.1"]
Platform	Windows-10-10.0.19041-SP0
Plugins	["Faker": "16.7.0", "allure-pytest": "2.12.0", "html": "3.2.0", "metadata": "2.0.4", "xdist": "3.1.0"]
Python	3.8.3

Summary

10 tests ran in 295.97 seconds.

(Un)check the boxes to filter the results.

☒ 10 passed, ☐ 0 skipped, ☐ 0 failed, ☐ 0 errors, ☐ 0 expected failures, ☐ 0 unexpected passes

Results

Show all details / Hide all details

Result	Test	Duration
Passed (show details)	test_Login.py::Test_001_Login::test_home_page_title[chrome]	13.17
Passed (show details)	test_Login.py::Test_001_Login::test_login[chrome]	17.24
Passed (show details)	test_Login.py::Test_001_Login::test_home_page_title[firefox]	22.15
Passed (show details)	test_Login.py::Test_001_Login::test_login[firefox]	13.60
Passed (show details)	test_Login_Data_Driven.py::Test_002_Login_DDT::test_DDT_login[chrome]	78.55
Passed (show details)	test_Login_Data_Driven.py::Test_002_Login_DDT::test_DDT_login[firefox]	92.41
Passed (show details)	test_invalidLogin.py::Test_006_Login::test_login_page_title[chrome]	12.97
Passed (show details)	test_invalidLogin.py::Test_006_Login::test_invalid_login[chrome]	11.46
Passed (show details)	test_invalidLogin.py::Test_006_Login::test_login_page_title[firefox]	19.99
Passed (show details)	test_invalidLogin.py::Test_006_Login::test_invalid_login[firefox]	13.37

Fig 17. Pytest HTML Pass Report

Test Case Execution Fail Report

Result	Test	Duration	Links
Failed (hide details)	test_SearchCustomerByEmail.py::Test_004_SerchCustomerByEmail::test_search_customer_by_email[chrome]	23.73	
<pre>self = <testcase.testSearchCustomerByEmail.Test_004_SerchCustomerByEmail object at 0x04179838> def test_search_customer_by_email(self): self.logger.info("*****TC_004 Search Customer By email Test*****") self.logger.info("*****Started Search Customer By email Test*****") self.loginPage = LoginPage(self.driver) self.loginPage.setUserName(TestData.USER_NAME) self.logger.info("*****Enter Username*****") self.loginPage.setPassword(TestData.PASSWORD) self.logger.info("*****Enter Password*****") self.loginPage.clickLogin() self.logger.info("*****Click on Login Button*****") self.addCustomer = AddCustomer(self.driver) self.addCustomer.clickOnCustomersMenu() ..\\STAF Hybrid Design\\testCase\\test_SearchCustomerByEmail.py:25: pageObject\\AddCustomerPage.py:16: In clickOnCustomersMenu self.wait_for_element(ele.linkCustomers_menu_xpath) pageObject\\BasePage.py:13: in wait_for_element WebDriverWait(self.driver, 10).until(EC.visibility_of_element_located(by_locator)) ----- self = <selenium.webdriver.support.wait.WebDriverWait (session="647e958bb736029ac8fd2b15bde6e792")> method = <function visibility_of_element_located.<locals>._predicate at 0x03E7CB20>, message = '' def until(self, method, message: str = ""): """Calls the method provided with the driver as an argument until the \ return value does not evaluate to 'False'.""" :param method: callable(WebDriver) :param message: optional message for :exc:`TimeoutException` :returns: the result of the last call to 'method' :raises: :exc:`selenium.common.exceptions.TimeoutException` if timeout occurs """ screen = None stacktrace = None end_time = time.monotonic() + self._timeout while True: try: value = method(self._driver) if value: return value except self._ignored_exceptions as exc:</pre>			

Fig 18. Pytest HTML Fail Report

However, these reports are not much very intuitive and detailed, so we have generated allure report, Allure report are very informative and provide detailed description of the test run. It can be also shared easily across the team, maintain the history of test run and percentage wise graphical dashboard. It is also very easy to integrate this report to CICD pipeline just write one more stage in the git lab yml file and generate the report online as well. In the next section we will see the sample allure-report generated during test case execution.

Allure Report Dashboard

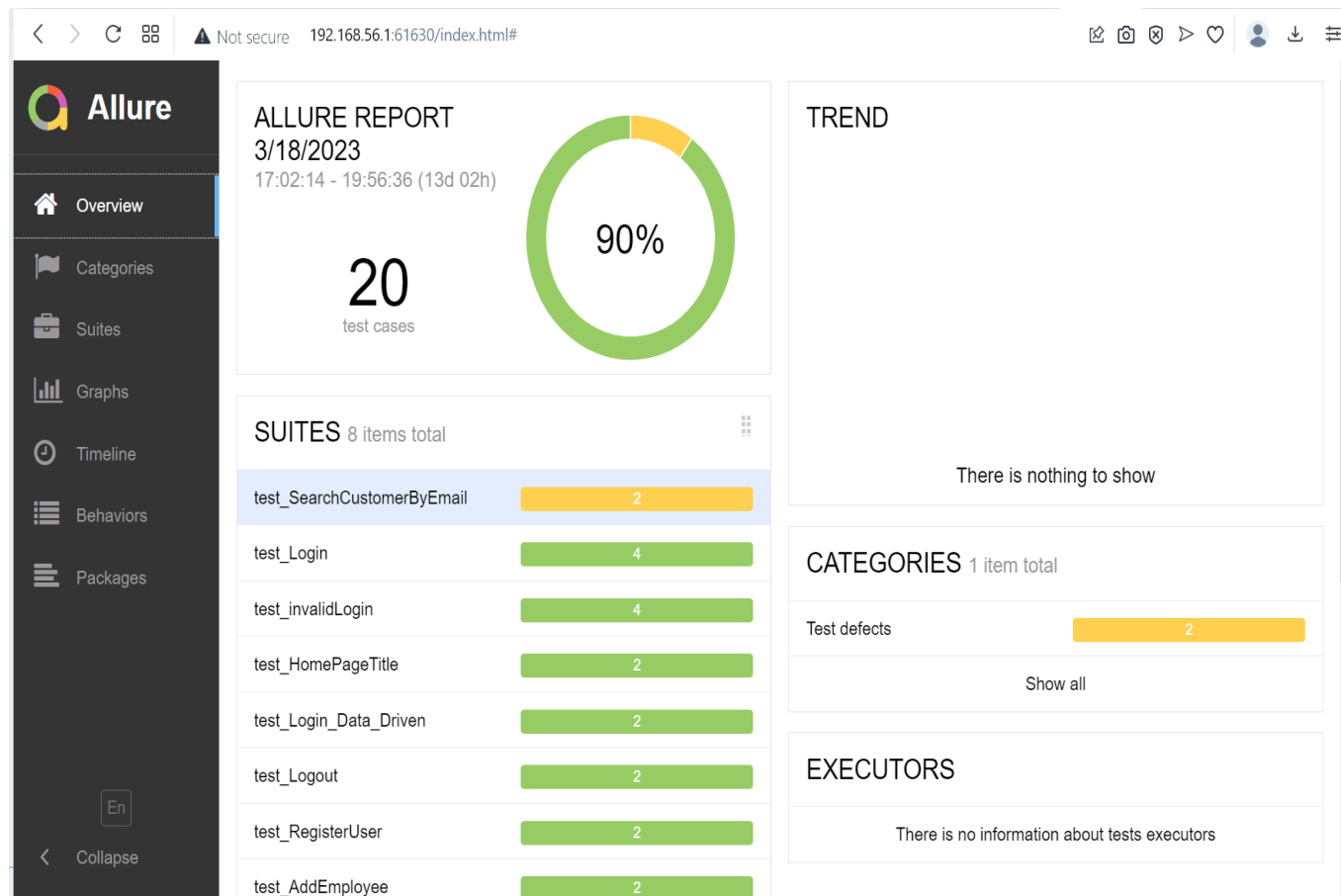


Fig 19. Allure Summary Report

Allure Report Suites

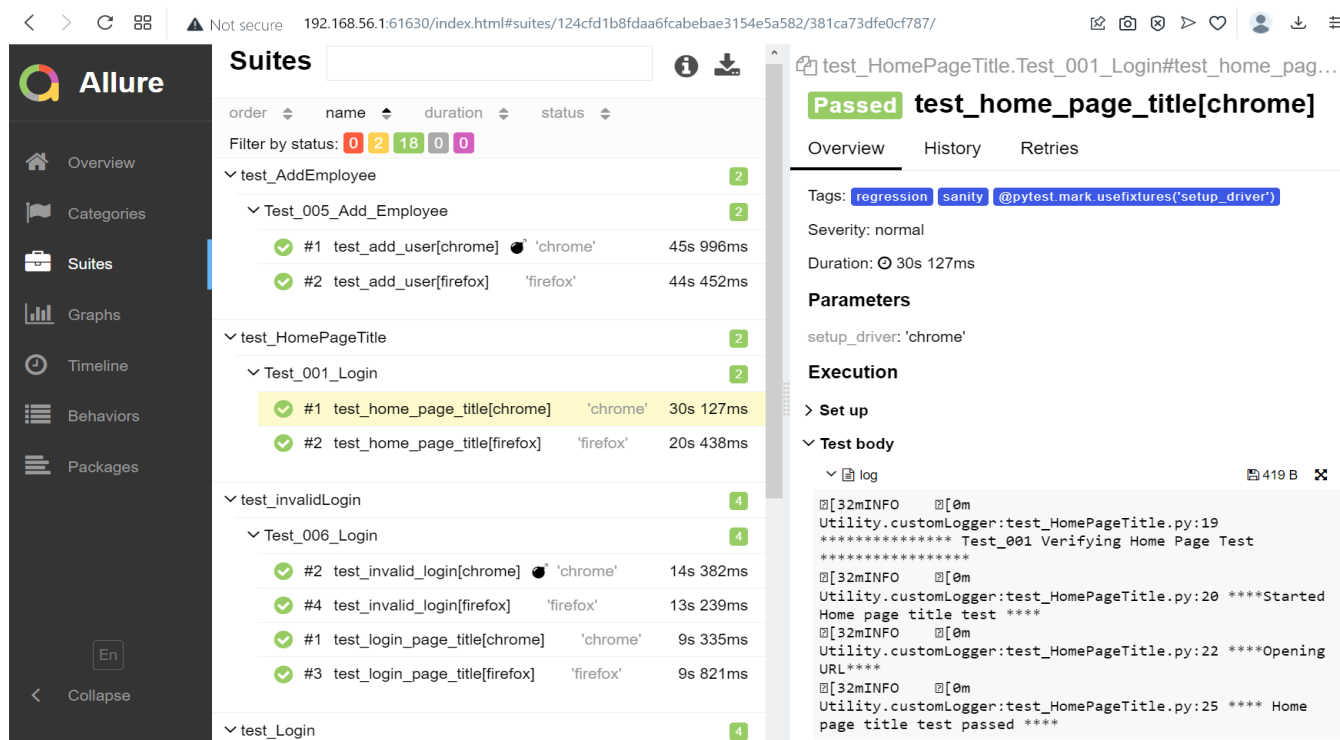


Fig 20. Allure Suites Report

Allure Report Graphs.

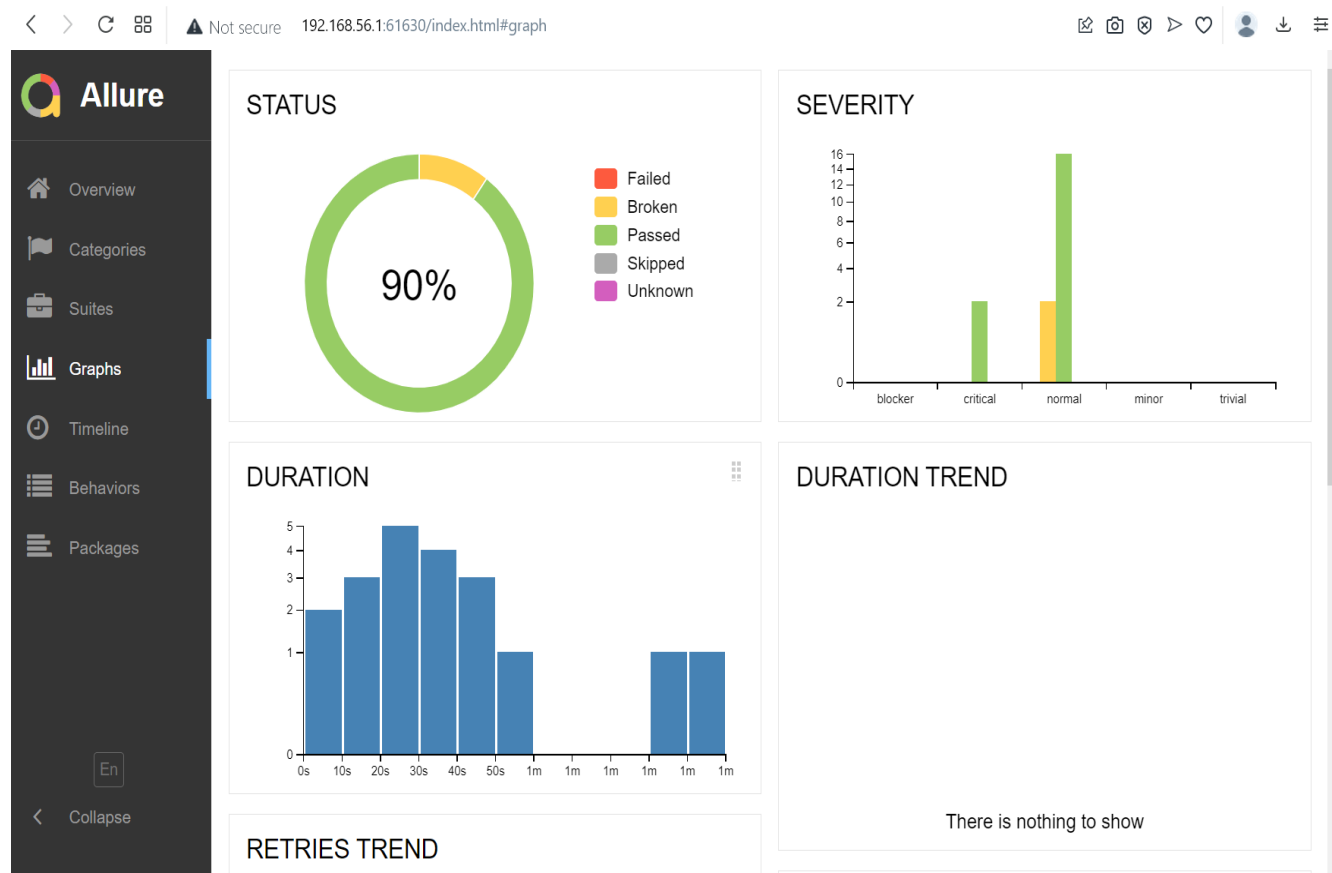


Fig 21. Allure Graph Report

Plan of Pending work & Timelines:

Expected Date	Work Plan
21-03-2023 to 06-04-2023	Push the code to git lab repo, write git lab CICD yml file for pipeline Integration of Test case. Perform regression and schedule testing on remote repo. Write code to Generate allure report in git lab artifact directory.
07-04-2023 to 15-04-2023	Plan the template and Prepare the Dissertation report, Review the report. Verify the report with checklist item for the final project work item.
16-03-2023 to 23-04-2023	Capture the important details from the report prepare the presentation for viva voice. Final report and PPT submission.