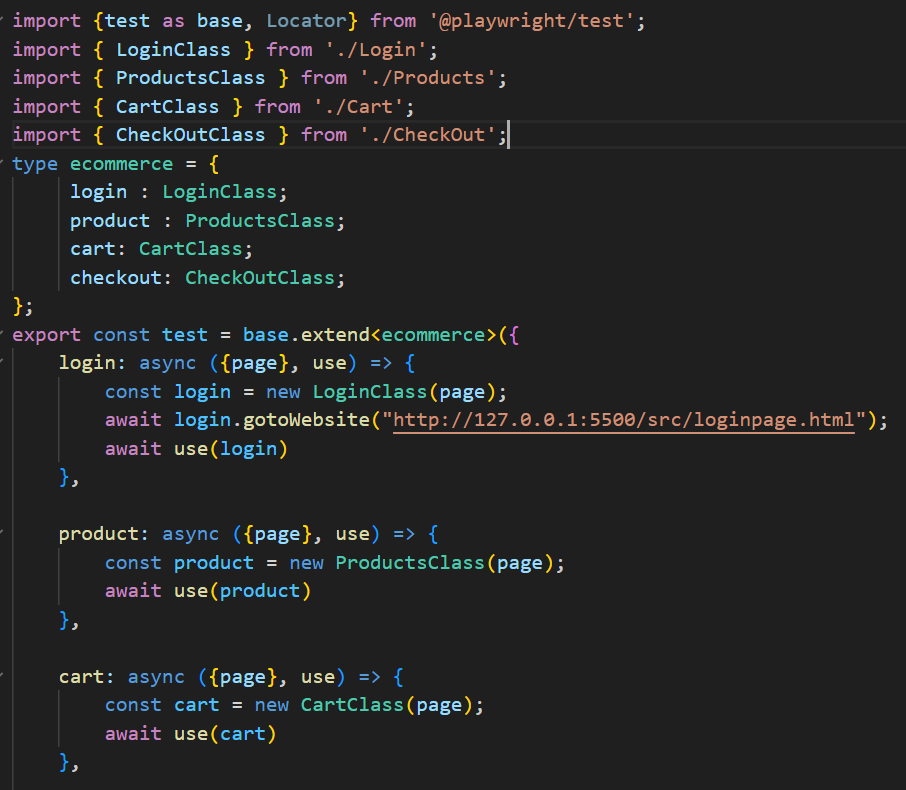
**MINI PROJECT DOCUMENTATION**

**ECOMMERSE WEBSITE - ESHOP**

**Test Cases and Fixtures:**

This code defines a custom Playwright test setup using fixtures for an e-commerce application. It creates a ecommerce type that includes instances of Login Class, Products Class, Cart Class, and Check Out Class, each representing different parts of the application. The test object is extended from the Playwright base test, and it initializes these classes with the Playwright page object. The login fixture navigates to the login page before use, while the others are simply instantiated and passed to the tests. This structure promotes modular, reusable, and readable test code. The expect assertion library is also re-exported for convenience. It supports a clean and scalable Playwright testing strategy.

**Fixture:**

****

**Fixtures Classes:**

1. **Login Class**
2. **Product class**
3. **Checkout Class**
4. **Cart Class**
5. **Order Success Class**

