**Hands-on .NetCore 3.0**

**ASP.Net Core MVC App**

Table of Contents

[Scaffolding Console App Basics 3](#_Toc24896659)

[What is MVC? 3](#_Toc24896660)

[Planning our App 3](#_Toc24896661)

[Requirements 3](#_Toc24896662)

[Creating ASP.Net MVC Core App using CLI 4](#_Toc24896663)

[MS SQL Database 5](#_Toc24896664)

[Working with MS SQL 5](#_Toc24896665)

[Connecting to the Database 5](#_Toc24896666)

[Creating a Database 5](#_Toc24896667)

[Creating the Model 7](#_Toc24896668)

[Entity Framework Core 3.0: Connecting to the Database 8](#_Toc24896669)

[Installing the Entity Framework Core 3.0 8](#_Toc24896670)

[Create a Context 9](#_Toc24896671)

[Add Connection String 10](#_Toc24896672)

[Adding AutoGenerateBindingRedirect 10](#_Toc24896673)

[Using DI to Read the Connections String 11](#_Toc24896674)

[Add DBContext to Startup 12](#_Toc24896675)

[Alternate to ConfigurationSettings 12](#_Toc24896676)

[Adding Migrations 13](#_Toc24896677)

[Issue with DotNet EF migrations 13](#_Toc24896678)

[Install dotnet-ef 13](#_Toc24896679)

[Add Migrations 13](#_Toc24896680)

[New Migrations Folder Added by the above Action 13](#_Toc24896681)

[Update Database 13](#_Toc24896682)

[Adding the MVC Razor View 13](#_Toc24896683)

[Install package Microsoft.VisualStudio.Web.CodeGeneration.Tools 13](#_Toc24896684)

[Folder for the View Items 14](#_Toc24896685)

[View All Food Items 14](#_Toc24896686)

[Edit Food Item View 15](#_Toc24896687)

[Create Food Item View 16](#_Toc24896688)

[Delete Item View 17](#_Toc24896689)

[Wiring up the Controller 18](#_Toc24896690)

[Add the FoodItemsController 18](#_Toc24896691)

[Add the Index Action 19](#_Toc24896692)

[Add the Create Action 19](#_Toc24896693)

[Add Edit Action 20](#_Toc24896694)

[Add Delete Action 21](#_Toc24896695)

[Add linkt to the FoodItemsController 22](#_Toc24896696)

[Update Database 23](#_Toc24896697)

[Run the Application 23](#_Toc24896698)

# Scaffolding Console App Basics

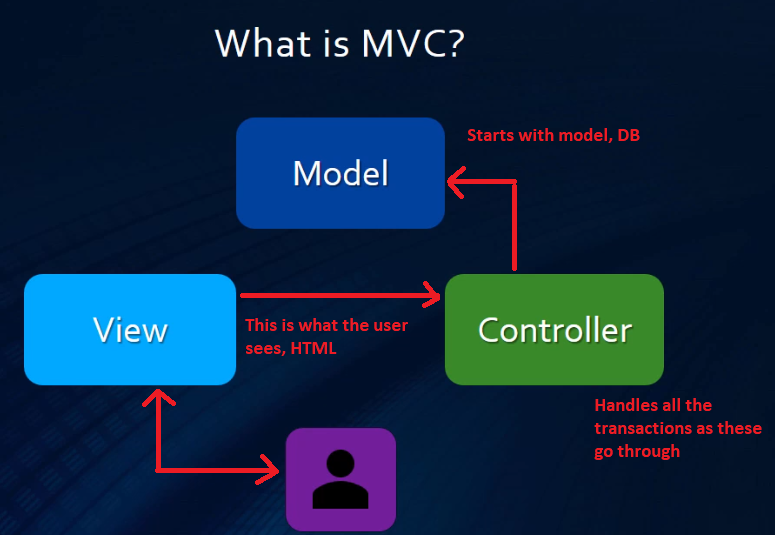
We have already run the commands multiple times, here is the info

* dotnet new console
* dotnet restore [pulls in the dependencies needed by the application]
* dotnet run [compiles and run the application]
* dotnet build [compiles the application]
* dotnet publish [packages up the files for reuse]

Take a look at this document for more details:

<https://itplate.blogspot.com/2019/11/scaffolding-applications-with-net-cli.html>

# What is MVC?



Check <https://docs.microsoft.com/en-us/aspnet/core/mvc/overview?view=aspnetcore-3.0> for more details. There is some great tutorials that go into depth than this tutorial.

# Planning our App

We are hosting a one dish party and each person will bring something to contribute to the over all meal.

We want to know what others are bringing to make sure we don’t have too many duplicate dishes.

## Requirements

|  |  |
| --- | --- |
| * Central storage area * Web based application * Accessible from internet | * Be able to add a dish * Assign it to a person * List viewable by every one |

# Creating ASP.Net MVC Core App using CLI

Run command **dotnet new mvc -o OneDishParty**

It has done the restore for us as well

Open the code with VS code

1. **CD OneDishParty**
2. and then typing **code .** [code space dot]

|  |  |
| --- | --- |
| Press **F5** to test the site. If you get into issue, make sure to compare the **tasks.json** and **launch.json** match the GitHub Repository sample. If all successful, you should see a page in your browser    You can also do on command prompt   * dotnet run | And VS Code view |

# MS SQL Database

## Working with MS SQL

This part will follow what we did in another tutorial. Plz check:

<https://itplate.blogspot.com/2019/10/di-configsettings-logging-netcore22.html>

GitHub Repository: <https://github.com/tahirjadoon/2019-DI-ConfigSettings-Logging-NetCore2.2>

and look at the notes for creating the DB: <https://github.com/tahirjadoon/2019-DI-ConfigSettings-Logging-NetCore2.2/blob/master/Notes.docx>

I am using SQL Express. For the SQL Server Express and SQL Server Management Studio (SSMS) are installed on my machine.

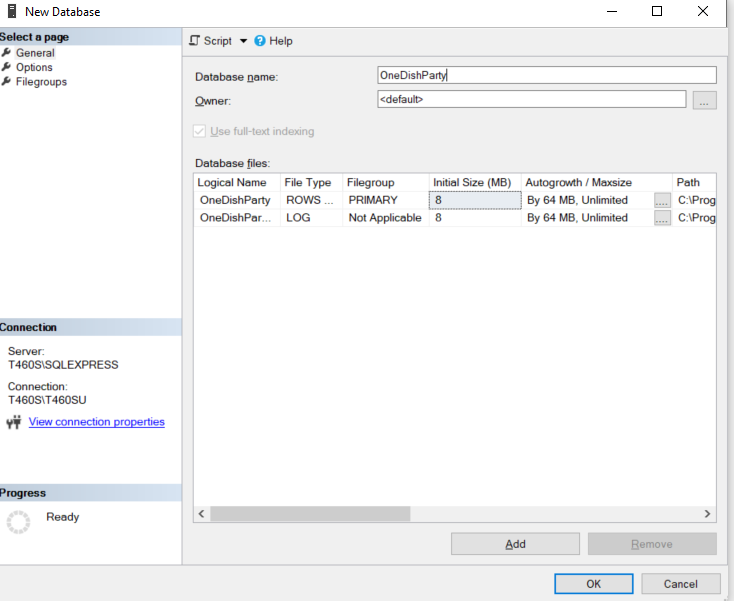
## Connecting to the Database

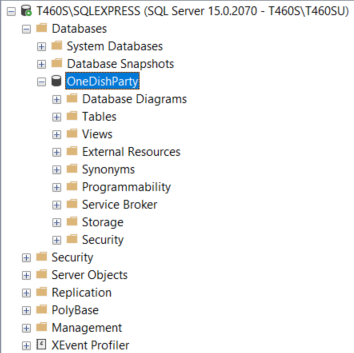
Once the installation is done, open the SSMS and connect to the database.

|  |  |
| --- | --- |
|  |  |

## Creating a Database

Right click on the “Databases” root folder and click “New Database” menu item. Provide the database name and click “Ok”.





The database that we will create will be stored in the following location.

C:\Program Files\Microsoft SQL Server\MSSQL15.SQLEXPRESS\MSSQL\DATA

You should see two files here:

* OneDishParty.mdf
* OneDishParty\_log.ldf

We will be using the Entity Framework to create the tables but for that we’ll need the database first and this is what we have done so far.

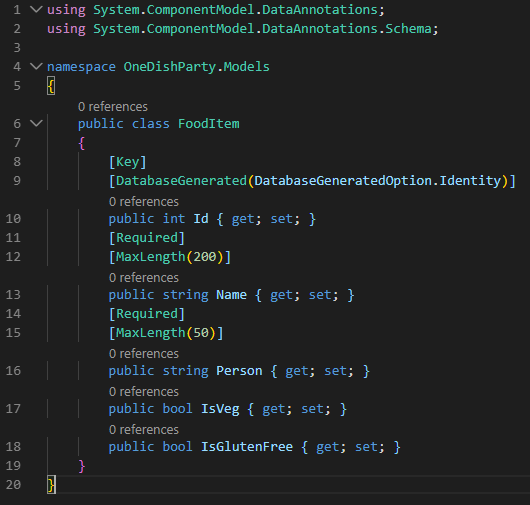
# Creating the Model

We will need a simple model to store the food items for our OneDishParty app.

Create and FoodItem.cs in the model class and add the following properties. You can generate the namespace by putting the cursor on the “public class FoodItem” row and the pressing CTRL+. together and then selecting generate NameSpace from the menu item. This class is called POCO (plain old class object).

Even though we don’t need the data annotations to specify the primary key, due to the name Id, it will be referred as primary key. But the best practice is to tell.

Mark the required fields as required and provide the max length for the two string fields.



# Entity Framework Core 3.0: Connecting to the Database

## Installing the Entity Framework Core 3.0

On command promt navigate to your working folder and then run the following command

>dotnet add package Microsoft.EntityFrameworkCore.Design

>dotnet add package Microsoft.EntityFrameworkCore.Tools

>dotnet add package Microsoft.EntityFrameworkCore.SqlServer

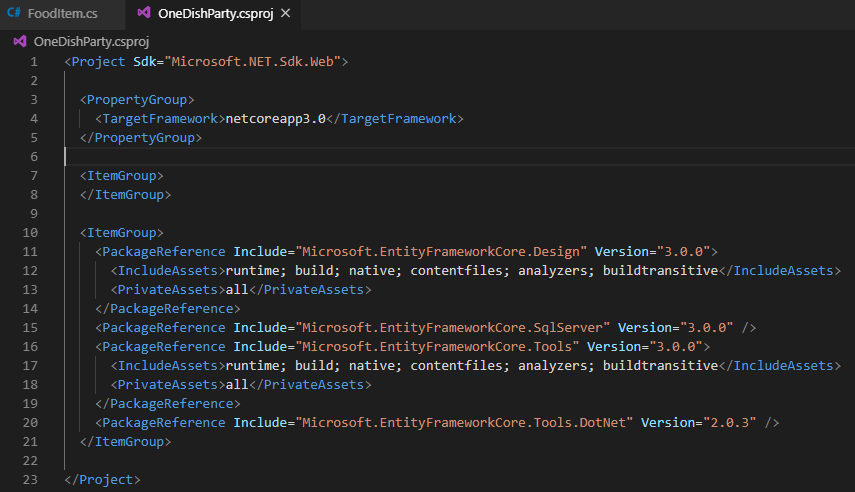
Following will install bunch of CLI tools that we will be using.

>dotnet add package Microsoft.EntityFrameworkCore.Tools.DotNet

And finally issue

>dotnet restore

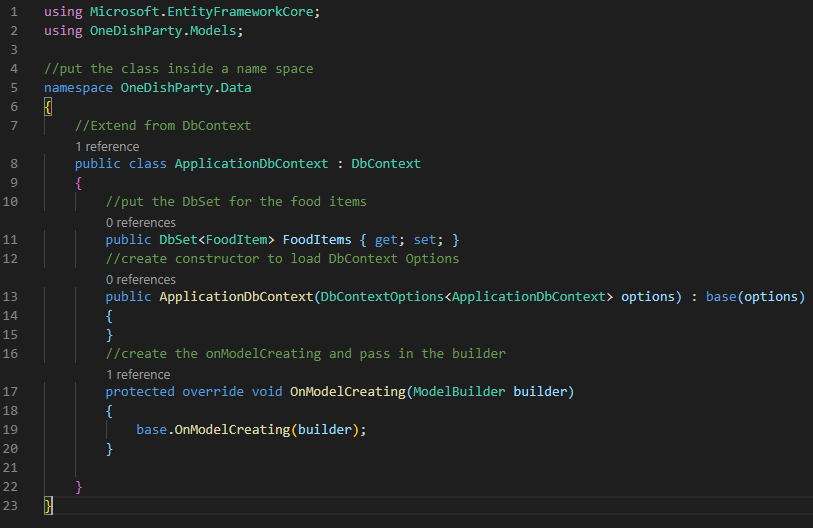
The .csproj file should look like



If the reference to the Tools.dotNet is missing then add it.

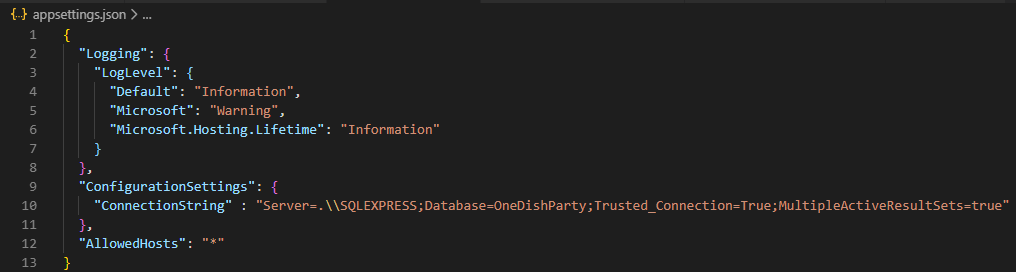
## Create a Context

Create a folder **data** in the root. Then create a folder **migrations** inside it. Then create a file **ApplicationDbContext.cs**.



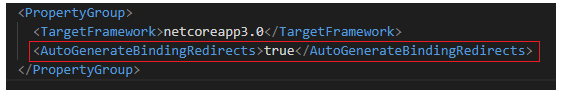
## Add Connection String

Open appsettings.json and add the connection string section to it. Since we are connections to the [SQL Server Express database that we created earlier](#_MS_SQL_Database), make sure to put in the proper name here.



## Adding AutoGenerateBindingRedirect

Add the following to the .csproj file



## Using DI to Read the Connections String

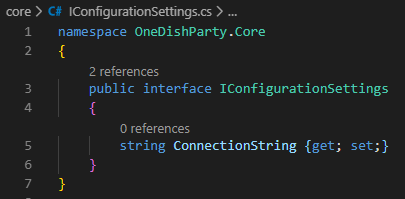
This part will follow what we did in another tutorial. Plz check:

<https://itplate.blogspot.com/2019/10/di-configsettings-logging-netcore22.html>

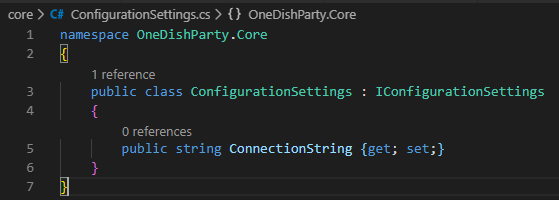
GitHub Repository: <https://github.com/tahirjadoon/2019-DI-ConfigSettings-Logging-NetCore2.2>

and look at the notes for **Setup with DI** 🡺 **Moving the connection string**: <https://github.com/tahirjadoon/2019-DI-ConfigSettings-Logging-NetCore2.2/blob/master/Notes.docx>

1. Create a folder name “core” in the root
2. Add an interface **IConfigurationSettings** to it



1. Create a class **ConfigurationSettings** and implement the interface **IConfigurationSettings** to it.

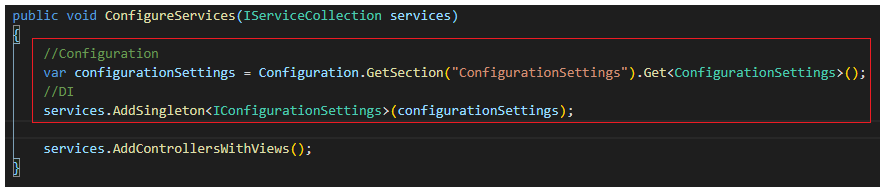


1. Then go to **Startup.cs** file in the root and

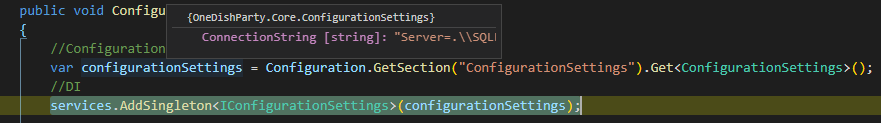
Add the using name space line at the top



And then add the two lines of code to read the section and adding a service for DI.



1. Add the break point and then run the app. ConfigurationSettings should have the connection string populated



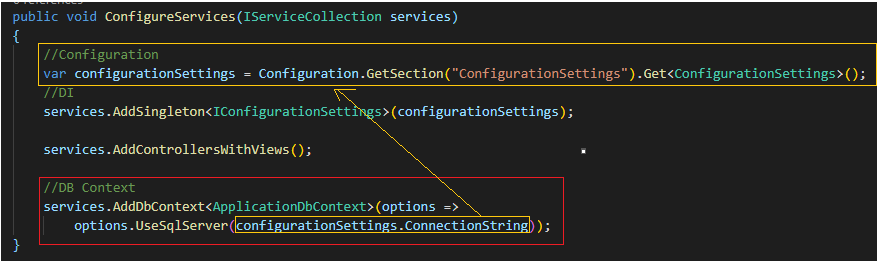
## Add DBContext to Startup

Add the following using statements





Then go to ConfigureServices and add the DbContext to it. Will need to pass in the connection string, so use the ConfigurationSettings used above to get the connection string.



### Alternate to ConfigurationSettings

Rather than getting the connection string into configurations, we can create a **ConnectionStrings** section in appsettings.json and then add **DefaultConnection** to it. And here since the DI is not needed, we can just directly use it as



## Adding Migrations

### Issue with DotNet EF migrations

<https://github.com/aspnet/EntityFrameworkCore/issues/15448> Use the below Install before adding the migrations.

### Install dotnet-ef

> **dotnet tool install -g dotnet-ef --version 3.0.0-\***

Since you just installed the .NET Core SDK, you will need to reopen the Command Prompt window before running the tool you installed.

You can invoke the tool using the following command: dotnet-ef

Tool 'dotnet-ef' (version '3.0.0') was successfully installed..

### Add Migrations

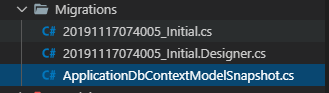
>**dotnet-ef migrations add Initial -s "OneDishParty.csproj"**

info: Microsoft.EntityFrameworkCore.Infrastructure[10403]

Entity Framework Core 3.0.0 initialized 'ApplicationDbContext' using provider 'Microsoft.EntityFrameworkCore.SqlServer' with options: None

Done. To undo this action, use 'ef migrations remove'

### New Migrations Folder Added by the above Action



### Update Database

Check the [section below](#_Update_Database), after the [controller](#_Wiring_up_the) has been wired up.

### Seeding the Data

Take a look at **Notes10 - WebApiApp.docx** 🡺 **Seeding the Data** section for an example.

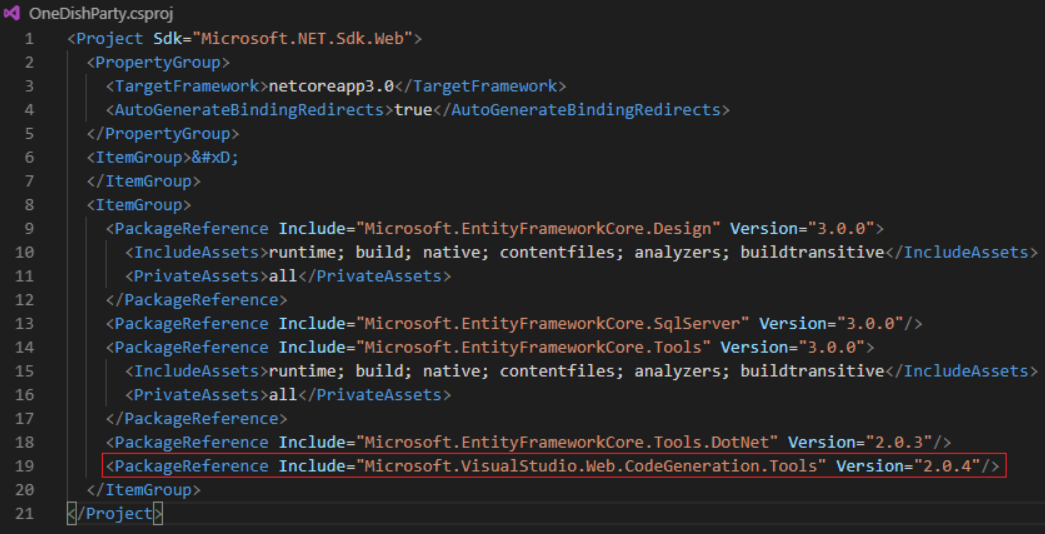
# Adding the MVC Razor View

## Install package Microsoft.VisualStudio.Web.CodeGeneration.Tools

Add nuget package

* Microsoft.VisualStudio.Web.CodeGeneration.Tools version 2.0.4

The csproj will look like



## Folder for the View Items

All the view for the food items will go in the **Views** folder. Create a sub folder called **FoodItem** here.

## View All Food Items

Create an **Index.cshtml** file in the **Views\FoodItem** folder. It will get the List of food items via model **FoodItem** that we created earlier. Make sure to put the **@model** line towards the top of the view.

Here we are using the layout that was installed when installing the application. So provide a **Title** and also reference the view to be used towards the top.

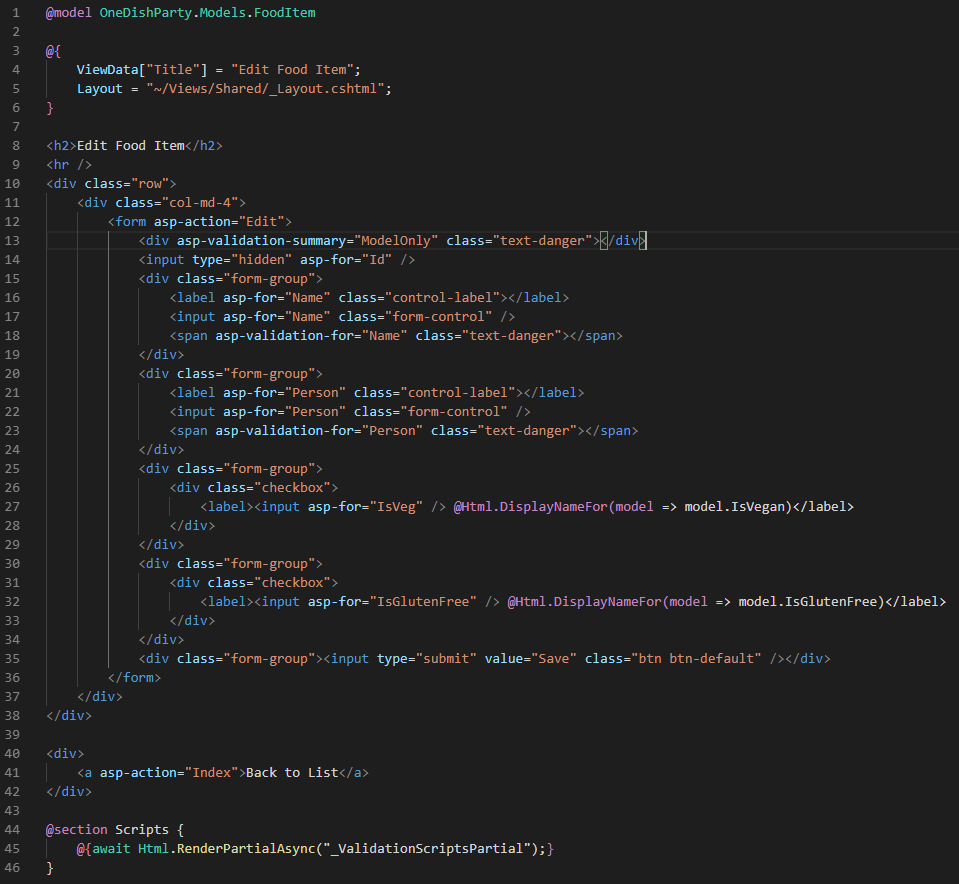
Then use the tabled to layout the food items already picked. Also add links to Add, Edit and Delete.



## Edit Food Item View

Just like the [view all](#_View_All_Food) view above, we’ll need the Edit view as well to edit the item. Create an **Edit.cshtml** file in the **Views\FoodItem** folder. It will get the selected food item via model **FoodItem** that we created earlier. Make sure to put the **@model** line towards the top of the view.

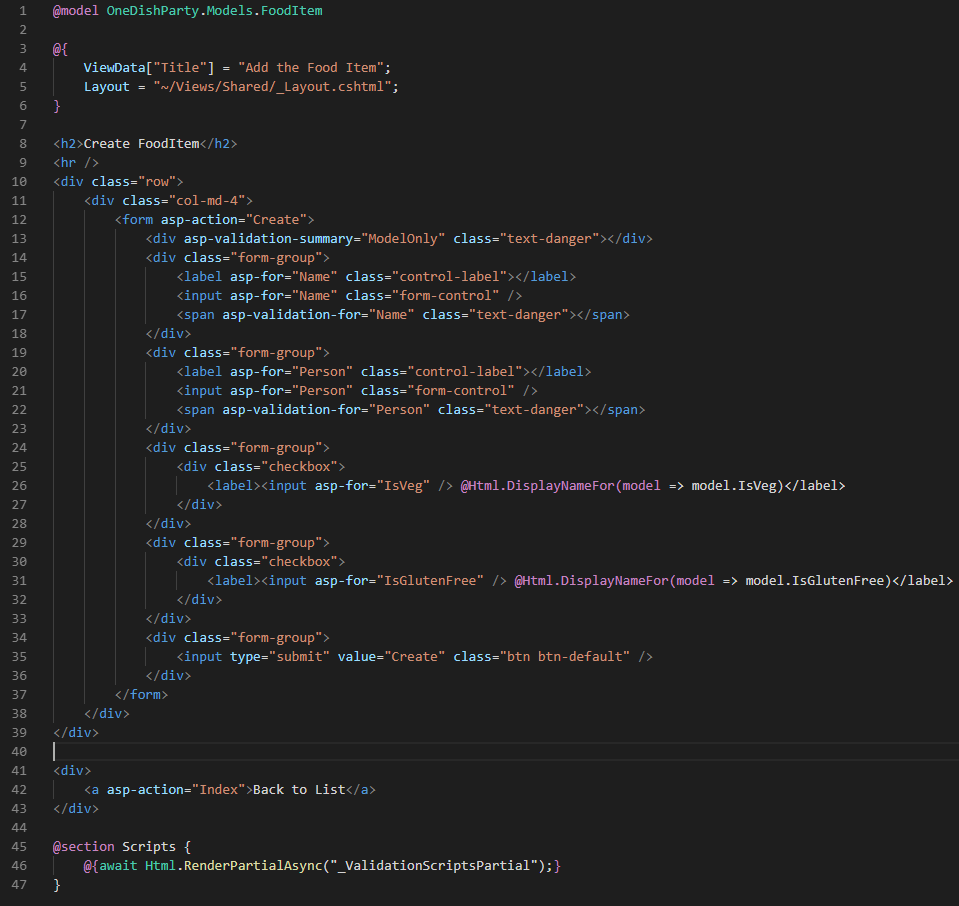
We’ll need a form to show current entries and the user can change these. Also, the form will be validated properly.



## Create Food Item View

Just like the [view all](#_View_All_Food) and [edit](#_Edit_Food_Item) views above, we’ll need the Create view as well to add the item. Create an **Create.cshtml** file in the **Views\FoodItem** folder. Make sure to put the **@model** line towards the top of the view.

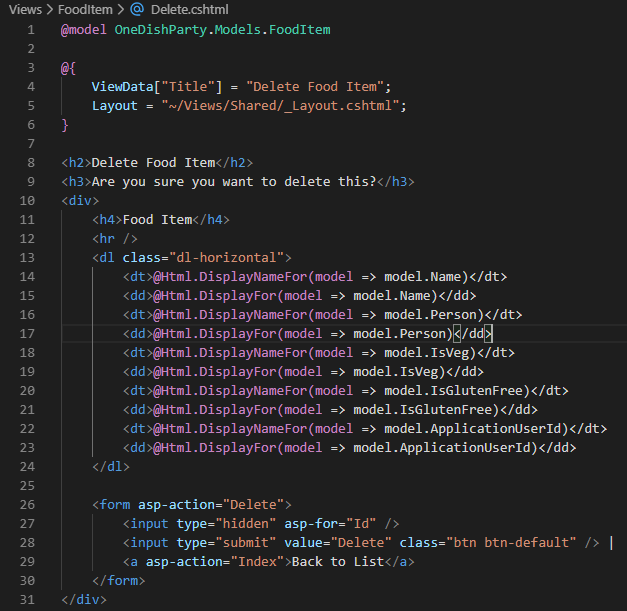
We’ll need a form to capture user data. Also, the form will be validated properly.



## Delete Item View

Just like the [view all](#_View_All_Food), [edit](#_Edit_Food_Item) and [create](#_Create_Food_Item) views above, we’ll need the Delete view as well to delete the item. Create an **Delete.cshtml** file in the **Views\FoodItem** folder. Make sure to put the **@model** line towards the top of the view. This will be confirmation page which will list the current item.

The page will have a form towards the botton with a hidden Id field.

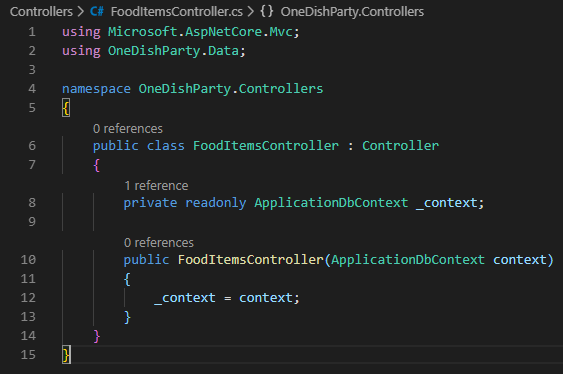


# Wiring up the Controller

## Add the FoodItemsController

Go to the **Controller** foler and create a new Controller called **FoodItemsController.cs**. It will extend the controller class. Also, the controller will have Index, Edit, Create and Delete Actions. Make are to put in the name space here.

We’ll need to drop in the [ApplicationDbContext](#_Create_a_Context) as well. Remember that it was created in the **data** folder. So use the name space **OneDishParty.Data**. Add a constructor to load the context.



## Add the Index Action

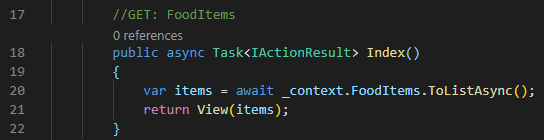
Add the Index action, it will be Async method and it will return the IActionResult. Since it is going to return a task, add the following:



Since we are using the DB calls directly here and there is no repository pattern, make sure to add the following as well.



The Index action will look like



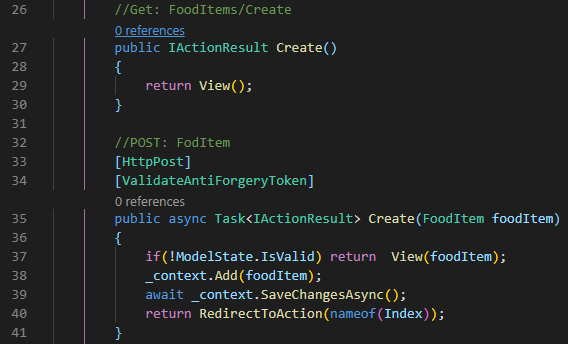
## Add the Create Action

Since this is going to create a new FoodItem, we are not pulling anything from the database. There will be two action, a GET when the link will be click and a POST when the submit button will be clicked.

Since the post will be expecting a model, make sure to add



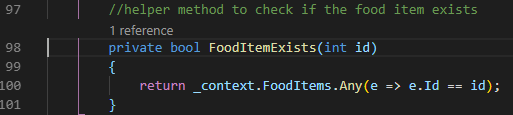
And the two actions will look like following. The POST will first check that the ModelState is valid and then will insert the data and redirect to the [Index](#_Add_the_Index).



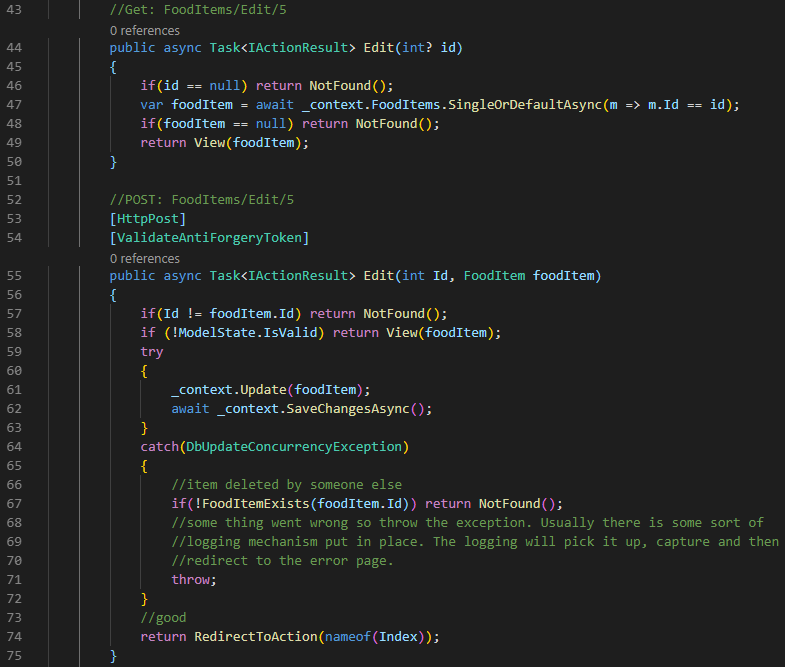
## Add Edit Action

Just like [Create](#_Add_the_Create) above, we’ll need two actions GET and POST to display and catpture the users changes. We’ll create a helper method here to check the food item first. For this make sure to do the following first.





And then the Edit actions will look like



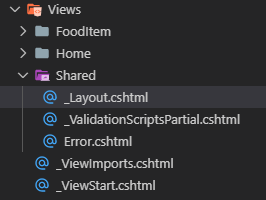
## Add Delete Action

Just like [Create](#_Add_the_Create) and [Edit](#_Add_Edit_Action), Delete will also need two GET and POST actions. The Get method is for the confirmation and post wil actually do the delete.

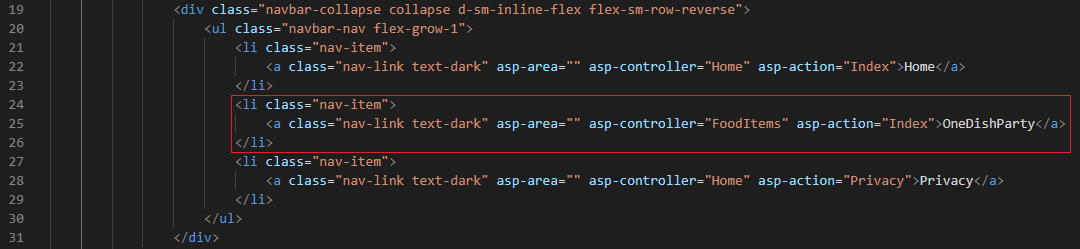


## Add linkt to the FoodItemsController

Open **\_Layout.cshtml** from inside the **Views\Shared** folder



And then add the following to the menu at the top

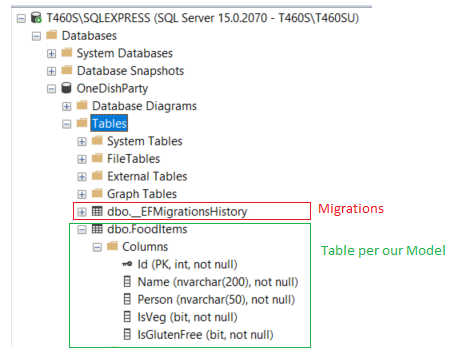


# Update Database

Just like [create migrations](#_Adding_Migrations), we need to update the database. Before issuing the command, make sure that the project compiles. Take care of any issues and then run the command.

>**dotnet ef database update**

Refresh the database and the migrations and table should be there.



# Run the Application

Either do **>dotnet run** or press **F5** to run the application. If doing dotnet run, to stop use **CTRL+C**.

Click the OneDishParty link and start playing with the app.

