Selenium-WebDriver-POM: Lab

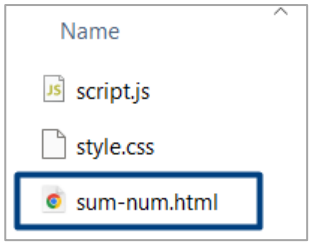
1. "Sum Two Numbers"

In this exercise, we will use the Sum-Num application. Which, what a surprise 😉, sums two numbers. You can find the Sum-Num App in the resources provided for this exercise.

How to Run the Sum-Num App (With and Without a Server)

Option 1: Run Without a Server (Simple Method)

- Navigate to the folder containing the Sum-Num app and double click on number-calculator.html



Option 2: Run with a Local Server

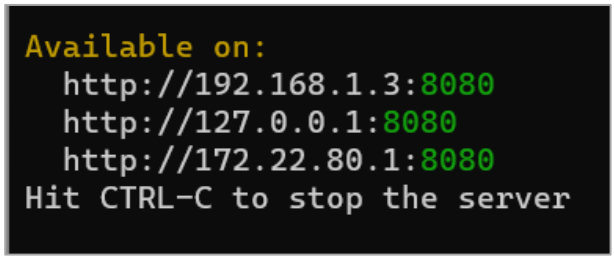
- Open Command Prompt (CMD) or PowerShell
- Go to Your Project Folder

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

- Start the Local Server with Node.js

```
npx http-server
```

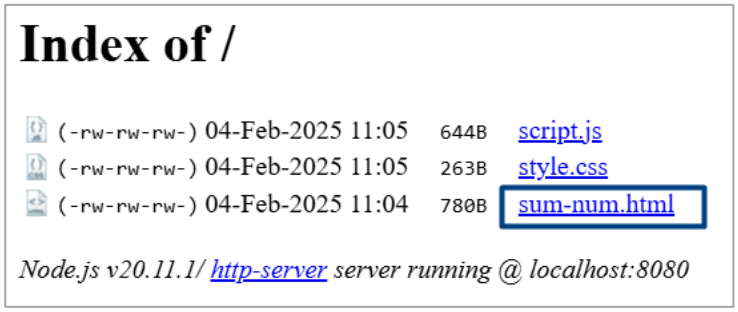
- This will start a local web server and show something like:



- CTRL + click on some of the links or open **Google Chrome** (or any browser) and go to:

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

- Click on sum-num.html to run your project.



1.1. Define the SumNumberPage Class:

The SumNumberPage class encapsulates all the operations that can be performed on the Sum Numbers page.

It contains properties for each UI element and methods to interact with these elements.

Fields:

- **const string PageUrl:**
  This field holds the URL of the web page that the SumNumberPage class interacts with. It is a constant string that is used to navigate to the Sum Numbers page.
- **public IWebElement FieldNum1 => driver.FindElement(By.CssSelector("input#number1"));**
  This field locates and represents the first number input field on the Sum Numbers page. It uses a CSS selector to find the element with the ID number1.
- **public IWebElement FieldNum2 => driver.FindElement(By.CssSelector("input#number2"));**
  This field locates and represents the second number input field on the Sum Numbers page. It uses a CSS selector to find the element with the ID number2.
- **public IWebElement ButtonCalc => driver.FindElement(By.CssSelector("button#calcButton"));**
  This field locates and represents the Calculate button on the Sum Numbers page. It uses a CSS selector to find the button element with the ID calcButton.
- **public IWebElement ButtonReset => driver.FindElement(By.CssSelector("button#resetButton"));**
  This field locates and represents the Reset button on the Sum Numbers page. It uses a CSS selector to find the button element with the ID resetButton.
- **public IWebElement ElementResult => driver.FindElement(By.CssSelector("#result"));**
  This field locates and represents the result element on the Sum Numbers page. It uses a CSS selector to find the element with the ID result, which displays the result of the sum of the two input numbers.

Methods:

- **OpenPage Method**: Navigates to the Sum Numbers page.
- **AddNumbers Method**: Sends values to the input fields and clicks the Calculate button.
- **ResetForm Method**: Resets the form to its initial state.
- **IsFormEmpty Method**: Checks if the form fields and result are empty.

1.2. Test Class:

Create a test class to define the setup, teardown, and test methods that validate the functionality of the page. Write test methods to perform actions on the page and assert the expected outcomes.

- The **SumNumberPageTests** class contains the setup and teardown methods to initialize and quit the ChromeDriver.

Three test methods validate different functionalities of the Sum Numbers page:

- **Test_AddTwoNumbers_ValidInput**: Tests the addition of two valid numbers.
- **Test_AddTwoNumbers_InvalidInput**: Tests the addition of a number and an invalid input.
- **Test_FormReset**: Tests the reset functionality of the form.

2. "Student Registry" App

You can find the Sum-Num App in the resources provided for this exercise.

How to Run the Student Registry App

- Open Command Prompt (CMD) or PowerShell

- Go to Your Project Folder

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

- Then, install all dependencies using:

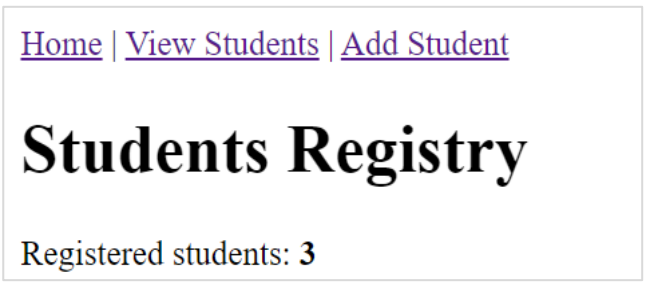
```
npm install
```

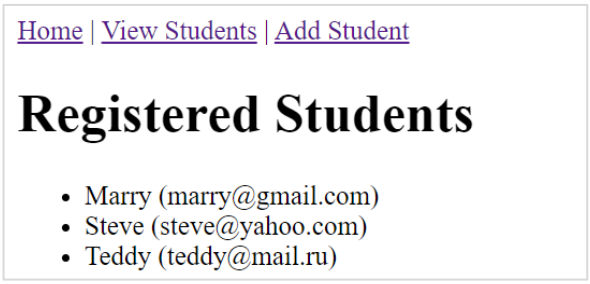
- Run the app using the command defined in the package.json:

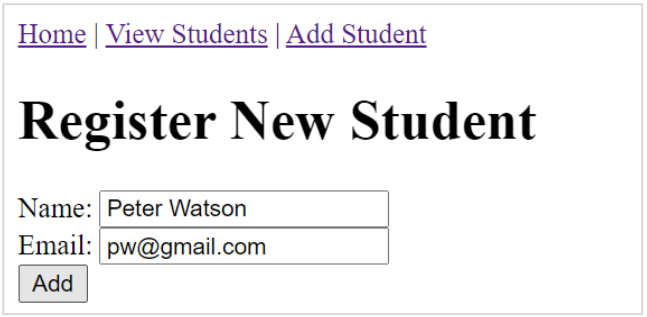
```
npm start
```

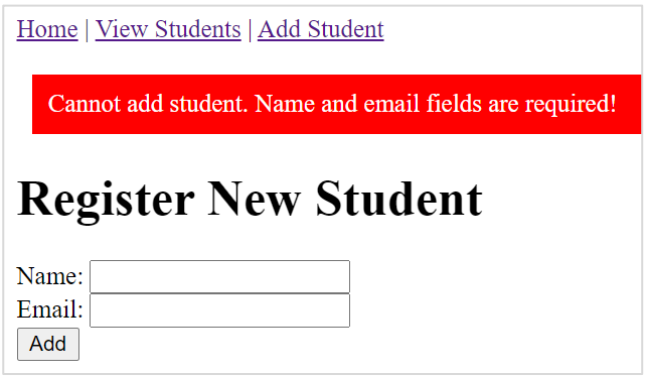
- Open **Google Chrome** (or any browser)
- Go to:

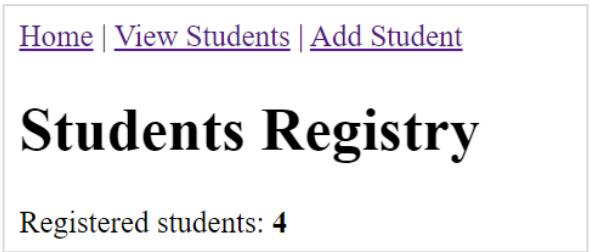
```
http://localhost:8080
```







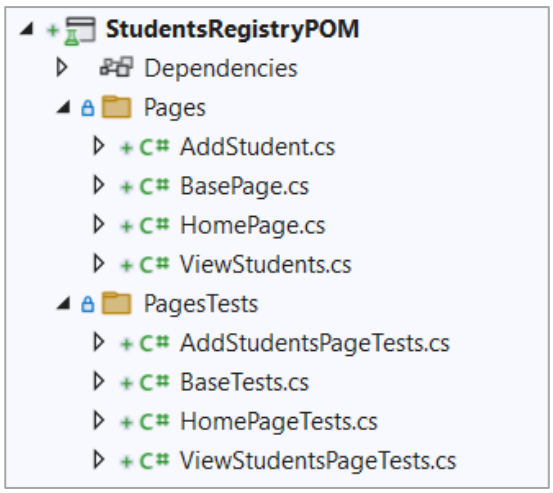




Write automated Selenium UI tests for the following app, which holds a registry of students:

2.1. Test Project Structure

Create a "**NUnit Test Project**" called "**StudentsRegistryPOM**". As we will use the **Page Object Model**, we will have **Page Object classes** and **Test classes**. **Separate them to folders**. You should have the following **project:**



2.2. Create Page Object Classes

Create **Page Object classes for each page** of the **Students Registry App**. You should have a **base PO class** with **common properties and methods** for all PO child classes. The other **PO classes** – the **HomePage**, the **ViewStudentsPage** and the **AddStudentPage classes** should **inherit** the **BasePage** base class.

Create the BasePage Page Object Class

The **BasePage class** is a **base class** for all **Page Object classes**. It should contain:

- Field: **IWebDriver driver (protected and readonly)**
- Constructor: **BasePage(IWebDriver driver)**
- Virtual property: **PageUrl**
- Properties: **LinkHomePage**, **LinkViewStudentsPage**, **LinkAddStudentsPage**, **ElementTextHeading**
- Method: **Open() => driver.Url = this.PageUrl;**
- Method: **IsOpen() => driver.Url == this.PageUrl;**
- Methods: **GetPageTitle()**, **GetPageHeading()**

The field of type **IWebDriver** should be **protected**, so that **only child classes can access it** but it should not be changed directly, so it is also **readonly**. The **driver** is accepted through the **constructor**. Also, it is a good idea to set **an implicit wait** for the driver. Additionally, we should have a **virtual property PageURL**, which will be **different for each child class**. Write the **field**, the **constructor** and the **property**:

```
public class BasePage
{
    protected readonly IWebDriver driver;

    3 references
    public BasePage(IWebDriver driver)
    {
        this.driver = driver;
        driver.Manage().Timeouts().ImplicitWait = TimeSpan.FromSeconds(3);
    }

    5 references
    public virtual string PageUrl { get; }
```

Other **properties** of the PO class keep **each of the links in the main menu**, which are **shared between all pages**. Only the **last property** is different – it **locates the page heading of the current page**. All elements are located in the usual way with different locator strategies. The properties look like this:

```
3 references | 3/3 passing
public IWebElement LinkHomePage =>
    driver.FindElement(By.LinkText("Home"));

3 references | 3/3 passing
public IWebElement LinkViewStudentsPage =>
   driver.FindElement(By.LinkText("View Students"));

3 references | 3/3 passing
public IWebElement LinkAddStudentsPage =>
   driver.FindElement(By.LinkText("Add Student"));

1 reference
public IWebElement PageHeading =>
    driver.FindElement(By.CssSelector("body > h1"));
```

Next, we have the **Open()** method, which is responsible for **opening a page on a given page URL**. The method is pretty simple:

```
    public void Open()
    {
        driver.Navigate().GoToUrl(this.PageUrl);
    }
```

We also have the **IsOpen() boolean** method, which **checks whether the current URL of the driver is the same as the page URL of our page**. If they are the same, then the user is on the right page and it is open. The method looks like this:

```
    public bool IsOpen()
    {
        return driver.Url == this.PageUrl;
    }
```

Our final **two methods** for this class **get the title and the heading text of the current page**. The **GetPageTitle()** method is the following:

```
    public string GetPageTitle()
    {
        return driver.Title;
    }
```

The **GetPageHeadingText() returns the text** from the **ElementPageHeading property**:

```
    public string GetPageHeadingText()
    {
        return ElementPageHeading.Text;
    }

}
```

Create the "HomePage" Page Object Class

The **HomePage class inherits** the **BasePage class** (can use its properties and methods) and should contain:

- Constructor: **HomePage(IWebDriver driver)**
- Properties: inherited + **PageUrl** (assigned correctly) + **ElementStudentsCount**
- Methods: inherited + **GetStudentsCount()**

Inherit the **BasePage** class and **use its constructor by providing it with the driver**:

```
public class HomePage : BasePage
{
    5 references | 4/4 passing
    public HomePage(IWebDriver driver) : base(driver)
    {
    }
```

Next, **override the PageURL property** with the **URL** of the **Home page** of the app (http://localhost:8080/):

```
public override string PageUrl =>
        "http://localhost:8080/";
```

