

OVERVIEW OF THE TECHNICAL ASSIGNMENT

Shraddha Shah

Shahshraddha2@gmail.com

Technical Assignment

Created the Console application to implement the technical assignment. Details about the attached solution are described in this document.

Solution:

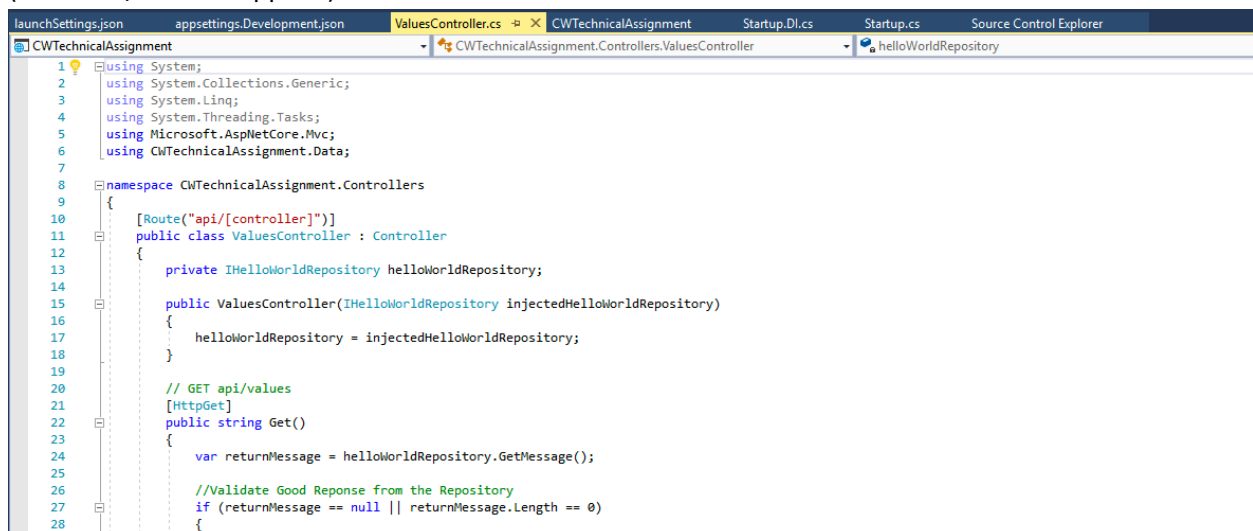
The solution has three projects.

- 1- CWTechnicalAssignment
- 2- CWTechnicalAssignmentTest
- 3- CWCommandLine

1. CWTechnicalAssignment

CWTechnicalAssignment.csproj is a **.net core web api** project.

This project has one exposed API, with two functions GET and POST. Get Method returns a string and POST Method takes string parameter to be saved in the backend storage. The POST method was added to meet the future requirement where data could be written to a backend storage of choice (database/Console app etc)



```
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Threading.Tasks;
5 using Microsoft.AspNetCore.Mvc;
6 using CWTechnicalAssignment.Data;
7
8 namespace CWTechnicalAssignment.Controllers
9 {
10     [Route("api/[controller]")]
11     public class ValuesController : Controller
12     {
13         private IHelloWorldRepository helloWorldRepository;
14
15         public ValuesController(IHelloWorldRepository injectedHelloWorldRepository)
16         {
17             helloWorldRepository = injectedHelloWorldRepository;
18         }
19
20         // GET api/values
21         [HttpGet]
22         public string Get()
23         {
24             var returnMessage = helloWorldRepository.GetMessage();
25
26             //Validate Good Reponse from the Repository
27             if (returnMessage == null || returnMessage.Length == 0)
28             {
```

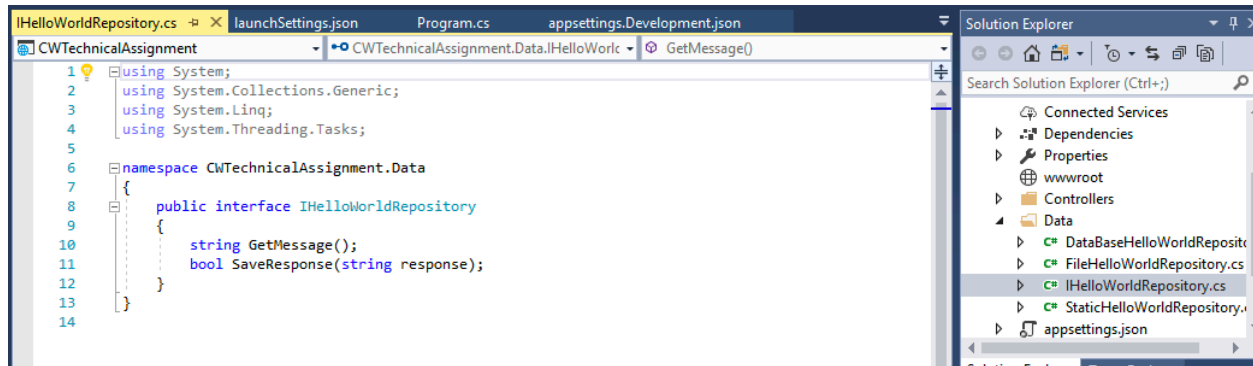
Dependency Injection

One of the requirement of the project for future extensibility was the ability to choose backend storage for saving or reading data, which I implemented using configuration file.

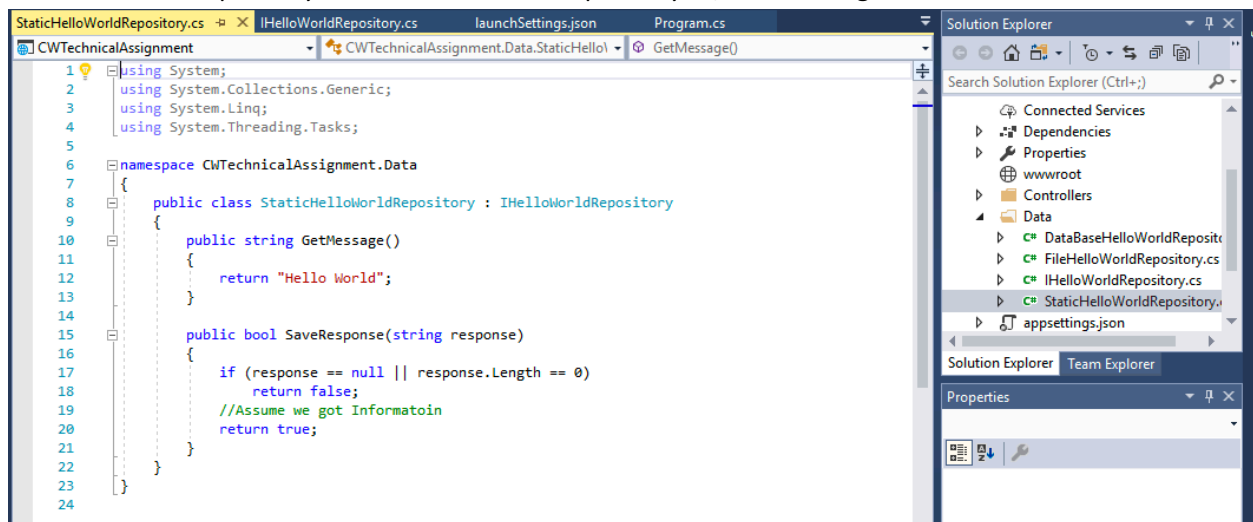
I achieved the goal using ASP.NET Core dependency injection as follows:

Repository

I created a repository Interface which defines what should be the interaction with the any data store. It is a following.



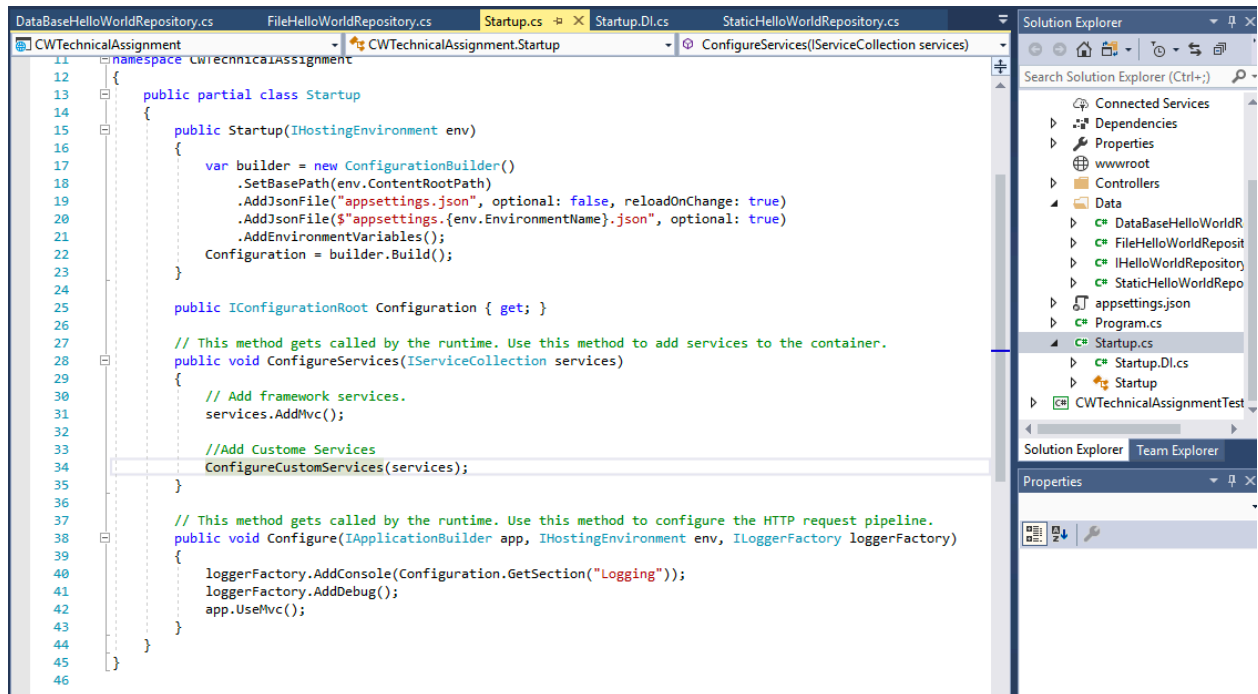
Then used this Repository in StaticHelloWorldRepository.cs, as following



Other implementations of this interface can also be implemented in future as needed. For example – FileHelloWorldRepository, DataBaseHelloWorldRepository.

