Software Testing And Automation Project Report

Software Testing Using Playwright For DemoBlaze.com

Group Members :

Hussain Raza (SP23-MSCS-0012),
Usman Bin Hamid(SP23-MSCS-0014)
Javeria Ahsan (SP23-MSCS-0036),

Table of Contents

1.Introduction

The aim of this project is to implement comprehensive test cases for the e-commerce website Demoblaze using Playwright. Playwright is a powerful and versatile end-to-end testing framework developed by Microsoft that allows for automated testing of web applications across various browsers. By leveraging Playwright's capabilities, this project seeks to ensure that Demoblaze operates seamlessly and provides an optimal user experience.

Demoblaze, a sample e-commerce platform, features a wide range of functionalities typical of online shopping websites, such as product browsing, user authentication, and purchase processing. Testing these features is crucial to guarantee their reliability and performance.

Automated testing with Playwright provides several advantages, including the ability to run tests in parallel across multiple browsers, simulate user actions accurately, and capture detailed diagnostic information in case of failures.

The project will follow a structured approach, starting with the identification of critical test scenarios, followed by the implementation of Playwright scripts, and culminating in the execution and analysis of test results.

2.Objectives

- **Identify Critical Test Scenarios**:

- Catalog key functionalities of the Demoblaze website.
- Prioritize features such as product browsing, user authentication, and purchase processes.

- **Develop Comprehensive Test Cases**:

- Create detailed test cases covering various user interactions and workflows.
- Include edge cases and typical user behavior scenarios.

- **Implement Test Scripts Using Playwright**:

- Write automated test scripts utilizing Playwright’s robust API.
- Ensure scripts are cross-browser compatible, covering Chrome, Firefox, and Safari.

- **Simulate User Actions Accurately**:

- Automate user actions like clicking, form filling, and navigation.
- Verify UI elements and their states under different conditions.

- **Execute Tests and Capture Results**:

- Run the test scripts in different environments to ensure consistency.
- Capture and log test results, including any errors or performance issues.

- **Analyze Test Results and Report Findings**:

- Analyze test outcomes to identify bugs and performance bottlenecks.
- Provide detailed reports with insights for improvements and bug fixes.

3. Data Overview

The project focuses on the Demoblaze website, a sample e-commerce platform. Key data points include user actions such as product searches, category navigation, user registration, login processes, and shopping cart interactions. Additionally, backend responses and UI element states will be examined. Test data will encompass a variety of products, user credentials, and transaction scenarios to simulate real-world usage. The aim is to ensure that all critical functionalities are thoroughly tested and validated. Playwright will be used to automate these tests, capturing detailed results and performance metrics for analysis and reporting

4. Methodology

1.1 PageClasses

Page classes are a design pattern used to encapsulate the interaction logic for a specific web page or component. This approach, often referred to as the Page Object Model (POM), helps in organizing test code, making it more maintainable, reusable, and readable. Each page class corresponds to a page or a significant component in the application and includes methods to interact with elements on that page

- **LoginClass:** The `LoginClass` encapsulates all login-related interactions for the Demoblaze website, providing methods to navigate to the login page, enter credentials, and submit the login form.
- **SignUpClass:** The `SignUpClass` manages all sign-up interactions for the Demoblaze website. It includes methods to navigate to the sign-up page, enter user details, and submit the registration form. This encapsulation streamlines test creation, ensuring maintainability and readability by isolating sign-up functionality from the main test scripts.
- **ContactUsClass:** The `ContactUsClass` handles interactions with the contact form on the Demoblaze website. It includes methods to navigate to the contact page, fill out user information, and submit inquiries. This class enhances test maintainability and readability by encapsulating contact form functionalities, allowing streamlined and organized test automation.
- **OpenUrlClass:** The `HomePageClass` encapsulates interactions with the homepage of the Demoblaze website. It includes methods to navigate the main sections, access featured products, and interact with key homepage elements. This class enhances test modularity and maintainability by isolating homepage functionalities, streamlining the creation and maintenance of test scripts.

- **FooterClass :** The `FooterExistClass` verifies the presence and content of the footer on the Demoblaze website. It includes methods to check for specific footer elements and ensure they are correctly displayed. This class improves test accuracy and completeness by ensuring that critical footer information is consistently validated across test cases

- **AddToCartClass** : It integrates methods for navigating to product pages, clicking "Add to Cart" buttons, and validating successful additions. This class enhances test automation efficiency and ensures accurate cart interaction testing.

- **PlaceAnOrderClass :** It includes methods to navigate through checkout steps, enter shipping and payment details, and confirm the order. This class ensures thorough testing of the ordering process, enhancing reliability and user experience validation.

1.2 JSON

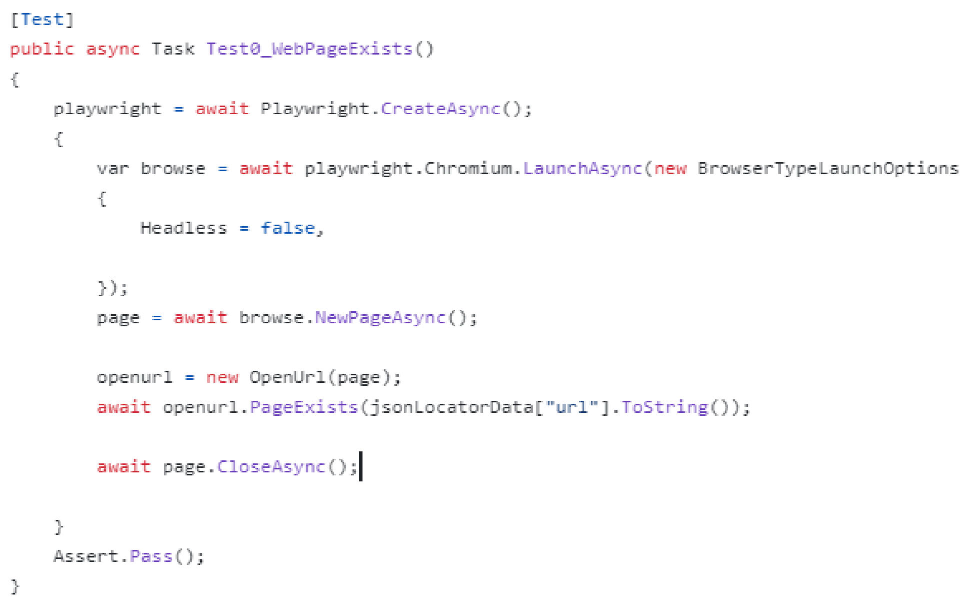
- **Data.JSON:**

In the project to implement test cases for https://www.demoblaze.com/ with Playwright, the data.json file serves as a crucial component. It encapsulates structured data in JSON format, enabling efficient organization and transmission of test data. This file facilitates seamless integration with Playwright, streamlining test execution and ensuring robust testing procedures.

- **Locator.JSON:** The Locator.JSON file holds paramount significance. It contains essential locators and element identifiers crucial for test automation. Leveraging these locators, Playwright efficiently interacts with elements on the web page, enhancing test script reliability and maintainability.

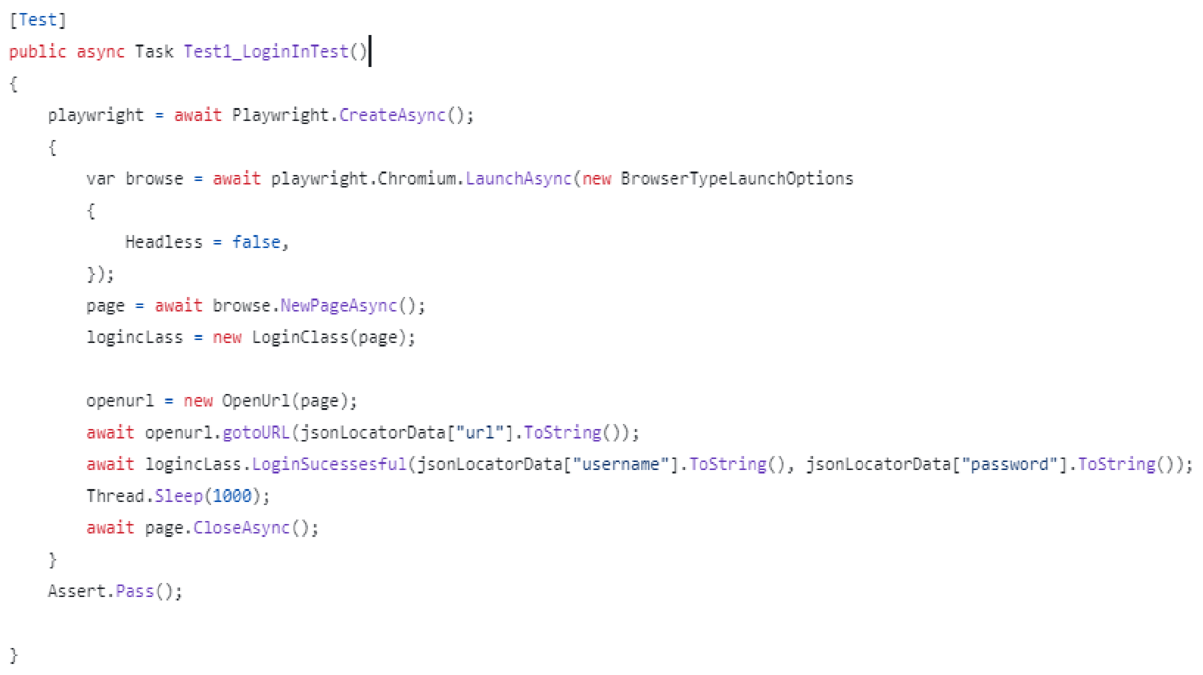
1.3 Unit Testcases : It is to implement test cases for https://www.demoblaze.com/ with Playwright, the Unit Testcases file assumes pivotal importance. It houses meticulously crafted test cases designed to validate the functionality of the website's features. These test cases, tailored for Playwright, ensure comprehensive coverage and robust validation of the application's behavior.

- **Test0_WebPageExists()**:It verifies the existence and accessibility of the web page, ensuring seamless navigation and functionality validation.



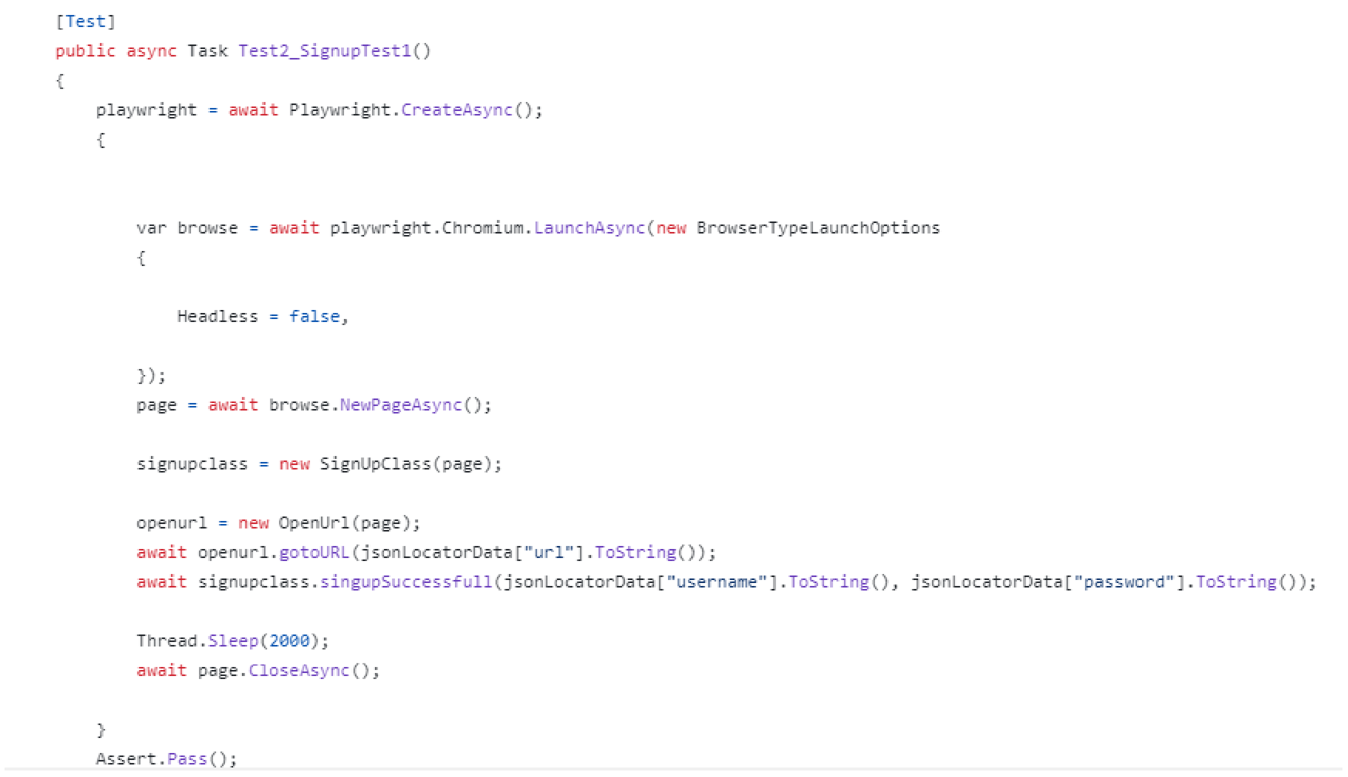
```
[Test]
public async Task Test0_WebPageExists()
{
    playwright = await Playwright.CreateAsync();
    {
        var browse = await playwright.Chromium.LaunchAsync(new BrowserTypeLaunchOptions
        {
            Headless = false,

        });
        page = await browse.NewPageAsync();

        openurl = new OpenUrl(page);
        await openurl.PageExists(jsonLocatorData["url"].ToString());

        await page.CloseAsync();

    }
    Assert.Pass();
}
```

- **Test1_LoginInTest() : It** validates the login functionality. This test ensures smooth user authentication, enhancing the website's security and user experience.



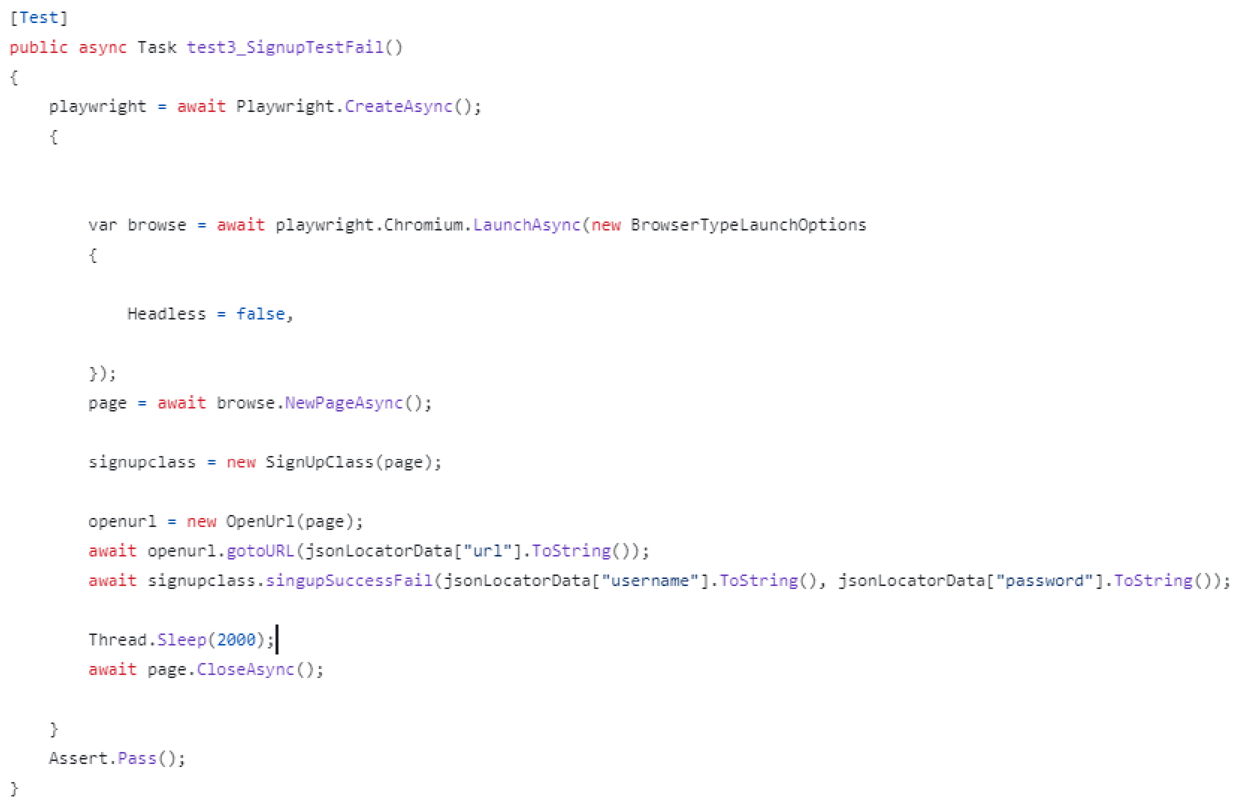
```
[Test]
public async Task Test1_LoginInTest()
{
    playwright = await Playwright.CreateAsync();
    {
        var browse = await playwright.Chromium.LaunchAsync(new BrowserTypeLaunchOptions
        {
            Headless = false,
        });
        page = await browse.NewPageAsync();
        logincLass = new LoginClass(page);

        openurl = new OpenUrl(page);
        await openurl.gotoURL(jsonLocatorData["url"].ToString());
        await logincLass.LoginSucessesful(jsonLocatorData["username"].ToString(), jsonLocatorData["password"].ToString());
        Thread.Sleep(1000);
        await page.CloseAsync();
    }
    Assert.Pass();

}
```

- **Test2_SignupTest1():** It scrutinizes the signup process. It verifies the seamless registration flow, ensuring users can sign up successfully, enhancing the website's functionality and user experience.



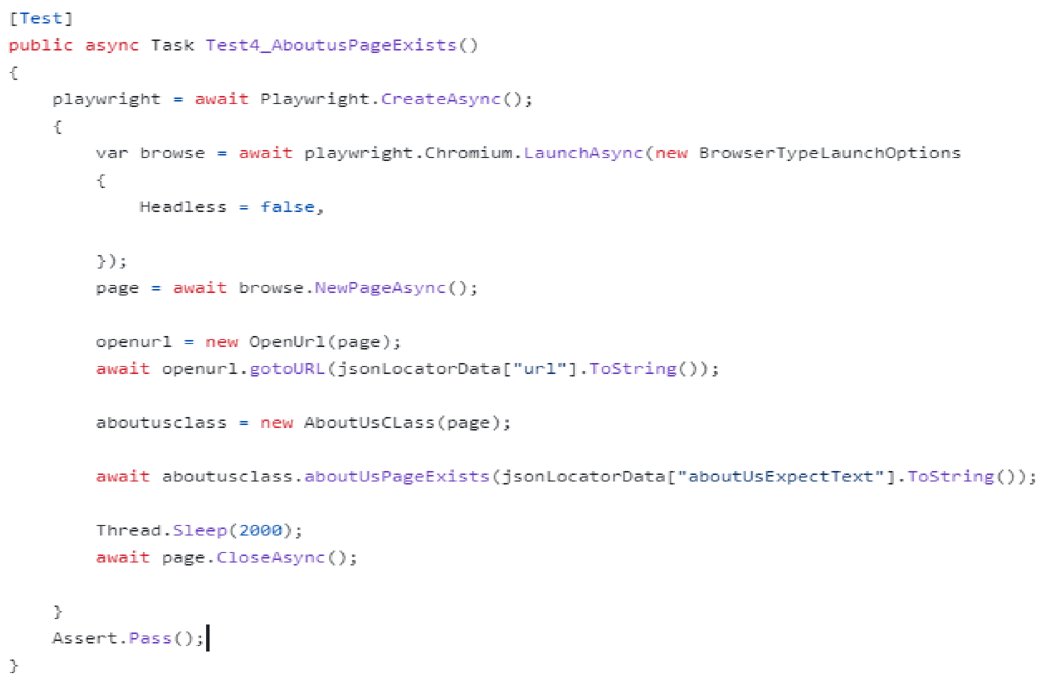
```
[Test]
public async Task Test2_SignupTest1()
{
    playwright = await Playwright.CreateAsync();
    {


        var browse = await playwright.Chromium.LaunchAsync(new BrowserTypeLaunchOptions
        {

            Headless = false,

        });
        page = await browse.NewPageAsync();

        signupclass = new SignUpClass(page);

        openurl = new OpenUrl(page);
        await openurl.gotoURL(jsonLocatorData["url"].ToString());
        await signupclass.singupSuccessfull(jsonLocatorData["username"].ToString(), jsonLocatorData["password"].ToString());

        Thread.Sleep(2000);
        await page.CloseAsync();

    }
    Assert.Pass();
```

- **test3_SignupTestFail():** It assesses the failure scenarios during the signup process. It ensures robust error handling, enhancing the website's reliability and user experience.



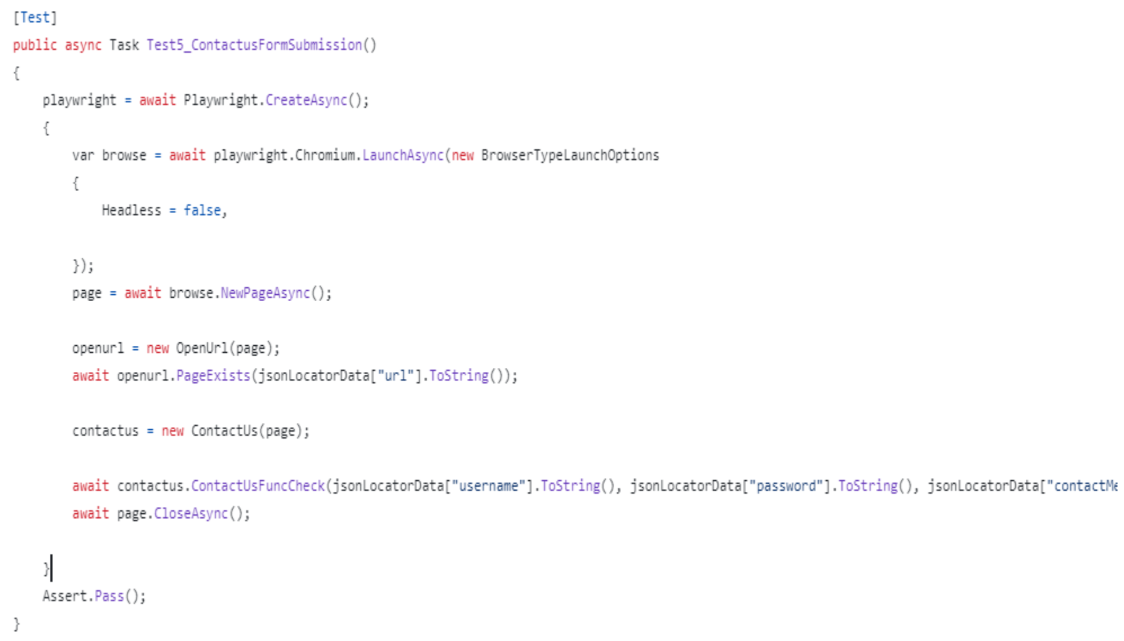
```
[Test]
public async Task test3_SignupTestFail()
{
    playwright = await Playwright.CreateAsync();
    {


        var browse = await playwright.Chromium.LaunchAsync(new BrowserTypeLaunchOptions
        {

            Headless = false,

        });
        page = await browse.NewPageAsync();

        signupclass = new SignUpClass(page);

        openurl = new OpenUrl(page);
        await openurl.gotoURL(jsonLocatorData["url"].ToString());
        await signupclass.singupSuccessFail(jsonLocatorData["username"].ToString(), jsonLocatorData["password"].ToString());

        Thread.Sleep(2000);
        await page.CloseAsync();

    }
    Assert.Pass();
}
```

- **Test4_AboutusPageExists():** It verifies the presence and accessibility of the "About Us" page. This ensures accurate navigation and enhances user engagement with the website's content.



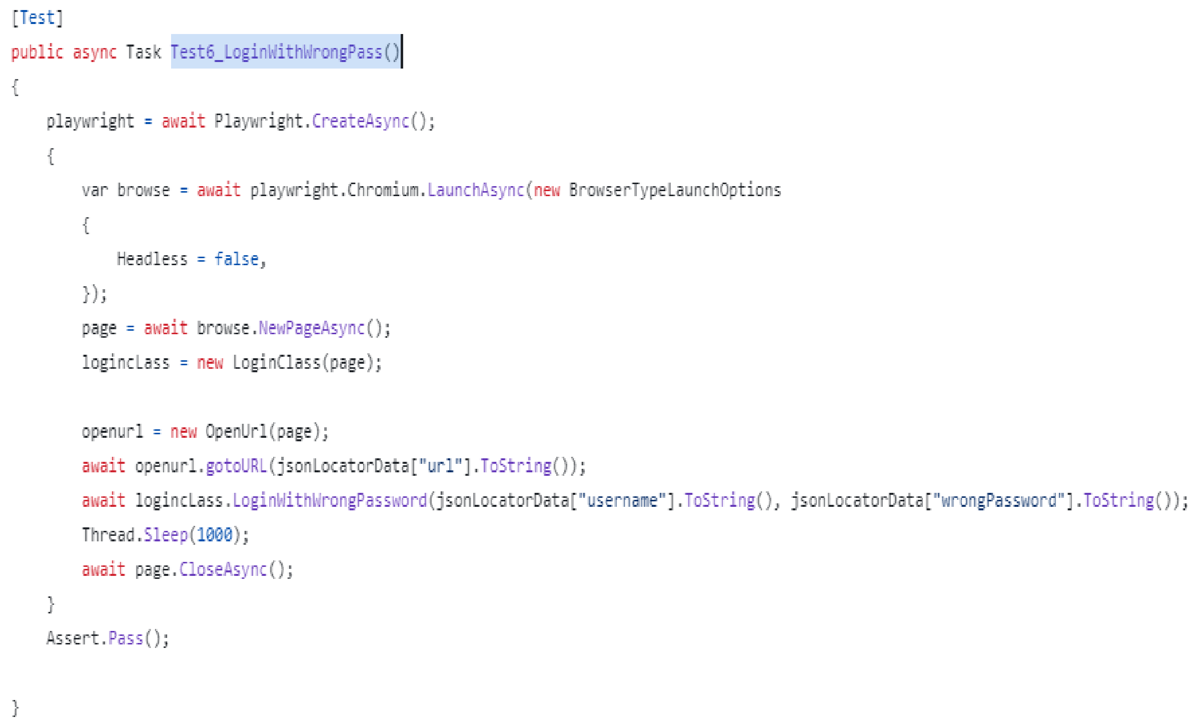
```
[Test]
public async Task Test4_AboutusPageExists()
{
    playwright = await Playwright.CreateAsync();
    {
        var browse = await playwright.Chromium.LaunchAsync(new BrowserTypeLaunchOptions
        {
            Headless = false,

        });
        page = await browse.NewPageAsync();

        openurl = new OpenUrl(page);
        await openurl.gotoURL(jsonLocatorData["url"].ToString());

        aboutusclass = new AboutUsCLass(page);

        await aboutusclass.aboutUsPageExists(jsonLocatorData["aboutUsExpectText"].ToString());

        Thread.Sleep(2000);
        await page.CloseAsync();

    }
    Assert.Pass();
}
```

- **Test5_ContactusFormSubmission() :** It focuses on automating and validating the Contact Us form for functionality and error handling



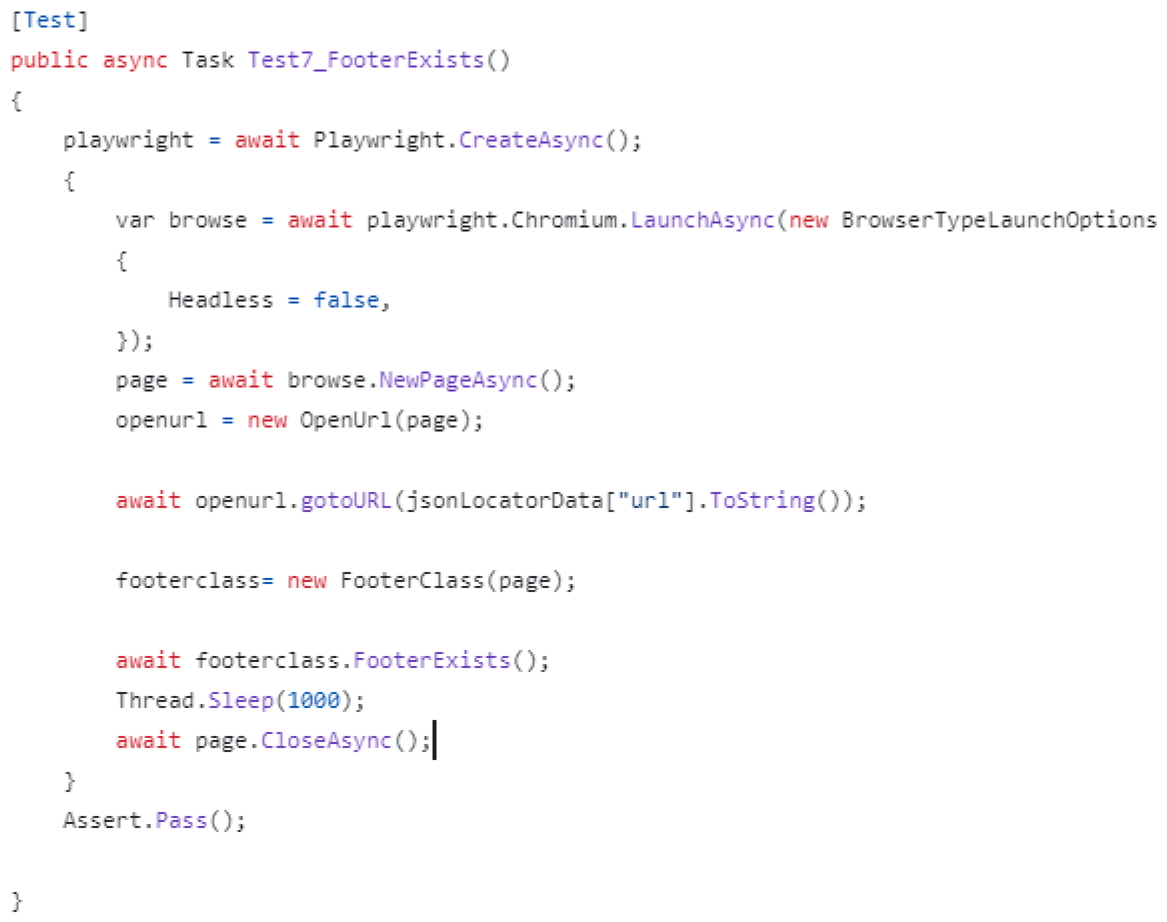
```
[Test]
public async Task Test5_ContactusFormSubmission()
{
    playwright = await Playwright.CreateAsync();
    {
        var browse = await playwright.Chromium.LaunchAsync(new BrowserTypeLaunchOptions
        {
            Headless = false,

        });
        page = await browse.NewPageAsync();

        openurl = new OpenUrl(page);
        await openurl.PageExists(jsonLocatorData["url"].ToString());

        contactus = new ContactUs(page);

        await contactus.ContactUsFuncCheck(jsonLocatorData["username"].ToString(), jsonLocatorData["password"].ToString(), jsonLocatorData["contactMe
        await page.CloseAsync();

    }
    Assert.Pass();
}
```

- **Test6_LoginWithWrongPass():** It focuses on testing the login functionality with incorrect passwords to verify error handling.

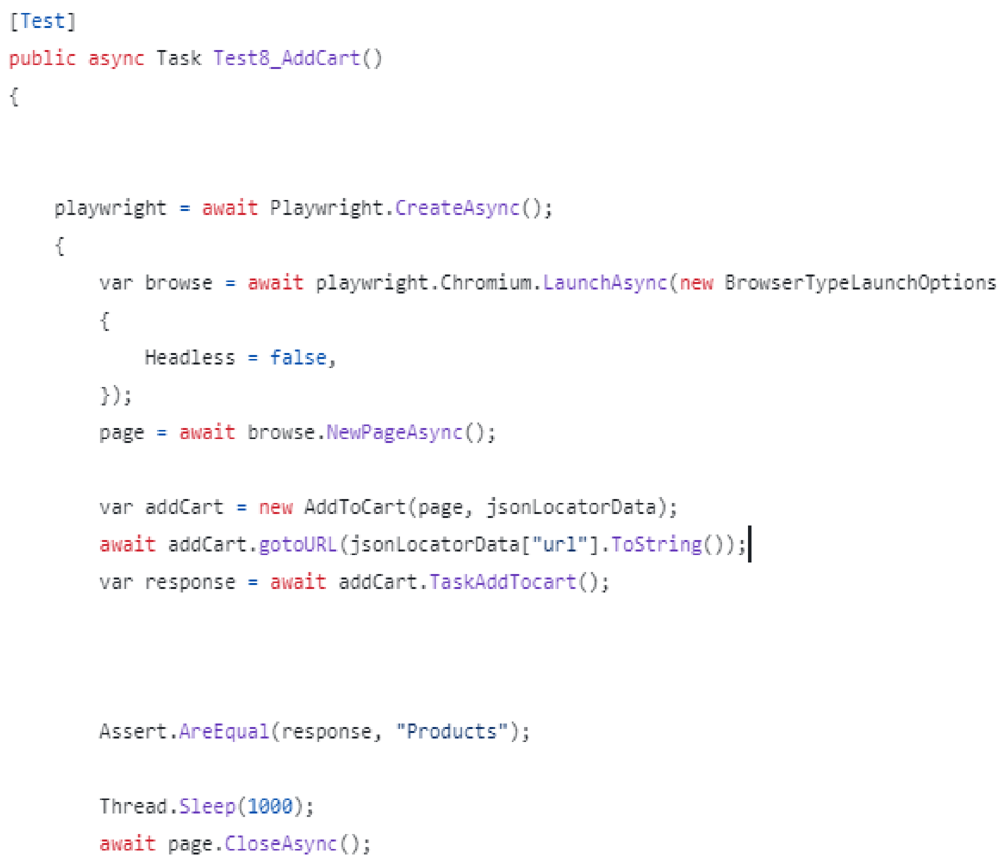


```
[Test]
public async Task Test6_LoginWithWrongPass()
{
    playwright = await Playwright.CreateAsync();
    {
        var browse = await playwright.Chromium.LaunchAsync(new BrowserTypeLaunchOptions
        {
            Headless = false,
        });
        page = await browse.NewPageAsync();
        loginclass = new LoginClass(page);

        openurl = new OpenUrl(page);
        await openurl.gotoURL(jsonLocatorData["url"].ToString());
        await loginclass.LoginWithWrongPassword(jsonLocatorData["username"].ToString(), jsonLocatorData["wrongPassword"].ToString());
        Thread.Sleep(1000);
        await page.CloseAsync();
    }
    Assert.Pass();

}
```

- **Test7_FooterExists() :** It focuses on verifying the existence and correct display of the website's footer section.

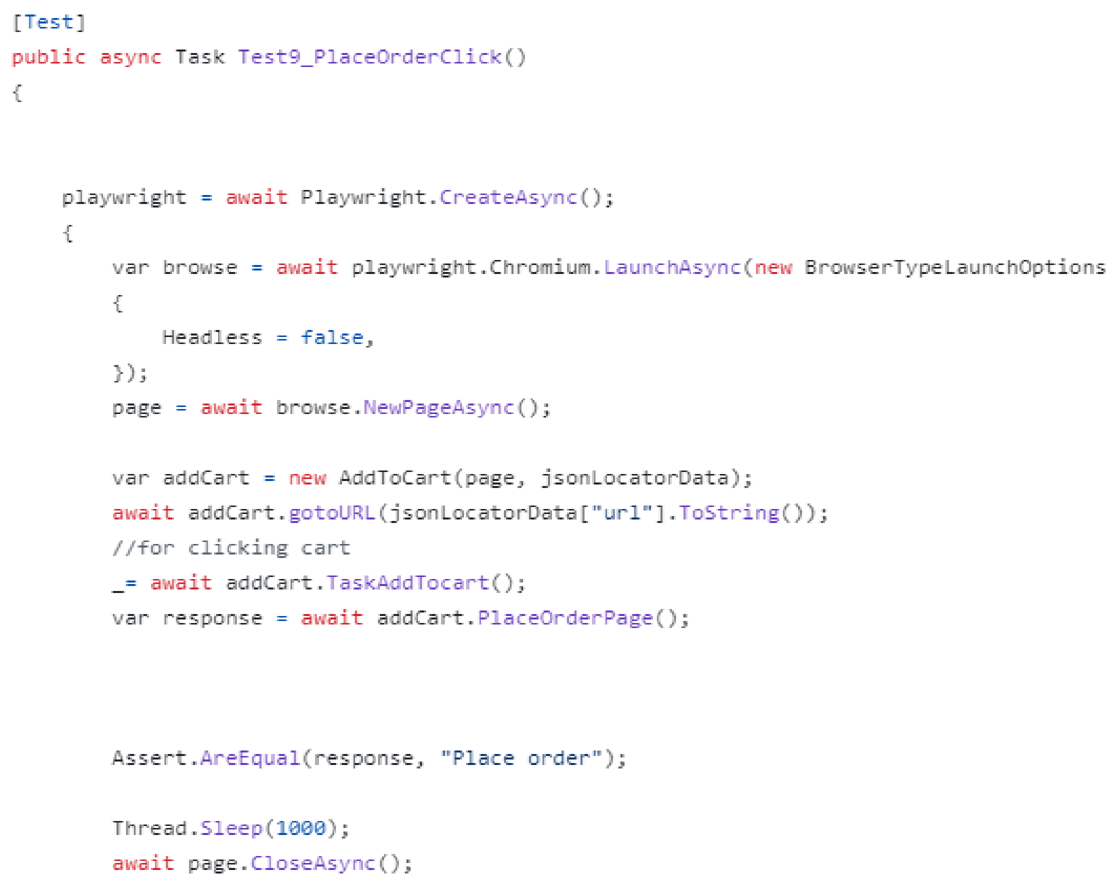

```
[Test]
public async Task Test7_FooterExists()
{
    playwright = await Playwright.CreateAsync();
    {
        var browse = await playwright.Chromium.LaunchAsync(new BrowserTypeLaunchOptions
        {
            Headless = false,
        });
        page = await browse.NewPageAsync();
        openurl = new OpenUrl(page);

        await openurl.gotoURL(jsonLocatorData["url"].ToString());

        footerclass= new FooterClass(page);

        await footerclass.FooterExists();
        Thread.Sleep(1000);
        await page.CloseAsync();
    }
    Assert.Pass();

}
```

- **Test8_AddCart() :** It validate the functionality of adding items to the cart, ensuring accuracy and reliability in the process.



