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

SUPERSET ID	NAME
4957579	Dibyojyoti Deb
4958879	Harsh Kumar Sinha
4958238	Aditya Raj
4956637	Anunay Kumar
4956997	Gaurav Kumar
4957628	Ankur Santosh Waghamode

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

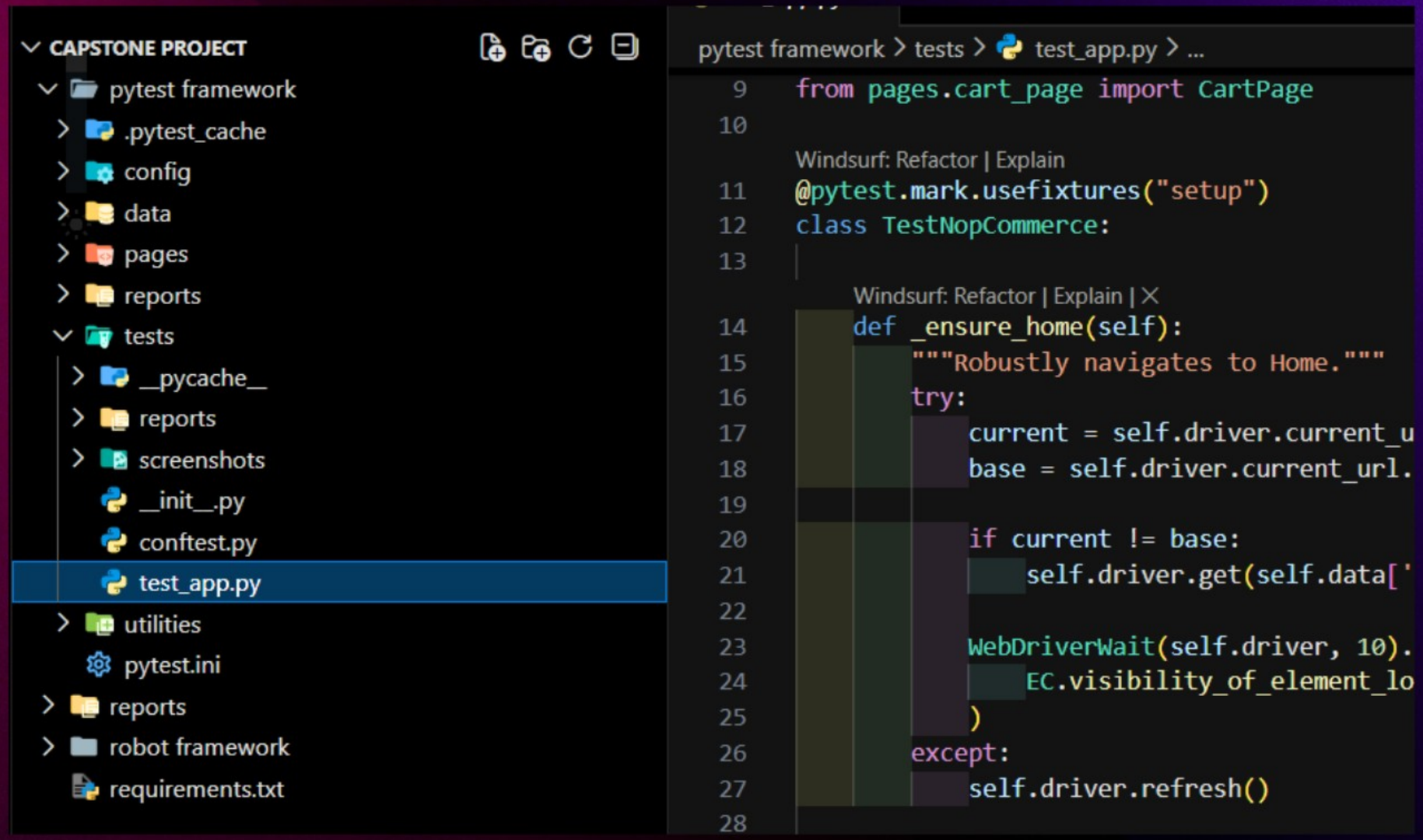
Application Under Test:

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



Project 1: Pytest Framework

Objective :

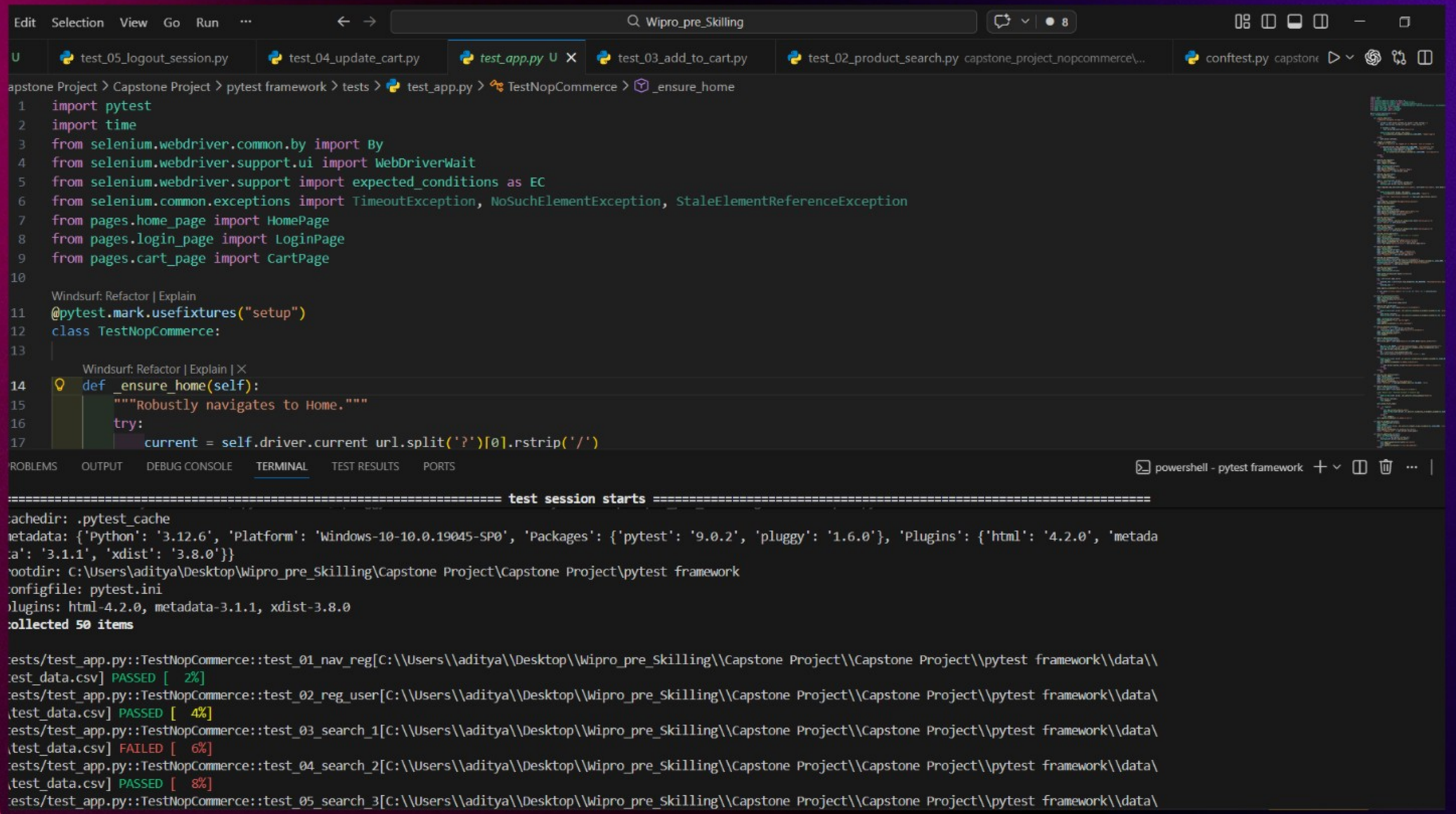
Performed UI automation testing on an e-commerce web application using Selenium with Python and Pytest to design, execute, and report automated test cases following industry-standard practices

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



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	• pytest: 8.1.1 • pluggy: 1.6.0
Plugins	• html: 4.1.1 • metadata: 3.1.1

Summary

50 tests took 00:13:21.