Then, **create** the **ElementStudentsCount property**, which **locates the count of registered students** on the page:

The property looks like this:

```
    public IWebElement ElementStudentsCount =>
        driver.FindElement(By.CssSelector("body > p > b"));
```

Create the "ViewStudentsPage" Page Object Class

The **ViewStudentsPage PO class** also **inherits** the **BasePage class**. In addition, it has:

- Properties: inherited + **PageUrl** (assigned correctly) + **ListItemsStudents** (of type **ReadOnlyCollection<IWebElement>**)
- Methods: inherited + **GetStudentsList()** (returns **string[]**)

The whole class looks like this:

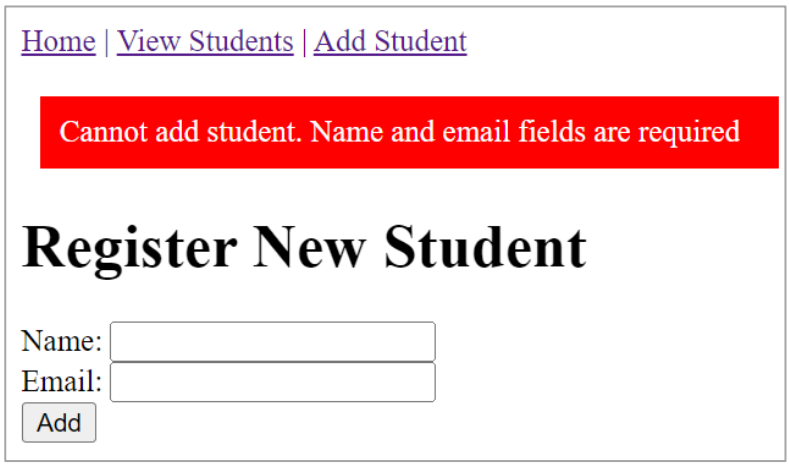
```
7 references
public class ViewStudents : BasePage
{
    6 references | 5/5 passing
    public ViewStudents(IWebDriver driver) : base(driver)
    {
    }
    3 references
    public override string PageUrl =>
             "http://localhost:8080/students";

    1 reference
    public ReadOnlyCollection<IWebElement> ListItemsStudents =>
        driver.FindElements(By.CssSelector("Body > ul > li"));

    2 references | 2/2 passing
    public string[] GetRegisterStudents()
    {
        var elementsStudents = this.ListItemsStudents.Select(s => s.Text).ToArray();
        return elementsStudents;
    }
}
}
```

Create the "AddStudentPage" Page Object Class

The **AddStudentPage** class **inherits** the **BasePage** class and has:

- Properties: inherited + **PageUrl** (assigned correctly) + **ElementErrorMsg**
- Form field properties: **FieldStudentName**, **FieldStudentEmail**, **ButtonAdd**
- Methods: **AddStudent(string name, string email)**, **GetErrorMsg()**

Write the **constructor** and the **properties** by yourself. Note that the **ElementErrorMsg** property **locates an error message**, which appears only when the **Register New Student form is filled with invalid data**:



The **AddStudent(string name, string email) fills the registration form**, using the **field properties**. It looks like this:

```
public void AddStudent(string name, string email)
{
    this.FieldName.SendKeys(name);
    this.FieldEmail.SendKeys(email);
    this.ButtonSubmit.Click();
}
```

The **GetErrorMsg()** method **returns the error text** from the **ElementErrorMsg property**. Write the method on your own.

2.3. Write Selenium POM Tests

Create the Tests Base Class

The **BaseTest class** is a **base class for all other test classes**. It has the **OneTimeSetUp()** and **OneTimeTearDown()** methods, which **initialize** and **quit** the **ChromeDriver()**. The class looks like this:

```
public class BaseTest
{
    protected IWebDriver driver;

    [OneTimeSetUp]
    0 references
    public void OneTimeSetUp()
    {
        driver = new ChromeDriver();
    }
```

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

Create "Home Page" Tests

On our **Home page**, we should **test the page content** and the **page links**. Create the **TestHomePage tests class**, which should **inherit** the **BaseTest** class to **access the driver**:

```
public class TestHomePage : BaseTest
```

The **Test_HomePage_Content()** test **will open the Home page** and **assert that it has a correct title**, **heading** and **students count**. To write the test method, follow these steps:

- **Instantiate** the **Home page with driver** and **open the page**:

```
[Test]
0 references
public void Test_HomePage_Content()
{
    var page = new HomePage(driver);
    page.Open();
```

- **Assert** the page **title** is correct (window title):

```
    Assert.AreEqual("MVC Example", page.GetPageTitle());
```

- **Assert** the page **heading** is correct (the top heading at the start of the page):

```
    Assert.AreEqual("Students Registry", page.GetPageHeadingText());
```

- **Invoke** the **GetStudentsCount()** method- it should **not throw any errors**:

```
    page.GetStudentsCount();
}
```

The **Test_HomePage_Links()** test will check whether the **Home page links open the correct pages**. To write the test method, follow these steps:

- **Instantiate** the **HomePage** class with **driver**:

```
[Test]
0 references
public void Test_HomePage_Links()
{
    var homePage = new HomePage(driver);
```

- **Go** to the **Home page**, **click on the Home page link** and **assert the Home page is open**:

```
    homePage.Open();
    homePage.LinkHomePage.Click();
    Assert.IsTrue(new HomePage(driver).IsOpen());
```

- Do the **same steps** from the previous bullet to **test** the **AddStudentsPage** and the **ViewStudentsPage links**:

```
    homePage.Open();
    homePage.LinkAddStudentPage.Click();
    Assert.IsTrue(new AddStudentPage(driver).IsOpen());

    homePage.Open();
    homePage.LinkViewStudentsPage.Click();
    Assert.IsTrue(new ViewStudentsPage(driver).IsOpen());
}
```

Create "View Students" Page Tests

The **TestViewStudentsPage test class** should inherit the **BaseTest base class**. Write the following **test methods** in the class: the **Test_ViewStudentsPage_Content()** method to **check page content** and the **Test_ViewStudentsPage_Links()** method to **check links to other pages**.

The **Test_ViewStudentsPage_Content()** test method should:

- **Instantiate** the **ViewStudentsPage** class, **open** the **View Students page** and **check its title and heading**:

```
[Test]
0 references
public void Test_ViewStudentsPage_Content()
{
    var page = new ViewStudentsPage(driver);
    page.Open();
    Assert.AreEqual("Students", page.GetPageTitle());
    Assert.AreEqual("Registered Students", page.GetPageHeadingText());
```

- **Invoke** the **GetRegisteredStudents()** method to **get all students on the page**:

```
    var students = page.GetRegisteredStudents();
```

- **Assert** that **each student record contains "(" and finishes with ")"**:

```
    foreach (string st in students)
    {
        Assert.IsTrue(st.IndexOf("(") > 0);
        Assert.IsTrue(st.LastIndexOf(")") == st.Length-1);
    }
}
```

For the **Test_ViewStudentsPage_Links()** test method, **go** to the **View Students page** and **click on each of the links**. They should **open the correct pages**. Write the test by yourself, it is very similar to the **Test_HomePage_Links()** test.

Create "Add Student" Page Tests

The **TestAddStudentPage** inherits the **BaseTest** class and has the following **test methods**:

- **Test_TestAddStudentPage_Content()**
  - **Instantiate** the **AddStudentPage class** with driver
  - **Open** the **Add Student page**
  - **Assert** the **page title** and **heading** are correct
  - **Assert** the **form fields are empty**
  - **Assert** that the **form button has a correct text**
- **Test_TestAddStudentPage_Links()**
  - **Instantiate** the **AddStudentPage class** with driver
  - **Open** the **Add Student** page
  - **Assert** the **Home page link** opens the page
  - **Assert** the **Add Student page link** opens the page
  - **Assert** the **View Students page link** opens the page
- **Test_TestAddStudentPage_AddValidStudent**
  - **Instantiate** the **AddStudentPage class** with driver
  - **Open** the **Add Student page**
  - Generate a **unique student name and email**:
    - **string name = "New student" + DateTime.Now.Ticks;**
    - **string email = "email" + DateTime.Now.Ticks + "@email.com";**
  - **Invoke** the **AddStudent(string name, string email)** method
  - **Instantiate** the **ViewStudentsPage** class with driver
  - **Assert** the **View Students** page is open
  - **Assert** the **page contains the new student**
    - **studentsPage.GetStudentsList()** collection should include the new student
- **Test_TestAddStudentPage_AddInvalidStudent()**
  - **Instantiate** the **AddStudentPage** class with driver
  - **Open** the **Add Student** page
  - **Invoke** the **AddStudent(string name, string email)** method with **invalid data**, e.g. an empty name
  - **Assert** the **Add Student** page is still open
  - **Assert** that the error message contains the **Cannot add student** text
  - **Invoke** the **GetErrorMsg()** method

You already know how to write these **test cases** – **write them on your own**.

Run Tests

Run all tests and **ensure they work correctly**.

This is how your **final set of tests** may look like: