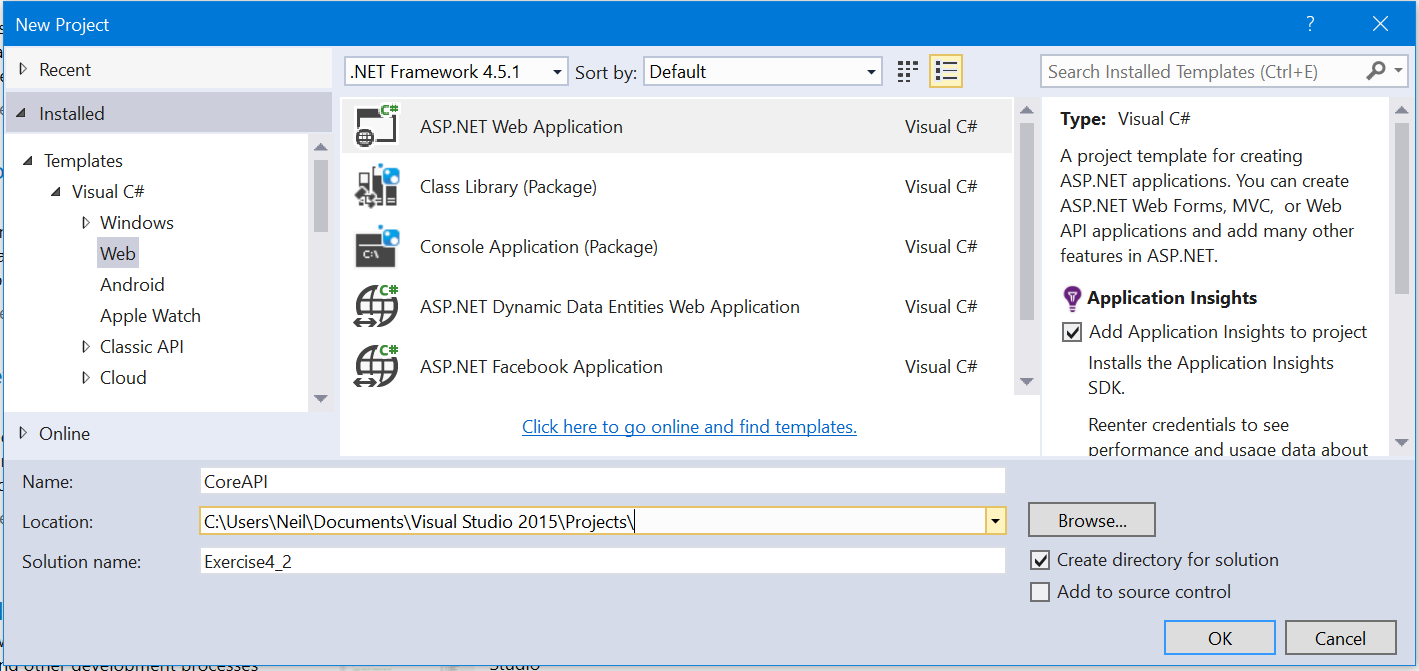
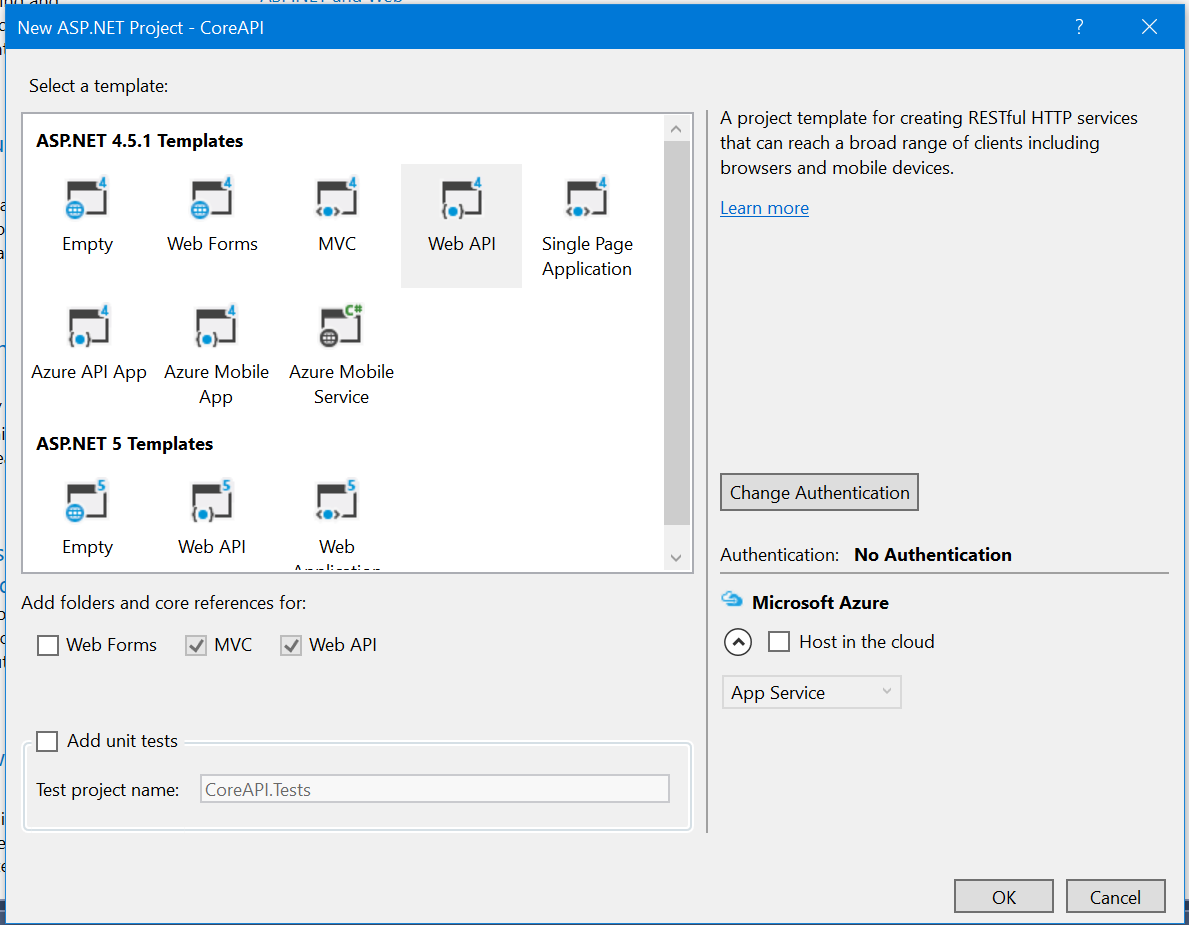
# Week 4.2 Sharing Model Accros CLients

# WeB API Project

* Create a new ASP.NET Application
* Take note that the Project and Solution names are not the same
  + Project is CoreAPI
  + Solution is Exercise 4\_2



* No Authentication for the Web API project

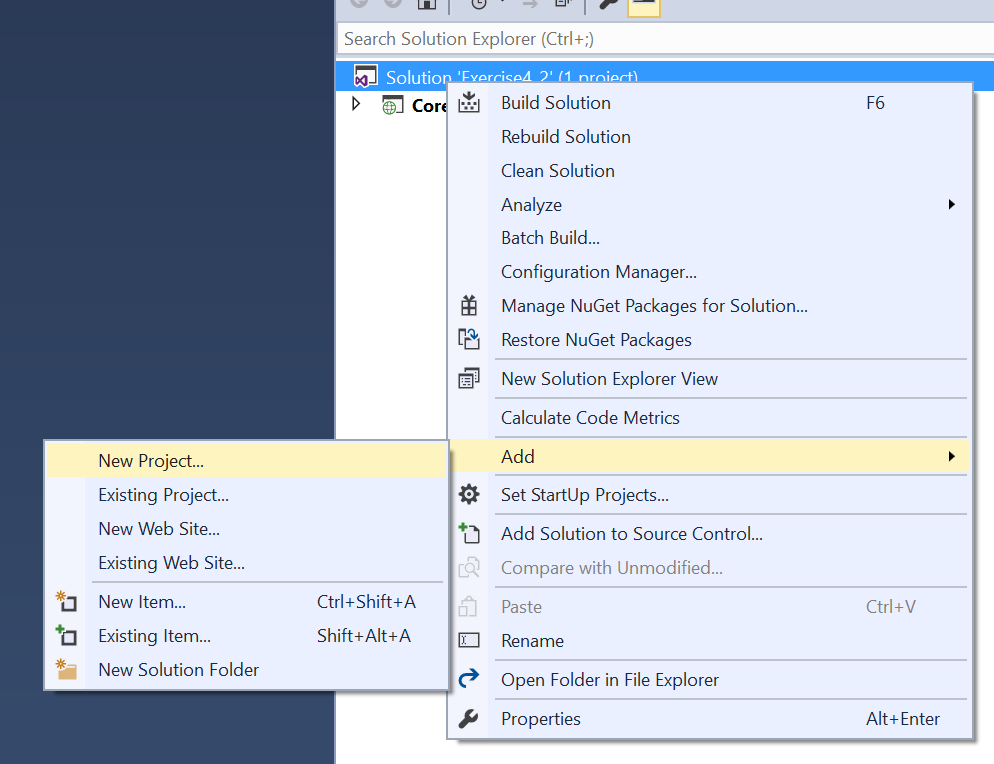


# Common CLass Library

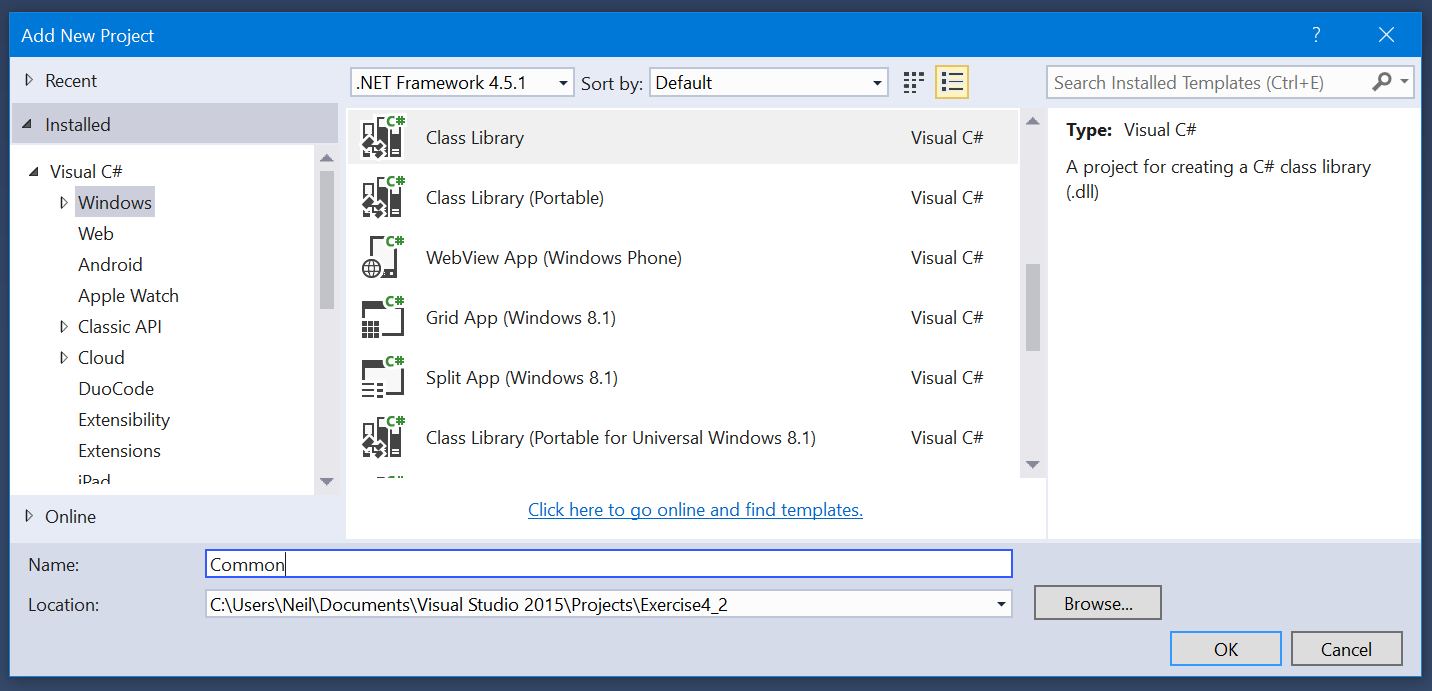
* The solution currently contains only a single project



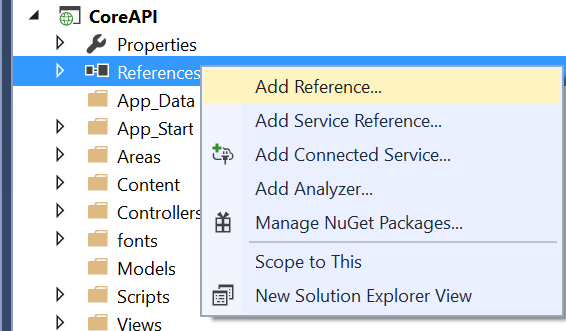
* Right click on the **solution**
  + Add -> New Project



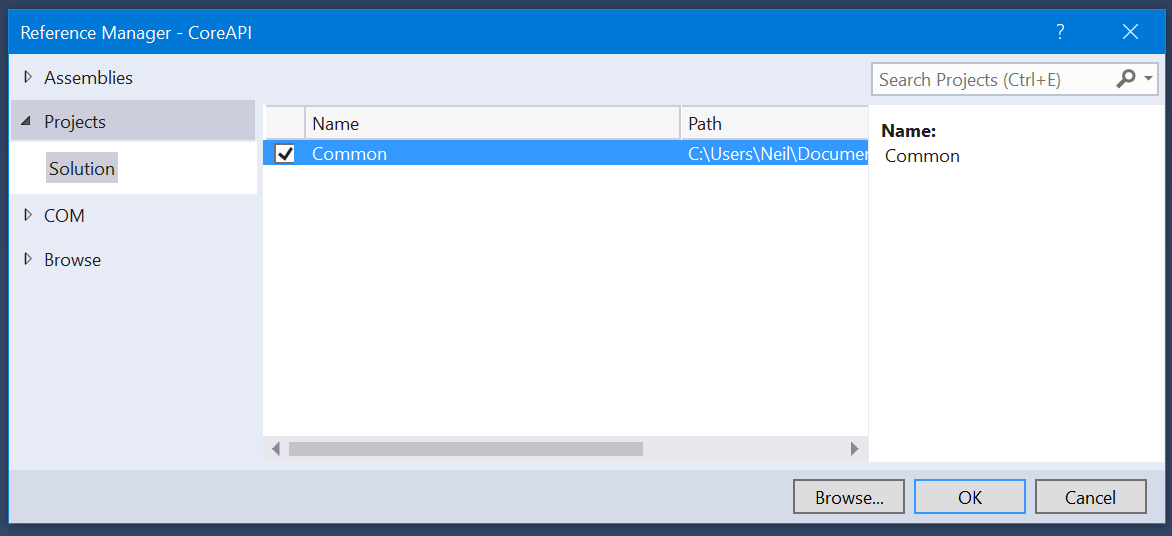
* Select the Windows category on the left
* Select the class library type
  + Not the portable class library
* Call this new project Common



* In the CoreAPI project
  + Right click References
  + Add Reference



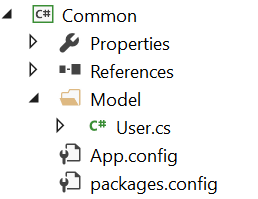
* Select the projects option in the dialog window if it’s not already selected
* Select the Common library
* All classes contained in the Common library are now available in the CoreAPI project
  + And any other project that adds a reference to it, including .NET clients (MVC, Desktop, Mobile)

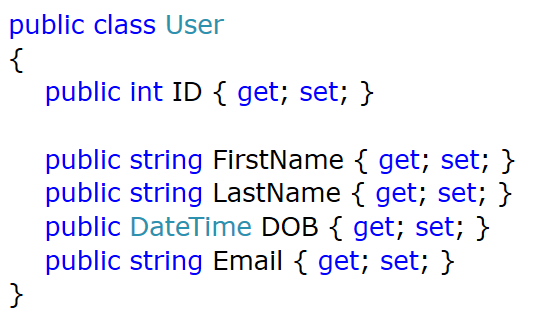


## Nuget

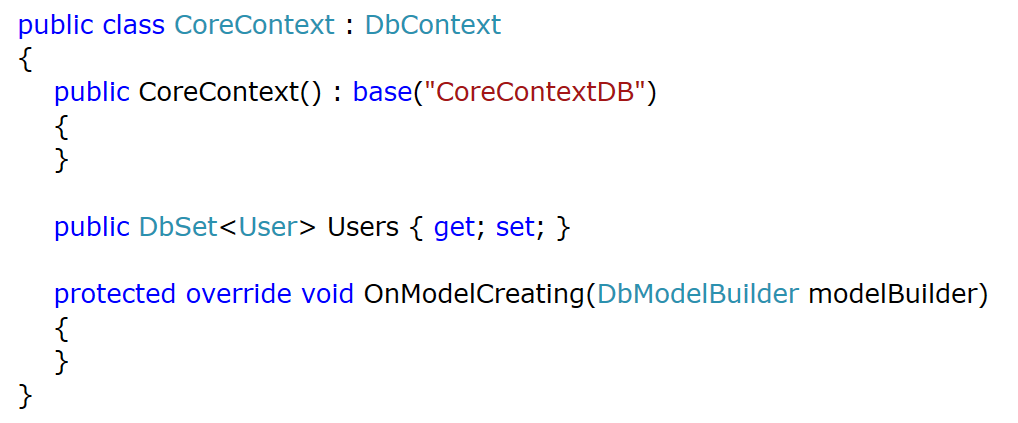
* In the Common project
  + Open Nuget
  + Install Entity Framework

## Model





## Context



## App.Config

* Add the following connection string to the App.Config file in the Common project

<connectionStrings>

<add

name="CoreContextDB"

connectionString="Data Source=(LocalDb)\MSSQLLocalDB;

AttachDbFilename=|DataDirectory|\CoreDB.mdf;

Initial Catalog=API-CoreDB;

Integrated Security=SSPI;

MultipleActiveResultSets=True"

providerName="System.Data.SqlClient" />

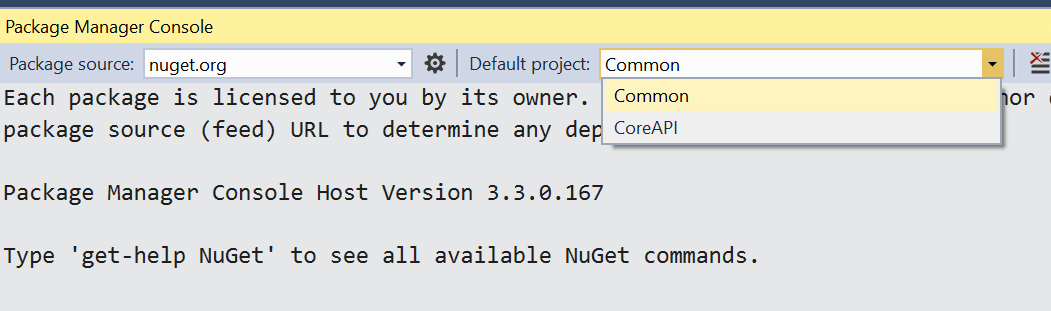
</connectionStrings>

## Web.Config

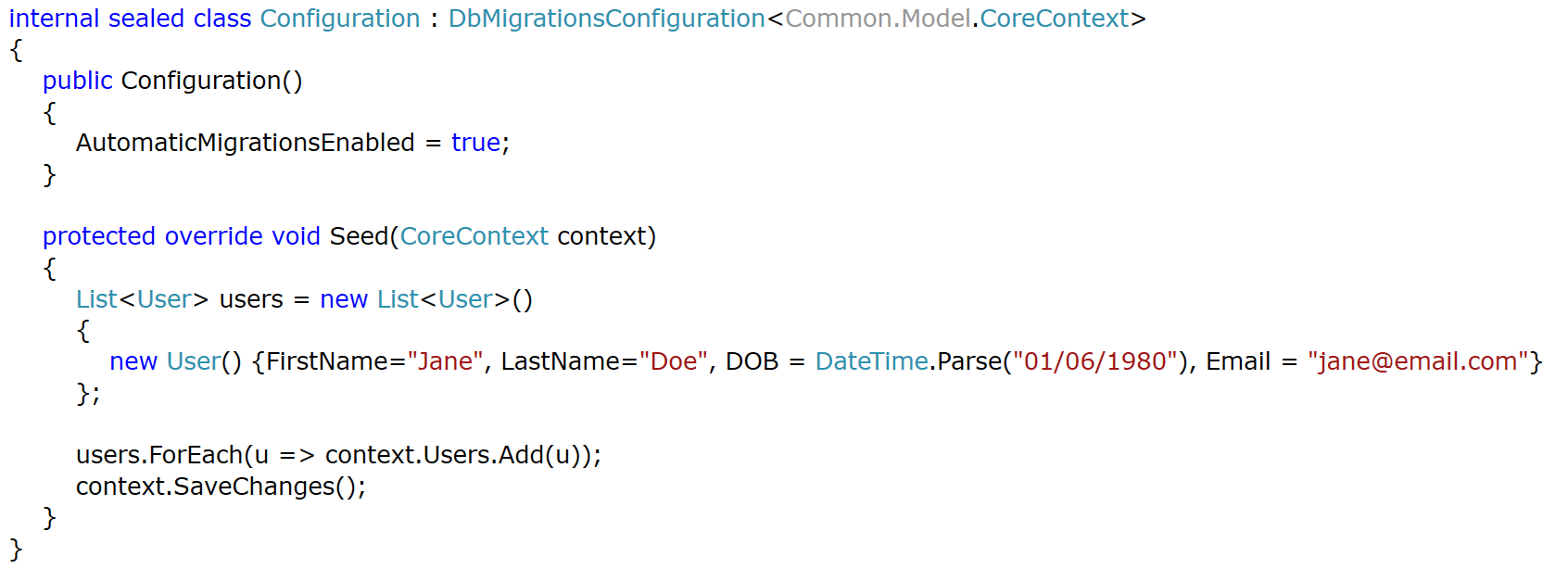
* Copy the exact same configuration settings into the web.config file of the CoreAPI project

## Enable Migrations

* Open the Packet Manager Console
  + You must execute the following commands for the Common project
  + Can be selected via the drop down list (see image)

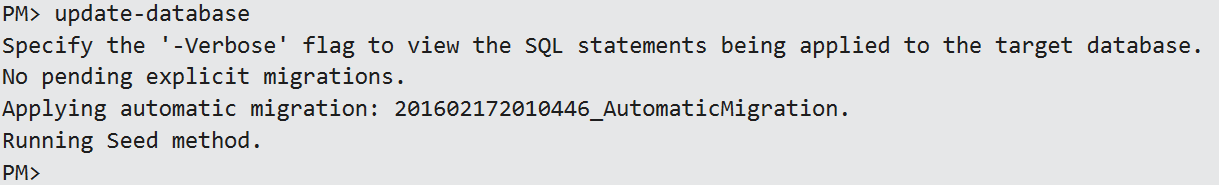


## Database Seed

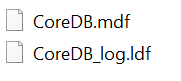


## Creating and Seeding Database

* Execute the update-database command

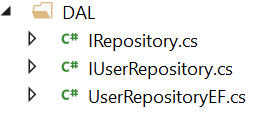


* Note that the database is created in the App\_Data folder of the CoreAPI project and not the class library
  + In a real world environment, we would not have a local database
  + The connection string would point to the actual source (online)
  + Therefore the DB would be created in the “incorrect” project



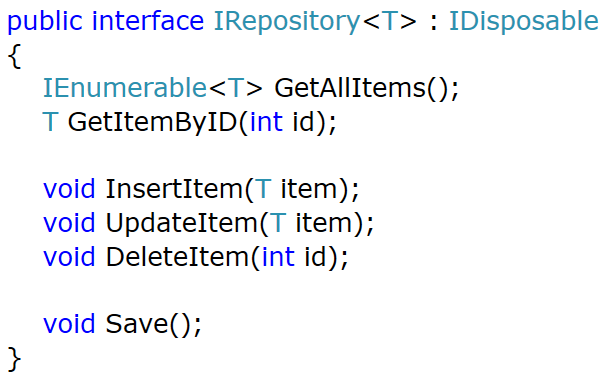
## Repositories

* Create the DAL folder in the Common project



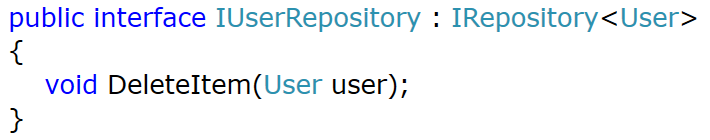
## IRepository

* Declare a generic interface that includes all the standard CRUD functions



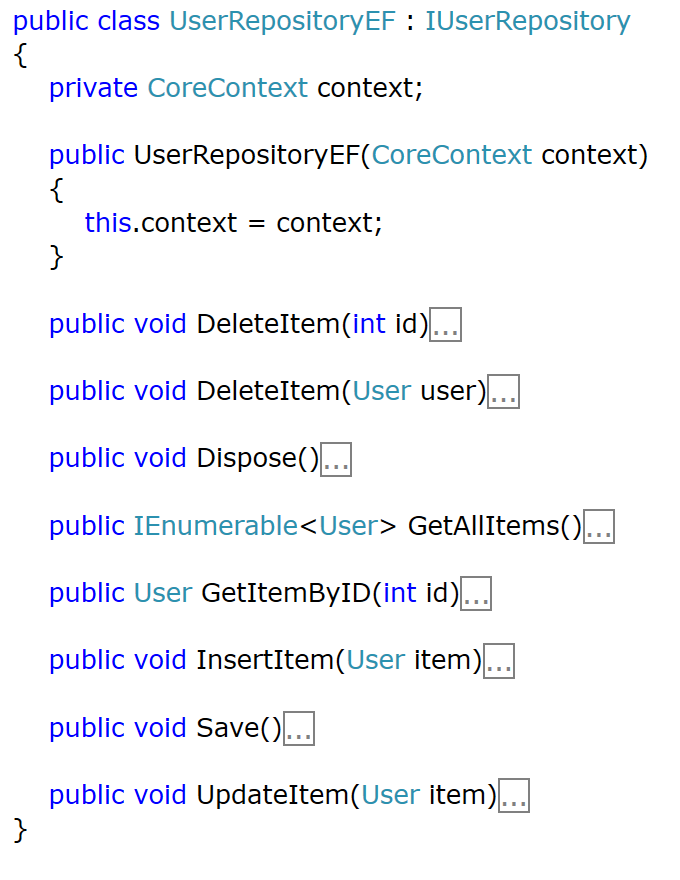
## IUserRepository

* Add a new member to the IUserRepository
* This will support the deletion of a user that maps perfectly onto the default delete logic of the upcoming API controller
  + This is an example of extending the generic repository interface to support custom functions



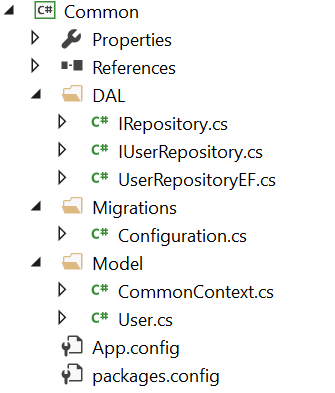
## User Repository EF

* Create a new class to implement the UserRespository
  + Look back at previous exercises for the implementation code
  + **You will need to know this going forward**



## Finished Library

* The completed Common library should now match the one shown below



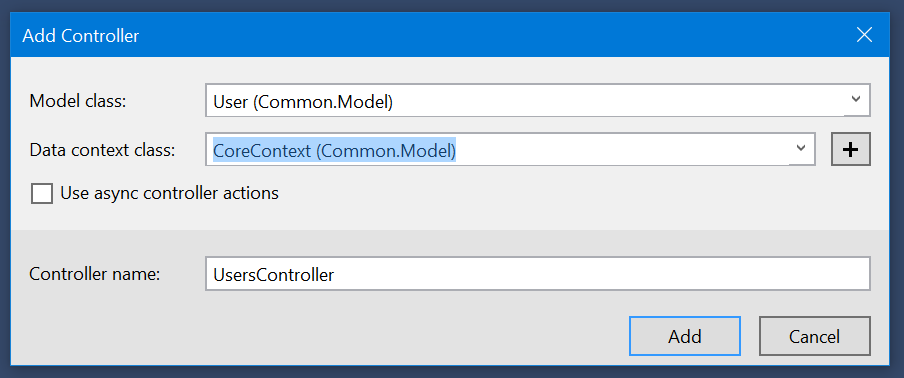
# Web API

## Controller

* Add a new Web API 2 controller to the CoreAPI project

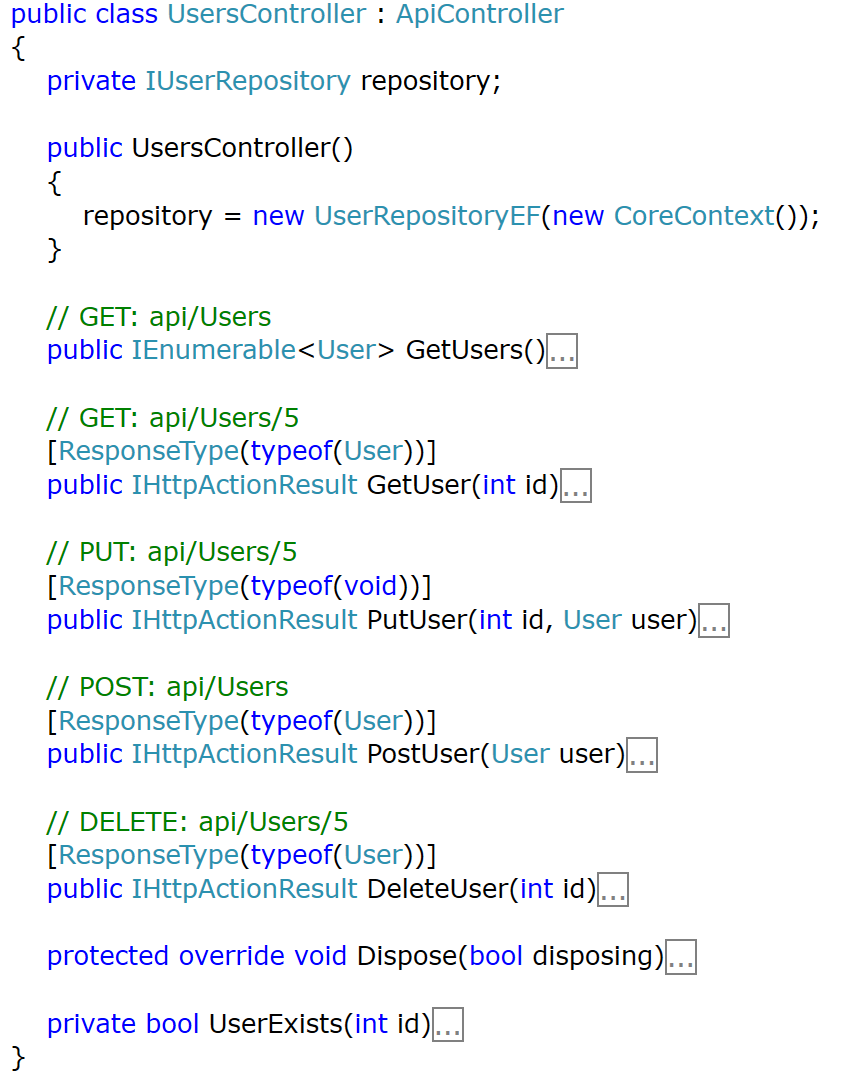


* Note how the Model and Context found in the library are fully available to us
* The API project has no knowledge of the actual implementation details

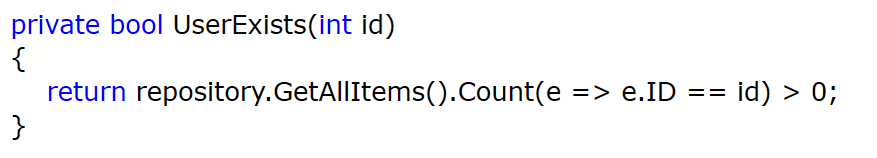


## Users Controller

* Edit the controller to use the repository instead of the default context provided

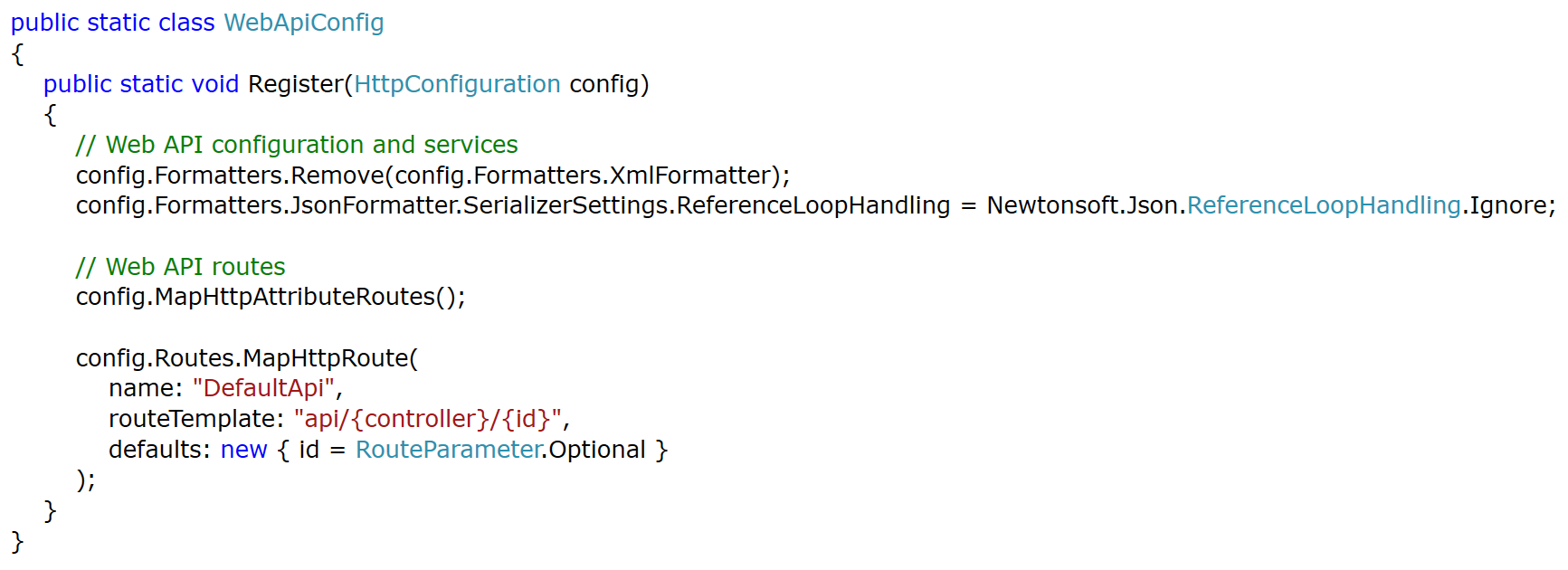


* Update the UserExists method to use our repository instead of the context
  + This method is using for the PutUser API action



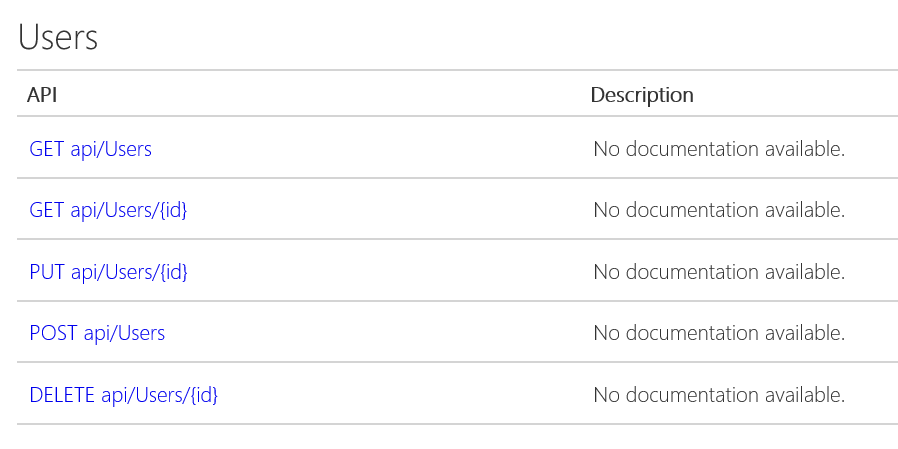
## API Configuration

* Finally update the WebApiConfig class to support only JSON
  + Note the ReferenceLoopHanding is also set to ignore
  + We only have a single table so this won’t affect us now but it’s good practice to implement this



## Test

* Run the application to test
  + The API section should no contain details on the UserController



* Calling api/Users should return the expected JSON

