

| Developing ASP.NET core Web Apps.

Overview

If you're currently taking 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/>
- You'll need to download the following repository (Place link for GitHub repository)

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.

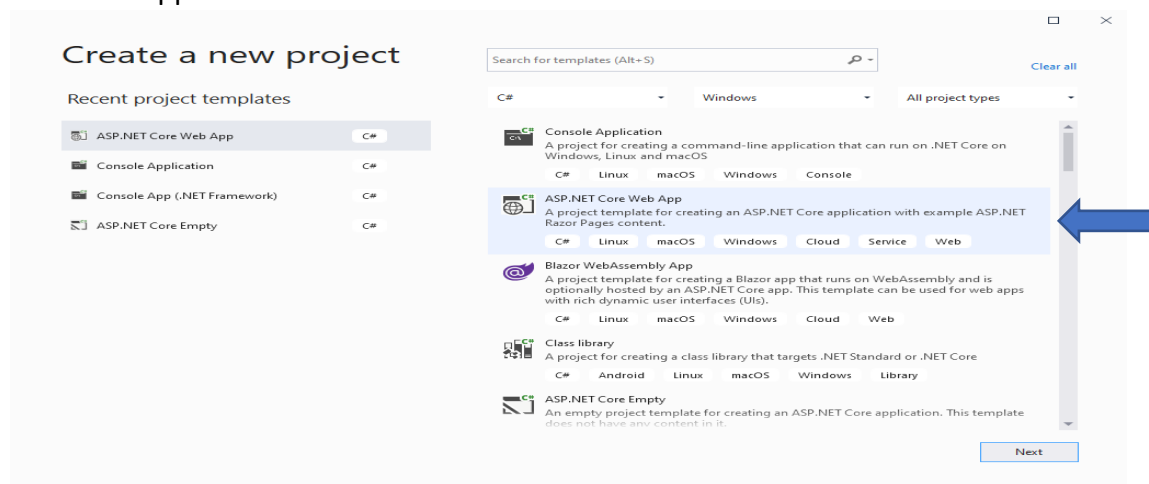
Objectives

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

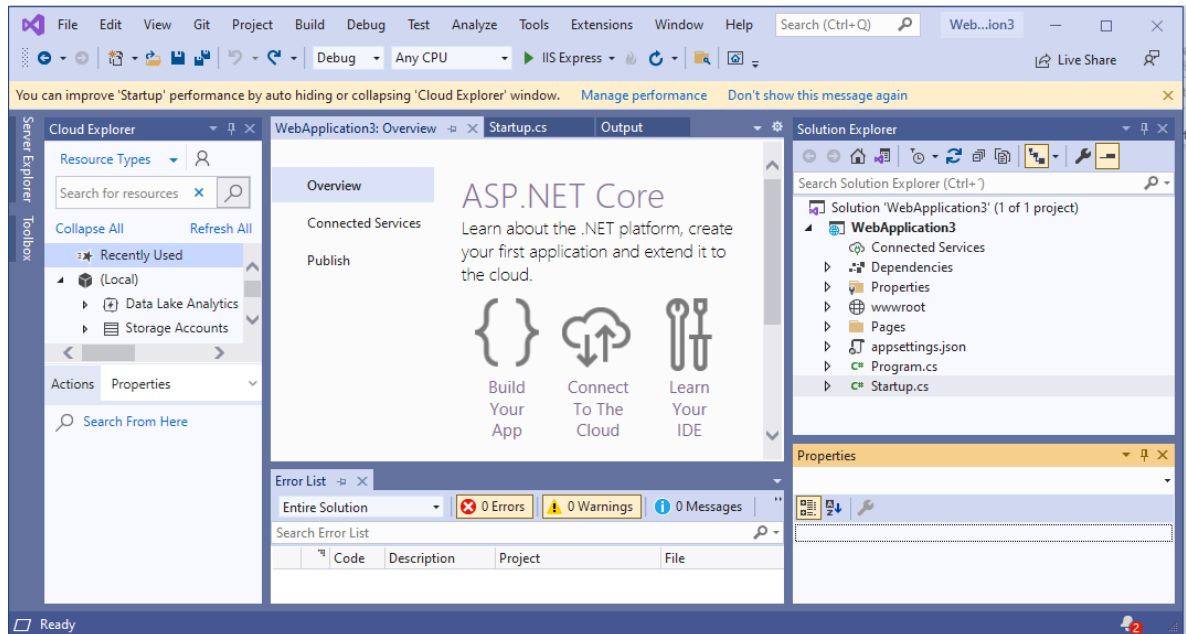
- a. Call an API.
- b. How routing works.
- c. Endpoints and endpoint routing.
- d. Become familiar with the ASP.NET core framework.
- e. Route Requests of an API.

Exercise 01: Endpoints

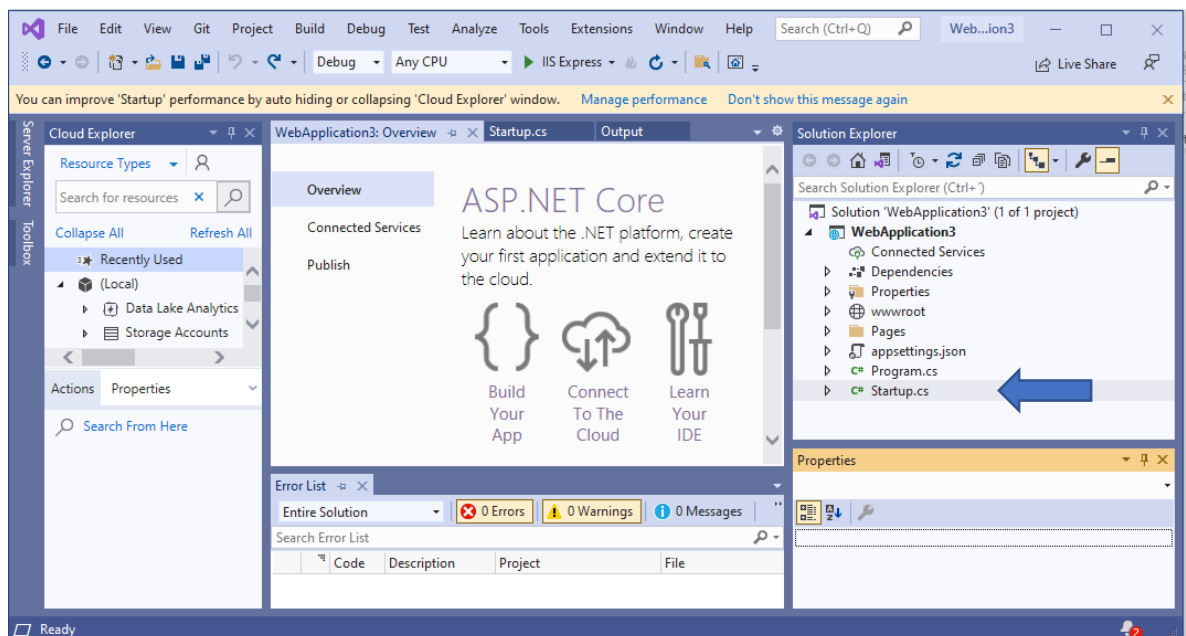
1. Open Visual Studio 2019.
2. Create a new Project, but this time choose the second option that says ASP.NET Core web app and click next.



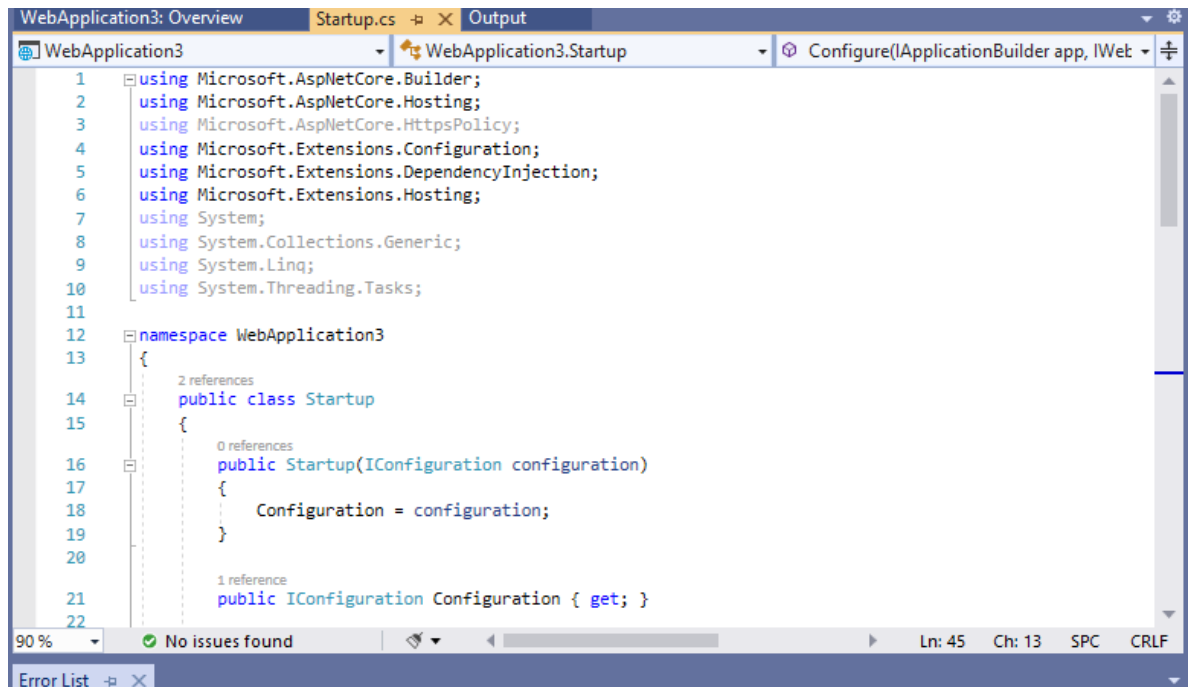
3. Choose a name for your project and click next.
4. Leave the default options and click next.
5. When the new project opens it, your Visual Studio window will look like this:



6. Please open your Startup.cs file.

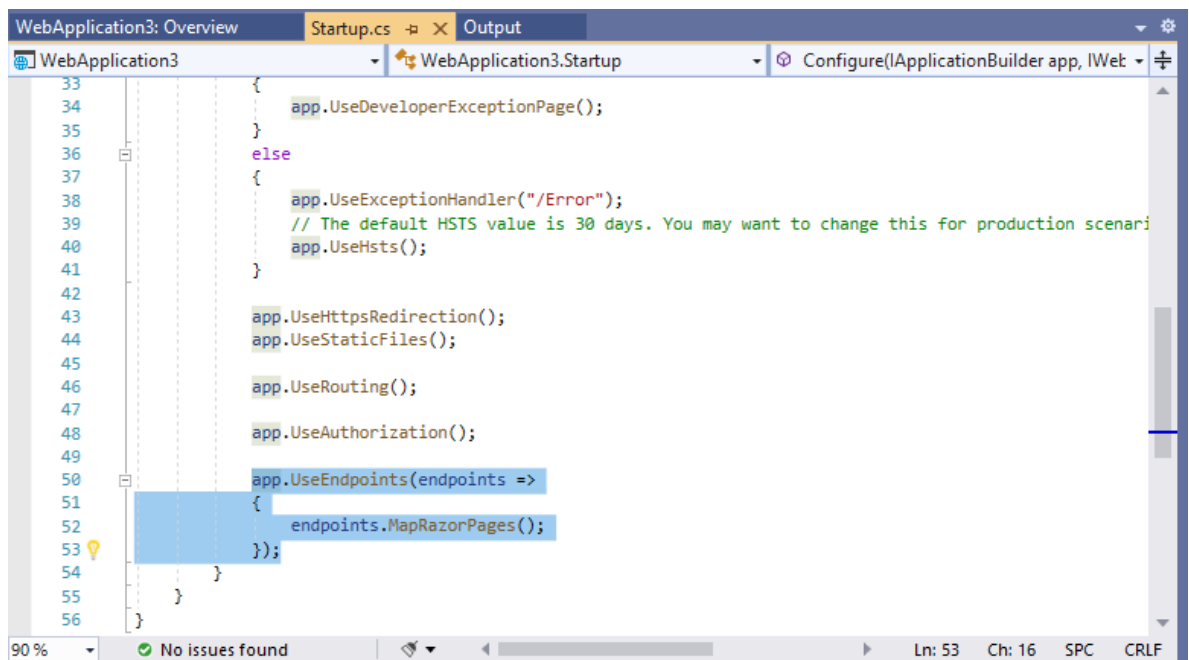


7. When you open the startup class, your window should be looking like this:



```
1 using Microsoft.AspNetCore.Builder;
2 using Microsoft.AspNetCore.Hosting;
3 using Microsoft.AspNetCore.HttpsPolicy;
4 using Microsoft.Extensions.Configuration;
5 using Microsoft.Extensions.DependencyInjection;
6 using Microsoft.Extensions.Hosting;
7 using System;
8 using System.Collections.Generic;
9 using System.Linq;
10 using System.Threading.Tasks;
11
12 namespace WebApplication3
13 {
14     2 references
15     public class Startup
16     {
17         0 references
18         public Startup(IConfiguration configuration)
19         {
20             Configuration = configuration;
21         }
22         1 reference
23         public IConfiguration Configuration { get; }
```

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



```
33
34     app.UseDeveloperExceptionPage();
35 }
36 else
37 {
38     app.UseExceptionHandler("/Error");
39     // The default HSTS value is 30 days. You may want to change this for production scenarios.
40     app.UseHsts();
41 }
42
43 app.UseHttpsRedirection();
44 app.UseStaticFiles();
45
46 app.UseRouting();
47
48 app.UseAuthorization();
49
50 app.UseEndpoints(endpoints =>
51 {
52     endpoints.MapRazorPages();
53 });
54 }
55 }
56 }
```

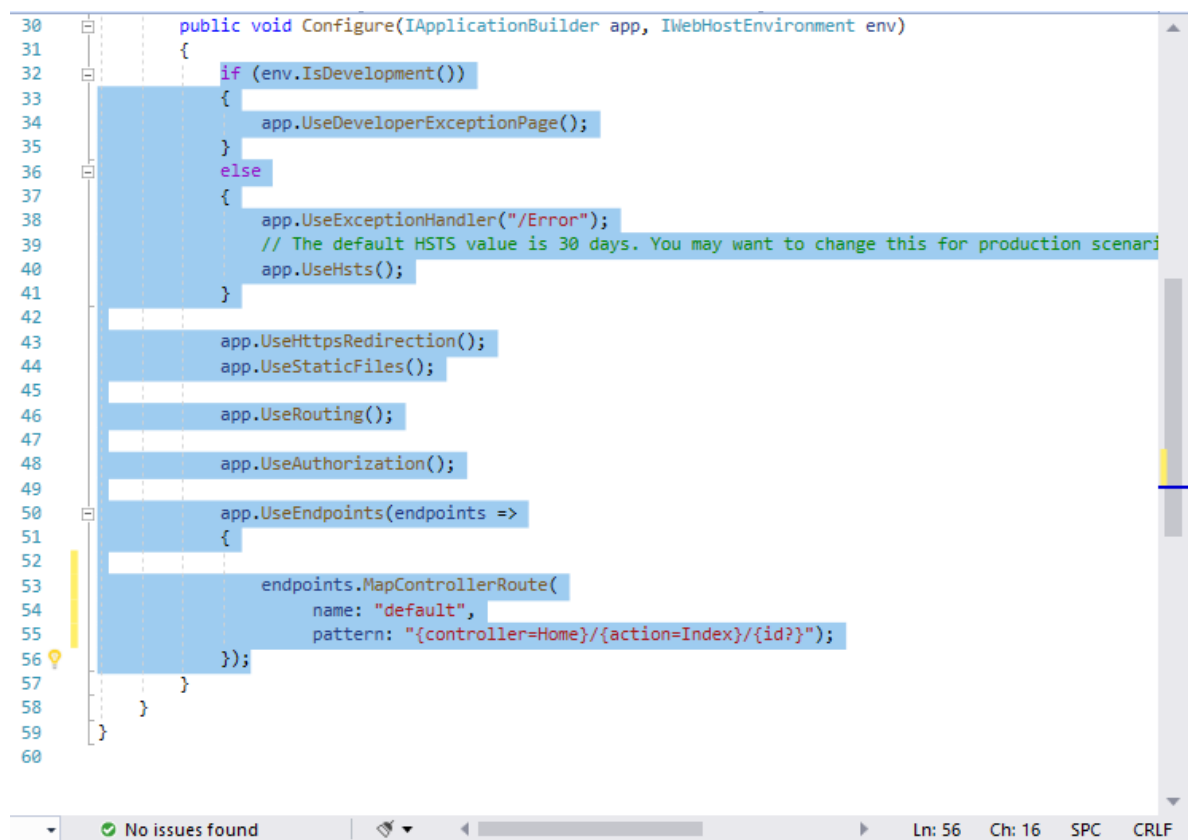
9. The highlighted method above, adds the default MVC Conventional route using the MapControllerRoute method. It also adds the default conventional route to

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:



```
30 public void Configure(IApplicationBuilder app, IWebHostEnvironment env)  
31 {  
32     if (env.IsDevelopment())  
33     {  
34         app.UseDeveloperExceptionPage();  
35     }  
36     else  
37     {  
38         app.UseExceptionHandler("/Error");  
39         // The default HSTS value is 30 days. You may want to change this for production scenarios  
40         app.UseHsts();  
41     }  
42  
43     app.UseHttpsRedirection();  
44     app.UseStaticFiles();  
45  
46     app.UseRouting();  
47  
48     app.UseAuthorization();  
49  
50     app.UseEndpoints(endpoints =>  
51     {  
52  
53         endpoints.MapControllerRoute(  
54             name: "default",  
55             pattern: "{controller=Home}/{action=Index}/{id?}");  
56     });  
57 }  
58  
59  
60
```

The screenshot shows a code editor with line numbers 30 to 60. The code is for a `Configure` method. Lines 32-41, 43-45, 47-49, and 53-55 are highlighted in blue. The status bar at the bottom indicates 'No issues found' and shows the cursor at line 56, column 16.

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.

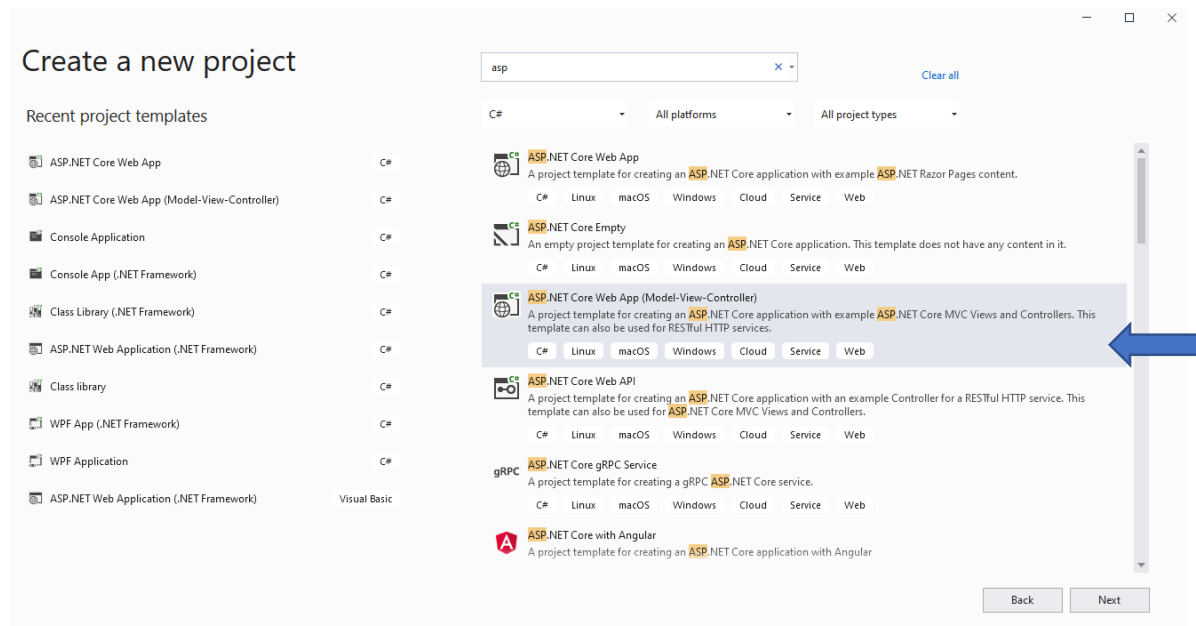
Objectives

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

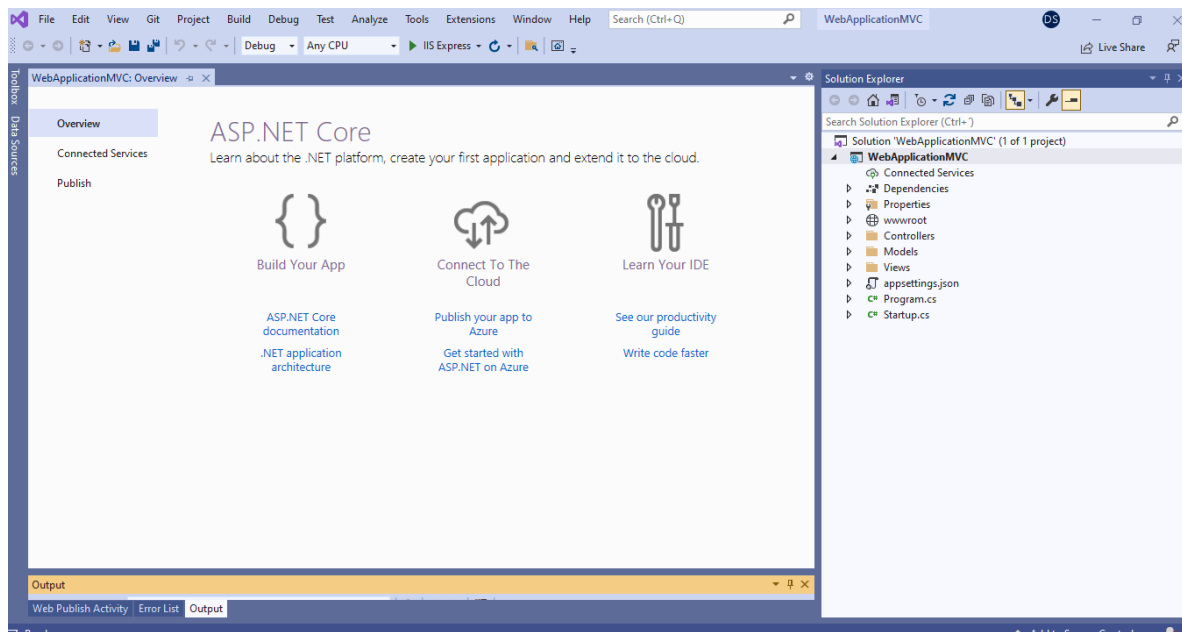
- a. Use template inheritance
- b. Modify the style of the base template

Exercise 01: Create the project

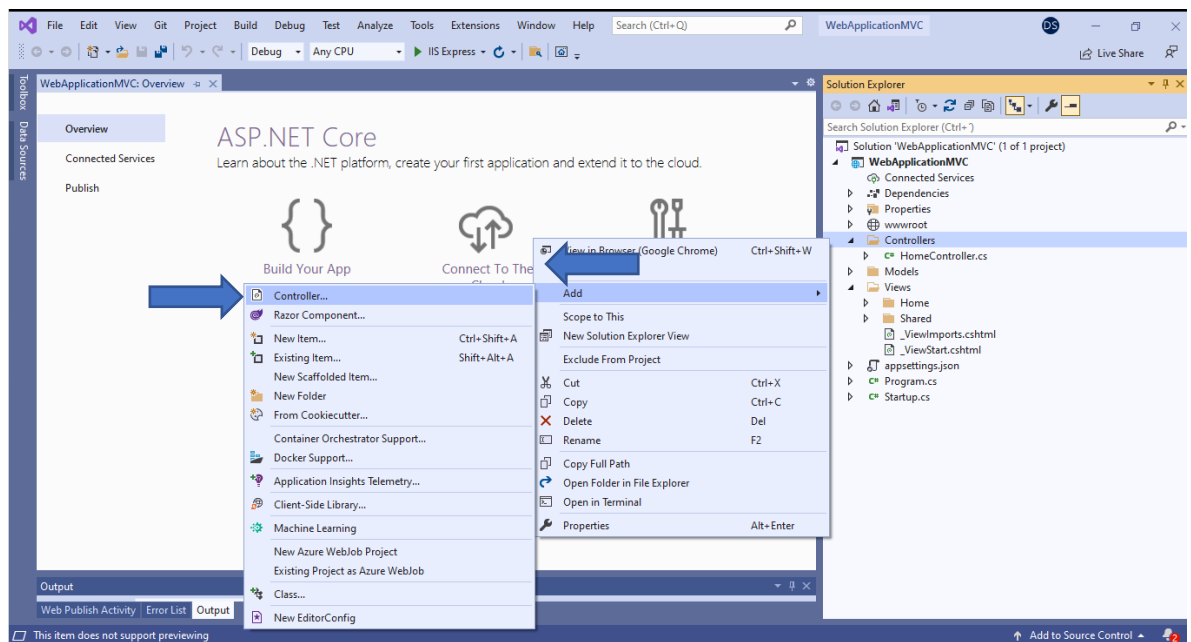
1. Open Visual Studio 2019.
Create a new Project. Select “ASP.NET Core Web App (Model-View-Controller)”



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:

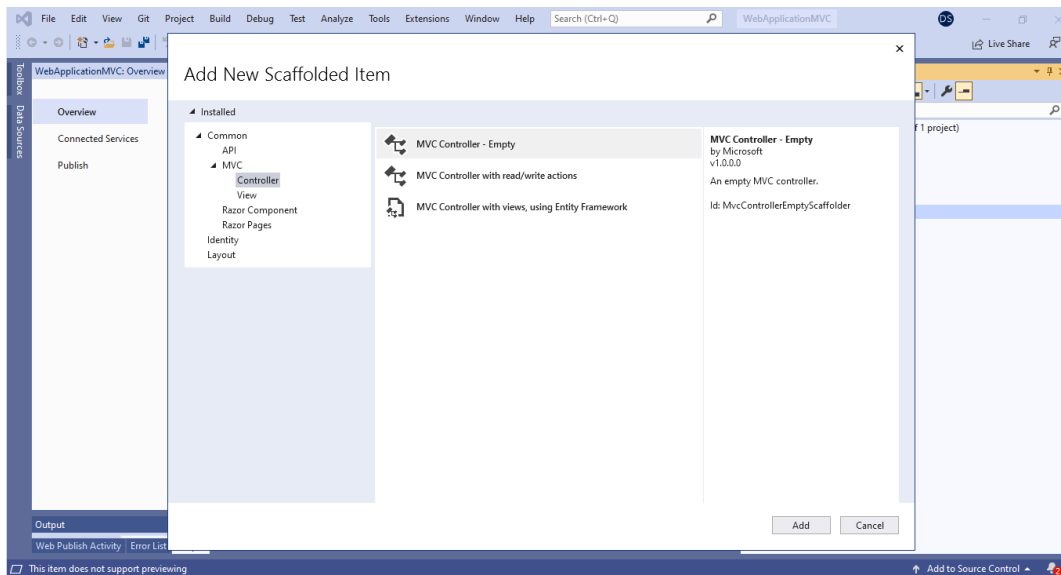


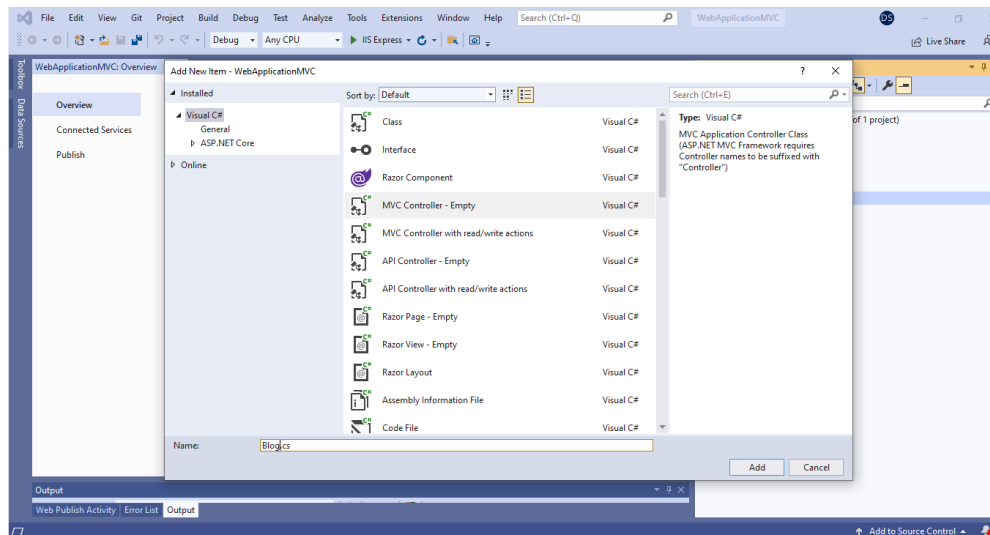
5. Note that different files and folders has been created.
6. On this lab we will focus on the following folders:
 - a. Controllers
 - b. Views
7. By default, the project contains just one controller and a Home folder.
8. Right click on Controllers Folder and then click “Add Controller”



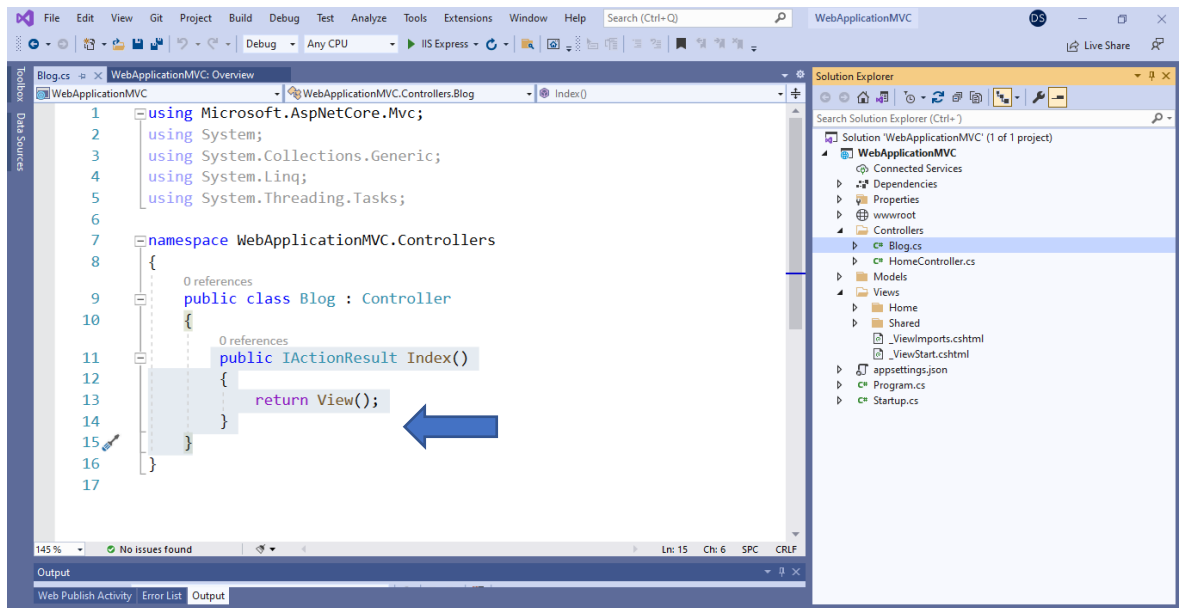
9. Select “MVC Controller – Empty”

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.

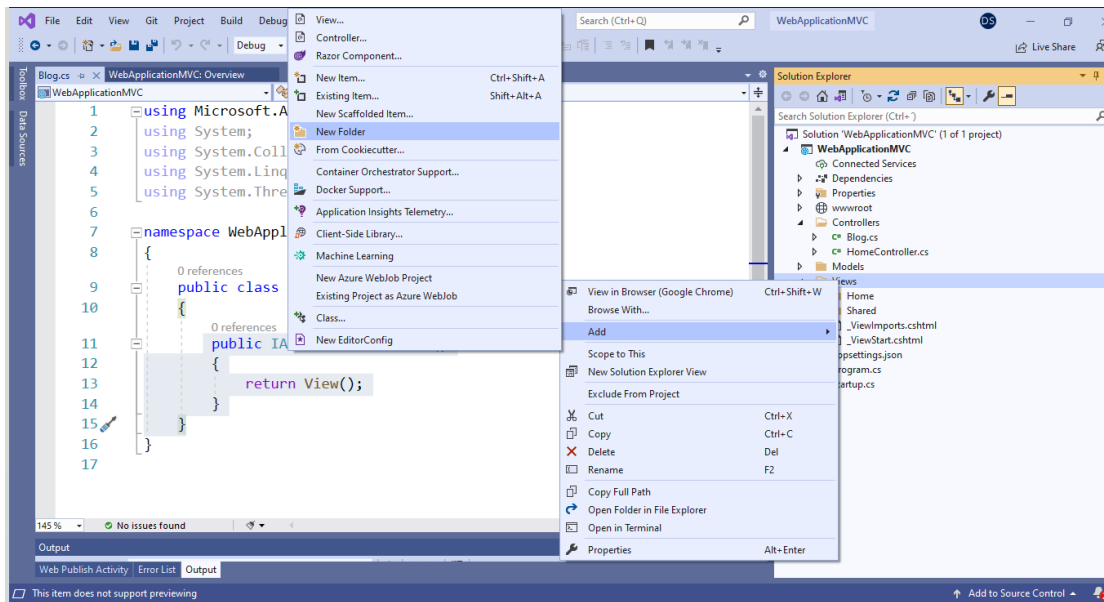




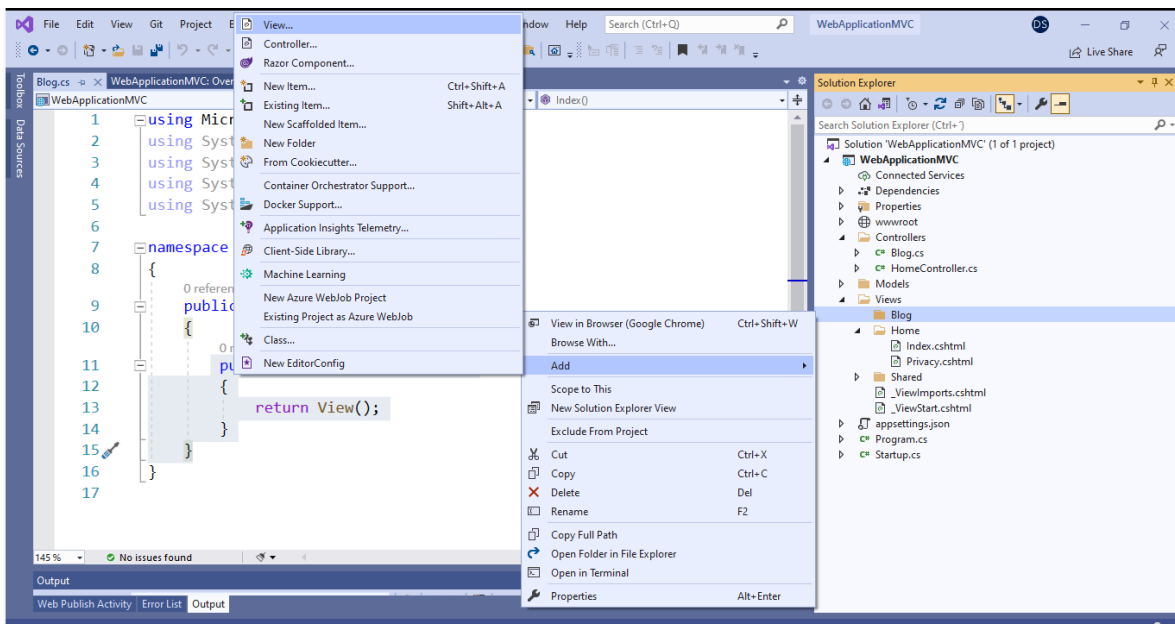
11. Once we click in “Add”, a new class (Blog.cs) will be created inside “Controllers” folder. Inside the file Blog.cs we will find a method named “Index”. This means that if we enter the url “mywebsit/blog” this will be the method that will be executed. However we haven’t provided a View (HTML Code)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

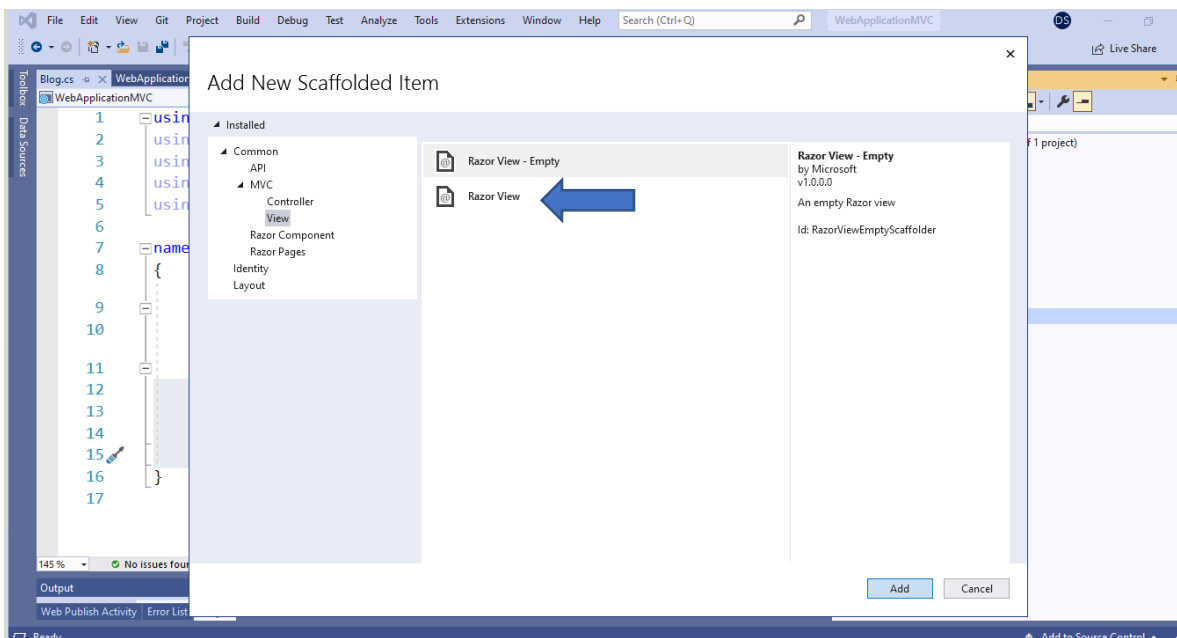


12. Now let’s add the HTML code that we want to return. Right click on “Views” folder and create a new folder with the name “Blog”



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





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:

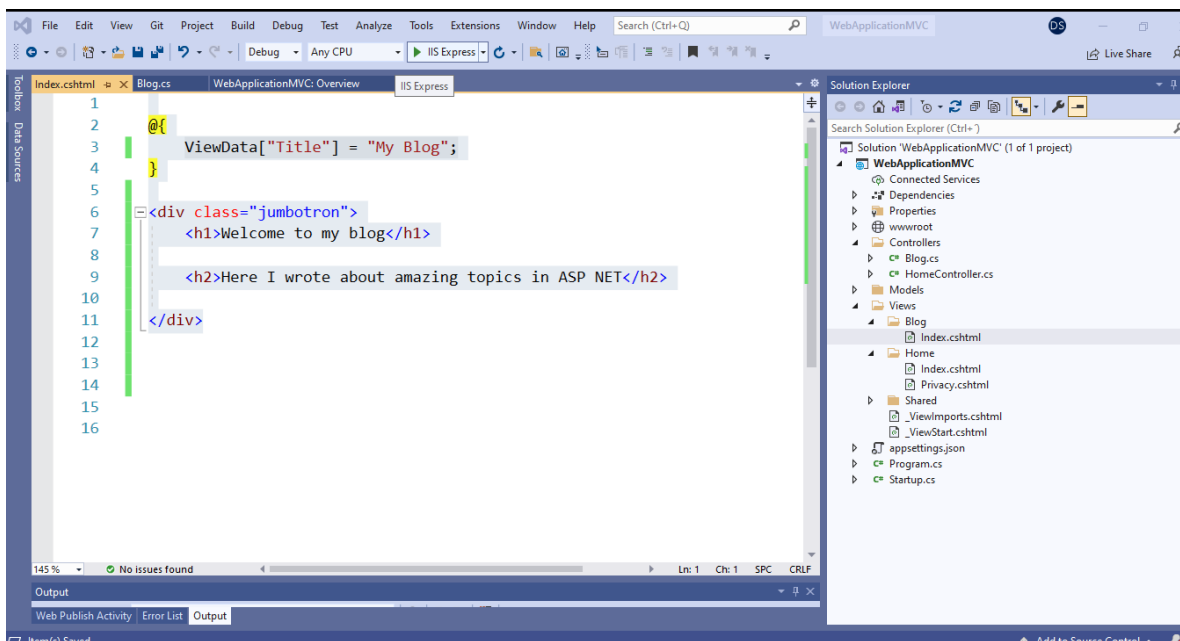
```
@{
    ViewData["Title"] = "My Blog";
}

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

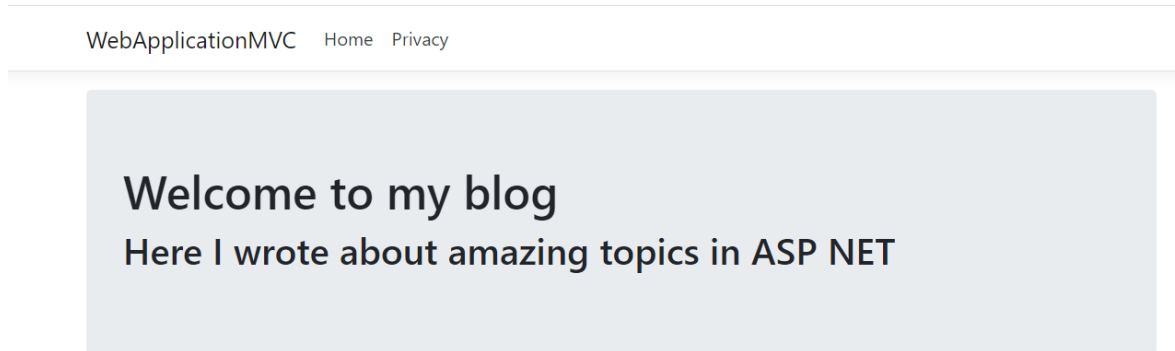
    <h2>Here I wrote about amazing topics in ASP NET</h2>

</div>
```

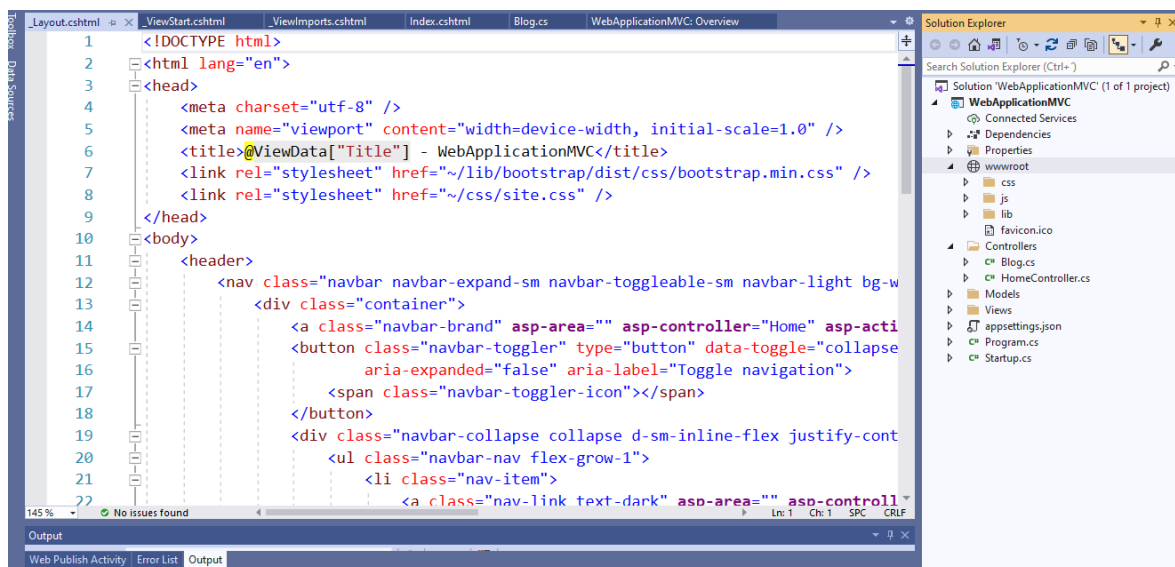
15. Once you update the content of this file. Click on “Run” button and wait until the web application runs on your default browser.



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



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.



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:

WebApplicationMVC Home Privacy



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

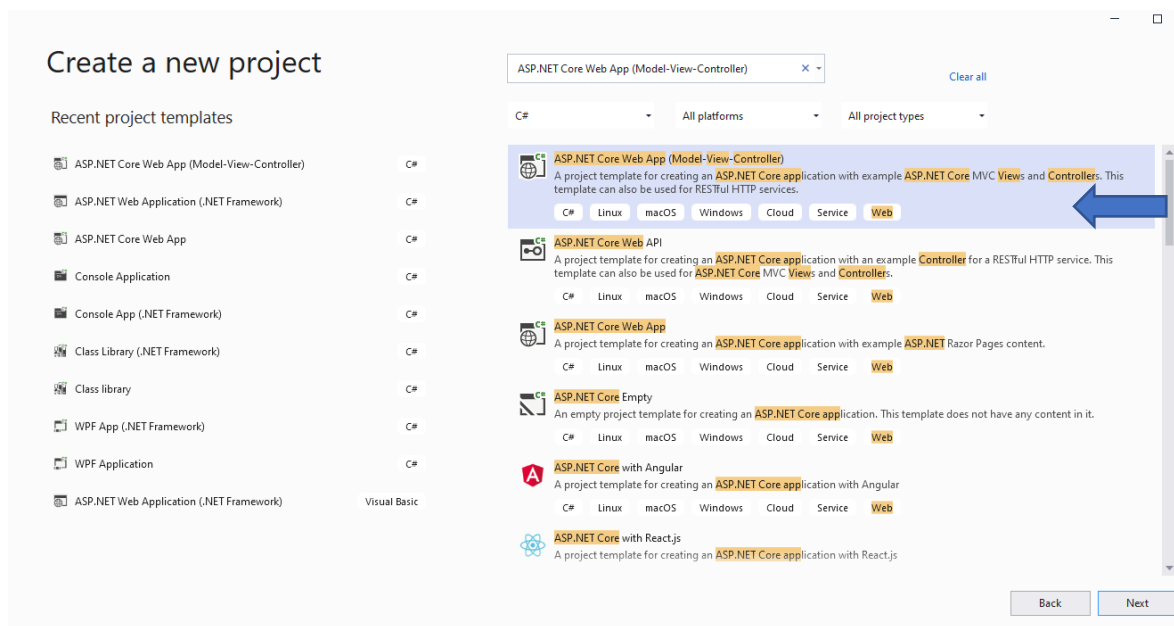
Objectives

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

- Create an ASP Net Application
- Create the Model
- Create the tables from the Database

Exercise 01: Create the project

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



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

Additional information

ASP.NET Core Web App (Model-View-Controller) C# Linux macOS Windows Cloud Service Web

Target Framework ⓘ

.NET 5.0 (Current)

Authentication Type ⓘ

Individual Accounts

None

Individual Accounts

Microsoft identity platform

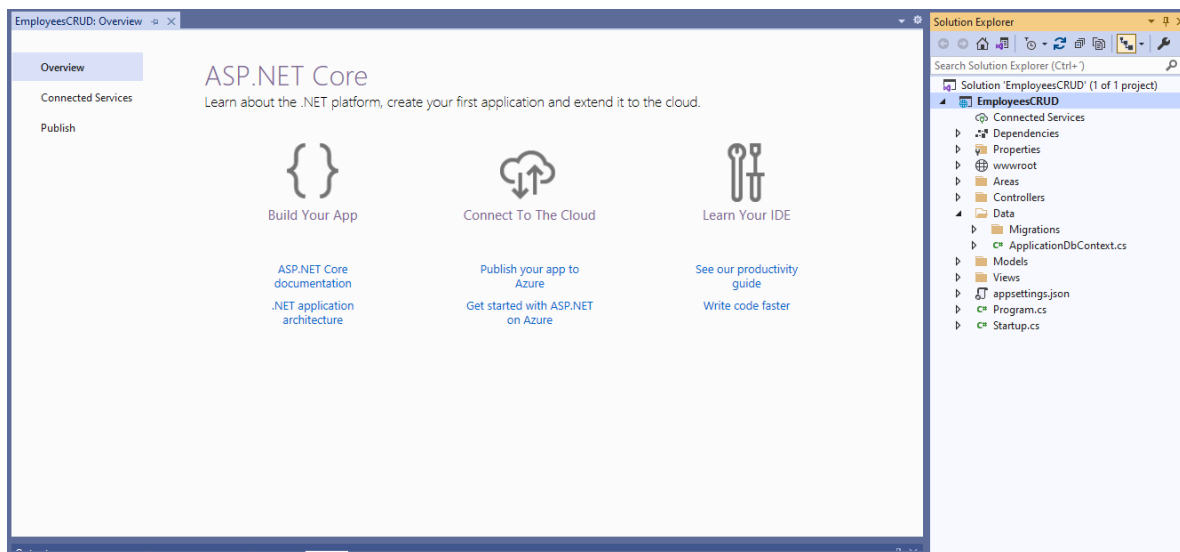
Windows ⓘ

Linux

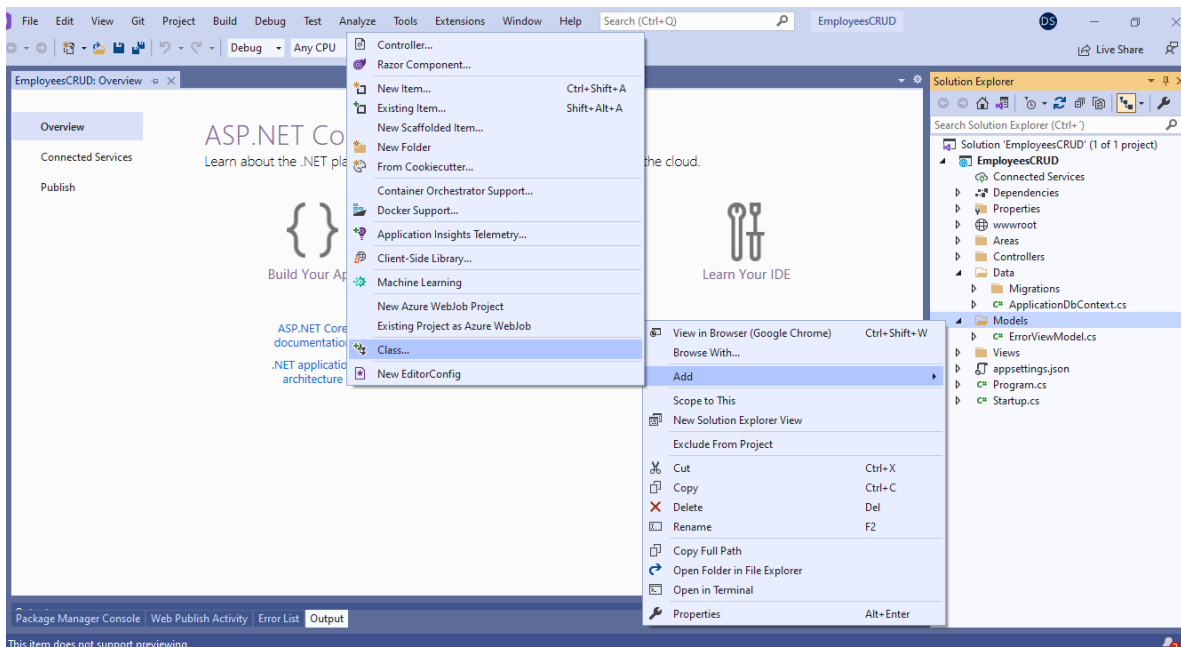
☐ Enable Razor runtime compilation ⓘ

Back Create

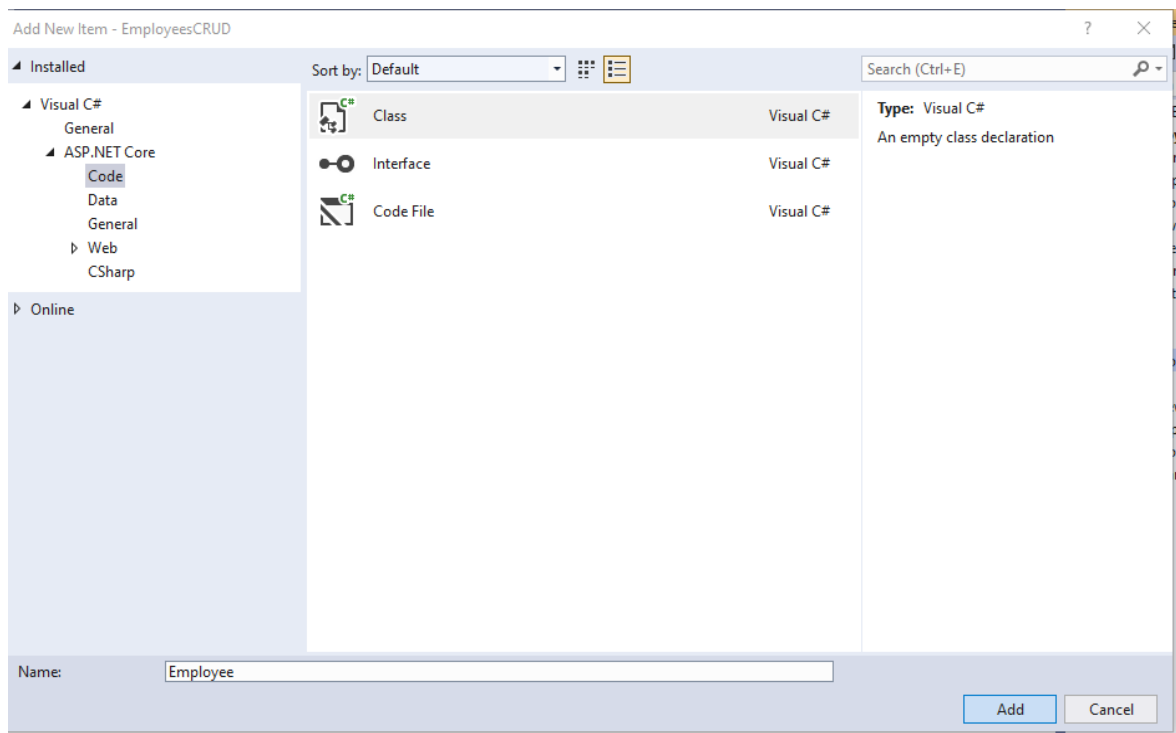
4. Select Next and then change the Authentication Type to “Individual Accounts”.
5. This will create the project with some files and folders. As MVC pattern we have the Model, Views and Controller folders. Additionally other folder named “Data” with Migrations and a class have been provided.



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; }

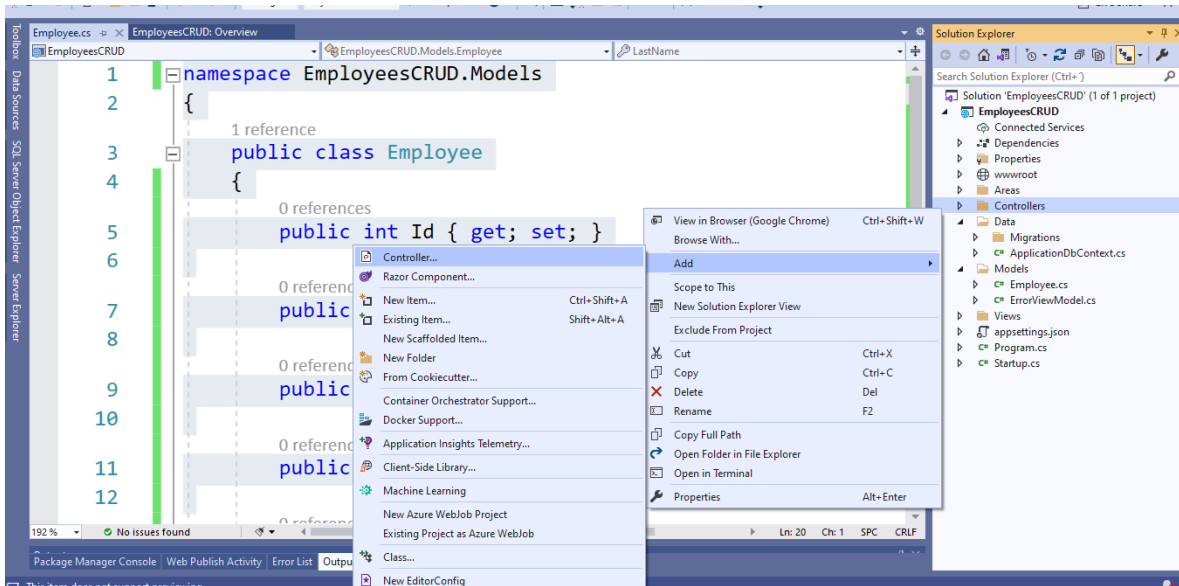
        public string LastName { get; set; }

        public string JobPosition { get; set; }

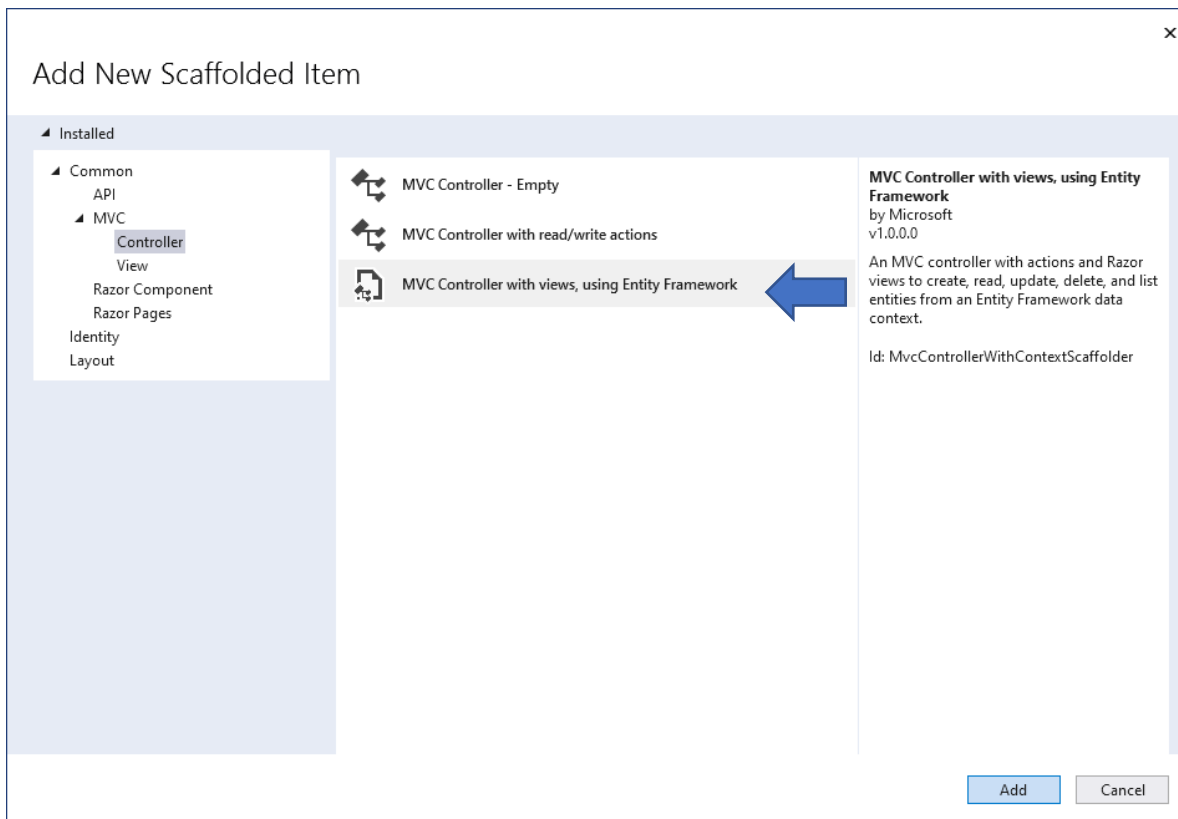
        public Employee()
        {
        }
    }
}

```

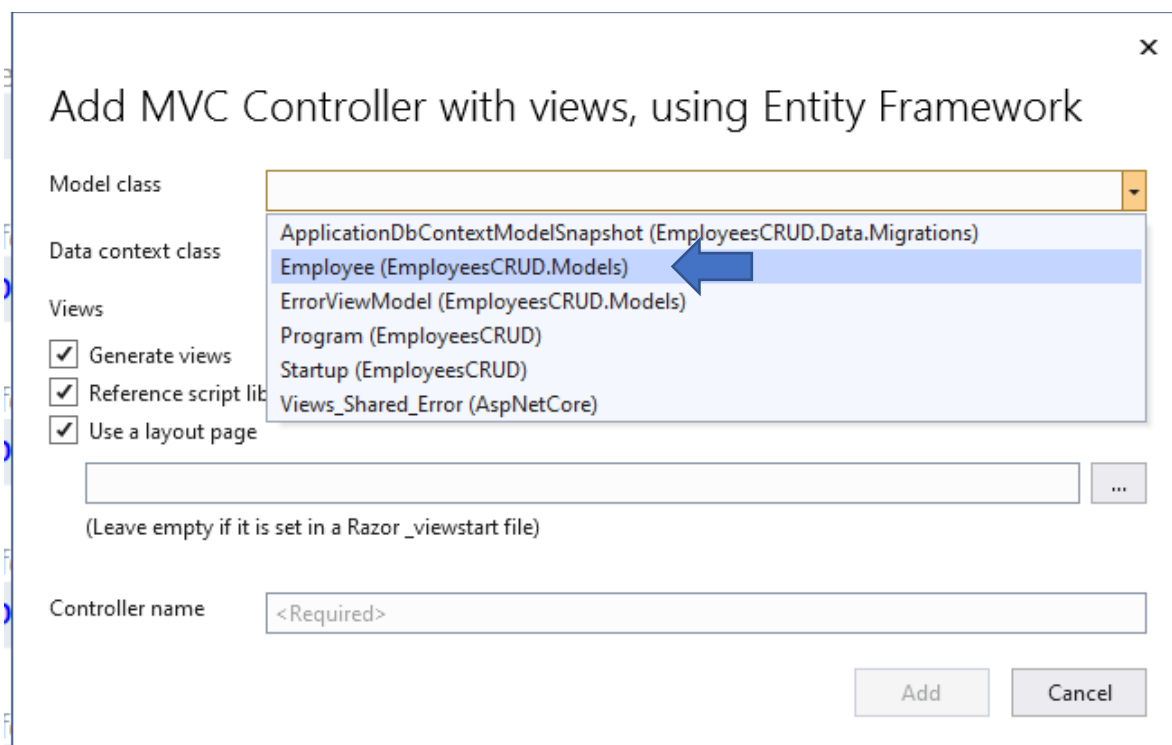
10. Once this model has been created, let's add the Controller. Right click on Controllers Folder and select "Add Controller"



11. Select “MVC Controller with views, using Entity Framework”



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



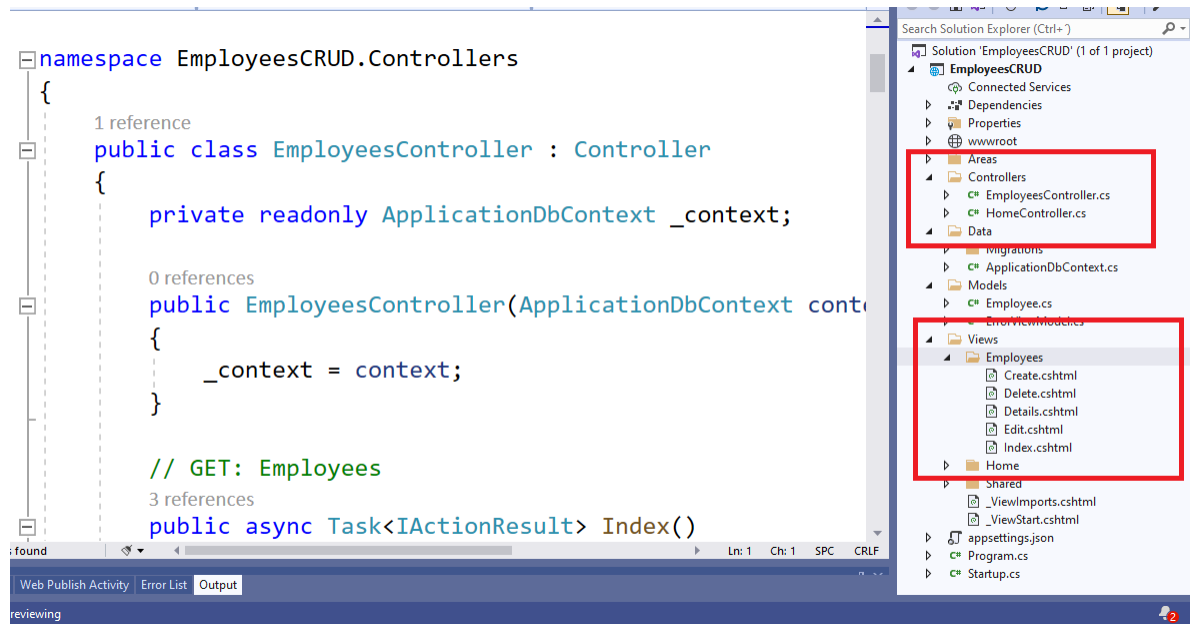
13. Add a new Data Context and enter the name
“EmployeesCRUD.Data.ApplicationDbContext”.

The screenshot shows the 'Add MVC Controller with views, using Entity Framework' dialog box. The 'Model class' is set to 'Employee (EmployeesCRUD.Models)'. The 'Data context class' is set to '<Required>'. A secondary dialog box titled 'Add Data Context' is open, showing 'New data context type' as 'EmployeesCRUD.Data.ApplicationDbContext' with 'Add' and 'Cancel' buttons.

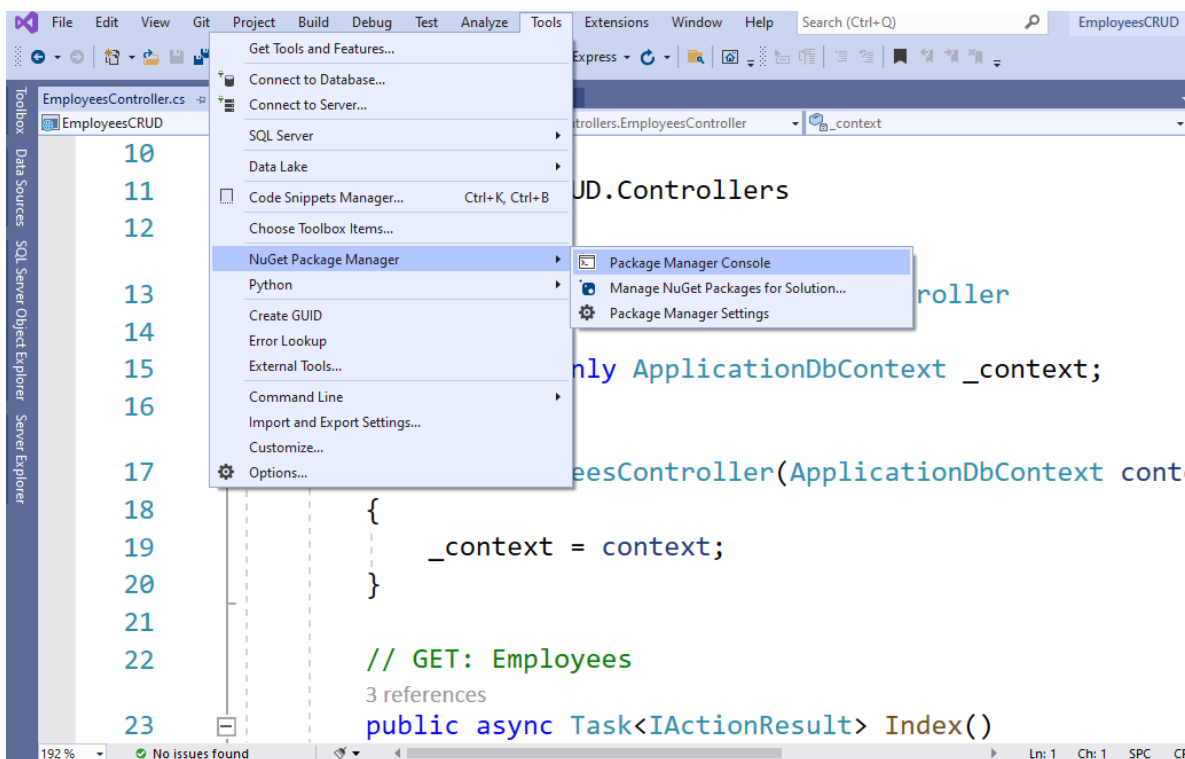
14. Make sure all the Views options are selected (Generate Views, Reference script libraries, Use a layout Page)

The screenshot shows the 'Add MVC Controller with views, using Entity Framework' dialog box. The 'Model class' is set to 'Employee (EmployeesCRUD.Models)'. The 'Data context class' is set to 'EmployeesCRUD.Data.ApplicationDbContext'. Under the 'Views' section, 'Generate views', 'Reference script libraries', and 'Use a layout page' are all checked. The 'Controller name' is set to 'EmployeesController'. 'Add' and 'Cancel' buttons are at the bottom.

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.



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”



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

The screenshot shows the Visual Studio IDE with the `EmployeesController.cs` file open. The code defines a namespace `EmployeesCRUD.Controllers` and a class `EmployeesController` that inherits from `Controller`. It has a private readonly property `_context` of type `ApplicationDbContext`. Below the code editor, the Package Manager Console is open, showing the command `PM> add-migration "initial setup"` entered.

```
10
11 namespace EmployeesCRUD.Controllers
12 {
13     1 reference
14     public class EmployeesController : Controller
15     {
16         private readonly ApplicationDbContext _context;
```

Package Manager Console
Package source: All Default project: EmployeesCRUD
PM> add-migration "initial setup"

19. Press enter and then type:
update-database

The screenshot shows the Visual Studio IDE with the `initialsetup.cs` file open. The code defines a namespace `EmployeesCRUD.Data.Migrations` and a partial class `initialsetup` that inherits from `Migration`. It includes a `using` statement for `Microsoft.EntityFrameworkCore.Migrations`. Below the code editor, the Package Manager Console is open, showing the command `PM> add-migration "initial setup"` followed by the output: `Build started...`, `Build succeeded.`, and `To undo this action, use Remove-Migration.` The command `PM> update-database` is also entered.

```
1 using Microsoft.EntityFrameworkCore.Migrations;
2
3 namespace EmployeesCRUD.Data.Migrations
4 {
5     1 reference
6     public partial class initialsetup : Migration
7     {
8         0 references
```

Package Manager Console
Package source: All Default project: EmployeesCRUD
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.

EmployeesCRUD

Home

Privacy

Register

Login

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 😊