# Technical Exercise

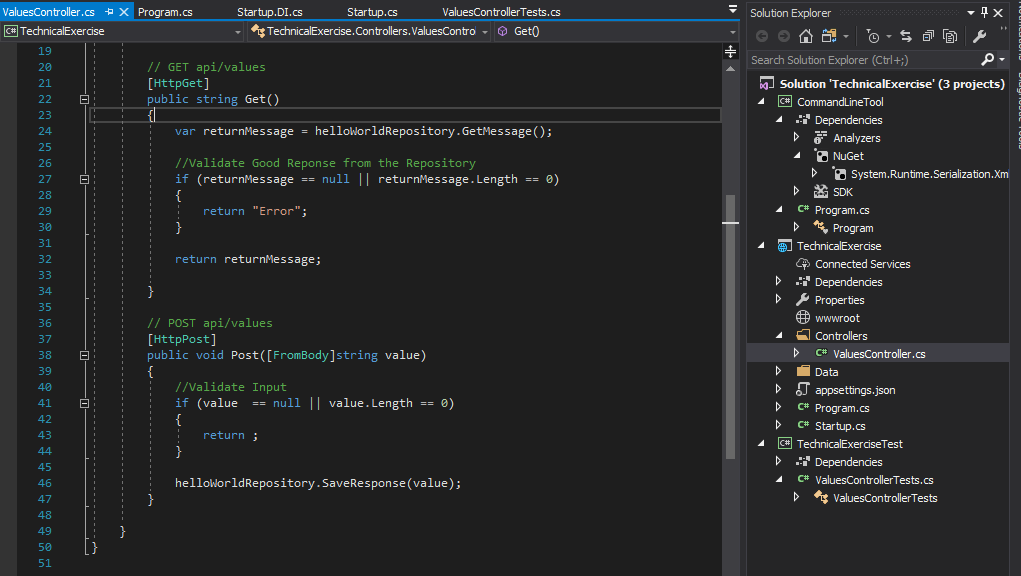
Following is some explanation about how I met the requirement of the technical exercise in the attached solution.

## Solution Projects:

The solution has three projects. Following is introduction and explanation of each.

### TechncalExercise

TechnicalExercise.csproj is the Web API project. It is a .net core web api project.

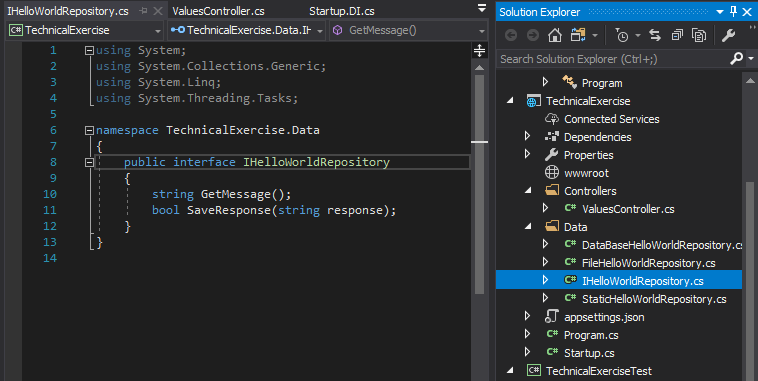
This project has one exposed API. With two functions GET and POST. Get returns a stiring and POST takes a string to save in the backing store. The POST method was added to meet the future requirement where data could be written to a backing store of choice (file/database etc)  


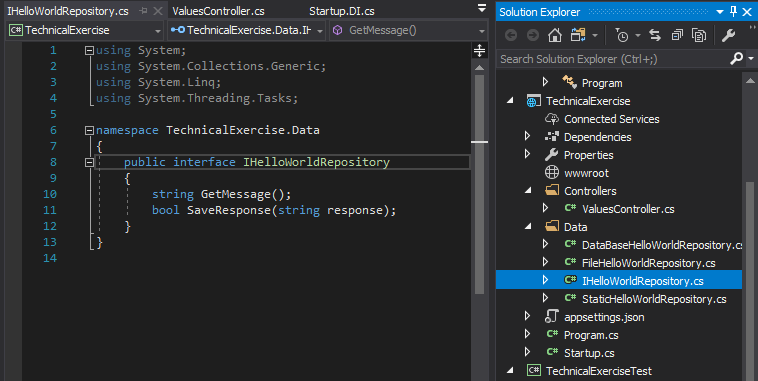
**Dependency Injection**

One of the requirement of the project for future extensibility was the ability to choose which backing store is used for saving or reading data. We need the ability to change the backing store using a configuration.

I achieved this goal using ASP.NET Core dependency injection as following:

#### Repository

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


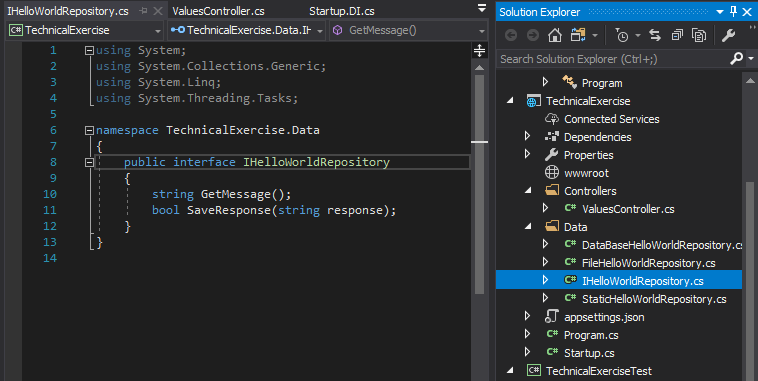
Then I implemented an implementation of this Repository Interface which we will use for the current requirement and named it StaticHelloWorldRepository.cs, as following  


As, you can see more implementations of this interface can be implemented in future as needed. For example – FileHelloWorldRepository, DataBaseHelloWorldRepository.

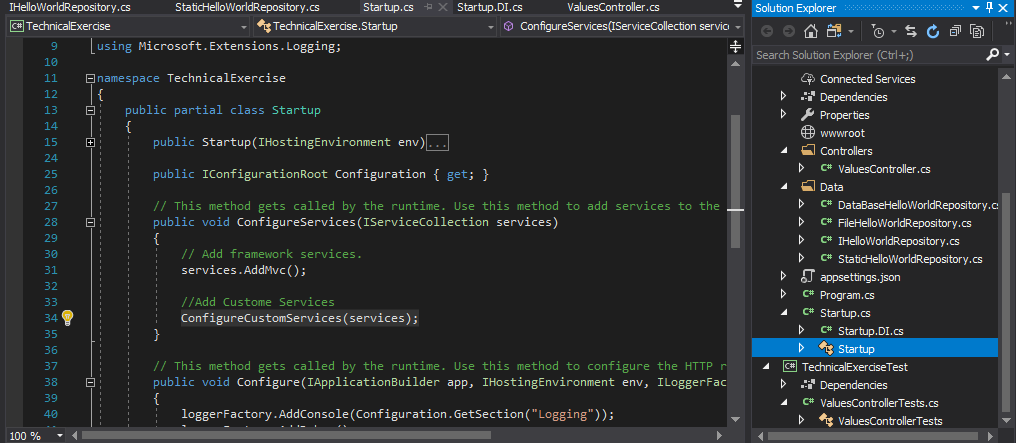
#### The Configuration

Now that we have different implementations of our repository interface IHelloWorldRepository, we need the ability to configure the application to use one or the other. I used ASP.Net core inbuilt DI infrastructure for this in the startup.cs.

I created a new partial class named startup.DI.cs in the project and added one function in it to define our Dependency Injection configuration. In this function we can choose which implementation of the IHelloWorldRepository will be used.

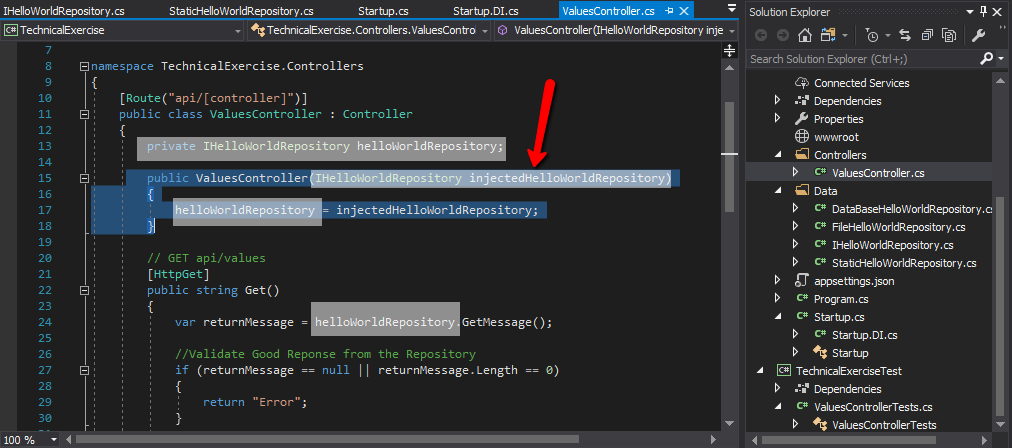


This function is called from the configureServices method in the startup.cs class as following.



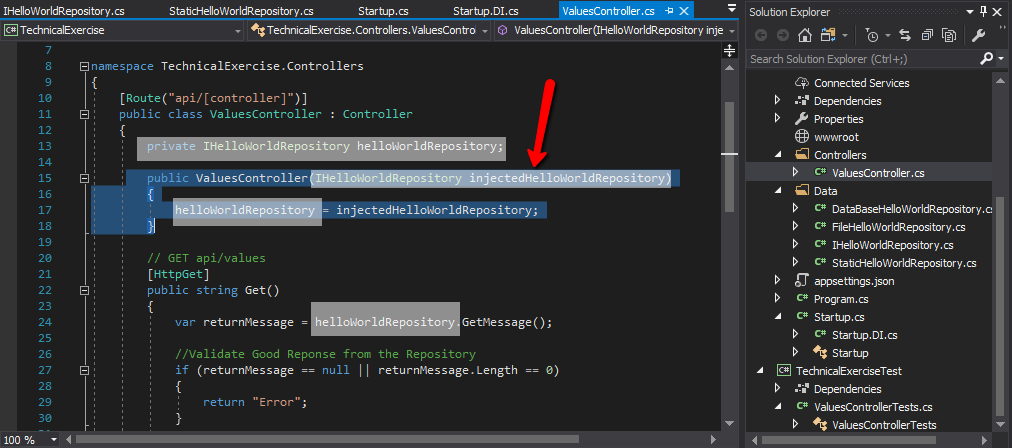
#### Using the Repository

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

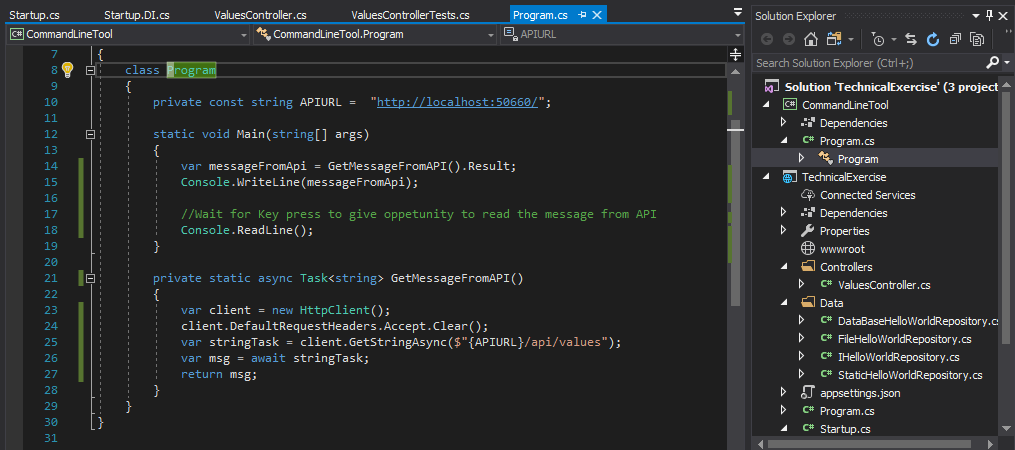


The API controller then calls functions on the local helloworldRepository object of IHelloWorldRepository interface without knowing what it really is. Based upon our configuration it could be StaticFile, Database, or even a test implementation. Our API controller does not know or care.

## Technical Exercise Test

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


## The Console Application

The third and final application is the console project which is the immediate requirement of the test 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. Following is the code to call the web api.  


Following is a screen shot of the web api and the console application showing the output:

