### **Automated Testing**

We have used **NUnit** (a testing framework) and **Selenium WebDriver** (tool for automating browsers) to make sure everything works correctly.

### **Prerequisites for Running the Tests**

Before you run the tests, you need a few tools installed:

1. **Visual Studio** (or any other C# editor)
2. **Google Chrome** (or another browser)
3. **Required Tools**:
   1. **NUnit** for running tests
   2. **Selenium WebDriver** for browser automation
   3. **WebDriverManager** to automatically download the correct driver for the browser

#### **Installing the Tools:**

To install the necessary tools, you can use **NuGet** package manager:

* **Install NUnit**:

Install-Package NUnit

* **Install Selenium**:

Install-Package Selenium.WebDriver

* **Install WebDriverManager**:

Install-Package WebDriverManager

Once everything is installed, you can run the tests using **Visual Studio's Test Explorer** or a tool called **NUnit Console Runner**.

### **Running the Tests**

To run the tests:

1. Open the project in **Visual Studio**.
2. Build the solution (this compiles everything).
3. Open the **Test Explorer** window.
4. Run all the tests or select the ones you want to test.
5. Check the results in the **Test Explorer** to see if any tests failed.

### **High-Priority Bugs and Their Failing Tests**

#### **Bug 1: Search Bar Not Working (BUG-001)**

**Why This Bug Is Important:**

The search bar helps users find books. If it’s broken, users can’t search for books, which is a major issue.

**Failing Test for This Bug:**

This test checks if searching for a book works:

[Test] // Test for search functionality  
public void TestSearchFunctionality()  
{  
 var searchBox = \_wait.Until(driver => \_driver.FindElement(By.CssSelector("input[placeholder='Search']")));  
 searchBox.SendKeys("The Golden Compass");  
 searchBox.SendKeys(Keys.Enter);  
  
 var results = \_wait.Until(driver => \_driver.FindElements(By.CssSelector(".book")));  
 Assert.That(results.Count > 0, "Search is not working - No results found");  
}

If this test fails, it means the search feature is broken, and users won’t be able to find books.

#### **Bug 2: Pagination Shows "Page -1" (BUG-002)**

**Why This Bug Is Important:**

Pagination (the ability to go to the next or previous page) is essential for navigating through a list of books. If it shows "Page -1", users can’t use it, which is a major problem.

**Failing Test for This Bug:**

This test checks if the pagination works properly:

[Test] // Test for pagination functionality  
public void TestPagination()  
{  
 var prevPageButton = \_wait.Until(driver => \_driver.FindElement(By.CssSelector("button[aria-label='Previous']")));  
 prevPageButton.Click();  
 prevPageButton.Click();  
 prevPageButton.Click();  
  
 var pageNumber = \_wait.Until(driver => \_driver.FindElement(By.CssSelector(".page-number")));  
 Assert.That(!pageNumber.Text.Contains("-1"), "Pagination is broken - Showing Page -1");  
}

If the page number shows "-1", this test will fail, showing that the pagination is broken.

### **Passing Test for a Working Feature**

#### **Feature: User Rating Display**

**Why This Feature Is Important:**

User ratings help people decide which books to read. It’s important that this feature works correctly so users can see ratings for each book.

**Passing Test for This Feature:**

This test checks if the user rating is visible:

[Test] // Test for user rating display  
public void TestUserRatingDisplayed()  
{  
 var ratingElement = \_wait.Until(driver => \_driver.FindElement(By.CssSelector("div.user-rating")));  
 Assert.That(ratingElement, Is.Not.Null, "User rating is missing");  
}

If the rating element is not found, this test will fail. It ensures that users can see ratings for books.

### **Writing Robust and Maintainable Tests**

To make sure our tests are reliable and easy to maintain, we follow a few key practices:

1. **Explicit Waits**: We use waits to make sure the page elements are ready before interacting with them. This prevents tests from failing due to timing issues.

Example:

\_wait.Until(driver => driver.FindElement(By.CssSelector("input[placeholder='Search']")));

1. **Reusable Code**: If we need to perform the same action multiple times, we write helper functions to avoid repeating code.
2. **Clear Error Messages**: When an assertion fails, we include a message that explains what went wrong, making it easier to understand the cause.
3. **Independent Tests**: Each test runs on its own without relying on data from other tests. This way, they won’t interfere with each other, and we get consistent results every time.
4. **Stable Data**: We make sure the tests don’t rely on real user data, which could change over time. Instead, we use mock data or set up the app in a way that ensures tests can run without depending on external changes.

### **Conclusion**

In this automated testing suite, we’ve created tests to check both **high-priority bugs** and **working features**:

1. **Failing Tests for Bugs**: We’ve created tests to check if the **search functionality** and **pagination** are working. These are important because they affect the basic usability of the app.
2. **Passing Test for Working Feature**: We’ve written a test to ensure the **user rating display** is working. This is important for users who want to see ratings before choosing a book.
3. **Best Practices**: The tests are written with best practices in mind, like using waits, keeping tests independent, and making sure the tests are clear and easy to maintain.