UI Testing

The objective of this framework is to automate the testing of essential user actions including adding items to a cart, removing items and completing orders. Playwright was used for end-to-end testing. The scope of this project is centered around testing a typical user ensuring that login and cart functionality and checkout processes work seamlessly.

- **Framework Architecture**

The framework structured in a modular way to keep things organized and maintainable. Each user interaction is represented by a separate test case, allowing us to focus on individual functionalities while maintaining clarity.

- **Modular Test Structure**: The framework is broken down into test cases for specific actions:
  - Login to site

    Each test starts by navigating to the site and logging in as a standard user. I added extra validation steps (such as checking the visibility of the logo and title) to ensure that the page loaded correctly and that the test started on a stable environment.
  - Adding a single item to cart
  - Adding multiple items to cart

    In the first test case, user adds a single item to cart. In the second case, multiple items are added. After each action, I validated cart to ensure it reflected the correct number of items.
  - Removing items from cart

    This test case handles removing items from cart. After removal, the cart badge disappears.
  - Proceeding to order completion

    After adding items to cart, user proceeds to checkout. The test validates cart contents, navigates through checkout flow, fills in personal details and completes the order. Finally, I validated the appearance of a confirmation message to ensure the order was successful.

- **Assertions**: Assertions used to validate each key step in the process. For instance, when an item is added to the cart, I check that the cart icon updates accordingly. This ensures that the system behaves as expected.

- **Framework Design Decisions**

- Playwright: I chose Playwright for its ability to handle end-to-end testing efficiently. Playwright's auto-wait feature ensures that the actions in the test don't fail due to timing issues, making tests more stable.
- Test Reliability: I ensured test reliability by validating not only the core actions but also extra page elements such as logos and titles. This allows for early detection of any navigation or rendering issues.
- Scalability: The framework can easily scale to include more complex user flows or additional functionality. For example, additional items can be added to the cart, or I could extend the checkout process to include different payment methods without changing the existing structure.
- Maintainability: The code can be maintained easily, especially with the possibility of moving to a Page Object Model structure, which would isolate page-specific elements from the core logic. This makes future updates less prone to errors when the UI changes.