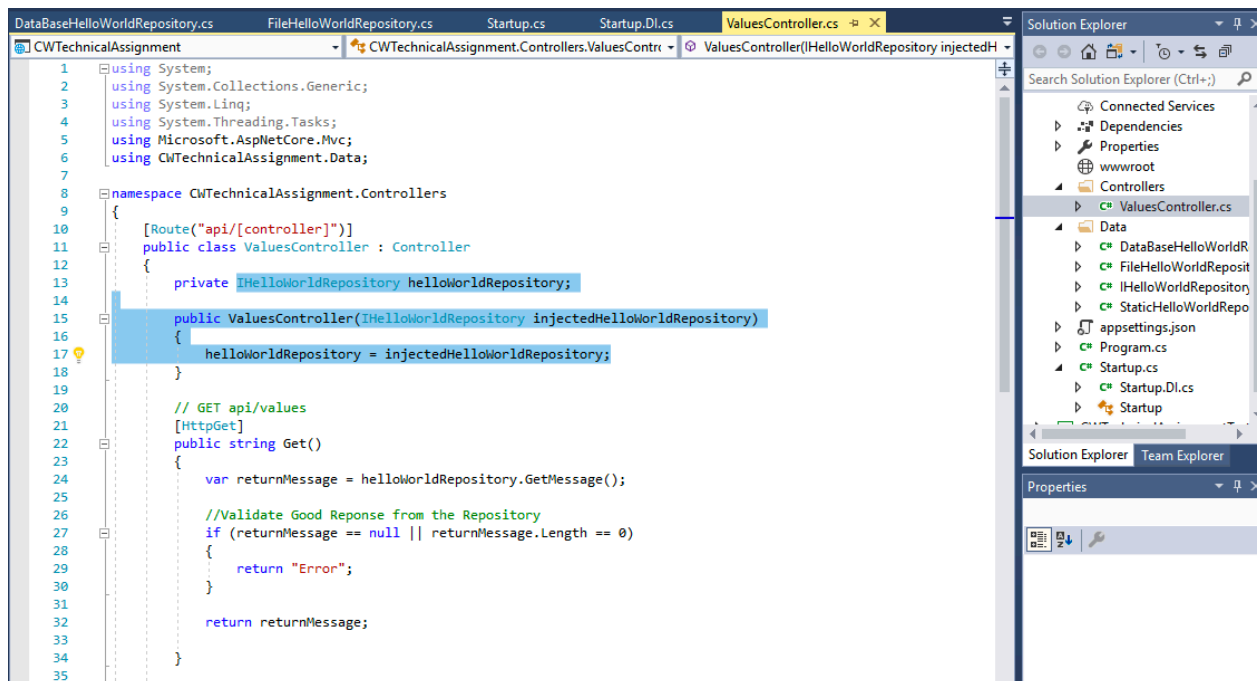
The Configuration

Inorder for application to use one of the repository, I have created startup.cs(ASP.Net core inbuilt DI). It also has partial class startup.DI.cs in the project. It has a function to define our Dependency Injection configuration i.e. to choose which repository to use (IHelloworldRepository). This function is called from the configureServices method in the startup.cs class as following.



Using the Repository

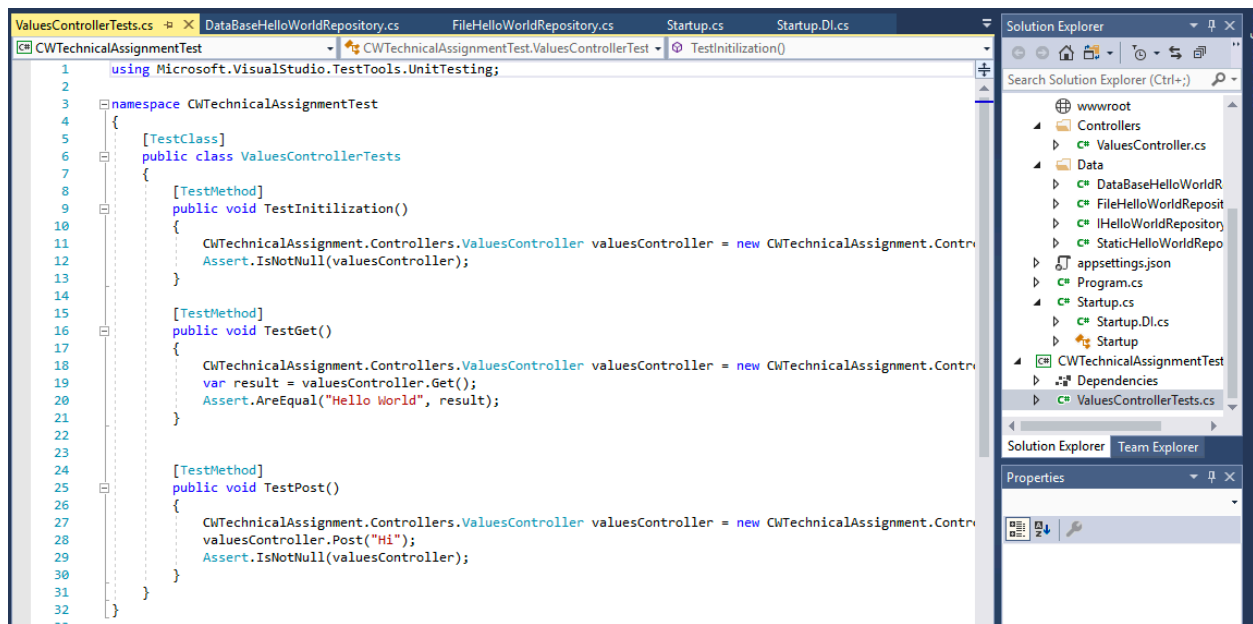
The final step in completing the Dependency Injection is using it in the controller tier. A public constructor in Values Controller (OUR Web API controller method) is added and got the implementation from ASP.NET core DI infrastructure based upon the configuration defined above as following:



The API controller then calls functions on the local helloworldRepository object of IHelloworldRepository interface. Based upon our configuration it could be Static File, Database, etc. The API controller is seamless of the approach.

2. CWTechnicalAssignmentTest

This is the project for unit test various functions of our Web API controller. Following are the tests:

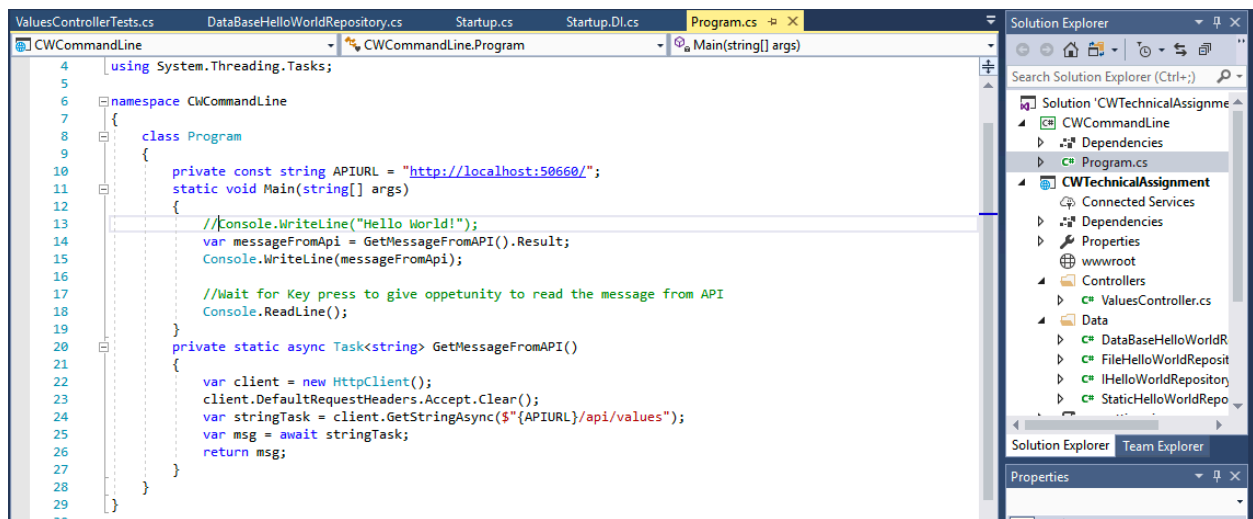


The screenshot displays the Visual Studio IDE with the 'CWTechnicalAssignmentTest' project selected. The main editor shows the 'ValuesControllerTests.cs' file, which contains three unit tests: 'TestInitialization()', 'TestGet()', and 'TestPost()'. The 'TestInitialization()' method creates a 'ValuesController' instance and asserts it is not null. The 'TestGet()' method calls 'Get()' on the controller and asserts the result is 'Hello World'. The 'TestPost()' method calls 'Post()' on the controller and asserts the result is not null. The Solution Explorer on the right shows the project structure, including 'Controllers', 'Data', 'Startup', and 'CWTechnicalAssignmentTest'.

```
1 using Microsoft.VisualStudio.TestTools.UnitTesting;
2
3 namespace CWTechnicalAssignmentTest
4 {
5     [TestClass]
6     public class ValuesControllerTests
7     {
8         [TestMethod]
9         public void TestInitialization()
10        {
11            CWTechnicalAssignment.Controllers.ValuesController valuesController = new CWTechnicalAssignment.Controllers.ValuesController();
12            Assert.IsNotNull(valuesController);
13        }
14
15        [TestMethod]
16        public void TestGet()
17        {
18            CWTechnicalAssignment.Controllers.ValuesController valuesController = new CWTechnicalAssignment.Controllers.ValuesController();
19            var result = valuesController.Get();
20            Assert.AreEqual("Hello World", result);
21        }
22
23        [TestMethod]
24        public void TestPost()
25        {
26            CWTechnicalAssignment.Controllers.ValuesController valuesController = new CWTechnicalAssignment.Controllers.ValuesController();
27            valuesController.Post("Hi");
28            Assert.IsNotNull(valuesController);
29        }
30    }
31 }
32
```

3. The Console Application

The third and final application is the console project. It is a simple asp.net console project which makes a call to our web api project and prints the output in a console window. Code is as following



Output is shown below:

