# Selenium-WebDriver-POM: Lab

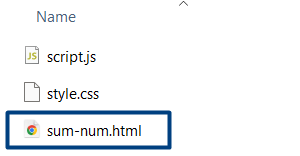
## "Sum Two Numbers"

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**

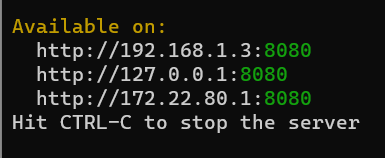
* Open Command Prompt (CMD) or PowerShell
* Go to Your Project Folder

**cd path\to\your\project**

* Start the Local Server with Node.js

**npx http-server**

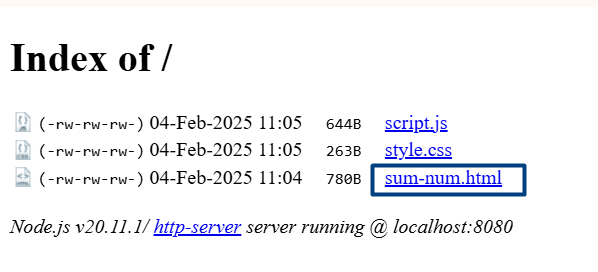
* This will start a local web server and show something like:



* 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.



### 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.

### 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.

## "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:

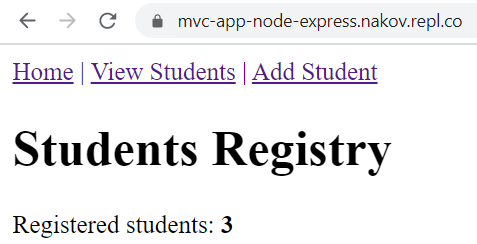
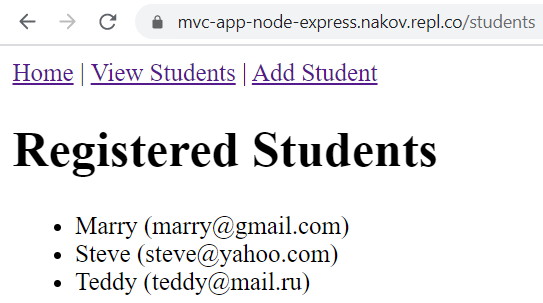
**npm install**

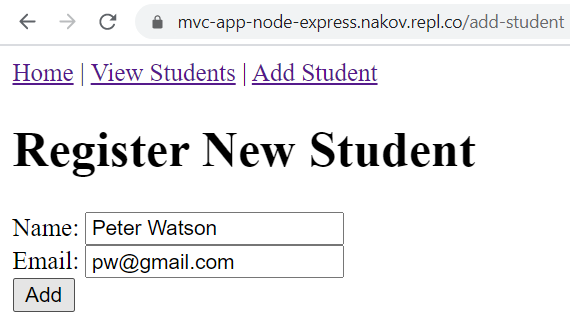
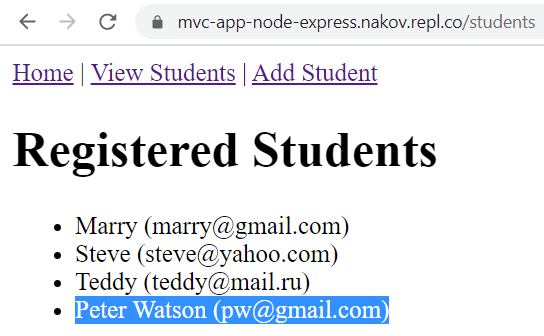
* Run the app using the command defined in the package.json:

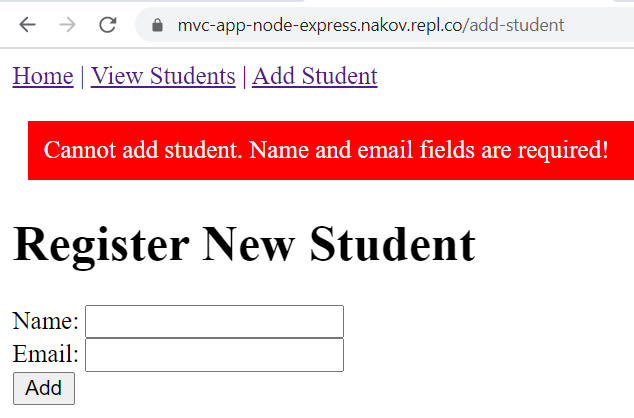
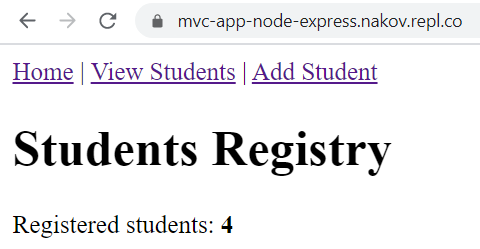
**npm start**

