# SIS – SoftUni Information Services

SIS is a combination of a Web Server and a MVC Framework. Ultimately it is designed to mimic Microsoft’s IIS and ASP.NET Core. Following several Lab documents you will build all components of the SIS.

# SIS: MVC Framework – Advanced

Problems for exercises and homework for the [“C# Web Development Basics” course @ SoftUni](https://softuni.bg/courses/csharp-web-development-basics).

We will now extend the Framework, so that we can build dynamic and functional MVC Web Applications which will be hosted on the Handmade HTTP Server.

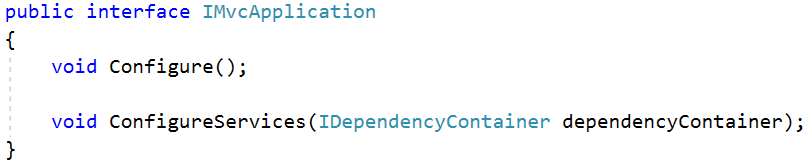
**NOTE**: Some functionalities will get removed, and new ones will be added on their place. This process is essential in development... Things get deprecated over time.

# WebHost & MVC Application

We’ve pretty much finished the our MVC Framework. There is nothing more add. Well there are a lot of things, but we cannot implement the next ASP.NET … That would take years to do. Anyway… We need to do several last things, before we proudly say we are finished.

## MVC Application

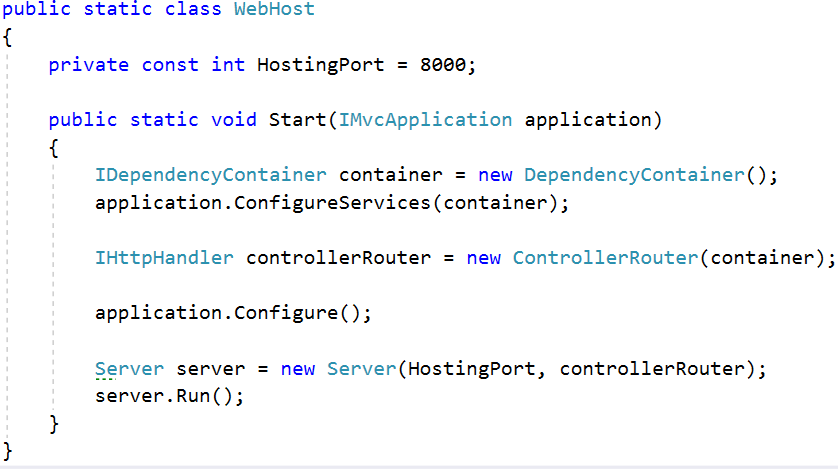
Create a namespace, called Api in the Framework project. Create an **interface**, called IMvcApplication and a **class** called MvcApplication, which implements the interface. The interface should look like this:



The class MvcApplication should only provide **virtual empty implementations** of the interface methods. Leave the implementations to the consumer.

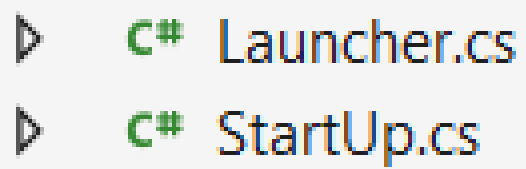
## WebHost

Create a **static** class, called WebHost, in the root of the Framework project. This class will encapsulate all the application initialization functionality and the Server consuming. By doing this we will finally make it so that our Framework consumes the Server. The class should look like this:

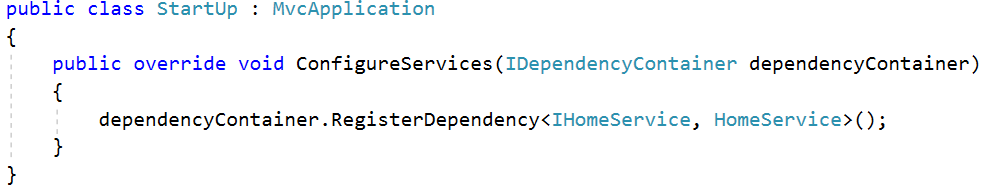


## Configuring Application

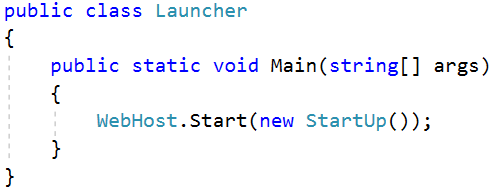
Your application should now have 2 main classes in the root of the project.



The StartUp.cs will be the main configuration – the Engine of the project, where all Services and Configurations are applied:



The Launcher.cs will just be the initiator of the application. It will invoke the Start() method of the WebHost class and pass an implementation of the StartUp class to it:



And with this our Framework and the way applications consume it, looks more and more like ASP.NET Core. Which was our target all along. We can easily say we are finished at this point but that would be quite a lie.

## Refactoring

During these trials of pain (where your fingers probably became practically senseless because of the thousands of lines of code you’ve written, and your head is probably hurting from all the information and concepts in the framework), we’ve written quite a lot of bugs and performance issues etc.

If you think that something can be optimized or refactored for the greater good, feel free to do so. There are a lot of things you can do. Just be sure to not change the way the Framework is consumed by the applications.

This is very optional (it is not mandatory for this homework) and very challenging but try to think of a way to deprecate and remove the MvcContext singleton.