

# E-COMMERCE UI AUTOMATION (NOPCOMMERCE)

*Capstone Project*



Presented By Group 7

# INTRODUCTION

A team of software tester with an academic foundation in Computer Science Engineering have designed a UI , API and manual automation framework which does the following:

- Test case design and execution
- API testing using Postman
- UI automation using Selenium with Python
- Framework development using Pytest and Robot Framework

# Our Team

PYTEST

## SUPERSET ID

## NAME

4957579

Dibyojoyti Deb

4958879

Harsh Kumar Sinha

4958238

Aditya Raj

4956637

Anunay Kumar

4956997

Gaurav Kumar

4957628

Ankur Santosh Waghmode

Mentored by  
Saritha R. Parthipan  
ma'am

## ***Key Takeaways / Learnings from the Program***

- Gained knowledge of manual and automation testing by designing and executing real-time test cases.
- Developed end-to-end automation using Selenium with Python (Pytest/Robot Framework).
- Built and executed automation frameworks while improving test reliability and reusability.
- Enhanced debugging, analytical, and scripting skills to ensure application quality.
- Strengthened communication and teamwork significantly.

# Problem Statement

- The nopCommerce demo website is an e-commerce web application where users can register, log in, search products, add items to the cart and place orders.
- Manual testing of these workflows is repetitive, time-consuming and prone to errors, especially with frequent application updates.
- Automation Solution Used:
  - Selenium WebDriver with Python
  - Pytest Framework
  - Robot Framework

# Project Overview

|   | Project                     | Type                   | Tools & Technologies  | Outcome  |
|---|-----------------------------|------------------------|---|--|
| 1 | Python Selenium Automation  | Web Automation         | Python, Selenium, Pytest, ChromeDriver                      | Automated Login, search, and checkout workflows                          |
| 2 | Robot Framework             | Keyword-Driven Testing | Robot Framework, Python, Selenium Library, Requests Library | Built reusable keyword-driven test suites for web & API testing          |
| 3 | Pytest Automation Framework | Web Testing Framework  | Pythan, Pytest, Selenium                                    | Built reusable scripts with automated HTML reporting                     |
| 4 | Rest API Automation         | Manual Testing         | Flask, Pytest, Postman, Robot Framework                     | Built and tested a complete food ordering REST API system by using Flask |

# Objective

To automate end-to-end testing of an e-commerce web application using Selenium with both Pytest and Robot Framework to build scalable and reusable automation frameworks.

## Key Activities

- Designed automation using Pytest and Robot Framework
- Automated end-to-end e-commerce scenarios (Registration, Login, Search, Cart, Logout)
- Implemented Page Object Model (POM) with reusable components
- Performed data-driven testing with external data
- Used fixtures, parameterization, and assertions for reliable testing.
- Captured screenshots for failed test cases and generated HTML reports
- Enabled command-line execution for CLI-ready automation

# Tools & Technologies

Python, Selenium WebDriver, Pytest, Robot Framework, SeleniumLibrary,  
HTML Reports

## Application Under Test:

<https://demo.nopcommerce.com/>  
NopCommerce Store

## Outcome

- Complete automation of the entire web-app without human intervention
- Built two industry-standard automation frameworks
- Generated structured HTML reports with screenshot evidence
- Improved test efficiency, maintainability, and reliability

# Pytest framework: file structure

The image shows a file explorer on the left and a code editor on the right. The file explorer displays the project structure:

- CAPSTONE PROJECT
  - pytest framework
    - .pytest\_cache
    - config
    - data
    - pages
    - reports
  - tests
    - \_\_pycache\_\_
    - reports
    - screenshots
    - \_\_init\_\_.py
    - conftest.py
    - test\_app.py
  - utilities
  - pytest.ini
  - reports
  - robot framework
  - requirements.txt

The code editor shows the content of `test_app.py`:

```
from pages.cart_page import CartPage
@pytest.mark.usefixtures("setup")
class TestNopCommerce:
    def _ensure_home(self):
        """Robustly navigates to Home."""
        try:
            current = self.driver.current_url
            base = self.driver.current_url.
        if current != base:
            self.driver.get(self.data['
        WebDriverWait(self.driver, 10).
            EC.visibility_of_element_lo
        )
    except:
        self.driver.refresh()
```

# **Project 1: Pytest Framework**

## **Objective :**

Performed UI automation testing on an e-commerce web application using Selenium with Python and Pytest to design, execute, and report automated test cases following industry-standard practices

## **Key Objective :**

- Built a Page Object Model (POM) based framework with reusable page classes and modular test structure.
- Automated end-to-end test scenarios covering login, product search, add to cart, cart update, and logout.
- Configured test data externally using CSV files and managed environment settings via config.ini
- Generated HTML test execution reports using pytest-html with pass / fail details

