

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

Dibyojyoti 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

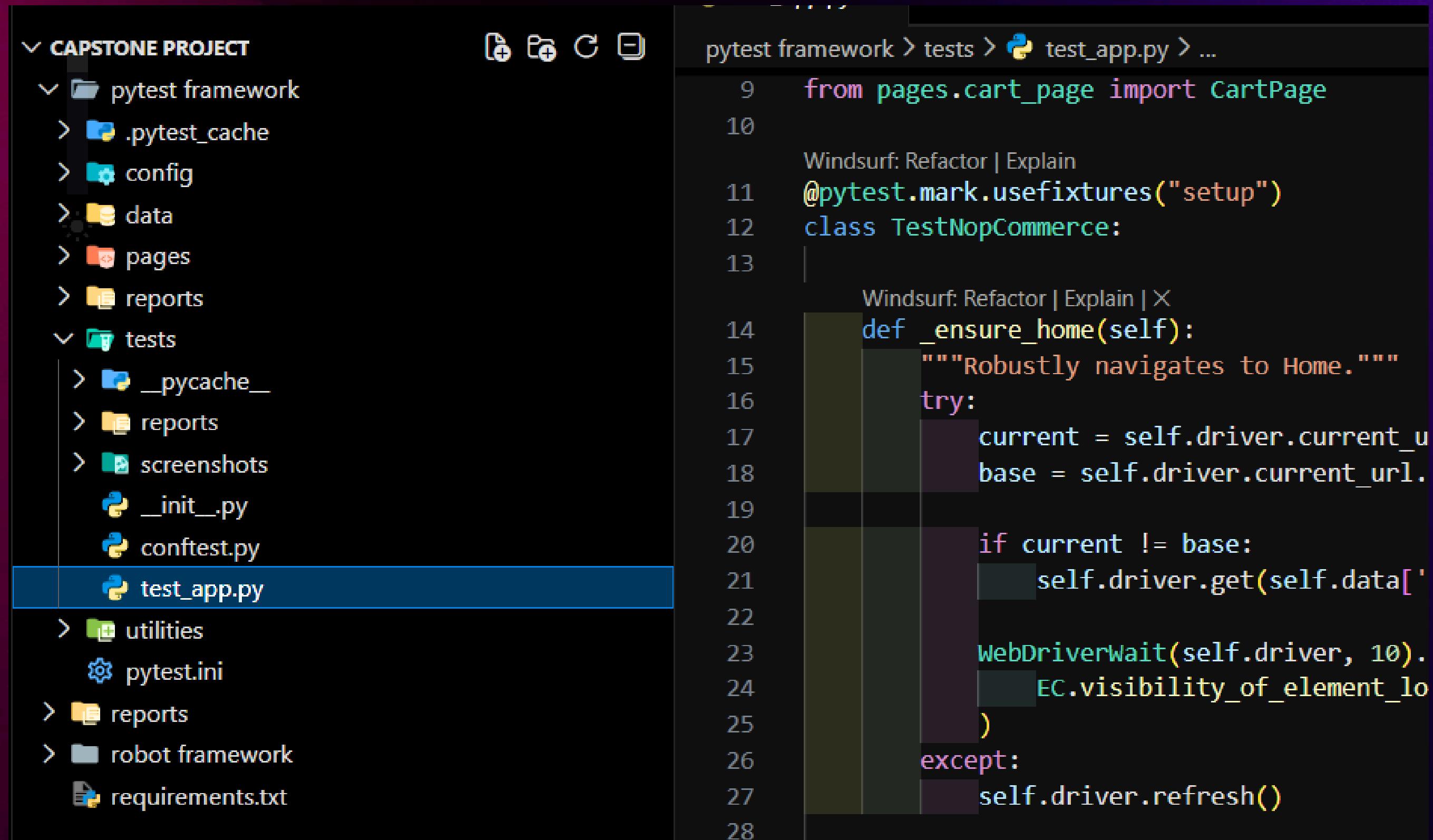
Application Under Test:

<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 code editor interface with two panes. The left pane displays the file structure of a 'CAPSTONE PROJECT'. The 'tests' directory contains several files: __init__.py, conftest.py, and test_app.py, which is currently selected and highlighted with a blue background. Other files in the project include .pytest_cache, config, data, pages, reports, and utilities. The right pane shows the content of test_app.py. The code defines a class TestNopCommerce with a method _ensure_home. This method uses a try-except block to navigate to the home page. It checks if the current URL matches the expected base URL. If it doesn't, it uses WebDriverWait to wait for the element to become visible before refreshing the driver.

```
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

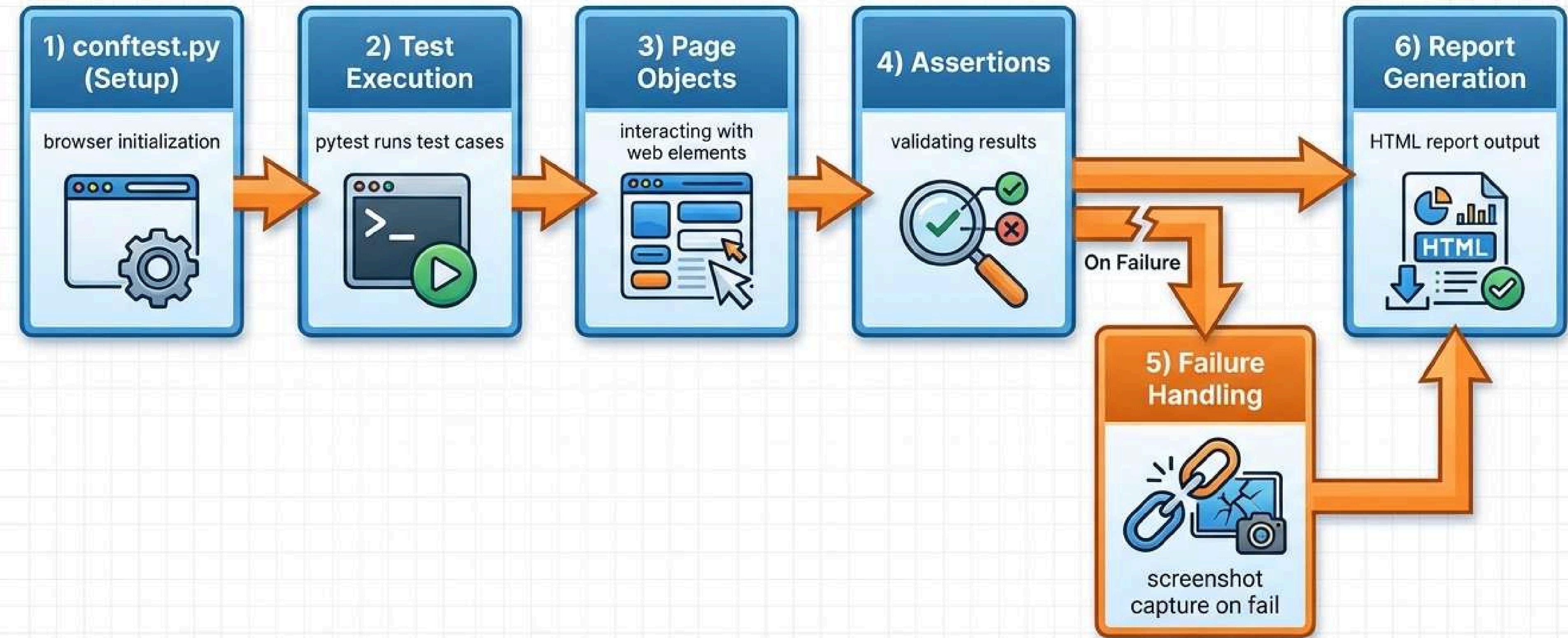
Objective :

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 dark-themed code editor interface with multiple tabs open. The active tab is `test_app.py`, which contains Python test code for a NopCommerce application. The code includes imports for `pytest`, `selenium.webdriver.common`, `selenium.webdriver.support.ui`, and various page objects. It defines a class `TestNopCommerce` with a method `_ensure_home` that robustly navigates to the home page. The code editor has a sidebar on the right showing a tree view of the project structure.

Below the code editor is a terminal window showing the output of a `pytest` run. The output starts with the message "test session starts" and provides details about the environment, including the Python version (3.12.6), platform (Windows-10-10.0.19045-SP0), and installed packages like pytest, pluggy, html, metadata, and xdist. It then lists 50 collected items, followed by a series of test results for methods named `test_01` through `test_05`. The results show some failures and one success:

```
==== 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] PASSED [ 10%]
```

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">• pytest: 8.1.1• pluggy: 1.6.0
Plugins	<ul style="list-style-type: none">• html: 4.1.1• metadata: 3.1.1

Summary

50 tests took 00:13:21.

(Un)check the boxes to filter the results.

<input checked="" type="checkbox"/> 0 Failed, <input checked="" type="checkbox"/> 50 Passed, <input type="checkbox"/> 0 Skipped, <input type="checkbox"/> 0 Expected failures, <input type="checkbox"/> 0 Unexpected passes, <input type="checkbox"/> 0 Errors, <input type="checkbox"/> 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



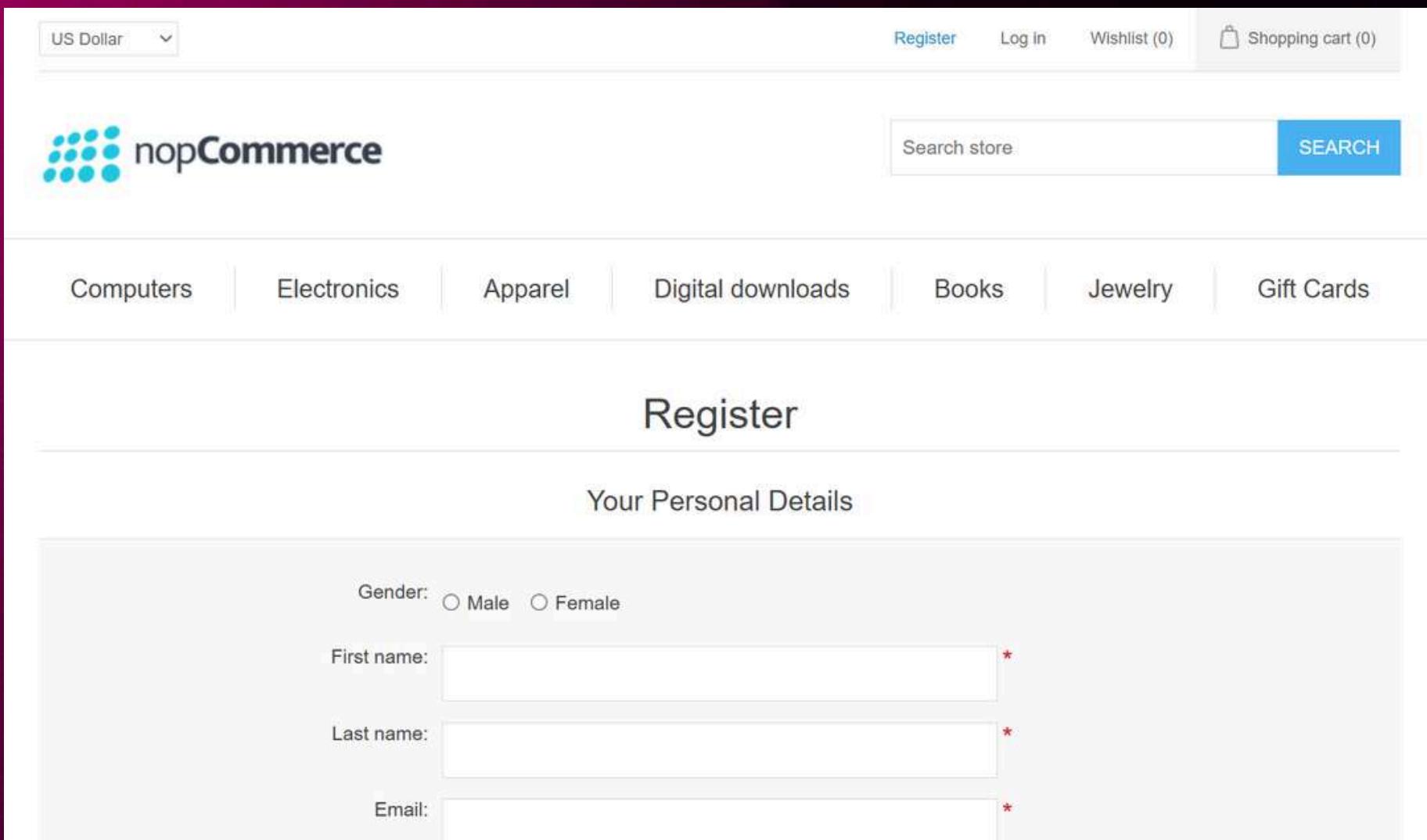
1 Separation of Concerns Test logic stays in test files; page interactions encapsulated in page classes

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

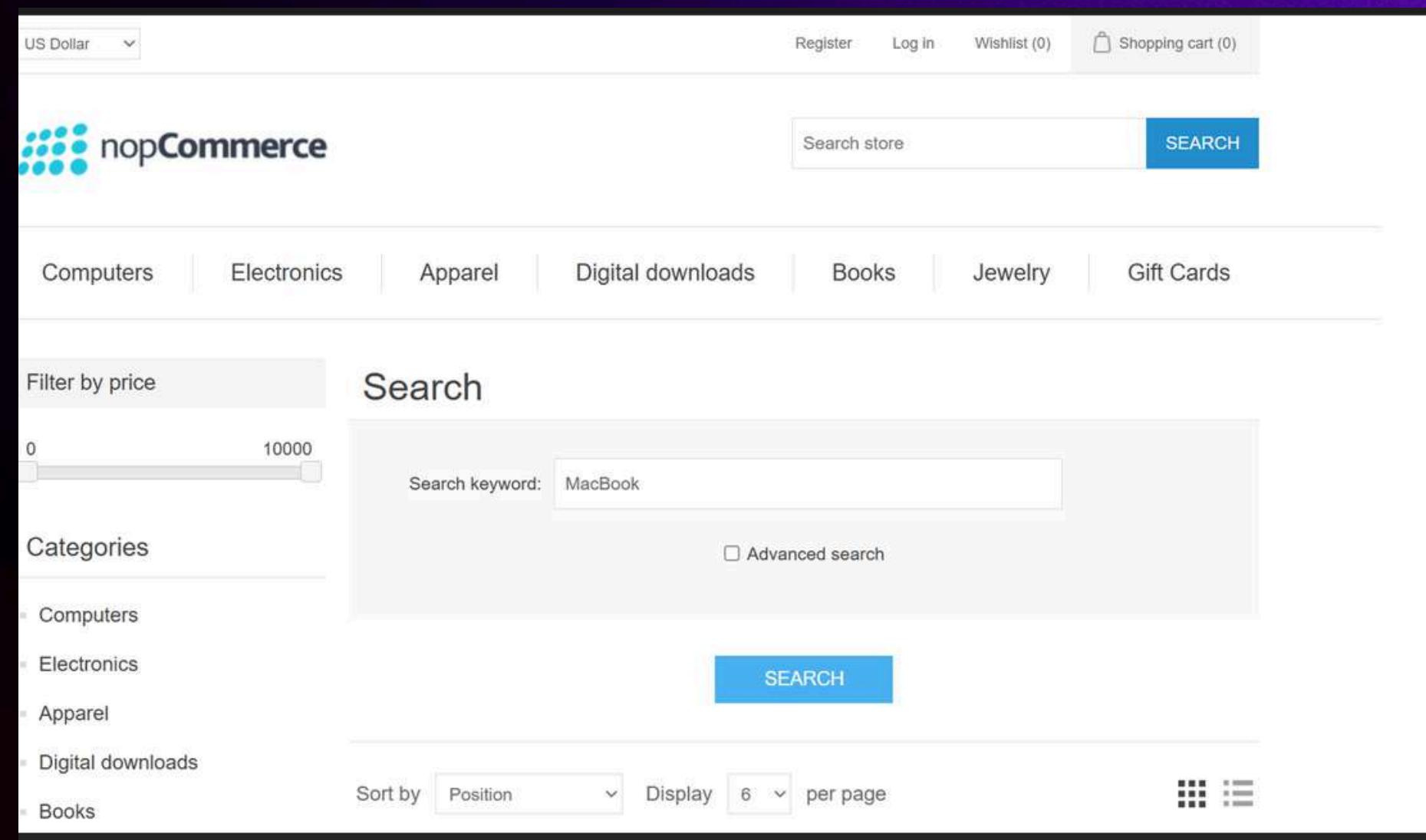
4 Readability Tests read like user stories: `login_page.enter_credentials()` is self-documenting

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 and a search bar labeled "Search store" with a blue "SEARCH" button. A horizontal menu bar includes categories like "Computers", "Electronics", "Apparel", "Digital downloads", "Books", "Jewelry", and "Gift Cards". The main content area is titled "Register" and contains a form for "Your Personal Details". It includes fields for "Gender" (radio buttons for "Male" and "Female"), "First name", "Last name", and "Email", each marked with a red asterisk indicating required fields. There are also "Forgot password?" and "Create account" links.

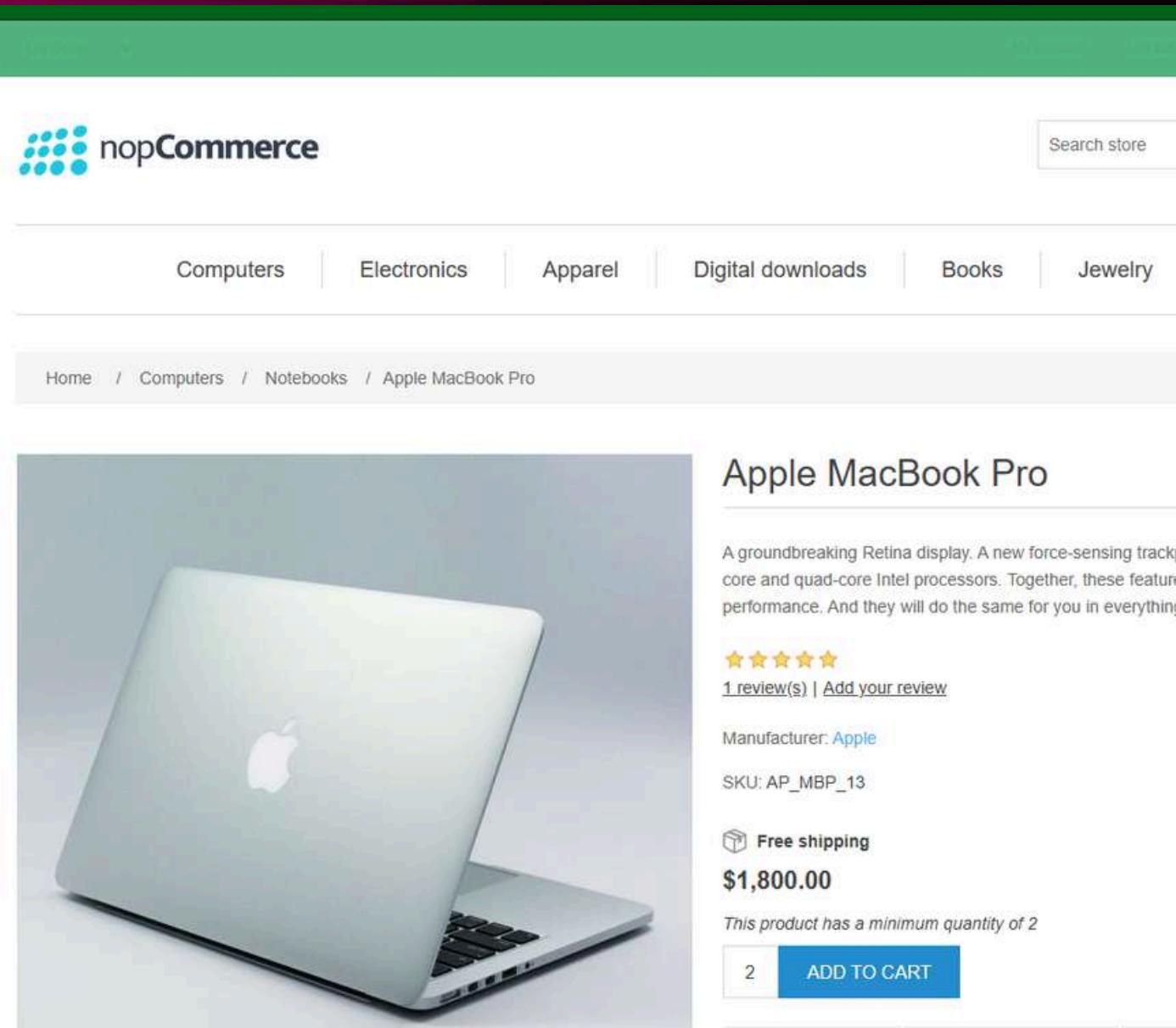
Search Product



The screenshot shows the product search page of the same nopCommerce website. The top navigation bar is identical to the registration page. The main content area is titled "Search" and features a "Search keyword" input field containing "MacBook", a "SEARCH" button, and a "Filter by price" slider set between 0 and 10000. To the left, there's a sidebar with "Categories" (Computers, Electronics, Apparel, Digital downloads, Books) and a "Sort by" dropdown set to "Position". At the bottom right, there are grid and list view icons. The footer of the page is visible at the very bottom.

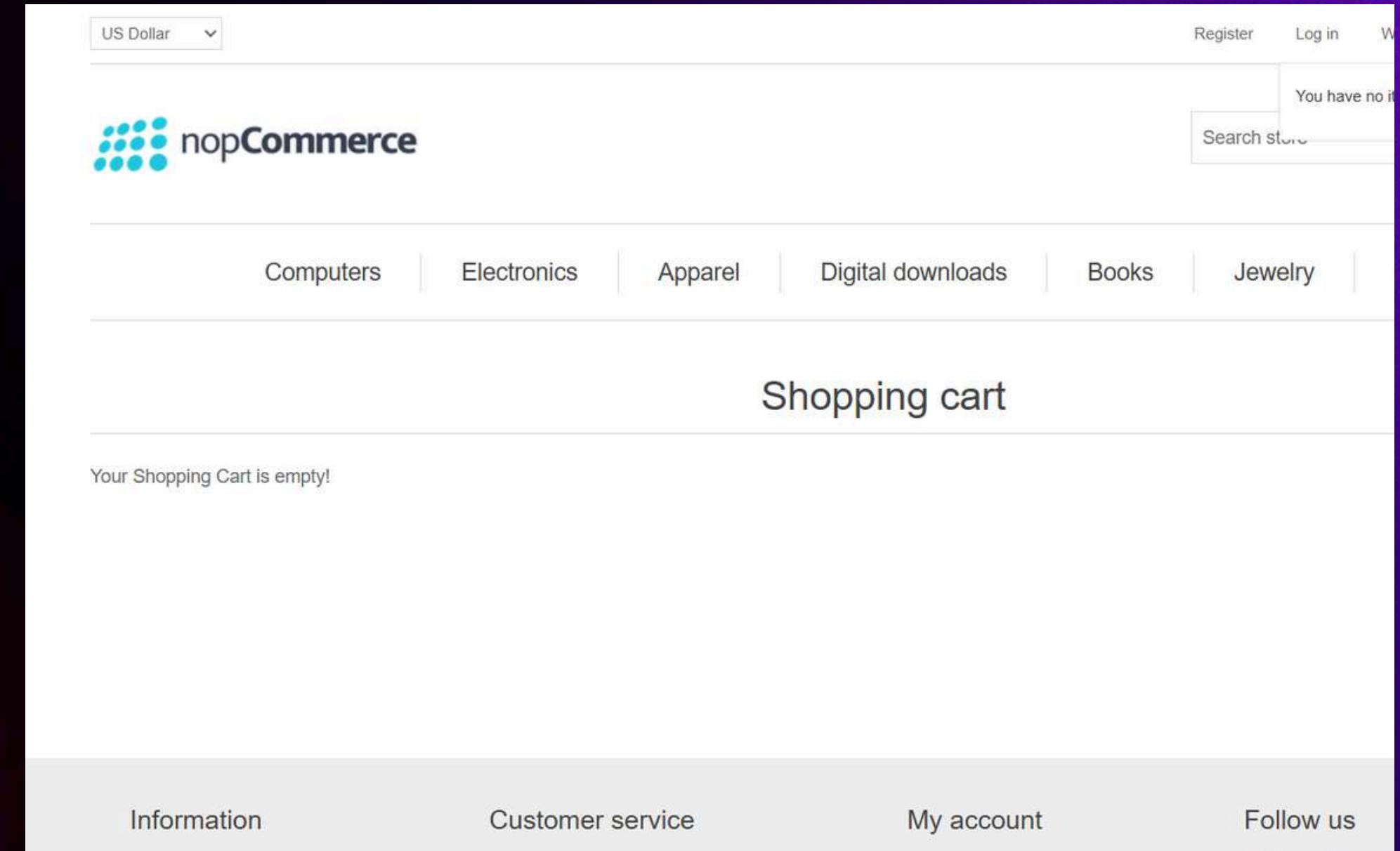
Registers a user with valid/invalid inputs and verifies successful login, then searches for a product and validates its search results and product details.

Add to Cart