(Un)check the boxes to filter the results.

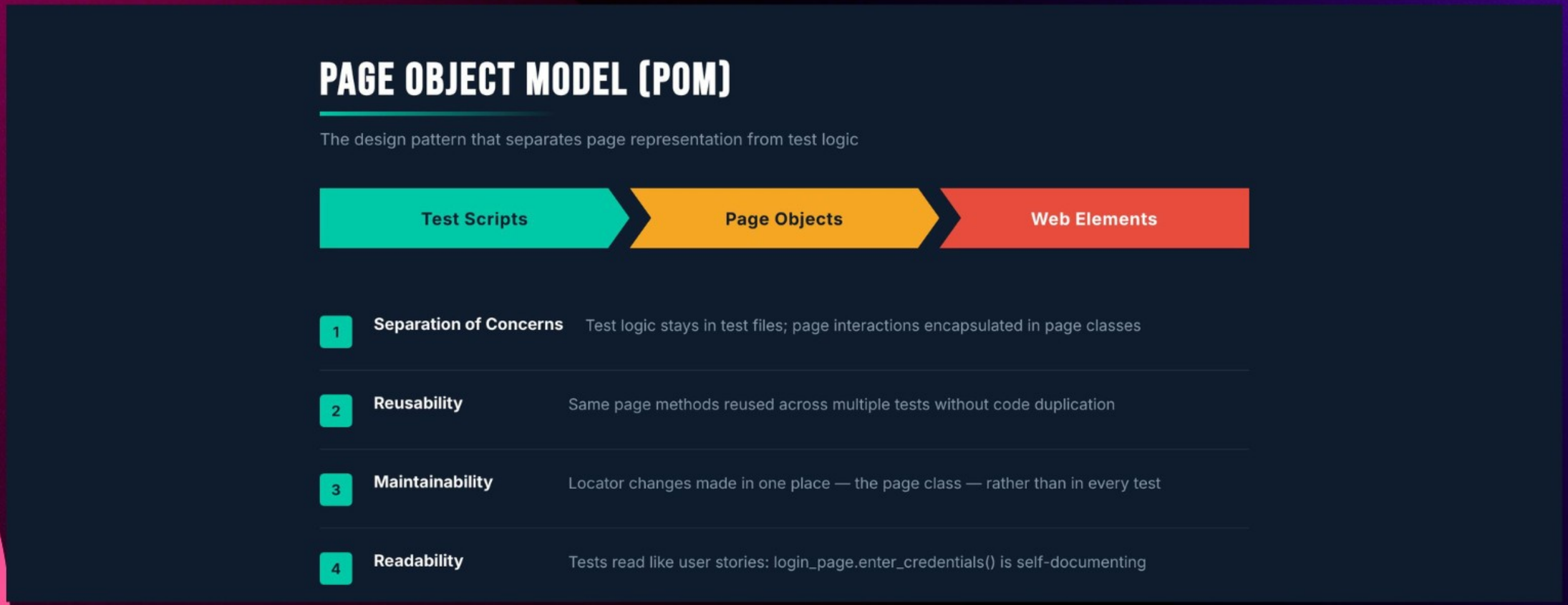
0 Failed, 50 Passed, 0 Skipped, 0 Expected failures, 0 Unexpected passes, 0 Errors, 0 Reruns

Show all details / Hide all details

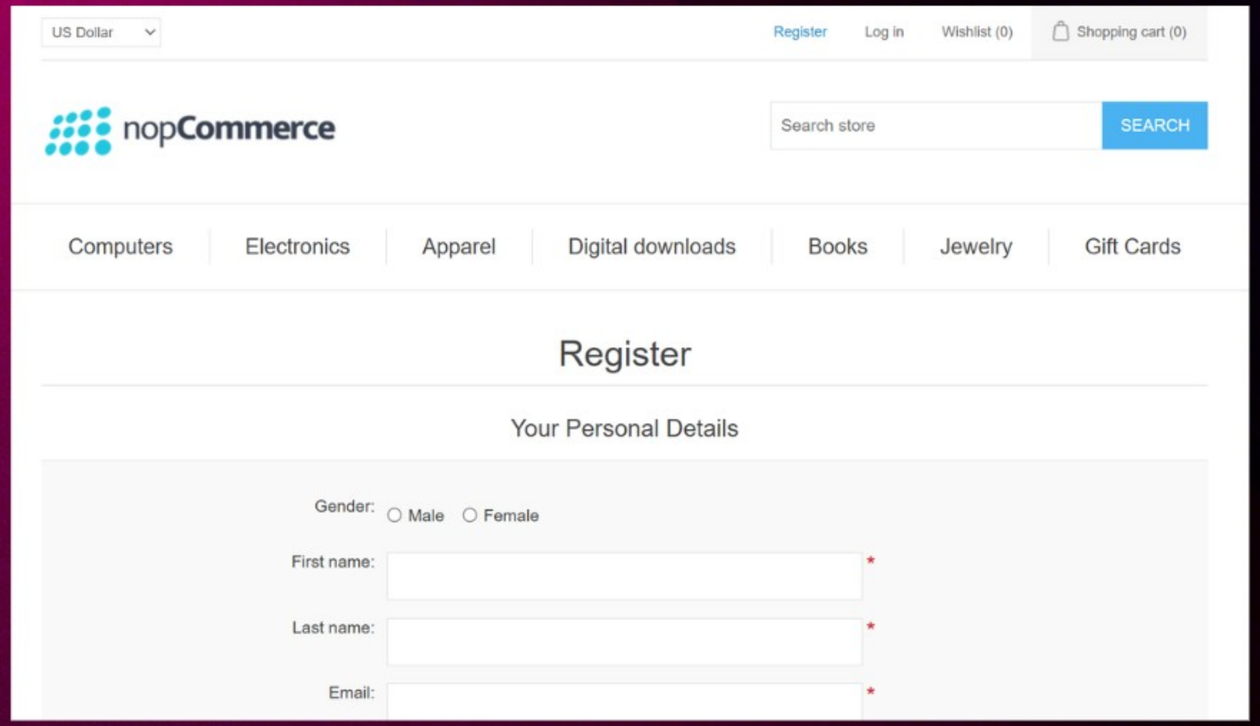
Result	Test	Duration	Links
Passed	tests/test_app.py::TestNopCommerce::test_01_nav_reg[C:\\Users\\Lenovo\\PycharmProjects\\Wipro-Training-2026\\Capstone Project\\pytest framework\\data\\test_data.csv]	00:00:17	
Passed	tests/test_app.py::TestNopCommerce::test_02_reg_user[C:\\Users\\Lenovo\\PycharmProjects\\Wipro-Training-2026\\Capstone 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