**Login Class:**

****

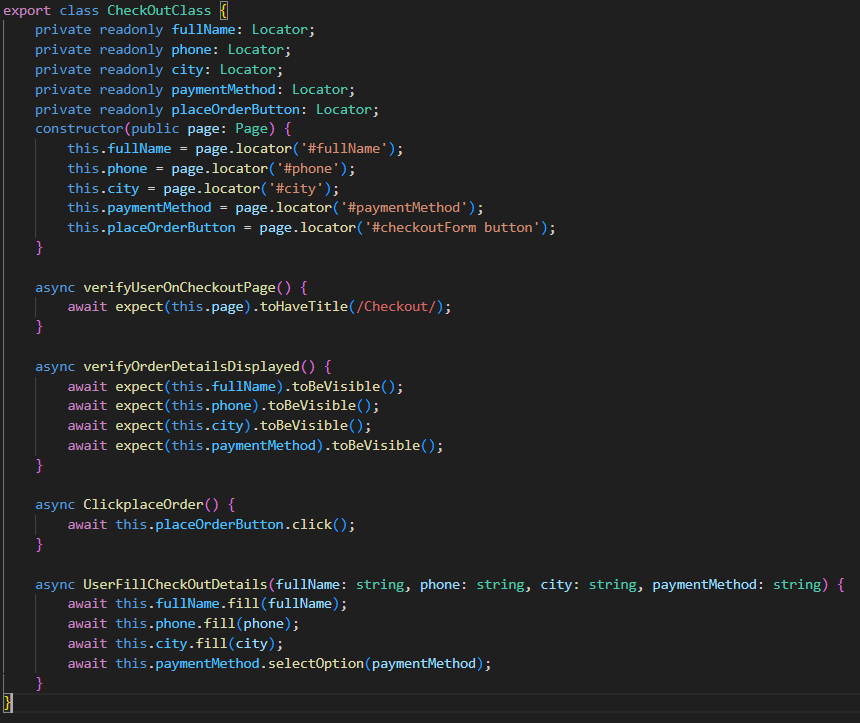
**Product Class:**

****

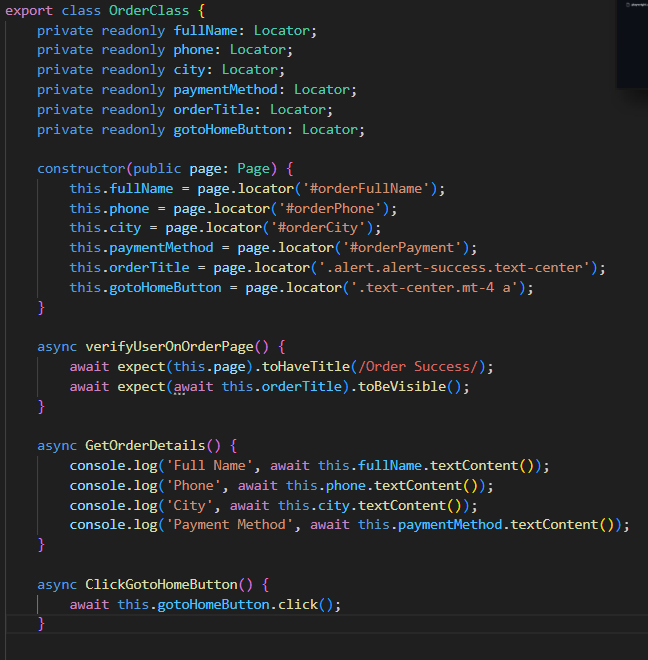
**Cart Class:**

****

**Check Out Class:**

****

**Order Success Class:**

****

**TEST CASES**

**Login Page Test Cases:**

1. Verify user on login page

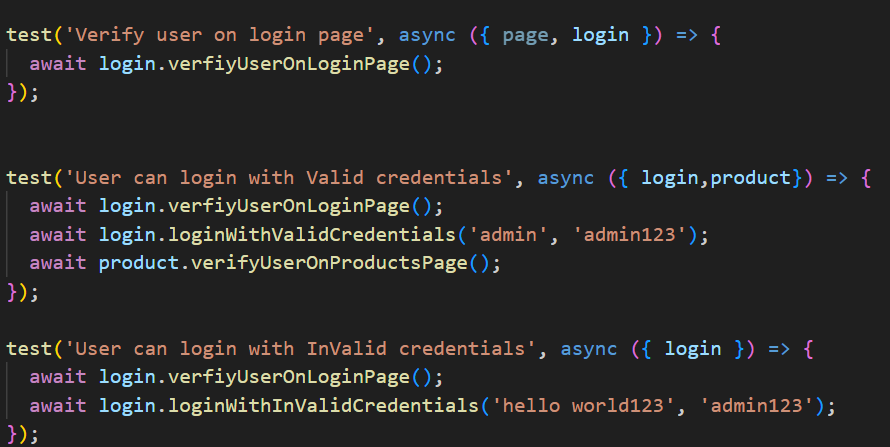
This test checks if the login page loads correctly and all expected elements (like input fields or titles) are visible. It ensures the application starts from the correct entry point before any login attempt is made.

2. User can login with valid credentials

This test enters correct login credentials (admin / admin123) and verifies that the user is successfully authenticated. After login, it checks that the user is navigated to the products page, confirming both login success and redirection.

3. User can login with invalid credentials

This test provides incorrect login credentials (hello world123 / admin123) to simulate a failed login attempt. It verifies that the login fails as expected and an error message or validation feedback is shown, ensuring proper handling of invalid input.

****

**Product page Test Cases:**

1. Verify user on Products Page

Ensures that after a successful login, the user lands on the Products Page and the cart icon is visible, indicating that the page has loaded correctly.

2. User can add product to cart

Verifies that a product can be added to the cart and that the cart count updates correctly. It also tests removing the product afterward.

3. User can remove product from cart

Checks that a product can be added and then removed from the cart, validating the cart’s dynamic behavior in response to user actions.

4. Verify user can navigate to cart page

Confirms that the user can access the cart page after login and that the page title matches “Cart Page,” ensuring proper navigation.

5. Verify the product count

Validates that the cart is initially empty (with a count of 0) after login and landing on the Products Page.

6. Verify the cart icon

Simply checks for the presence of the cart icon on the Products Page, confirming that key UI elements are visible.



**Cart page Test Cases:**

1. Verify user on Cart Page

Confirms that after logging in and navigating to the cart, the user is correctly taken to the Cart Page with the expected page title.

2. Verify total count price

Adds 8 products to the cart and checks that the cart count reflects this. Then navigates to the cart and verifies that the total price displayed matches the expected value $173.92.

3. Verify the total items count

Adds 4 products to the cart, navigates to the cart page, and asserts that 4 individual cart items are displayed correctly in the UI.

4. Remove product from cart

Adds 4 products, navigates to the cart, then removes one item and checks that only 3 items remain in the cart list, ensuring item removal updates the cart as expected.

5. Verify the proceed to checkout button

Adds a product, navigates to the cart, and verifies that the Proceed to Checkout button is clickable and functioning properly.



**Check Out Page Test Cases:**

1. Verify user on Checkout Page

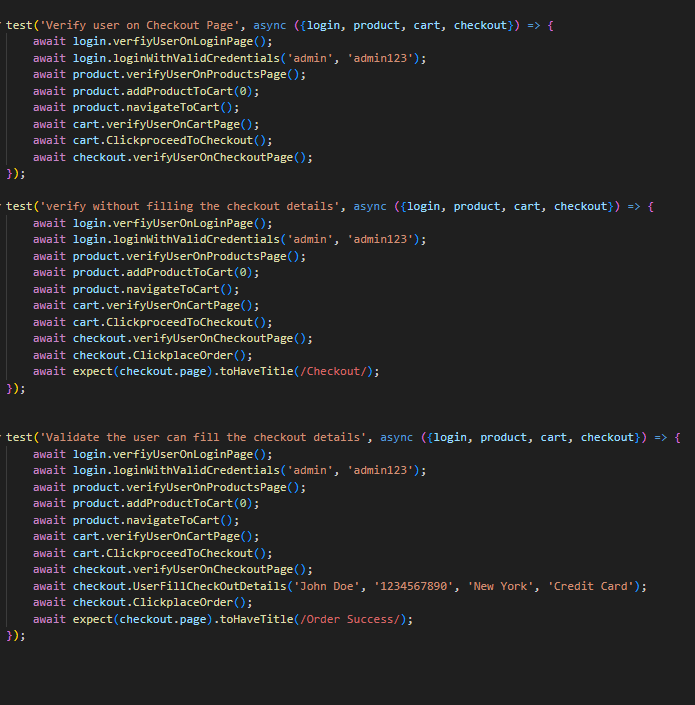
After logging in, adding a product, and navigating through the cart, this test confirms that the user reaches the Checkout Page successfully.

2. Verify without filling the checkout details

This test checks that if the user tries to place an order without entering any checkout information, they remain on the Checkout Page, implying validation is working correctly.

3. Validate the user can fill the checkout details

Simulates a full checkout process by filling in valid details (name, phone, city, payment method) and placing the order, then verifies navigation to the Order Success Page.

****

**Order Page Test Cases:**

1. Verify user on Order Success Page

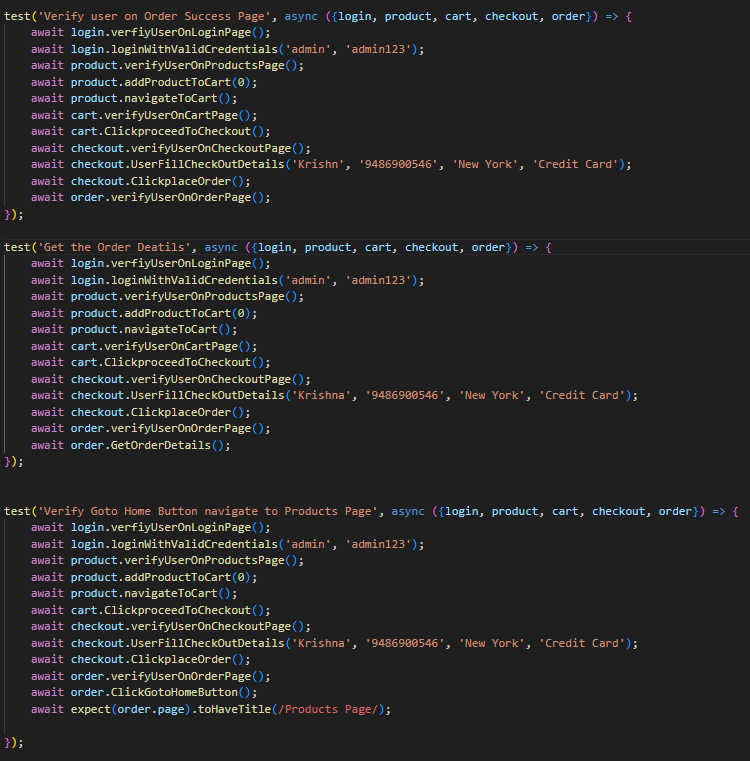
This test goes through the complete checkout flow and verifies that after placing an order, the user lands on the Order Success Page, confirming order completion.

2. Get the Order Details

Performs the same steps as above but additionally verifies that the user can retrieve and view the order details (e.g., name, city, payment method), confirming proper display of stored data.

3. Verify "Goto Home" button navigates to Products Page

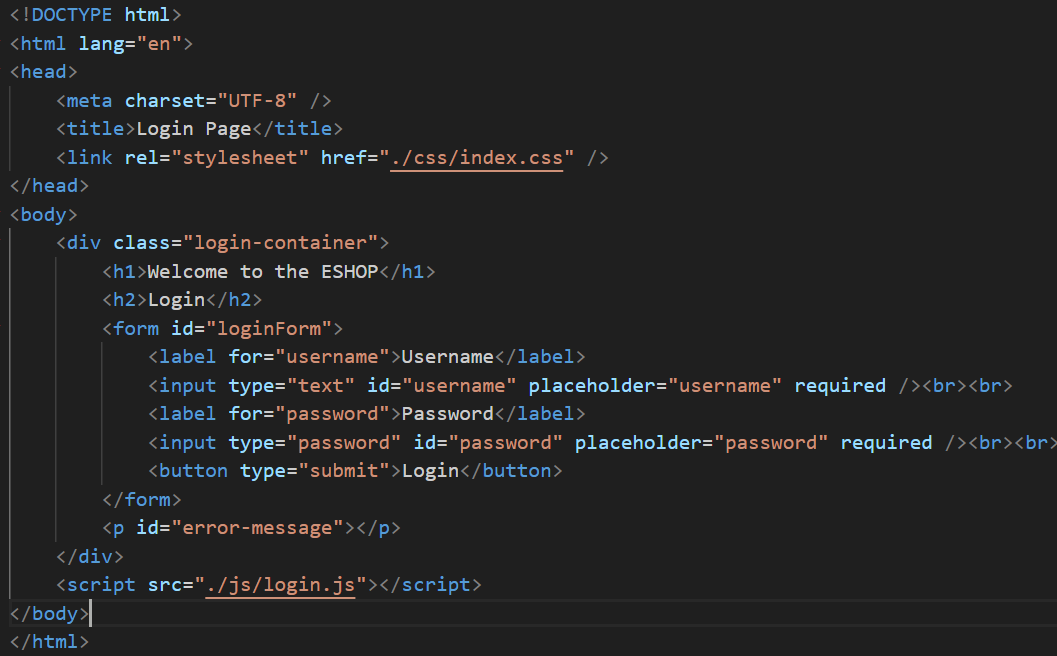
After order placement, this test confirms that clicking the "Goto Home" button redirects the user back to the Products Page, ensuring navigation flow is user-friendly and functional.