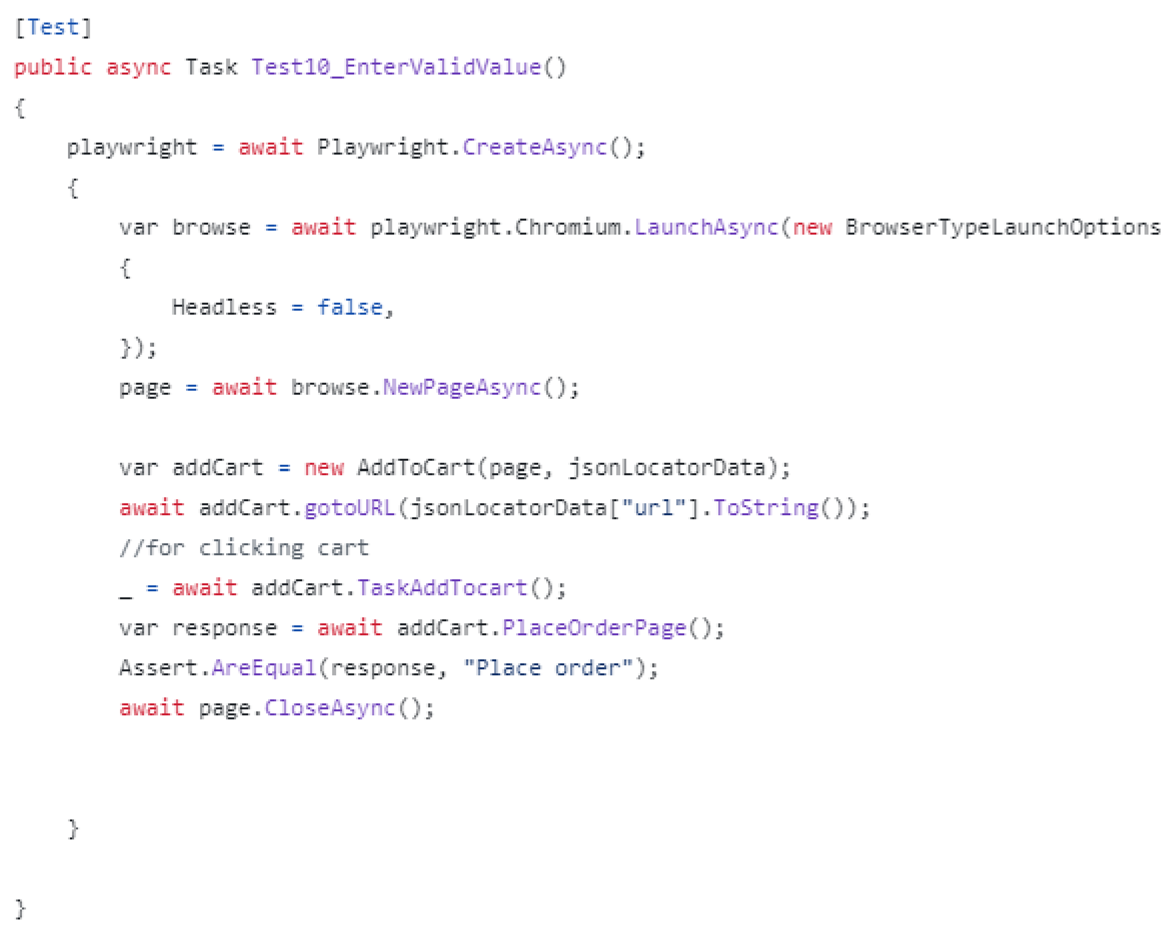
```
[Test]
public async Task Test8_AddCart()
{


    playwright = await Playwright.CreateAsync();
    {
        var browse = await playwright.Chromium.LaunchAsync(new BrowserTypeLaunchOptions
        {
            Headless = false,
        });
        page = await browse.NewPageAsync();

        var addCart = new AddToCart(page, jsonLocatorData);
        await addCart.gotoURL(jsonLocatorData["url"].ToString());
        var response = await addCart.TaskAddTocart();



        Assert.AreEqual(response, "Products");

        Thread.Sleep(1000);
        await page.CloseAsync();
```

- **Test9_PlaceOrderClick() :** It focuses to verify the functionality of placing an order ensuring seamless user experience and reliability.



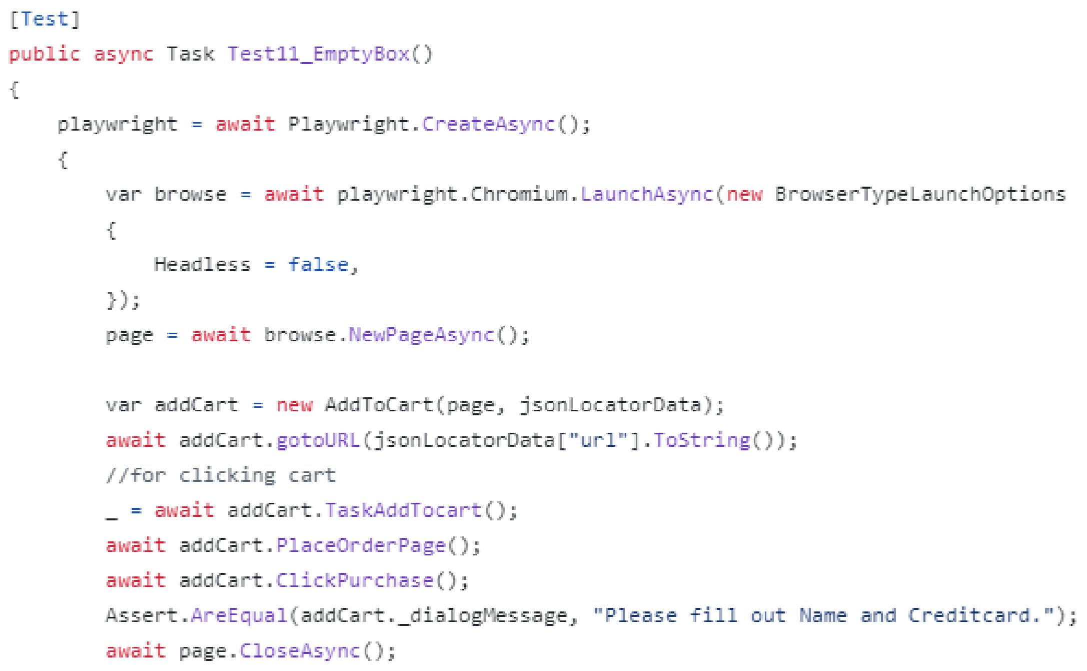
```
[Test]
public async Task Test9_PlaceOrderClick()
{


    playwright = await Playwright.CreateAsync();
    {
        var browse = await playwright.Chromium.LaunchAsync(new BrowserTypeLaunchOptions
        {
            Headless = false,
        });
        page = await browse.NewPageAsync();

        var addCart = new AddToCart(page, jsonLocatorData);
        await addCart.gotoURL(jsonLocatorData["url"].ToString());
        //for clicking cart
        _= await addCart.TaskAddTocart();
        var response = await addCart.PlaceOrderPage();



        Assert.AreEqual(response, "Place order");

        Thread.Sleep(1000);
        await page.CloseAsync();
```

- **Test10_EnterValidValue():** It focuses to verify all the valid values of placing an order for good user experience



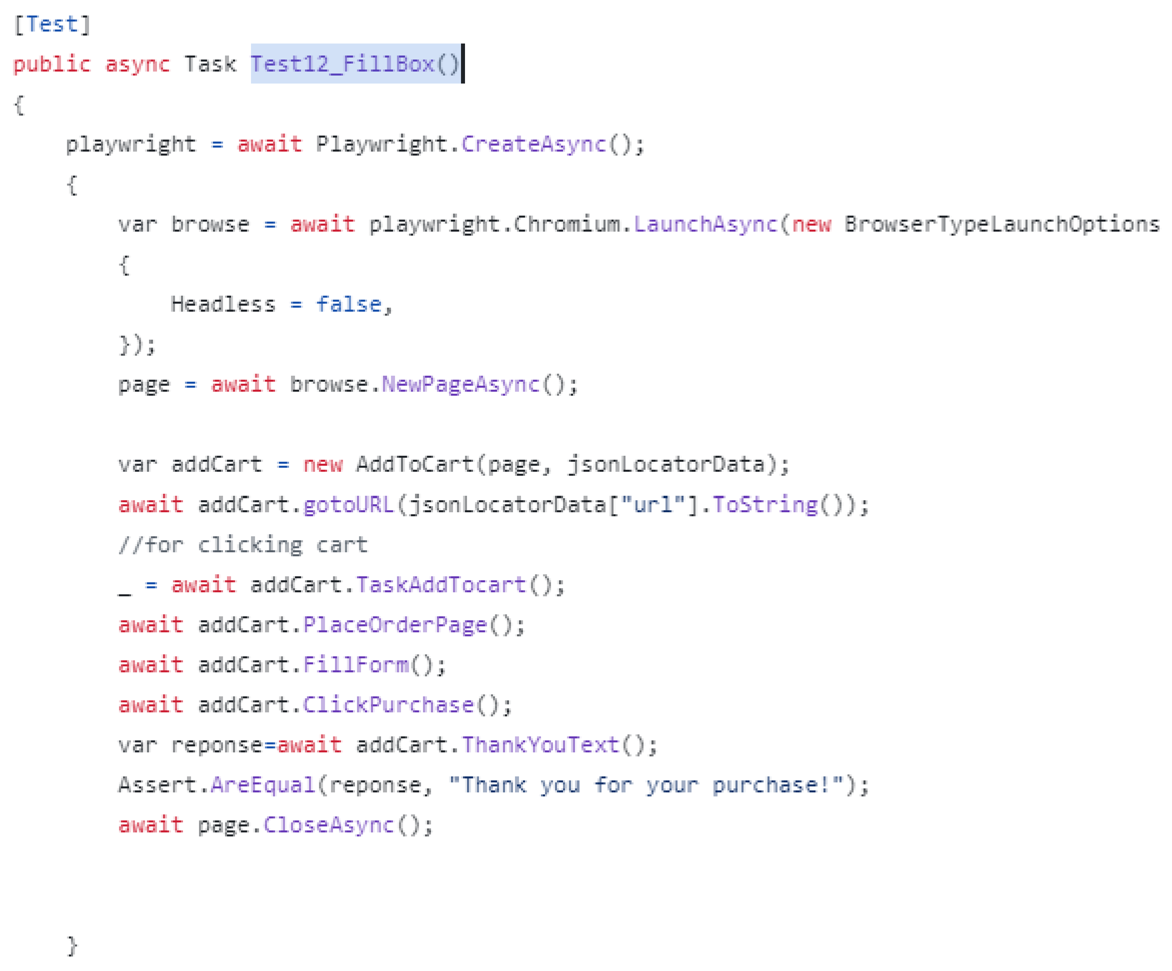
```
[Test]
public async Task Test10_EnterValidValue()
{
    playwright = await Playwright.CreateAsync();
    {
        var browse = await playwright.Chromium.LaunchAsync(new BrowserTypeLaunchOptions
        {
            Headless = false,
        });
        page = await browse.NewPageAsync();

        var addCart = new AddToCart(page, jsonLocatorData);
        await addCart.gotoURL(jsonLocatorData["url"].ToString());
        //for clicking cart
        _ = await addCart.TaskAddTocart();
        var response = await addCart.PlaceOrderPage();
        Assert.AreEqual(response, "Place order");
        await page.CloseAsync();


    }

}
```

- **Test11_EmptyBox():** It focuses to verify all the empty values of placing an order ,it may show “Please fill out Name and Creditcard details”



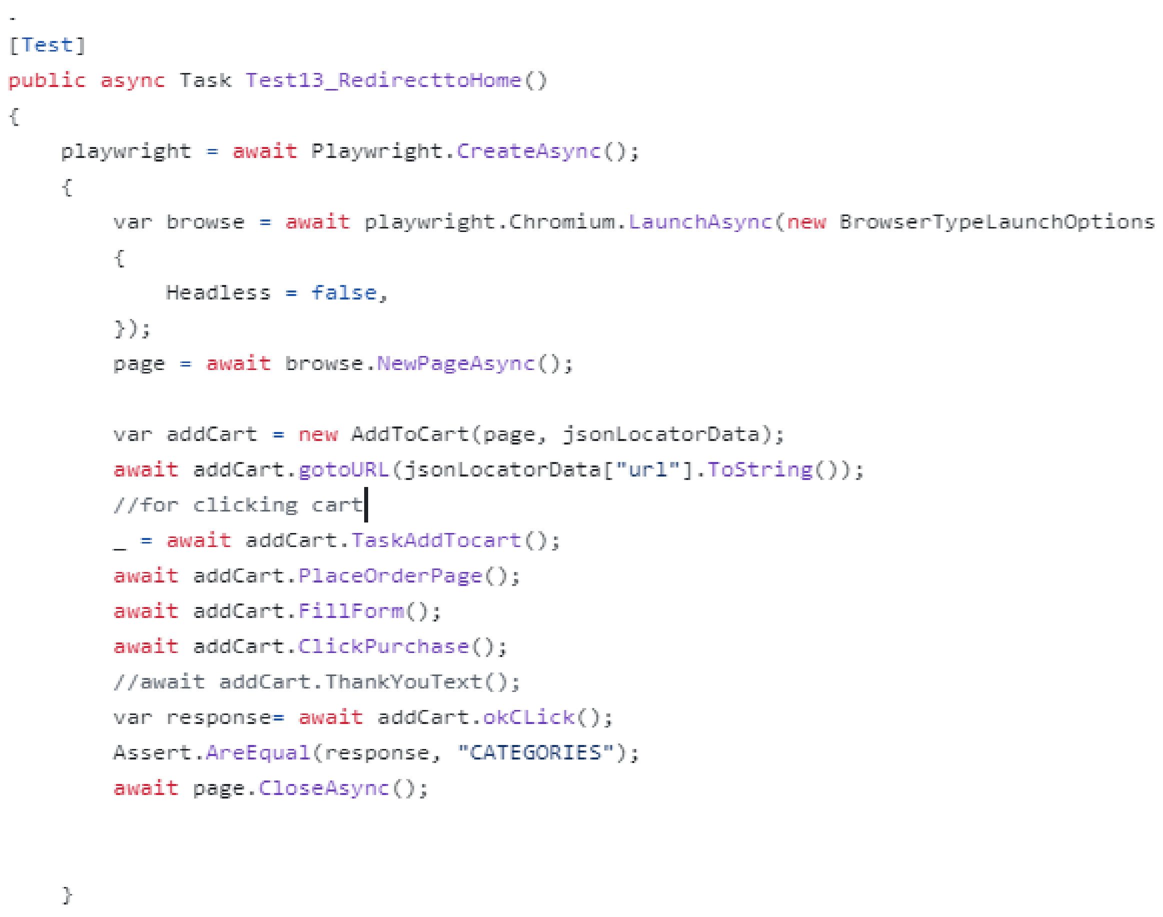
```
[Test]
public async Task Test11_EmptyBox()
{
    playwright = await Playwright.CreateAsync();
    {
        var browse = await playwright.Chromium.LaunchAsync(new BrowserTypeLaunchOptions
        {
            Headless = false,
        });
        page = await browse.NewPageAsync();

        var addCart = new AddToCart(page, jsonLocatorData);
        await addCart.gotoURL(jsonLocatorData["url"].ToString());
        //for clicking cart
        _ = await addCart.TaskAddTocart();
        await addCart.PlaceOrderPage();
        await addCart.ClickPurchase();
        Assert.AreEqual(addCart._dialogMessage, "Please fill out Name and Creditcard.");
        await page.CloseAsync();
```

- **Test12_FillBox():** It focuses to verify values in fill form by placing an order ensuring seamless user experience and reliability.



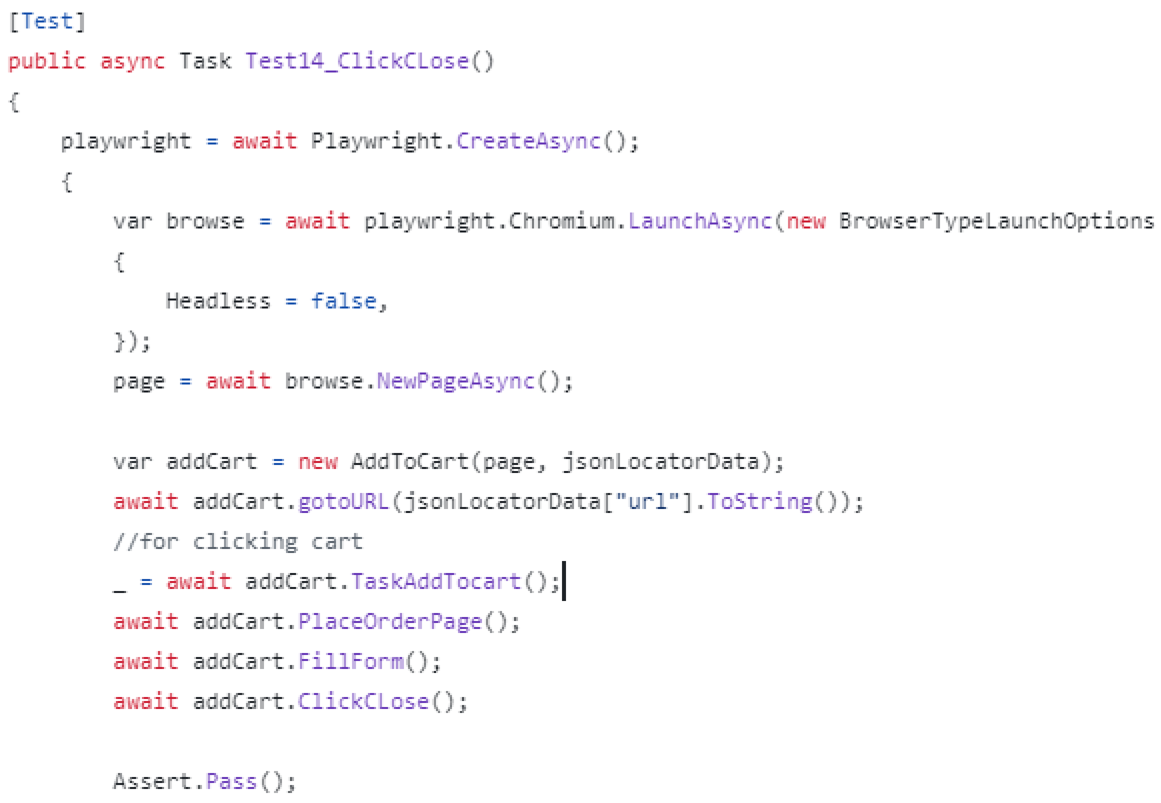
```
[Test]
public async Task Test12_FillBox()
{
    playwright = await Playwright.CreateAsync();
    {
        var browse = await playwright.Chromium.LaunchAsync(new BrowserTypeLaunchOptions
        {
            Headless = false,
        });
        page = await browse.NewPageAsync();

        var addCart = new AddToCart(page, jsonLocatorData);
        await addCart.gotoURL(jsonLocatorData["url"].ToString());
        //for clicking cart
        _ = await addCart.TaskAddTocart();
        await addCart.PlaceOrderPage();
        await addCart.FillForm();
        await addCart.ClickPurchase();
        var reponse=await addCart.ThankYouText();
        Assert.AreEqual(reponse, "Thank you for your purchase!");
        await page.CloseAsync();


    }
```

- **Test13_RedirecttoHome():** It focuses to verify redirection to home url to ensuring accuracy and reliability in the process.



```
-
[Test]
public async Task Test13_RedirecttoHome()
{
    playwright = await Playwright.CreateAsync();
    {
        var browse = await playwright.Chromium.LaunchAsync(new BrowserTypeLaunchOptions
        {
            Headless = false,
        });
        page = await browse.NewPageAsync();

        var addCart = new AddToCart(page, jsonLocatorData);
        await addCart.gotoURL(jsonLocatorData["url"].ToString());
        //for clicking cart
        _ = await addCart.TaskAddTocart();
        await addCart.PlaceOrderPage();
        await addCart.FillForm();
        await addCart.ClickPurchase();
        //await addCart.ThankYouText();
        var response= await addCart.okCLick();
        Assert.AreEqual(response, "CATEGORIES");
        await page.CloseAsync();


    }
```

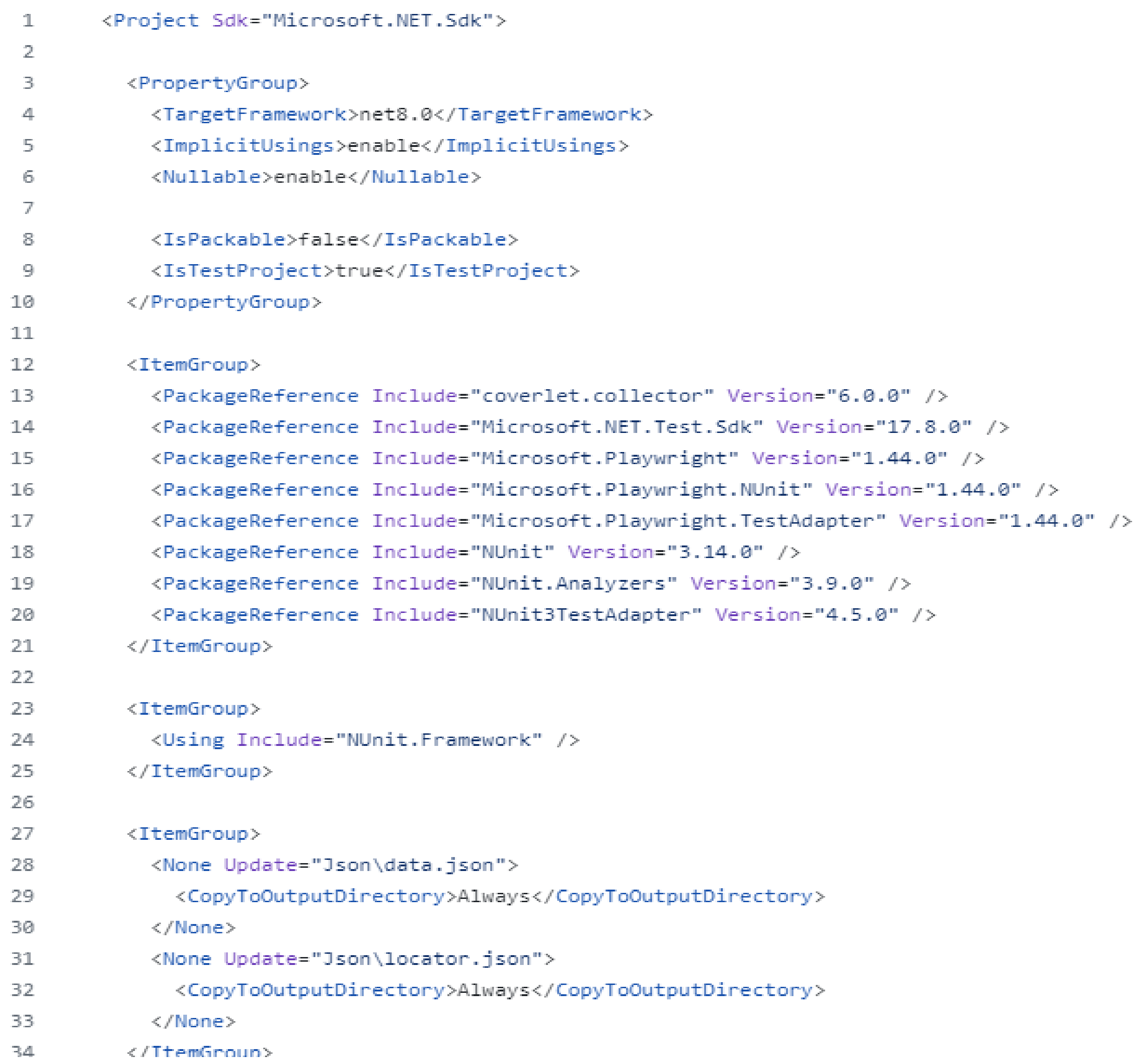
- **Test14_ClickCLose():** It focuses to verify ClickClose by ensuring seamless user experience and reliability.



```
[Test]
public async Task Test14_ClickCLose()
{
    playwright = await Playwright.CreateAsync();
    {
        var browse = await playwright.Chromium.LaunchAsync(new BrowserTypeLaunchOptions
        {
            Headless = false,
        });
        page = await browse.NewPageAsync();

        var addCart = new AddToCart(page, jsonLocatorData);
        await addCart.gotoURL(jsonLocatorData["url"].ToString());
        //for clicking cart
        _ = await addCart.TaskAddTocart();
        await addCart.PlaceOrderPage();
        await addCart.FillForm();
        await addCart.ClickCLose();

        Assert.Pass();
```

1.4 Packages

By installing necessary packages: Playwright, Node.js, and npm. Use `npm init` to create a project, followed by `npm install playwright` to add Playwright. Ensure compatibility by updating to the latest package versions and installing any additional dependencies.



```xml
<Project Sdk="Microsoft.NET.Sdk">

  <PropertyGroup>
    <TargetFramework>net8.0</TargetFramework>
    <ImplicitUsings>enable</ImplicitUsings>
    <Nullable>enable</Nullable>

    <IsPackable>false</IsPackable>
    <IsTestProject>true</IsTestProject>
  </PropertyGroup>

  <ItemGroup>
    <PackageReference Include="coverlet.collector" Version="6.0.0" />
    <PackageReference Include="Microsoft.NET.Test.Sdk" Version="17.8.0" />
    <PackageReference Include="Microsoft.Playwright" Version="1.44.0" />
    <PackageReference Include="Microsoft.Playwright.NUnit" Version="1.44.0" />
    <PackageReference Include="Microsoft.Playwright.TestAdapter" Version="1.44.0" />
    <PackageReference Include="NUnit" Version="3.14.0" />
    <PackageReference Include="NUnit.Analyzers" Version="3.9.0" />
    <PackageReference Include="NUnit3TestAdapter" Version="4.5.0" />
  </ItemGroup>

  <ItemGroup>
    <Using Include="NUnit.Framework" />
  </ItemGroup>

  <ItemGroup>
    <None Update="Json\data.json">
      <CopyToOutputDirectory>Always</CopyToOutputDirectory>
    </None>
    <None Update="Json\locator.json">
      <CopyToOutputDirectory>Always</CopyToOutputDirectory>
    </None>
  </ItemGroup>
```

