A logo with colorful lines

AI-generated content may be incorrect.

**Guideline Automation Test Framework – Playwright C#**

|  |  |
| --- | --- |
| Created by: | Luca (Tai Pham) |
| Approved by: | Samuel (Khanh Sam) |

Contents

[CHANGE HISTORY 2](#_Toc1811695844)

[1. Introduction 2](#_Toc612005606)

[2. Goals of Automation for Manual Testers 2](#_Toc1327420962)

[3. Basic knowledge 2](#_Toc312866509)

[4. Understanding the Automation Framework Basics 3](#_Toc901779770)

[5. Setup environment 3](#_Toc1208918097)

[6. Technology Stack 5](#_Toc862152701)

[7. Access to the testing branch 5](#_Toc2077650775)

[8. Project structure 6](#_Toc2037413692)

[9. Writing Test 12](#_Toc501329739)

[Test Class Structure 12](#_Toc339894169)

[Page Action Pattern 13](#_Toc932832950)

[10. Best Practice 13](#_Toc1137887374)

[1. Test Organization 13](#_Toc640780116)

[2. Page Object Model 13](#_Toc419632855)

[3. Test Data Management 13](#_Toc1685983658)

[4. Error Handling 13](#_Toc1960771451)

[5. Parallel Execution 13](#_Toc1546785390)

[11. Running Test 14](#_Toc1358279857)

[1. Via Visual Studio Test Explorer 14](#_Toc330642780)

[2. Via Command Line 14](#_Toc576524385)

[12. Test Generator 15](#_Toc1268341288)

# **CHANGE HISTORY**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Date** | **Version** | **Created by** | **Job Title** | **Description of change** |
| 09/06/2023 | V0.1 | Luca | QC Auto | New status: in progress. |
|  | V0.1 | Samuel (Khanh) Sam | Automation QC Lead | New status: in review. |
|  | V1.0 | Samuel (Khanh) Sam | Automation QC Lead | New status: approved. |
|  | V1.1 | Samuel (Khanh) Sam | Automation QC Lead | Add Node install at Test Generator |
|  | V1.2 | Samuel (Khanh) Sam | Automation QC Lead | Add 7. Access to the testing branch  Add Image to setup repo |

# Introduction

* JLWeb\_GUI\_Playwright is an automated UI testing project designed to ensure the quality and reliability of the JLWeb application. It uses the Playwright framework, which allows automated browser testing across multiple browsers (such as Chrome, Firefox, and Edge) using C# and .NET 8.
* Key purpose:
  + **Automate User Actions:** Simulates real user interactions with the JLWeb application, such as clicking buttons, filling out forms, and navigating between pages.
  + **Regression Testing:** Quickly check that new changes do not break existing functionality.
  + **Cross-Browser Validation:** Ensures the application works consistently across different browsers.
  + **Repeatable and Reliable:** Automated tests can be run as often as needed, providing consistent results and reducing manual effort.

# Goals of Automation for Manual Testers

* **Accelerated Testing:** Quickly execute repetitive and time-consuming tests.
* **Early Feedback:** Identify defects earlier in the development cycle.
* **Regression Coverage:** Ensure that new changes do not introduce regressions in existing functionality.
* **Efficiency:** Free up manual testers to focus on exploratory testing, complex scenarios, and edge cases.
* **Collaboration:** Foster a better understanding and collaboration between manual and automation testing efforts.

# Basic knowledge

* **Basic C# and .NET Concepts:** 
  + Understand how to read C# code, especially test methods and assertions.
  + Recognize the structure of a test class, test methods (**[TestMethod]**), and test setup/cleanup (**[TestInitialize]**, **[TestCleanup]**).
  + Reference link: [CSharp](https://dotnet.microsoft.com/en-us/learn/csharp), [CSharp W3C](https://www.w3schools.com/cs/index.php), [MS Test](https://learn.microsoft.com/en-us/dotnet/core/testing/unit-testing-mstest-intro)
* **Playwright Test Automation Basics**
  + Know what Playwright is: a tool for automating browser actions for testing.
  + Understand that tests simulate user actions (clicks, navigation, form entry) and verify UI behavior.
  + Be able to interpret test steps in code and relate them to manual actions in the browser.
  + Reference link:
    - [Playwright](https://playwright.dev/dotnet/docs/intro)
    - [Build your first end-to-end test with Playwright - Training | Microsoft Learn](https://learn.microsoft.com/en-us/training/modules/build-with-playwright/)
* **Basic knowledge of selector in playwright, also known as Css or Xpath**
  + Reference Link:
    - [CSS cheatsheet](https://devhints.io/css)
    - [Xpath cheatsheet](https://devhints.io/xpath)
* **Repo**: [JobLogicTesting - Repos](https://joblogicltd.visualstudio.com/TMS/_git/JobLogicTesting)
  + Testing branch name: Iris/Refactoring/PlaywrightC#Demo

# Understanding the Automation Framework Basics

* **Framework Name:** Playwright C#
* **Repository Location:** [URL](https://joblogicltd.visualstudio.com/TMS/_git/JobLogicTesting?version=GBIris%2FRefactoring%2FMergePlaywrightc%23)
* **Key Components (Brief Overview):**
  + **Test Suites:** Collections of related test cases.
  + **Test Scripts:** Individual automated tests designed to verify specific functionalities.
  + **Environment:** How to configure a running environment.
  + **Test Data:** The data will come from API (Get or Create)
  + **Reporting:** How test results are generated and viewed.

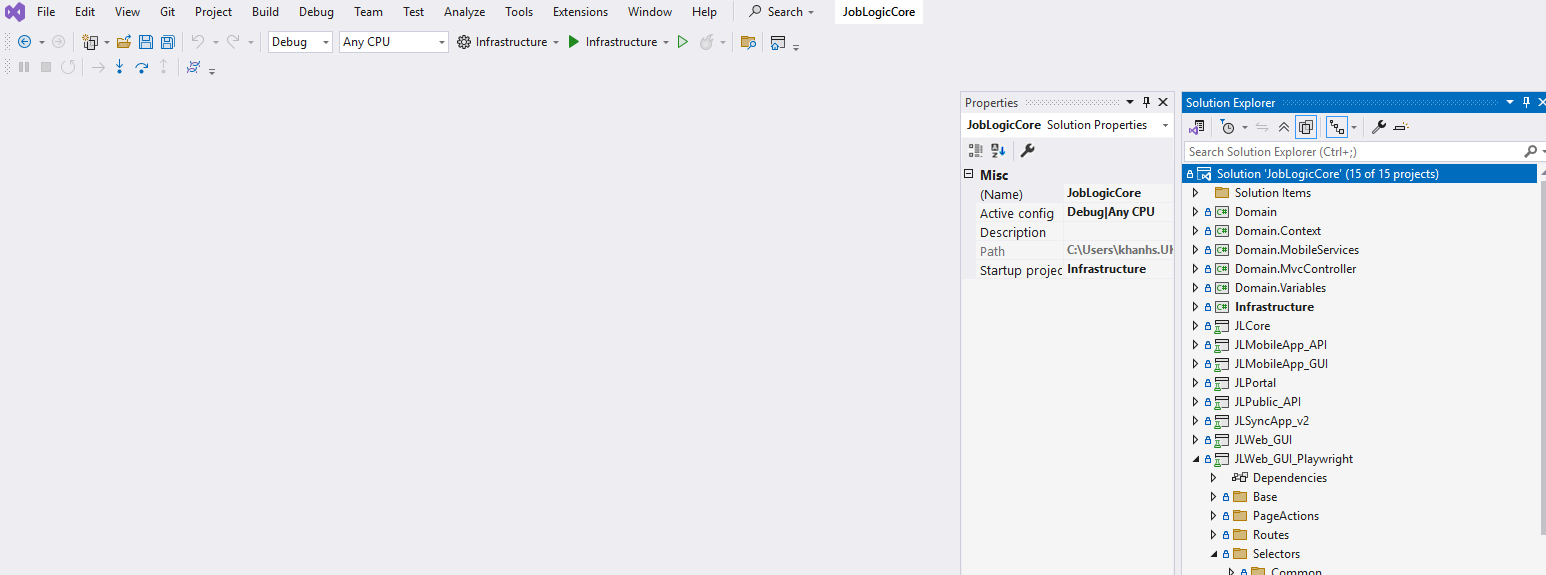
# Setup environment

* + Install .NET 8 SDK:Download and install from [**Microsoft .NET Download**](https://dotnet.microsoft.com/download/dotnet/8.0).
  + Install Node.js (optional):Some Playwright features/tools may require Node.js. Download from [**Node.js Download**](https://nodejs.org/).
  + InstallVisual Studio 2022 : [Visual Studio 2022 IDE - AI for coding debugging and testing](https://visualstudio.microsoft.com/vs/)
  + Install VsCode: [Download Visual Studio Code - Mac, Linux, Windows](https://code.visualstudio.com/download)
  + Clone the Project Repository at [JobLogicTesting - Repos](https://joblogicltd.visualstudio.com/TMS/_git/JobLogicTesting) by using [Git - Downloads](https://git-scm.com/downloads) or using [Sourcetree | Free Git GUI for Mac and Windows](https://www.sourcetreeapp.com/)
  + Import project to Visual Studio by clicking on **.sln** file

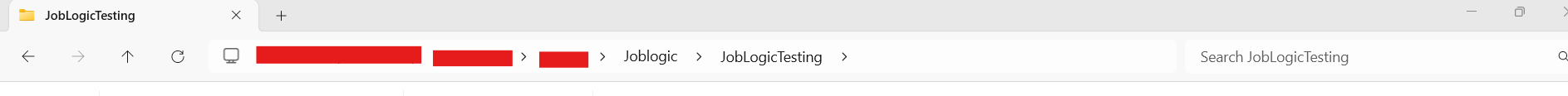
A screenshot of a computer

AI-generated content may be incorrect.

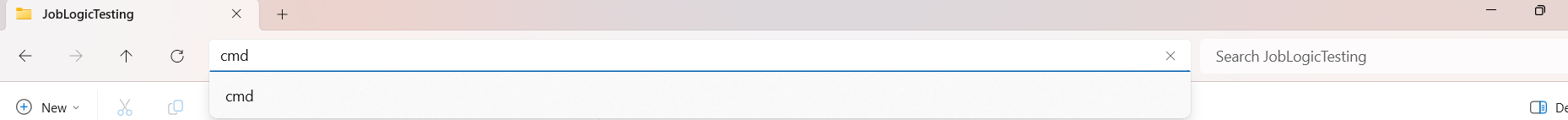
* + Wait until the Vscode is opened



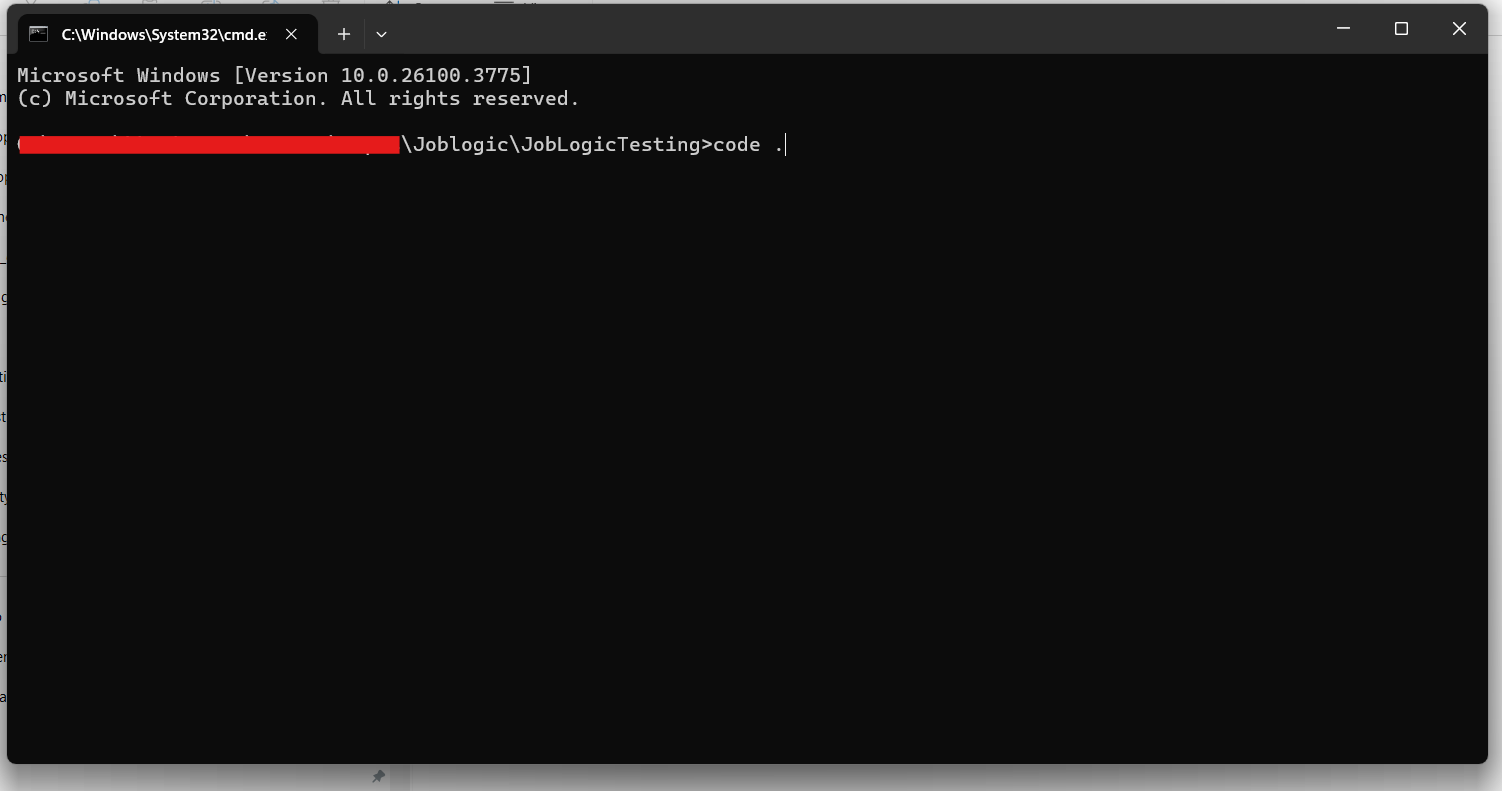
* Navigate to the source code again

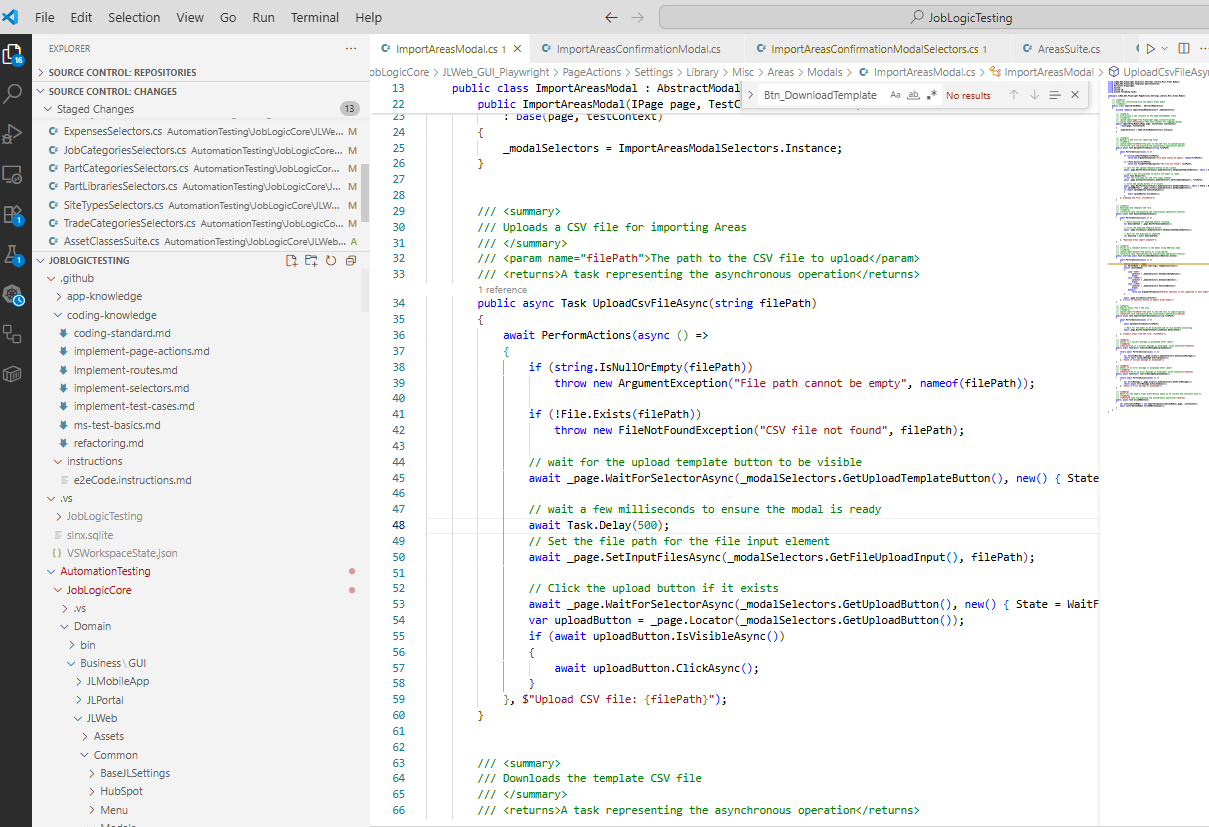


* + Type `cmd` on the search bar to open the command line interface directly



* Type `code .` to open the vscode





# Technology Stack

Test Framework: MSTest.Sdk 3.6.4

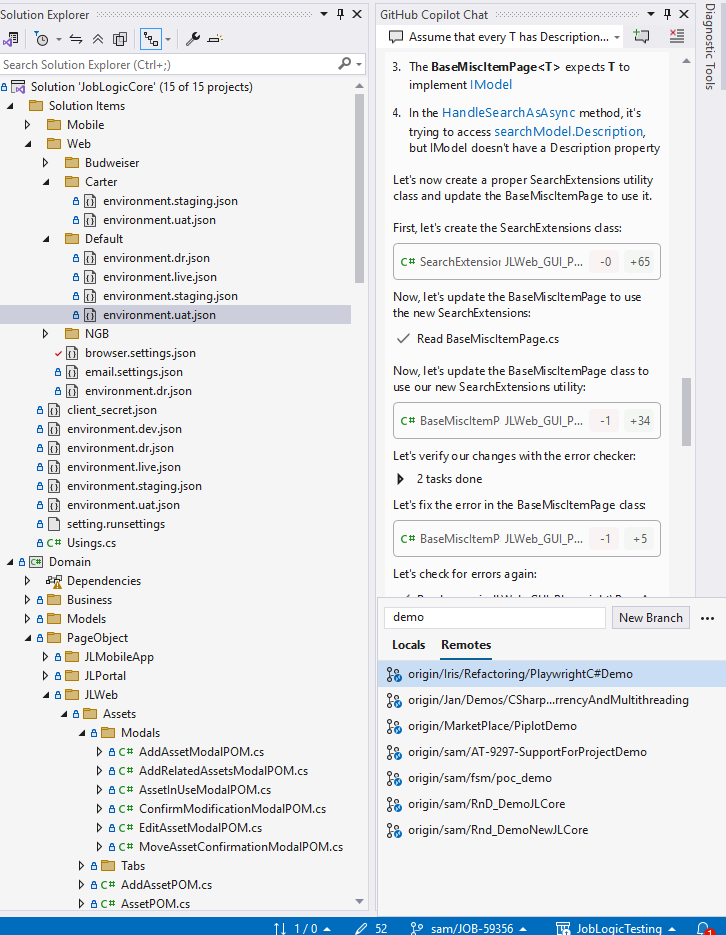
Browser Automation: Microsoft.Playwright.MSTest 1.52.0

Target Framework: .NET 8.0

Language: C# with latest language features

Architecture: Page Object Model (POM) with Action-based pattern

# Access to the testing branch

1. Open the Visual Studio
2. On the right side of the IDE, Click on the Git icon
3. Navigate to Remote Tab > Search Playwright C#Demo
4. Select the 1st result
5. Wait until the Playwright project is visible
6. Right click on the Solution “JoblogicCore” at Solution Explorer
7. Click on “Build Solution” or “Ctrl + Shift + B”
8. Switch to your created branch before moving to the next step (Git Knowledge)
9. Start to follow the next steps to understand about the project
10. 

# Project structure

**Project structure overview**

A screen shot of a computer

AI-generated content may be incorrect.

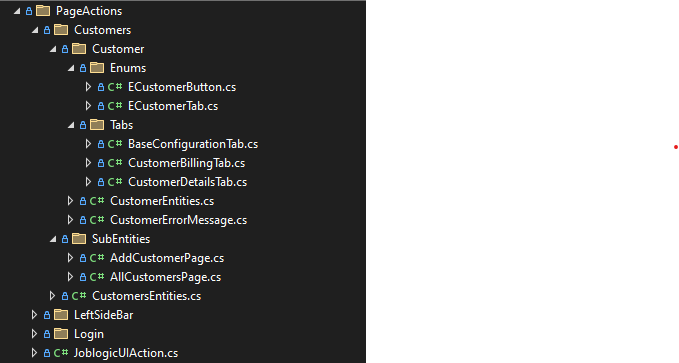
**Details**

* **Base**

A screen shot of a computer

AI-generated content may be incorrect.

* + AppPlaywrightUI: acts as the entry point for initializing, configuring, and managing the Playwright UI test environment, ensuring consistent context, environment settings, and reporting utilities are available for all automated UI tests.
  + PlaywrightBaseTest: acts as the base class for all Playwright UI tests, handling setup, cleanup, and providing shared context and utilities. This ensures all tests are consistent, reliable, and easy to maintain.
* **PageActions**



* + Each feature will have each folder accordingly, e.g Customers
  + Inside feature, based on the UI, there are 3 types of components: Page, Tab, Modal
  + Same as for Jobs, Quote features
* **Routes**

A screenshot of a computer

AI-generated content may be incorrect.

* + Contains url of object, such as ”{domain}/Customer/Create”, “{domain}/ Customer/Detail/”
  + Create this one for another feature
* **Selectors**

A screenshot of a computer

AI-generated content may be incorrect.

* + The selector is the ‘address’ of the element.
  + Organize selector by **Page, Tab, Modal**
  + How to find a selector of element: [Reference](https://www.browserstack.com/guide/css-selectors-in-selenium), [Reference2](https://www.selenium.dev/documentation/webdriver/elements/locators/)
  + Note: Unlike Selenium, Playwright only has one type of selector that is ‘string’, no matter if it is CSS Selector, ID, Xpath, … Can put all of these as a string.
* **Templates**

A screenshot of a computer

AI-generated content may be incorrect.

* + Purpose: Reusable components and templates help standardize test steps and data setup.
  + Depending on the component that the script is interacting with:
    - If it is a **Page**, it will have the necessary action of the page such as input required fields or optional fields (**Abstract**) in a form and validation things, create something by OTF (**Interface**), …
    - If it is a **Tab**, it will have the necessary action of the tab such as input required fields or optional fields (**Abstract) in** a form and validation things (**Interface**)
    - If it is a **Modal**, it will have the necessary action such as Click on command button (Cancel, Confirm, OK, ..) or handle OTF action (**Interface)**
* **Utils**

A black screen with white text

AI-generated content may be incorrect.

* + The Utils folder is a toolbox for the automation project, providing shared, general-purpose code that supports and simplifies the writing and maintenance of automated tests. Manual testers do not usually need to modify these files, but understanding their purpose can help when reading test code or troubleshooting issues.
  + Purpose:
    - **Code Reusability:**  
      Contains functions or classes that can be used in multiple places, reducing code duplication.
    - **Simplify Test Code:**  
      Provides ready-made solutions for common tasks (e.g., date formatting, string manipulation, file handling, waiting elements, logging, etc.), making test scripts cleaner and easier to read.
    - **Centralized Maintenance:**  
      If a utility function needs to be updated or fixed, it can be changed in one place (the Utils folder) and all tests benefit from the update.
    - **Framework Support:**  
      May include helpers for Playwright-specific actions, custom assertions, configuration readers, or reporting tools.
* **Tests**

A black screen with white text

AI-generated content may be incorrect.

* + Including test suites of features
  + What is inside?

A screen shot of a computer program

AI-generated content may be incorrect.

* + - Based on manual test case, this case is about access to billing tab of customer and set billing information then save and verify
    - Step-by-Step Logic
      * 1. Create a New Customer
        + The script creates a new customer record in the system.
        + This is like a manual tester going to the "Add Customer" screen and filling in the required details.
      * 2. Navigate to Customer Details
        + The test navigates to the detail page of the newly created customer.
      * 3. Prepare Billing Address Data
        + The script generates fake billing address information (name, email, address lines, phone, etc.).
      * 4. Navigate to the Billing Tab
        + The script switches to the "Billing" tab in the customer’s profile.
        + As manually, you would click on the "Billing" tab.
      * 5. Enter and Save Billing Information
        + The test fills in the billing address fields and clicks the "Save" button.
        + Manually, you would enter the details and click "Save" to store the billing address.
      * 6. Verify Billing Information is Saved
        + The script checks that the billing information was saved correctly.
* **Environment**
  + **A screenshot of a computer

    AI-generated content may be incorrect.**
  + All credentials are placed in Web/Default/\*.json and by each environment



* + How to set up a credential you want to run the script. e.g

{

"EmailAddress": "youremail@joblogic.com",

"Password": "yourpassword",

"Region": "UK",

"Type": "**JLVN-ABCDEFGH**"

}

* + Put the type **JLVN-ABCDEFGH** to file setting.runsettings for running the specified account  
    A screenshot of a computer program

    AI-generated content may be incorrect.
  + The framework uses `setting.runsettings` for configuration:
    - **Environment**: dev, uat, live, dr, staging
    - **Browser**: Chrome
    - **Execution Mode**: Headless or Headed
    - **Parallel Execution**: Configurable worker count (default: 3)
    - **Test Categories**: Filter tests by categories

# Writing Test

### Test Class Structure

[TestClass]  
[TestCategory("Your-Category")]  
public class YourTestSuite : PlaywrightBaseTest  
{  
 #region Declaration  
 private string? testName;  
 private YourModel? testModel;  
 #endregion  
  
 #region Preconditions  
 [TestInitialize]  
 public void PreCond()  
 {  
 ServicesRegistry.GetServices<BaseRest>().SetMetaData(TestContext.Properties[MetaDataKey]);  
 testName = TestContext.TestName ?? "Unknown";  
 }  
 #endregion  
  
 #region Cleanup  
 [TestCleanup]  
 public void Cleanup()  
 {  
 // Clean up test data created during test execution  
 // Use API calls to remove created entities  
 }  
 #endregion  
  
 #region Tests  
 [TestMethod("Test Description")]  
 [Priority(1)]  
 public async Task YourTestMethod()  
 {  
 // Arrange: Set up test data via API  
 testModel = await YourEntityVal.Instance.Create(new YourModel());  
   
 // Act: Perform UI actions  
 JoblogicUIAction joblogicPages = new(Page, TestContext);  
 await joblogicPages.NavigateToEntityAsAsync(EJoblogicEntity.YOUR\_ENTITY, testModel.Id);  
   
 // Assert: Verify expected behavior  
 // Perform assertions  
 }  
 #endregion  
}

### Page Action Pattern

public class YourPageAction : AbstractLoggerAction  
{  
 public YourPageAction(IPage page, TestContext testContext) : base(page, testContext)  
 {  
 }  
  
 public async Task PerformActionAsync(YourModel model)  
 {  
 await PerformActions(async () =>  
 {  
 // Your UI interactions here  
 await \_page.ClickAsync(YourSelectors.SubmitButton);  
 await \_page.FillAsync(YourSelectors.NameField, model.Name);  
 }, "Description of the action being performed");  
 }  
}

# Best Practice

### 1. Test Organization

* **Group tests by feature/module** (Customer, Settings, etc.)
* **Use descriptive test names** that explain the scenario
* **Add test categories** for easy filtering
* **Set test priorities** for execution order

### 2. Page Object Model

* **Separate selectors** into dedicated selector classes
* **Create page actions** for reusable UI interactions
* **Use the Abstract Action pattern** for consistent logging
* **Implement interfaces** for contract-based development

### 3. Test Data Management

* **Create test data via API** in TestInitialize
* **Clean up test data** in TestCleanup
* **Use meaningful test data** with proper naming conventions
* **Avoid hard-coded values** - use configuration or generated data

### 4. Error Handling

* **Automatic screenshot capture** on test failures
* **Structured logging** with test steps
* **Exception handling** in cleanup methods
* **Retry mechanisms** for flaky elements

### 5. Parallel Execution

* **Tests run in parallel** by default (3 workers)
* **Each test gets isolated browser context**
* **Credential management** handles concurrent logins
* **Thread-safe test data creation**

# Running Test

### 1. Via Visual Studio Test Explorer

1. Build the solution
2. Open Test Explorer
3. Filter by categories if needed
4. Right-click and run selected tests

### 2. Via Command Line

# Run all tests  
 dotnet test  
  
 # Run specific category  
 dotnet test --filter "TestCategory=Customer"  
  
 # Run with custom settings  
 dotnet test --settings setting.runsettings  
  
 # Run in headless mode  
 dotnet test --settings setting.runsettings -- Playwright.LaunchOptions.Headless=true

# Test Generator

Install PowerShell on Windows Machine (make sure that you have enough privilege to execute this command)

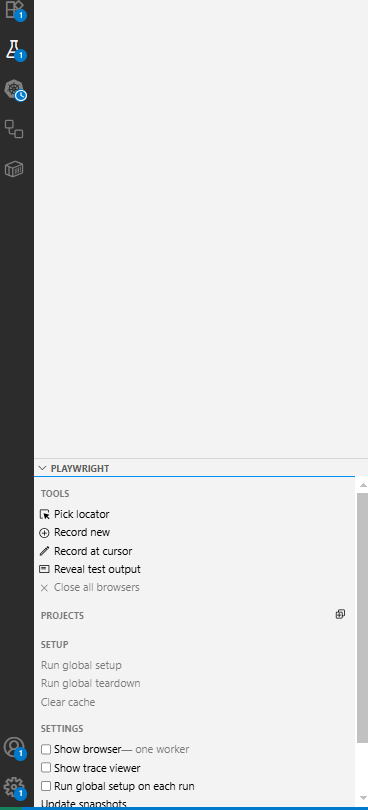
* Open the **Command Prompt** and run following command
  + winget install --id Microsoft.Powershell --source winget
  + winget install --id Microsoft.Powershell.Preview --source winget

In case, you cannot perform the 1st step of using PowerShell. Please follow the installation instructions at [Installation | Playwright](https://playwright.dev/docs/intro)

What is a Test Generator?  
 pwsh .\AutomationTesting\JobLogicCore\JLWeb\_GUI\_Playwright\bin\Debug\net8.0\playwright.ps1 codegen

Reference link: [Test Generator](https://playwright.dev/dotnet/docs/codegen)

Test Generator Extension in VsCode: [Playwright Test for VSCode - Visual Studio Marketplace](https://marketplace.visualstudio.com/items?itemName=ms-playwright.playwright)



How to use it?

Please refer: [GenerateCode.mp4](https://joblogicltd-my.sharepoint.com/:v:/g/personal/taip_joblogic_com/ERZ9m3mircZGkOnW0T90FnkBxG4PrmqzCMa9ztTMYrxtaA?e=BKbejF)

How to run the associated test cases

Please refer : [GuidelineforAssociatedTestScript.mp4](https://joblogicltd.sharepoint.com/:v:/r/sites/DevelopmentDept/Shared%20Documents/JOBLOGIC/Training/BAU%20Training/QualityControl/AutomationTraining/AI%20%26%20Playwright/GuidelineforAssociatedTestScript.mp4?csf=1&web=1&e=JyQBi0), [GuidelineforFixingIssue\_RelatedPipeline.mp4](https://joblogicltd.sharepoint.com/:v:/r/sites/DevelopmentDept/Shared%20Documents/JOBLOGIC/Training/BAU%20Training/QualityControl/AutomationTraining/AzureDevOps/GuidelineforFixingIssue_RelatedPipeline.mp4?csf=1&web=1&e=UNIcY8&nav=eyJyZWZlcnJhbEluZm8iOnsicmVmZXJyYWxBcHAiOiJTdHJlYW1XZWJBcHAiLCJyZWZlcnJhbFZpZXciOiJTaGFyZURpYWxvZy1MaW5rIiwicmVmZXJyYWxBcHBQbGF0Zm9ybSI6IldlYiIsInJlZmVycmFsTW9kZSI6InZpZXcifX0%3D)

Getting Started with Playwright VsCode: [Getting started - VS Code | Playwright](https://playwright.dev/docs/getting-started-vscode)