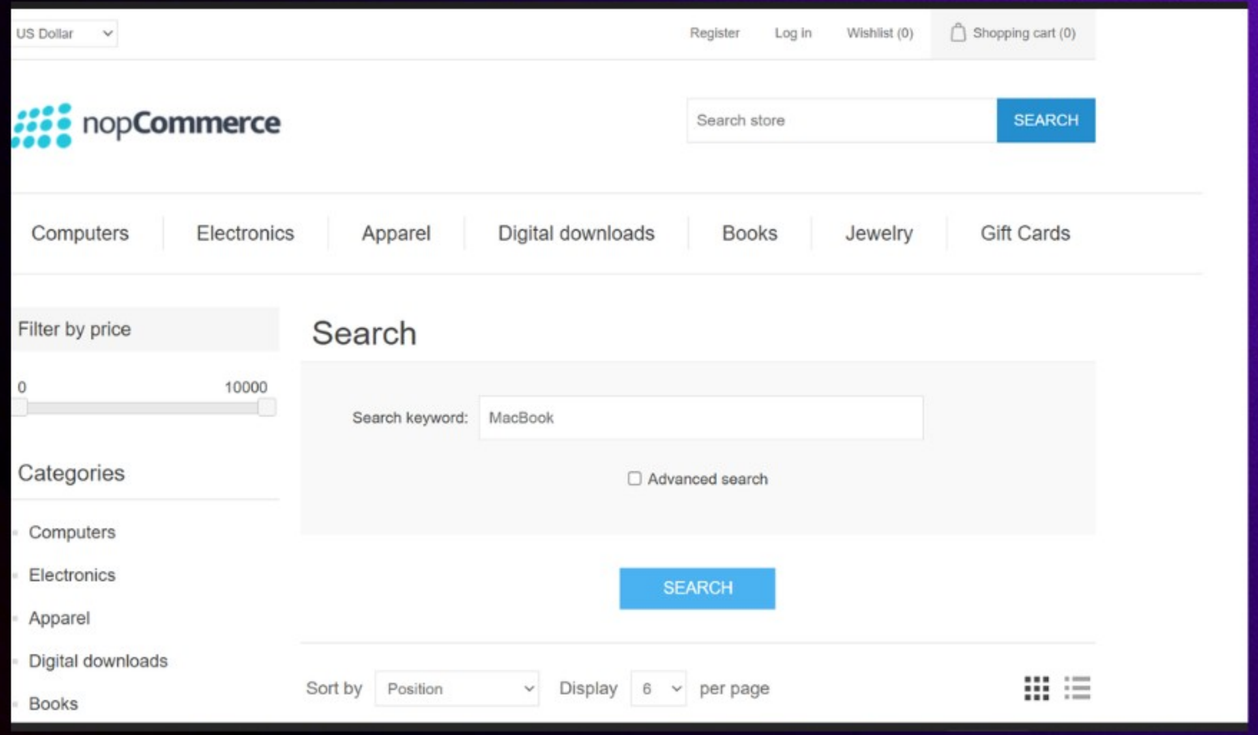
The Page Object Model (POM) is a test automation design pattern that represents each webpage as a separate class, storing UI elements and actions in one place. It separates test logic from locators, making tests reusable, readable, and easy to maintain when the UI changes.



User Registration

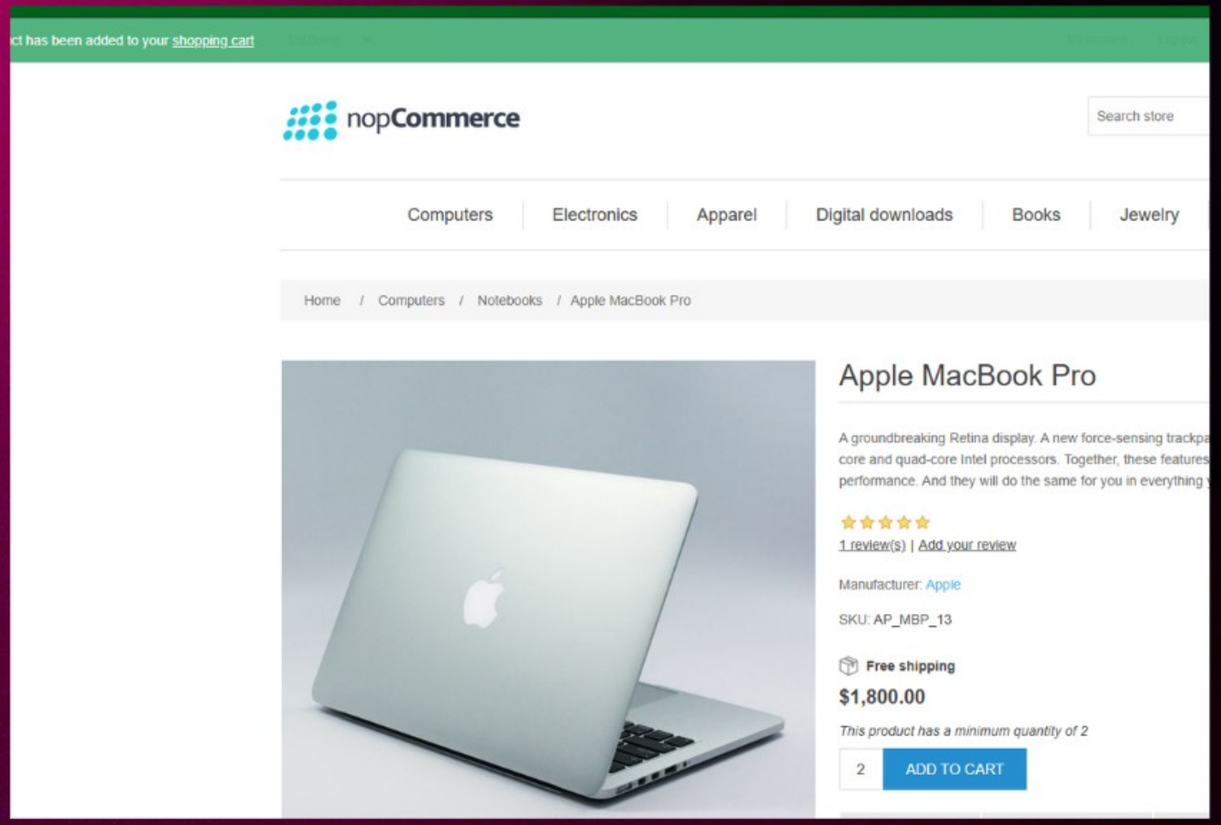


Search Product

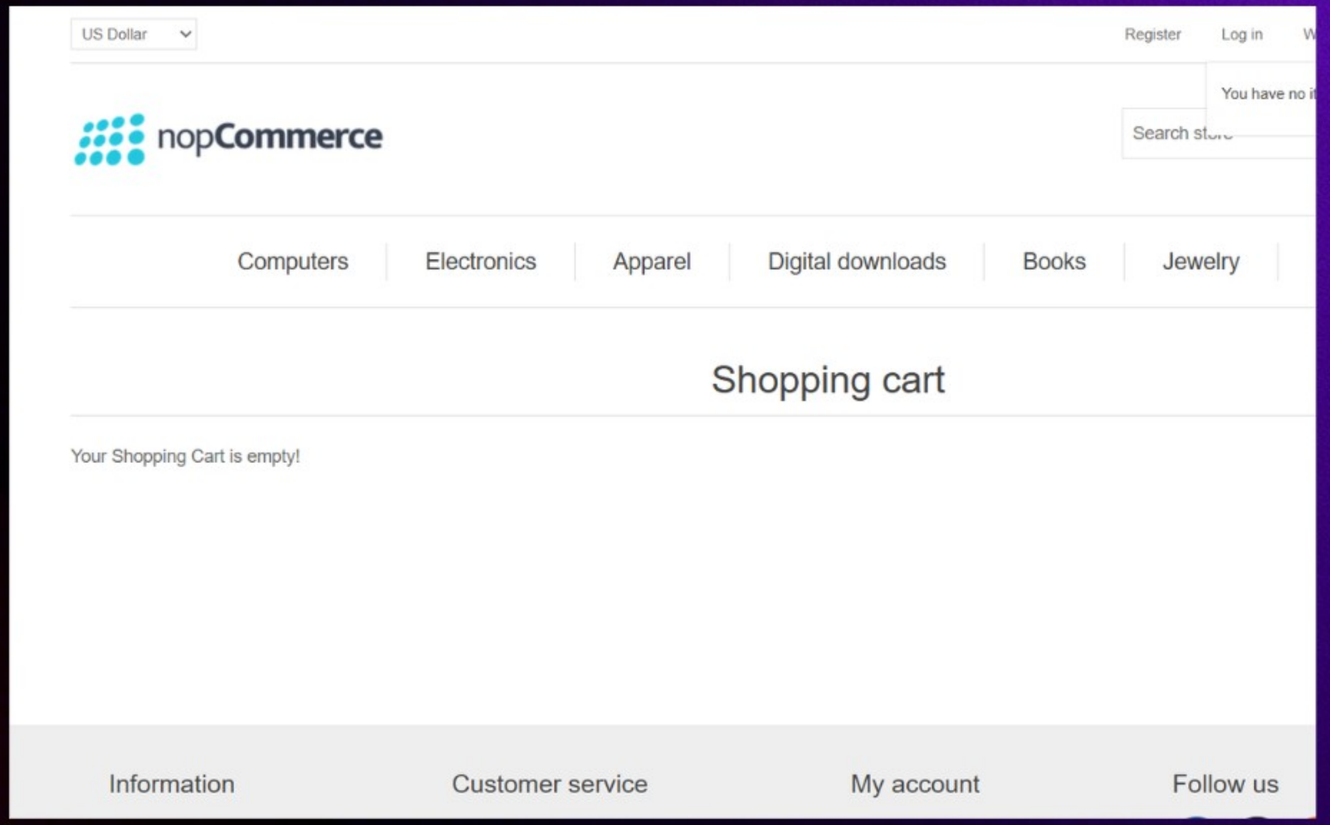


Registers a user with valid/invalid inputs and verifies successful login, then searches for a product and validates its search results and product details.

Add to Cart

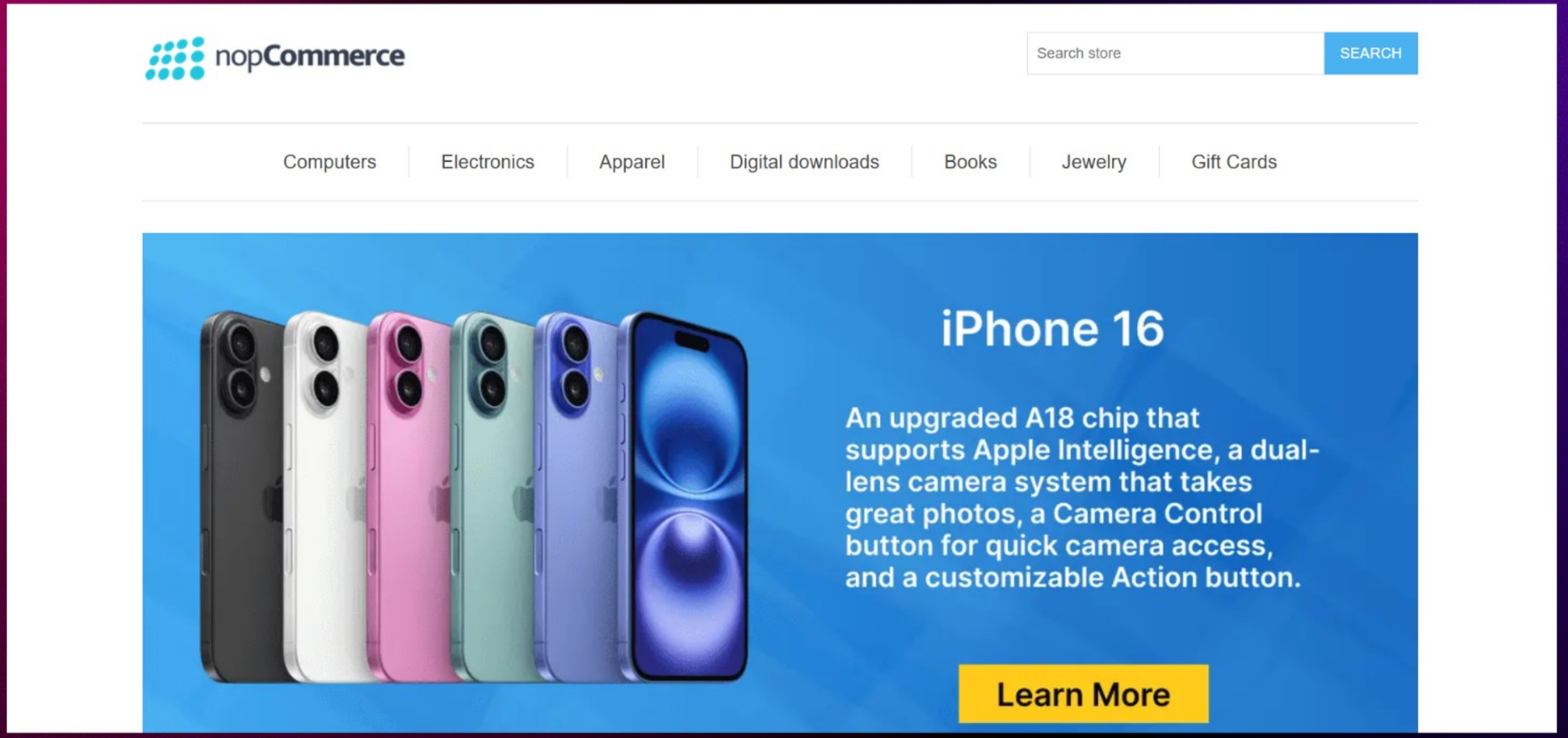


Remove Items



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



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

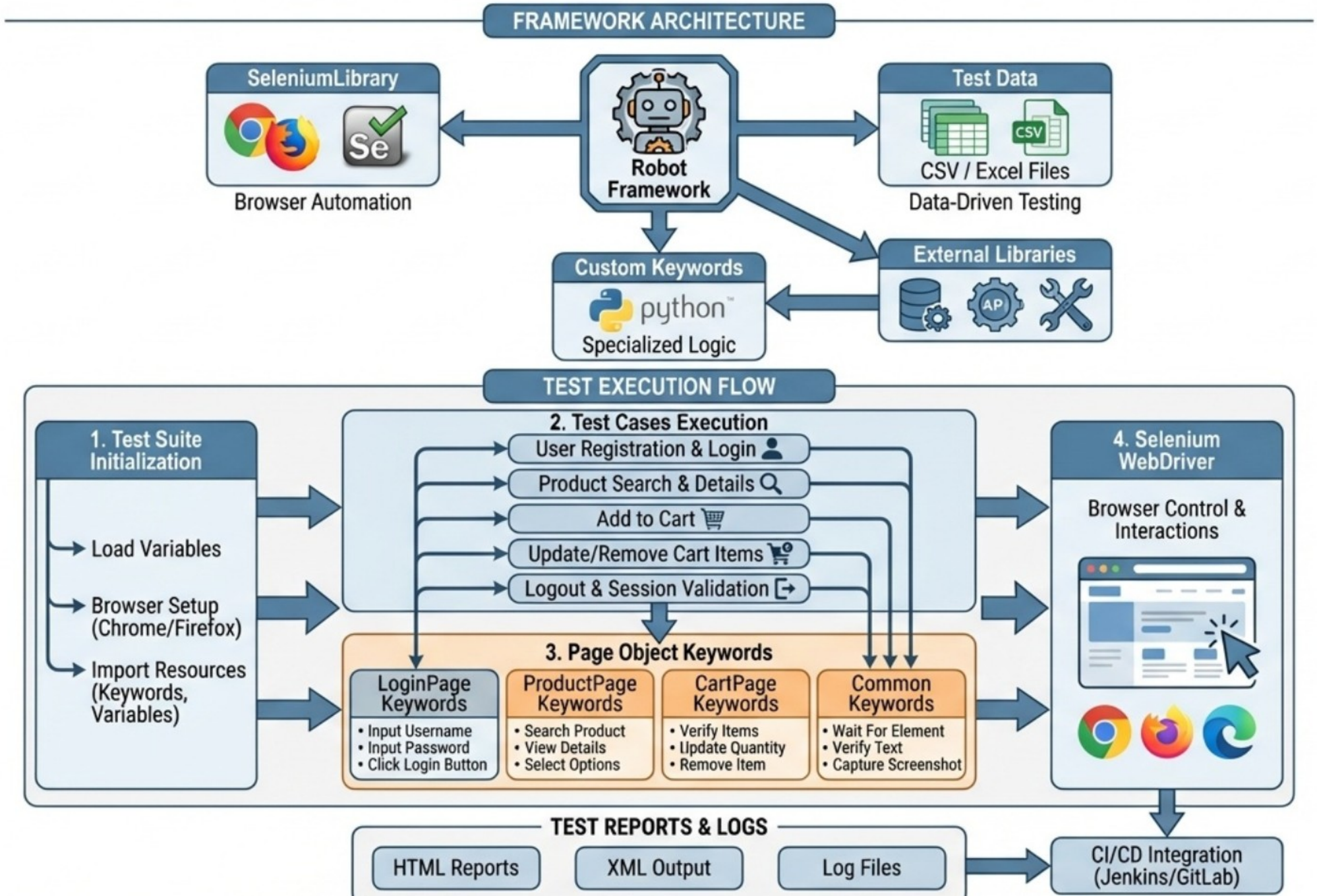
Objective :

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 | shop.robot | run_all_robot.py | tests\report.html | test_app.py | reports\report.html | test_data_user

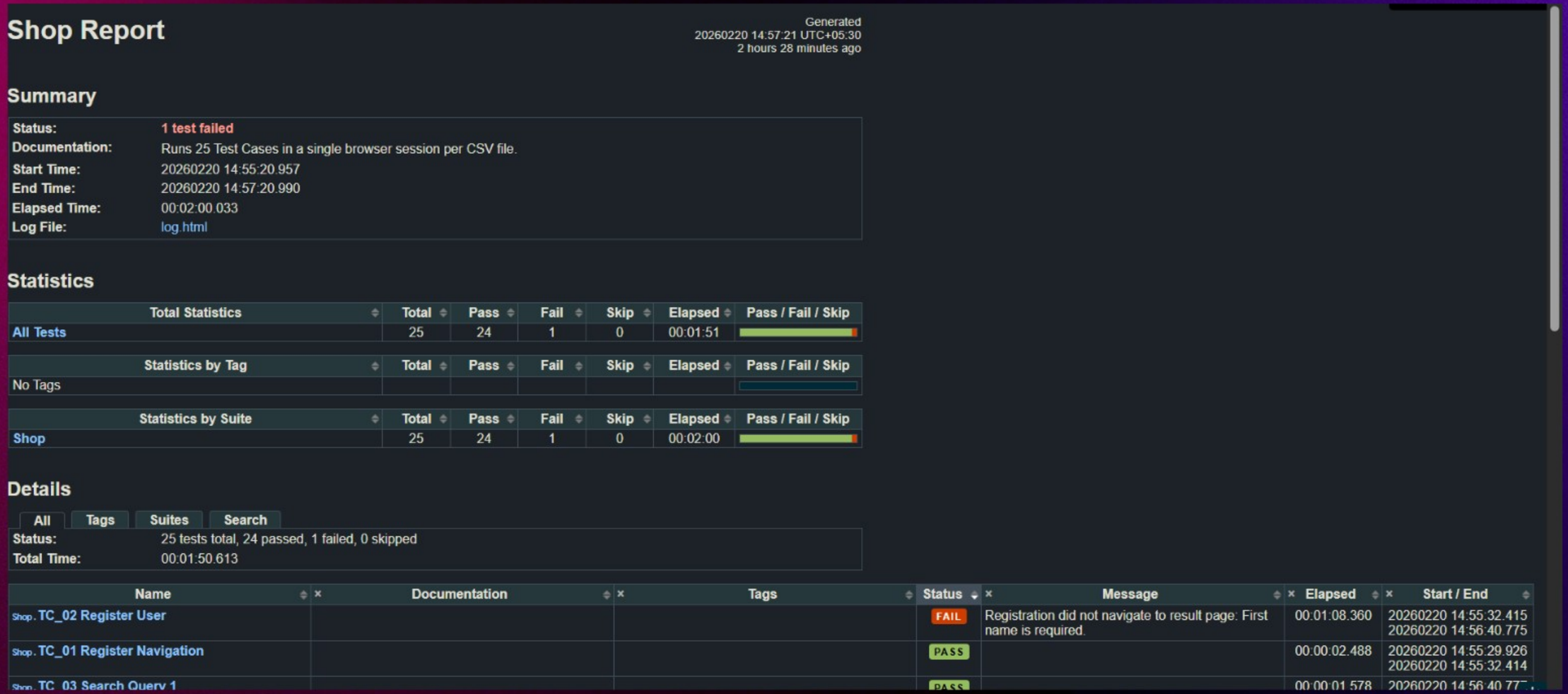
```
1   *** Settings ***
2   Documentation     Runs 25 Test Cases in a single browser session per CSV file.
3   Resource          ../resources/common.resource
4   Library           BuiltIn
5
6   Suite Setup       Setup Everything    ${CSV_PATH}
7   Suite Teardown    Finalize Execution
8   Test Teardown     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 | Command Prompt

```
'robot' 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:	log.html

Statistics

Total Statistics	Total	Pass	Fail	Skip	Elapsed	Pass / Fail / Skip
All Tests	25	24	1	0	00:01:51	

Statistics by Tag	Total	Pass	Fail	Skip	Elapsed	Pass / Fail / Skip
No Tags						

Statistics by Suite	Total	Pass	Fail	Skip	Elapsed	Pass / Fail / Skip
Shop	25	24	1	0	00:02:00	

Details

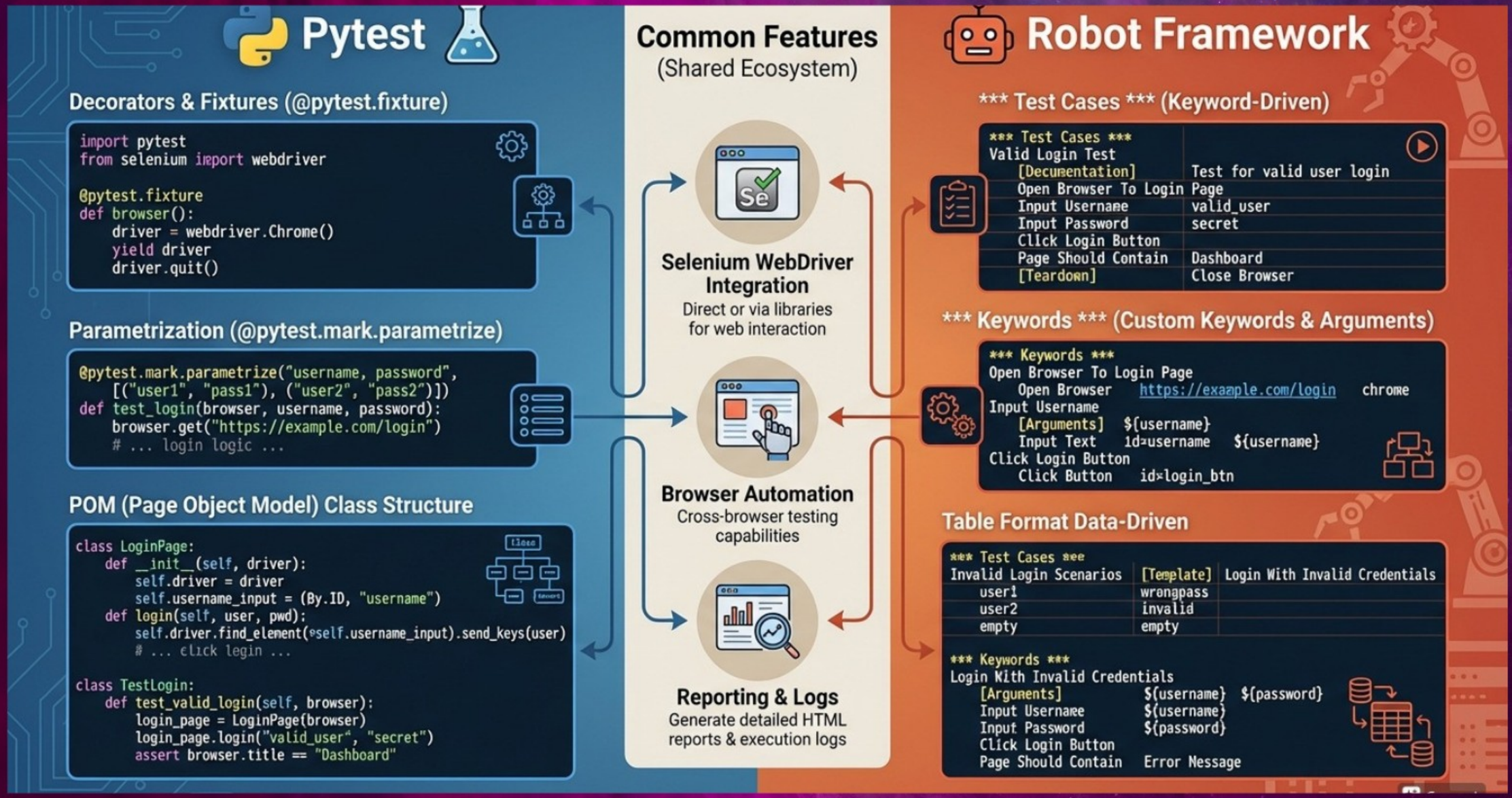
All | Tags | Suites | Search

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 20260220 14:56:40.775
Shop . TC_01 Register Navigation			PASS		00:00:02.488	20260220 14:55:29.926 20260220 14:55:32.414
Shop . TC_03 Search Query 1			PASS		00:00:01.578	20260220 14:56:40.77

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

BACKEND API TESTING PROJECT

v1.0.0 // 2026

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.

TESTING STRATEGY

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

COMPREHENSIVE QUALITY ASSURANCE

Pytest Execution Console

Web Browser Output

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")
```

```
rminal   Local ×   Command Prompt ×   Command Prompt ×
env456) C:\Users\Lenovo\PycharmProjects\Wipro-Training-2026\FoodieApp>python -m pytest test_app.py
============================================================ test session starts ============================
atform win32 -- Python 3.10.11, pytest-9.0.2, pluggy-1.6.0
otdir: C:\Users\Lenovo\PycharmProjects\Wipro-Training-2026\FoodieApp
ugins: xdist-3.8.0
llected 18 items

st_app.py ..................

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

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

Restaurant Tests Report

Generate
20260220 19:17:53 UTC+05:3
7 seconds ag

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:	log.html

Statistics

Total Statistics	Total	Pass	Fail	Skip	Elapsed	Pass / Fail / Skip
All Tests	18	18	0	0	00:00:06	

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	

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.

POM separates page representation from test logic

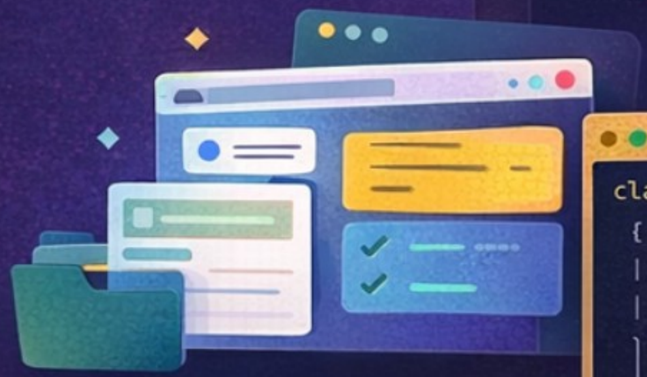
Test Scripts → Page Objects → Web Elements

Why Use POM in Foodie App?

- **Separation of Concerns**
  - Test logic stays in test files, UI interactions stay in page classes.
- **Reusability**
  - Same page methods reused across multiple tests.
- **Maintainability**
  - If locator changes, update in one place only.
- **Readability**
  - Tests look clean and easy to understand.

Foodie UI Automation Structure:

```
pages/
  login_page.py
  restaurant_page.py
  dish_page.py
  order_page.py
tests/
  test_login.py
  test_restaurant.py
  test_order.py
conftest.py (Driver Setup)
```



```
class loginPage:
  {
    def _init,,(& _griver)
      login.loginPage, "admin123")
    }

    conftest.py (Driver Setup)
```

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

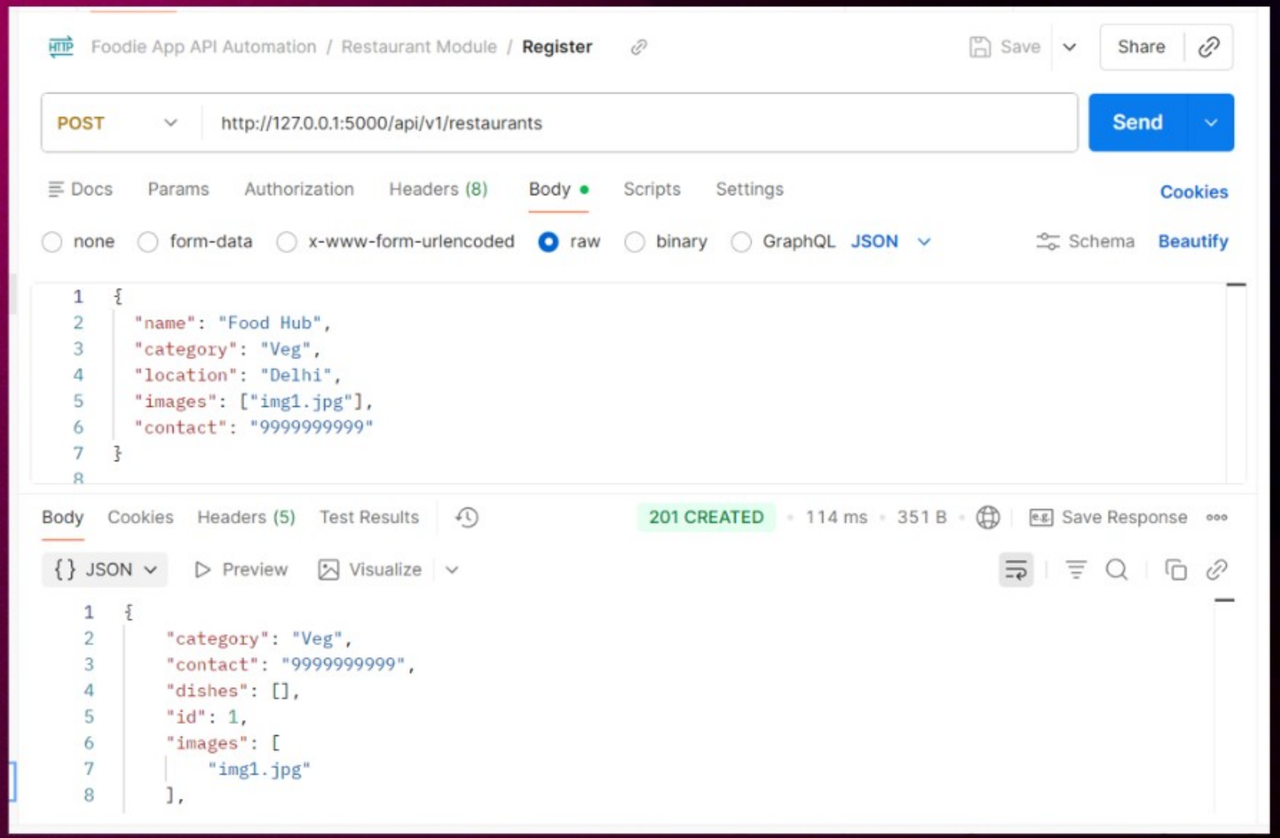
Dish Module

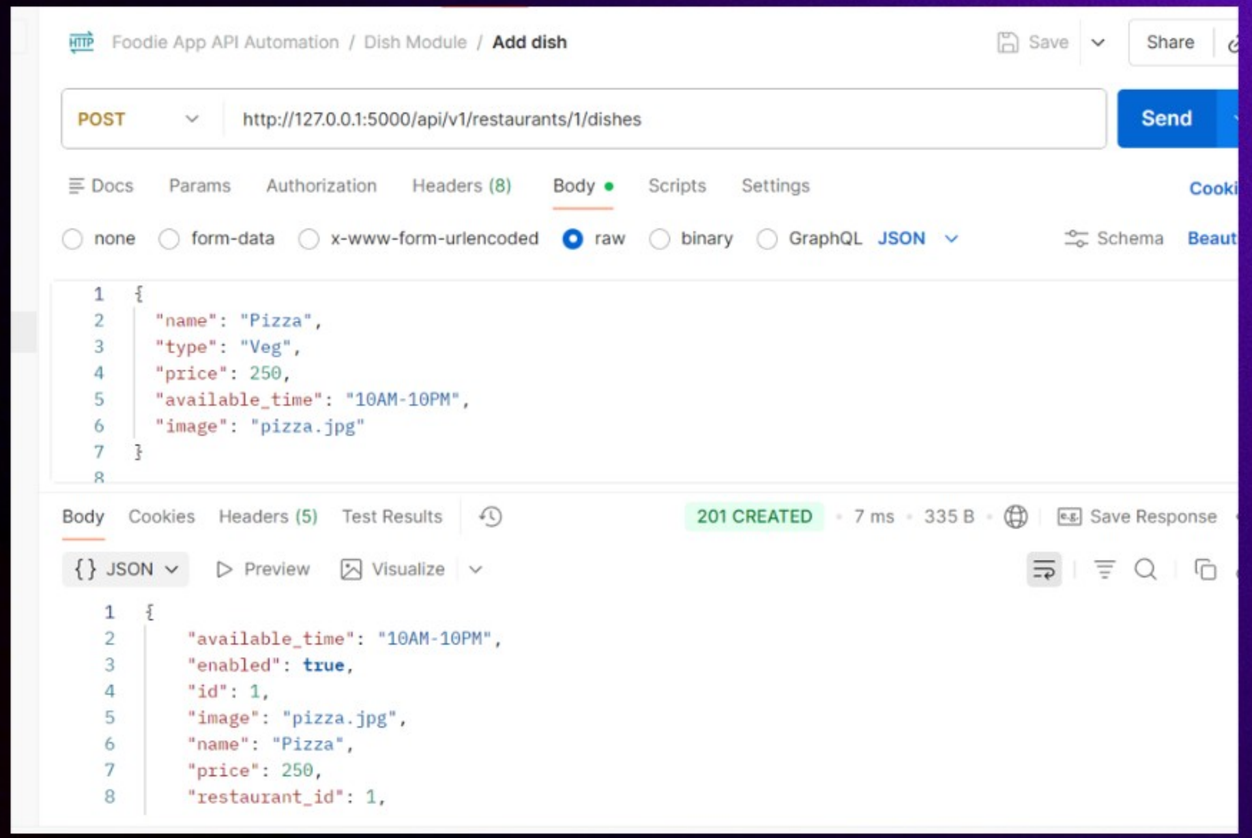
Dish Module :

- 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	POST	/api/v1/restaurants/{restaurant_id}/dishes	201 Created, 400 Bad Request
6	Update Dish	PUT	/api/v1/dishes/{dish_id}	200 OK, 404 Not Found
7	Enable / Disable Dish	PUT	/api/v1/dishes/{dish_id}/status	200 OK, 404 Not Found
8	Delete Dish	DELETE	/api/v1/dishes/{dish_id}	200 OK, 404 Not Found

Postman Collection - Restaurant & Dish Module





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	PUT	/api/v1/admin/restaurants/{restaurant_id}/approve	200 OK, 404 Not Found
Disable Restaurant	PUT	/api/v1/admin/restaurants/{restaurant_id}/disable	200 OK, 404 Not Found
View Customer Feedback	GET	/api/v1/admin/feedback	200 OK
View Order Status	GET	/api/v1/admin/orders	200 OK

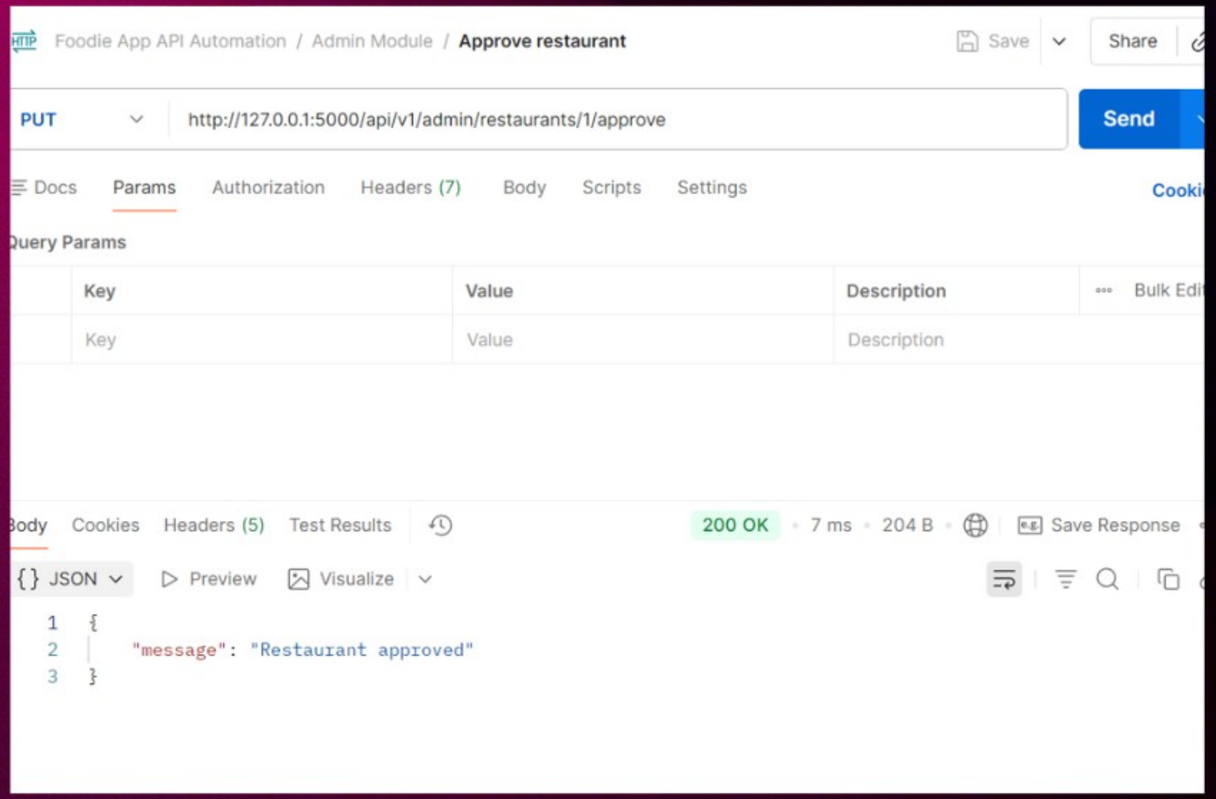
User Module

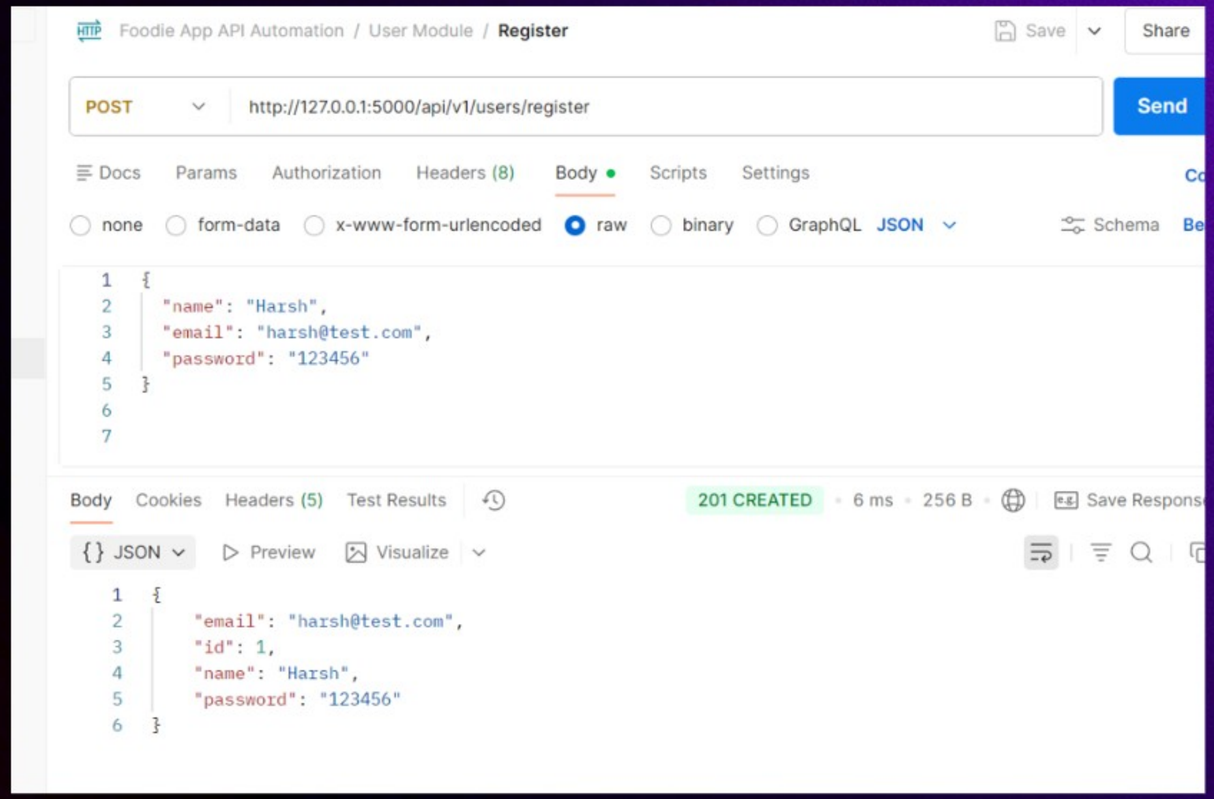
User Interaction :

- 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	POST	/api/v1/users/register	201 Created, 409 Conflict
Search Restaurants	GET	/api/v1/restaurants/search?name=&location=&dish=	200 OK
Place Order	POST	/api/v1/orders	201 Created, 400 Bad Request
Give Rating	POST	/api/v1/ratings	201 Created, 400 Bad Request

Postman Collection – Admin & User Module





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

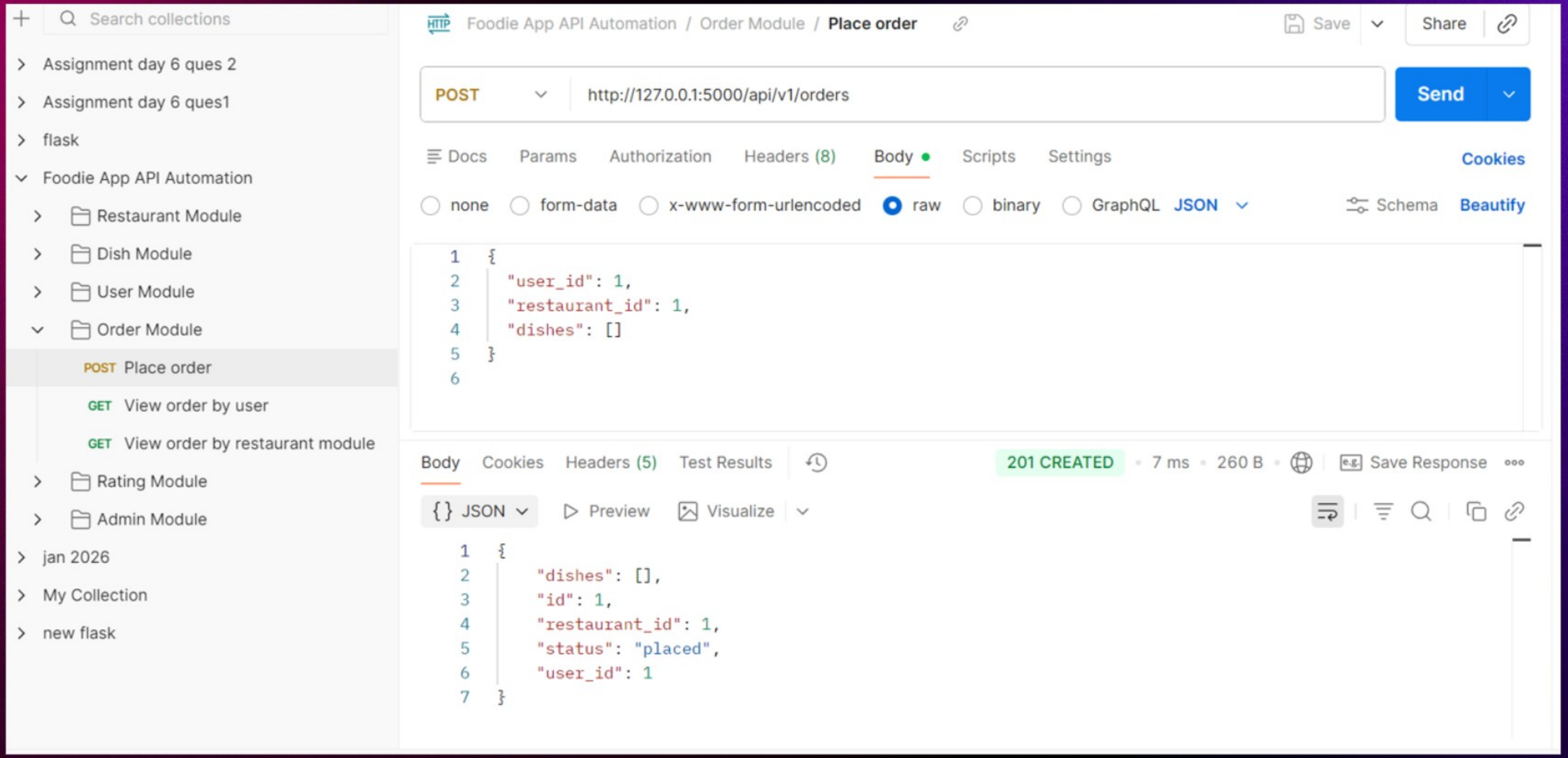
Order Module

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	GET	/api/v1/restaurants/{restaurant_id}/orders	200 OK
View Orders by User	GET	/api/v1/users/{user_id}/orders	200 OK

Postman Collection – Order Module



Summary

18	5	3	4
API Endpoints	Modules	Test Frameworks	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
- Reliabilty: Eflr: Prone to error cocctiow (Fatigue)
- Ideal For: Explorcatory, UI/UX Lons ton (Short-term)

Human-Centric Checks

AUTOMATION TESTING

POSTMAN

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

Efficient Cont 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!


pytest


Robot
Framework


Flask


Postman


Foodie App


TECHADEMY


wipro