5. Evaluation

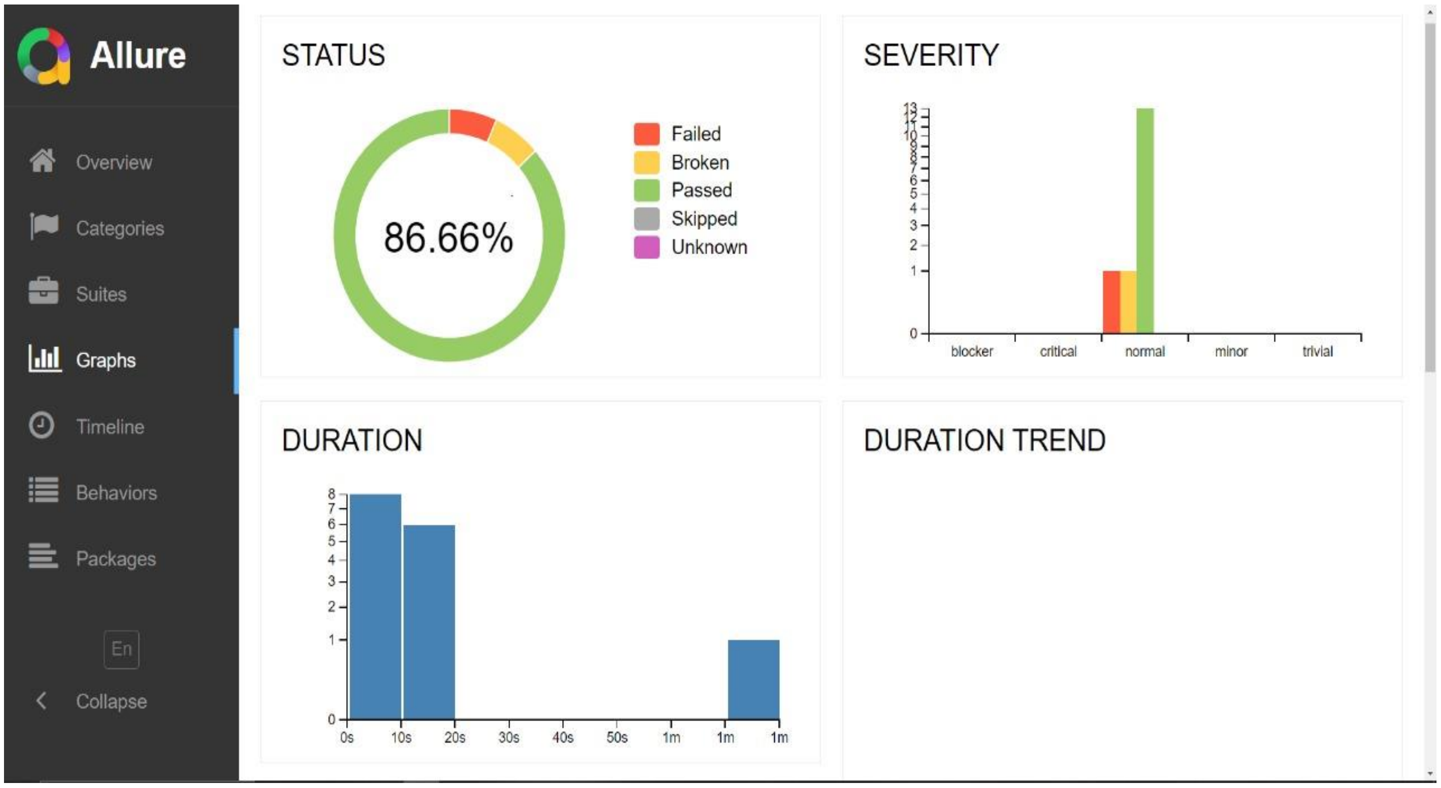
5.1 . Metrics

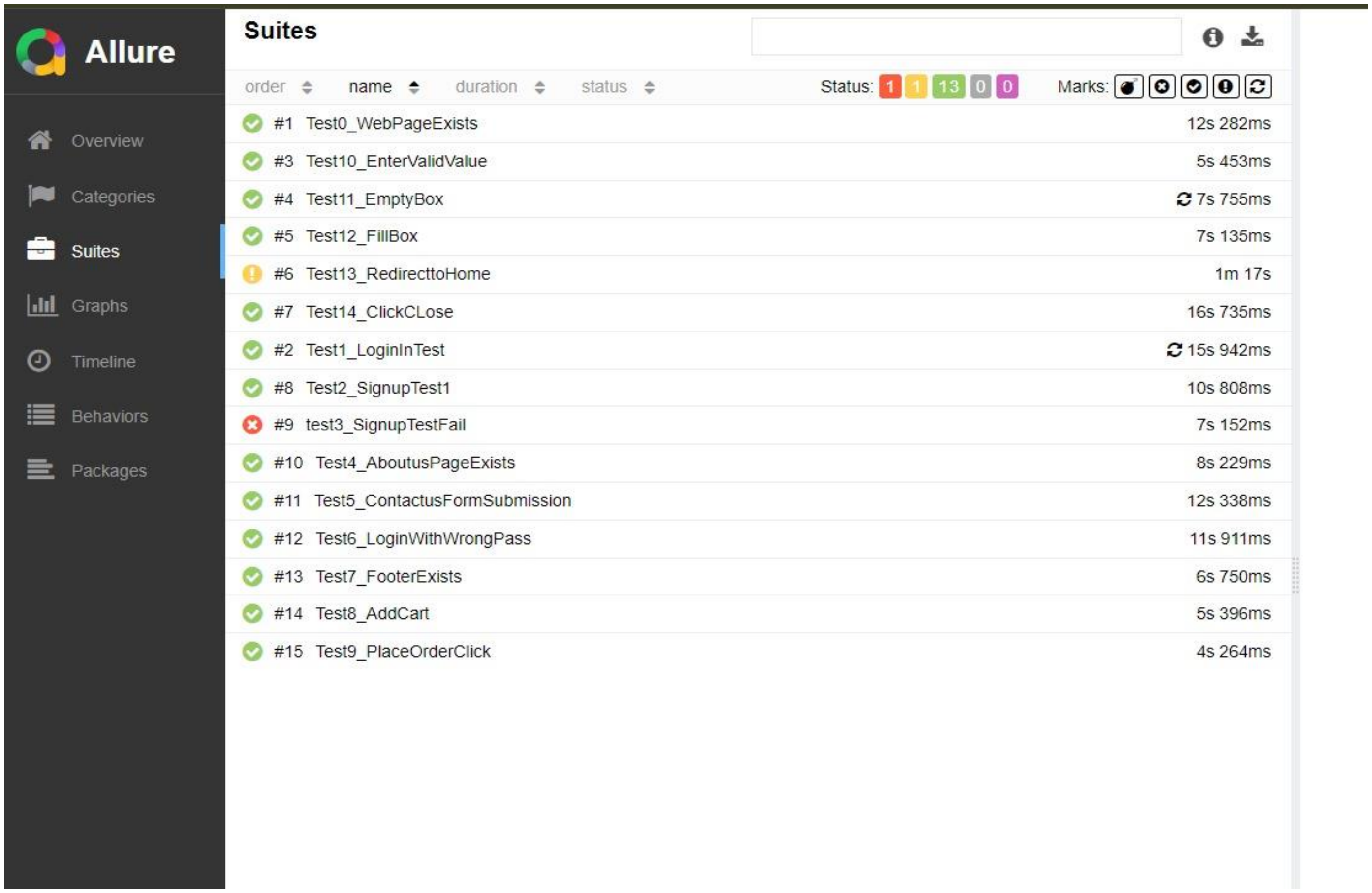
- Number of Test Cases: **15**
- Number of pass: **12**
- Number of Fail: **3**

5.2. Results

- **Test Cases Pass/Fail and Error Logs**:
  - Test Case Pass: **12**
  - Test Case Fail: **3**

ALLURE REPORT

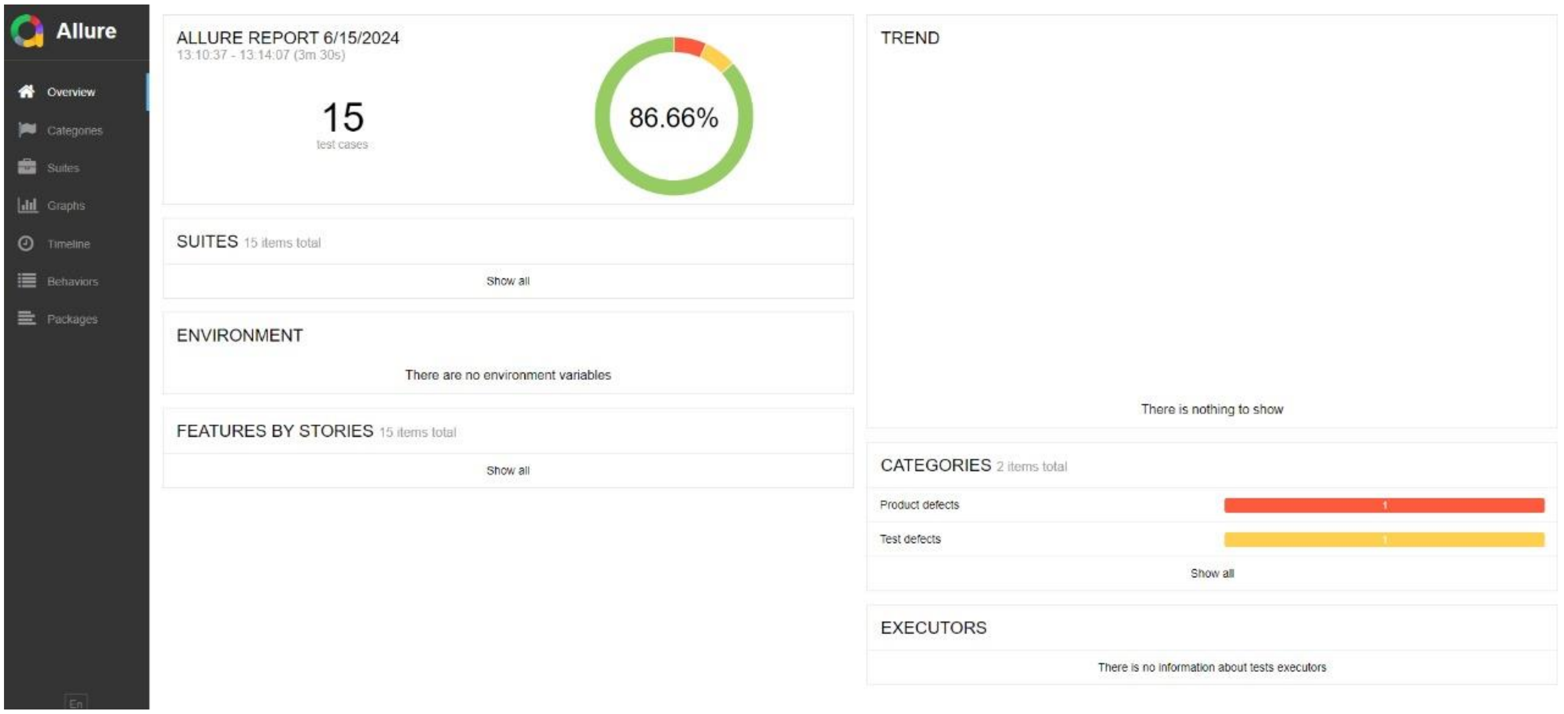




Suites

order name duration status — Status: 1 1 13 0 0 — Marks:

✓ #1 Test0_WebPageExists	12s 282ms
✓ #3 Test10_EnterValidValue	5s 453ms
✓ #4 Test11_EmptyBox	7s 755ms
✓ #5 Test12_FillBox	7s 135ms
! #6 Test13_RedirecttoHome	1m 17s
✓ #7 Test14_ClickCLose	16s 735ms
✓ #2 Test1_LoginInTest	15s 942ms
✓ #8 Test2_SignupTest1	10s 808ms
✗ #9 test3_SignupTestFail	7s 152ms
✓ #10 Test4_AboutusPageExists	8s 229ms
✓ #11 Test5_ContactusFormSubmission	12s 338ms
✓ #12 Test6_LoginWithWrongPass	11s 911ms
✓ #13 Test7_FooterExists	6s 750ms
✓ #14 Test8_AddCart	5s 396ms
✓ #15 Test9_PlaceOrderClick	4s 264ms



ALLURE REPORT 6/15/2024
13:10:37 - 13:14:07 (3m 30s)

15
test cases

86.66%

SUITES 15 items total
Show all

ENVIRONMENT
There are no environment variables

FEATURES BY STORIES 15 items total
Show all

TREND
There is nothing to show

CATEGORIES 2 items total

Product defects	1
Test defects	1

Show all

EXECUTORS
There is no information about tests executors

6. Discussion

Playwright Performance: It covers test execution speed, browser compatibility, and script stability.

Challenges: It include handling dynamic content, managing asynchronous operations, and ensuring reliable element selection. Collaborative problem-solving and optimization techniques are essential for enhancing testing efficiency and accuracy.

7. Conclusion

In conclusion, using Playwright has proven effective for automating and validating critical functionalities of the Demoblaze website. The developed test cases ensure the reliability and performance of key features, such as the Contact Us form, login process, and footer visibility. Despite challenges like dynamic content handling and asynchronous operations, the project's collaborative efforts have resulted in a robust testing framework. Continuous improvement and regular updates to the test scripts will maintain high standards of quality and user experience, ultimately contributing to the website's success and reliability.

8.Future Work

Looking ahead, future work for implementing test cases on https://www.demoblaze.com/ using Playwright includes expanding test coverage to additional website functionalities such as product navigation. Enhancing test scenarios for edge cases and integrating with CI/CD pipelines for automated regression testing will ensure ongoing quality assurance. Furthermore, exploring Playwright's capabilities for performance testing and cross-browser compatibility will be beneficial. Collaboration with developers to incorporate test-driven development principles and leveraging analytics for user behavior insights can further optimize the testing framework. Continuous refinement and adaptation to emerging technologies will support maintaining a robust and reliable website experience.