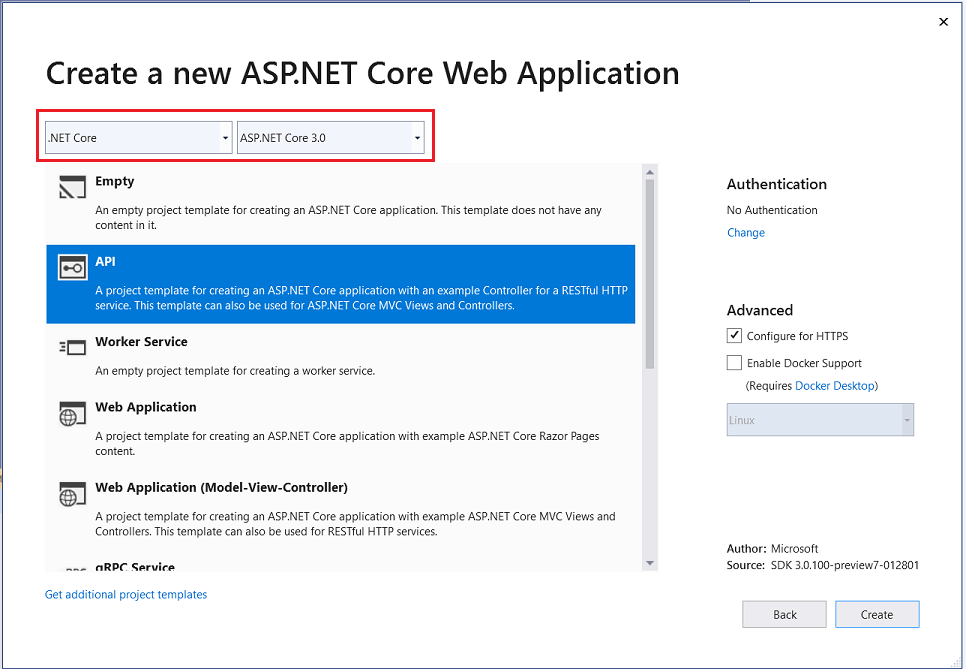
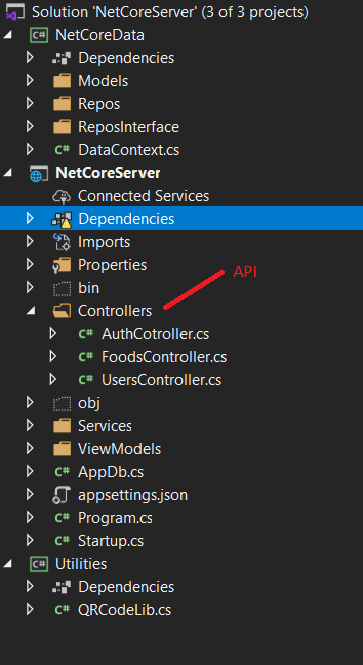
**Guideline setup ASP.Net Core API and EF Core**

**Create Web API project:**

* From the File menu, select New > Project.
* Select the ASP.NET Core Web Application template and click Next.
* Name the project … and click Create.
* In the Create a new ASP.NET Core Web Application dialog, confirm that .NET Core and ASP.NET Core 3.1 are selected. Select the API template and click Create.

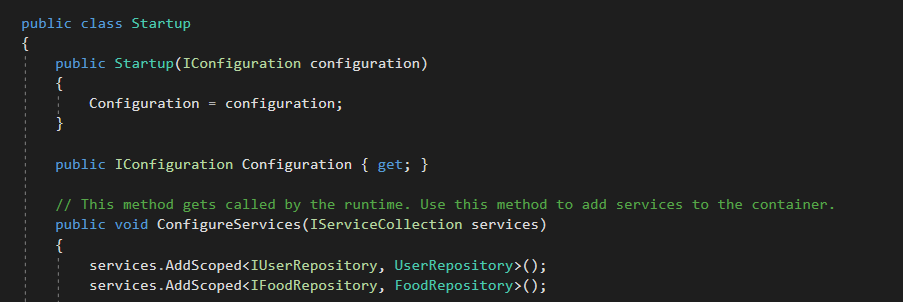


**Project structure:**

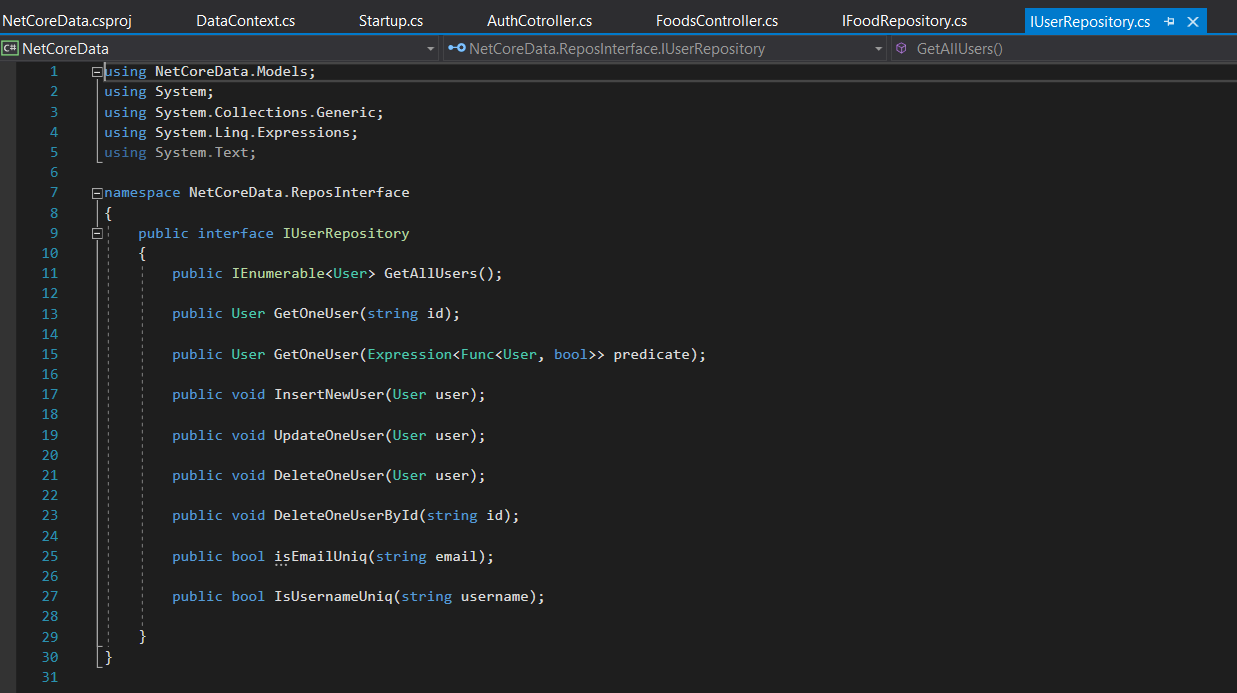


**Dependency Injection**

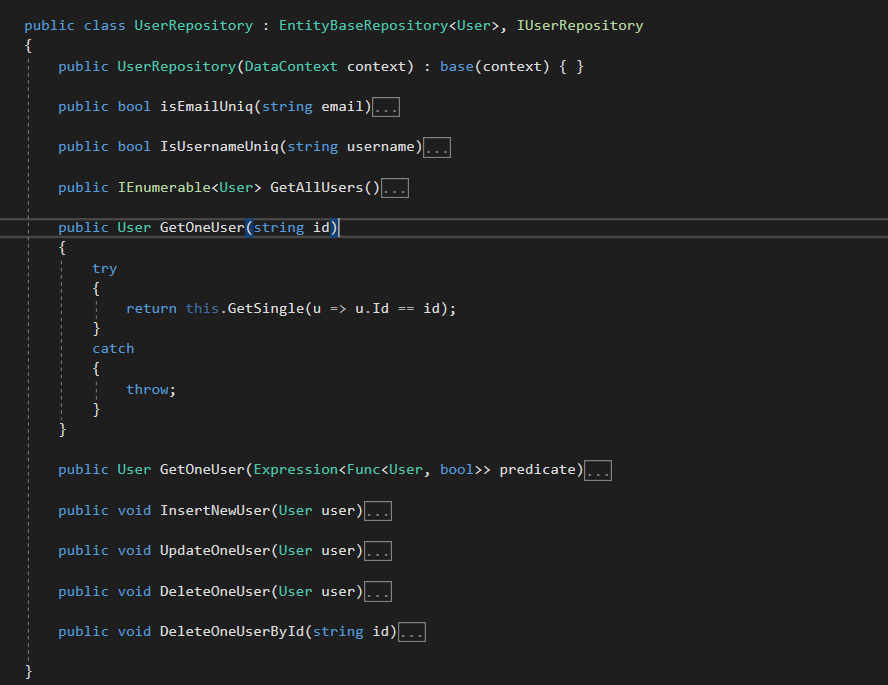
* In ASP.NET Core support dependency injection pattern. Registration of the dependency in a service container. ASP.NET Core provides a built-in service container, **IServiceProvider**. Services are registered in the app's **Startup.ConfigureServices** method.
* Injection of the service into the constructor of the class where it's used. The framework takes on the responsibility of creating an instance of the dependency and disposing of it when it's no longer needed.
* In our app, we have 2 services **IUserRepository**, **IFoodRepository** in **NetCoreData** library. The IUserRepository service is registered with the concrete type UserRepository and IFoodRepository service is registered with the concrete type FoodRepository.
* AddScoped is the service lifetimes.



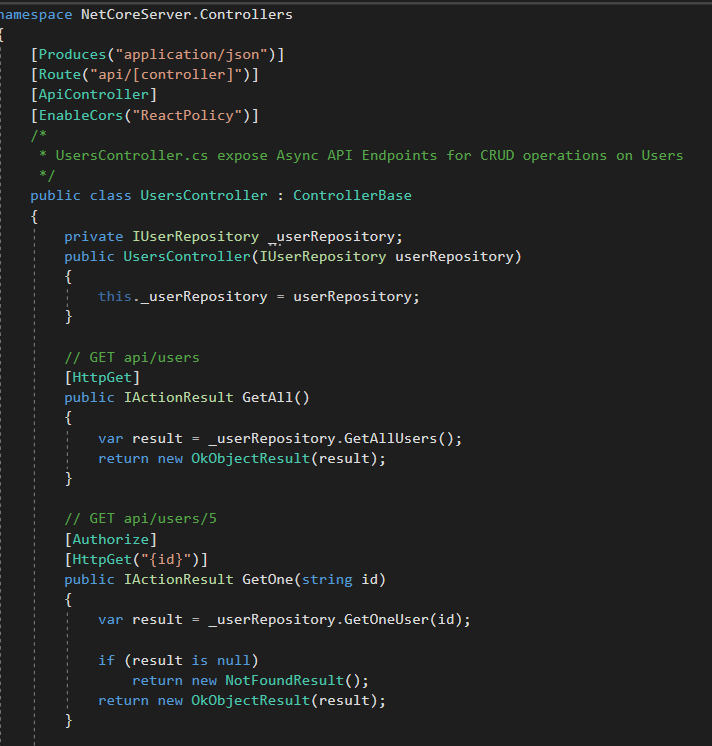
* **IUserRepository**, **IFoodRepository** are two interface to define function must implement. In this project we use Repository pattern to create an abstract layer through which the application communicates with the data stores.



* Then we create a class to implement the above interface and provides the implementation for the interface methods.



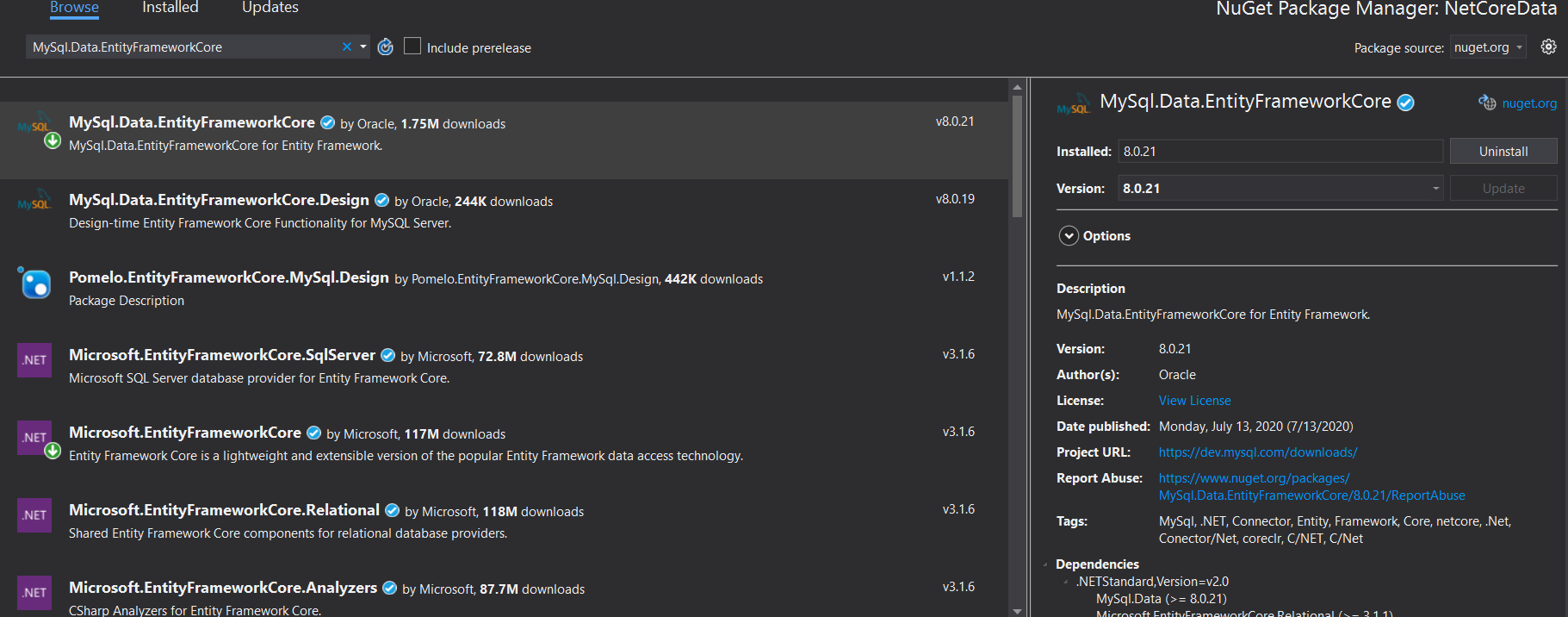
* An instance of the service is requested via the constructor of a class where the service is used and assigned to a private field. The field is used to access the service as necessary throughout the class.
* In **UsersController** API, the IUserRepository instance is requested and used to call the services’s GetAllUser, GetOneUser method



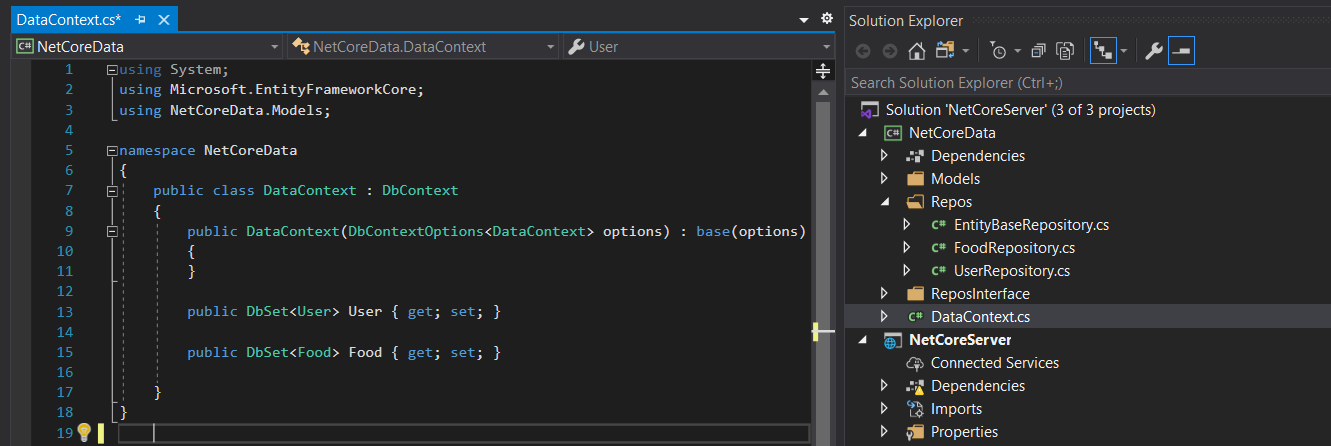
**Setup EF Core**

**Add a database context**

* The database context is the main class that coordinates Entity Framework functionality for a data model. Because we use MySQL database in the project so we need to install MySql.Data.EntityFrameworkCore and Microsoft.EntityFrameworkCore package
* From the Tools menu, select NuGet Package Manager > Manage NuGet Packages for Solution.
* Select the Browse tab, and then enter MySql.Data.EntityFrameworkCore in the search box.
* Select MySql.Data.EntityFrameworkCore in the left pane.
* Select the NetCoreData (this is library to interact with DB) check box in the right pane and then select Install.

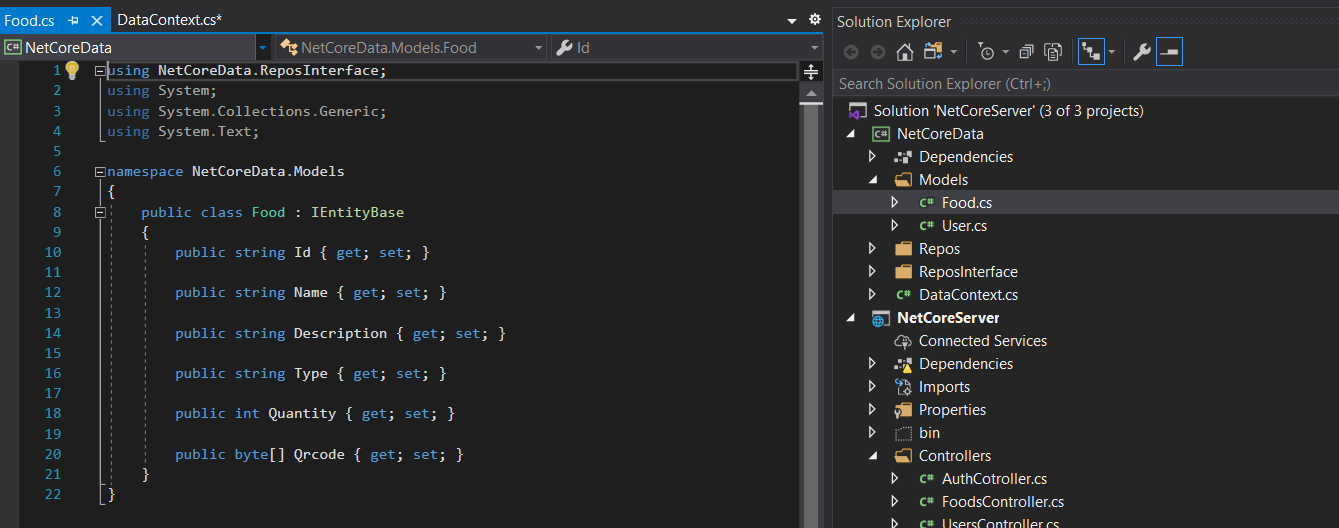


* Add DataContext class



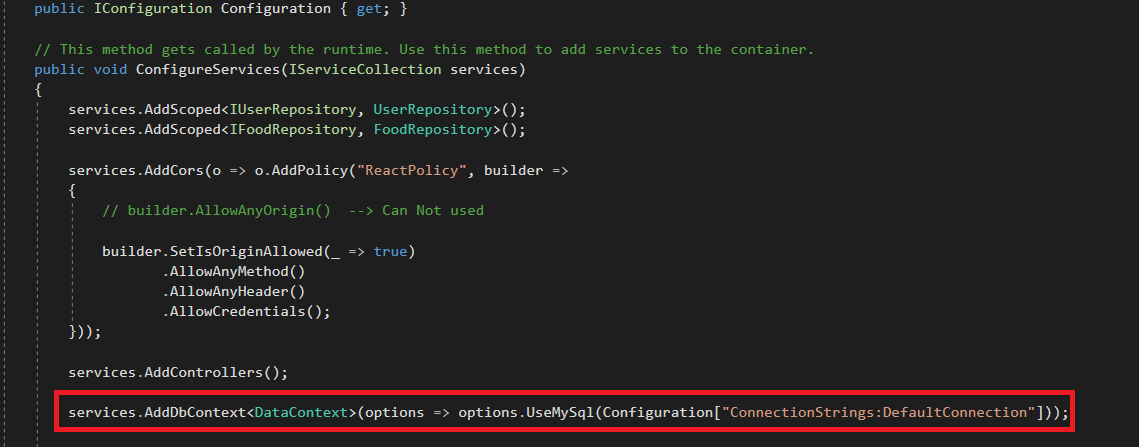
**Add model class**

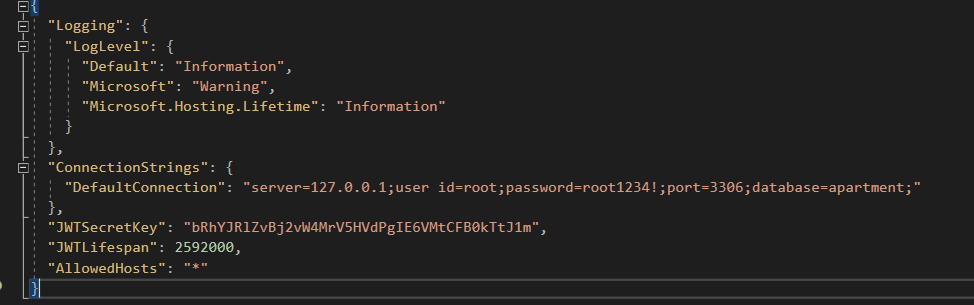
* A model is a set of classes that represent the data that the app manages. In this app we have 2 model are Food and User



**Register the database context**

* In ASP.NET Core, services such as the DB context must be registered with the dependency injection (DI) container. The container provides the service to controllers.
* Update Startup.cs file:



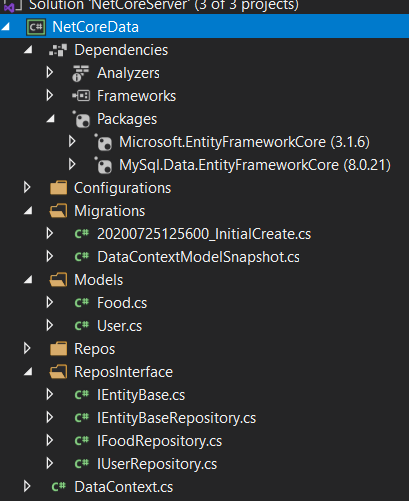
* We also need provide connection string for database connection in appsettings.json file
* 

**Add migration script**

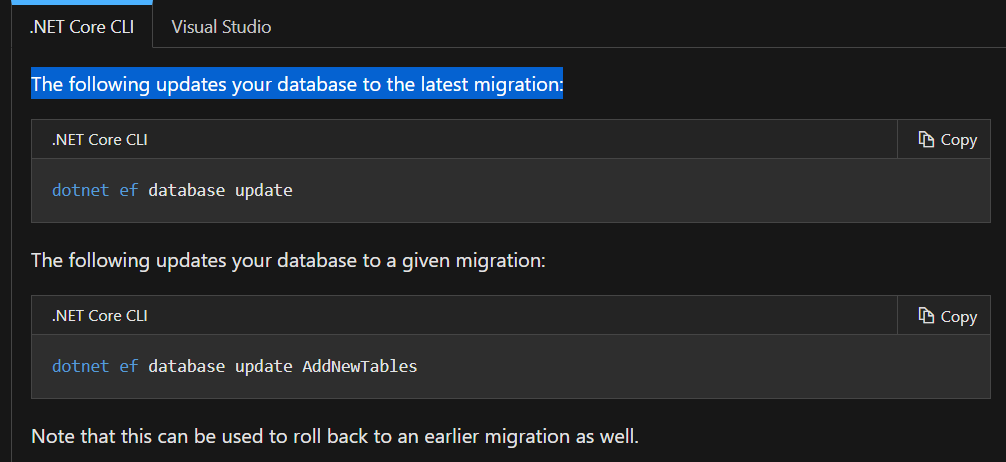
* First we need install Microsoft.EntityFrameworkCore.Design package to support generate migration script in startup project (web site project).
* Create first migration: run command
  + dotnet ef migrations add InitialCreate --project ..\NetCoreData

--project {directory contain data context}

After we run this command the system generate migrations file InitialCreate.cs automatically.



* Once your migrations have been added, they need to be deployed and applied to your databases.
* The EF command-line tools can be used to apply migrations to a database



But now in MySQL, it not support for creating MigrationHistory table so that we can store migration history version.

So we need to execute raw script in MySql

CREATE TABLE `\_\_EFMigrationsHistory` (

`MigrationId` varchar(150) NOT NULL,

`ProductVersion` varchar(32) NOT NULL,

PRIMARY KEY (`MigrationId`)

);

After that we can run above command successfully

dotnet ef database update