# Pytest Execution Workflow



# Test Execution Console

The screenshot shows a code editor interface with multiple tabs open. The active tab is `test_app.py`, which contains Python test code for a web application. The code includes imports for `pytest`, `selenium.webdriver.common`, `selenium.webdriver.support.ui`, and various page objects. A test class `TestNopCommerce` is defined with a method `_ensure_home`. The terminal below the editor shows the output of running the tests, including the session start, configuration details, and the results of each test case.

```
File: test_app.py
1 import pytest
2 import time
3 from selenium.webdriver.common import By
4 from selenium.webdriver.support.ui import WebDriverWait
5 from selenium.webdriver.support import expected_conditions as EC
6 from selenium.common.exceptions import TimeoutException, NoSuchElementException, StaleElementReferenceException
7 from pages.home_page import HomePage
8 from pages.login_page import LoginPage
9 from pages.cart_page import CartPage
10
11 @pytest.mark.usefixtures("setup")
12 class TestNopCommerce:
13
14     def _ensure_home(self):
15         """Robustly navigates to Home."""
16         try:
17             current = self.driver.current_url.split('?')[0].rstrip('/')
18
19     def test_01_nav_reg(self):
20         self._ensure_home()
21         self.login_page.open()
22         self.login_page.register_new_user("testuser123@gmail.com", "1234567890")
23         self.login_page.logout()
24         self._ensure_home()
25         self.home_page.open()
26         self.home_page.verify_login_successful()
27
28     def test_02_reg_user(self):
29         self._ensure_home()
30         self.login_page.open()
31         self.login_page.register_new_user("testuser123@gmail.com", "1234567890")
32         self.login_page.logout()
33         self._ensure_home()
34         self.home_page.open()
35         self.home_page.verify_login_successful()
36
37     def test_03_search_1(self):
38         self._ensure_home()
39         self.home_page.open()
40         self.home_page.search_product("Laptops")
41         self.home_page.verify_search_results("Laptops")
42
43     def test_04_search_2(self):
44         self._ensure_home()
45         self.home_page.open()
46         self.home_page.search_product("Smartphones")
47         self.home_page.verify_search_results("Smartphones")
48
49     def test_05_search_3(self):
50         self._ensure_home()
51         self.home_page.open()
52         self.home_page.search_product("Tablets")
53         self.home_page.verify_search_results("Tablets")
54
55     def test_06_logout_session(self):
56         self._ensure_home()
57         self.login_page.open()
58         self.login_page.register_new_user("testuser123@gmail.com", "1234567890")
59         self.login_page.logout()
60         self._ensure_home()
61         self.home_page.open()
62         self.home_page.verify_login_successful()
63
64     def test_07_update_cart(self):
65         self._ensure_home()
66         self.home_page.open()
67         self.home_page.search_product("Laptops")
68         self.home_page.verify_search_results("Laptops")
69         self.home_page.add_product_to_cart("Laptops")
70         self.home_page.verify_product_in_cart("Laptops")
71
72     def test_08_add_to_cart(self):
73         self._ensure_home()
74         self.home_page.open()
75         self.home_page.search_product("Smartphones")
76         self.home_page.verify_search_results("Smartphones")
77         self.home_page.add_product_to_cart("Smartphones")
78         self.home_page.verify_product_in_cart("Smartphones")
79
80     def test_09_product_search(self):
81         self._ensure_home()
82         self.home_page.open()
83         self.home_page.search_product("Tablets")
84         self.home_page.verify_search_results("Tablets")
85
86     def test_10_product_search(self):
87         self._ensure_home()
88         self.home_page.open()
89         self.home_page.search_product("Tablets")
90         self.home_page.verify_search_results("Tablets")
91
92     def test_11_product_search(self):
93         self._ensure_home()
94         self.home_page.open()
95         self.home_page.search_product("Tablets")
96         self.home_page.verify_search_results("Tablets")
97
98     def test_12_product_search(self):
99         self._ensure_home()
100        self.home_page.open()
101        self.home_page.search_product("Tablets")
102        self.home_page.verify_search_results("Tablets")
103
104    def test_13_product_search(self):
105        self._ensure_home()
106        self.home_page.open()
107        self.home_page.search_product("Tablets")
108        self.home_page.verify_search_results("Tablets")
109
110    def test_14_product_search(self):
111        self._ensure_home()
112        self.home_page.open()
113        self.home_page.search_product("Tablets")
114        self.home_page.verify_search_results("Tablets")
115
116    def test_15_product_search(self):
117        self._ensure_home()
118        self.home_page.open()
119        self.home_page.search_product("Tablets")
120        self.home_page.verify_search_results("Tablets")
121
122    def test_16_product_search(self):
123        self._ensure_home()
124        self.home_page.open()
125        self.home_page.search_product("Tablets")
126        self.home_page.verify_search_results("Tablets")
127
128    def test_17_product_search(self):
129        self._ensure_home()
130        self.home_page.open()
131        self.home_page.search_product("Tablets")
132        self.home_page.verify_search_results("Tablets")
133
134    def test_18_product_search(self):
135        self._ensure_home()
136        self.home_page.open()
137        self.home_page.search_product("Tablets")
138        self.home_page.verify_search_results("Tablets")
139
140    def test_19_product_search(self):
141        self._ensure_home()
142        self.home_page.open()
143        self.home_page.search_product("Tablets")
144        self.home_page.verify_search_results("Tablets")
145
146    def test_20_product_search(self):
147        self._ensure_home()
148        self.home_page.open()
149        self.home_page.search_product("Tablets")
150        self.home_page.verify_search_results("Tablets")
151
152    def test_21_product_search(self):
153        self._ensure_home()
154        self.home_page.open()
155        self.home_page.search_product("Tablets")
156        self.home_page.verify_search_results("Tablets")
157
158    def test_22_product_search(self):
159        self._ensure_home()
160        self.home_page.open()
161        self.home_page.search_product("Tablets")
162        self.home_page.verify_search_results("Tablets")
163
164    def test_23_product_search(self):
165        self._ensure_home()
166        self.home_page.open()
167        self.home_page.search_product("Tablets")
168        self.home_page.verify_search_results("Tablets")
169
170    def test_24_product_search(self):
171        self._ensure_home()
172        self.home_page.open()
173        self.home_page.search_product("Tablets")
174        self.home_page.verify_search_results("Tablets")
175
176    def test_25_product_search(self):
177        self._ensure_home()
178        self.home_page.open()
179        self.home_page.search_product("Tablets")
180        self.home_page.verify_search_results("Tablets")
181
182    def test_26_product_search(self):
183        self._ensure_home()
184        self.home_page.open()
185        self.home_page.search_product("Tablets")
186        self.home_page.verify_search_results("Tablets")
187
188    def test_27_product_search(self):
189        self._ensure_home()
190        self.home_page.open()
191        self.home_page.search_product("Tablets")
192        self.home_page.verify_search_results("Tablets")
193
194    def test_28_product_search(self):
195        self._ensure_home()
196        self.home_page.open()
197        self.home_page.search_product("Tablets")
198        self.home_page.verify_search_results("Tablets")
199
200    def test_29_product_search(self):
201        self._ensure_home()
202        self.home_page.open()
203        self.home_page.search_product("Tablets")
204        self.home_page.verify_search_results("Tablets")
205
206    def test_30_product_search(self):
207        self._ensure_home()
208        self.home_page.open()
209        self.home_page.search_product("Tablets")
210        self.home_page.verify_search_results("Tablets")
211
212    def test_31_product_search(self):
213        self._ensure_home()
214        self.home_page.open()
215        self.home_page.search_product("Tablets")
216        self.home_page.verify_search_results("Tablets")
217
218    def test_32_product_search(self):
219        self._ensure_home()
220        self.home_page.open()
221        self.home_page.search_product("Tablets")
222        self.home_page.verify_search_results("Tablets")
223
224    def test_33_product_search(self):
225        self._ensure_home()
226        self.home_page.open()
227        self.home_page.search_product("Tablets")
228        self.home_page.verify_search_results("Tablets")
229
230    def test_34_product_search(self):
231        self._ensure_home()
232        self.home_page.open()
233        self.home_page.search_product("Tablets")
234        self.home_page.verify_search_results("Tablets")
235
236    def test_35_product_search(self):
237        self._ensure_home()
238        self.home_page.open()
239        self.home_page.search_product("Tablets")
240        self.home_page.verify_search_results("Tablets")
241
242    def test_36_product_search(self):
243        self._ensure_home()
244        self.home_page.open()
245        self.home_page.search_product("Tablets")
246        self.home_page.verify_search_results("Tablets")
247
248    def test_37_product_search(self):
249        self._ensure_home()
250        self.home_page.open()
251        self.home_page.search_product("Tablets")
252        self.home_page.verify_search_results("Tablets")
253
254    def test_38_product_search(self):
255        self._ensure_home()
256        self.home_page.open()
257        self.home_page.search_product("Tablets")
258        self.home_page.verify_search_results("Tablets")
259
260    def test_39_product_search(self):
261        self._ensure_home()
262        self.home_page.open()
263        self.home_page.search_product("Tablets")
264        self.home_page.verify_search_results("Tablets")
265
266    def test_40_product_search(self):
267        self._ensure_home()
268        self.home_page.open()
269        self.home_page.search_product("Tablets")
270        self.home_page.verify_search_results("Tablets")
271
272    def test_41_product_search(self):
273        self._ensure_home()
274        self.home_page.open()
275        self.home_page.search_product("Tablets")
276        self.home_page.verify_search_results("Tablets")
277
278    def test_42_product_search(self):
279        self._ensure_home()
280        self.home_page.open()
281        self.home_page.search_product("Tablets")
282        self.home_page.verify_search_results("Tablets")
283
284    def test_43_product_search(self):
285        self._ensure_home()
286        self.home_page.open()
287        self.home_page.search_product("Tablets")
288        self.home_page.verify_search_results("Tablets")
289
290    def test_44_product_search(self):
291        self._ensure_home()
292        self.home_page.open()
293        self.home_page.search_product("Tablets")
294        self.home_page.verify_search_results("Tablets")
295
296    def test_45_product_search(self):
297        self._ensure_home()
298        self.home_page.open()
299        self.home_page.search_product("Tablets")
300        self.home_page.verify_search_results("Tablets")
301
302    def test_46_product_search(self):
303        self._ensure_home()
304        self.home_page.open()
305        self.home_page.search_product("Tablets")
306        self.home_page.verify_search_results("Tablets")
307
308    def test_47_product_search(self):
309        self._ensure_home()
310        self.home_page.open()
311        self.home_page.search_product("Tablets")
312        self.home_page.verify_search_results("Tablets")
313
314    def test_48_product_search(self):
315        self._ensure_home()
316        self.home_page.open()
317        self.home_page.search_product("Tablets")
318        self.home_page.verify_search_results("Tablets")
319
320    def test_49_product_search(self):
321        self._ensure_home()
322        self.home_page.open()
323        self.home_page.search_product("Tablets")
324        self.home_page.verify_search_results("Tablets")
325
326    def test_50_product_search(self):
327        self._ensure_home()
328        self.home_page.open()
329        self.home_page.search_product("Tablets")
330        self.home_page.verify_search_results("Tablets")
```

===== test session starts =====

```
cachedir: .pytest_cache
metadata: {'Python': '3.12.6', 'Platform': 'Windows-10-10.0.19045-SP0', 'Packages': {'pytest': '9.0.2', 'pluggy': '1.6.0'}, 'Plugins': {'html': '4.2.0', 'metadata': '3.1.1', 'xdist': '3.8.0'}}
rootdir: C:\Users\aditya\Desktop\Wipro_pre_Skilling\Capstone Project\Capstone Project\pytest framework
configfile: pytest.ini
plugins: html-4.2.0, metadata-3.1.1, xdist-3.8.0
collected 50 items
```

```
tests/test_app.py::TestNopCommerce::test_01_nav_reg[C:\Users\aditya\Desktop\Wipro_pre_Skilling\Capstone Project\Capstone Project\pytest framework\data\test_data.csv] PASSED [ 2%]
tests/test_app.py::TestNopCommerce::test_02_reg_user[C:\Users\aditya\Desktop\Wipro_pre_Skilling\Capstone Project\Capstone Project\pytest framework\data\test_data.csv] PASSED [ 4%]
tests/test_app.py::TestNopCommerce::test_03_search_1[C:\Users\aditya\Desktop\Wipro_pre_Skilling\Capstone Project\Capstone Project\pytest framework\data\test_data.csv] FAILED [ 6%]
tests/test_app.py::TestNopCommerce::test_04_search_2[C:\Users\aditya\Desktop\Wipro_pre_Skilling\Capstone Project\Capstone Project\pytest framework\data\test_data.csv] PASSED [ 8%]
tests/test_app.py::TestNopCommerce::test_05_search_3[C:\Users\aditya\Desktop\Wipro_pre_Skilling\Capstone Project\Capstone Project\pytest framework\data\test_data.csv]
```

# Web Browser Output

## report.html

Report generated on 20-Feb-2026 at 11:55:38 by [pytest-html](#) v4.1.1

### Environment

|          |   |
|----------|---|
| Python   | 3.12.10   |
| Platform | Windows-11-10.0.26100-SP0   |
| Packages | <ul style="list-style-type: none"><li>• pytest: 8.1.1</li><li>• pluggy: 1.6.0</li></ul> |
| Plugins  | <ul style="list-style-type: none"><li>• html: 4.1.1</li><li>• metadata: 3.1.1</li></ul> |

### Summary

50 tests took 00:13:21.

(Un)check the boxes to filter the results.

0 Failed,  50 Passed,  0 Skipped,  0 Expected failures,  0 Unexpected passes,  0 Errors,  0 Reruns

[Show all details](#) / [Hide all details](#)

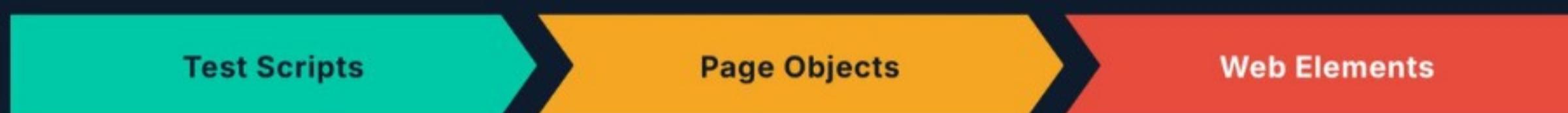
| Result  | Test  | Duration | Links |
|--|---|----------|-------|
| Passed   | tests/test_app.py::TestNopCommerce::test_01_nav_reg[C:\\\\Users\\\\Lenovo\\\\PycharmProjects\\\\Wipro-Training-2026\\\\Capstone Project\\\\pytest framework\\\\data\\\\test_data.csv]       | 00:00:17 |       |
| Passed   | tests/test_app.py::TestNopCommerce::test_02_reg_user[C:\\\\Users\\\\Lenovo\\\\PycharmProjects\\\\Wipro-Training-2026\\\\Capstone Project\\\\pytest framework\\\\data\\\\test_data.csv]      | 00:00:19 |       |
| Passed   | tests/test_app.py::TestNopCommerce::test_03_search_1[C:\\\\Users\\\\Lenovo\\\\PycharmProjects\\\\Wipro-Training-2026\\\\Capstone Project\\\\pytest framework\\\\data\\\\test_data.csv]      | 00:00:10 |       |
| Passed   | tests/test_app.py::TestNopCommerce::test_04_search_2[C:\\\\Users\\\\Lenovo\\\\PycharmProjects\\\\Wipro-Training-2026\\\\Capstone Project\\\\pytest framework\\\\data\\\\test_data.csv]      | 00:00:10 |       |
| Passed   | tests/test_app.py::TestNopCommerce::test_05_search_3[C:\\\\Users\\\\Lenovo\\\\PycharmProjects\\\\Wipro-Training-2026\\\\Capstone Project\\\\pytest framework\\\\data\\\\test_data.csv]      | 00:00:12 |       |
| Passed   | tests/test_app.py::TestNopCommerce::test_06_verify_apple[C:\\\\Users\\\\Lenovo\\\\PycharmProjects\\\\Wipro-Training-2026\\\\Capstone Project\\\\pytest framework\\\\data\\\\test_data.csv]  | 00:00:44 |       |
| Passed   | tests/test_app.py::TestNopCommerce::test_07_nav_computers[C:\\\\Users\\\\Lenovo\\\\PycharmProjects\\\\Wipro-Training-2026\\\\Capstone Project\\\\pytest framework\\\\data\\\\test_data.csv] | 00:00:32 |       |

# Page Object Model

The Page Object Model (POM) is a test automation design pattern that represents each webpage as a separate class, storing UI elements and actions in one place. It separates test logic from locators, making tests reusable, readable, and easy to maintain when the UI changes.

## PAGE OBJECT MODEL (POM)

The design pattern that separates page representation from test logic



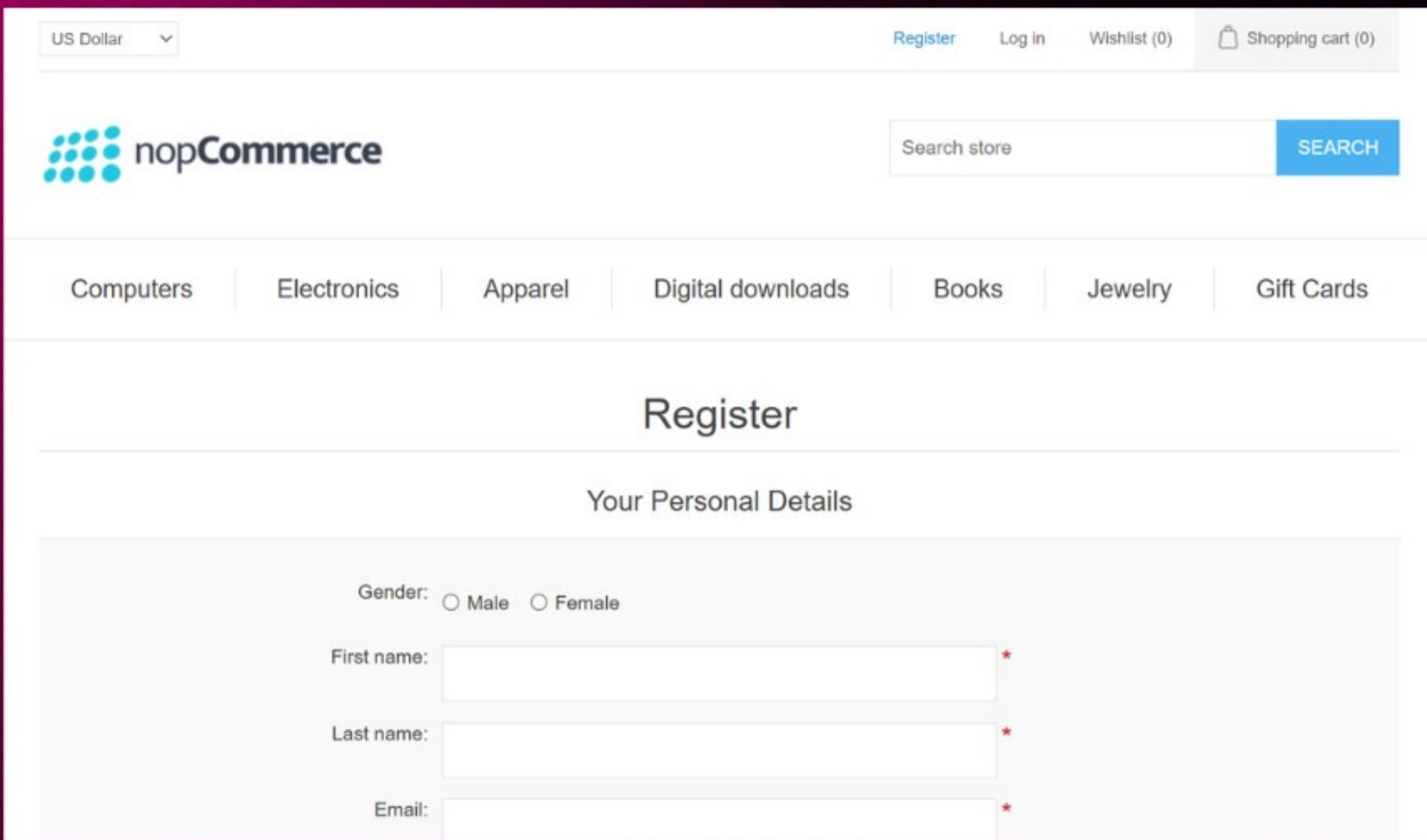
**1 Separation of Concerns** Test logic stays in test files; page interactions encapsulated in page classes

**2 Reusability** Same page methods reused across multiple tests without code duplication

**3 Maintainability** Locator changes made in one place — the page class — rather than in every test

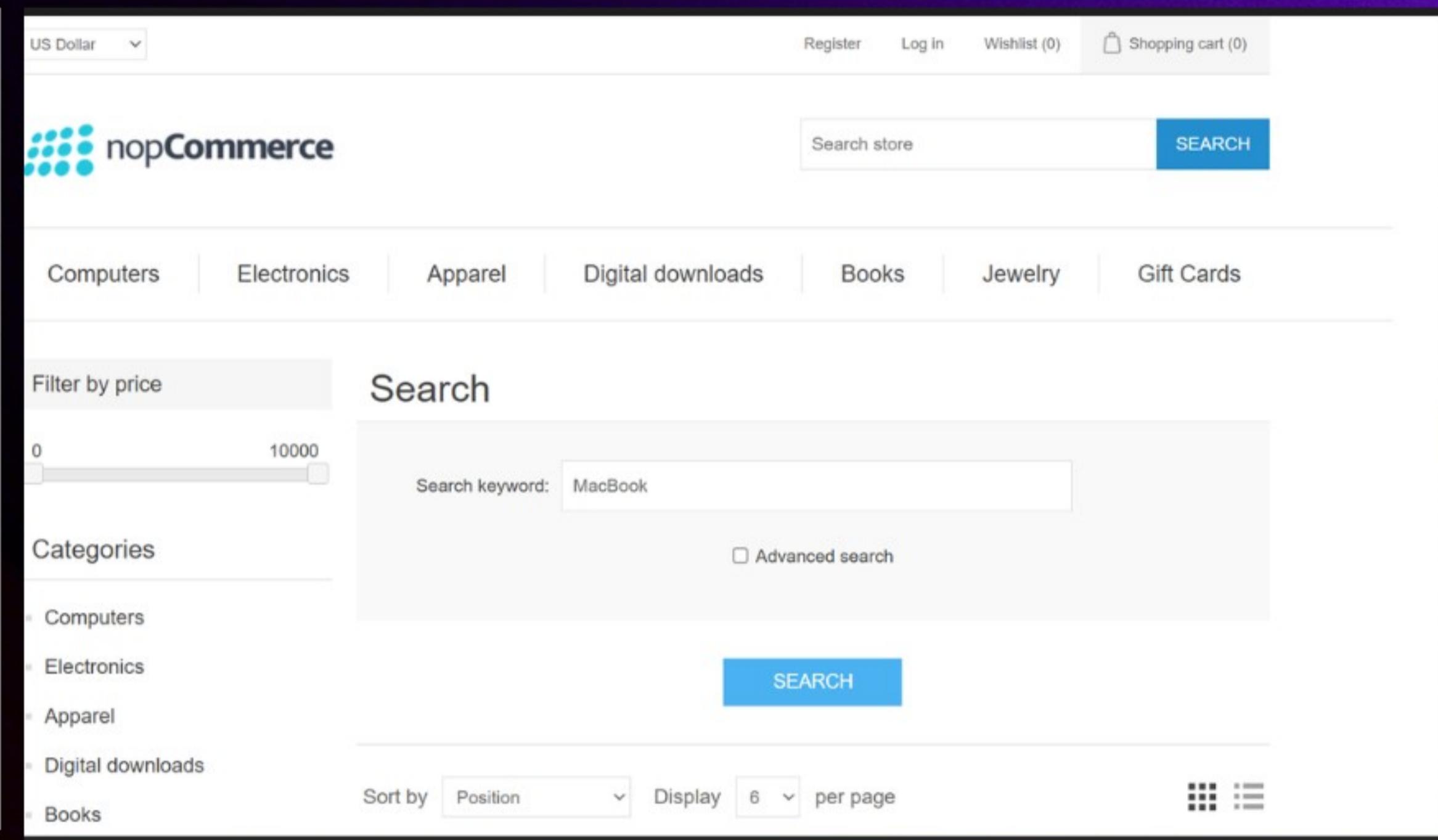
**4 Readability** Tests read like user stories: `login_page.enter_credentials()` is self-documenting

# User Registration



The screenshot shows the User Registration page of a nopCommerce website. At the top, there's a navigation bar with links for "Register", "Log in", "Wishlist (0)", and "Shopping cart (0)". Below the navigation is the nopCommerce logo. A search bar with the placeholder "Search store" and a blue "SEARCH" button are positioned above a horizontal menu bar. The menu bar includes links for "Computers", "Electronics", "Apparel", "Digital downloads", "Books", "Jewelry", and "Gift Cards". The main content area is titled "Register" and contains a section for "Your Personal Details". It includes fields for "Gender" (radio buttons for "Male" and "Female"), "First name" (text input field with a red asterisk), "Last name" (text input field with a red asterisk), and "Email" (text input field with a red asterisk). The entire registration form is set against a light gray background.

# Search Product



The screenshot shows the Search Product page of a nopCommerce website. At the top, there's a navigation bar with links for "Register", "Log in", "Wishlist (0)", and "Shopping cart (0)". Below the navigation is the nopCommerce logo. A search bar with the placeholder "Search store" and a blue "SEARCH" button are positioned above a horizontal menu bar. The menu bar includes links for "Computers", "Electronics", "Apparel", "Digital downloads", "Books", "Jewelry", and "Gift Cards". To the left of the search bar is a "Filter by price" slider ranging from 0 to 10000. Below the search bar is a "Categories" sidebar with links for "Computers", "Electronics", "Apparel", "Digital downloads", and "Books". The main search area has a "Search keyword" input field containing "MacBook", a "SEARCH" button, and an "Advanced search" checkbox. At the bottom of the page are buttons for "Sort by Position" and "Display 6 per page", along with a grid and list view switcher icon.

Registers a user with valid/invalid inputs and verifies successful login, then searches for a product and validates its search results and product details.

# Add to Cart

The screenshot shows a product page for an Apple MacBook Pro. At the top, there's a green header bar with the text "1 item has been added to your shopping cart". Below the header is the nopCommerce logo. A navigation menu includes links for Computers, Electronics, Apparel, Digital downloads, Books, and Jewelry. The main content area features a large image of the laptop, its name "Apple MacBook Pro", a short description, a 5-star rating with 1 review, manufacturer information, SKU, shipping details, price (\$1,800.00), and a quantity selector set to 2. A prominent blue "ADD TO CART" button is located at the bottom right of the product details.

# Remove Items

The screenshot shows a shopping cart page. At the top, there's a dropdown for "US Dollar". The main content area displays the message "Your Shopping Cart is empty!". Below this, there are links for Information, Customer service, My account, and Follow us. The footer of the page also features the nopCommerce logo and navigation links for Computers, Electronics, Apparel, Digital downloads, Books, and Jewelry.

Navigates to a product page, adds the item to the cart and verifies its name, quantity, and price, then updates the quantity, checks the total price, removes the item, and confirms the cart is empty.

# User Logout & Session Validation

 **nopCommerce**

Search store

---

Computers | Electronics | Apparel | Digital downloads | Books | Jewelry | Gift Cards



## iPhone 16

An upgraded A18 chip that supports Apple Intelligence, a dual-lens camera system that takes great photos, a Camera Control button for quick camera access, and a customizable Action button.

[Learn More](#)

CLICKS LOGOUT FROM THE ACCOUNT MENU AND VERIFIES THE SESSION IS TERMINATED BY ASSERTING THE USER IS REDIRECTED TO THE HOMEPAGE WITHOUT ANY AUTHENTICATED ACCESS.

# Project 2: Robot Framework

## Objective:

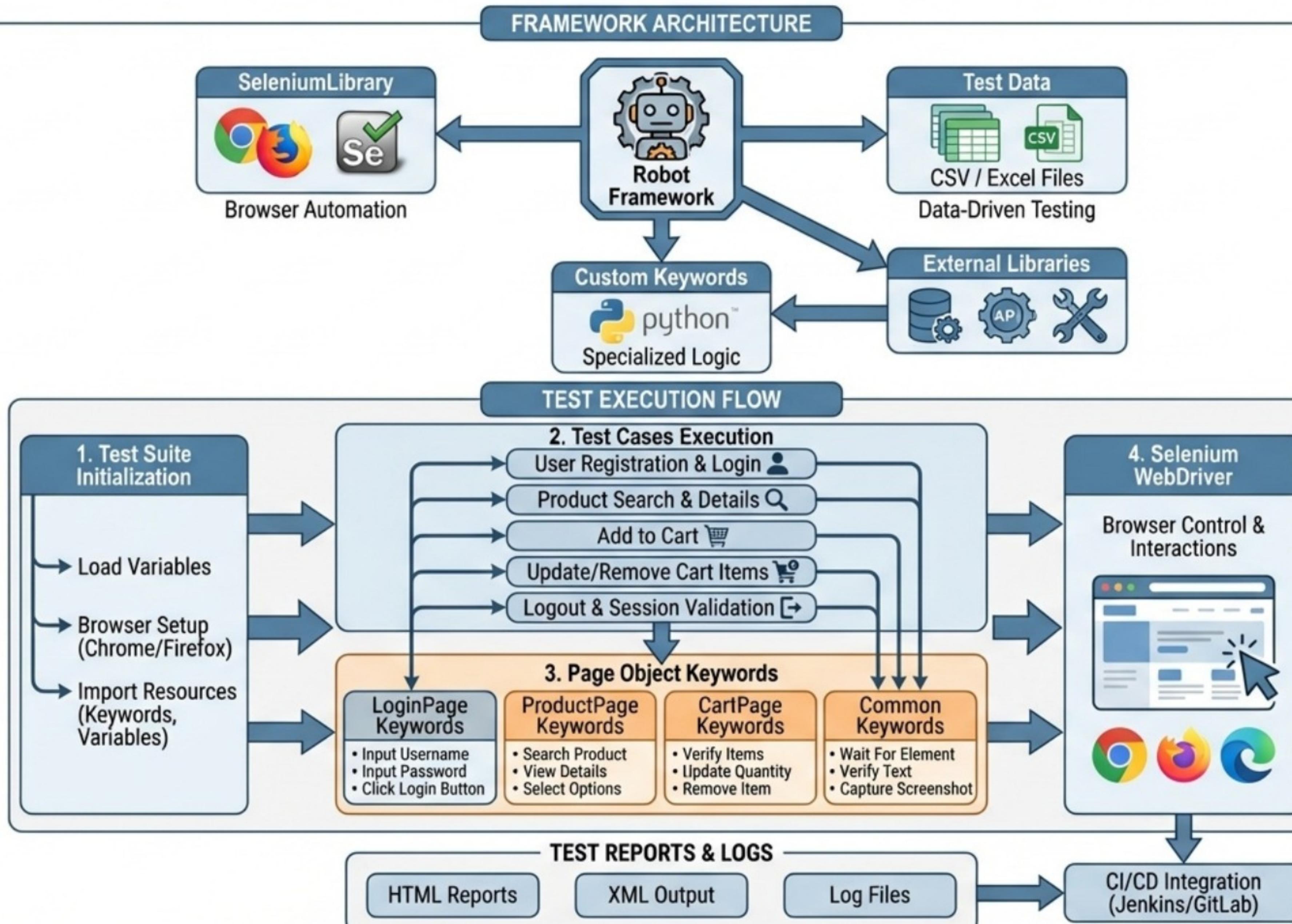
Performed end-to-end UI automation testing on an e-commerce web application using Selenium with Robot Framework, implementing a keyword-driven and reusable test architecture.

## Key Objective:

- Implement keyword-driven automation
- Automate end-to-end e-commerce scenarios
- Use SeleniumLibrary for UI interaction
- Enable data-driven testing
- Generate reports and capture failure screenshots
- Support command-line execution for CLI



# Robot Framework Workflow Diagram for E-Commerce UI Automation



# Test Execution Console

```
<> robot framework\report.html    R shop.robot x  Python run_all_robot.py    <> tests\report.html    Python test_app.py    <> reports\report.html    test_data_user v : 6 ^ ^

1  *** Settings ***
2  Documentation    Runs 25 Test Cases in a single browser session per CSV file.
3  Resource          ..../resources/common.resource
4  Library           BuiltIn
5
6  Suite Setup       Setup Everything    ${CSV_PATH}
7  Suite Teardown    Finalize Execution
8  Test Teardown     Log Test Result
9
10 *** Variables ***
11 ${CSV_PATH}        ${CURDIR}/../../pytest framework/data/test_data.csv
12
13 ▷ *** Test Cases ***
14 ▷ TC_01 Register Navigation
15      Ensure Home
```

Terminal Local x Command Prompt x

```
python is not recognized as an internal or external command,
operable program or batch file.
```

```
C:\Users\User\Downloads\Capstone Project\Capstone Project\robot framework\tests>python -m robot shop.robot
=====
Shop :: Runs 25 Test Cases in a single browser session per CSV file.
=====
TC_01 Register Navigation | PASS |
-----
TC_02 Register User | PASS |
```

# Web Browser Output

## Shop Report

Generated  
20260220 14:57:21 UTC+05:30  
2 hours 28 minutes ago

### Summary

|                |  |
|----------------|--|
| Status:        | 1 test failed  |
| Documentation: | Runs 25 Test Cases in a single browser session per CSV file. |
| Start Time:    | 20260220 14:55:20.957  |
| End Time:      | 20260220 14:57:20.990  |
| Elapsed Time:  | 00:02:00.033   |
| Log File:      | <a href="#">log.html</a>                                     |

### Statistics

| Total Statistics    |  | Total | Pass | Fail | Skip | Elapsed  | Pass / Fail / Skip  |
|---------------------|--|-------|------|------|------|----------|---|
| All Tests           |  | 25    | 24   | 1    | 0    | 00:01:51 | <div style="width: 96%; background-color: #28a745; height: 10px;"></div>  |
| <hr/>               |  |       |      |      |      |          |   |
| Statistics by Tag   |  | Total | Pass | Fail | Skip | Elapsed  | Pass / Fail / Skip  |
| No Tags             |  |       |      |      |      |          | <div style="width: 100%; background-color: #007bff; height: 10px;"></div> |
| Statistics by Suite |  | Total | Pass | Fail | Skip | Elapsed  | Pass / Fail / Skip  |
| Shop                |  | 25    | 24   | 1    | 0    | 00:02:00 | <div style="width: 96%; background-color: #28a745; height: 10px;"></div>  |

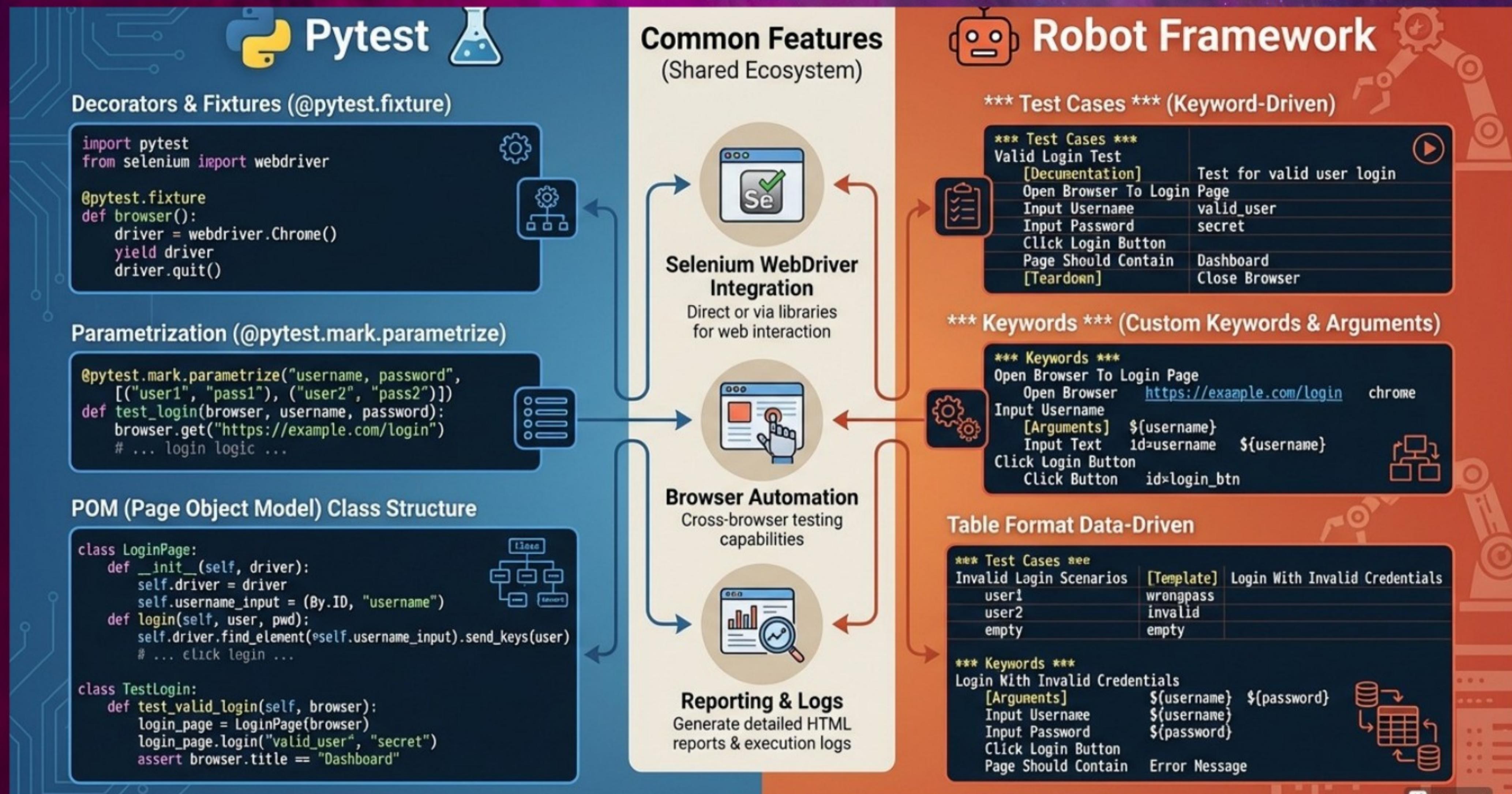
### Details

| All         | Tags   | Suites | Search |
|-------------|--|--------|--------|
| Status:     | 25 tests total, 24 passed, 1 failed, 0 skipped |        |        |
| Total Time: | 00:01:50.613                                   |        |        |

| Name                           | Documentation | Tags | Status | Message   | Elapsed      | Start / End                                    |
|--------------------------------|---------------|------|--------|---|--------------|--|
| Shop.TC_02 Register User       |               |      | FAIL   | Registration did not navigate to result page: First name is required. | 00:01:08.360 | 20260220 14:55:32.415<br>20260220 14:56:40.775 |
| Shop.TC_01 Register Navigation |               |      | PASS   |   | 00:00:02.488 | 20260220 14:55:29.926<br>20260220 14:55:32.414 |
| Shop.TC_03 Search Query 1      |               |      | PASS   |   | 00:00:01.578 | 20260220 14:56:40.775                          |

The application blocked registration because the First Name field was empty / not accepted, so it never reached the success page.

# Comparision between Pytest and robot framework





# FOODIE APP

Features & REST API Requirement List



Flask



Pytest



Robot Framework



Postman

# Problem Statement

*“How can a food ordering platform ensure reliable, scalable, and fully tested REST APIs that handle restaurant management, order processing, and customer interactions without defects in production?”*



## Untested Endpoints

REST APIs lack automated verification, leading to undetected bugs reaching production environments.



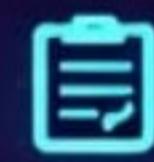
## Manual Testing Gaps

Relying purely on manual Postman checks is time-consuming and prone to human error.



## No Regression Coverage

Without automated test suites, every code change risks breaking existing functionality.



## Data Integrity Issues

Missing validation on request/response bodies causes inconsistent data and failed integrations.

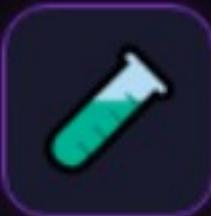
# **Project 3: Foodie App**

## **Objective :**

To design a Flask-based Foodie App REST API and implement automated testing using Pytest and Robot Framework to ensure reliability, accuracy, and scalable API validation.

## **Key Objectives :**

- Implement RESTful endpoints using Flask.
- Build API Automation Framework Using Pytest.
- Implement Keyword-Driven Testing Using Robot Framework.
- Ensure End-to-End Workflow Testing.



# API Testing Approach



3-Layer Verification



## Manual Testing

POSTMAN

- ✓ Validate individual request & response payloads
- ✓ Verify HTTP status codes correctness
- ✓ Exploratory testing of positive & negative scenarios



Ad-hoc Checks



## Pytest Automation

PYTHON + REQUESTS

- ✓ Programmatic validation using `requests` library
- ✓ Deep validation of response body & JSON schema
- ✓ Advanced usage of fixtures and parameterization



Component Integration



## Robot Framework

AUTOMATION SUITE

- ✓ High-level testing via `RequestsLibrary`
- ✓ Keyword-driven & data-driven test cases
- ✓ Managed test lifecycle with Setup & Teardown



End-to-End Flows

# Pytest Execution Console

The screenshot shows a PyCharm interface. On the left, a code editor window titled 'test\_app.py' displays Python test code. On the right, a terminal window titled 'Command Prompt' shows the command 'python -m pytest test\_app.py' being run, followed by the pytest output indicating 18 passed tests in 7.42 seconds.

```
test_app.py
> import ...

LOG_FILE = "test_app_results.txt"
BASE_URL = "http://127.0.0.1:5000/api/v1"

# Initialize log file
with open(LOG_FILE, "w") as f:
    f.write("TEST APP EXECUTION LOG\n=====\n")

def log_to_file(test_name, status): 18 usages
    with open(LOG_FILE, "a") as f:
        f.write(f"{test_name}: {status}\n")

Terminal Local × Command Prompt × Command Prompt ×
env456) C:\Users\Lenovo\PycharmProjects\Wipro-Training-2026\FoodieApp>python -m pytest test_app.py
===== test session starts =====
platform win32 -- Python 3.10.11, pytest-9.0.2, pluggy-1.6.0
rootdir: C:\Users\Lenovo\PycharmProjects\Wipro-Training-2026\FoodieApp
plugins: xdist-3.8.0
collected 18 items

test_app.py ......

===== 18 passed in 7.42s =====

env456) C:\Users\Lenovo\PycharmProjects\Wipro-Training-2026\FoodieApp>
```

# Web Browser Output

The screenshot shows a web-based test report titled 'Restaurant Tests Report'. The top right corner indicates the report was generated at 2026-02-20 19:17:53 UTC+05:30, 7 seconds ago. The page includes a 'Summary' section with basic test metadata and a detailed 'Statistics' section with three tables: 'Total Statistics', 'Statistics by Tag', and 'Statistics by Suite'. The 'Statistics by Suite' table shows 18 total tests, all passed, with 0 fails, 0 skips, and an elapsed time of 00:00:09. Below the statistics is a 'Details' section with tabs for 'All', 'Tags', 'Suites', and 'Search', and input fields for 'Suite', 'Test', and 'Include'.

### Restaurant Tests Report

Generate  
20260220 19:17:53 UTC+05:30  
7 seconds ago

#### Summary

|               |                          |
|---------------|--------------------------|
| Status:       | All tests passed         |
| Start Time:   | 20260220 19:17:44.361    |
| End Time:     | 20260220 19:17:53.256    |
| Elapsed Time: | 00:00:08.895             |
| Log File:     | <a href="#">log.html</a> |

#### Statistics

|                  | Total Statistics    | Total | Pass | Fail | Skip | Elapsed  | Pass / Fail / Skip  |
|------------------|---------------------|-------|------|------|------|----------|---|
| All Tests        |                     | 18    | 18   | 0    | 0    | 00:00:06 | <span style="background-color: #2e7131; color: white;">Success</span> |
|                  | Statistics by Tag   | Total | Pass | Fail | Skip | Elapsed  | Pass / Fail / Skip  |
| No Tags          |                     |       |      |      |      |          |   |
|                  | Statistics by Suite | Total | Pass | Fail | Skip | Elapsed  | Pass / Fail / Skip  |
| Restaurant Tests |                     | 18    | 18   | 0    | 0    | 00:00:09 | <span style="background-color: #2e7131; color: white;">Success</span> |

#### Details

All Tags Suites Search

Suite:

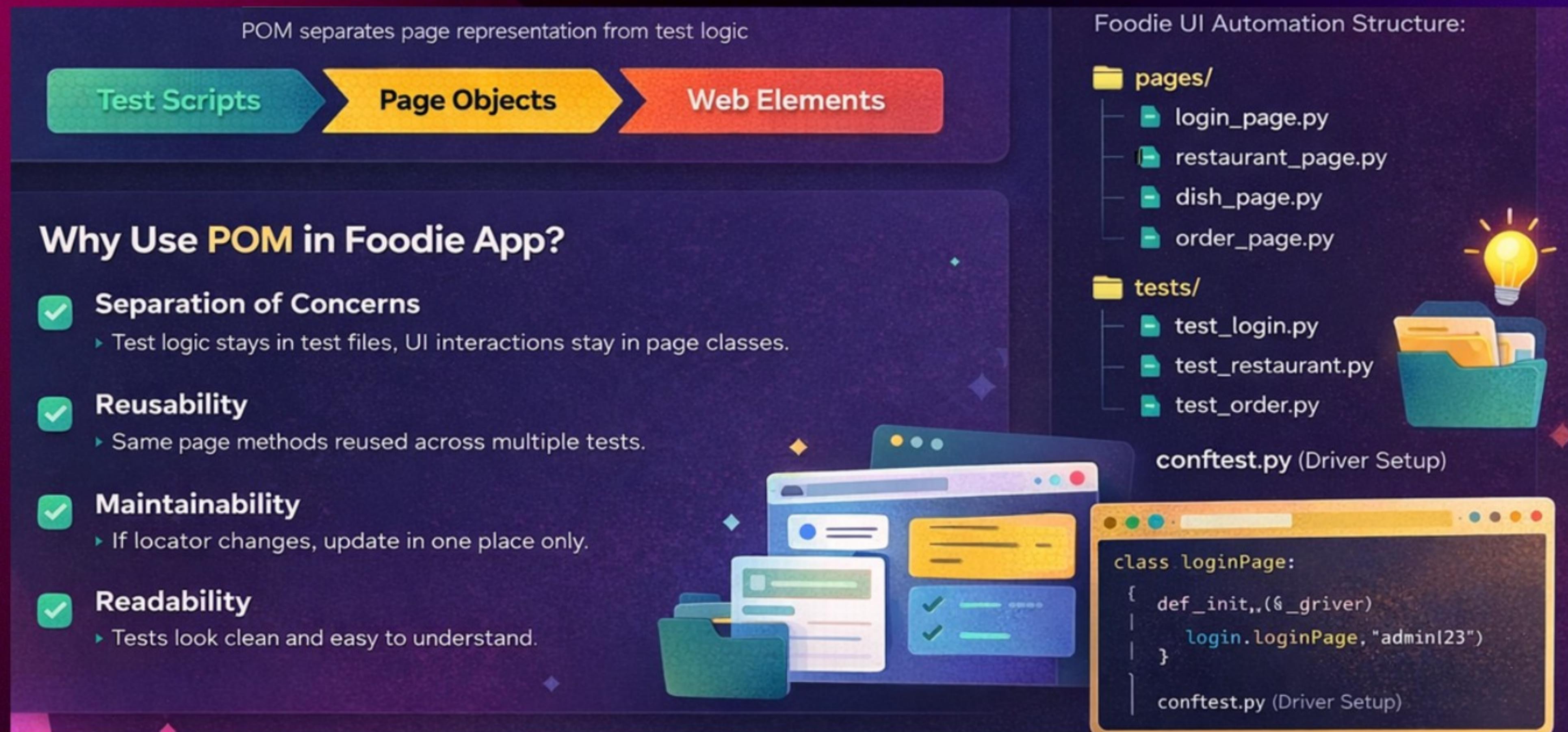
Test:

Include:

- Displays automated test execution in the console alongside generated browser-based reports, demonstrating real-time results and structured test outcome summaries

# Page Object Model

The Page Object Model (POM) is a design pattern in test automation that represents each web page as a separate class.



# Restaurant Module

## Restaurant Management :

- Register Restaurant: (Registers a new restaurant with details like name category, location, contact, images).
- Update Restaurant: (Modifies existing restaurant details).
- Disable Restaurant: (Marks a restaurant as disabled).
- View Restaurant Profile: (Retrieves details of a specific restaurant).

|   |                           |             |   |  |
|---|---------------------------|-------------|---|--|
| 1 | Register Restaurant       | <b>POST</b> | /api/v1/restaurants                         | 201 Created, 400 Bad Request, 409 Conflict |
| 2 | Update Restaurant Details | <b>PUT</b>  | /api/v1/restaurants/{restaurant_id}         | 200 OK, 404 Not Found                      |
| 3 | Disable Restaurant        | <b>PUT</b>  | /api/v1/restaurants/{restaurant_id}/disable | 200 OK, 404 Not Found                      |
| 4 | View Restaurant Profile   | <b>GET</b>  | /api/v1/restaurants/{restaurant_id}         | 200 OK, 404 Not Found                      |

# Dish Module

## Dish Module :

- Add Dish: (Adds a new dish with name, type, price, available time, image).
- Update Dish: (Modifies existing dish details).
- Enable/Disable Dish: (Changes a dish's availability status).
- Delete Dish: (Removes a dish from the system).

| # | Requirement           | Method        | URI  | Status Codes                 |
|---|-----------------------|---------------|--|------------------------------|
| 5 | Add Dish              | <b>POST</b>   | /api/v1/restaurants/{restaurant_id}/dishes | 201 Created, 400 Bad Request |
| 6 | Update Dish           | <b>PUT</b>    | /api/v1/dishes/{dish_id}                   | 200 OK, 404 Not Found        |
| 7 | Enable / Disable Dish | <b>PUT</b>    | /api/v1/dishes/{dish_id}/status            | 200 OK, 404 Not Found        |
| 8 | Delete Dish           | <b>DELETE</b> | /api/v1/dishes/{dish_id}                   | 200 OK, 404 Not Found        |

# Postman Collection - Restaurant & Dish Module

HTTP Foodie App API Automation / Restaurant Module / Register

POST http://127.0.0.1:5000/api/v1/restaurants

Body (8) **Body**  raw

```
1 {  
2   "name": "Food Hub",  
3   "category": "Veg",  
4   "location": "Delhi",  
5   "images": ["img1.jpg"],  
6   "contact": "9999999999"  
7 }  
8
```

201 CREATED • 114 ms • 351 B • Save Response

Body Cookies Headers (5) Test Results

{ } JSON ▾ Preview Visualize

```
1 {  
2   "category": "Veg",  
3   "contact": "9999999999",  
4   "dishes": [],  
5   "id": 1,  
6   "images": [  
7     "img1.jpg"  
8   ],
```

HTTP Foodie App API Automation / Dish Module / Add dish

POST http://127.0.0.1:5000/api/v1/restaurants/1/dishes

Body (8) **Body**  raw

```
1 {  
2   "name": "Pizza",  
3   "type": "Veg",  
4   "price": 250,  
5   "available_time": "10AM-10PM",  
6   "image": "pizza.jpg"  
7 }  
8
```

201 CREATED • 7 ms • 335 B • Save Response

Body Cookies Headers (5) Test Results

{ } JSON ▾ Preview Visualize

```
1 {  
2   "available_time": "10AM-10PM",  
3   "enabled": true,  
4   "id": 1,  
5   "image": "pizza.jpg",  
6   "name": "Pizza",  
7   "price": 250,  
8   "restaurant_id": 1,
```

Postman collection demonstrating successful Restaurant and Dish module API testing with CRUD operations.

# Admin Module

## Administrator Actions :

- Approve Restaurant: (Approves a restaurant).
- Disable Restaurant (Admin): (Disables a restaurant by admin).
- View Customer Feedback: (Retrieves a list of all customer feedback).
- View Order Status: (Retrieves a list of all orders).

| Requirement            | Method     | URI   | Status Codes          |
|------------------------|------------|---|-----------------------|
| Approve Restaurant     | <b>PUT</b> | /api/v1/admin/restaurants/{restaurant_id}/approve | 200 OK, 404 Not Found |
| Disable Restaurant     | <b>PUT</b> | /api/v1/admin/restaurants/{restaurant_id}/disable | 200 OK, 404 Not Found |
| View Customer Feedback | <b>GET</b> | /api/v1/admin/feedback                            | 200 OK                |
| View Order Status      | <b>GET</b> | /api/v1/admin/orders                              | 200 OK                |

# User Module

## User Interaction :

- User Registration: (Registers a new user with name, email, password).
- Search Restaurants: (Filters restaurants by name, location, dish, or rating).
- Place Order: (Creates a new order for a user, restaurant, and selected dishes).
- Give Rating: (Submits a rating and comment for a specific order).

| Requirement        | Method      | URI  | Status Codes                 |
|--------------------|-------------|--|------------------------------|
| User Registration  | <b>POST</b> | /api/v1/users/register                           | 201 Created, 409 Conflict    |
| Search Restaurants | <b>GET</b>  | /api/v1/restaurants/search?name=&location=&dish= | 200 OK                       |
| Place Order        | <b>POST</b> | /api/v1/orders                                   | 201 Created, 400 Bad Request |
| Give Rating        | <b>POST</b> | /api/v1/ratings                                  | 201 Created, 400 Bad Request |

# Postman Collection – Admin & User Module

The image shows two side-by-side Postman request panels.

**Left Panel: Admin Approval API**

- Method:** PUT
- URL:** http://127.0.0.1:5000/api/v1/admin/restaurants/1/approve
- Headers:** (7)
- Body:** (Raw JSON)
- ```
1 {  
2   "message": "Restaurant approved"  
3 }
```
- Response:** 200 OK

**Right Panel: User Registration API**

- Method:** POST
- URL:** http://127.0.0.1:5000/api/v1/users/register
- Headers:** (8)
- Body:** (Raw JSON)
- ```
1 {  
2   "name": "Harsh",  
3   "email": "harsh@test.com",  
4   "password": "123456"  
5 }  
6  
7 
```
- Response:** 201 CREATED

Postman collection demonstrating Admin approval and User registration APIs with successful 200 OK and 201 Created responses.

# Order Module

## Order Viewing :

- View Orders by Restaurant: (Retrieves orders associated with a specific restaurant).
- View Orders by User: (Retrieves orders placed by a specific user).

| Requirement               | Method     | URI  | Status Codes |
|---------------------------|------------|--|--------------|
| View Orders by Restaurant | <b>GET</b> | /api/v1/restaurants/{restaurant_id}/orders | 200 OK       |
| View Orders by User       | <b>GET</b> | /api/v1/users/{user_id}/orders             | 200 OK       |

# Postman Collection – Order Module

The screenshot shows the Postman application interface. On the left, the sidebar displays a collection structure under 'Foodie App API Automation'. The 'Order Module' section contains a POST request named 'Place order'. The main workspace shows the 'Place order' endpoint details:

- Method:** POST
- URL:** http://127.0.0.1:5000/api/v1/orders
- Headers:** (8)
- Body:** (Raw JSON)

```
1 {
2   "user_id": 1,
3   "restaurant_id": 1,
4   "dishes": []
5 }
```

The response section shows a successful 201 CREATED status with the following JSON body:

```
1 {
2   "dishes": [],
3   "id": 1,
4   "restaurant_id": 1,
5   "status": "placed",
6   "user_id": 1
7 }
```

# Summary

**18**

API Endpoints

**5**

Modules

**3**

Test Frameworks

**4**

HTTP Methods

- **Restaurant** registration, update, disable & profile management
- **Admin controls** approve, disable restaurants & view feedback
- **Order tracking** by restaurant or user
- **Dish CRUD** with enable/disable toggling
- **Customer** registration, search, order placement & ratings
- **Full test coverage** Manual (Postman), Pytest & Robot Framework

# Challenges Faced During the Project

- **Timeout & Synchronization Issues**
- **CAPTCHA Handling Issues**
- **Cross-Browser Compatibility Issues**
- **Internet Speed & Network Dependency**
- **System Performance & Resource Issues**

# MANUAL VS. AUTOMATION TESTING



## Manual TESTING

### POSTMAN

- Execution: Human-led
- Validate individual request & response payloads
- Reliability: Error Prone to error detection (Fatigue)
- Ideal For: Exploratory, UI/UX Testing (Short-term)

Human-Centric Checks



## AUTOMATION TESTING

### POSTMAN

- Execution: Tool-led (Scripts)
- Speed: Slower Consistent
- Keyword-driven vs. data-driven test cases
- Higher For Regression, Load

Efficient Test Integration

# Conclusion

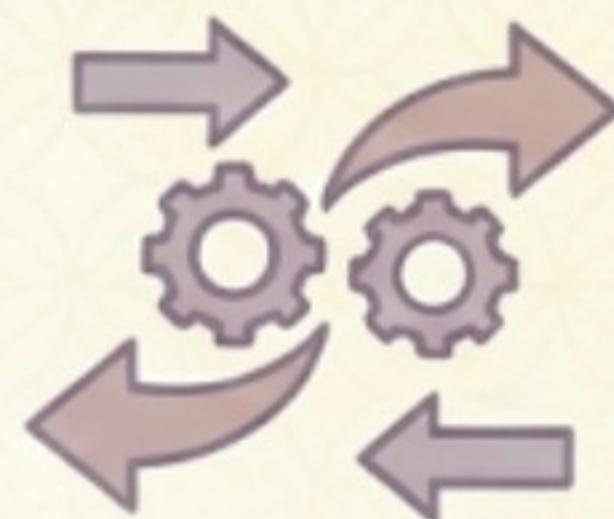
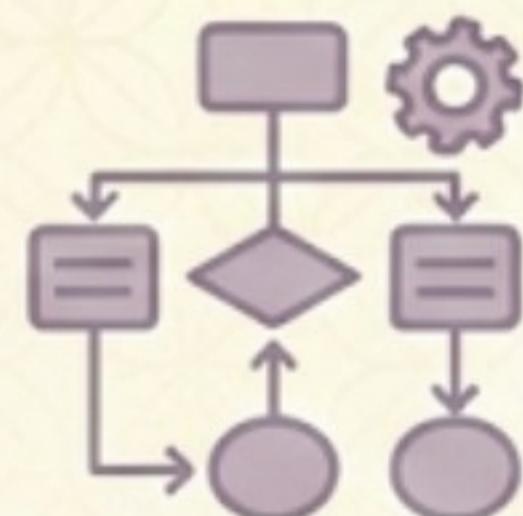
- Successfully architected and implemented modular, data-driven UI automation frameworks using Pytest and Robot Framework with Selenium.
- Engineered robust end-to-end UI test suites covering critical e-commerce workflows, including registration, product search, and cart management.
- Developed a functional REST API backend for a custom application using Flask, gaining hands-on understanding of server-side routing and HTTP status codes.
- Validated full-stack data integrity by executing comprehensive API tests using Python's requests library and Postman for standard HTTP methods.
- Overcame complex web automation challenges by implementing explicit waits, handling dynamic elements, and writing resilient locators.

- Integrated automated screenshot capture and detailed logging mechanisms within the test frameworks to ensure clear traceability and rapid defect isolation.
- Transitioned from static to dynamic test execution by integrating CSV data parsing, allowing for scalable, multi-scenario test coverage.
- Applied Object-Oriented Programming principles to implement the Page Object Model (POM) design pattern, significantly improving code maintainability.
- Bridged the gap between frontend user interactions and backend system architecture, demonstrating a comprehensive understanding of software quality assurance.
- Synthesized comprehensive QA methodologies into practical, production-ready capabilities, preparing for real-world automation testing roles.

# A Heartfelt Thank You to Our Mentor & Guide



Saritha R. Parthipan Ma'am



It is with immense gratitude and gratefulness that we extend our thanks to you for helping us through the successful completion of this project and the entire training program. You have been a source of both **inspiration and learning**. Your approachableness allowed us to be open and friendly, making the learning journey really memorable. You ensured that we are equipped with the knowledge required in the corporate world by moulding us through your engaging modules and lectures.



*~ Thank you Ma'am*

# Thank You!

