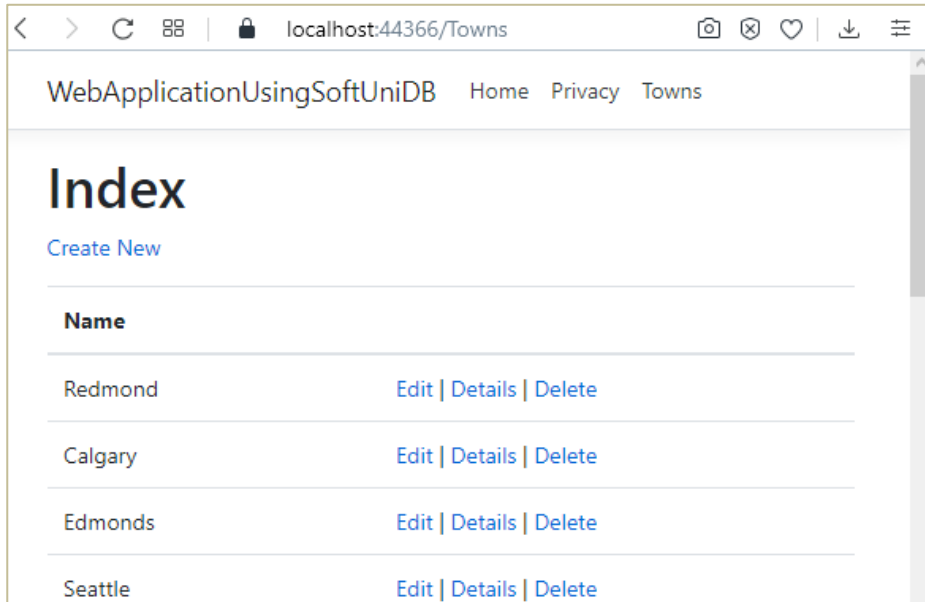


# Lab: ASP.NET MVC with DB

This document defines several walkthroughs for creating ASP.NET MVC-based apps, from setting up the framework to implementing the fully functional applications.

## 1. Application with SoftUni Database

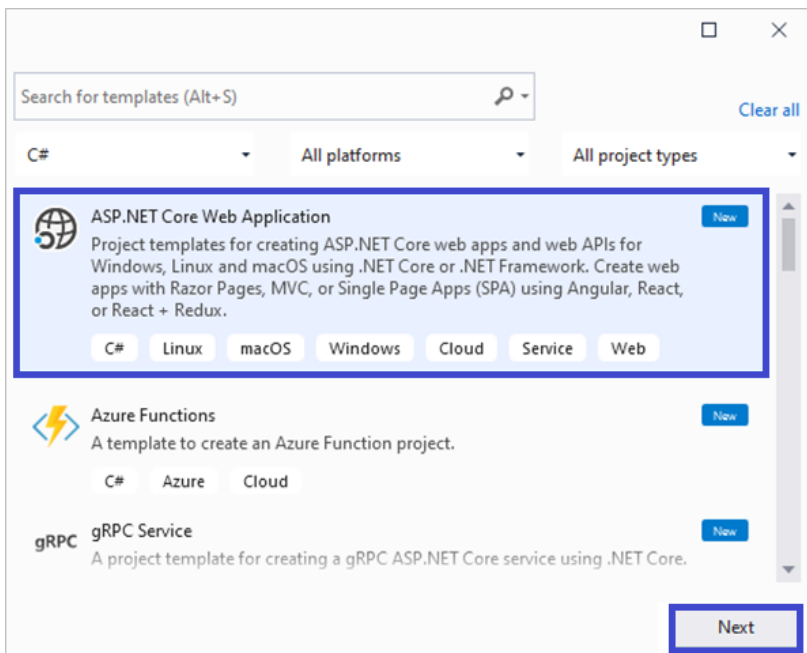
Create a Web application, which **displays and modifies** the data from table **Towns** in the **SoftUni Database**. The app should look like this:



We are going to **use** the existing **SoftUni** database. **Start** your **Microsoft SQL Management Studio**.

## Create a New ASP.NET MVC Project

Open **Visual Studio** and create a C# web project using the **ASP.NET Web Application (.NET Core)** template.





In the "Create a new ASP.NET Core Web Application" window, choose **[Model-View-Controller (MVC)]**.


# Create a new ASP.NET Core web application


.NET Core


ASP.NET Core 3.1

**Empty**  
An empty project template for creating an ASP.NET Core application. This template does not have any content in it.

**API**  
A project template for creating an ASP.NET Core application with an example Controller for a RESTful HTTP service. This template can also be used for ASP.NET Core MVC Views and Controllers.

**Web Application**  
A project template for creating an ASP.NET Core application with example ASP.NET Razor Pages content.

**Web Application (Model-View-Controller)**  
A project template for creating an ASP.NET Core application with example ASP.NET Core MVC Views and Controllers. This template can also be used for RESTful HTTP services.

**React.js**

Get additional project templates

**Authentication**  
No Authentication  
[Change](#)

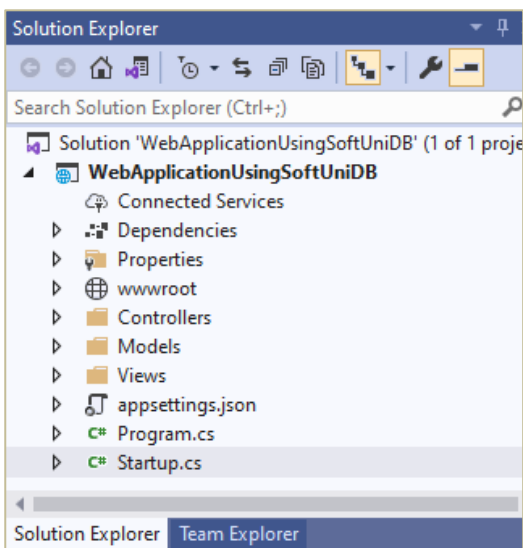
**Advanced**  
☒ Configure for HTTPS  
☐ Enable Docker Support  
(Requires [Docker Desktop](#))  

Linux

Author: Microsoft  
Source: Templates 3.1.12

Back

Create



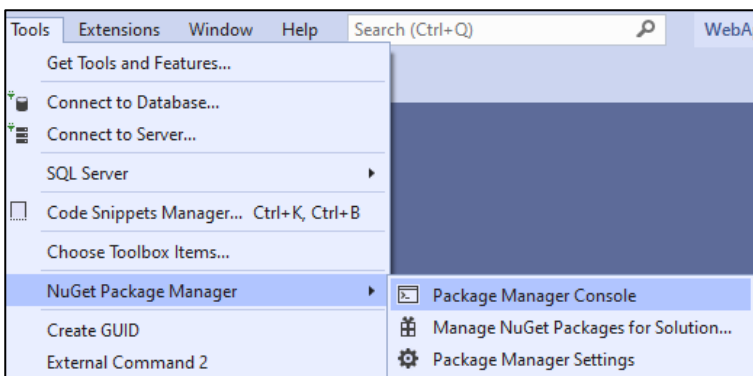
## Install Entity Framework Core

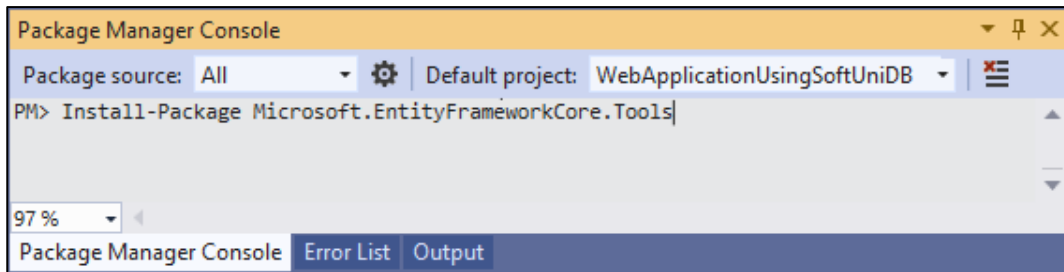
Go to [Tools] → [NuGet Package Manager] → [Package Manager Console] and run the following commands individually:

```
Install-Package Microsoft.EntityFrameworkCore.Tools
```

```
Install-Package Microsoft.EntityFrameworkCore.SqlServer.Design
```

```
Install-Package Microsoft.EntityFrameworkCore.SqlServer
```

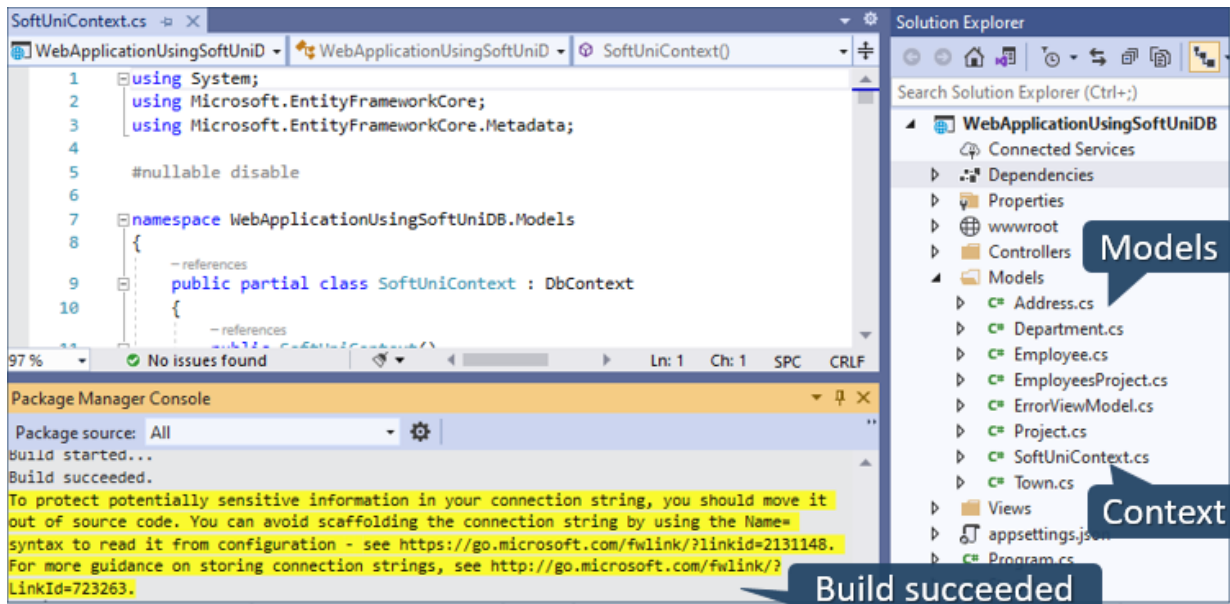




## Create Model and Database Context

In the **Package Manager Console** run the following command:

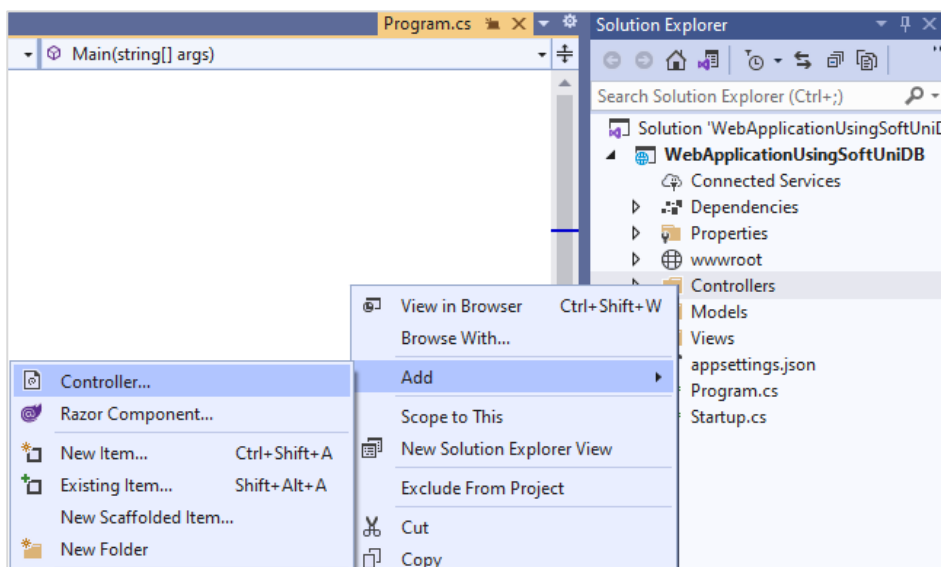
```
Scaffold-DbContext -Connection
"Server=(localdb)\MSSQLLocalDB;Database=SoftUni;Integrated Security=True;" -
Provider Microsoft.EntityFrameworkCore.SqlServer -OutputDir Models
```



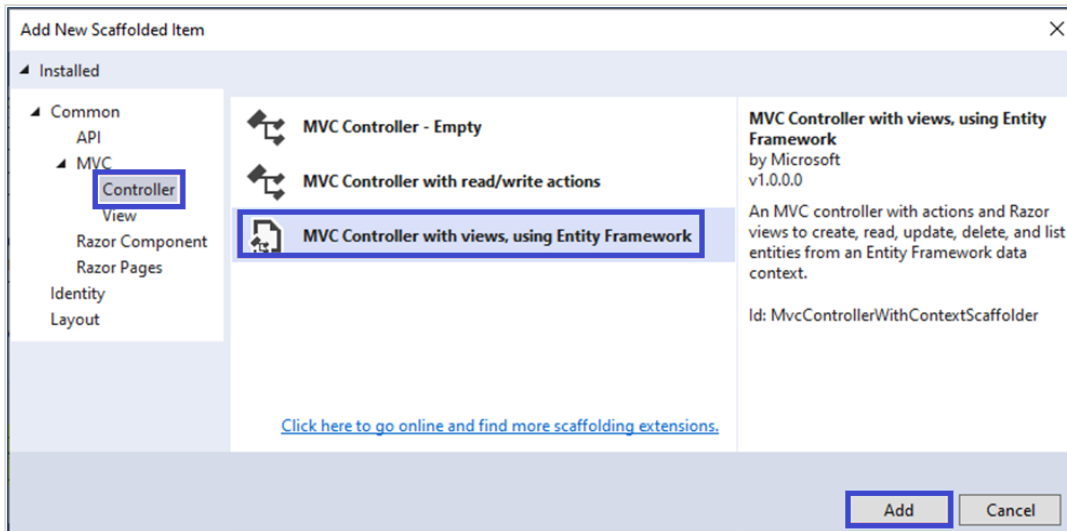
## Scaffold Controller with Views

Create an **ASP.NET Core MVC Controller** for handling data and performing basic CRUD operations.

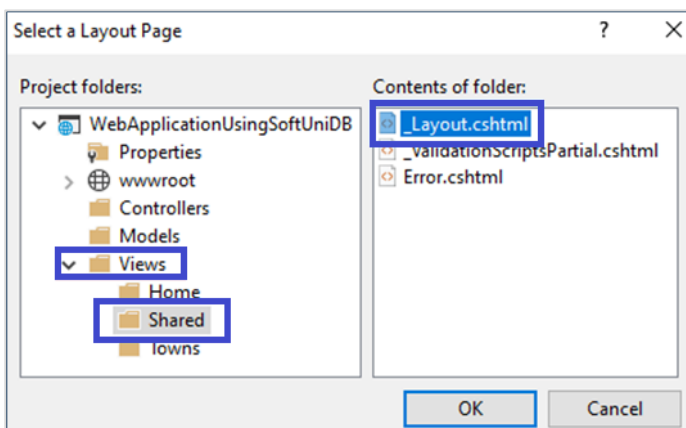
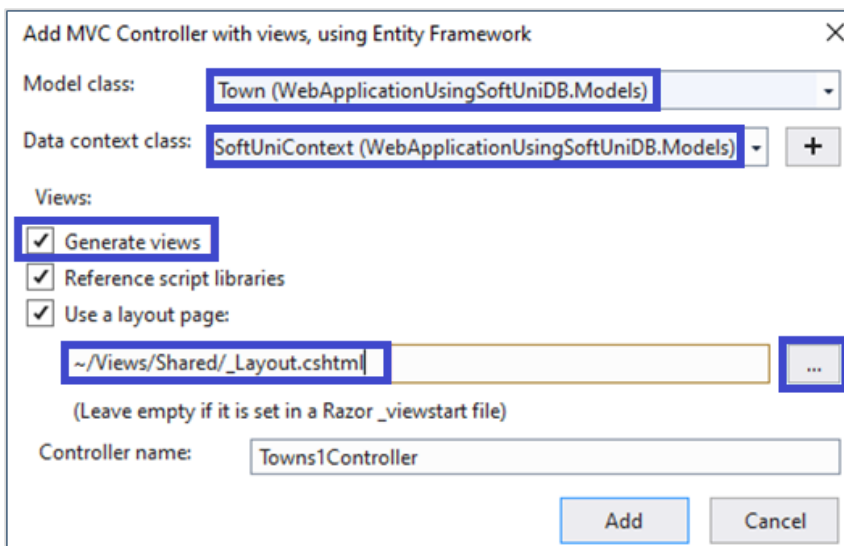
Right-click on the **Controllers** folder in Solution Explorer and select **[Add] → [Controller]**:



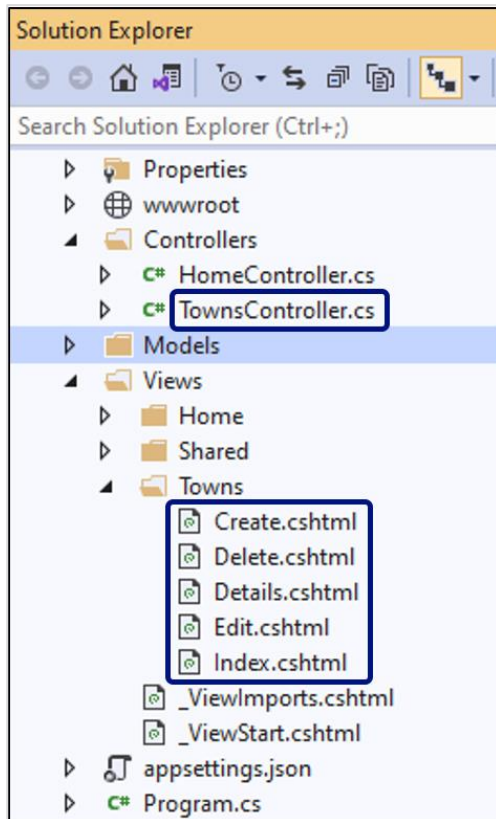
Select **[MVC Controller with Views using Entity Framework]**.



- Select **Town** as **Model class**
- **SoftUniContext** as the **Data Context class**
- Tick the **[Generate Views]** option
- In the "Select a Layout Page", browse through **Views** → **Shared** → **\_Layout.cshtml**
- Give a good name for the controller class, e.g. **TownsController**



Now, you will have generated a new **controller** class, with a few **views** in the code:



## Register the DB Context Class in the App Configuration

Register your configuration class at the **ConfigureServices()** method in **Startup.cs**, by adding:

```
services.AddDbContext<SoftUniContext>();
```



## Display the “Towns” Page in the Menu

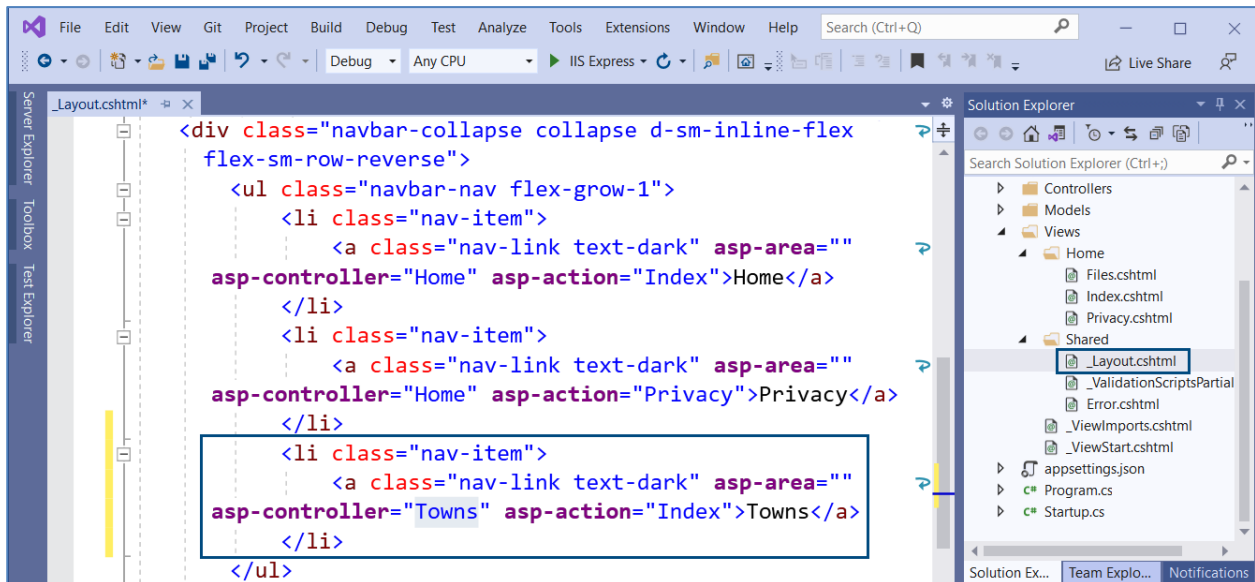
To display the “Towns” page in the app, you need to add it to the layout view:

In Solution Explorer go to: **Views** → **Shared** → **\_Layout.cshtml**.

In the `<ul class="navbar-nav flex-grow-1">` add new `<li>` element:

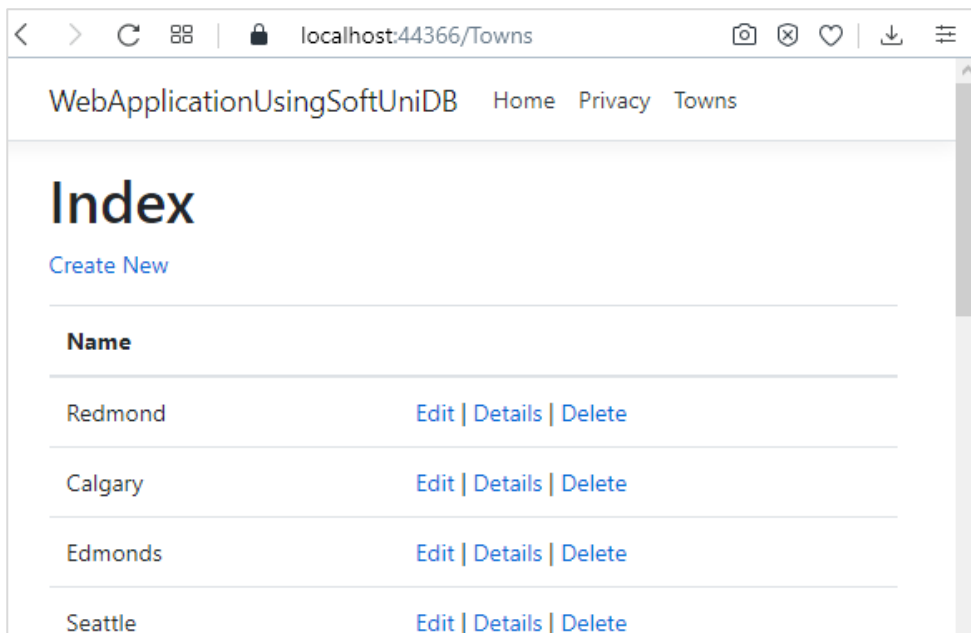
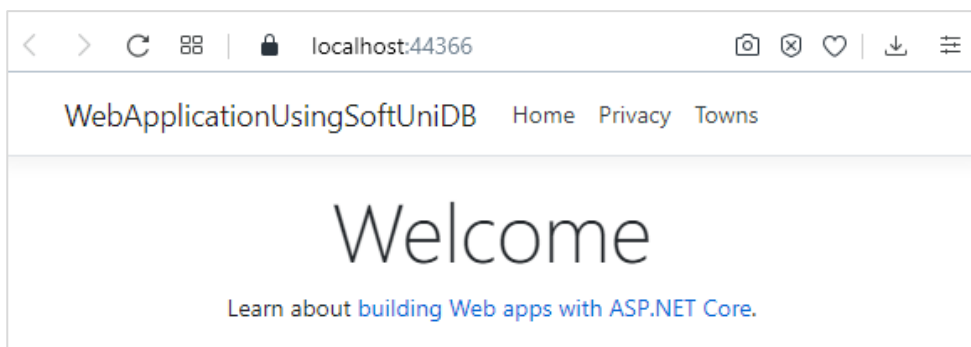
```
<li class="nav-item">
  <a class="nav-link text-dark" asp-area="" asp-controller="Towns" asp-
  action="">Towns</a>
</li>
```

This is how it looks like:



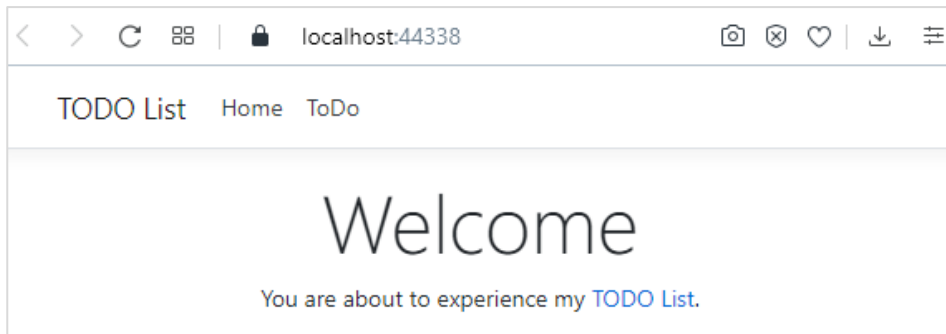
## Run and Test the App

Now **build** and **run** the application using [Ctrl] + [F5].



## 2. TODO List


Create a “**TODO List**” Web application, which keeps track of a person’s **tasks** inside a **database**. The application should support **creating** tasks and **deleting** tasks.



## Create the Database

Create **database** named "ToDoDB" in **Microsoft SQL Management Studio**. It will have a **ToDoTasks** table with **2 columns**:

- **Id** – a unique **integer**, with which to differentiate tasks from one another.
- **Title** – the **title** of the task, stored as a **string**.

	Column Name	Data Type	Allow Nulls
	Id	int	<input type="checkbox"/>
	Title	nchar(50)	<input checked="" type="checkbox"/>
			<input type="checkbox"/>

Column Properties	
Identity Specification	Yes
(Is Identity)	Yes
Identity Increment	1
Identity Seed	1

## Insert Some Sample Data

Create 3 tasks in the table:

Tasks
Water the flowers.
Buy cat food.
Make the project for Wednesday.

	Id	Title
	1002	Water the flowers. ...
	1004	Buy cat food.
	1005	Make the project for Wednesday. ...
»*	NULL	NULL

## Create a New Project

Create new project and name it **TODOTasksList**.

Open **Visual Studio** and create a C# web project using the **ASP.NET Web Application (.NET Core)** template.

In the **Create a new ASP.NET Core Web Application** window, choose **Model-View-Controller (MVC)**.

**Integrating Entity Framework Core:**

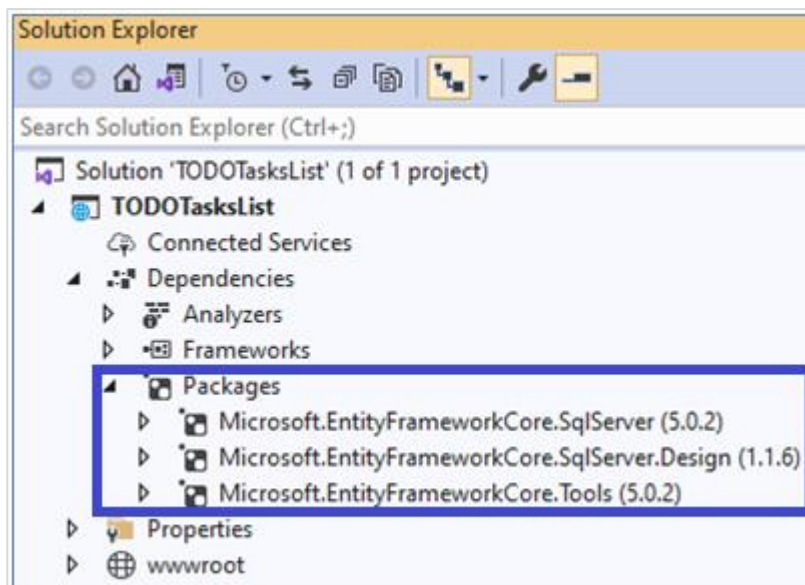
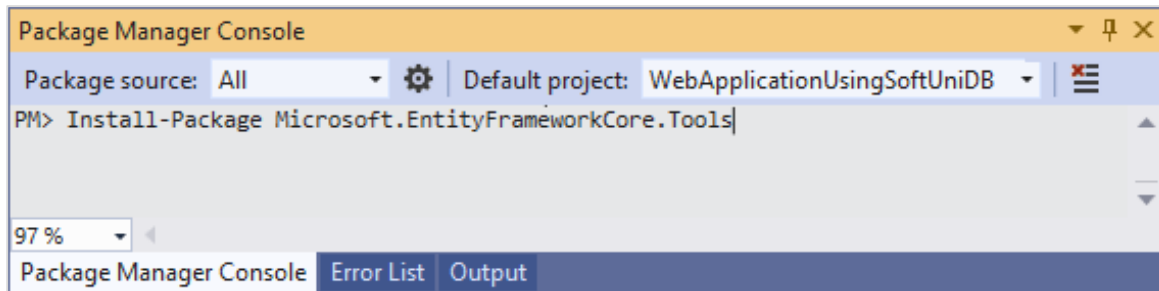
Go to **[Tools] → [NuGet Package Manager] → [Package Manager Console]** and run the following commands individually:



```
Install-Package Microsoft.EntityFrameworkCore.Tools
```

```
Install-Package Microsoft.EntityFrameworkCore.SqlServer.Design
```

```
Install-Package Microsoft.EntityFrameworkCore.SqlServer
```

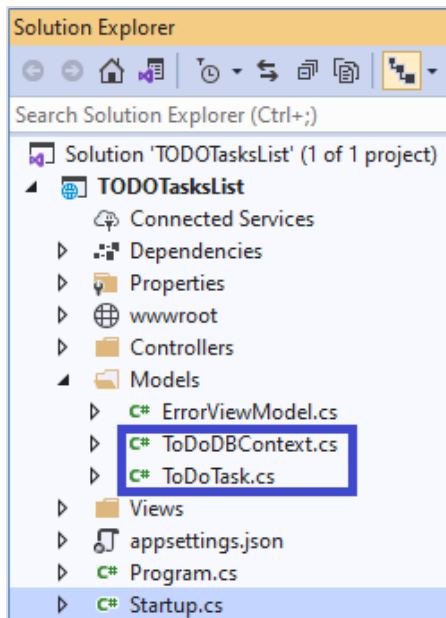


## Create Model and Database Context

In the **Package Manager Console** run the following command:

```
Scaffold-DbContext -Connection  
"Server=(localdb)\MSSQLLocalDB;Database=ToDoDB;Integrated Security=True;" -Provider  
Microsoft.EntityFrameworkCore.SqlServer -OutputDir Models
```





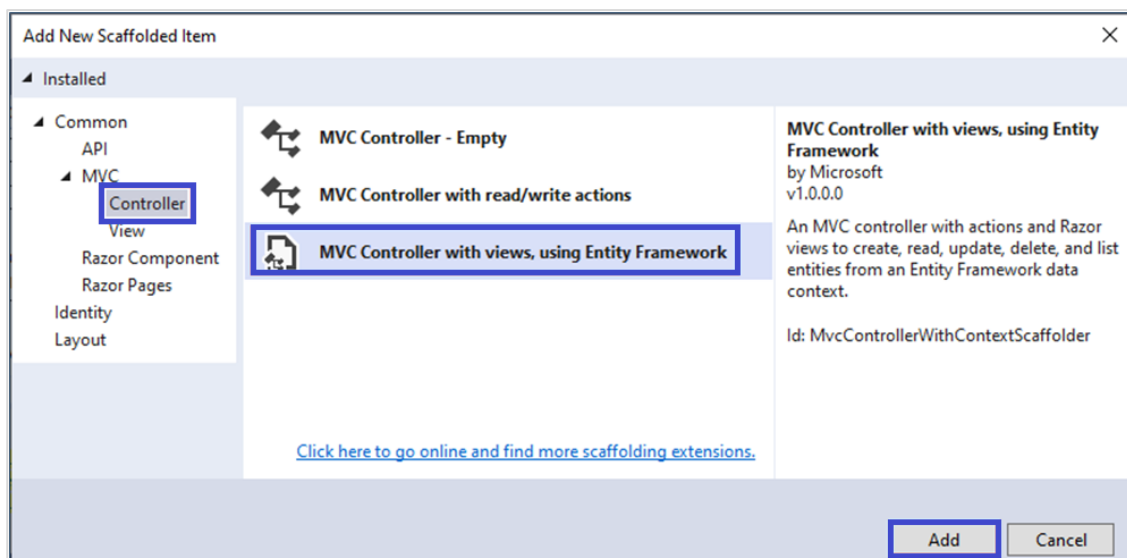
## Scaffold Controller with Views

Create an **ASP.NET Core MVC Controller** for handling data and performing basic CRUD operations.

Right-click on the **Controllers** folder in Solution Explorer and select **[Add] > [Controller]**

Select **MVC Controller with Views using Entity Framework**.

- Select **ToDoTasks** as **Model class**
- **ToDoDbContext** as the **Data Context class**
- Tick the **[Generate Views]** option
- In the "Select a Layout Page", browse through **Views → Shared → \_Layout.cshtml**



Add MVC Controller with views, using Entity Framework

Model class: **ToDoTask (TODOTasksList.Models)**

Data context class: **ToDoDbContext (TODOTasksList.Models)** +

Views:

☒ **Generate views**

☒ Reference script libraries

☒ Use a layout page:

**~/Views/Shared/\_Layout.cshtml** ...

(Leave empty if it is set in a Razor \_viewstart file)

Controller name: **ToDoTasksController**

Add Cancel

Select a Layout Page

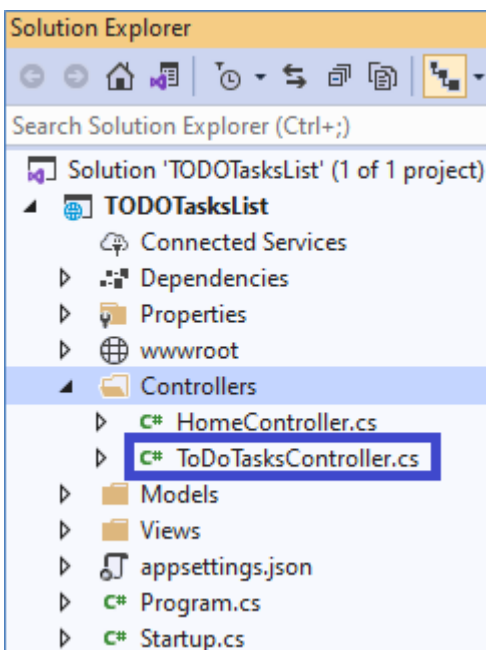
Project folders:

- ▼ TODOTasksList
  - Properties
  - wwwroot
    - Controllers
    - Models
    - ▼ Views
      - Home
      - Shared

Contents of folder:

- \_Layout.cshtml**
- \_validationScriptsPartial.cshtml
- Error.cshtml

OK Cancel



## Register the DbContext Class in the App Config

Register your configuration class at the **ConfigureServices()** method in **Startup.cs**, by adding:

```
services.AddDbContext<ToDoDbContext>();
```

```

0 references
public void ConfigureServices(IServiceCollection services)
{
    services.AddControllersWithViews();
    services.AddDbContext<ToDoDbContext>();
}

```

## Display the “Tasks” Page

To **display the page** in the app, you need to add it to the title page.

In **Solution Explorer** go to: **Views** → **Shared** → **\_Layout.cshtml**.

In the `<ul class="navbar-nav flex-grow-1">` add new `<li>` element:

```

<li class="nav-item">
  <a class="nav-link text-dark" asp-area="" asp-controller="ToDoTasks" asp-
    action="Index">ToDo</a>
</li>

```

```

<div class="container">
  <a class="navbar-brand" asp-area="" asp-controller="ToDoTasks" asp-action="Index">TODO List</a>
  <button class="navbar-toggler" type="button" data-toggle="collapse" data-target=".navbar-collapse" aria-control
    aria-expanded="false" aria-label="Toggle navigation">
    <span class="navbar-toggler-icon"></span>
  </button>
  <div class="navbar-collapse collapse d-sm-inline-flex flex-sm-row-reverse">
    <ul class="navbar-nav flex-grow-1">
      <li class="nav-item">
        <a class="nav-link text-dark" asp-area="" asp-controller="Home" asp-action="Index">Home</a>
      </li>
      <li class="nav-item">
        <a class="nav-link text-dark" asp-area="" asp-controller="ToDoTasks" asp-action="Index">ToDo</a>
      </li>
    </ul>
  </div>
</div>

```

## Change the App Details

Change the **title** and **link**. In **\_Layout.cshtml** change the text of the first `<a>` element to **TODO List** and the controller to lead to **ToDoTasks**. Delete the `<li class="nav-item">` that gives the **[Privacy]** button in the navigation bar.

```

<nav class="navbar navbar-expand-sm navbar-toggleable-sm navbar-light bg-white border-bottom box-shadow mb-3">
  <div class="container">
    <a class="navbar-brand" asp-area="" asp-controller="ToDoTasks" asp-action="Index">TODO List</a>

```

Change the index page. In **Solution Explorer** go to: **Views** → **Home** → **Index.cshtml**. Change the **paragraph text** to: "You are about to experience my TODO List." and make the **last two words link** which leads to **TODO Page**.

```

<div class="text-center">
  <h1 class="display-4">Welcome</h1>
  <p>You are about to experience my <a href="ToDoTasks">TODO List</a>.</p>
</div>

```

Change the **TODO Page**. In **Solution Explorer** go to: **Views** → **ToDoTasks** → **Index.cshtml**. Change the heading to "To Do Tasks":

```

@model IEnumerable<TODOTasksList.Models.ToDoTask>

@{
    ViewData["Title"] = "Index";
    Layout = "~/Views/Shared/_Layout.cshtml";
}

<h1>To Do Tasks</h1>

```

## Run and Test the App

Now **build** and **run** the application using **[Ctrl] + [F5]**.

