Exercise: Selenium WebDriver

1. Working with HTML Elements

1.1. Handling Form Input

In this exercise, you will **use Selenium to automate the process of filling out a form**. The steps involve **launching a web application**, **navigating to a form**, filling out **various input fields** such as **text boxes**, **radio buttons**, and **date pickers**, and then **submitting the form**. This exercise demonstrates how to locate and interact with different types of HTML elements commonly found in **Register User Scenarios**.
Note: We won't explain how to create a test project, install Selenium WebDriver or ChromeDriver, or how to write setup and teardown methods because you are already familiar with these steps.

- Open the application using the URL: **http://practice.bpbonline.com/.** It's up to you which method you use to navigate to the URL. Here are **two options:**
  - **`driver.Url Property`** - Navigate to a specific URL. Immediately loads the given URL in the current browser window.

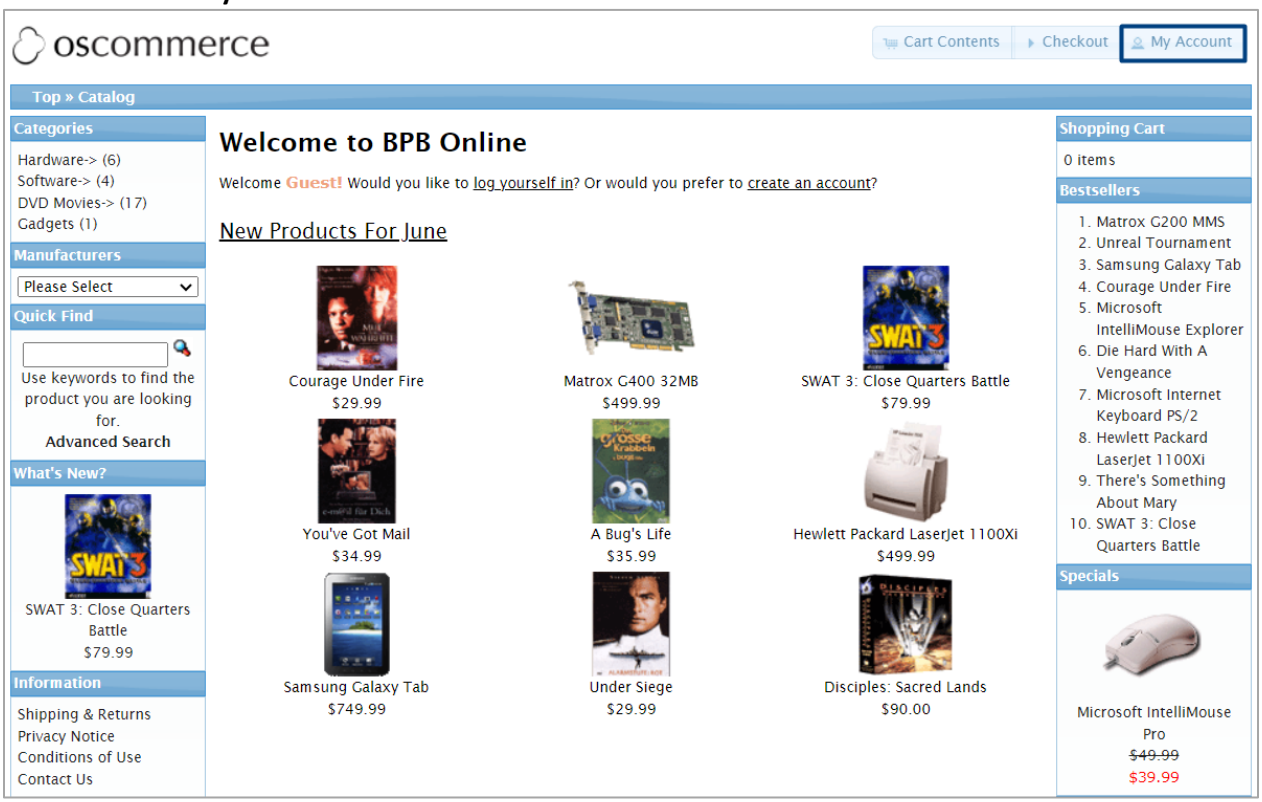
```
driver.Url = "http://practice.bpbonline.com/";
```

  - **`driver.Navigate().GoToUrl() Method`** A more explicit way to navigate to a URL, part of a broader set of navigation commands. Performs the same action as setting the Url property, but also allows for more complex navigation actions like going back, forward, or refreshing the page.

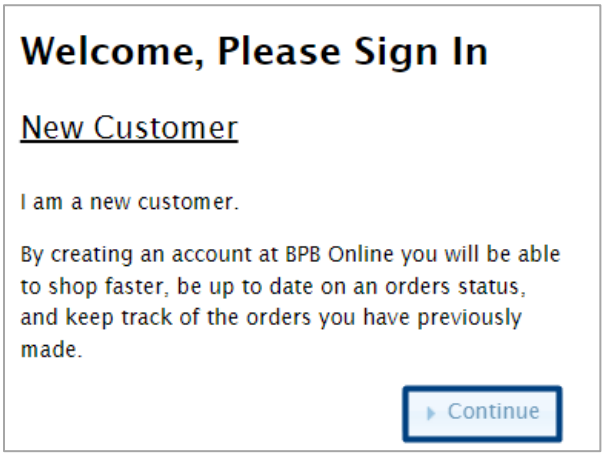
```
driver.Navigate().GoToUrl("http://practice.bpbonline.com/");
```

  - In this case, since we are performing a straightforward navigation to the URL, the more appropriate method is to use **`driver.Url`** for its simplicity and readability. However, you can choose to use **`driver.Navigate().GoToUrl()`** if you prefer a more explicit approach.

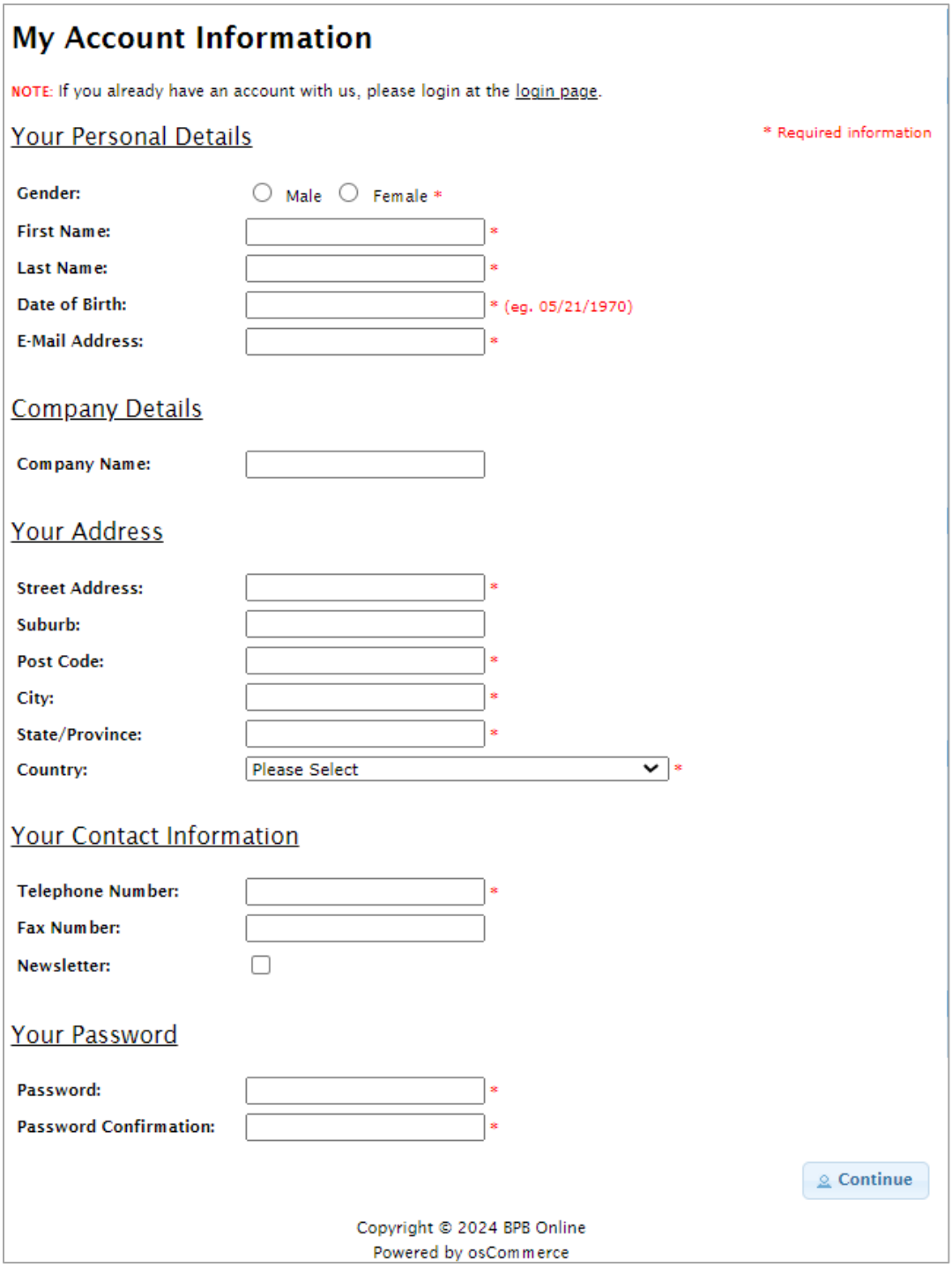
- Click on the **My Account link**



- In the **New Customer section**, click on the **Continue button**.



- This will take you to the **Account Creation page**.



- For the **My Account Information page**, you will need to fill in all the mandatory information for account creation.
- **Select Gender**.
- Enter **First Name**.
- Enter **Last Name.**
- Enter **Date of Birth.**
- Since the registration will work only once per email address**, generate a unique email address.**

```
// Generate a unique email address
Random rnd = new Random();
// Generate a random number between 1000 and 9999
int num = rnd.Next(1000, 9999);
String email = "fiona.apple" + num.ToString() + "@example.com";
```

- Enter **Email Address.**
- Enter **Company Name.**
- Enter **Street Address.**
- Enter **Suburb.**
- Enter **Postcode.**
- Enter **City.**
- Enter **State.**
- Select **Country from dropdown.**

When dealing with dropdown elements in web forms, Selenium provides a convenient class called **SelectElement** which is part of the **OpenQA.Selenium.Support.UI namespace**. This class contains methods specifically designed for interacting with dropdowns.

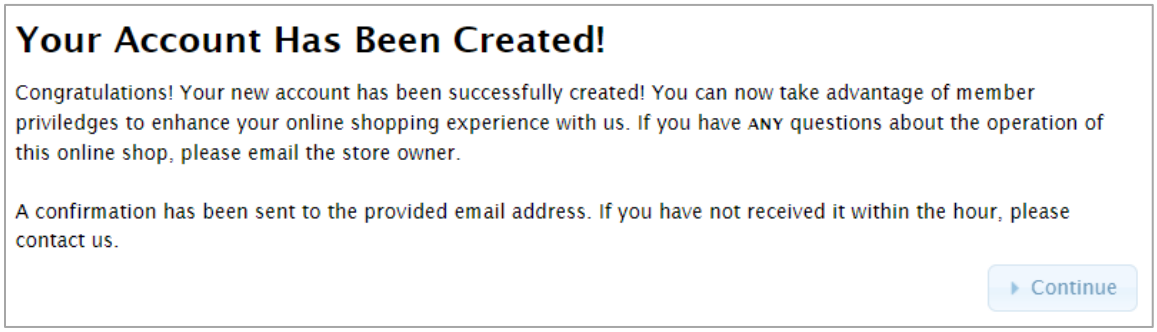
- Install Selenium Support NuGet package.
- Import the necessary namespace.
- Identify the dropdown element using a locator (e.g., by name).
- Create a SelectElement object by passing the identified dropdown element to its constructor.
- Select an option from the dropdown by its visible text.

```
IWebElement countryDropdown = driver.FindElement(By.Name("country"));
SelectElement selectCountry = new SelectElement(countryDropdown);
selectCountry.SelectByText("United Kingdom");
```

- You can also chain method calls to simplify the selection of an option from a dropdown list. Instead of breaking down the steps into multiple lines, chaining combines them into a single, streamlined statement.

```
new SelectElement(driver.FindElement(By.Name("country"))).SelectByText("United Kingdom");
```

- Enter **Telephone Number**.
- Opt for **Newsletter Subscription**.
- Enter **Password**.
- **Confirm Password**.
- **Submit the form**.
- Assert **Your Account Has Been Created Page**.



```
// Assert account creation success
Assert.IsTrue(driver.PageSource.Contains("Your Account Has Been Created!"), "Account creation failed.");
```

- Click on **Log Off link**.
- Click on **Continue button**.
- **Print success message to the console**.

```
Running selected tests in C:\Users\mddim\source\repos\SeleniumBasicExercise\HTML_Elements_01\bin\Debug\net6.0\HTML_Elements_01.dll
   NUnit3TestExecutor discovered 1 of 1 NUnit test cases using Current Discovery mode, Non-Explicit run
User Account Created with email: fiona.apple2274@example.com
```

1.2. Working with Web Tables

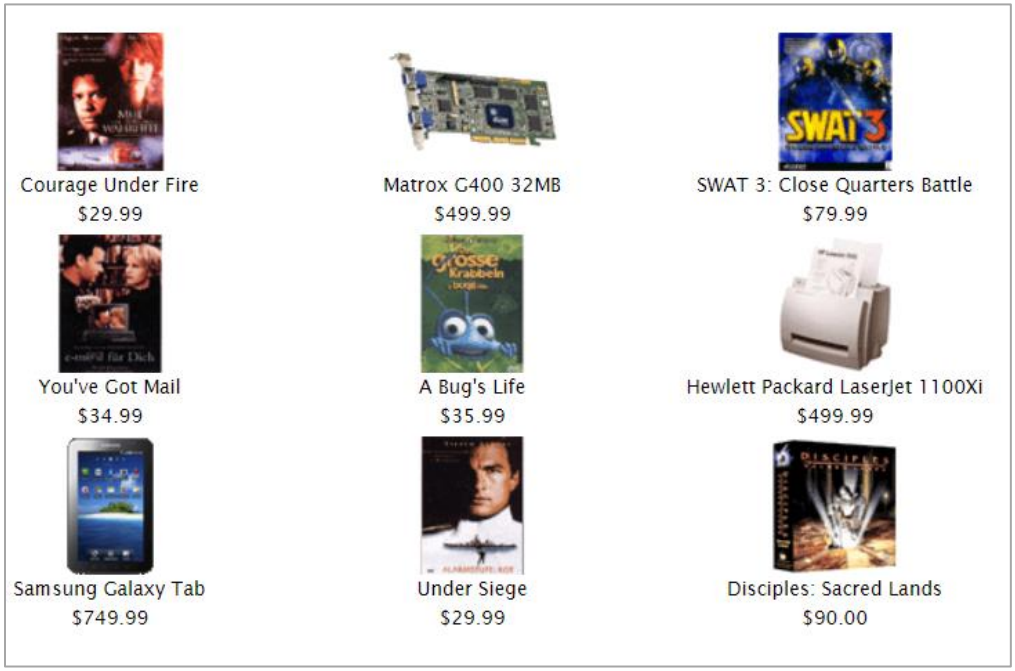
Many web pages need to show data in a clear, organized way. Usually, a web table element is used for this. A web table is an HTML element represented by the <table> tag. It contains other HTML elements, making it a container HTML element. Here is what a **<table> tag** can include:

```
<table>
    <tr>
        <th>Country</th>
        <th>Capital</th>
        <th>Population</th>
    </tr>
    <tr>
        <td>France</td>
        <td>Paris</td>
        <td>67,848,156</td>
    </tr>
    <tr>
        <td>Spain</td>
        <td>Madrid</td>
        <td>47,450,795</td>
    </tr>
    <tr>
        <td>Italy</td>
        <td>Rome</td>
        <td>60,317,116</td>
    </tr>
</table>
```

The table created using the preceding elements will be displayed in the browser as a web table:

Country	Capital	Population
France	Paris	67,848,156
Spain	Madrid	47,450,795
Italy	Rome	60,317,116

The scenario we will use to work with a web table is again from the **http://practice.bpbonline.com/** application. The home page of this website shows a list of products. In the background, this list is actually a web table.
On the home page products are listed as follows:



We can see **the <table> tag**, which includes a **<tbody>.** Inside, there are three <tr> tags representing table rows. Each row contains three <td> tags representing columns. The <td> tags include two anchor elements for the product image and product page, as well as text displaying the product's price.

```
<table border="0" width="100%" cellspacing="0" cellpadding="2">
  <tbody>
    <tr> == $0
      <td width="33%" align="center" valign="top">…</td>
      <td width="33%" align="center" valign="top">…</td>
      <td width="33%" align="center" valign="top">…</td>
    </tr>
    <tr>
      <td width="33%" align="center" valign="top">
        <a href="https://practice.bpbonline.com/product_info.php?products_id=7">
          <img src="images/dvd/youve_got_mail.gif" alt="You've Got Mail" title="You've Got Mail" width="100" height="80">
        </a>
        <br>
        <a href="https://practice.bpbonline.com/product_info.php?products_id=7">You've Got Mail</a>
        <br>
        "$34.99"
      </td>
      <td width="33%" align="center" valign="top">…</td>
      <td width="33%" align="center" valign="top">…</td>
    </tr>
    <tr>…</tr>
  </tbody>
</table>
```

By the end of this exercise, you will be able to:

- Traverse a web table and extract information.
- Save the extracted information to a CSV file.

- Open the application using the URL: **http://practice.bpbonline.com/.** Use either driver.Url or driver.Navigate().GoToUrl() to navigate to the URL.
- Identify the web table on the home page. Use XPath to locate the web table on the home page.

```
IWebElement productTable = driver.FindElement(By.XPath("//*[@id='bodyContent']/div/div[2]/table"));
```

- Find all rows in the web table:

```
ReadOnlyCollection<IWebElement> tableRows = productTable.FindElements(By.XPath("//tbody/tr"));
```

- Create a CSV file to save product information:

```
// Path to save the CSV file
string path = System.IO.Directory.GetCurrentDirectory() + "/productinformation.csv";
```

- This optional step ensures that if a CSV file with the same name already exists in the specified location, it will be deleted before creating a new one. This prevents appending data to an old file and ensures that each test run generates a fresh CSV file with only the current data.

```
if (File.Exists(path))
    File.Delete(path);
```

- Traverse through table rows to find the table columns:
  - Loop through each row and then each column within the row.
  - Extract the text from each cell, split the text to separate product name and cost, and format it.
  - Append the formatted text to the CSV file.

```
foreach (IWebElement trow in tableRows)
{
    ReadOnlyCollection<IWebElement> tableCols = trow.FindElements(By.XPath("td"));
    foreach (IWebElement tcol in tableCols)
    {
        // Extract product name and cost
        String data = tcol.Text;
        String[] productinfo = data.Split('\n');
        String printProductinfo = productinfo[0].Trim() + "," + productinfo[1].Trim() + "\n";

        // Write product information extracted to the file
        File.AppendAllText(path, printProductinfo);
    }
}
```

- Use assertions to check that the CSV file was created and is not empty.

```
Assert.That(File.Exists(path), Is.True, "CSV file was not created.");
Assert.That(new FileInfo(path).Length, Is.GreaterThan(0), "CSV file is empty.");
```

- Check how the file looks like in the bin → Debug → net{version} directory

Courage Under Fire	$29.99
Matrox G400 32MB	$499.99
SWAT 3: Close Quarters Battle	$79.99
You've Got Mail	$34.99
A Bug's Life	$35.99
Hewlett Packard LaserJet 1100Xi	$499.99
Samsung Galaxy Tab	$749.99
Under Siege	$29.99
Disciples: Sacred Lands	$90.00

Note: Keep in mind that the products might be slightly different, as they change each month.

1.3. Drop-down Practice

In this exercise, you will use Selenium to automate the interaction with dropdown elements. A dropdown web element is represented by the <select> tag in HTML and contains <option> tags, each representing a selectable item in the dropdown list.

To work with dropdown elements in Selenium, you use the SelectElement class from the OpenQA.Selenium.Support.UI namespace. This class provides several methods to select options by text, index, or value.
Here is a basic example of a dropdown in HTML:

```
<select name="country">
<option value="1">Italy</option>
<option value="2">Greece</option>
<option value="3">Spain</option>
<option value="4">Germany</option>
<option value="5">UK</option>
</select>
```

To work with dropdown elements in Selenium, you use the **`SelectElement`** class from the `OpenQA.Selenium.Support.UI` namespace. This class provides several methods to select options by text, index, or value.
By the end of this exercise, you will be able to:

- Traverse a dropdown and extract information.
- Save the extracted information to a text file.

- Open the application using the URL: **http://practice.bpbonline.com/**. Use either `driver.Url` or `driver.Navigate().GoToUrl()` to navigate to the URL.
- Determine the path to save the text file and delete any existing file with the same name.

```
string path = Directory.GetCurrentDirectory() + "/manufacturer.txt";
```

```
if (File.Exists(path))
{
    File.Delete(path);
}
```

- Identify the dropdown element using name property.

```
SelectElement manufDropdown = new SelectElement(driver.FindElement(By.Name("manufacturers_id")));
```

- Retrieve all options from the dropdown and store them in a list.

```
IList<IWebElement> allManufacturers = manufDropdown.Options;
```

- Create a string list to fill in all manufacturers and remove the "Please Select" option.

```
foreach (IWebElement manufName in allManufacturers)
{
    manufNames.Add(manufName.Text);
}
// Remove the "Please Select" option from the list
manufNames.RemoveAt(0);
```

- Iterate through the manufacturers to fetch the product information related to each:
    - Loop through each manufacturer and select it from the dropdown.
    - Check if there are no products available for the selected manufacturer.
    - If products are available, fetch all rows from the product table and print the information to the text file.

```
foreach (string mname in manufNames)
{
    manufDropdown.SelectByText(mname);
    manufDropdown = new SelectElement(driver.FindElement(By.XPath("//select[@name='manufacturers_id']")));

    if (driver.PageSource.Contains("There are no products available in this category."))
    {
        File.AppendAllText(path, $"The manufacturer {mname} has no products\n");
    }
    else
    {
        // Create the table element
        IWebElement productTable = driver.FindElement(By.ClassName("productListingData"));

        // Fetch all table rows
        File.AppendAllText(path, $"\n\nThe manufacturer {mname} products are listed--\n");
        ReadOnlyCollection<IWebElement> rows = productTable.FindElements(By.XPath("//tbody/tr"));

        // Print the product information in the file
        foreach (IWebElement row in rows)
        {
            File.AppendAllText(path, row.Text + "\n");
        }
    }
}
```

- Close the browser and end the session.
- Check how the file looks like in the bin → Debug → net{version} directory.

```
The manufacturer Canon has no products


The manufacturer Fox products are listed--
Product Name+ Price Buy Now
Courage Under Fire $38.99  $29.99
Buy Now
Die Hard With A Vengeance $39.99
Buy Now
Speed $39.99
Buy Now
Speed 2: Cruise Control $42.00
Buy Now
There's Something About Mary $49.99
Buy Now


The manufacturer GT Interactive products are listed--
Product Name+ Price Buy Now
Disciples: Sacred Lands $90.00
Buy Now
The Wheel Of Time $99.99
Buy Now
Unreal Tournament $89.99
Buy Now
```

Note: Keep in mind that the products might be slightly different, as they change each month.

2. Data-Driven Tests

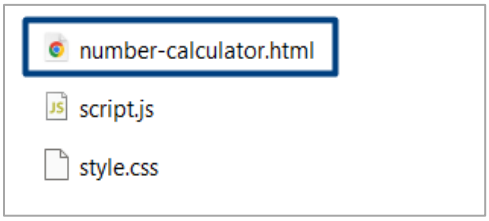
In this exercise, we will write automated Selenium tests for the "Number Calculator" web application. We will implement data-driven testing to make our tests more efficient and maintainable.
You can find the "Number Calculator" in the resources provided for this exercise.
By the end of this exercise, you will be able to:

Implement data-driven testing to reuse test logic for multiple sets of data, enhancing test clarity and efficiency.

How to Run the Number Calculator app (With and Without a Server)

Option 1: Run Without a Server (Simple Method)

- Navigate to the folder containing the Number Calculator app and double click on number-calculator.html



Option 2: Run with a Local Server

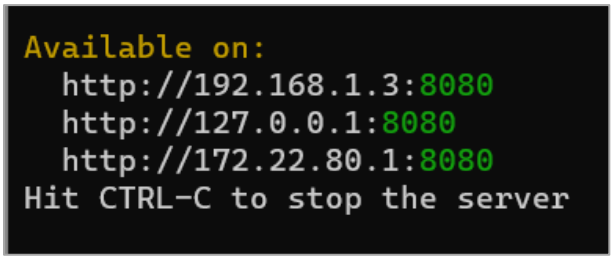
- Open Command Prompt (CMD) or PowerShell
- Go to Your Project Folder

```
cd path\to\your\project
```

- Start the Local Server with Node.js

```
npx http-server
```

- This will start a local web server and show something like:



- Open **Google Chrome** (or any browser)
- Go to:

```
http://localhost:8080
```

- Click on number-calculator.html to run your project.

Next

- Set up your project in VS.
- Install the necessary Selenium packages via NuGet.
- Create fields for the driver and web elements.

```
[TestFixture]
0 references
public class TestCalculator
{
    IWebDriver driver;
    IWebElement textBoxFirstNum;
    IWebElement textBoxSecondNum;
    IWebElement dropDownOperation;
    IWebElement calcBtn;
    IWebElement resetBtn;
    IWebElement divResult;
```

- Initialize the ChromeDriver and navigate to the application URL.
- Locate the necessary web elements on the page.

```
[OneTimeSetUp]
0 references
public void SetUp()
{
    driver = new ChromeDriver();
    driver.Manage().Timeouts().ImplicitWait = TimeSpan.FromSeconds(10);
    driver.Url = "http://softuni-qa-loadbalancer-2137572849.eu-north-1.elb.amazonaws.com/number-calculator/";

    textBoxFirstNum = driver.FindElement(By.Id("number1"));
    dropDownOperation = driver.FindElement(By.Id("operation"));
    textBoxSecondNum = driver.FindElement(By.Id("number2"));
    calcBtn = driver.FindElement(By.Id("calcButton"));
    resetBtn = driver.FindElement(By.Id("resetButton"));
    divResult = driver.FindElement(By.Id("result"));
}
```

- Define a method that accepts strings for the first number, operator, second number, and expected result.
- Implement the steps to interact with the web elements.

```
1 reference | 5/5 passing
public void PerformCalculation(string firstNumber, string operation,
                               string secondNumber, string expectedResult)
{
    // Click the [Reset] button
    resetBtn.Click();

    // Send values to the corresponding fields if they are not empty
    if (!string.IsNullOrEmpty(firstNumber))
    {
        textBoxFirstNum.SendKeys(firstNumber);
    }

    if (!string.IsNullOrEmpty(secondNumber))
    {
        textBoxSecondNum.SendKeys(secondNumber);
    }

    if (!string.IsNullOrEmpty(operation))
    {
        new SelectElement(dropDownOperation).SelectByText(operation);
    }

    // Click the [Calculate] button
    calcBtn.Click();

    // Assert the expected and actual result text are equal
    Assert.That(divResult.Text, Is.EqualTo(expectedResult));
```

- Use the `[TestCase]` attribute to write various test cases with different inputs.

```
[Test]
[TestCase("5", "+ (sum)", "10", "Result: 15")]
[TestCase("3.5", "- (subtract)", "1.2", "Result: 2.3")]
[TestCase("2e2", "* (multiply)", "1.5", "Result: 300")]
[TestCase("5", "/ (divide)", "0", "Result: Infinity")]
[TestCase("invalid", "+ (sum)", "10", "Result: invalid input")]
0 references
public void TestNumberCalculator(string firstNumber, string operation,
                                 string secondNumber, string expectedResult)
{
    PerformCalculation(firstNumber, operation, secondNumber, expectedResult);
}
```

- Create the TearDown method.

```
[OneTimeTearDown]
0 references
public void TearDown()
{
    driver.Quit();
}
```

In data-driven testing the test data is separated from test logic. Note that each test case is executed as a separate test, but the code of the test method is not repeated, it is reused. Each test case runs independently with its own data set, ensuring that the logic is applied consistently across different scenarios.

- Write some more test cases.

At the end, your test cases may look like this 😊

TestCalculatorWebApp (36)	4.8 sec
TestCalculatorWebApp("","-","3","Result: invalid input")	100 ms
TestCalculatorWebApp("","*","3","Result: invalid input")	100 ms
TestCalculatorWebApp("","/","3","Result: invalid input")	113 ms
TestCalculatorWebApp("","+","3","Result: invalid input")	103 ms
TestCalculatorWebApp("1","+","Infinity","Result: Infinity")	135 ms
TestCalculatorWebApp("1.5e53","*","150","Result: 2.25e+55")	131 ms
TestCalculatorWebApp("1.5e53","/","150","Result: 1e+51")	129 ms
TestCalculatorWebApp("12","/","3","Result: 4")	151 ms
TestCalculatorWebApp("12.5","/","4","Result: 3.125")	118 ms
TestCalculatorWebApp("2","-","Infinity","Result: -Infinity")	138 ms
TestCalculatorWebApp("3","!!!!!!","7","Result: invalid operation")	134 ms
TestCalculatorWebApp("3","","7","Result: invalid operation")	103 ms
TestCalculatorWebApp("3","*","Infinity","Result: Infinity")	133 ms
TestCalculatorWebApp("3","@","7","Result: invalid operation")	129 ms
TestCalculatorWebApp("3.14","-","12.763","Result: -9.623")	135 ms
TestCalculatorWebApp("3.14","*","-7.534","Result: -23.65676")	132 ms
TestCalculatorWebApp("3.14","*","asfd","Result: invalid input")	127 ms
TestCalculatorWebApp("4","/","Infinity","Result: 0")	136 ms