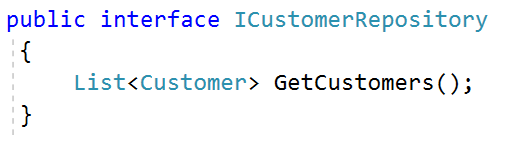
Dependency Injection in ASP.Net Core

* It’s a technique for achieving loose coupling between objects and their collaborators or dependencies.
* Rather than directly instantiating collaborators or using static references, the objects a class needs in order to perform its actions are provided to the class through some sort of constructor injection or in some other way.
* When classes are designed with DI in mind they are more loosely coupled because they do not have direct hard coded dependencies.
* DI Principal States that : High level modules should not depend on the low level modules both should depend on abstractions. So instead of ref specific implementations, classes request abstractions typically those are interfaces Which are provides to them when the class is constructed.
* When a System is designed to use DI, with many classes requesting their dependencies via their constructor, its helpful to have a class dedicated to creating these classes with their associated dependencies these classes are called as containers or IOC container or DI containers.
* Containers are factory that is responsible for providing instances of types that are requested from it.It not only create dependencies but also manage object lifetimes within the application.
* ASP.Net Core includes simple built in container represented by IServiceProvider interface that supports constructor injection by default. ASP.Net makes certain services available through DI and these container manages services.
* You configure built in containers services in the configureServices method in your application StartUp class.

Demo : How to Inject Services in ASP.Net MVC Core

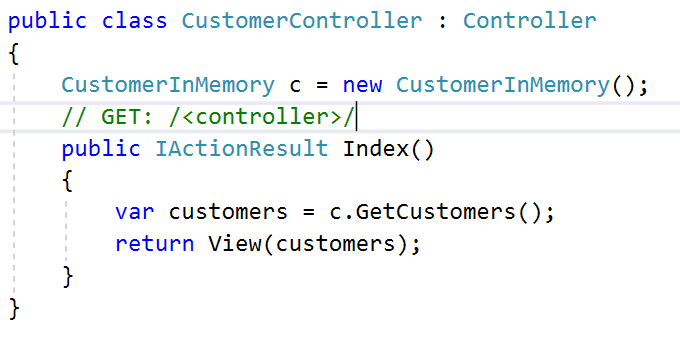
1. Create Empty ASP.Net Core Web application
2. Create ICustomerRepository



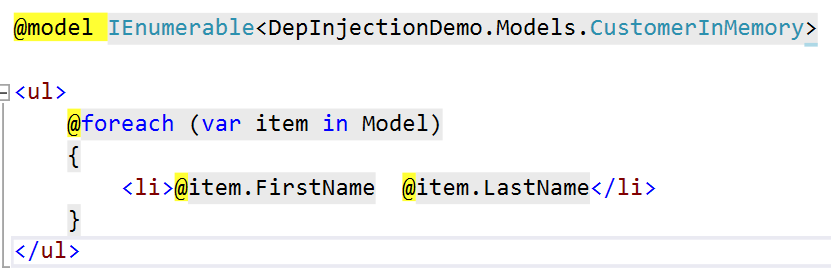
1. Create CustomerInMemory Class



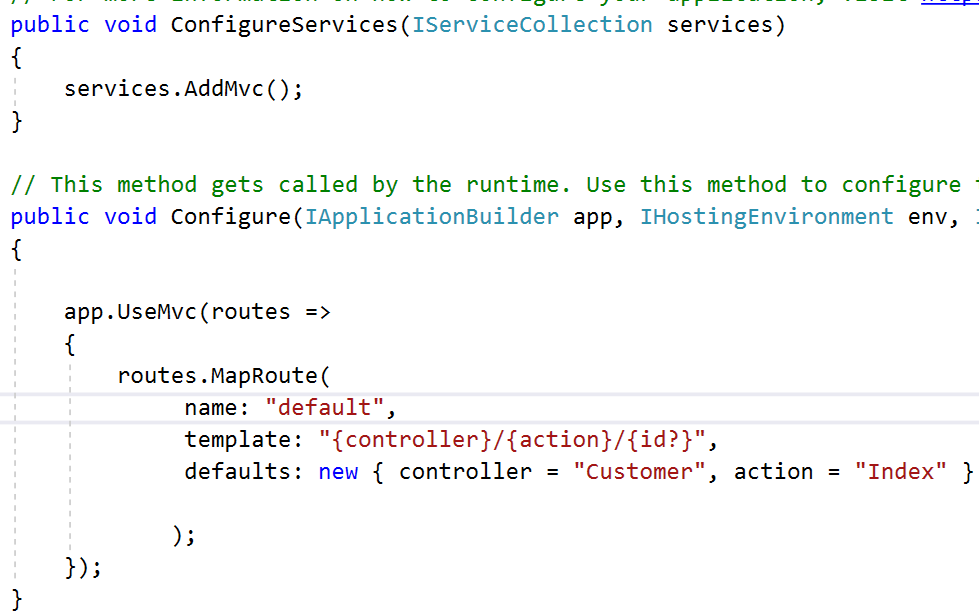
1. Now Create CustomerController and use this CustomerInmemory class



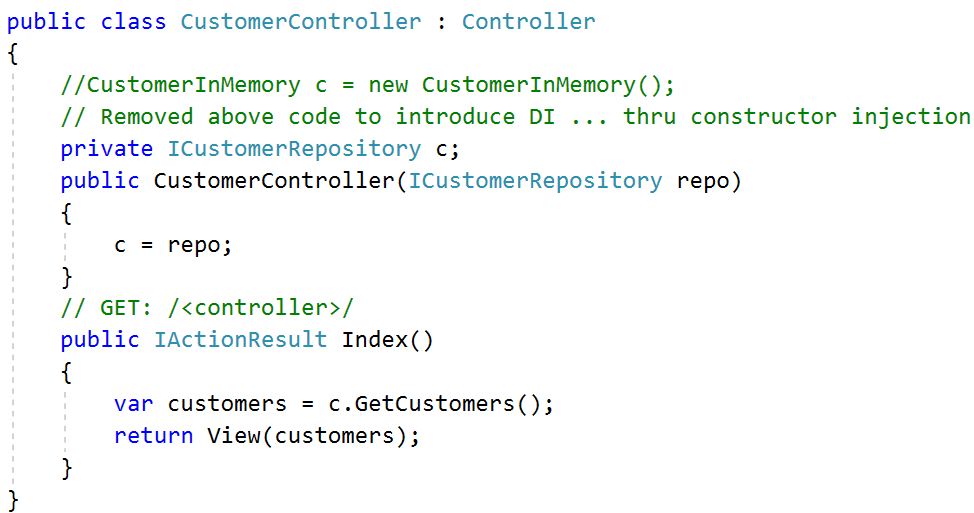
1. Create CustomerView.



1. StartUp class Add MVC Support and Routing Support .



1. Check the Output
2. Problem : if you check the controller code . its tightly coupled with Customer class. To avoid this we use Dependency Injection.
3. So instead of using Customer class use ICustomerRepository Interface



1. Check the Output.

Using Framework-Provided Services

* ConfigureServices method in the startup class is responsible for defining the services the application will use including features like EFcore and ASP.Net MVC.

Register your own Services

* You can register your won services but you have to pass first generic type that will be requested from the container the second generic type represents the concrete type that will be instantiated by the container and used to fulfil such requests.

Service Lifetimes and Registration Options

1. Transient

* Transient lifetime services are created each time they are requested .
* Useful for lightweight stateless services

1. Scoped

* Created once per request

1. Singleton

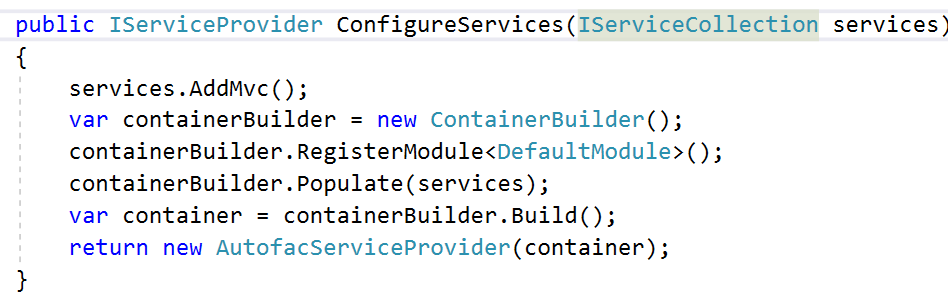
* Created the first time they are requested is run and subsequent request will use same instance.

Replacing the default Services container

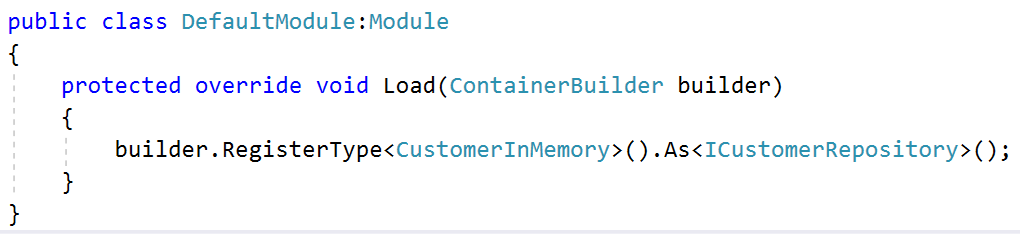
* Built in Services container is meant to serve basic needs of the framework and most applications built on it.
* Developer can replace this container with their preferred container.

Demo 2: How to use Autofac Container instead of built in container

1. Install the Autofac and Extensions.DependencyInjection
2. Configure Container in ConfigureServices and Return IServiceProvider



1. Configure Autofac as normal in DefaultModule , at runtime Autofac will be used to resolve types and inject dependencies.



1. See the Output.