****

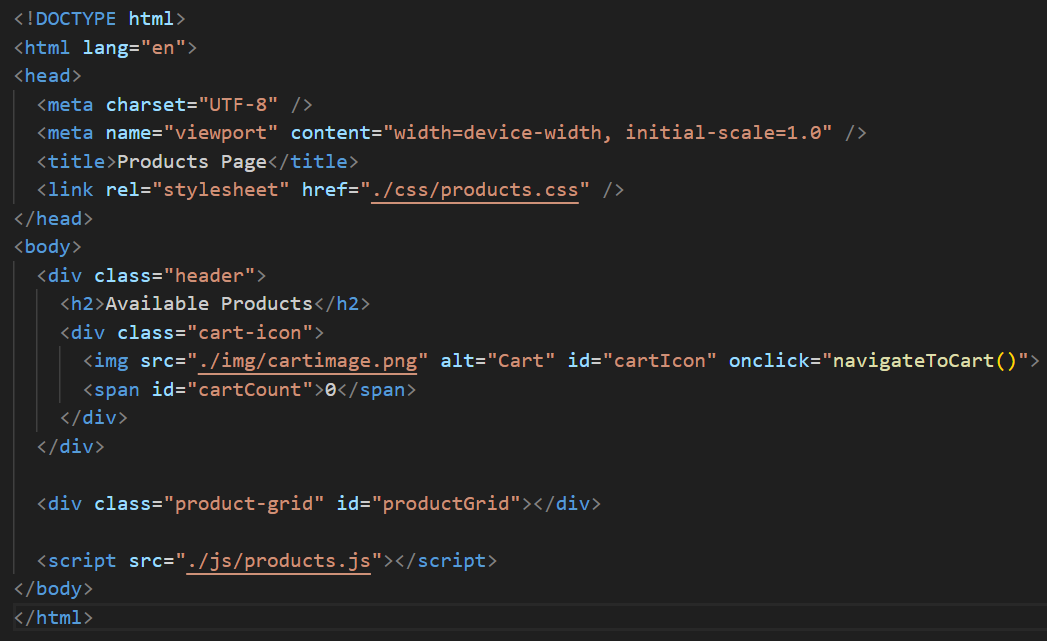
**Number of pages:**

1. Login Page
2. Products Page
3. Cart Page
4. Checkout Page
5. Order Success Page

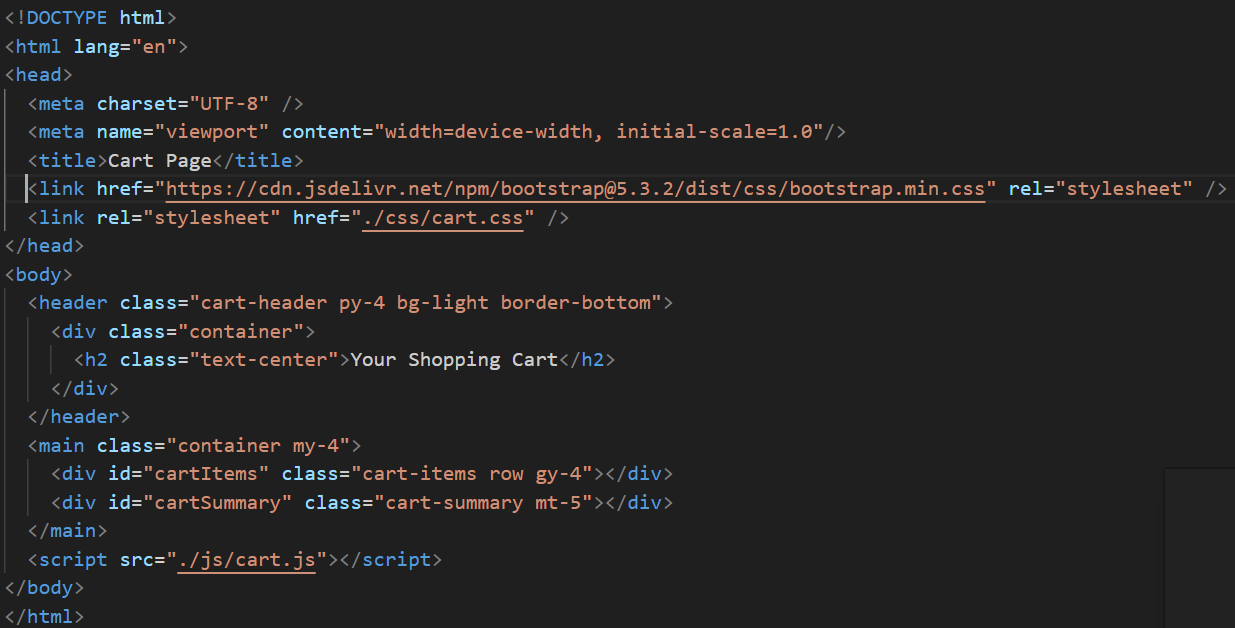
Login Page HTML:



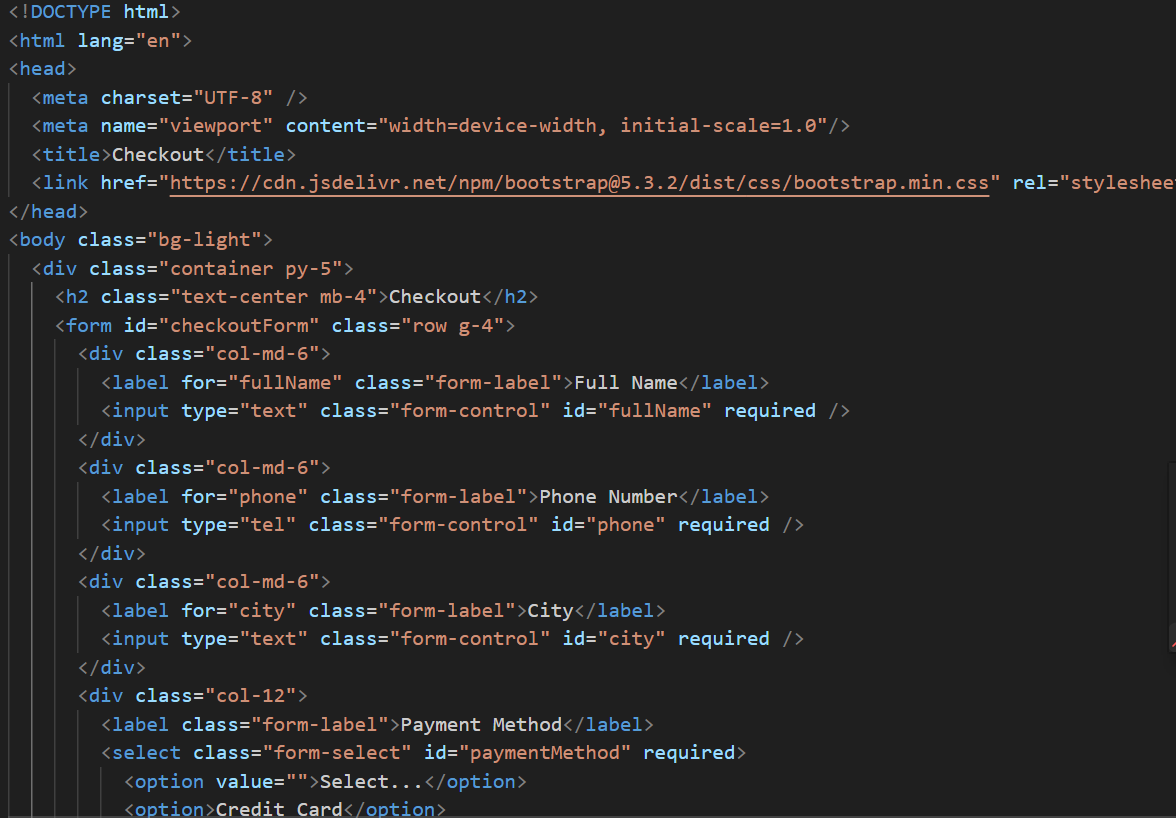
Products Page HTML:



Cart Page HTML:



Checkout Page HTML:



Order Page HTML:

