**SELENIUM HYBRID FRAMEWORK**

**STEP 1:** Create the project and install the required packages/plugins

Python

Pytest :unit test framework

Selenium: selenium libraries

pytest-html: generate HTML reports

pytest-xdist: Run the test parallel

openpyxl: excel support

Allure-pytest: allure reports

Package (\_\_init\_\_.py file ) in visual studio code

**STEP 2: Create the folder structure**

ProjectName

|

Page object (Package)

|

testCases(Package)

|

utilities(Package)

|

TestData(Folder)

|

Configuration(Folder)

|

Logs(Folder)

|

Screenshot(Folder)

|

Report(Folder)

|

**Run.bat**

**STEP 3: Automating the login test case**

1)Create the login page Page object class under the Page Object

2)create login test under testCases

3)Create the conftest.py under testCases

**STEP 4: Capture the screenshot on the failure**

1. Update the login test with screenshot under testCases

**STEP 5: Read the common values from the .ini file**

1. Add the config.ini file in the configuration folder
2. Create a readproperties.py utility file under the utilities package to read the common data
3. Replace the hardcoded value in the Login testcase

**Flow for ini**

**Ini file --- utilities --- test case**

**STEP 6: ADD log file**

1)Add the customLogger.py in utilities folder

2)add the changes in the test\_login.py

**STEP 7: Run the test in the desired browser /cross-browser/ parallel**

**To run test in the desired browser**

**pytest -s -v testCases/test\_login.py --browser chrome**

**pytest -s -v testCases/test\_login.py --browser firefox**

**To run test in parallel**

**pytest -s -v -n=3 testCases/test\_login.py --browser chrome**

**STEP 8: Generate the HTML report**

Update the conftest.py with python hooks

To generate the HTML run the below command:

**pytest -s -v -n=3 --html=Reports\report.html testCases/test\_login.py --browser chrome**

**STEP 9: AUTOMATING DATA-DRIVEN TEST CASE**

9.1 Prepare test data in Excel sheet place the excel file inside in the TestData folder

9.2 Create “ExcelUtils.py” utility class under utilities package

9.3 Create LoginDataDrivenTest under testCases

9.4 Run the test case

**Step 10: Adding new testcases**

1. Add new customer
2. search customer by name
3. search customer by email

Random test data

**Step 11:Grouping Tests**

11.1 Grouping markers Add markers to every test method

@pytest.mark.sanity

@pytest.mark.regression

To run the test case as sanity and regression

**pytest -s -v -m "sanity" --html=Reports\report.html testCases\ --browser chrome**

**Step 12: Run the test in the command prompt and run.bat file**

1. **create the run.bat file**

**pytest -s -v -m "sanity" --html=Reports\report.html testCases\ --browser chrome**

**Step 13: Push the code to Git and GitHub Repository**

Initial steps (only for one time)

**1)git init 🡪 create the empty repository(local repository)**

**2) git remote add origin \***

**https://github.com/Automationtest1234/Ecommerence**

**Before doing the commit first time we need to execute the below commands**

**git config --global user.name “avinash”**

**git config --global user.email** [**avinashsmart241198@gmail.com**](mailto:avinashsmart241198@gmail.com)

**3)git status**

**4) git add -A 🡪 add all the files into the staging/indexing area**

**5) git commit -m “first my commit”**

**6)git push -u origin master**

**https://github.com/Automationtest1234/Ecommerence.git**

**git log -> show the history of the all commits in chronological order of the active branch**

**git pull 🡪 command to pull the file from the GitHub**

**git add -A**

**git status**

**git commit -m “this is my second commit”**

**git push -u origin master**

[**http://localhost:8080/**](http://localhost:8080/)

**Item name in Jenkins**

**Jenkins CI/CD tool**

1)Download Jenkins.war file

2)Create the Jenkins directory

3)Paste the war file in the Jenkins directory

4)Open the command prompt

5)Run this command

**java -jar jenkins.war**

6)open the browser and enter the URL

**localhost:8080**

1)click on the New item

2)enter the item name

3)click on the freestyle project

4)Enter the repository URL

**Step 14: Generate the Allure report**

To install the allure report in Windows:

Allure framework go to download click on Maven Central download the latest version

Add the path to the environment variable in the path variable:

**C:\allure-2.21.0\bin**

To install allure report in python

**pip install allure-pytest**

to run the below command to generate the allure report

**pytest -v -s --alluredir=”C:\Users\avinash.anand\PycharmProjects\WEBHMI\Reports” testlogin.py**

To generate the allure report in the command prompt

allure serve C:\Users\avinash.anand\PycharmProjects\WEBHMI\Reports

Race condition in Selenium:

Race condition happens whenever the automation attempts to interact with the page before it is fully loaded

To avoid the race condition always wait for the target element or page property to be ready

Two kinds of waits:

Implicit waits are specified once and applied to all interactions

Explicit waits must be specified per interaction

Basic locators:

|  |  |
| --- | --- |
| ID | To find the element by ID of the element |
| Classname | To find the element by Classname of the element |
| Name | To find the element by name of the element |
| Link text | To find the element by text of the link |
| XPath | XPath required for finding the dynamic element and traverse between various elements of the web page |
| CSS path | CSS path also locates elements having no name, class or ID. |

XPath :

XPath contains the path of the element situated at the web page. Standard XPath syntax for creating XPath is.

**Xpath=//tagname[@attribute='value']**

* **// :** Select current node.
* **Tagname:**Tagname of the particular node.
* **@:** Select attribute.
* **Attribute:** Attribute name of the node.
* **Value:** Value of the attribute.

**Types of X-path**

There are two types of XPath:

**1) Absolute XPath**

**2) Relative XPath**

**Absolute XPath:**

It is the direct way to find the element, but the disadvantage of the absolute XPath is that if there are any changes made in the path of the element then that XPath gets failed.

The key characteristic of XPath is that it begins with the single forward slash(/) ,which means you can select the element from the root node.

**Relative Xpath:**

**Relative Xpath** starts from the middle of HTML DOM structure. It starts with double forward slash (//). It can search elements anywhere on the webpage, means no need to write a long xpath and you can start from the middle of HTML DOM structure. Relative Xpath is always preferred as it is not a complete path from the root element.

Below is the example of a relative XPath expression of the same element shown in the below screen. This is the common format used to find element by XPath.

//div[@class=”featured-box”]//\*[text()=’Testing’]

**Basic Xpath:**

<input type=”text” colour=”validateuserid()”maxlength=”10” name=”uid”/>

Xpath=//input[@name=’uid’]

<label id=”message23”>userid is blank</label>

Xpath=//label[@id=’message23’]

//tagname[text()=’Text of the Web Element’]

Bookstore/book[@category=’cooking’]/title

### Contains():

Contains() is a method used in XPath expression. It is used when the value of any attribute changes dynamically, for example, login information.

The contain feature has an ability to find the element with partial text as shown in below XPath example.

In this example, we tried to identify the element by just using partial text value of the attribute. In the below XPath expression partial value ‘sub’ is used in place of submit button. It can be observed that the element is found successfully.

Complete value of ‘Type’ is ‘submit’ but using only partial value ‘sub’.

Xpath=//\*[contains(@type,'sub')]

Complete value of ‘name’ is ‘btnLogin’ but using only partial value ‘btn’.

Xpath=//\*[contains(@name,'btn')]

In the above expression, we have taken the ‘name’ as an attribute and ‘btn’ as an partial value as shown in the below screenshot. This will find 2 elements (LOGIN & RESET) as their ‘name’ attribute begins with ‘btn’.

Similarly, in the below expression, we have taken the ‘id’ as an attribute and ‘message’ as a partial value. This will find 2 elements (‘User-ID must not be blank’ & ‘Password must not be blank’) as its ‘id’ attribute begins with ‘message’.

Xpath=//\*[contains(@id,'message')]

In the below expression, we have taken the “text” of the link as an attribute and ‘here’ as a partial value as shown in the below screenshot. This will find the link (‘here’) as it displays the text ‘here’.

Xpath=//\*[contains(text(),'here')]  
Xpath=//\*[contains(@href,'guru99.com')]

river.find\_element(By.XPATH,"//div[@class='job\_title'][normalize-space()='one']").click()  
time.sleep(2)  
driver.find\_element(By.XPATH,"//div[@class='activate-label']").click()  
time.sleep(1)

driver.find\_element(By.XPATH,"//div[@class='job\_title'][normalize-space()='one']").click()  
time.sleep(2)  
driver.find\_element(By.XPATH,"//div[@class='activate-label']").click()  
time.sleep(1)

API:

Application Programming Interface. Ports test scripts you write in Ruby, Java, Python, or C# to Selenese (Selenium’s own scripting language), through bindings.

##### Library

Houses the API and language-specific bindings. Although plenty of third-party bindings exist to support different programming languages, the core client-side bindings supported by the main project are: Selenium Java (as selenium jar files), Selenium Ruby, Selenium dotnet (or Selenium C#, available as .dll files), Selenium Python, and Selenium JavaScript (Node).

##### Driver

Executable module that opens up a browser instance and runs the test script. Browser-specific—for instance, Google develops and maintains Chromedriver for Selenium to support automation on Chromium/Chrome.

##### Framework

Support libraries for integration with natural or programming language test frameworks, like Selenium with Cucumber or Selenium with TestNG.

**How it Works:**

The WebDriver protocol has a local end (‘client’) which sends the commands (test script) to a browser-specific driver. The driver executes these commands on its browser-instance. So, if the test script calls for execution on Chrome and Firefox, the ChromeDriver will execute the test on Chrome; the GeckoDriver will do the same on Firefox.

### **How Selenium Testing Boosts Agile Development**

##### What is Agile?

Agile is a development methodology. It starts with the simplest working version of the product design—one that can be continuously improved.

Here’s what a typical Agile workflow looks like:

* Stakeholders agree upon the ‘simplest working’ design of the product.
* The design gets divided into smaller modules.
* **Each module is assigned to a cross-functional team of developers, designers, and Quality Assurance personnel.**
* Teams work in sprints to create their modules within a time-frame (‘iteration’)—a window of one to four weeks.
* At the end of each iteration, finished modules are put together. **Tests are run and a functional product (with minimum bugs) is demonstrated to the stakeholders**.
* The stakeholders evaluate project priorities, add customer feedback, and adapt as needed.

The whole cycle begins again with the next iteration and a new set of modules. A ‘market-ready’ product or a new feature will always need multiple iterations.

##### Where does testing automation come in:

* QAs are involved from early stages to run a series of unit and acceptance tests on modules.
* Integration tests on every iteration ensure that separately coded modules don’t break when put together.
* Each new iteration requires regression tests (so it doesn’t break the previous working iteration).

It’s essential to keep track of code as well as test cases, so all iterations are well documented. While we’re on the subject, you should note that this recurrent testing is a theme in any sub-category of rapid, iterative development based on Agile, like CI/CD.

### **How Selenium Testing is Integral to Continuous Integration/Delivery (CI/CD)**

##### What is CI/CD?

Continuous Integration/Delivery prioritizes delivery of new releases of a build, frequently and quickly. A project that’s launched remains open to continuous iterations (like Agile).

The only difference is this: the project also remains ready to be shipped at all times (instead of waiting for iterations to run their course).

##### A CI/CD pipeline looks like this:

* A developer has code they wants to integrate into the project
* An external CI server does an ‘integration’ test—it grabs the source files and attempts to do a build with the new code.
* If the build completes successfully, the server packages the changes with source files. If not, the server notifies members of the team.

CI engines (like Jenkins or Bamboo) have dashboards that display current and previous builds, logs of previous check-ins and their status (successful/failed), what broke (and when), etc. Everyone remains informed about any change in code, infrastructure, or configuration. This ensures that deployment failures are caught (and fixed) early.

**Note:** There’s a difference between a ‘successful build’ and ‘quality build’. Even if a new integration is successful, it’s not considered ready to ship until it has passed a series of tests by QA engineers. That’s where automation testing with Selenium comes in handy.

Selenium automates frequent and recurrent functional, performance, and compatibility testing. This gives developers near-instant feedback for faster debugging, leaving them with more time to code business logic for newer versions/features.

