Advanced Selenium Assignments

These assignments will help you practice implementing the Page Object Model (POM) pattern with NUnit testing framework using https://the-internet.herokuapp.com, a website specifically designed for automation practice.

Prerequisites

- Visual Studio (2019 or later)
- .NET Framework or .NET Core
- NUnit NuGet package
- NUnit3TestAdapter NuGet package
- Selenium.WebDriver NuGet package
- Selenium.Support NuGet package
- WebDriver executable for your preferred browser (ChromeDriver, GeckoDriver, etc.)

General Requirements for All Assignments

1. Implement proper Page Object Model pattern
    - Each page should have its own class
    - Page elements should be encapsulated
    - Use meaningful method names for actions
    - Return appropriate page objects for navigation actions
2. Follow NUnit best practices
    - Use appropriate attributes (`[TestFixture]`, `[Test]`, `[TestCase]`, etc.)
    - Implement proper setup and teardown methods
    - Use descriptive test names
    - Implement appropriate assertions
3. Use explicit waits when necessary
    - Avoid using `Thread.Sleep()`
    - Use `WebDriverWait` with appropriate `ExpectedConditions`
4. Implement proper exception handling
    - Tests should be resilient against common Selenium exceptions
    - Handle timeouts appropriately
5. Include meaningful comments where necessary
    - Document complex logic or wait conditions
    - Explain test case coverage

Assignment 1: Login Page Automation

Objective: Create a POM implementation for the login functionality with data-driven tests.

URL: https://the-internet.herokuapp.com/login

Requirements:

1. Create a `LoginPage` class with:
    - Element locators for username, password, login button, and status messages
    - Methods for entering credentials, clicking login, and checking error/success messages
2. Create a `SecurePage` class for the page after successful login with:
    - Element locators for logout button and success message
    - Method to verify successful login
    - Method to logout
3. Implement the following test cases using the created page objects:
    - Valid login (username: "tomsmith", password: "SuperSecretPassword!")
    - Invalid username
    - Invalid password
    - Empty credentials
4. Use the `[TestCase]` attribute to create data-driven tests for the scenarios above
5. Use assertions to verify:
    - Error messages for invalid login attempts
    - Success message for valid login
    - URL changes appropriately

Expected Output: A complete test suite that validates the login functionality with appropriate page objects and test cases.

Assignment 2: Dynamic Loading Elements

Objective: Create a robust synchronization implementation using POM and explicit waits.

URL: https://the-internet.herokuapp.com/dynamic_loading

Requirements:

1. Create a base `DynamicLoadingPage` class with common elements and methods
2. Create derived classes for each example:
    - `DynamicLoadingExample1Page` (Element hidden)
    - `DynamicLoadingExample2Page` (Element rendered after the fact)
3. Implement methods that:
    - Click the start button
    - Wait for the loading indicator to disappear

    - Wait for the hidden element to appear
    - Return the text of the element once loaded

4. Create test cases that:

    - Verify the correct text appears after loading for each example
    - Verify the loading process works correctly
    - Test the behavior with different timeout values

5. Create a reusable waiting utility class that can be used by both page objects

Expected Output: A robust implementation that demonstrates proper use of explicit waits with the Page Object Model pattern.

Assignment 3: Data Tables and Sorting

Objective: Create a POM for data tables with verification of sorting functionality.

URL: https://the-internet.herokuapp.com/tables

Requirements:

1. Create a `DataTablesPage` class with:

    - Methods to get all row data
    - Methods to sort by a specific column
    - Methods to verify if data is sorted correctly (ascending/descending)

2. Implement test cases that:

    - Verify the initial state of the tables
    - Click on column headers to sort
    - Verify data is sorted correctly after clicking
    - Test sorting for each column

3. Create a utility method to compare values (strings, numbers, currency, dates)

4. Use `[TestCase]` to parameterize tests for different columns and sort directions

5. Handle both tables on the page (Example 1 and Example 2)

Expected Output: A test suite that verifies the sorting functionality of the tables using POM pattern.

Assignment 4: Multiple Windows and Frames

Objective: Implement a POM that handles window switching and frame navigation.

URLs:

- https://the-internet.herokuapp.com/windows
- https://the-internet.herokuapp.com/iframe

Requirements:

1. Create a `WindowsPage` class with:
    - Method to click the "Click Here" link
    - Method to switch to the new window
    - Method to get text from the new window
    - Method to switch back to original window
2. Create a `FramePage` class with:
    - Method to switch to the frame
    - Methods to interact with the rich text editor
    - Method to switch back to main content
3. Implement test cases that:
    - Verify opening a new window and its content
    - Verify entering text in the iframe editor
    - Verify text formatting in the iframe editor (bold, italic)
    - Verify switching between multiple contexts
4. Ensure proper cleanup by closing additional windows in teardown
5. Create base methods in a parent class that can be reused for window/frame handling

Expected Output: A test suite that demonstrates proper handling of multiple windows and frames using the Page Object Model pattern.

Assignment 5: Comprehensive Test Suite

Objective: Create an integrated test suite that tests multiple features with shared page objects and utilities.

URLs: Multiple endpoints on https://the-internet.herokuapp.com

Requirements:

1. Create a `BasePage` class with common functionality:
    - Navigation methods
    - Wait utilities
    - Logging
    - Screenshot capabilities
2. Create specific page objects for at least 4 different features:
    - Form Authentication (/login)
    - Dropdown List (/dropdown)
    - Checkboxes (/checkboxes)
    - File Upload (/upload)
3. Implement a `TestBase` class that:
    - Sets up WebDriver with proper options
    - Takes screenshots on test failure

- Provides logging functionality
- Implements proper driver lifecycle management

4. Create category-based test organization:
   - Group tests by feature using `[Category]` attribute
   - Create smoke test category with critical tests
   - Create regression test category with all tests
5. Implement cross-browser capability:
   - Use a factory pattern to create different browsers
   - Parameterize tests to run on Chrome, Firefox, and Edge
   - Handle any browser-specific differences

Expected Output: A complete, well-organized test suite that demonstrates mastery of the Page Object Model pattern, NUnit testing framework, and Selenium WebDriver.

Submission Guidelines

1. Create a GitHub repository with your solution
2. Organize your code in the following structure:
   - `Pages/` - All page object classes
   - `Tests/` - All test classes
   - `Utils/` - Utility classes and helpers
   - `Drivers/` - WebDriver factory and configuration
3. Include a README.md with:
   - Setup instructions
   - Brief description of your implementation
   - Any assumptions or design decisions you made
4. Ensure all tests pass before submission
5. Include a brief report that discusses:
   - Challenges faced
   - How you applied POM principles
   - How you handled synchronization issues
   - Any improvements you would make with more time