# Automation Testing Framework for E-commerce Website

## Objective

This document outlines the steps to create an automation framework/test suite to automate the search and add-to-cart functionalities for a sample e-commerce website. The tests will be executed using the latest Chrome browser.

## Step 1: Test Scenarios

### 1. Search for a Non-Existing Product

* Input: "abcd123"
* Expected Outcome: "No results found" message should be displayed.

### 2. Search for an Existing Product

* Input: "Monitor"
* Expected Outcome: Product results should display "Monitor" on the page.

## Step 2: Add to Cart Functionality

### 3. Add a Product to the Cart

* Select the 4th product from the results.
* Expected Outcome: The product should be added to the cart with correct quantity and price details.

### 4. Update Product Quantity

* Change quantity to 4.
* Expected Outcome: The cart should reflect the updated quantity and price.

## Step 3: Remove Product from Cart

### 5. Remove Product

* Expected Outcome: The cart should be empty.

## Technology Stack

* **Programming Language**: Python
* **Dependency Management**: Pip using requirements.txt
* **Test Library**: Pytest
* **UI Automation Tool**: Selenium

## Framework Requirements

1. **Environment Setup**:
   * Use the latest version of Chrome.
   * Web URL: https://www.amazon.in/
2. **Framework Design**:
   * Modular and reusable architecture.
   * Implement Page Object Model (POM).
   * Reporting mechanism for capturing test results.
3. **Code Quality**:
   * Follow best practices for readability and maintainability.
   * Include comments.
   * Implement exception handling.
4. **Execution**:
   * Automated execution of all tests.
   * Generate a report with pass/fail status.
5. **Submission**:
   * Push code to a public GitHub repository.
   * Provide a README file with setup and execution instructions.

## Sample Code Structure

bash

Copy code

/ecommerce\_tests

│

├── /tests

│ ├── test\_search.py

│ ├── test\_cart.py

│

├── /pages

│ ├── base\_page.py

│ ├── search\_page.py

│ ├── cart\_page.py

│

├── /utils

│ ├── report.py

│

├── requirements.txt

└── README.md

## Sample Code

### requirements.txt

plaintext

Copy code

selenium

pytest

### base\_page.py

python

Copy code

from selenium import webdriver

class BasePage:

def \_\_init\_\_(self, driver):

self.driver = driver

### search\_page.py

python

Copy code

from selenium.webdriver.common.by import By

from base\_page import BasePage

class SearchPage(BasePage):

SEARCH\_INPUT = (By.ID, "twotabsearchtextbox")

SEARCH\_BUTTON = (By.ID, "nav-search-submit-button")

NO\_RESULTS\_MESSAGE = (By.XPATH, "//span[contains(text(), 'no results')]")

def search\_product(self, product\_name):

self.driver.find\_element(\*self.SEARCH\_INPUT).send\_keys(product\_name)

self.driver.find\_element(\*self.SEARCH\_BUTTON).click()

def get\_no\_results\_message(self):

return self.driver.find\_element(\*self.NO\_RESULTS\_MESSAGE).text

### cart\_page.py

python

Copy code

from selenium.webdriver.common.by import By

from base\_page import BasePage

class CartPage(BasePage):

ADD\_TO\_CART\_BUTTON = (By.ID, "add-to-cart-button")

CART\_QUANTITY = (By.XPATH, "//input[@name='quantity']")

def add\_product\_to\_cart(self):

self.driver.find\_element(\*self.ADD\_TO\_CART\_BUTTON).click()

def update\_quantity(self, quantity):

quantity\_input = self.driver.find\_element(\*self.CART\_QUANTITY)

quantity\_input.clear()

quantity\_input.send\_keys(quantity)

### test\_search.py

python

Copy code

import pytest

from selenium import webdriver

from pages.search\_page import SearchPage

from pages.cart\_page import CartPage

class TestEcommerce:

@pytest.fixture(scope="class")

def setup\_class(self):

self.driver = webdriver.Chrome()

self.driver.get("https://www.amazon.in/")

yield

self.driver.quit()

def test\_search\_non\_existing\_product(self, setup\_class):

search\_page = SearchPage(self.driver)

search\_page.search\_product("ld345tsxslfer")

assert "no results" in search\_page.get\_no\_results\_message().lower()

def test\_search\_existing\_product(self, setup\_class):

search\_page = SearchPage(self.driver)

search\_page.search\_product("Laptop")

# Additional assertions can be added here

### README.md

markdown

Copy code

# E-commerce Automation Tests

## Setup Instructions

1. Clone the repository.

2. Install the dependencies:

```bash

pip install -r requirements.txt

1. Run the tests:

bash

Copy code

pytest tests

## Test Cases

* Search for a non-existing product.
* Search for an existing product.
* Add product to cart.
* Update product quantity in cart.
* Remove product from cart.

## CI/CD Integration

For bonus points, consider integrating with Jenkins or GitHub Actions for automated execution on code push.

Conclusion

This framework provides a solid foundation for automating the search and add-to-cart functionalities of an e-commerce website. By following the outlined structure and code samples, you can create a scalable and maintainable test suite.