**Inspired Testing Automation Assessment**

**Task 1**

API endpoint testing using RestSharp and Visual Studio 2019. For this task a Unit Test Project has been used.

5 test cases are present as part of Task 1.

* 1 POST a new Customer
* 2 GET a Single Customer (one using JSON)
* 2 GET 10 Customers (one using JSON)

**How to run the test:**

1. Download the Inspired\_Testing\_Automation\_Assessment repository from GitHub
2. Open Inspired\_Testing\_Automation\_Task1
3. Import the solution file into Visual Studio
4. Build the solution file.
5. 2 Classes should be available – JsonContent.cs & Endpoint\_Testing.cs
   1. JsonContent.cs contains the set & get functions for the POST Method
   2. Endpoint\_Testing.cs is a Test Class that contains the 5 test cases to be executed
6. There are two ways to run the tests
   1. By going to the Test Explorer menu and run all the tests from there.
   2. By going to the Endpoint\_Testing.cs and right click on the Header of each Test Case and click on Run Test(s)
7. To check the results of the Test the desired test should be selected from the Test Explorer then go to Test Details Summary and click on “Open additional output for this result”.
8. You can right-click, Copy All and Paste the output to another file for further use.

**Task 2**

Frontend Testing using Selenium Chromedriver and Visual Studio 2019. For this task a Unit Test Project has been used.

4 test cases are present as part of Task 2. The test cases can only be run in Chrome.

* Test 1 – Login with a single user, make a deposit and logout
* Test 2 – Login with all users, make a deposit and logout
* Test 3 – Login with a single user, make a deposit, make a withdrawal and logout
* Test 4 – Login with a single user, make a deposit, make a withdrawal and logout using a JSON

**How to run the test:**

1. Download the Inspired\_Testing\_Automation\_Assessment repository from GitHub
2. Open Inspired\_Testing\_Automation\_Task2
3. Import the solution file into Visual Studio
4. Build the solution file.
5. 3 Classes should be available – Functions.cs, TestCases.cs & Task\_4.cs
   1. Functions.cs contains the general functions & variables used to create the Test Cases
   2. TestCases.cs is a Test Class and contains the 4 test cases to be executed
   3. Task\_4.cs contains the JSON string used by Test4
6. There are two ways to run the tests
   1. By going to the Test Explorer menu and run all the tests from there.
   2. By going to the TestCases.cs and right click on the Header of each Test Case and click on Run Test(s)
7. This project creates a Automation\_Test\_Screenshots folder on the Desktop of the user containing Screenshots from the Test Case to help validate the functionality of the system.
8. To check the results of the Test go to the Test Explorer and check the Test Case Status (failed/passed). If the Test Cases failed then they should include an error message explaining the reason of failure. Moreover as mentioned above the Screenshots can be used to confirm the functionality we are testing. Lastly the tests are not run in headless mode so the user can run them and actually check in real time that the test is been executed correctly and the results are as expected.

**Tools Used for this Project**

**Task 1**

* **RestSharp**: RestSharp is one of the most popular HTTP client library for .NET. Featuring automatic serialization and deserialization, request and response type detection, variety of authentications and other useful features. Due to not been so experienced with Endpoint Automation and the time constraints that the project had library that is popular and has extensive documentation was used. RestSharp was perfect as the functions used to create the Test Cases were straightforward and for the most part easy to use.
* **Coverlet.collector**: Coverlet is a cross platform code coverage library. NET, with support for line, branch and method coverage. Library that comes as part of the Visual Studio Unit Test Projects.
* **Microsoft**.**NET.Test.Sdk:** The MSbuild targets and properties for building .NET test projects. Library that comes as part of the Visual Studio Unit Test Projects.
* **MSTest**.**TestAdapter**: The adapter to discover and execute MSTest Framework based tests. Native library that comes as part of the Visual Studio Unit Test Projects.
* **MSTest**.**TestFramework**: Test Framework for testing native code. Native library that comes as part of the Visual Studio Unit Test Projects.

**Task 2**

* **ChroPath**: ChroPath is as a development tool to edit, inspect and generate XPath and CSS Selectors. ChroPath makes it easy to write, edit, extract and evaluate XPath and CSS queries on any webpage. It was mostly used to find XPaths for this project.
* **Microsoft.Extensions.Configuration**: Like other packages are packages that were created as part of the ASP.NET Core framework. The way ASP.NET Core and all its related packages were built however is in a very modular way, so all the libraries can be used within the ASP.NET Core context, or without.
* **Microsoft.Extensions.Configuration.FileExtensions**: Extension methods for configuring file-based configuration providers for Microsoft.Extensions.Configuration.
* **Microsoft.Extensions.Configuration.Json**: Provides support for reading JSON files
* **Selenium.Support:** Selenium is a set of different software tools each with a different approach. to supporting browser automation. These tools are highly flexible, allowing. many options for locating and manipulating elements within a browser, and one of its key features is the support for automating multiple browser platforms. This package contains .NET support utilites and classes that users may find useful in using Selenium WebDriver. It provides support for WebDriver.WaitExtensions that are used in this project .
* **Selenium.Webdriver**: WebDriver drives a browser natively, as a user would, either locally or on a remote machine using the Selenium server, marks a leap forward in terms of browser automation.
* **Selenium.Webdriver.ChromeDriver**: Provides direct support to Chrome. Without this package I was not able to execute tests using Chrome.
* **Coverlet.collector**: Coverlet is a cross platform code coverage library. NET, with support for line, branch and method coverage. Library that comes as part of the Visual Studio Unit Test Projects.
* **Microsoft**.**NET.Test.Sdk:** The MSbuild targets and properties for building .NET test projects. Library that comes as part of the Visual Studio Unit Test Projects.
* **MSTest**.**TestAdapter**: The adapter to discover and execute MSTest Framework based tests. Native library that comes as part of the Visual Studio Unit Test Projects.
* **MSTest**.**TestFramework**: Test Framework for testing native code. Native library that comes as part of the Visual Studio Unit Test Projects.