Automation Assignment:

Building Automation Framework for https://demoqa.com/

Test Cases to Implement:

1. TC01: Verify user can successfully submit student registration form with valid data
2. TC02: Verify form displays appropriate validation messages for empty required fields
3. TC03: Verify email field validates different input formats correctly
4. TC04: Verify dynamic button state changes and interactions
5. TC05: Verify drag and drop functionality handles different scenarios
6. TC06: Verify web table supports CRUD operations
7. TC07: Verify alert handling and responses
8. TC08: Verify frame navigation and content
9. TC09: Verify book store search and collection management
10. TC10: Verify slider functionality and value updates
11. TC11: Verify progress bar operations
12. TC12: Verify date picker functionality
13. TC13: Verify modal dialogs display and close correctly
14. TC14: Verify accordion sections expand and collapse properly
15. TC15: Verify auto-complete functionality for single and multiple inputs

Session 1: Test Automation Fundamentals

Requirements:

- Create basic xUnit project
- Install dependencies:
  + Selenium.WebDriver
  + Selenium.Support
  + WebDriverManager
- Setup basic test structure

Session 2: Element Location Practice

Requirements:

1. Find locators for defined 15 test cases
2. For each element provide:
   - ID selector (if available)
   - CSS selector (minimum 2 different ways)
   - XPath (minimum 2 different ways)
   - Comments explaining preferred choice and why

3. Required Format:

```
public class ElementLocators
{
    public class TC01_VerifyStudentRegistration
    {
        // Element: First Name Input
        // ID: Good stability, preferred
        public static By FirstName_Id = By.Id("firstName");

        // CSS: Alternative options
        public static By FirstName_CSS1 = By.CssSelector("#firstName");
        public static By FirstName_CSS2 = By.CssSelector("input[placeholder='First Name']");

        // XPath: Alternative options
        public static By FirstName_XPath1 = By.XPath("//input[@id='firstName']");
        public static By FirstName_XPath2 = By.XPath("//input[@placeholder='First Name']");
    }
}
```

Session 3: Selenium WebDriver Basic

Requirements:

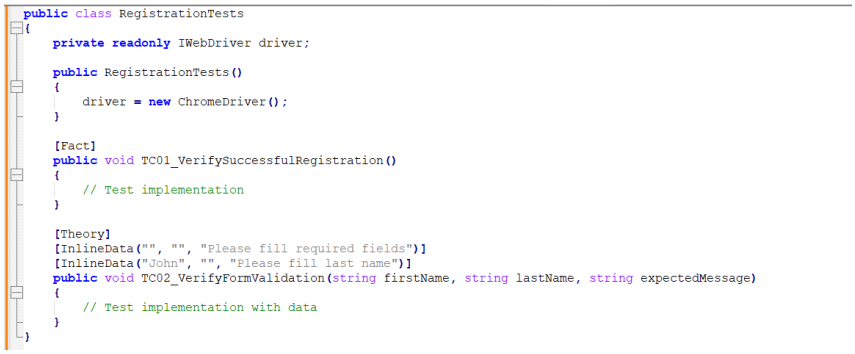
1. Implement basic test structure for 15 test cases:

```
public class TestCases
{
    private IWebDriver driver;
    public void Setup()
    {
        driver = new ChromeDriver();
    }

    public void Test01_StudentRegistration()
    {
        try
        {
            // Navigation
            // Element interactions
            // Validation
        }
        catch(Exception ex)
        {
            TakeScreenshot();
            throw;
        }
    }

    public void Cleanup()
    {
        driver?.Quit();
    }
}
```

Session 4: xUnit Framework Integration

Requirements:

1. Refactor existing tests to use xUnit:



```
public class RegistrationTests
{
    private readonly IWebDriver driver;

    public RegistrationTests()
    {
        driver = new ChromeDriver();
    }

    [Fact]
    public void TC01_VerifySuccessfulRegistration()
    {
        // Test implementation
    }

    [Theory]
    [InlineData("", "", "Please fill required fields")]
    [InlineData("John", "", "Please fill last name")]
    public void TC02_VerifyFormValidation(string firstName, string lastName, string expectedMessage)
    {
        // Test implementation with data
    }
}
```

2. Requirements:

- Convert all test cases to xUnit format
- Implement proper assertions
- Add test categories using [Trait]
- Setup test context sharing
- Implement setup and cleanup
- Use test data attributes

Session 5: Page Object Model Implementation

Requirements:

1. Create page classes for all pages:

```
public class RegistrationPage
{
    private readonly IWebDriver driver;

    // Elements
    private IWebElement FirstName => driver.FindElement(By.Id("firstName"));
    private IWebElement LastName => driver.FindElement(By.Id("lastName"));
    private IWebElement Submit => driver.FindElement(By.Id("submit"));

    // Actions
    public void EnterFirstName(string name) => FirstName.SendKeys(name);
    public void EnterLastName(string name) => LastName.SendKeys(name);
    public void SubmitForm() => Submit.Click();

    // Validations
    public bool IsRegistrationSuccessful() => // validation logic
}
```

Implementation Requirements:

Create page classes for all test scenarios

Encapsulate elements and actions

Implement navigation methods

Add validation methods

Handle page transitions

Implement waiting strategies

Framework Structure:

/Pages

- RegistrationPage.cs
- TablePage.cs
- AlertsPage.cs
- etc.

/Tests

- Using page objects

Must Include:

Element encapsulation

Action methods

Validation methods

Navigation logic

Wait strategies

Error handling

Session 6: Data-Driven Framework

Requirements:

1. Setup Test Data Structure:

/TestData

/Registration

- valid-users.json
- invalid-users.json

/WebTables

- table-data.json

/BookStore

- books.json

2. Implement Data Providers:

```
public class TestData
{
    public static IEnumerable<object[]> RegistrationData()
    {
        return JsonConvert.DeserializeObject<List<object[]>>(
            File.ReadAllText("TestData/Registration/valid-users.json"));
    }
}
```

3. Update Tests:

```
[Theory]
[MemberData(nameof(TestData.RegistrationData), MemberType = typeof(TestData))]
public void TC01_VerifyRegistration(string fname, string lname, string email)
{
    // Test implementation using data
}
```

4. Requirements:

- Create test data files for all applicable tests
- Implement data readers
- Update tests to use test data
- Add data validations
- Support multiple data formats
- Handle test data cleanup

Session 7: Framework Architecture

Requirements:

1. Project Structure:

/Core

/Driver

- DriverFactory.cs
- BrowserConfig.cs

/Config

- ConfigReader.cs
- Settings.json

/Logging

- Logger.cs

/Utils

- Screenshot.cs
- WaitHelper.cs

/Pages

/Tests

/TestData

2. Base Classes Implementation:

```
public abstract class BasePage
{
    protected readonly IWebDriver Driver;
    protected readonly Logger Logger;

    protected BasePage(IWebDriver driver)
    {
        Driver = driver;
        Logger = new Logger();
    }
}

public abstract class BaseTest
{
    protected IWebDriver Driver;
    protected ConfigReader Config;

    [OneTimeSetUp]
    public void Init()
    {
        Config = new ConfigReader();
        Driver = DriverFactory.CreateDriver(Config.Browser);
    }
}
```

3. Core Features:

- Cross-browser support
- Configuration management
- Logging system
- Screenshot capture
- Wait utilities
- Error handling
- Resource management

Session 8: Reporting Integration

Requirements:

1. ExtentReports Setup:

```
public class ReportManager
{
    private static ExtentReports extent;
    private static ExtentTest test;

    public static void StartTest(string testName)
    {
        test = extent.CreateTest(testName);
    }

    public static void LogStep(Status status, string message)
    {
        test.Log(status, message);
    }
}
```

2. Test Implementation:

```
[Fact]
public void TC01_VerifyRegistration()
{
    ReportManager.StartTest("Student Registration");
    ReportManager.LogStep(Status.Info, "Navigating to form");
    // Test steps with logging
    ReportManager.LogStep(Status.Pass, "Registration successful");
}
```

Session 9: API Testing

Requirements:

1. Setup API Tests:

```
public class BookStoreApiTests
{
    private readonly RestClient client;

    [Fact]
    public async Task VerifyGetBooks()
    {
        var request = new RestRequest("BookStore/v1/Books");
        var response = await client.ExecuteAsync(request);
        Assert.Equal(200, (int)response.StatusCode);
    }
}
```

2. Implementation Requirements:

- Setup RestSharp
- Create API test cases
- Handle authentication
- Validate responses

- Test error scenarios
- Combine UI and API tests
- API test data management

3. API Test Coverage:

- GET operations
- POST operations