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

**http://localhost:8080**

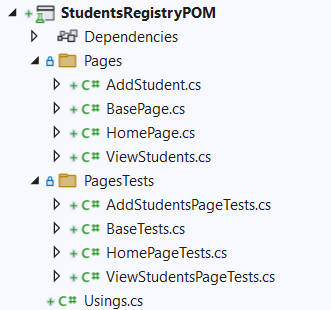
 

**Write automated Selenium UI tests for the following app, which holds a registry of students:**

### 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:**



### 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:

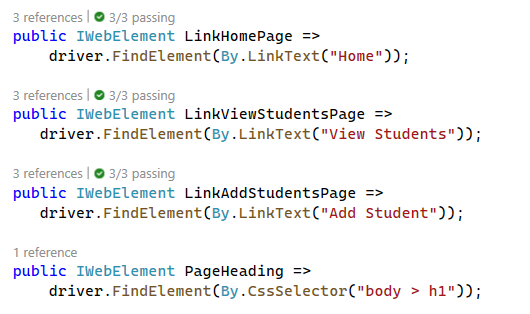
* Field: IWebDriver driver (protected and readonly)
* Constructor: BasePage(IWebDriver driver)
* Virtual property: PageUrl
* Properties: LinkHomePage, LinkViewStudentsPage, LinkAddStudentsPage, ElementTextHeading
* Method: Open() => driver.Url = this.PageUrl;
* Method: IsOpen() => driver.Url == this.PageUrl;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

Картина, която съдържа текст

Описанието е генерирано автоматично

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:



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

Картина, която съдържа текст

Описанието е генерирано автоматично

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:

Картина, която съдържа текст

Описанието е генерирано автоматично

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

Картина, която съдържа текст

Описанието е генерирано автоматично

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

Картина, която съдържа текст

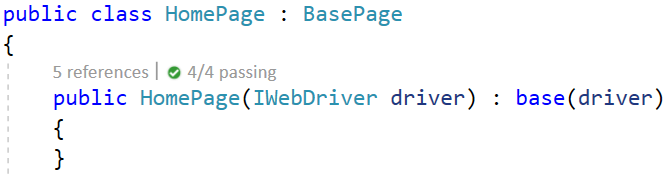
Описанието е генерирано автоматично

#### Create the "HomePage" Page Object Class

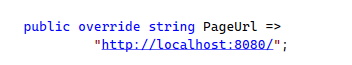
The HomePage **class** **inherits** the BasePage **class** (can use its properties and methods) and should contain:

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

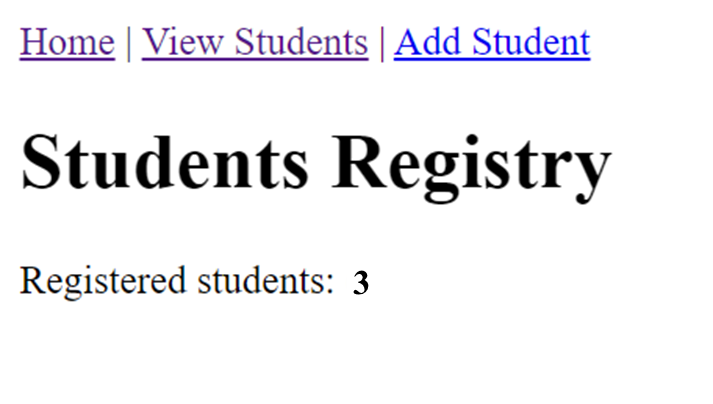
**Inherit** the BasePage class and **use its constructor** **by** **providing it with the driver**:



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



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



The property looks like this:



#### 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:

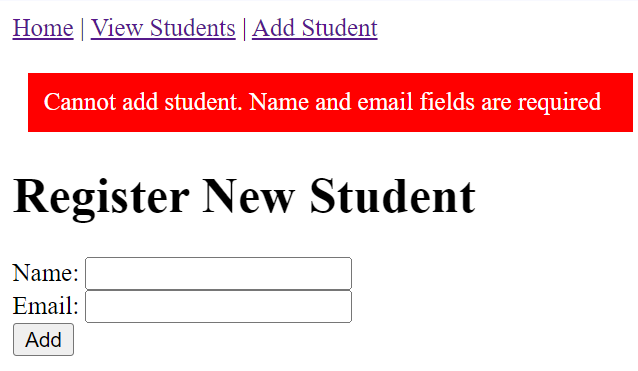


#### Create the "AddStudentPage" Page Object Class

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:

Картина, която съдържа текст

Описанието е генерирано автоматично

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

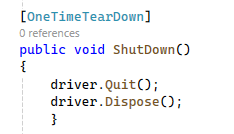
### Write Selenium POM Tests

#### 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:

Картина, която съдържа текст

Описанието е генерирано автоматично



#### 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**:



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**:

Картина, която съдържа текст

Описанието е генерирано автоматично

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



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



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

Картина, която съдържа текст

Описанието е генерирано автоматично

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**:

Картина, която съдържа текст

Описанието е генерирано автоматично

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

Картина, която съдържа текст

Описанието е генерирано автоматично

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

Картина, която съдържа текст, закрито, екранна снимка

Описанието е генерирано автоматично

#### 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**:

Картина, която съдържа текст

Описанието е генерирано автоматично

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



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

Картина, която съдържа текст

Описанието е генерирано автоматично

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:

Картина, която съдържа текст

Описанието е генерирано автоматично