Modern web development needs Selenium testing because:

* It automates repeated testing of smaller components of a large(r) code-base
* It’s integral to agile development and CI/CD
* It frees resources from manual testing
* It’s consistently reliable; catches bugs that human testers might miss
* It can provide extensive test coverage
* It’s precise; the customizable error reporting is an added plus
* It’s reusable; you can refactor and reuse an end-to-end test script every time a new feature gets deployed.
* It’s scalable; over time, you can develop an extensive library of repeatable test cases for a product

## What Types of Testing can be Automated with Selenium?

Types of testing that are commonly automated with Selenium are:

##### Compatibility Testing:

Done by QA professionals/Testers to ensure that the web app meets performance benchmarks on different browser-OS combinations. For example, testing on different devices (mobile and desktop) to ensure that the front-end fits to scale (responsive); testing on different browsers to see if video ads render on the pages as they should.

##### Performance Testing:

Series of tests done by QA professionals/Testers to ensure that the project meets performance benchmarks set by the stakeholders. Tester writes a script that checks whether all elements on homepage load within 2 seconds on different browsers/browser versions.

##### Integration Testing:

Done by developers to verify that units/modules coded separately (that work on their own), also work when put together. Parallel Test Calculator, for instance, has separate layers. UI takes input and business logic calculates the output—then sends it back to UI to display. The tester could verify whether they are able to relay data/output when integrated.

##### System Testing:

aka Black Box testing. Done by Testers/QA professionals with no context of the code or any previously executed tests. Typically centered on a single user workflow. The check-out process on a product website, for instance, comprises of: validating user credentials, fetching products from the cart, checking their availability, and validating payment details—before redirecting to the bank website. The tester could write a script to verify that the entire system is functional.

##### End-to-end Testing:

Also done by Testers/QA professionals, typically from the user’s point of view. The aim is to verify that all touchpoints on the web app are functional. From the previous example, the tester could write a series of test cases to check that sign-up, product search, checkout, review, bookmark, and all other features function as intended (and fail when invalid values are entered in input fields).

##### Regression Testing:

A series of tests done to ensure that newly built features work with the existing system. From the same example, say the product website launches a new feature (promotional codes) that automatically apply to eligible items before checkout. The tester could write cases to verify that it doesn’t break the rest of the checkout feature.

Well-written test suites can also automate Smoke and Sanity testing with Selenium.

**Note:**Selenium testing is not meant to replace manual testing. Testing automation, by its very definition, automates that which does not merit human evaluation. You can’t automate the testing of your newly revamped UI for human-usability. But for everything else, there’s Selenium.

## Who Uses Selenium?

Short answer: Everyone who cares about the state of their web app.

Part of the reason why Selenium is so popular is its flexibility. Anyone who codes for the web can use Selenium to test their code/app–from individual freelance developers running a quick series of tests for debugging to UI engineers doing visual regression tests after a new integration.

In an enterprise environment, testing with Selenium falls under the purview of QA engineers. They are tasked with writing focused, non-flaky (i.e., deterministic) scripts to maximize test coverage and accuracy, refactoring old test suites for newer versions of the project, and maintaining test infrastructure (from the Hub to the test-case library).

They would be the ones creating comprehensive test suites to pinpoint ‘show-stopper’ bugs and advising stakeholders about updating performance benchmarks for the project. Their end goal is to ensure maximum test coverage and efficacy, which in turn boosts the overall productivity of the engineers at work.

## Which Browsers can I run Selenium Tests on?

Most desktop/ mobile browsers today have built-in support for automation testing with Selenium. Consumer browser vendors like Firefox, Chrome, Safari, IE, and Opera develop and ship their browser’s drivers.

In the years since it was first open-sourced, others have contributed to the Selenium project by adding third-party drivers for specialized browsers like BlackBerry 10 and HtmlUnit, as well as bindings for integration with development frameworks like PhantomJS, Qt, etc.

**Automation testing life cycle Methodology**

The implementation of automation testing life cycle executes in parallel with the software development life cycle process.

The structured automation testing life cycle comprises of a multi-stage process that supports the activities required to utilize and introduce an automated test tool, develop and run test cases, develop test design, build and handle test data and environment.

**Automation testing life cycle (ATLC):**

* Determining The Scope Of Test Automation
* Selecting The Right Tool For Automation
* Test Plan + Test Design + Test Strategy
* Setting Up The Test Environment
* Automation Test Script Development + Execution
* Analysis + Generation Of Test Reports

### **Determining The Scope Of Test Automation**

It is the first stage of automation testing life cycle and it aims to identify the feasibility of automation. Every aspect should be considered while analyzing the feasibility.

Also, it is essential to perform a feasibility analysis on the manual test case pack that allows automation engineers to design the test scripts.

In this particular stage, the following things should be taken care of without a failure.

1. Which modules of the applications can be automated and which not?
2. Which test can be automated and how to automate them?
3. Factors like cost, team size and expertise should also be considered.

Following feasibility checks should be done before starting the test automation:

* Test Case Automation Feasibility
* AUT Automation Feasibility

Navigate through the application screens under test and mention all UI components of the application to perform feasibility analysis in an effective way. Identify a percentage of UI components to be automated via an automation testing tool.

### **Selecting The Right Automation Tool**

Automation testing is highly tool dependent. That is why finding the right automation testing tool is a critical phase for an automation testing life cycle. When you are looking for an automation tool you need to keep in mind the budget, technologies being used in the project, familiarity of the tool with resources on-board, intuitiveness, flexibility and more. Choose a tool that provides you with a support team who can take care of queries or issues.

### **Test Plan + Test Design + Test Strategy**

It is the most critical phase of automation testing life cycle methodology that defines how to approach and accomplish the goal of test automation. Selecting a test automation framework is the first and foremost thing to do in the Test Strategy phase of Automation Testing Life Cycle.

Selecting a tool depends on the technologies used in the application. Understand your product completely before starting the automation test.

For example, if it is a desktop application, find which language it is built upon. Or, if you want to test a web application, know about the deprecated feature your used which may not be compatible across various browsers.

During the test planning phase, the testing team decides the test procedure creation standards and guidelines; hardware; software and network to support test environment; a preliminary test schedule; test data requirements; defect tracking procedure and associated tracking tool and a procedure to control test configuration and staging environments.

The team of test engineers develops a test architecture to describe the test program structure and the way test procedures are managed after the test program model is designed.

Post designing comes the test architecture where the structure of the entire test program is described along with the management of test procedure.

Make sure to consider the following things when planning a [test management strategy](https://www.lambdatest.com/blog/why-you-need-to-understand-test-management-strategy-to-become-pro/):

1. Gather all manual test cases from the test management tool to identify which test case needs to be automated.
2. Identify which framework to be used after understanding the pros and cons of the testing tool.
3. Build a test suite for Automation test case in the tool for test management.
4. Ensure to mention background, risk, background and dependency between the tool and application in the test plan.
5. Seek approval on test strategy from clients or stakeholders.

### **Setting Up The Test Environment**

As the name indicates, this stage of [Automation Testing](https://www.lambdatest.com/automation-testing) Life Cycle involves setting up a machine or remote machine where test cases will be executed. Why would we need remote machines? Because unless we live in an ideal world, your users would be using different machines to access your website or web-app on the internet.

Keeping a check on different devices is one thing, but we also need to be cautious about various browsers and browser versions. As your website may render differently from one browser to another. Cross browser compatibility testing also known as [cross browser testing](https://www.lambdatest.com/) is a procedure where we test a website or a web-app across multiple browser versions to make sure we deliver a seamless user experience to all our customers.

Environment setup phase needs a thorough planning, you need to ensure that you are able to maximize your test coverage across as many different scenarios as possible. It is the responsibility of test team to schedule and track environment setup activities; install test environment software, network resources and hardware; refine test databases and develop test bed scripts and environment setup scripts.

#### Key Areas For Test Environment Setup

* **Test data** – Many times test environment setup are not populated with similar data with comparison to Production, this makes your product very brittle once the code changes are put into production environment
* **Front-End Running Environment** – Make sure your have a front-end running environment to perform load testing for analyzing the capability of handling web traffic.
* **Checklist of all the Systems, modules and applications** to be put under test.
* An isolated **Database server** for staging environment.
* Test across various **client operating systems**.
* Test across maximum **browsers and browser versions**.
* Make sure you test your website on **low and high network** to realize the difference between rendering time and overall appearance of the website or web-app.
* **Documentation is key** – Make sure you cover all the **Configuration guides/Installation guides/User manuals** and so on in a central repository.

Test environment setup involves the following tasks:

1. Tool licenses.
2. Setup utilities like advanced text editors and comparison tools.
3. Automation framework implementation
4. AUT access and valid credentials
5. Add-ins licenses

Various organizations utilize a staging environment to test the software. The best approach is to copy production data to test. It helps the test engineer to uncover the issues without corrupting the production data. Here are [13 reasons that leads to failure of staging environments](https://www.lambdatest.com/blog/13-reasons-why-staging-environment-is-failing-for-your-organization/).

#### Best Practices To Setup A Test Environment Management:

1. Gather and understand the test environments thoroughly and train the testing team members.
2. Check for the required software, licenses and hardware.
3. Maintain a checklist of automation tools and their configurations.
4. Maintain a cross browser testing matrix to ensure you cover the tests on numerous browsers and versions with respect to priority and market share.
5. Make sure to test using real-time traffic to ensure your changes are more sustainable.
6. Planning the scheduled use of the test environment.

### **Automation Test Script Development + Execution**

Once you install the test environment, it is the time to execute the test script. So, this phase of automation testing life cycle is dedicated to the execution of all test scripts.

To perform script execution, signed-off and unit-tested test scripts are delivered to automation testing team.

it is essential to ensure that all test scripts are running correctly. So, there is a need to take care of the following things before developing a test script:

1. Creation of test scripts based on actual requirements.
2. Create a common method of function that can be used throughout the testing process.
3. Make sure to create a reusable, structured, and easy script so that a third person can understand it clearly. Here are [8 actionable insights for writing better automation code](https://www.lambdatest.com/blog/8-actionable-insights-to-write-better-automation-code/).
4. Perform [code reviewing of test scripts for better quality assurance](https://www.lambdatest.com/blog/how-code-reviewing-can-help-with-quality-assurance/).
5. Make use of better reporting.

Once the test script is developed successfully, it should be executed by keeping the following things in mind:

1. A test script should include all functional aspects according to the test case.
2. Ensure to run test scripts in multiple environments and across multiple platforms.
3. If possible, batch execution can be done to save time and efforts.
4. If the failure occurs due to some functionality, write a bug report.

To execute test scripts and procedures, the test team has to comply with a schedule decided for procedure execution. Evaluations for test outcomes are executed and test results documentation is prepared during this phase.

Test outcome evaluations are executed and documentation for test results is prepared.

Plans designed for the unit, system, user acceptance and integration testing are run to test the system as a whole. Code profiling is done at the time of unit testing. Profiling discovers instances when there is inappropriate scaling of algorithms, resource utilization and instantiations.

### **Analysis + Generation Of Test Results & Test Reports**

After all types of testing are performed, the testing team analyzes to identify particular functionality or components that experience a relative number of problem reports.

The result of the analysis suggests that if it requires additional test efforts and procedures or not.

Test results generated from the analysis can confirm whether executed test scripts/procedures can identify errors.

It is the last phase of automation testing life cycle and the test reports are shared with all involved stakeholders at this stage. This is why, test reports are crucial for analyzing how well your web-app responds to adversity.

**Pytest framework**

* Pytest framework easy to write small tests

To install pytest framework

pip install pytest

**Always create the file name test\_**

Ex: test\_login.py

**Always define the function test\_**

Ex: def test\_login()

**To run the pytest in terminal**

pytest -v -s filename.py

-v verbose the information: PASSED

-s is used to print the information inside the function

Pytest fixtures

The purpose of test fixtures is to provide a fixed baseline upon which tests can reliably and repeatedly execute

**@pytest.fixture()**

* Executes specific method before every test method

**@pytest.yield\_fixture()**

* Execute specific method before and after every test method

Pytest-html

pip install pytest-html

To generate the html report

pytest -v -s --html=report.html test\_login.py

setup()

print(“before”)

yield

print(“after”)

test\_method()