|Developing ASP.NET core Web Apps.

Overview

If you´re currently raking this laboratory exercises, it means that you have experience programming in C# and you´re looking to improve your developer skill by learning how to use the ASP.NET technology to develop apps and to work with APIs.

Practice makes perfect, so they say. Well, let´s prove it! 😉

Requirements

- For this lab exercises you will need experience programming in C#
- You should be familiar with basic concepts of C#.
- You should have some experience at creating simple console apps in Visual Studio.
- You´ll need to install Visual Studio 2019 (IDE) https://visualstudio.microsoft.com/

General Objectives

In this lab sessions we will put in practice the concepts that we learned in class, with this lab sessions students will also become familiar with the fundamental concepts of using ASP.NET technologies to develop apps and manage APIs. At the end of each module, you will be able to find a series of tasks that will challenge your knowledge so you can develop solutions with ASP.NET on your own.

Module 01: Routing APIs

Lab Setup

Estimated time: **50-60 min.**

Introduction

When we talk about the web and how it works, we can think of information that is traveling over the internet, and in order to start its journey from a user in some one point of the world to a server allocated in a completely different region or part of the world, it´s necessary that the requesting user information has to over a protocol (HTTP) to start that communication with the server.

The successful communication between user and server will now be execute different methods/controllers depending on each request type.

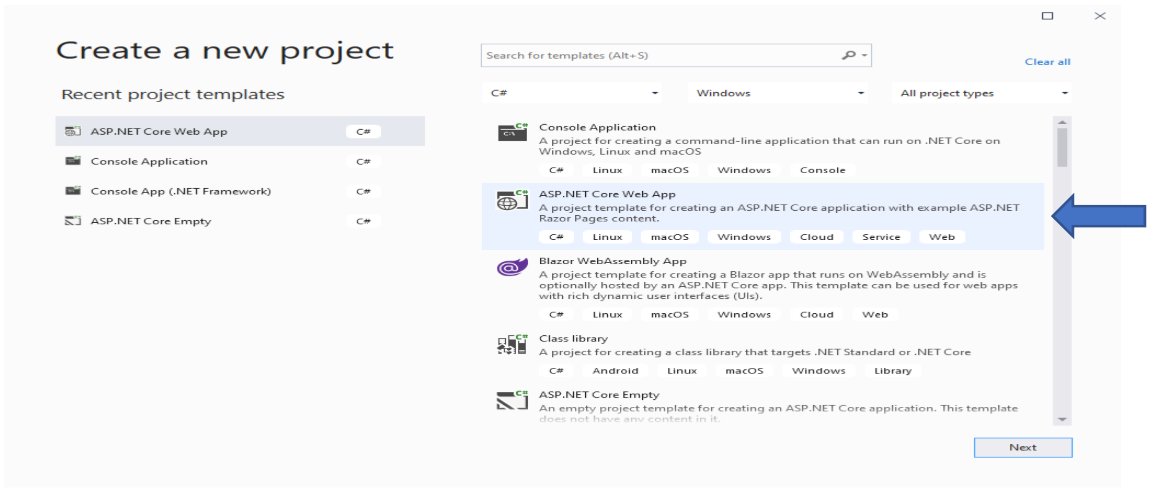
Objectives

In this lab we will perform a basic routing exercise in ASP.NET core

a. Call an API.
b. How routing works.
c. Endpoints and endpoint routing.
d. Become familiar with the ASP.NET core framework.
e. Route Requests of an API.

Exercise 01: Endpoints

1. Open Visual Studio 2019.
2. Create a new Project, but this time choose the second option that says ASP.NET Core web app and click next.


Create a new project

Recent project templates

- ASP.NET Core Web App C#
- Console Application C#
- Console App (.NET Framework) C#
- ASP.NET Core Empty C#

Search for templates (Alt+S)

Clear all

C# | Windows | All project types

Console Application
A project for creating a command-line application that can run on .NET Core on Windows, Linux and macOS
C# Linux macOS Windows Console

ASP.NET Core Web App
A project template for creating an ASP.NET Core application with example ASP.NET Razor Pages content.
C# Linux macOS Windows Cloud Service Web

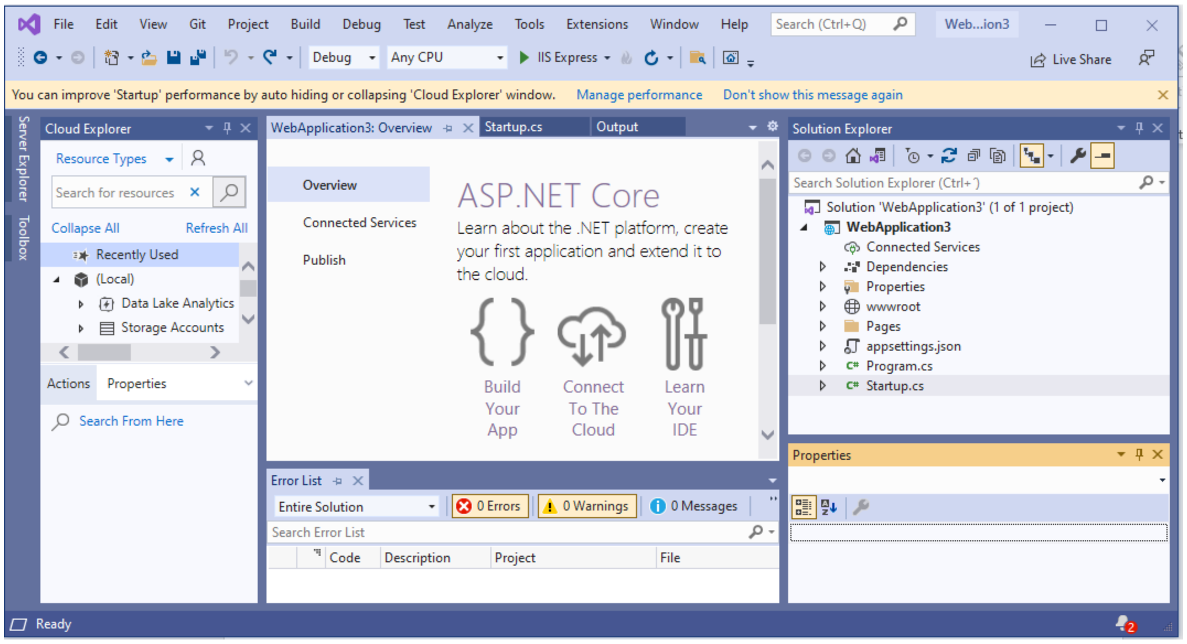
Blazor WebAssembly App
A project template for creating a Blazor app that runs on WebAssembly and is optionally hosted by an ASP.NET Core app. This template can be used for web apps with rich dynamic user interfaces (UIs).
C# Linux macOS Windows Cloud Web

Class library
A project for creating a class library that targets .NET Standard or .NET Core
C# Android Linux macOS Windows Library

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

Next

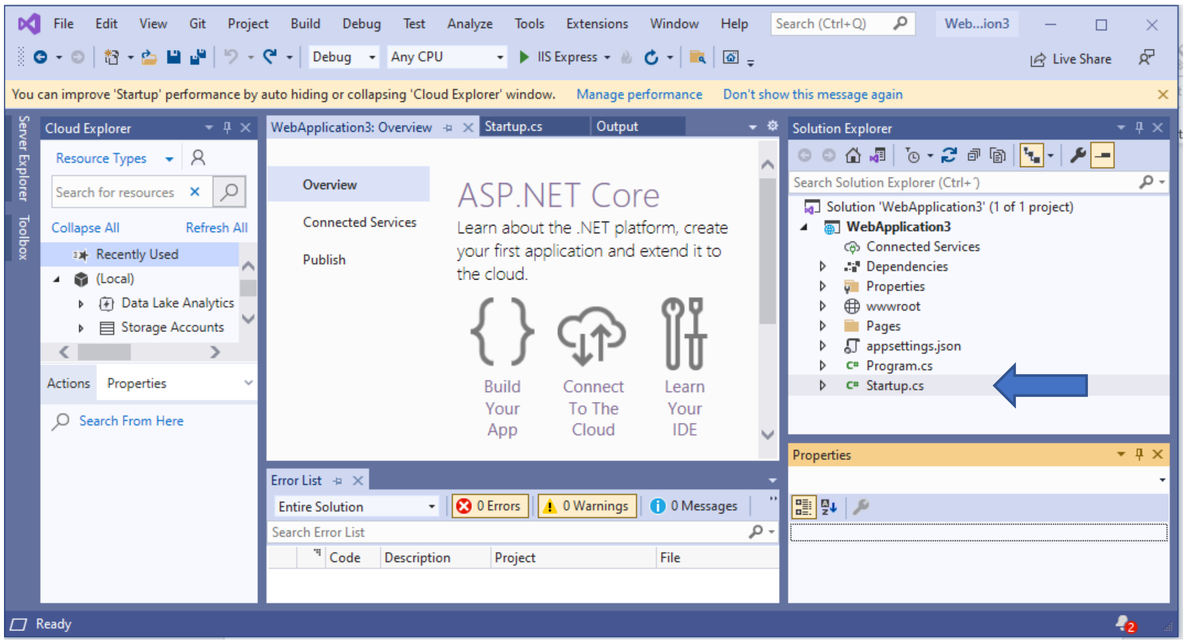
3. Choose a name for your project and click next.
4. Leave the default options and click next.
5. When the new project opens it, your Visual Studio window will look like this:


You can improve 'Startup' performance by auto hiding or collapsing 'Cloud Explorer' window. Manage performance Don't show this message again

ASP.NET Core

Learn about the .NET platform, create your first application and extend it to the cloud.

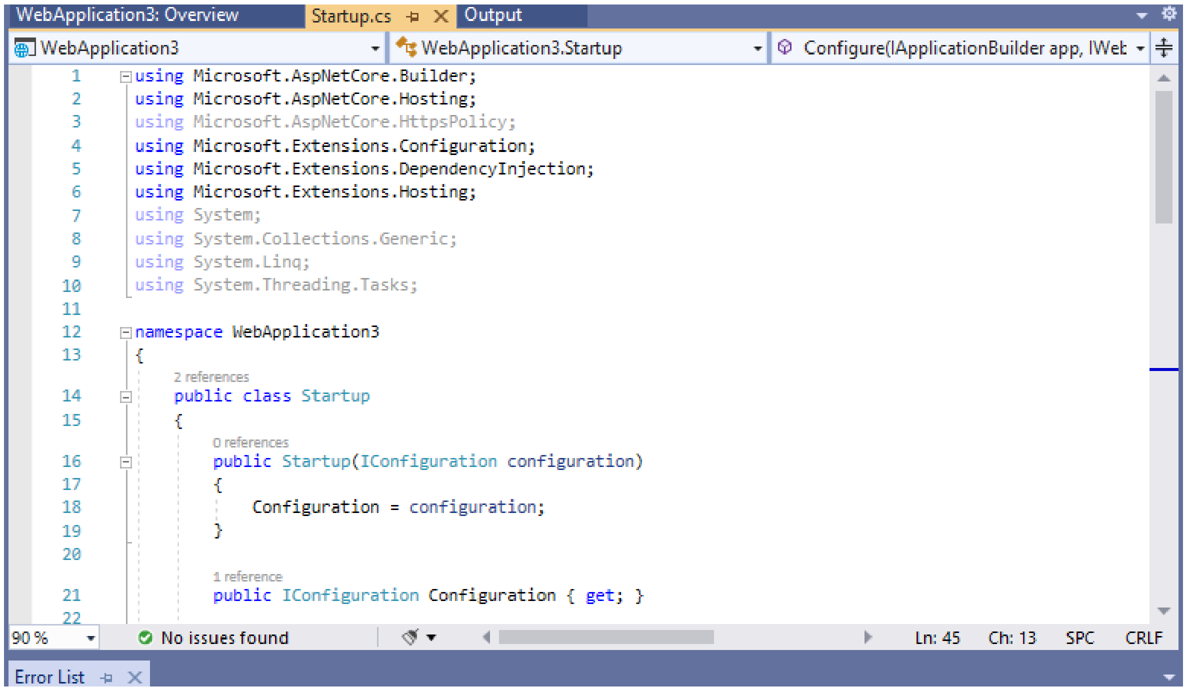
6. Please open your Startup.cs file.


You can improve 'Startup' performance by auto hiding or collapsing 'Cloud Explorer' window. Manage performance Don't show this message again

ASP.NET Core

Learn about the .NET platform, create your first application and extend it to the cloud.

7. When you open the startup class, your window should be looking like this:


WebApplication3: Overview | Startup.cs | Output

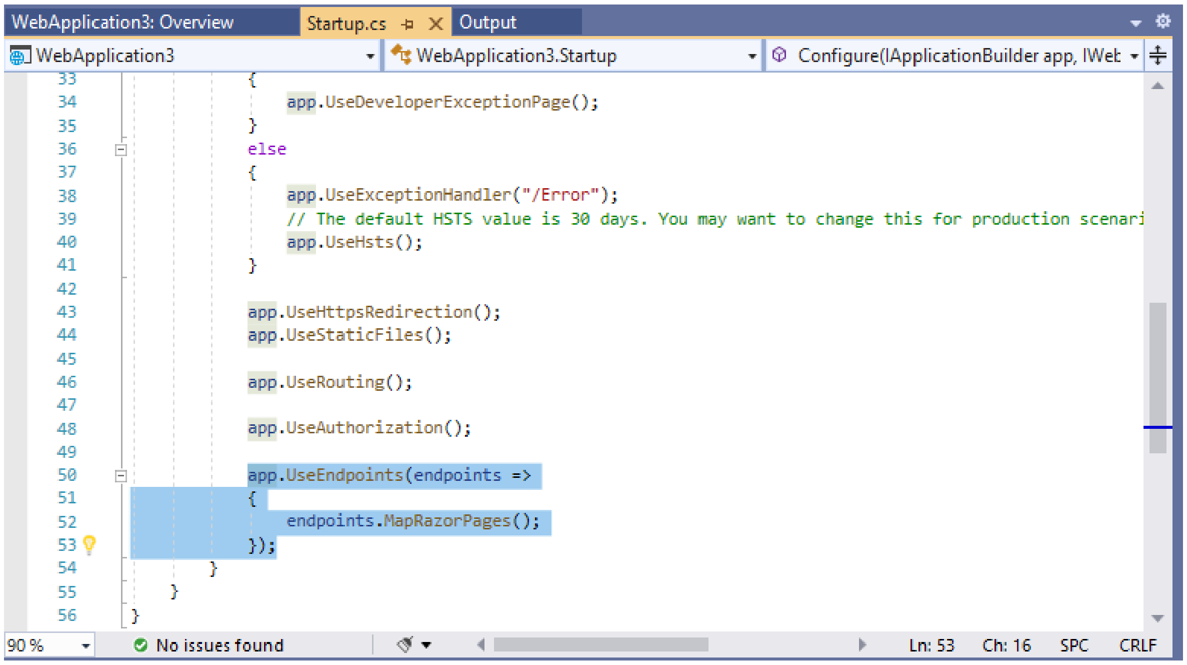
WebApplication3 | WebApplication3.Startup | Configure(IApplicationBuilder app, IWeb

```
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.AspNetCore.HttpsPolicy;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace WebApplication3
{
    2 references
    public class Startup
    {
        0 references
        public Startup(IConfiguration configuration)
        {
            Configuration = configuration;
        }

        1 reference
        public IConfiguration Configuration { get; }
```

90 % | No issues found | Ln: 45 Ch: 13 SPC CRLF

Error List

8. We´re going to configure the endpoint in this file, so please scroll all the way down until you find the following code:


WebApplication3: Overview | Startup.cs | Output

WebApplication3 | WebApplication3.Startup | Configure(IApplicationBuilder app, IWeb

```
            {
                app.UseDeveloperExceptionPage();
            }
            else
            {
                app.UseExceptionHandler("/Error");
                // The default HSTS value is 30 days. You may want to change this for production scenari
                app.UseHsts();
            }

            app.UseHttpsRedirection();
            app.UseStaticFiles();

            app.UseRouting();

            app.UseAuthorization();

            app.UseEndpoints(endpoints =>
            {
                endpoints.MapRazorPages();
            });
        }
    }
}
```

90 % | No issues found | Ln: 53 Ch: 16 SPC CRLF

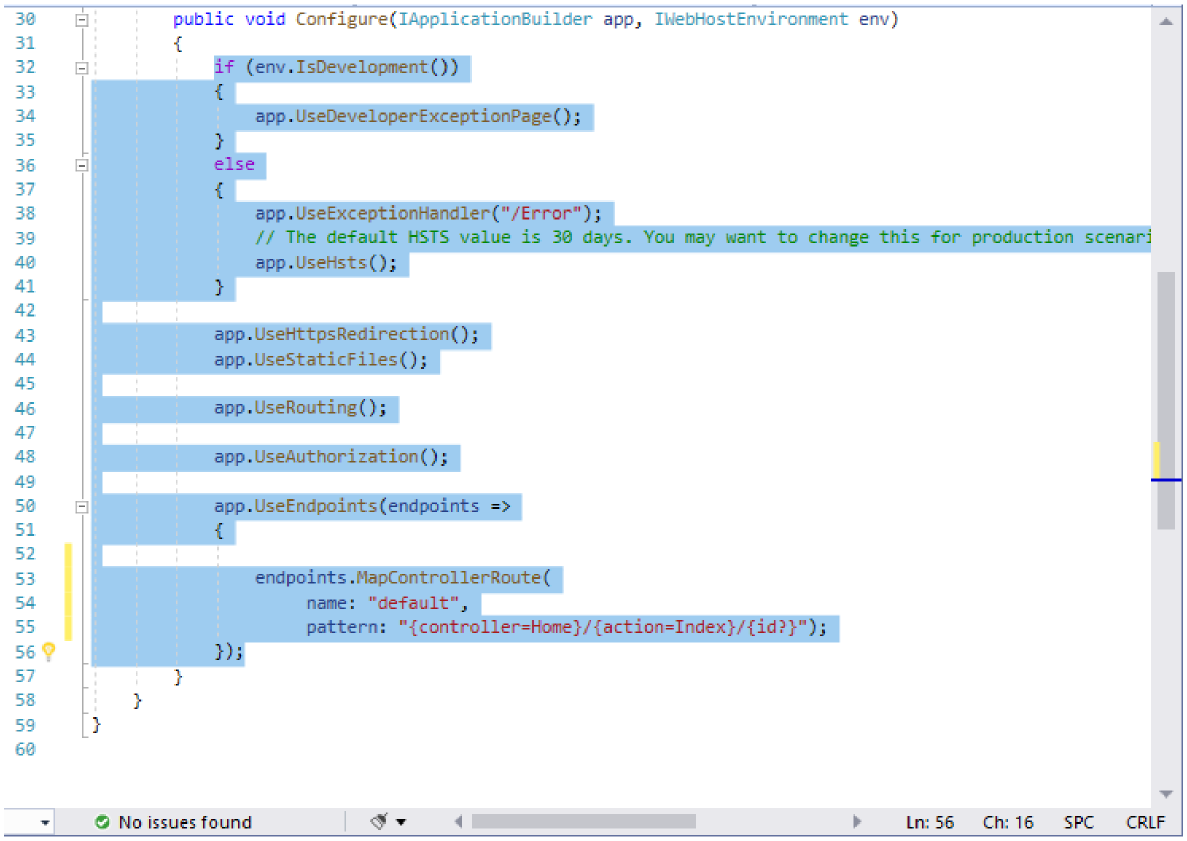
9. The highlighted method above, adds the default MVC Conventional route using the MapControllerRoute method. It also adds the default conventional route to

MVC Controllers using a pattern.

10. To configure your endpoint please look at the high lighted method above and change the instructions inside that method for the following endpoint configuration:

```
endpoints.MapControllerRoute(
name: "default",
pattern: "{controller=Home}/{action=Index}/{id?}");
```

11. It´s time to action! Please scroll up to where your main method is and change the the highlighted text:



```
        public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
        {
            if (env.IsDevelopment())
            {
                app.UseDeveloperExceptionPage();
            }
            else
            {
                app.UseExceptionHandler("/Error");
                // The default HSTS value is 30 days. You may want to change this for production scenari
                app.UseHsts();
            }

            app.UseHttpsRedirection();
            app.UseStaticFiles();

            app.UseRouting();

            app.UseAuthorization();

            app.UseEndpoints(endpoints =>
            {

                endpoints.MapControllerRoute(
                    name: "default",
                    pattern: "{controller=Home}/{action=Index}/{id?}");
            });
        }
    }
}
```

No issues found   Ln: 56   Ch: 16   SPC   CRLF

In this example we´ll use MapGet method to create 2 custom endpointsPlease paste the following code where the highlighted area:

```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
     {

         //All Middleware till here cannot access the Endpoint
```

```
                app.UseRouting();


                //All middleware from here onwards can access the Endpoint from the
HTTP Context

                app.UseEndpoints(endpoints =>
                {


                    endpoints.MapGet("/", async context =>
                    {
                        await context.Response.WriteAsync("Hello World");
                    });

                    endpoints.MapGet("/hello", async context =>
                    {
                        await context.Response.WriteAsync(helloWorld());
                    });


                   });
```

12. Any Middleware between the UseRouting and UseEndpoints can access the the Endpoint from the context using the GetEndpoint method.

13. In the following code adds a custom middleware after UseRouting. If Endpoint is not null, it extracts the DisplayName, RoutePattern & metdata from the Endpoint and writes it to the response stream.

```
        Public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
          {
              //All Middleware till here cannot access the Endpoint

              app.UseRouting();

              app.Use(async (context, next) =>
              {
                  var endpoint = context.GetEndpoint();

                  if (endpoint != null)
                  {
                      await context.Response.WriteAsync("<html> Endpoint :" +
          endpoint.DisplayName + " <br>");

                      if (endpoint is RouteEndpoint routeEndpoint)
                      {

                          await context.Response.WriteAsync("RoutePattern :"
          + routeEndpoint.RoutePattern.RawText + " <br>");

                      }

                      foreach (var metadata in endpoint.Metadata)
                      {
```

```
                    await context.Response.WriteAsync("metadata : " +
metadata + " <br>");

                }

            }
            else
            {
                await context.Response.WriteAsync("End point is null");
            }

            await context.Response.WriteAsync("</html>");
            await next();
        });



        app.UseEndpoints(endpoints =>
        {


            endpoints.MapGet("/", async context =>
            {
                await context.Response.WriteAsync("Hello World");
            });

            endpoints.MapGet("/hello", async context =>
            {
                await context.Response.WriteAsync(helloWorld());
            });

        });
       }
```

Module 02: Template Inheritance

Lab Setup

Estimated time: **50-60 min.**

Introduction

When we use a Web Framework such as ASP NET it is very common to use base a template that can be reused across multiple pages in our website. In this lab we are going to create different pages that will be shared the same base structure. At the same time we will modify the style that is provided by default.

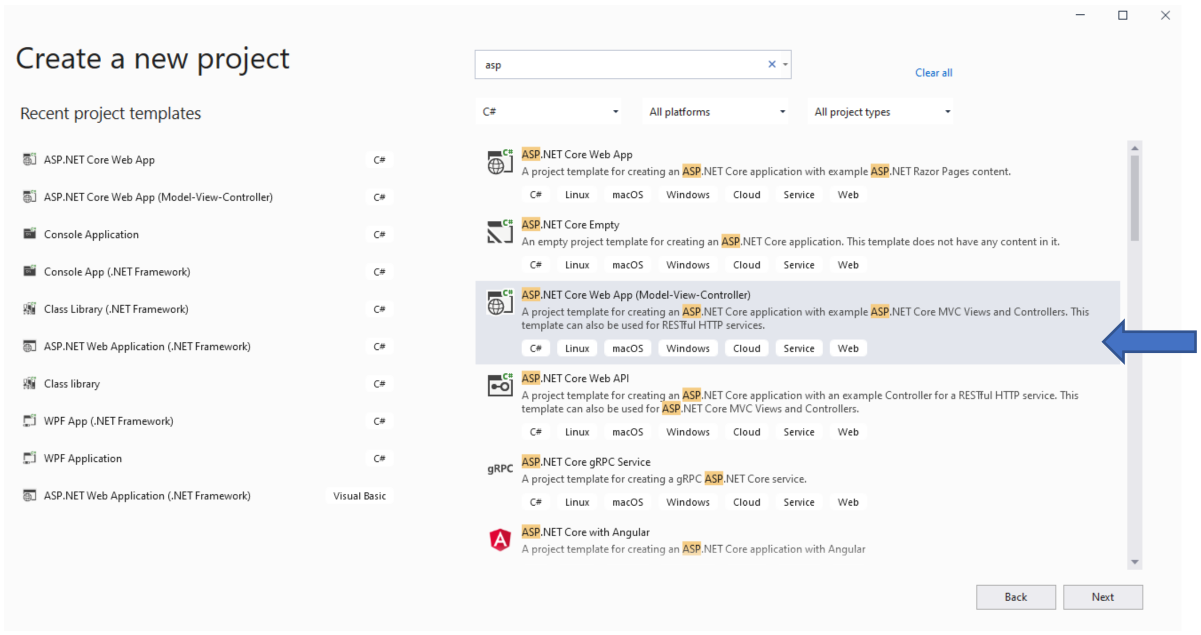
Objectives

In this lab we will use template inheritance in ASP NET and we will modify the style of our website.

a. Use template inheritance
b. Modify the style of the base template

Exercise 01: Create the project

1. Open Visual Studio 2019.
   Create a new Project. Select "ASP.NET Core Web App (Model-View-Controller)"



Create a new project

Recent project templates

ASP.NET Core Web App	C#
ASP.NET Core Web App (Model-View-Controller)	C#
Console Application	C#
Console App (.NET Framework)	C#
Class Library (.NET Framework)	C#
ASP.NET Web Application (.NET Framework)	C#
Class library	C#
WPF App (.NET Framework)	C#
WPF Application	C#
ASP.NET Web Application (.NET Framework)	Visual Basic

asp

Clear all

C# · All platforms · All project types

ASP.NET Core Web App
A project template for creating an ASP.NET Core application with example ASP.NET Razor Pages content.
C# Linux macOS Windows Cloud Service Web

ASP.NET Core Empty
An empty project template for creating an ASP.NET Core application. This template does not have any content in it.
C# Linux macOS Windows Cloud Service Web

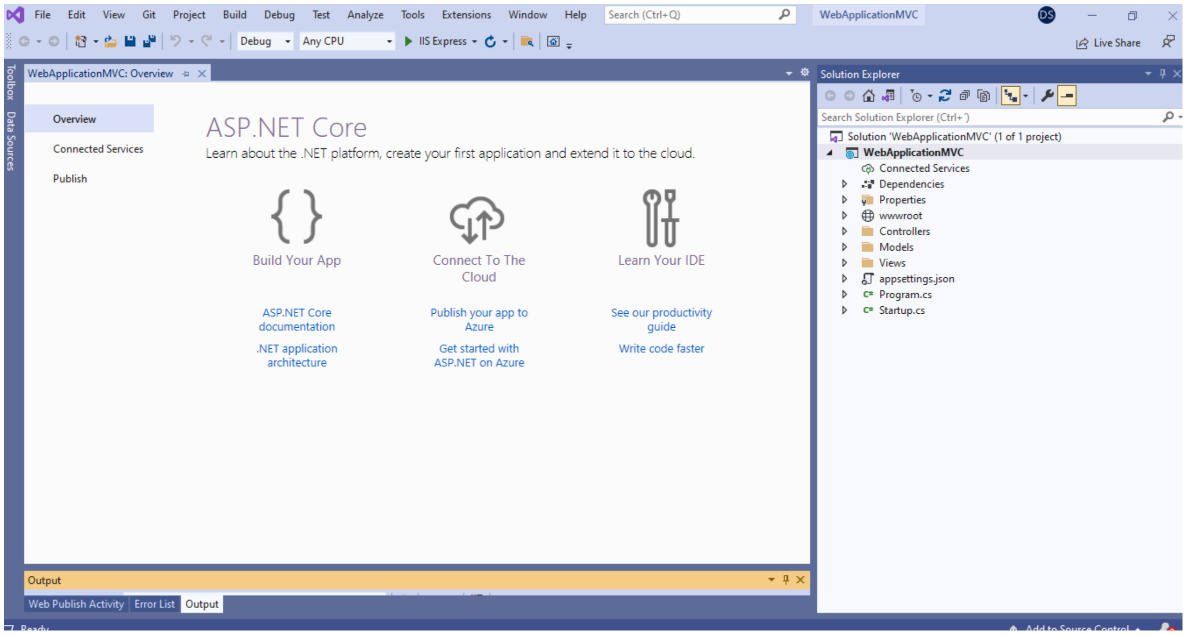
ASP.NET Core Web App (Model-View-Controller)
A project template for creating an ASP.NET Core application with example ASP.NET Core MVC Views and Controllers. This template can also be used for RESTful HTTP services.
C# Linux macOS Windows Cloud Service Web

ASP.NET Core Web 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.
C# Linux macOS Windows Cloud Service Web

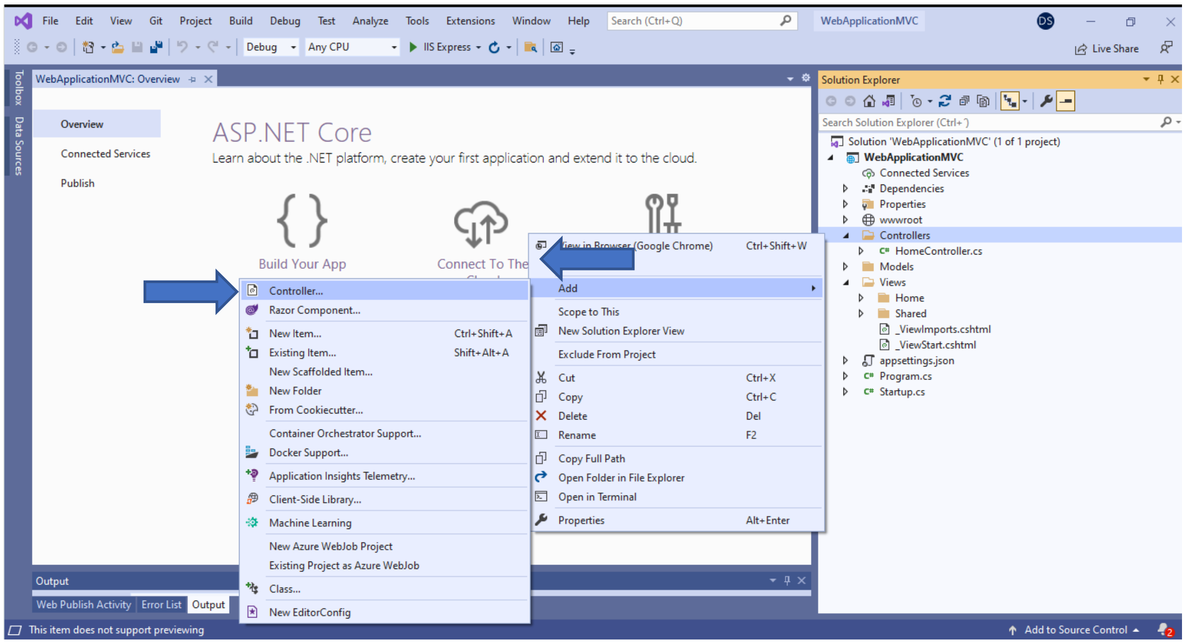
ASP.NET Core gRPC Service
A project template for creating a gRPC ASP.NET Core service.
C# Linux macOS Windows Cloud Service Web

ASP.NET Core with Angular
A project template for creating an ASP.NET Core application with Angular

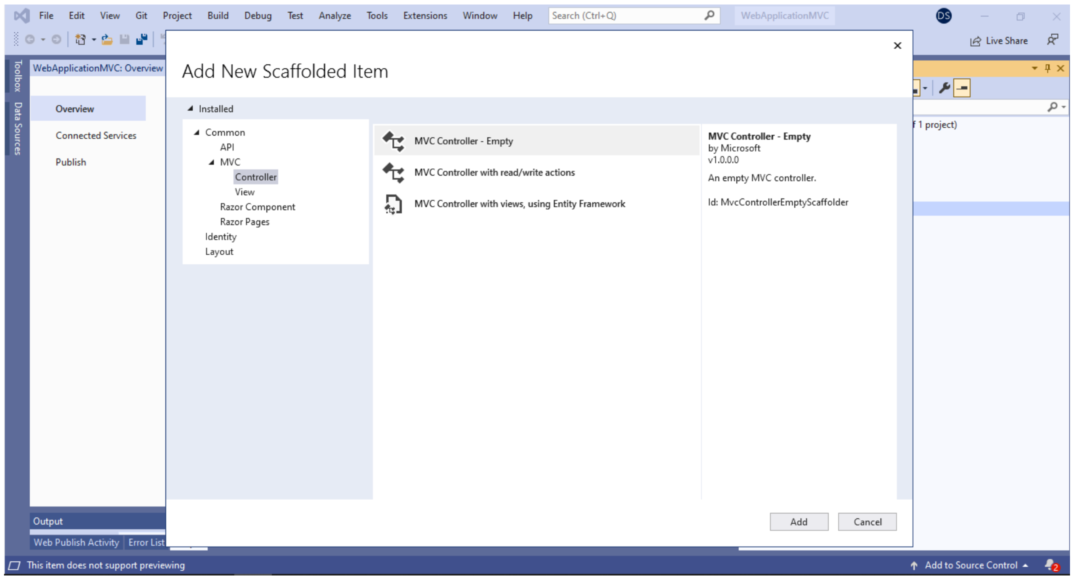
2. Choose a name for your project and click next.
3. Leave the default options and click next.
4. When the new project opens it, your Visual Studio window will look like this:

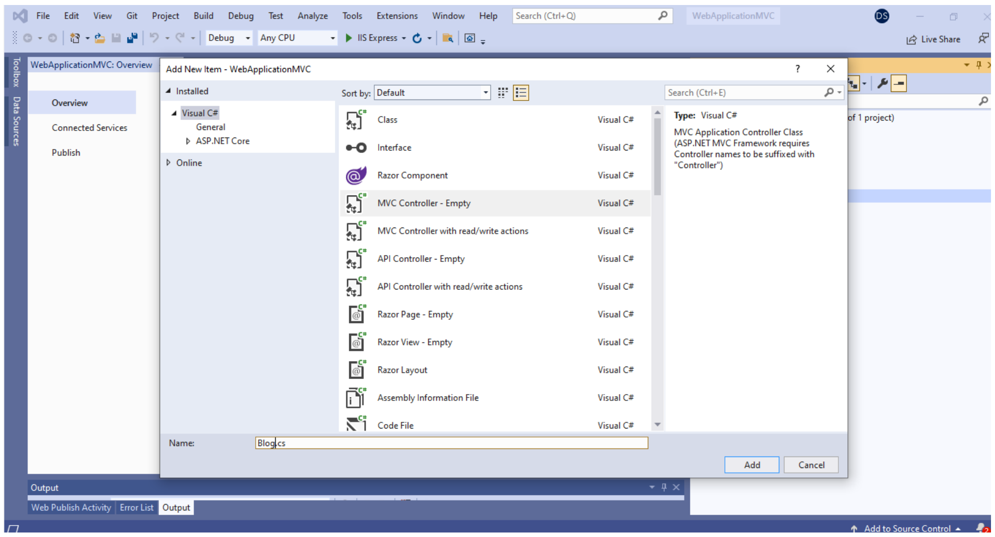


5. Note that different files and folders has been created.
6. On this lab we will focus on the following folders:
    a. Controllers
    b. Views
7. By default, the project contains just one controller and a Home folder.
8. Right click on Controllers Folder and then click “Add Controller”



9. Select “MVC Controller – Empty”
10. Click “Add” and provide the name “Blog.cs”. It is important to mention you can provide any name you want, however make sure to follow the convention name.





Add New Item - WebApplicationMVC

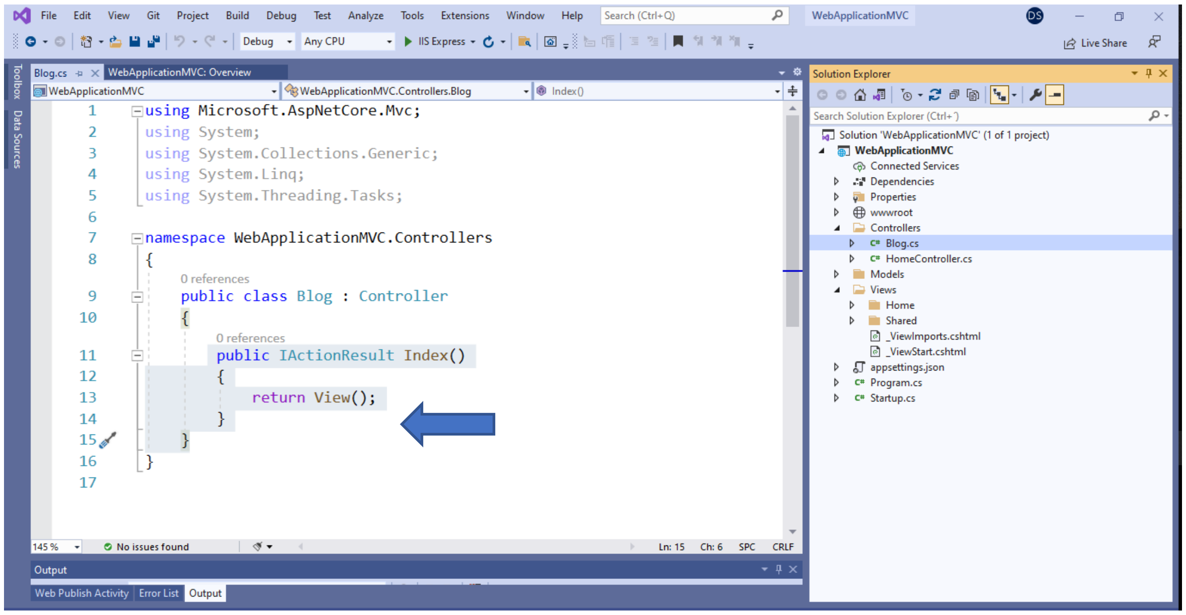
Class	Visual C#
Interface	Visual C#
Razor Component	Visual C#
MVC Controller - Empty	Visual C#
MVC Controller with read/write actions	Visual C#
API Controller - Empty	Visual C#
API Controller with read/write actions	Visual C#
Razor Page - Empty	Visual C#
Razor View - Empty	Visual C#
Razor Layout	Visual C#
Assembly Information File	Visual C#
Code File	Visual C#

Type: Visual C#

MVC Application Controller Class (ASP.NET MVC Framework requires Controller names to be suffixed with "Controller")

Name: Blog.cs

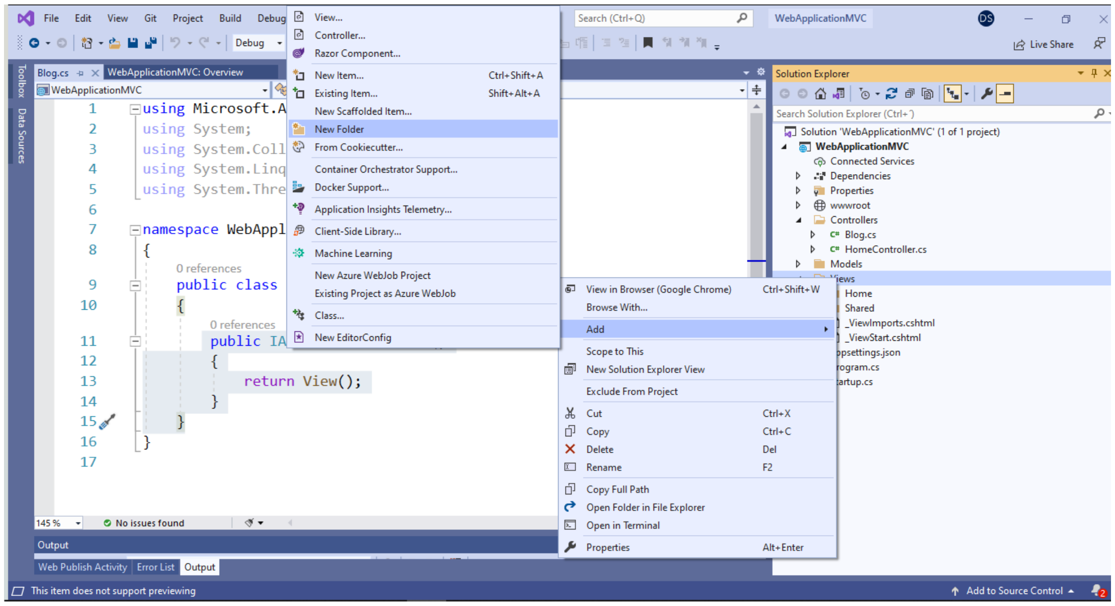
11. Once we click in "Add", a new class (Blog.cs) will be created inside "Controllers" folder. Inside the file Blog.cs we will find a method named "Index". This means that if we enter the url "mywebsit/blog" this will be the method that will be executed. However we haven't provided a View (HTML Code) to return in this method.



```
using Microsoft.AspNetCore.Mvc;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace WebApplicationMVC.Controllers
{
    0 references
    public class Blog : Controller
    {
        0 references
        public IActionResult Index()
        {
            return View();
        }
    }
}
```

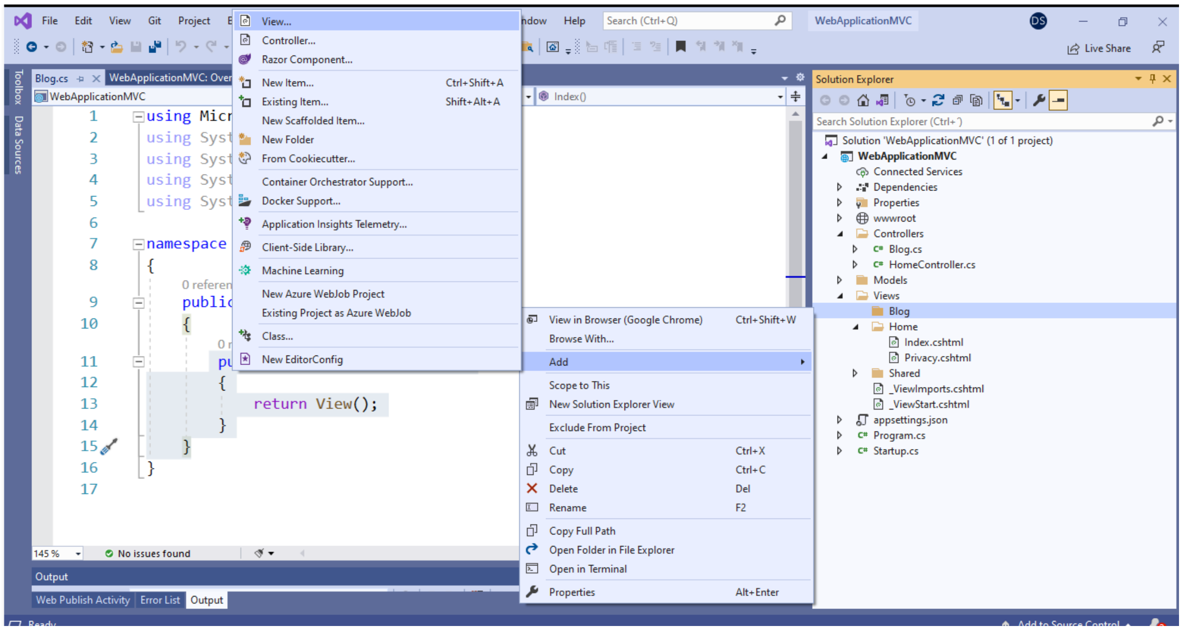
Solution 'WebApplicationMVC' (1 of 1 project)
- WebApplicationMVC
  - Connected Services
  - Dependencies
  - Properties
  - wwwroot
  - Controllers
    - Blog.cs
    - HomeController.cs
  - Models
  - Views
    - Home
    - Shared
    - _ViewImports.cshtml
    - _ViewStart.cshtml
  - appsettings.json
  - Program.cs
  - Startup.cs

12. Now let's add the HTML code that we want to return. Right click on "Views" folder and create a new folder with the name "Blog"

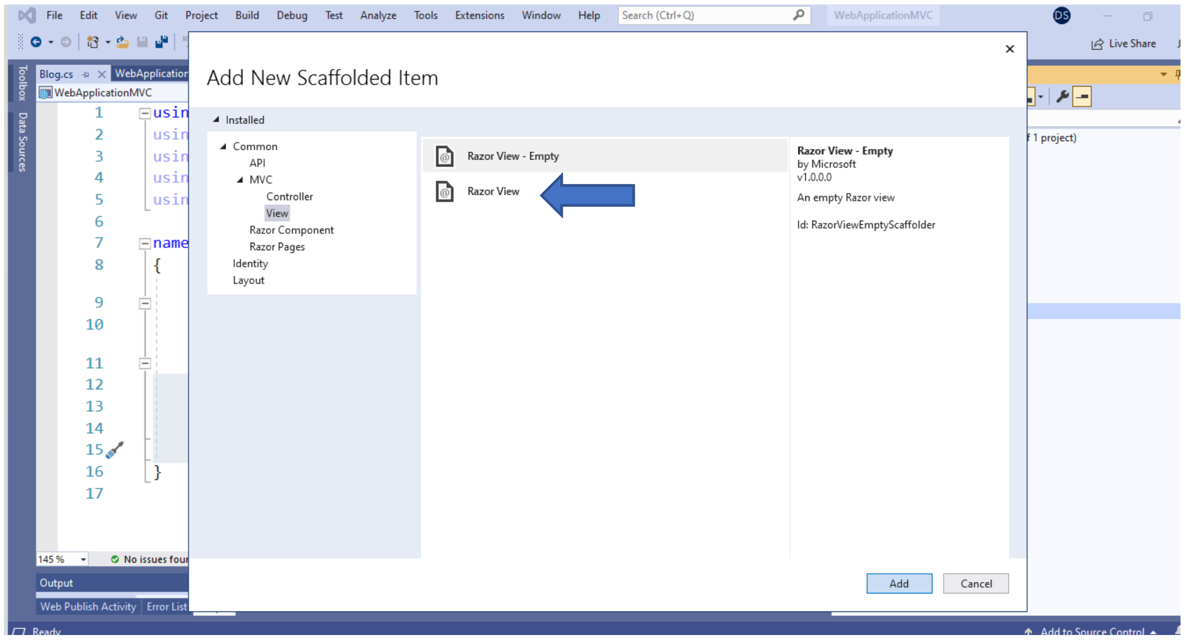


```
using Microsoft.A
using System;
using System.Coll
using System.Linq
using System.Thre

namespace WebAppl
{
    0 references
    public class
    {
        0 references
        public IA
        {
            return View();
        }
    }
}
```

13. Inside the “Blog” folder let’s add a new view. Select “Razor View” with the name “Index.cshtml”



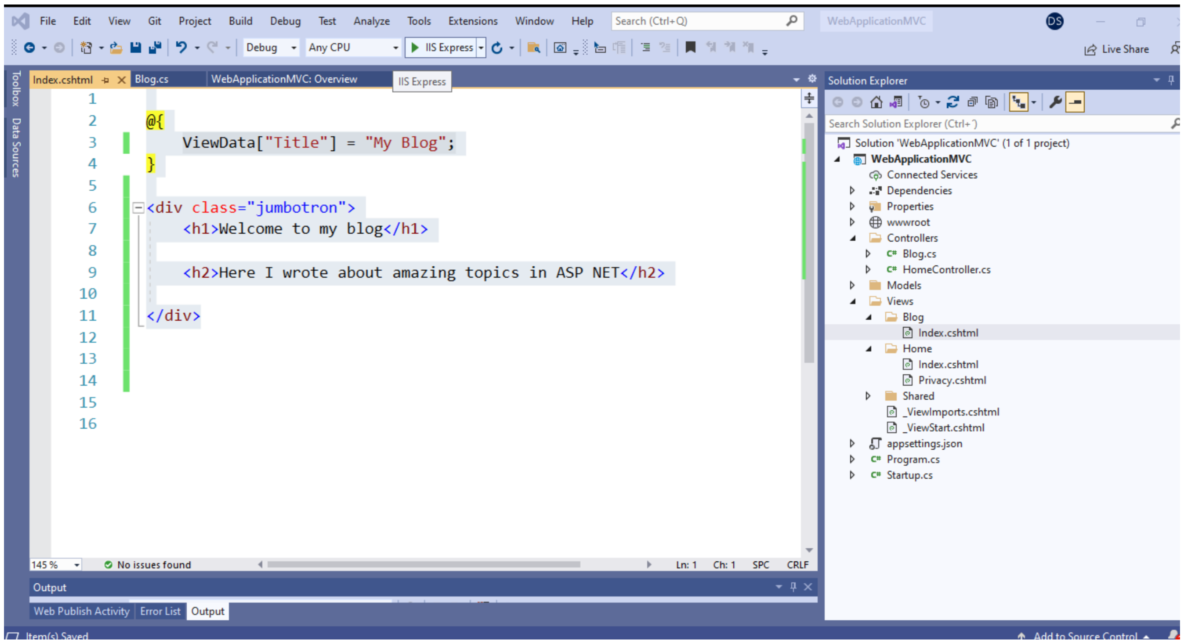
```
using Micr
using Syst
using Syst
using Syst
using Syst

namespace
{
    0 referen
    public
    {
        0 r
        pu
        {
            return View();
        }
    }
}
```



14. Once the Index.cshtml file has been created inside the Blog Folder. Open the file and replace the entire code with the following code:

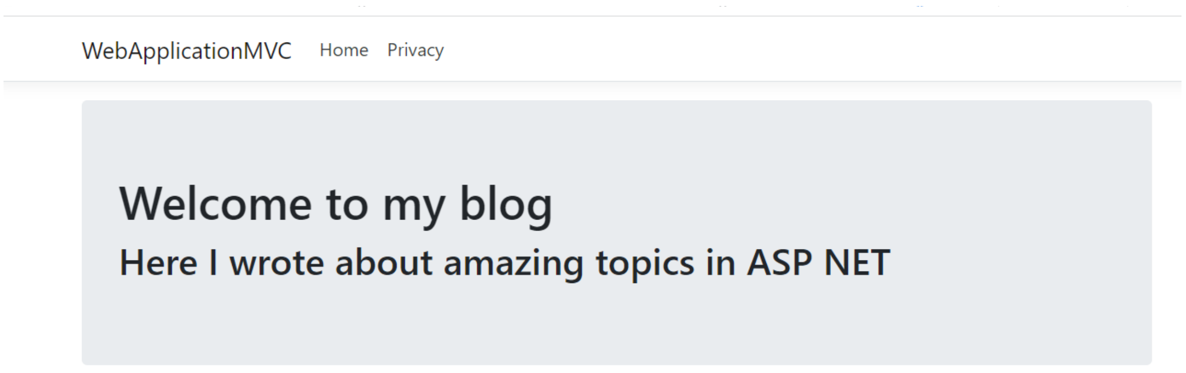
```
@{
    ViewData["Title"] = "My Blog";
}

<div class="jumbotron">
    <h1>Welcome to my blog</h1>

    <h2>Here I wrote about amazing topics in ASP NET</h2>

      </div>
```

15. Once you update the content of this file. Click on “Run” button and wait until the web application runs on your default browser.



```

@{
    ViewData["Title"] = "My Blog";
}

<div class="jumbotron">
    <h1>Welcome to my blog</h1>

    <h2>Here I wrote about amazing topics in ASP NET</h2>

</div>
```

16. Once the browser is open, in the url append “/blog” and you will see the web page rendered.

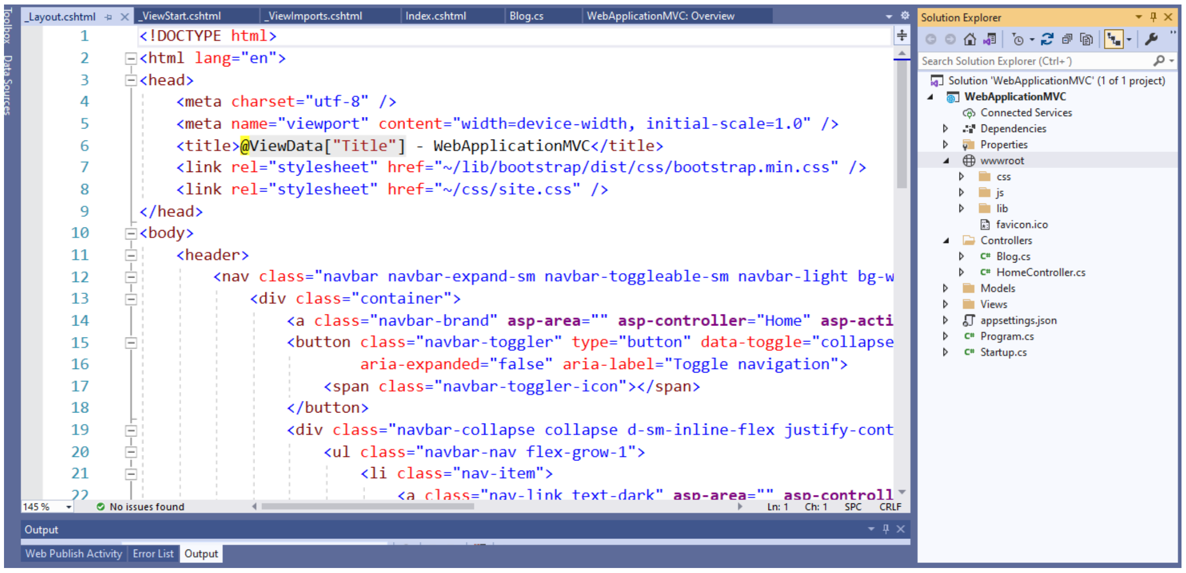


WebApplicationMVC Home Privacy

Welcome to my blog

Here I wrote about amazing topics in ASP NET

17. If you notice we didn’t change the navbar and the footer because they’re already provided. We are inheriting from a template you can find at Views/Shared/_Layout.cshtml
18. Open that file and take a look.
19. This file contains some references you can find at wwwroot folder.

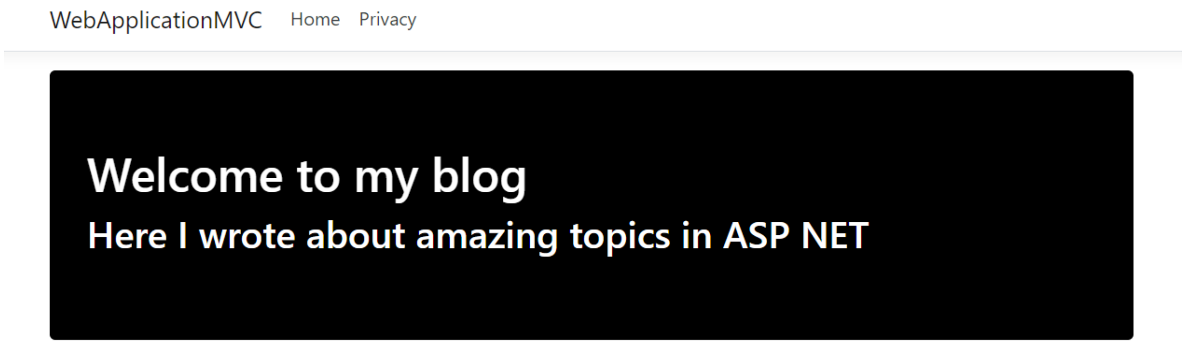


```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>@ViewData["Title"] - WebApplicationMVC</title>
    <link rel="stylesheet" href="~/lib/bootstrap/dist/css/bootstrap.min.css" />
    <link rel="stylesheet" href="~/css/site.css" />
</head>
<body>
    <header>
        <nav class="navbar navbar-expand-sm navbar-toggleable-sm navbar-light bg-w
            <div class="container">
                <a class="navbar-brand" asp-area="" asp-controller="Home" asp-acti
                <button class="navbar-toggler" type="button" data-toggle="collapse
                        aria-expanded="false" aria-label="Toggle navigation">
                    <span class="navbar-toggler-icon"></span>
                </button>
                <div class="navbar-collapse collapse d-sm-inline-flex justify-cont
                    <ul class="navbar-nav flex-grow-1">
                        <li class="nav-item">
                            <a class="nav-link text-dark" asp-area="" asp-controll
```

20. With this base you can add more pages to your website that will reuse the base Layout template. This is very useful if you want to reuse your navbar and the footer or other common components across your application.
21. Now let’s change the style. Open the site.css (wwroot/css/site.css) file and let’s modify the CSS.
22. At the bottom of the file add the following code:

```
.jumbotron{
  background: #000000;
 color: #FFFFFF;
   }
```

23. This will change the aspect of the jumbotron class. Open again your website and go to your blog page that the style has changed:



Welcome to my blog

Here I wrote about amazing topics in ASP NET

24. If you don’t see the change, try to press Ctrl+F5 at the same time in your keyboard.

Module 03: CRUD in ASP NET

Lab Setup

Estimated time: **50-60 min.**

Introduction

When we use a Web Framework such as ASP NET it is very common to use some operations that allow us to interact with a Database. Normally these operations are known as CRUD operations, which stands for: Create, Retrieve, Update and Delete. Each of these operations will be related to a specific table in an existing Database. Entity Framework is a Package that can be used to automatically create the Tables in a Database and the views to create these operations.

For this lab you must have installed Visual Studio 2019 (or higher) and SQL Server Studio

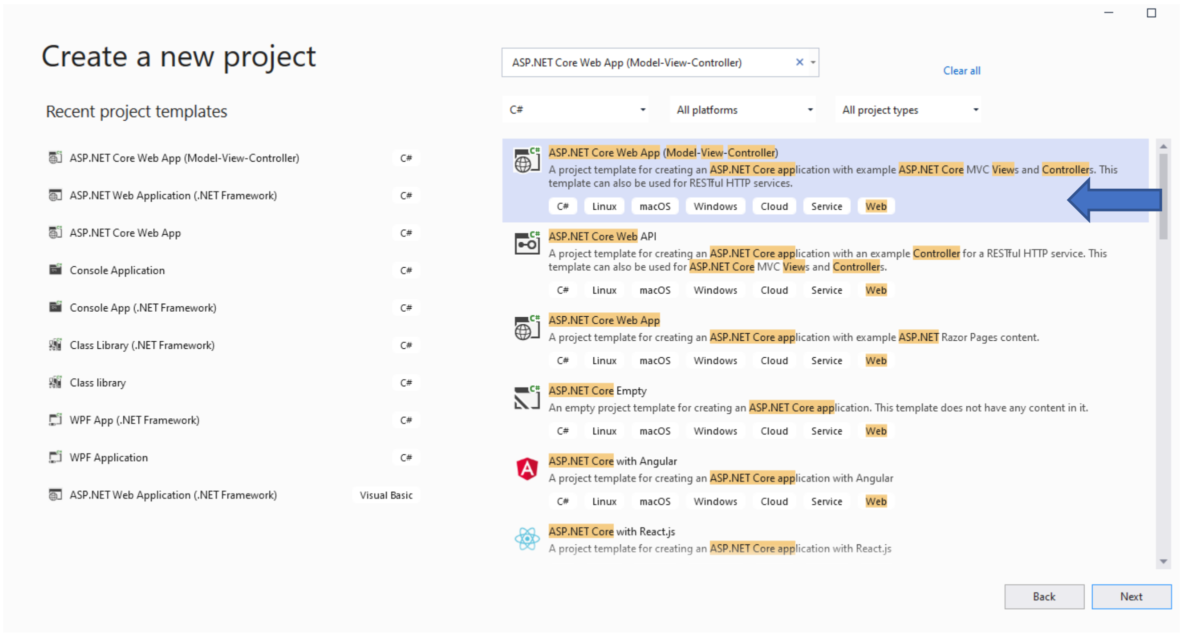
Objectives

In this lab we will implement CRUD operations in an ASP NET application.

a. Create an ASP Net Application
b. Create the Model
c. Create the tables from the Database

Exercise 01: Create the project

1. Open Visual Studio 2019.
   Create a new Project. Select "ASP.NET Core Web App (Model-View-Controller)"



Create a new project

Recent project templates

ASP.NET Core Web App (Model-View-Controller)	C#
ASP.NET Web Application (.NET Framework)	C#
ASP.NET Core Web App	C#
Console Application	C#
Console App (.NET Framework)	C#
Class Library (.NET Framework)	C#
Class library	C#
WPF App (.NET Framework)	C#
WPF Application	C#
ASP.NET Web Application (.NET Framework)	Visual Basic

ASP.NET Core Web App (Model-View-Controller)

Clear all

C# | All platforms | All project types

ASP.NET Core Web App (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.
C# Linux macOS Windows Cloud Service Web

ASP.NET Core Web 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.
C# Linux macOS Windows Cloud Service Web

ASP.NET Core Web App
A project template for creating an ASP.NET Core application with example ASP.NET Razor Pages content.
C# Linux macOS Windows Cloud Service Web

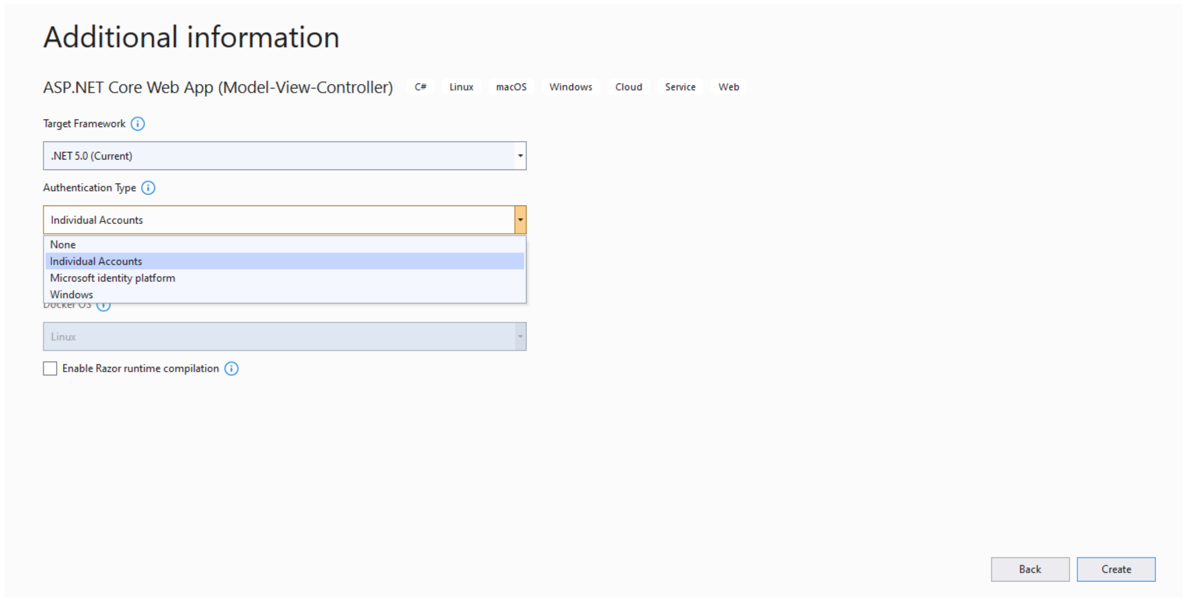
ASP.NET Core Empty
An empty project template for creating an ASP.NET Core application. This template does not have any content in it.
C# Linux macOS Windows Cloud Service Web

ASP.NET Core with Angular
A project template for creating an ASP.NET Core application with Angular
C# Linux macOS Windows Cloud Service Web

ASP.NET Core with React.js
A project template for creating an ASP.NET Core application with React.js

Back | Next

3. Select Next and then change the Authentication Type to “Individual Accounts”


Additional information

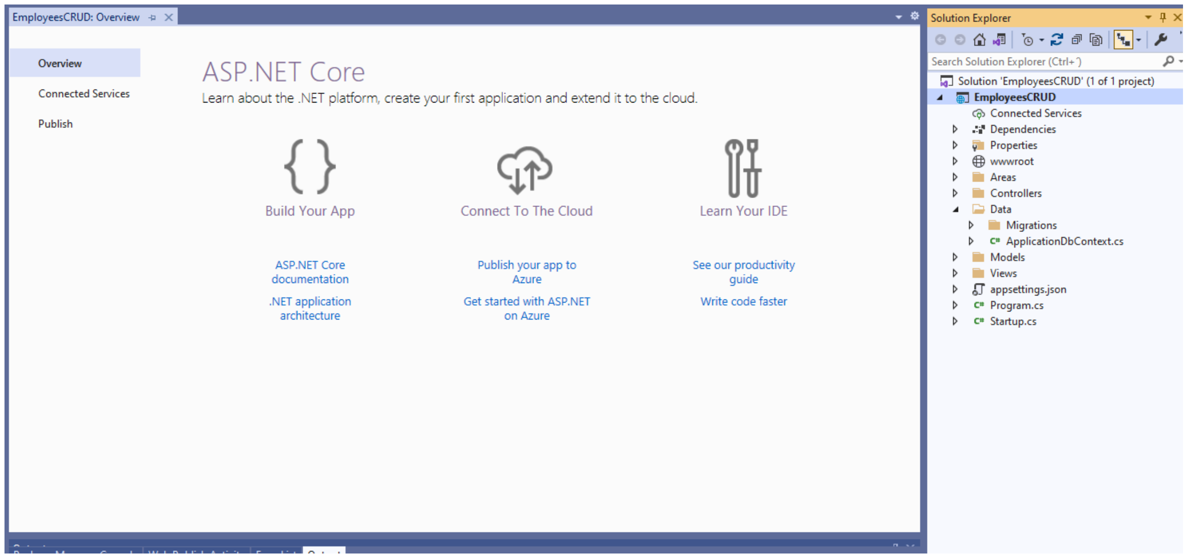
ASP.NET Core Web App (Model-View-Controller)

Target Framework: .NET 5.0 (Current)

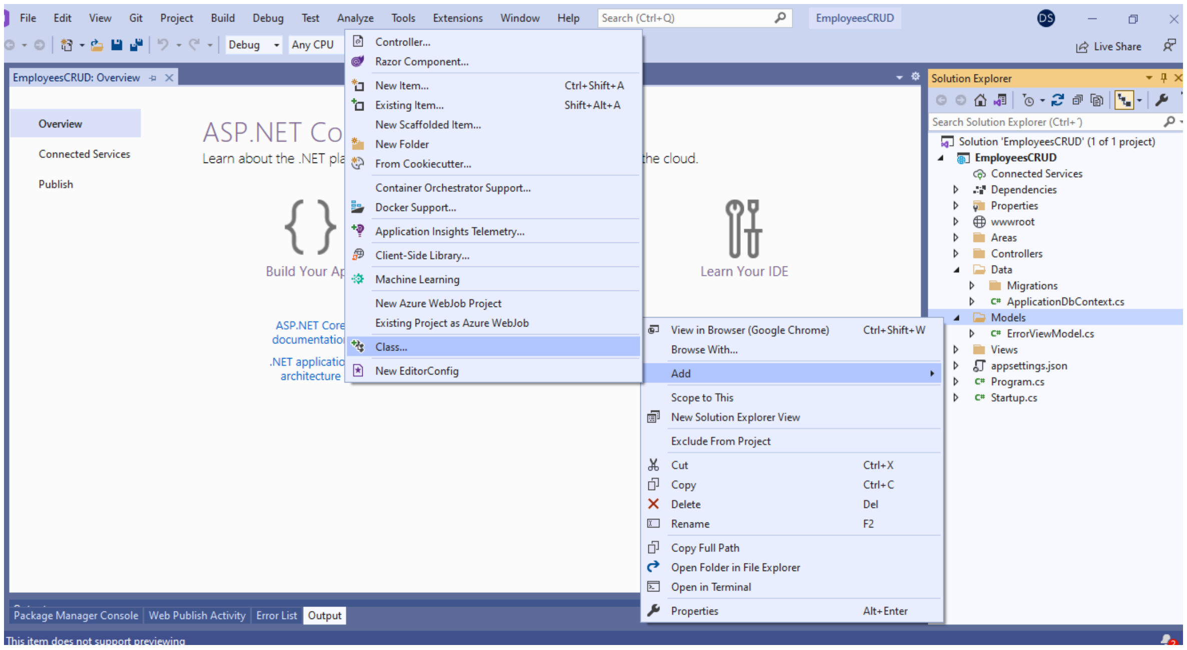
Authentication Type: Individual Accounts

Enable Razor runtime compilation

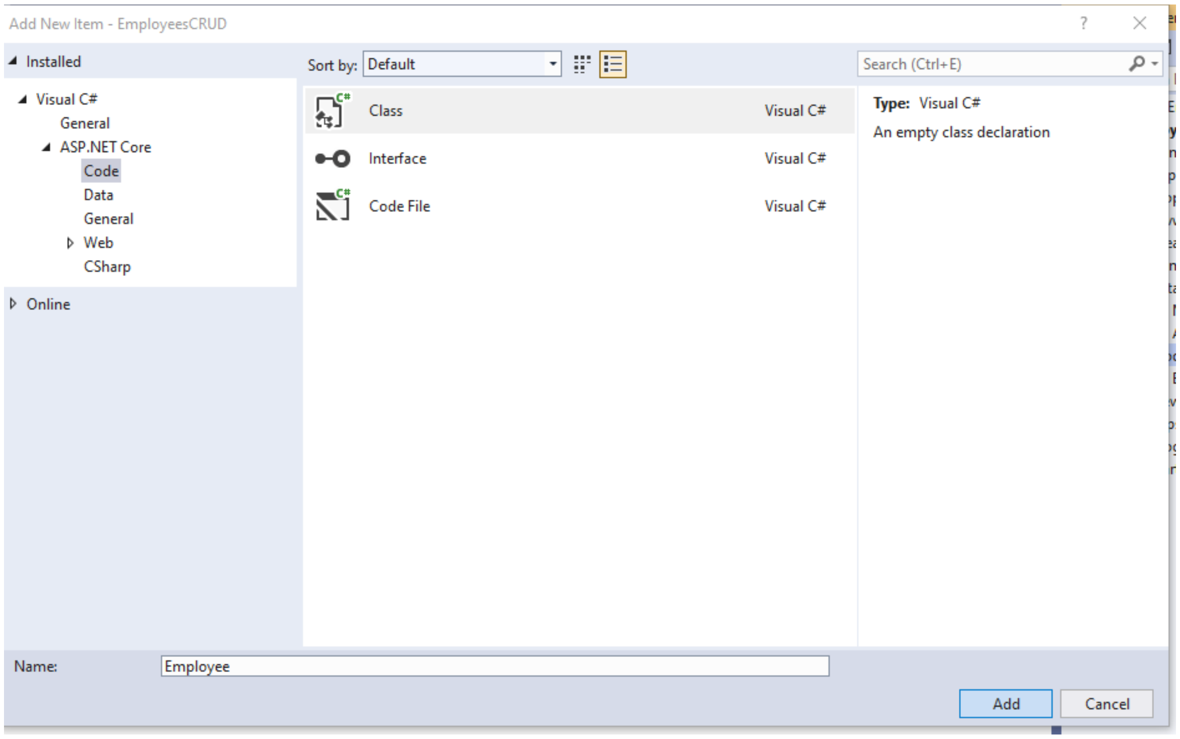
4. Select Next and then change the Authentication Type to “Individual Accounts”.
5. This will create the project with some files and folders. As MVC pattern we have the Model, Views and Controller folders. Additionally other folder named “Data” with Migrations and a class have been provided.



6. Let’s create our model for this application. Right click on Models, then select “Class”



7. Make sure “Class” option is select. Enter the name “Employee” and then click on Add.

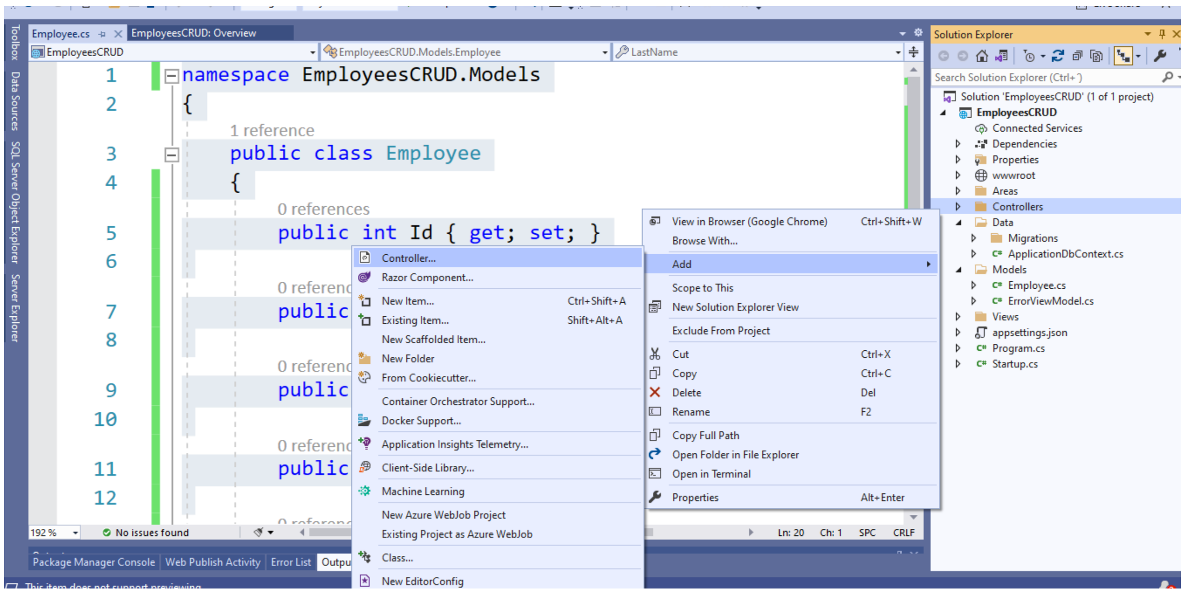


8. Make sure “Class” option is select. Enter the name “Employee” and then click on Add.
9. This will create a new file inside Models with the name “Employee.cs” open it and replace the entire code with the following code:

```
namespace EmployeesCRUD.Models
```

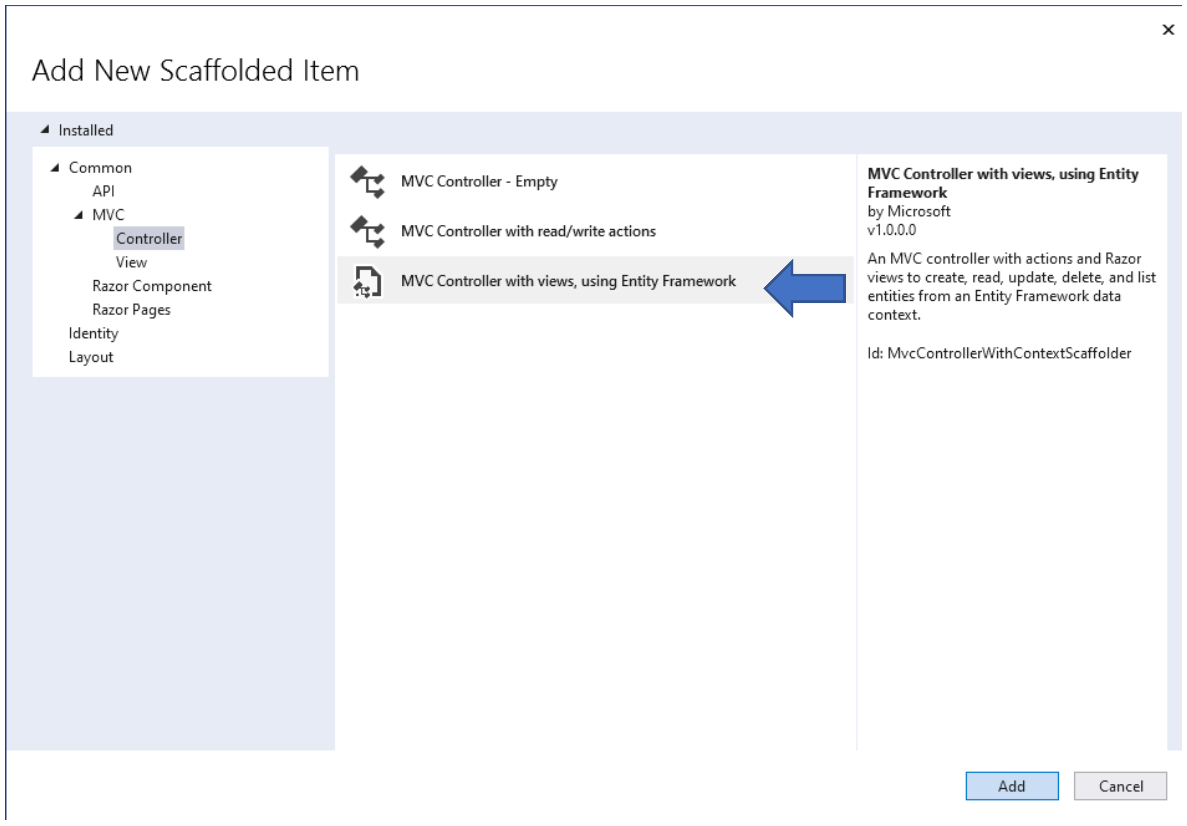
```
{
    public class Employee
    {
        public int Id { get; set; }

        public string FirstName { get; set; }

        public string LastName { get; set; }

        public string JobPosition { get; set; }

        public Employee()
        {

        }

    }
}
```

10. Once this model has been created, let’s add the Controller. Right click on Controllers Folder and select “Add Controller”



```
namespace EmployeesCRUD.Models
{
    1 reference
    public class Employee
    {
        0 references
        public int Id { get; set; }

        0 referenc
        public

        0 referenc
        public

        0 referenc
        public
```

11. Select "MVC Controller with views, using Entity Framework"

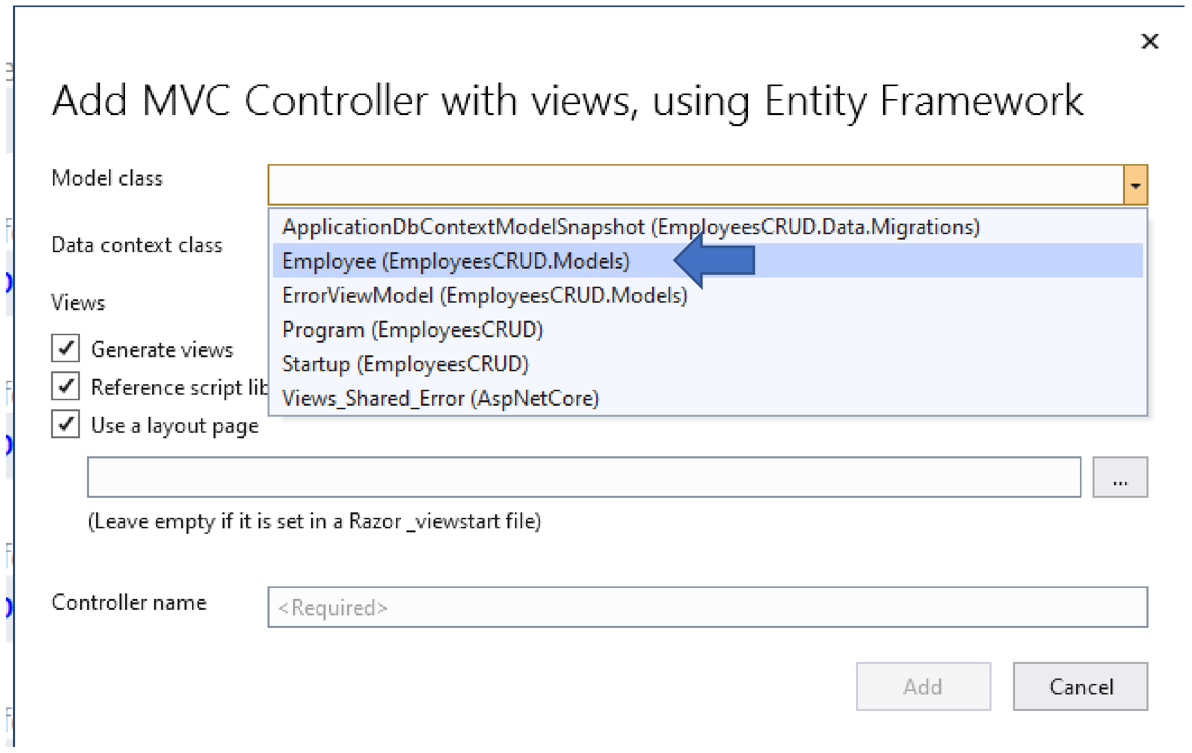


MVC Controller with views, using Entity Framework
by Microsoft
v1.0.0.0

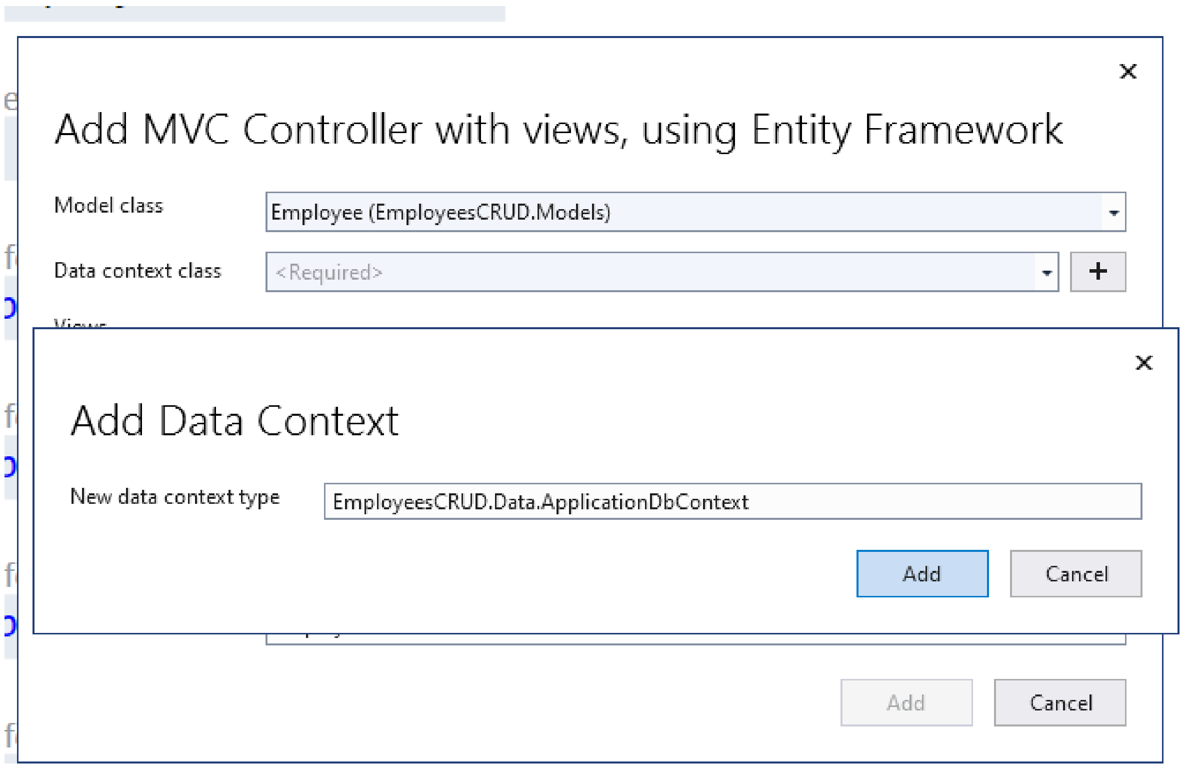
An MVC controller with actions and Razor views to create, read, update, delete, and list entities from an Entity Framework data context.

Id: MvcControllerWithContextScaffolder

12. In the Model Class dropdown menu select the Employee class we created before.


Add MVC Controller with views, using Entity Framework
Model class
Data context class
Views
Generate views
Reference script lib
Use a layout page
ApplicationDbContextModelSnapshot (EmployeesCRUD.Data.Migrations)
Employee (EmployeesCRUD.Models)
ErrorViewModel (EmployeesCRUD.Models)
Program (EmployeesCRUD)
Startup (EmployeesCRUD)
Views_Shared_Error (AspNetCore)
...
(Leave empty if it is set in a Razor _viewstart file)
Controller name
<Required>
Add
Cancel

13. Add a new Data Context and enter the name “EmployeesCRUD.Data.ApplicationDbContext”.



Add MVC Controller with views, using Entity Framework

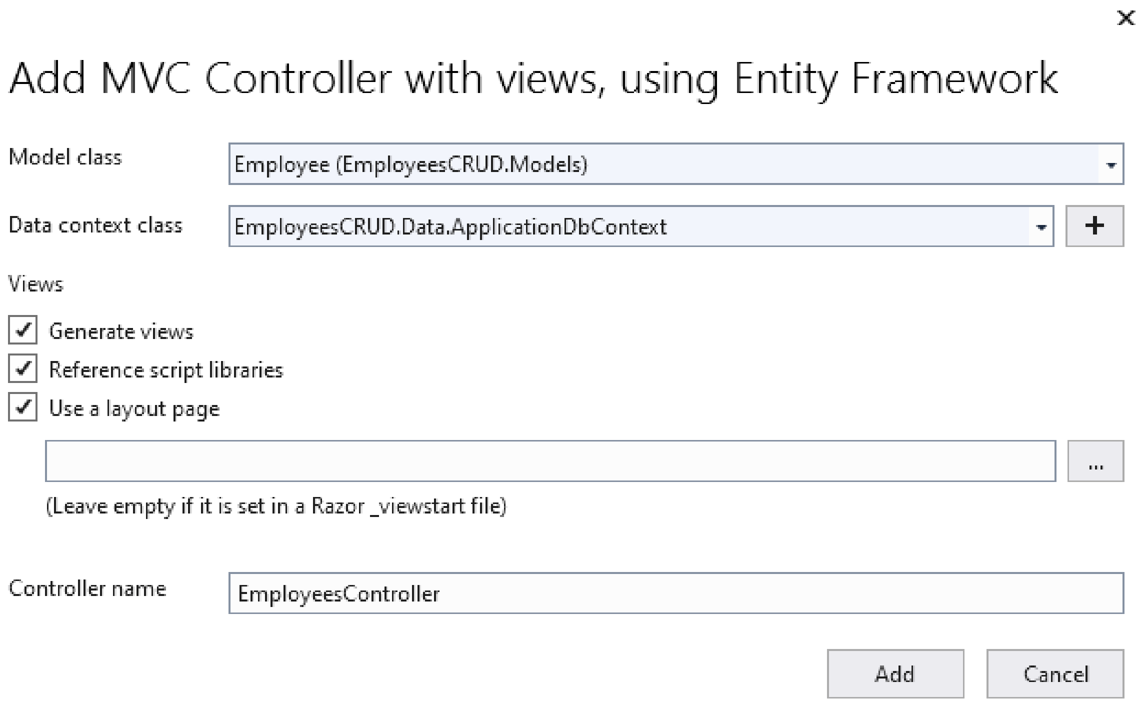
Model class: Employee (EmployeesCRUD.Models)

Data context class: <Required>

Add Data Context

New data context type: EmployeesCRUD.Data.ApplicationDbContext

14. Make sure all the Views options are selected (Generate Views, Reference script libraries, Use a layout Page)



Add MVC Controller with views, using Entity Framework

Model class: Employee (EmployeesCRUD.Models)

Data context class: EmployeesCRUD.Data.ApplicationDbContext

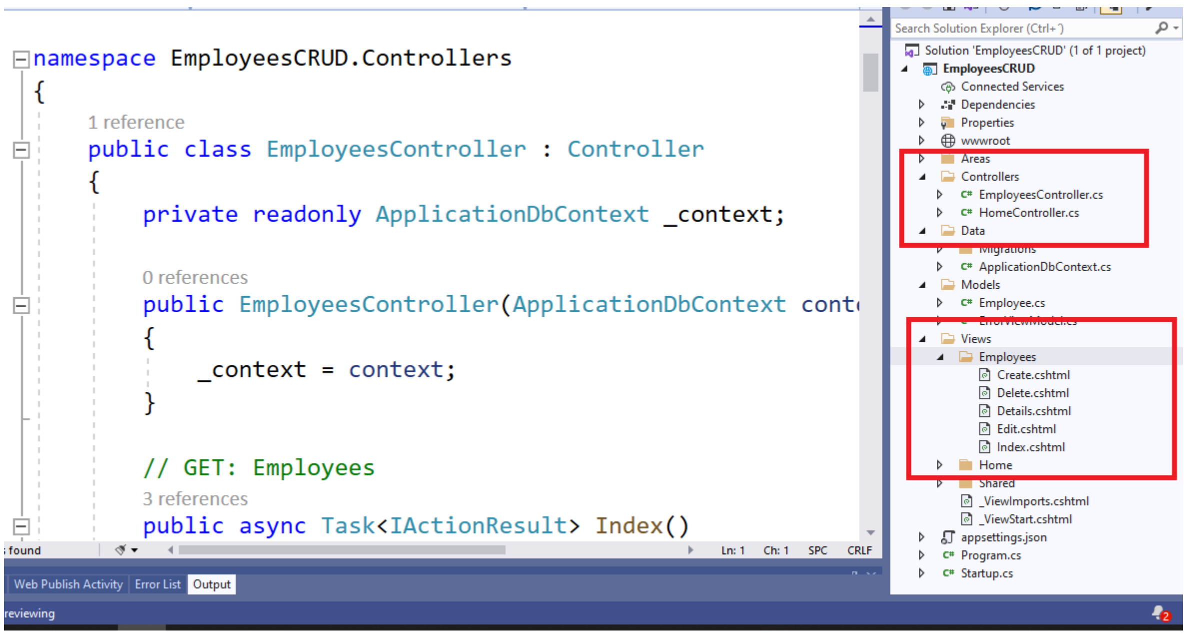
Views

- [✓] Generate views
- [✓] Reference script libraries
- [✓] Use a layout page

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

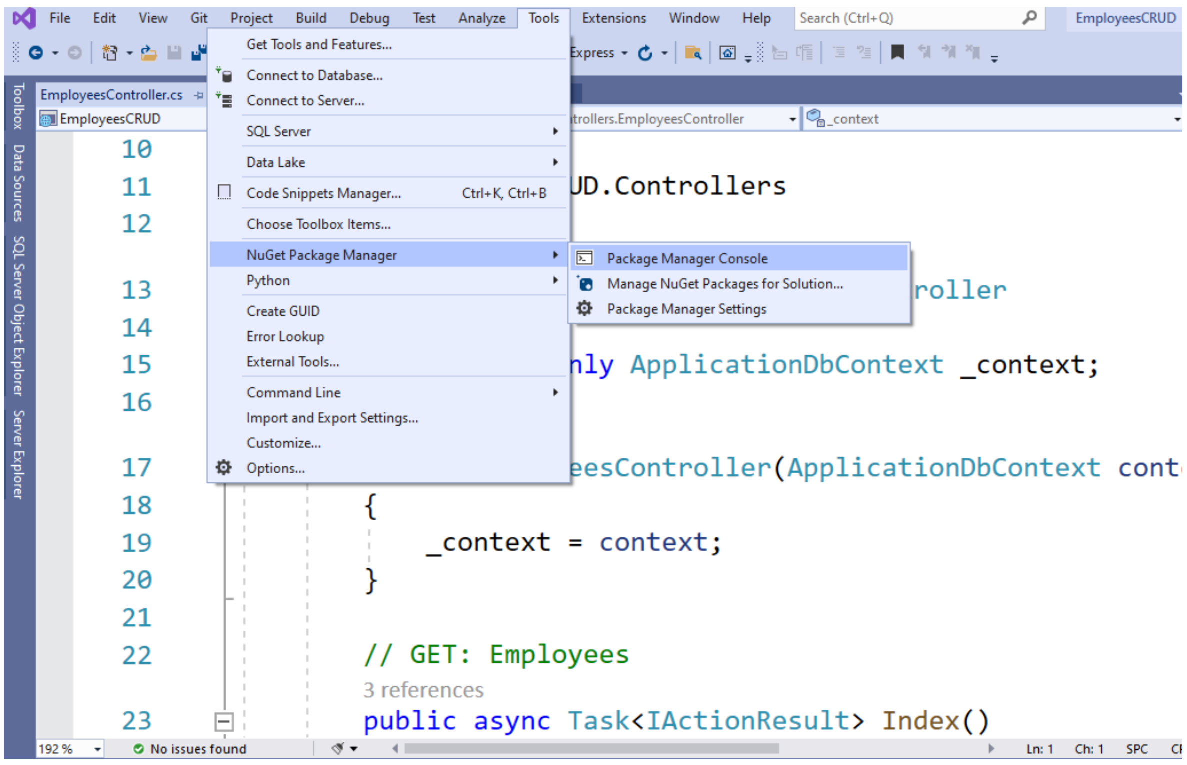
Controller name: EmployeesController

15. Click on “Add” and wait for a couple of minutes.
16. Once the process has completed. Notice the Employees Controller has been created and in the Views folder some views for the Employees has been created too. Each view for each operation: Create, Delete, Details and Edit.



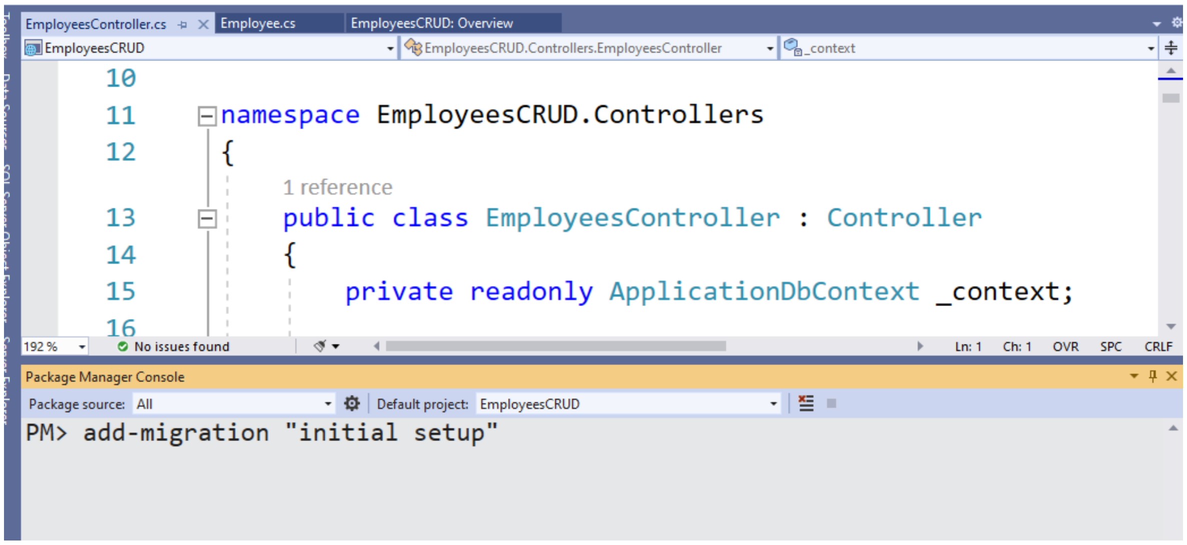
```
namespace EmployeesCRUD.Controllers
{
    1 reference
    public class EmployeesController : Controller
    {
        private readonly ApplicationDbContext _context;

        0 references
        public EmployeesController(ApplicationDbContext cont
        {
            _context = context;
        }

        // GET: Employees
        3 references
        public async Task<IActionResult> Index()
```

17. This means that if we append /employees at the end of the URL of our website, the EmployeesController will act on this. However, we haven’t created any database yet. For this let’s open the Package Manager Console in Tools, NuGet Package Manager and “Package Manager Console”



```
UD.Controllers

roller

nly ApplicationDbContext _context;

eesController(ApplicationDbContext cont
        {
            _context = context;
        }

        // GET: Employees
        3 references
        public async Task<IActionResult> Index()
```

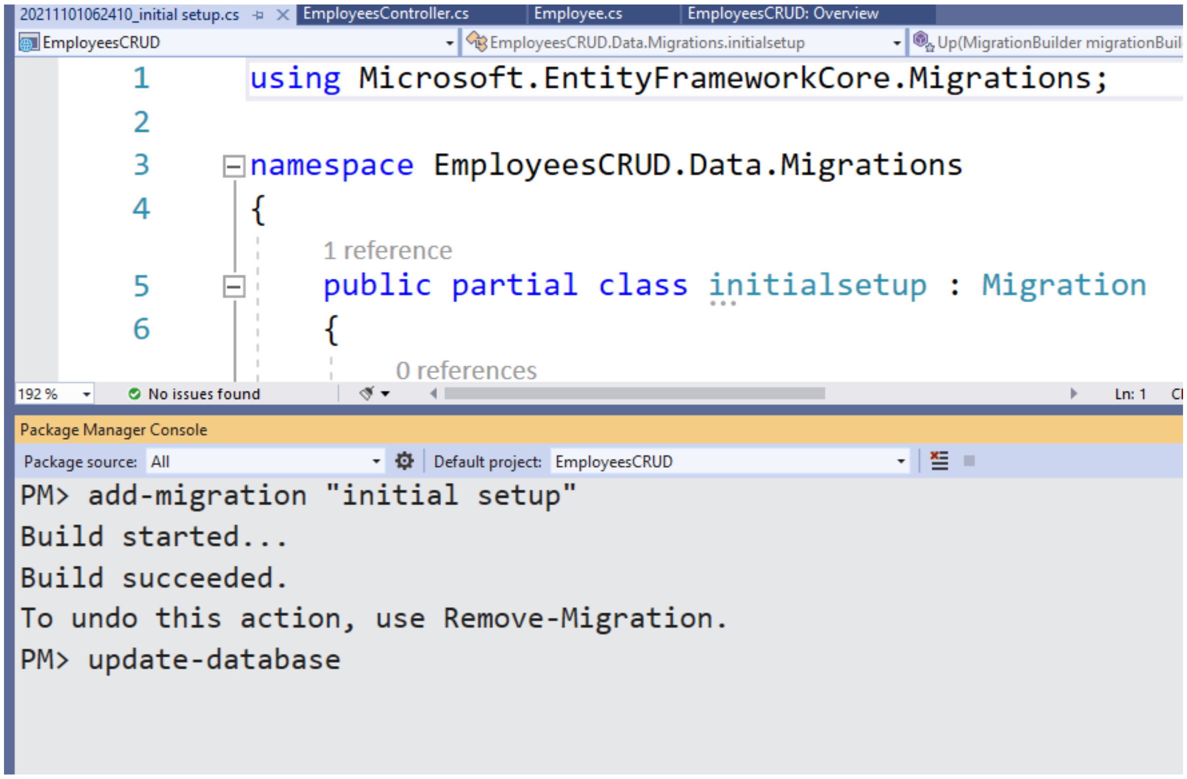
18. In the Project Manager console type:
    `add-migration "initial setup"`



```
namespace EmployeesCRUD.Controllers
{
    1 reference
    public class EmployeesController : Controller
    {
        private readonly ApplicationDbContext _context;
```

```
PM> add-migration "initial setup"
```

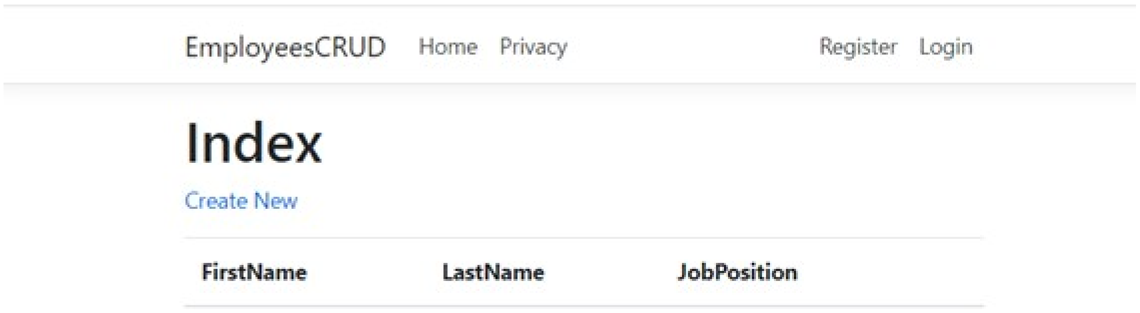
19. Press enter and then type:
    `update-database`


20211101062410_initial setup.cs
EmployeesController.cs
Employee.cs
EmployeesCRUD: Overview
EmployeesCRUD
EmployeesCRUD.Data.Migrations.initialsetup
Up(MigrationBuilder migrationBuil
192 %
No issues found
Ln: 1
Package Manager Console
Package source: All
Default project: EmployeesCRUD

```
using Microsoft.EntityFrameworkCore.Migrations;

namespace EmployeesCRUD.Data.Migrations
{
    1 reference
    public partial class initialsetup : Migration
    {
        0 references
```

```
PM> add-migration "initial setup"
Build started...
Build succeeded.
To undo this action, use Remove-Migration.
PM> update-database
```

20. The previous two steps creates the files with the instructions to create the table inside our Database. You can take a look to this files inside Data Folder, Migrations.
21. Now run your application and append “employees” at the end of the URL.

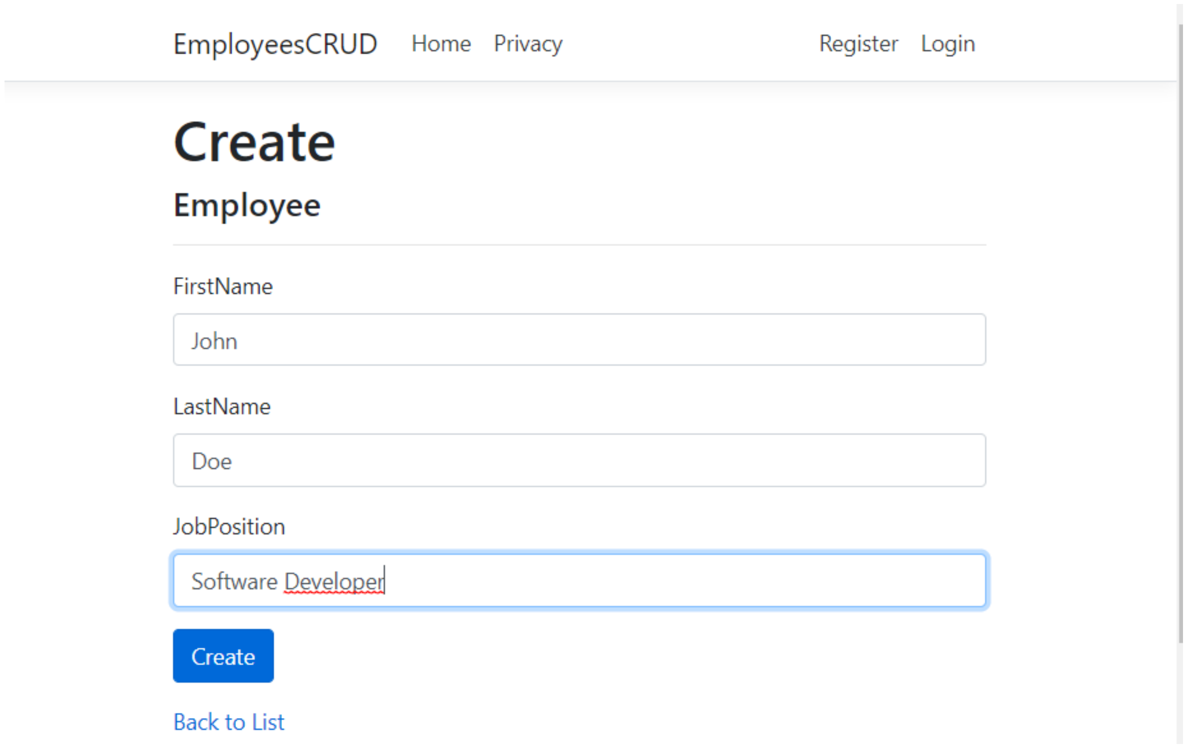

EmployeesCRUD Home Privacy Register Login

Index

Create New

FirstName	LastName	JobPosition

22. With this interface you can add a new employee by clicking on “Create New”.


EmployeesCRUD Home Privacy Register Login

Create

Employee

FirstName

John

LastName

Doe

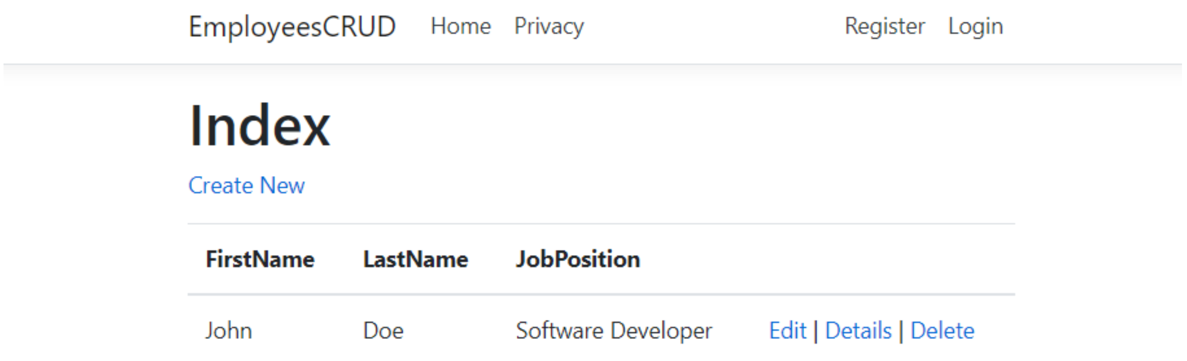
JobPosition

Software Developer

Create

Back to List

23. Once you add the first Employee, you will have the options to Edit, See Details and Delete it. All of this operations were created automatically by using the Entity Framework.



Index

Create New

FirstName	LastName	JobPosition	
John	Doe	Software Developer	Edit | Details | Delete

24. Congratulations. You have created a CRUD application by using ASP Net and Entity Framework 😊