A screenshot of a nopCommerce e-commerce website. 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 and a search bar labeled "Search store". A navigation menu includes categories like Computers, Electronics, Apparel, Digital downloads, Books, and Jewelry. The breadcrumb navigation shows the user is at Home > Computers > Notebooks > Apple MacBook Pro. The main content area features a large image of a silver Apple MacBook Pro laptop. To its right, the product title "Apple MacBook Pro" is displayed, followed by a detailed product description: "A groundbreaking Retina display. A new force-sensing trackpad. And quad-core Intel processors. Together, these features deliver a new level of performance. And they will do the same for you in everything you do." Below the description are ratings (5 stars), reviews ("1 review(s) | Add your review"), manufacturer information ("Manufacturer: Apple"), and SKU ("SKU: AP_MBP_13"). It also highlights "Free shipping" and lists the price as "\$1,800.00". A note states "This product has a minimum quantity of 2". At the bottom of the product card is a blue "ADD TO CART" button with the number "2" next to it.

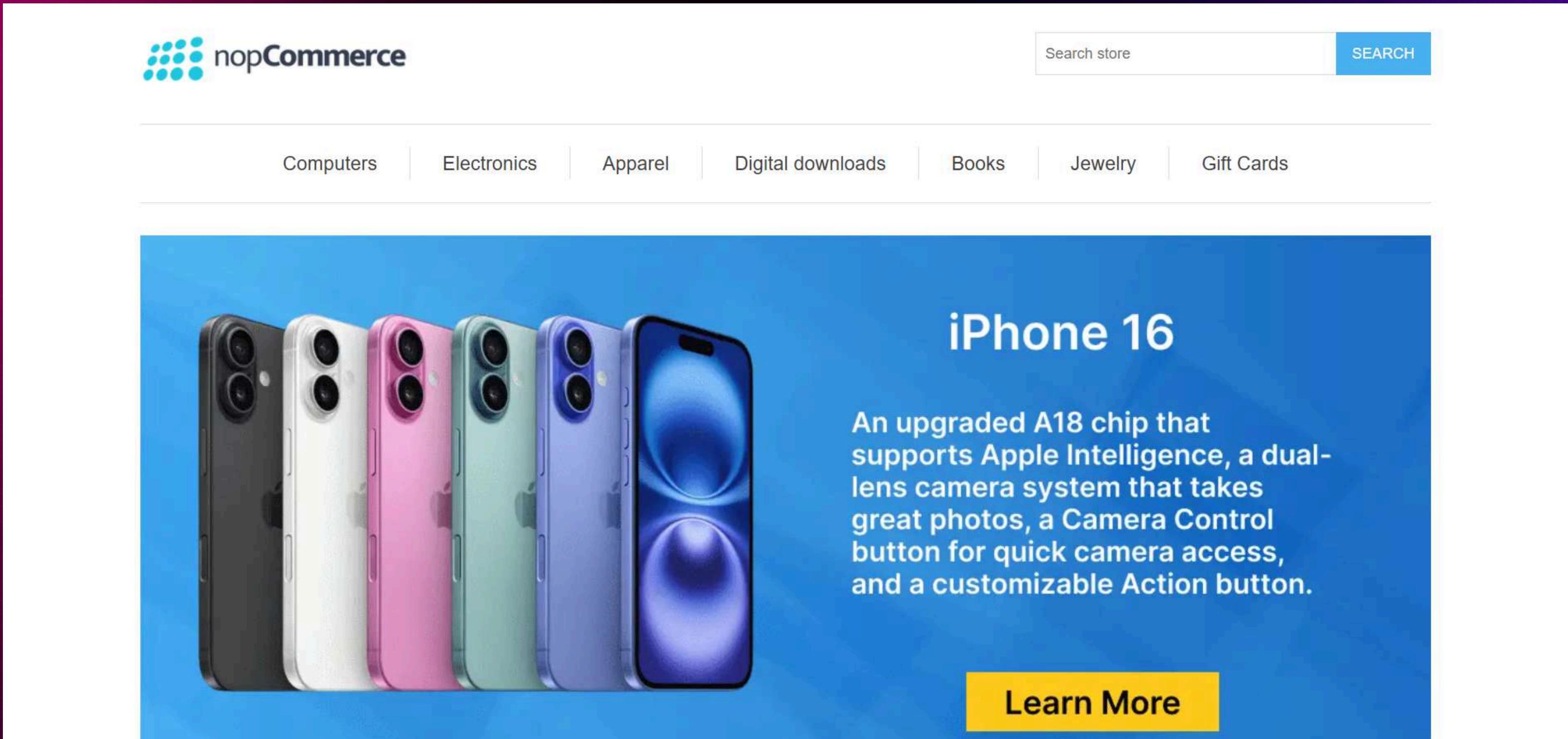
Remove Items



A screenshot of the same nopCommerce website showing the shopping cart. The top navigation bar includes a dropdown for "US Dollar", "Register", "Log in", and a search bar. The main content area is titled "Shopping cart" and displays the message "Your Shopping Cart is empty!". At the bottom of the page, there are links for "Information", "Customer service", "My account", and "Follow us".

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



The image shows the homepage of a nopCommerce e-commerce website. At the top left is the logo 'nopCommerce'. To its right is a search bar with the placeholder 'Search store' and a blue 'SEARCH' button. Below the header is a navigation menu with categories: Computers, Electronics, Apparel, Digital downloads, Books, Jewelry, and Gift Cards. A prominent banner in the center features six iPhone 16 phones in various colors (black, white, pink, light green, medium green, and blue) displayed vertically. To the right of the phones, the text 'iPhone 16' is displayed in large white letters. Below this, a detailed description of the phone's features is given: '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.' At the bottom right of the banner is a yellow 'Learn More' button.

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:

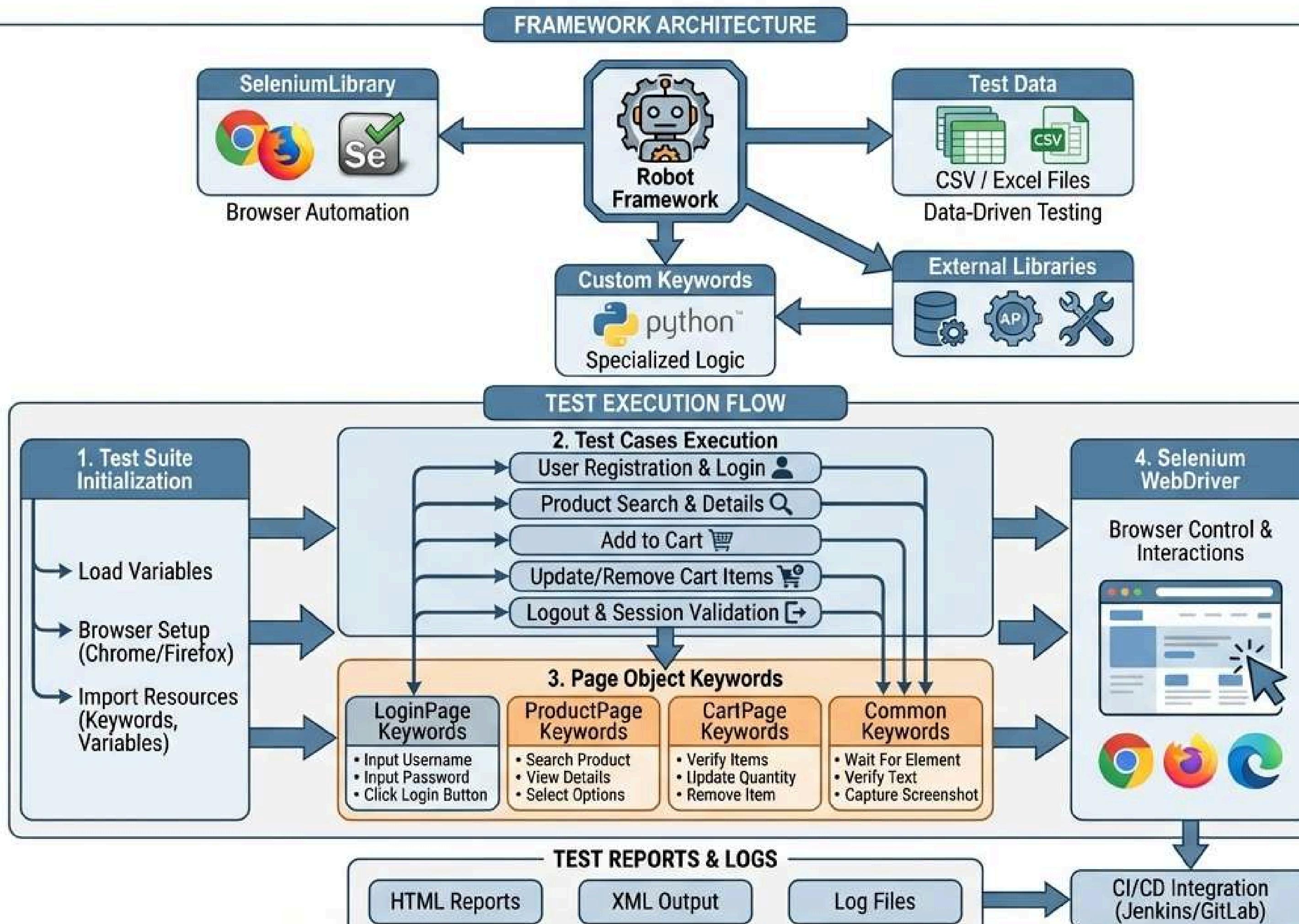
Performed end-to-end UI automation testing on an e-commerce web application using Selenium with Robot Framework, implementing a keyword-driven and reusable test architecture.

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 ✘  Python run_all_robot.py    tests\report.html    Python 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 ×
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:	log.html

Statistics

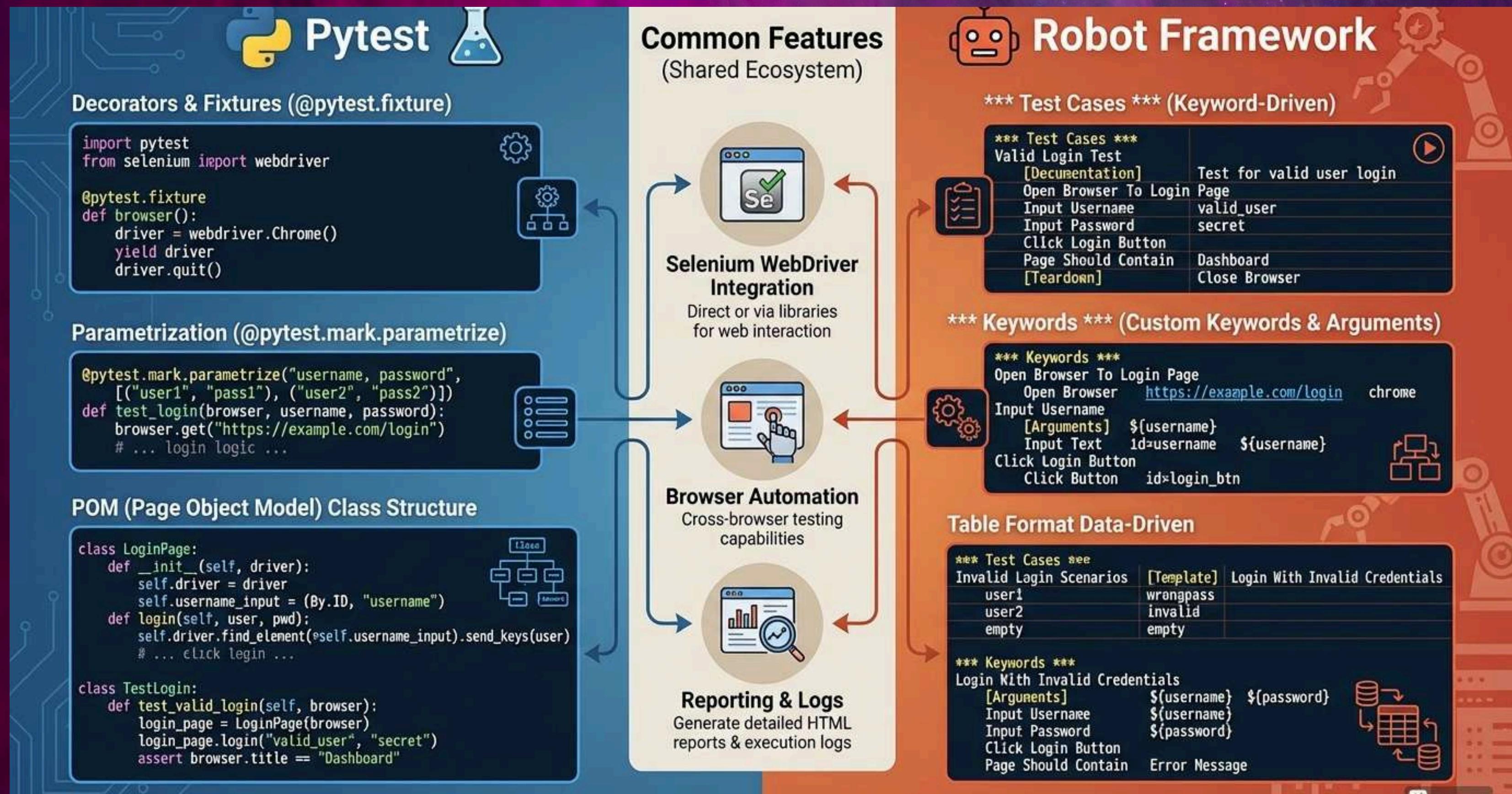
Total Statistics	Total	Pass	Fail	Skip	Elapsed	Pass / Fail / Skip
All Tests	25	24	1	0	00:01:51	<div style="width: 96%; background-color: #2e7131; height: 10px;"></div>
Statistics by Tag	Total	Pass	Fail	Skip	Elapsed	Pass / Fail / Skip
No Tags						<div style="width: 100%; background-color: #2e7131; height: 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: #2e7131; height: 10px;"></div>

Details

All	Tags	Suites	Search
Status:	25 tests total, 24 passed, 1 failed, 0 skipped		
Total Time:	00:01:50.613		
<hr/>			
Name	Documentation	Tags	Status
Shop.TC_02 Register User			FAIL
Shop.TC_01 Register Navigation			PASS
Shop.TC_03 Search Querry 1			PASS

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



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 7.42s.

```
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):
    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 report includes a summary table with details like start time, end time, elapsed time, and a log file link. Below the summary is a statistics table showing total test counts and pass/fail/skip details for "All Tests" and "Restaurant Tests". The bottom section provides search and filter options for specific test details.

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:	log.html

Statistics

	Total Statistics	Total	Pass	Fail	Skip	Elapsed	Pass / Fail / Skip
All Tests		18	18	0	0	00:00:06	<div style="width: 100%; background-color: #2e7131; height: 10px;"></div>
	Statistics by Tag	Total	Pass	Fail	Skip	Elapsed	Pass / Fail / Skip
No Tags							<div style="width: 100%; background-color: #2e7131; height: 10px;"></div>
	Statistics by Suite	Total	Pass	Fail	Skip	Elapsed	Pass / Fail / Skip
Restaurant Tests		18	18	0	0	00:00:09	<div style="width: 100%; background-color: #2e7131; height: 10px;"></div>

Details

All Tags Suites Search

Suite:

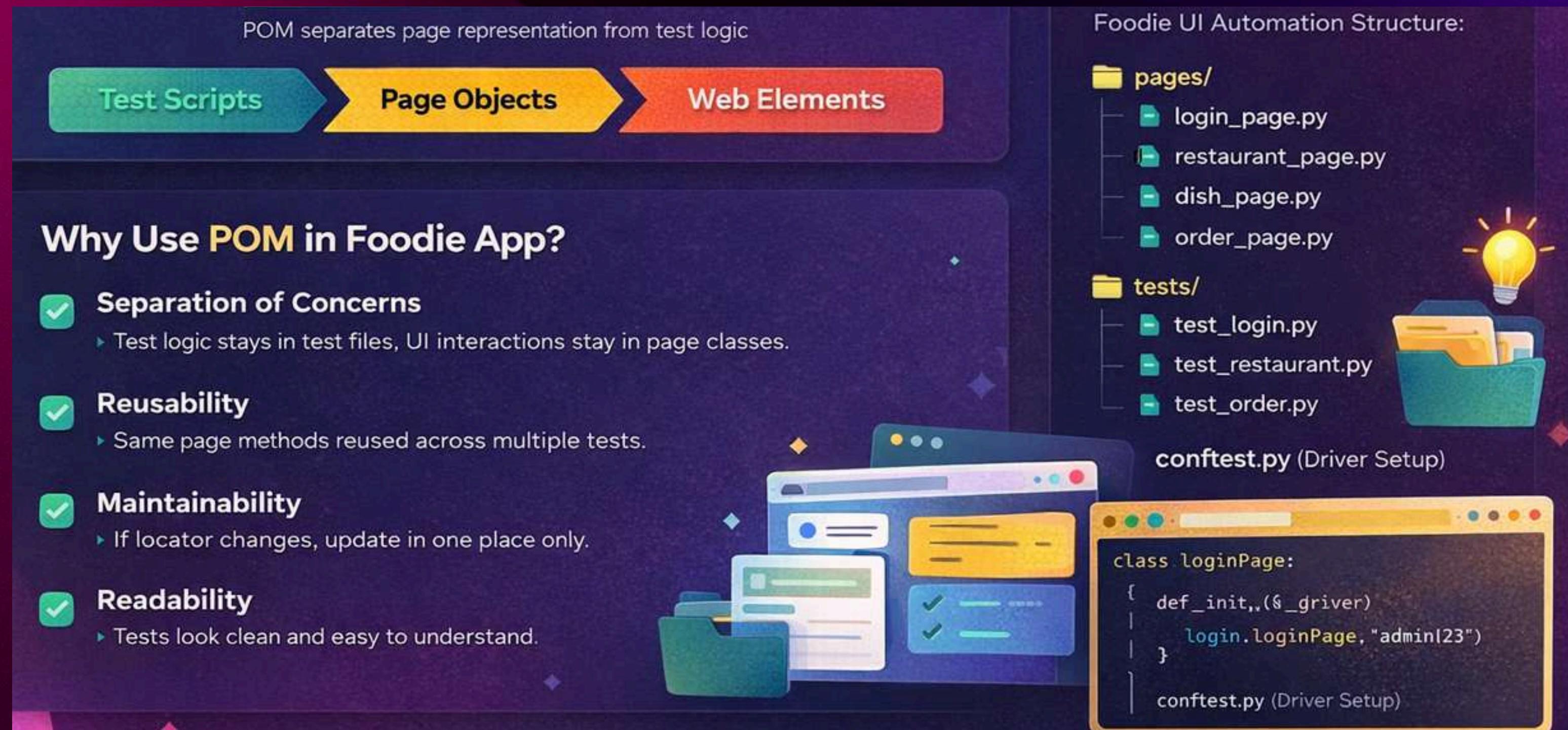
Test:

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	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

The screenshot shows the Postman interface for the 'Foodie App API Automation / Restaurant Module / Register' collection. It displays a POST request to `http://127.0.0.1:5000/api/v1/restaurants`. The request body is defined in JSON format:

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

The response status is 201 CREATED, with a response time of 114 ms and a response size of 351 B. The response body is also in JSON format:

```
1 {  
2   "category": "Veg",  
3   "contact": "9999999999",  
4   "dishes": [],  
5   "id": 1,  
6   "images": [  
7     "img1.jpg"  
8   ],  
9 }
```

The screenshot shows the Postman interface for the 'Foodie App API Automation / Dish Module / Add dish' collection. It displays a POST request to `http://127.0.0.1:5000/api/v1/restaurants/1/dishes`. The request body is defined in JSON format:

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

The response status is 201 CREATED, with a response time of 7 ms and a response size of 335 B. The response body is also in JSON format:

```
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,  
9 }
```

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

The image shows two side-by-side Postman API requests.

Left Request: Admin Approval

- Method: PUT
- URL: <http://127.0.0.1:5000/api/v1/admin/restaurants/1/approve>
- Headers (7): Not explicitly shown in the screenshot, but implied by the context of the collection.
- Body (raw JSON):

Key	Value	Description
name	Restaurant approved	

- Response: 200 OK, 7 ms, 204 B. Body: {"message": "Restaurant approved"}

Right Request: User Registration

- Method: POST
- URL: <http://127.0.0.1:5000/api/v1/users/register>
- Headers (8): Authorization, Content-Type, Content-Length, Host, Connection, User-Agent, Accept, Accept-Encoding, Accept-Language, DNT, Origin, Referer, Upgrade-Insecure-Requests, Te, Pragma, Cache-Control, Content-Type, Content-Length, Host, Connection, User-Agent, Accept, Accept-Encoding, Accept-Language, DNT, Origin, Referer, Upgrade-Insecure-Requests, Te, Pragma, Cache-Control.
- Body (raw JSON):

```
1 {  
2   "name": "Harsh",  
3   "email": "harsh@test.com",  
4   "password": "123456"  
5 }  
6  
7
```

- Response: 201 CREATED, 6 ms, 256 B. Body: {"id": 1, "name": "Harsh", "email": "harsh@test.com", "password": "123456"}

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

The screenshot shows the Postman application interface. On the left, the sidebar displays a collection structure for "Foodie App API Automation". The "Order Module" section contains a "Place order" POST request. 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: Human Prone to error cognition (Fatigue)
- Ideal For: Exploratory, UI/UX Linting (Short-term)

Human-Centric Checks



AUTOMATION TESTING

POSTMAN

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

Efficient Test Integration

Conclusion

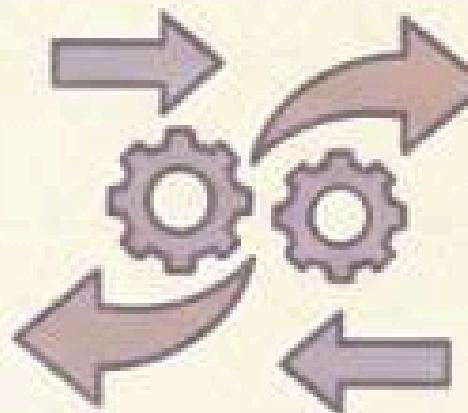
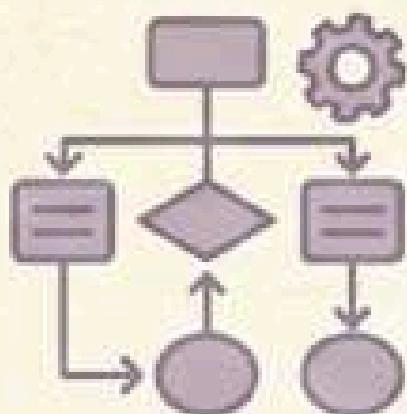
- Successfully architected and implemented modular, data-driven UI automation frameworks using Pytest and Robot Framework with Selenium.
- Engineered robust end-to-end UI test suites covering critical e-commerce workflows, including registration, product search, and cart management.
- Developed a functional 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!



TECHADEMY

