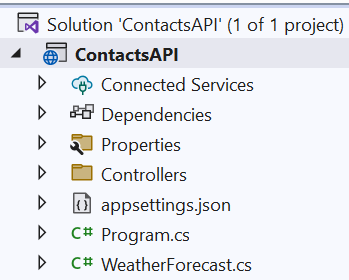
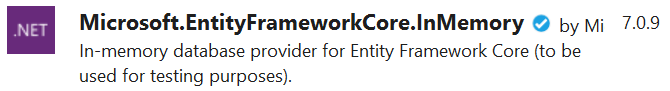
**ASP.NET Core Web API CRUD With Entity Framework**

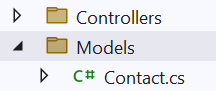
Create a new Web Api project

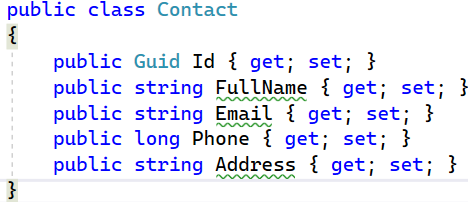


Install **Microsoft.EntityFramework.InMemory** package



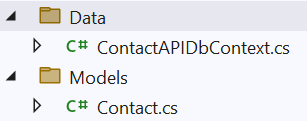
Now create **domain models** for our ContactsApi

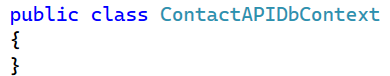




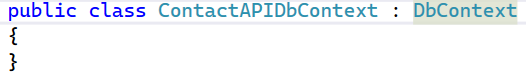
Now, create a class file that talk to our database, which acts like domain access layer

As we are using Entity Framework Core, we use the concept of **DbContext** , which acts as a class that talks to database



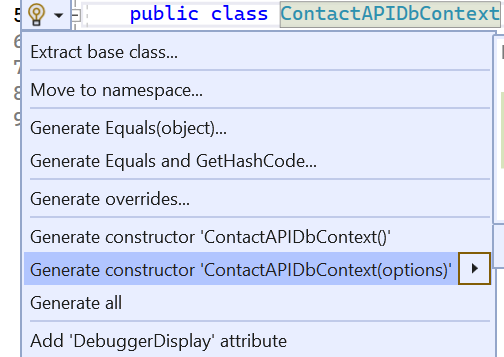


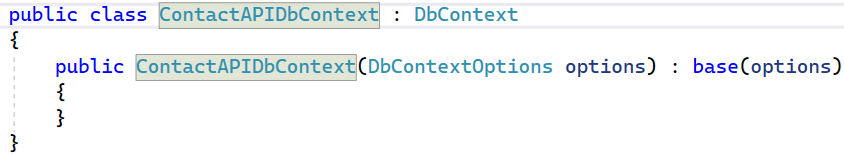
This class actually inherit from **DbContext**

****

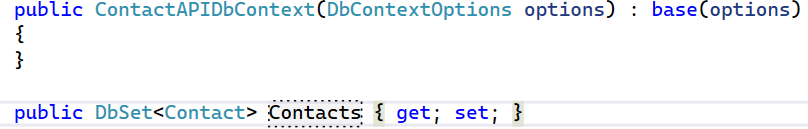
After that we need to create a constructor

Click **Ctrl+.** on the class and select **Generate constructor with options**

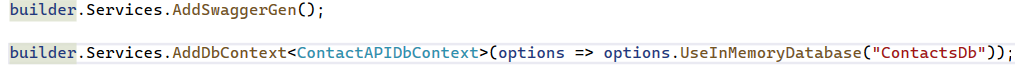




After that in the DbContext, we need to create properties that acts as tables for EF Core



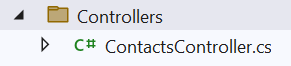
Its time to inject the **ContactsAPIDbContext** into the services of the solution

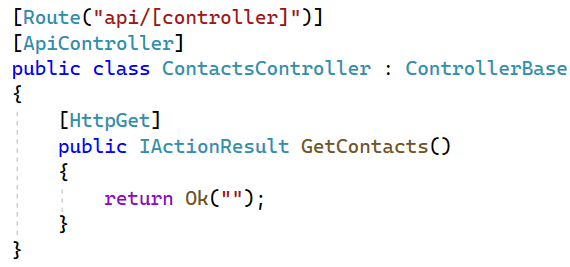


Here we are registering the DbContext and also mentioning to use in-memory database of name “ContactsDb”

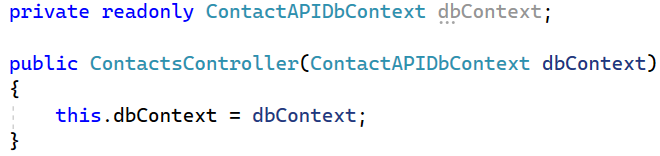
Now, we have EF everything it needs, like to an create in-memory database, and it also knows about the tables because we have given **DbSet** as well (**DbSet<Contact>**)

Now its time to create a Controller and inject this DbContext, so that we can read and write to in-memory database



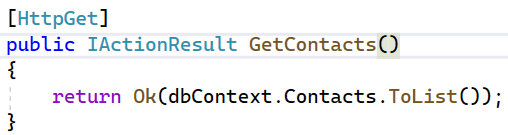


Now we want to inject the db context because we want to talk to our in-memory database



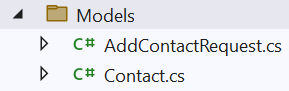
Now we can use this private readonly field **dbContext** to talk to our in-memory database

Now, go to GetContacts() method and use this dbContext field and put a dot to say talk to the **contacts table** because we have given the dbset property inside the context and it knows that contacts is a table in our database

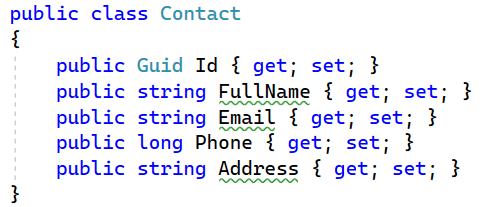


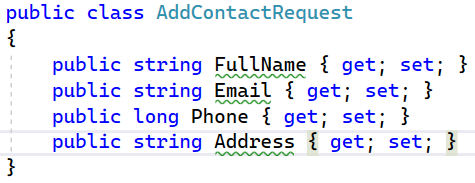
Now add a method to add contacts to our in-memory database

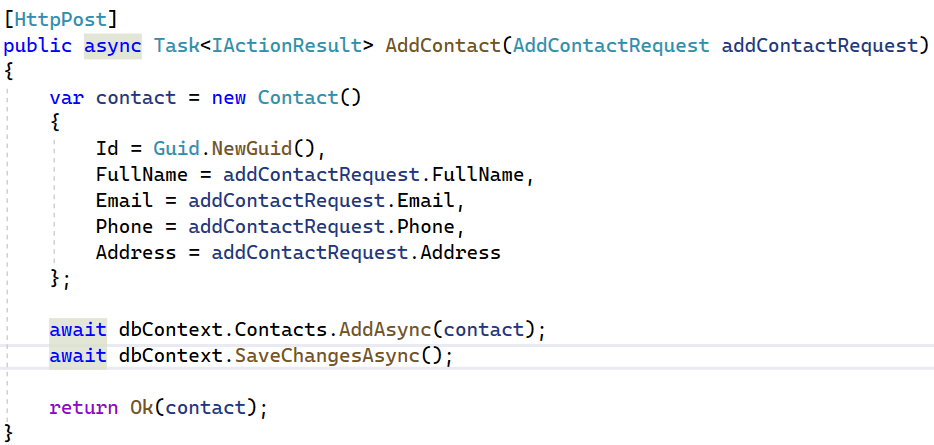
In here we will first have to get some values from the user. So, I will use another model and I will a create request model like create a class called **AddContactRequest**



We have created another model because we want to get only certain fields and we want to store it in the database and we look that Contact model we have five fields in total and we don’t to get the Id from the user and we want to give it our own id.

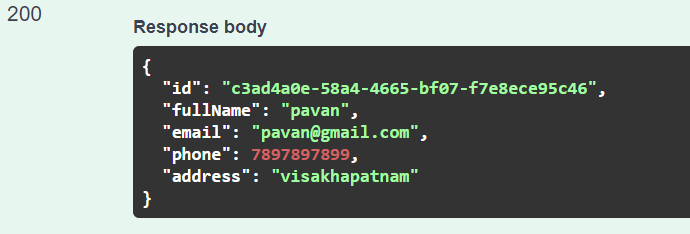




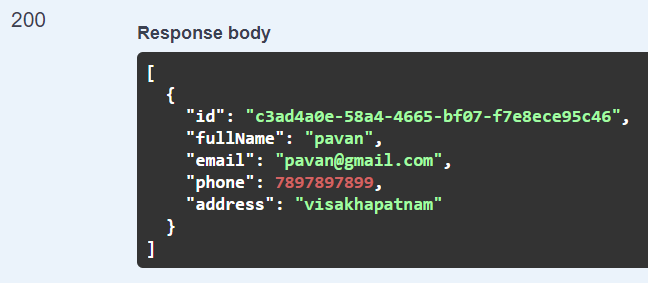


Now run and add a contact





Now check the **GetContacts** method

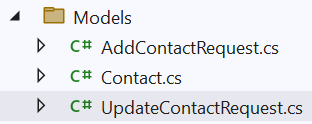


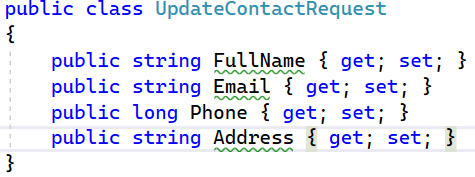
Keep in mind that because this is an in-memory database, the moment we close our application, the memory will be cleaned up and the database will have zero results

Now, its time add Update Contact method in our controller



We also want to know the updated fields. So, we need a new request object (class => UpdateContactRequest)





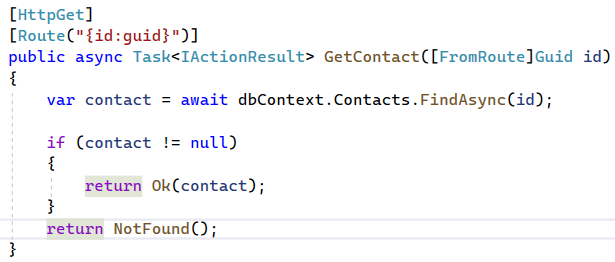
Here mention the properties which we want to update and here in our case we want to update all except id

Now the question may arise that because **UpdateContactRequest** is similar to **AddContactRequest** and why are we not using that?

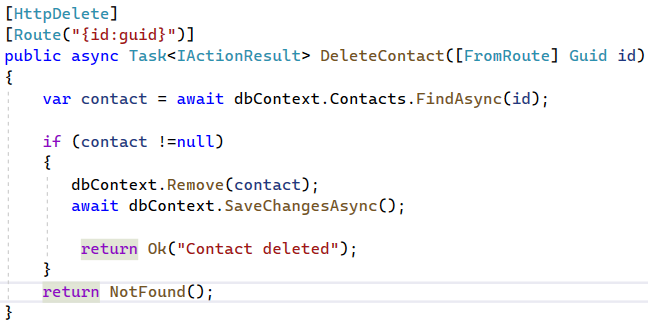
Because the use case is totally different and, in some cases, we want only few fields to updated then in that case the properties of Add and Update will be different. So, it is always better to have separate request objects



Now, we want add a method to get a single contact based on id



Now, add delete method

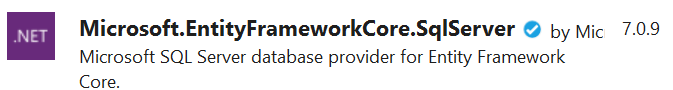


Now run and test all the action methods

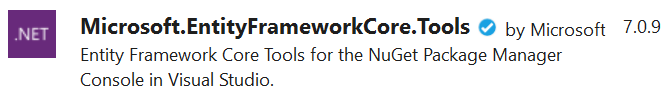
**Now Connect to Actual SQL Server Database and perform CRUD operations**

First, we need the install below three packages

**Microsoft.EntityFrameworkCore.Sqlserver**

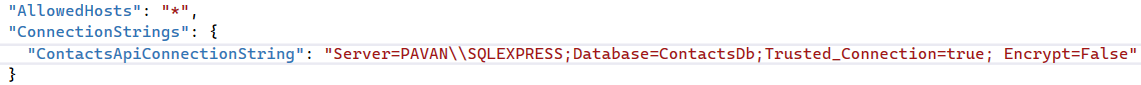
****

**Microsoft.EntityFrameworkCore.Tools**

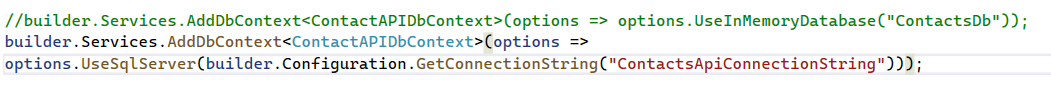
****

After the packages are installed, we need to **create a connection string**

We create connectionstrings inside **appsettings.json**



Its time to use this connection string, where we injected in-memory database

****

After that we don’t have to do anything. We just have to **create the sql server database using migrations**

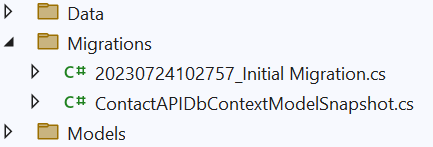
To start migrations, **Tools => NuGet Package Manager => Package Manager Console**

We type two commands

Command: **Add-Migration “migration name”**

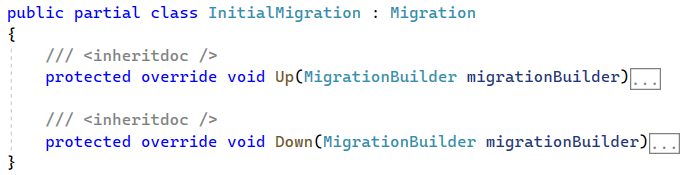
** and click on enter**

This is creating a migration for us, meaning a class file with all the code required to create a new database and will soon see a migration folder has been created



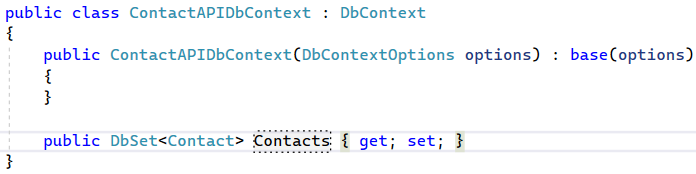
Inside that we have **Initial Migration** class file

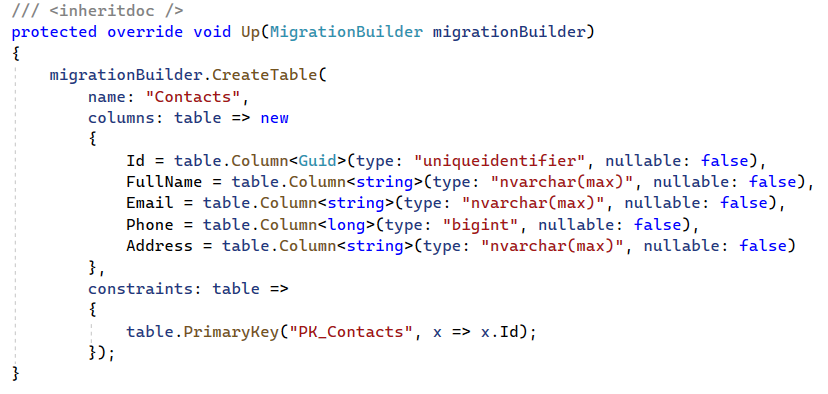
It has two methods,



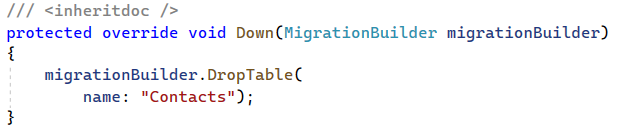
**Up method:**

It has everything it needs from the db context file (ContactAPIDbContext) to create a new database and to create new tables with all the properties required





**Down method:** contains code to drop the table



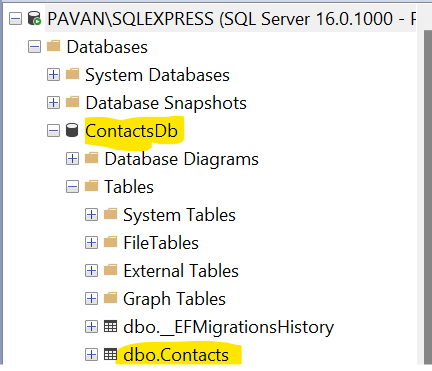
Now execute the second command

Command: **Update-Database**

** and click enter**

Update-Database will actually run the migration and creates database and tables in SQL server

Once command run completed, check in SSMS the created database and tables



Let’s run and test all the crud operations. I should be working as expected