

## ASP.NET Core

ASP.NET Core is the new version of the ASP.NET web framework mainly targeted to run on .NET Core platform.

ASP.NET Core is a cross-platform, high-performance, open-source framework for building modern, cloud-enabled, Internet-connected apps.

## Razor View

Razor View Engine is **a markup syntax which helps us to write HTML and server-side code in web pages using C# or VB.Net.**

It is server-side markup language however it is not at all a programming language

<div>ASP.NET vs ASP.NET Core</div>		
Criteria	ASP.NET	ASP.NET Core
Platform support	Windows only	Windows, Linux, Mac OS
Performance	High-performing framework	Demonstrated a better performance than ASP.NET
Architecture	Based on .NET Framework only	Based on .NET Framework and Core Framework
Components	Web Form, MVC, Web API	MVC, Web pages, Web API - no WebForms support
Dependencies	More dependencies, less monitoring options	A stricter control over the number of dependencies
Supported files	WCF, WF, WPF, VB, Web config and others	No support for Global.asax files and Web config, but appsettings.json is integrated
Isolation, modules, containers	Low level of isolation, not highly suitable for microservice and container development	Improved modular support, integration with docker, powerful isolation algorithms
Visual Studio Support	Supports all versions	Only the latest versions are supported starting from 2015

Jelvix

jelvix.com

ASP.NET	ASP.NET Core
ASP.NET Build For Windows	ASP.NET Core Build For Windows, Mac Linux
ASP.NET Has A Good Performance	ASP.NET Core Gives 4X Times The High Performance
Runs On .Net Framework Or Full .Net Framework	It Runs On .Net Core And Full .Net Framework.
Supports WebForm, Asp.Net MVC And Asp.Net WebAPI.	Supports MVC, Web API And Asp.Net Web Pages Originally Added In .Net Core 2.0.
It Used The Only IIS With Dependant On System.Web.Dll.	It Has No Dependent System.Web.Dll And So The IIS.
Supports C#, VB, WCF, WPF, And WF	Supports C#, VB, F# Language. No Support To WCF (Except Client Libraries), WPF, And WF.
ASP.NET MVC Application Added Web.Config, Global.Asax, Application Start.	Core Did Not Support Web.Config And Global.Asax Files. It Supports Appsettings.Json.
Container Support Is Not More Than Better As The ASP.Net Core Application.	Container Support Best Suited For Deployments Like Docker.
We Need To Re-Compile After The Code Change.	Core Browser Refresh Will Compile And Execute The Code With No Need For Re-Compile.

## Understanding Startup.cs file in ASP.NET 5

The Startup.cs file **establishes the entry point and environment for your ASP.NET Core application**; it creates services and injects dependencies so that the rest of the app can use them.

*Startup.cs* file is a replacement of *Global.asax* file in ASP.NET Web Form application. We know that *Global.asax* is an optional file to handle Application level event in ASP.NET application. In ASP.NET 5, *Startup.cs* file has introduced to server same.

Though the purpose is the same, the implementation of *Startup.cs* file is entirely different. *Global.asax* file contains mostly application level pre-defined events where *Startup.cs* file is more about registering services and injection of modules in HTTP pipeline. *Startup.cs* file contains **Startup** class which triggers at first when application launches and even in each HTTP request/response.

Actually, the inception of **Startup** class is in OWIN application which is a specification to reduce dependency of application on server.

**Startup** class is allowed to put in any namespace, but the class name must not change. In runtime, it looks for **Startup** keyword in metadata.

Now the question may come, is **startup** class mandatory in application? Yes, **startup** class is mandatory in each ASP.NET 5 application. It can be decorated with any access specifier. Multiple **Startup** classes are allowed in a single application. ASP.NET will select the appropriate class based on its namespace. Here is rule to select.

At first, it matches with project's root namespace first, otherwise using the class in the alphabetically first namespace.

Here is an example of a typical **startup** class which will scaffold once you choose empty ASP.NET5 template.

C#

Copy Code

```
public class Startup
{
    //For more information on how to configure your application,
    //visit http://go.microsoft.com/fwlink/?LinkID=398940
    public void ConfigureServices(IServiceCollection services)
    {
    }

    public void Configure(IApplicationBuilder app)
    {
    }
}
```

## Startup Constructor

We can specify constructor of a **Startup** class and by default, it takes three parameters:

C#

Copy Code

```
public class Startup
{
    public Startup(IApplicationEnvironment env,
        IHostingEnvironment hostenv, ILoggerFactory logger)
    {
    }
}
```

**IApplicationEnvironment** interface contains method and property related to current environment. Mostly, it can deal with environment variables in application.

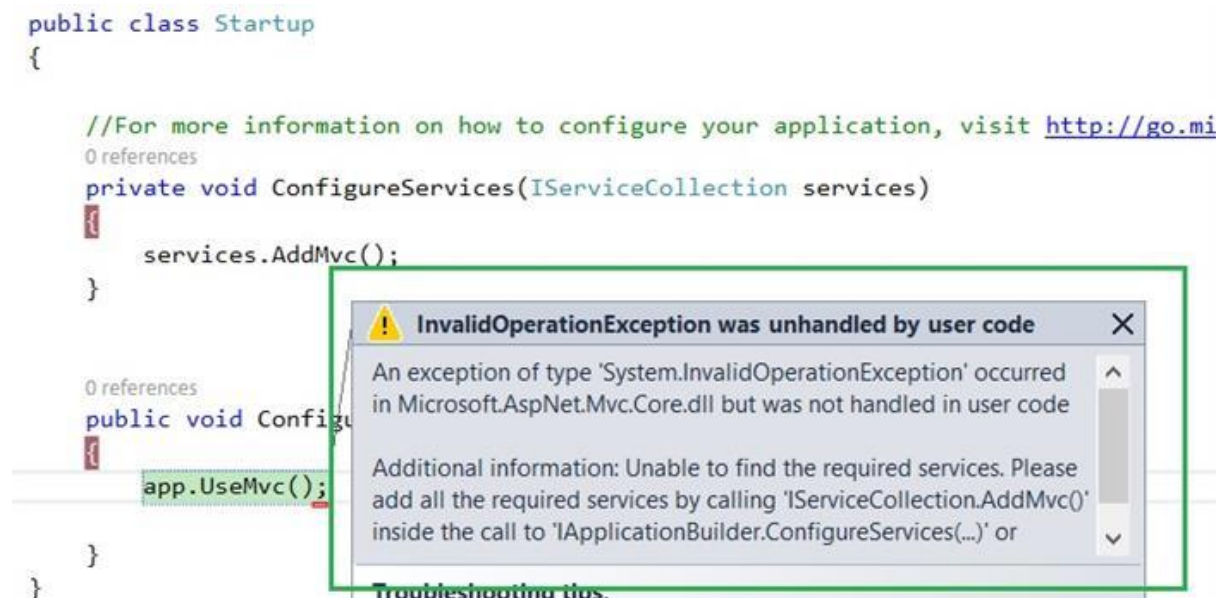
**IHostingEnvironment** interface deals with current hosting environment of application. It could be development, UAT or production. Based on hosting environment, we can change behavior of application.

**IloggerFactory** interface is to provide default logging service in ASP.NET 5 application. It supports various options to implement logging.

Now, let's understand **ConfigureService** and **Configure** methods now.

## ConfigureService Method

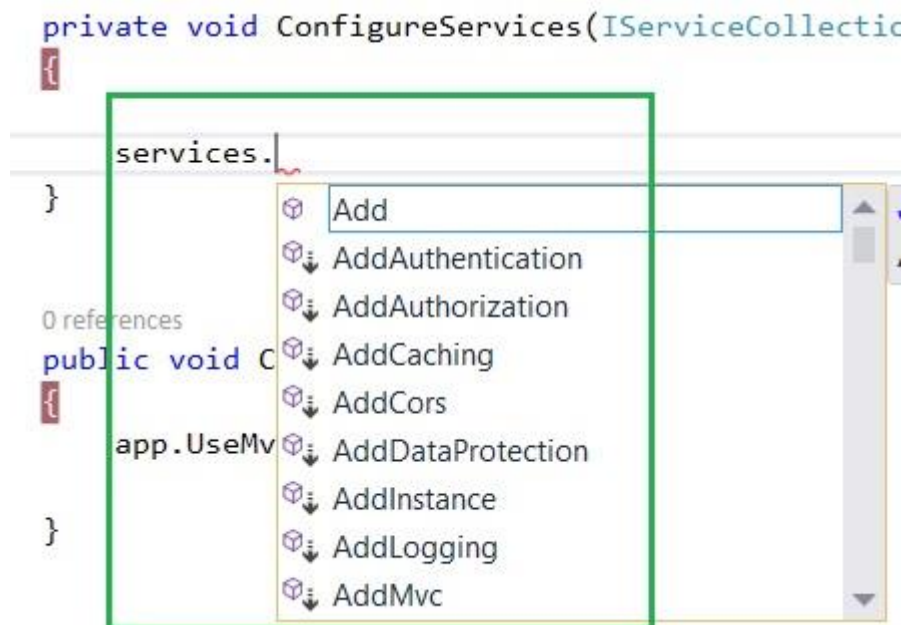
This is an optional method in **Startup** class which is used to configure services which will be used by application. Once the application launches and when first request comes, it hits the **ConfigureService** method. The method must be declared as **public** visibility scope otherwise environment will not be able to read the content from metadata. Here, we have defined **ConfigureService** in **private** mode and seeing that:



It was not able to attach MVC service.

**ConfigureService** method expects implementation of **IServiceCollection** type as argument which is needed to register the services. **ConfigureService** method executes

before **Configuration** method because sometimes, it needs to register the service before injecting to HTTP pipeline. **IServiceCollection** interface has implemented in many classes so that it offers many Add[something] extension methods to register services. Here, we are seeing a couple of extension methods like **AddAuthentication()**, **AddCors()** etc. to register respective services.



Not only the default services, we can register the dependencies which will be used in application in **ConfigurationService** method. In this example, we will use default IoC container of ASP.NET5. Here, we have created **ILog** interface and implemented in **Log** class.

C#

Copy Code

```
public interface ILog { }  
public class Log : ILog { }
```

This is how we can register the dependency by calling **AddTransient<T,T>()** method:

C#

Copy Code

```
public void ConfigureServices(IServiceCollection services)  
{  
    services.AddMvc();  
    services.AddTransient<ILog, Log>();  
}
```

Now, we are allowed to use **ILog** in any controller and in runtime, the implementation of **Ilog** will server automatically.

# Configure Method

The **Configure** method is used to specify how the application will respond in each HTTP request. Which implies that **Configure** method is HTTP request specific. The most typical use of **configure** method is to inject middleware in HTTP pipeline. The **configure** method must accept **IApplicationBuilder** parameter with some optional in build services like **IHostingEnvironment** and **ILoggerFactory**. Once we add some service in **ConfigureService** method, it will be available to **Configure** method to be used. That's why **ConfigureService** executes before **Configure**. For example:

C#

Copy Code

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddMvc();
}

public void Configure(IApplicationBuilder app)
{
    app.UseMvc();
}
```

To use **UseMvc()** extension method in **Configure()**, we need to add MVC service in **ConfigureService()** method at first by calling **AddMvc()**, because MVC framework is not shipped by default in ASP.NET architecture.

Here is a very common operation which we can perform in **Configure** method.

C#

Copy Code

```
public void Configure(IApplicationBuilder app ,
IHostingEnvironment env, ILoggerFactory loggerFactory)
{
    //Add MVC module and specify Root
    app.UseMvc();
    app.UseMvc(routes =>
    {
        routes.MapRoute(
            name: "default",
            template: "{controller=Home}/{action=Index}/{id?}");
    });

    //Perform environment specific task
    if (env.IsDevelopment())
    {
        //When it is dev environment
    }
}
```

```
}else
{
    //Other then Development
}
//Add static file module
app.UseStaticFiles();
//Use Identity Module
app.UseIdentity();
}
```

Besides that, we can inject our own middleware in HTTP pipeline within **Configure()** method. Here is one small example of the same.

C#

Copy Code

```
public void Configure(IApplicationBuilder app)
{
    app.UseMvc();
    app.Run(async context =>
    {
        await context.Response.WriteAsync("Return From Run.");
    });
}
```

We have attached our own HTTP handler in **Run()** extension which will short circuit all HTTP requests and return HTTP response from there itself.

## Conclusion

**Startup** class is the entry point of ASP.NET 5 application like *Global.asax* in an earlier version of ASP.NET. Application may have more than one **startup** class but ASP.NET will handle such a situation gracefully. **Startup** class is mandatory in ASP.NET 5 application.



## What is MVC?

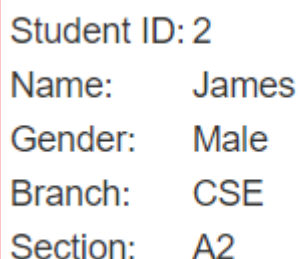
MVC stands for Model View and Controller. It is an architectural design pattern that means this design pattern is used at the architecture level of an application. So, the point that you need to remember is MVC is not a programming language, MVC is not a Framework, it is a design pattern. When we design an application, first we create the architecture of that application, and MVC plays an important role in the architecture of that particular application.

MVC Design Pattern is basically used to develop interactive applications. An interactive application is an application where there is user interaction involved and based on the user interaction some event handling occurred. The most important point that you need to remember is, it is not only used for developing web-based applications but also we can use this MVC design pattern to develop the Desktop or mobile-based application.

The MVC (Model-View-Controller) design pattern was introduced in the 1970s which divides an application into 3 major components. They are Model, View, and Controller. The main objective of the MVC design pattern is the separation of concerns. It means the domain model and business logic are separated from the user interface (i.e. view). As a result, maintaining and testing the application becomes simpler and easier.

## How does MVC Design Pattern work in ASP.NET Core?

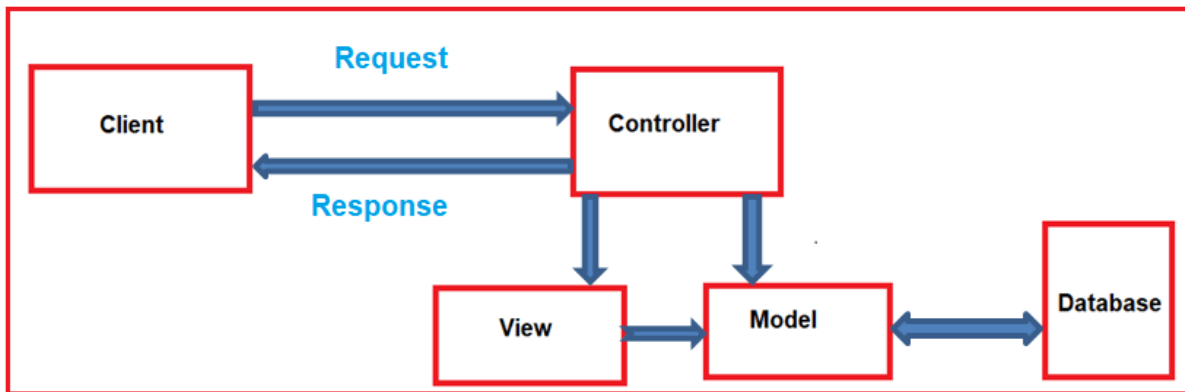
Let us see an example to understand how the MVC pattern works in the ASP.NET Core MVC application. For example, we want to design an application, where we need to display the student details on a web page as shown below.



Student ID: 2
Name: James
Gender: Male
Branch: CSE
Section: A2

So, when we issue a request something like "<http://dotnettutorials.net/student/details/2>" from a web browser then the following things are happening in order to handle the request.





The controller is the component in the MVC design pattern, who actually handles the incoming request. In order to handle the request, the controller components do several things as follows. The controller component creates the model that is required by a view. The model is the component in the MVC design pattern which basically contains classes that are used to store the domain data or you can say business data.

In the MVC design pattern, the Model component also contains the required logic in order to retrieve the data from a database. Once the model created by the controller, then the controller selects a view to render the domain data or model data. While selecting a view, it is also the responsibility of the controller to pass the model data.

In the MVC design pattern, the only responsibility of view is to render the model data. So, in MVC, the view is the component whose responsibility is to generate the necessary HTML in order to render the model data. Once the HTML is generated by the view, then that HTML is then sent to the client over the network, who initially made the request.

So, the three major components of an ASP.NET Core MVC Application are Model, View, and Controller. Let us discuss each of these components of the MVC design pattern in detail.

## Model:

The Model is the component in the MVC Design pattern which is used to manage that data i.e. state of the application in memory. The Model represents a set of classes that are used to describe the application's validation logic, business logic, and data access logic. So in our example, the model consists of Student class and the StudentBusinessLayer class.

```
public class Student
```

```
{
    public int StudentID { get; set; }
    public string Name { get; set; }
    public string Gender { get; set; }
    public string Branch { get; set; }
    public string Section { get; set; }
}

public class StudentBusinessLayer
{
    public IEnumerable<Student> GetAll()
    {
        //logic to return all employees
    }

    public Student GetById(int StudentID)
    {
        //logic to return an employee by employeeId
        Student student = new Student()
        {
            StudentID = StudentID,
            Name = "James",
            Gender = "Male",
            Branch = "CSE",
            Section = "A2",
        };
        return student;
    }

    public void Insert(Student student)
    {
        //logic to insert an student
    }

    public void Update(Student student)
    {

```

```
//logic to Update an student
}
public void Delete(int StudentID)
{
//logic to Delete an student
}
}
```

Here, in our example, we use the Student class to hold the student data in memory. The StudentBusinessLayer class is used to manage the student data i.e. going to perform the CRUD operation.

So, in short, we can say that a Model in MVC design pattern contains a set of classes that is used to represent the data and also contains the logic to manage those data. In our example, the Student class is the class that is used to represent the data. The StudentBusinessLayer class is the class that is used to manage the Student data.

## View:

The view component in the MVC Design pattern is used to contain the logic to represent the model data as a user interface with which the end-user can interact. Basically, the view is used to render the domain data (i.e. business data) which is provided to it by the controller.

For example, we want to display Student data in a web page. In the following example, the Student model carried the student data to the view. As already discussed, the one and only responsibility of the view is to render that student data. The following code does the same thing.

```
@model ASPCoreApplication.Models.Student
<html>
<head>
<title>Student Details</title>
</head>
<body>
<br/>
<br/>
<table>
<tr>
<td>Student ID: </td>
<td>@Model.StudentID</td>
</tr>
<tr>
<td>Name: </td>
<td>@Model.Name</td>
</tr>
<tr>
<td>Gender: </td>
<td>@Model.Gender </td>
</tr>
<tr>
<td>Branch: </td>
<td>@Model.Branch</td>
```

```
</tr>
<tr>
<td>Section: </td>
<td>@Model.Section </td>
</tr>
</table>
</body>
</html>
```

## Controller:

A Controller is a .cs (for C# language) file which has some methods called Action Methods. When a request comes on the controller, it is the action method of the controller which will handle those requests.

The Controller is the component in an MVC application that is used to handle the incoming HTTP Request and based on the user action, the respective controller will work with the model and view and then sends the response back to the user who initially made the request. So, it is the one that will interact with both the models and views to control the flow of application execution. In our example, when the user issued a request the following URL

<http://dotnettutorials.net/student/details/2>

Then that request is mapped to the Details action method of the Student Controller. How it will map to the Details action method of the Student Controller that will discuss in our upcoming articles.

```
public class StudentController : Controller
{
    public ActionResult Details(int studentId)
    {
        StudentBusinessLayer studentBL = new StudentBusinessLayer();
        Student studentDetail = studentBL.GetById(studentId);
        return View(studentDetail);
    }
}
```

As you can see in the example, the Student Controller creates the Student object within the Details action method. So, here the Student is the Model. To fetch the Student data from the database, the controller uses the StudentBusinessLayer class.

Once the controller creates the Student model with the necessary student data, then it passes that Student model to the Details view. The Details view then generates the necessary HTML in order to present the Student data. Once the HTML is generated, then this HTML is sent to the client over the network who initially made the request.

**Note:** In the MVC design pattern both the Controller and View depend on the Model. But the Model never depends on either view or controller. This is one

of the main reasons for the separation of concerns. This separation of concerns allows us to build the model and test independently of the visual presentation.



## Where MVC is used in the real-time three-layer application?

In general, a real-time application may consist of the following layers

1. **Presentation Layer:** This layer is responsible for interacting with the user.
2. **Business Layer:** This layer is responsible for implementing the core business logic of the application.
3. **Data Access Layer:** This layer is responsible for interacting with the database to perform the CRUD operations.

The MVC design pattern is basically used to implement the Presentation Layer of the application. Please have a look at the following diagram.

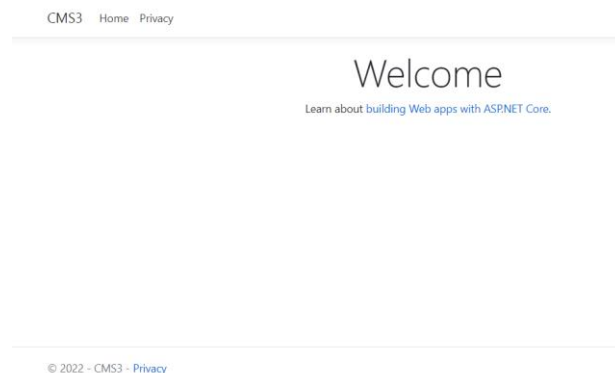


## What is ASP.NET Core MVC?

The ASP.NET Core MVC is a lightweight, open-source, highly testable presentation framework that is used for building web apps and Web APIs using the Model-View-Controller (MVC) design pattern. So, the point that you need to remember is, MVC is a design pattern and ASP.NET Core MVC is the framework that is based on MVC Design Pattern.

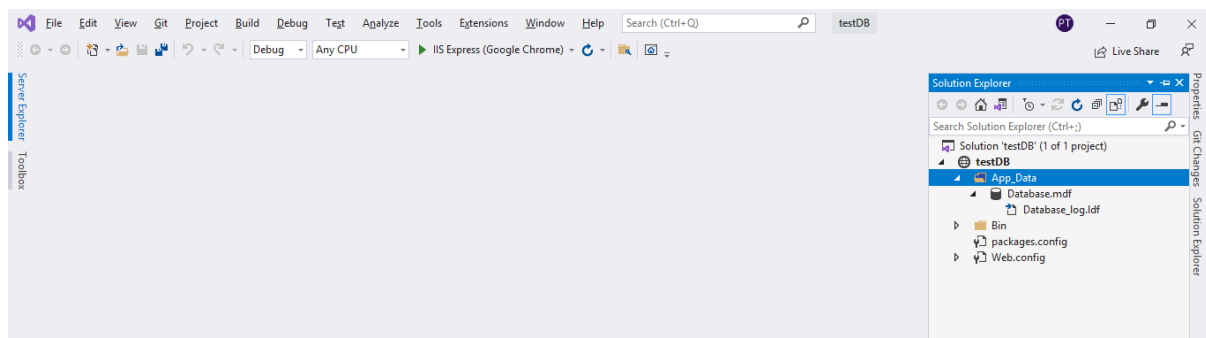
The ASP.NET Core MVC Framework provides us with a patterns-based way to develop dynamic websites and web apps with a clean separation of concerns. This ASP.NET Core MVC framework provides us the full control over the mark-up. It also supports for Test-Driven Development and also uses the latest web standards.

# Exercise : 1 : Create a MVC Core project and run default implementation

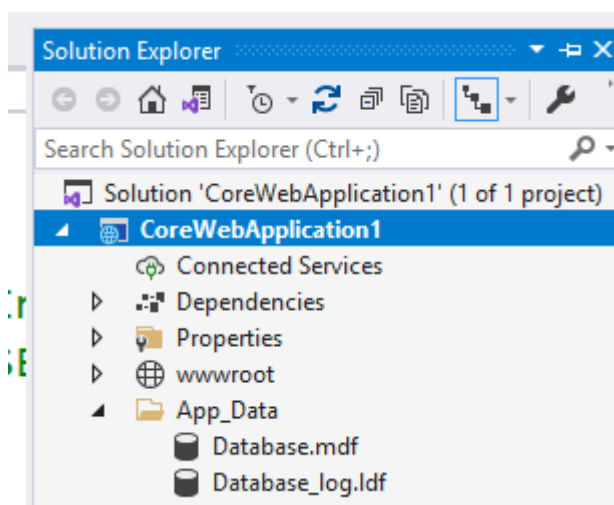


## Solution: Campus Management System

Step : 1 : Create a new website and add database into it (i.e. within App\_Data Folder)



Step : 2 : Create new .net core 3.1 application and paste above database into this project.



Step : 3 : Open this database and add following tables into it.

Table Name : tbl\_emp

The screenshot shows the SQL Server Enterprise Designer interface for a table named `tbl_emp`. The top pane displays the table's design with columns: `ID` (int, primary key, not null), `Name` (varchar(50), not null), `Dept` (varchar(50), not null), and `Salary` (int, not null). The bottom pane shows the T-SQL script for creating the table.

Name	Data Type	Allow Nulls	Default
ID	int	<input type="checkbox"/>	
Name	varchar(50)	<input checked="" type="checkbox"/>	
Dept	varchar(50)	<input checked="" type="checkbox"/>	
Salary	int	<input checked="" type="checkbox"/>	

```
1 CREATE TABLE [dbo].[tbl_emp] (  
2     [ID] INT IDENTITY (1, 1) NOT NULL,  
3     [Name] VARCHAR (50) NULL,  
4     [Dept] VARCHAR (50) NULL,  
5     [Salary] INT NULL,  
6     CONSTRAINT [PK_tbl_emp] PRIMARY KEY CLUSTERED ([ID] ASC)  
7 );  
8
```

Table Name : tbl\_stud

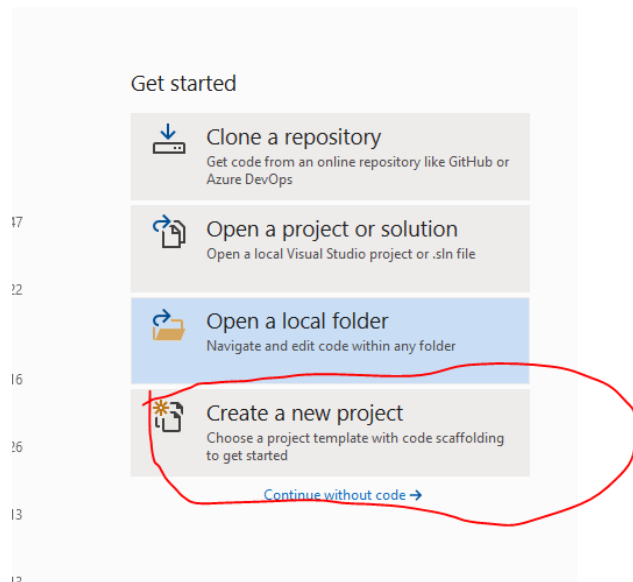
The screenshot shows the SQL Server Enterprise Designer interface for a table named `tbl_stud`. The top pane displays the table's design with columns: `Id` (int, primary key, not null), `Name` (varchar(50), not null), `Email` (varchar(50), not null), `Age` (int, not null), and `Sem` (int, not null). The bottom pane shows the T-SQL script for creating the table.

Name	Data Type	Allow Nulls	Default
Id	int	<input type="checkbox"/>	
Name	varchar(50)	<input checked="" type="checkbox"/>	
Email	varchar(50)	<input checked="" type="checkbox"/>	
Age	int	<input checked="" type="checkbox"/>	
Sem	int	<input checked="" type="checkbox"/>	

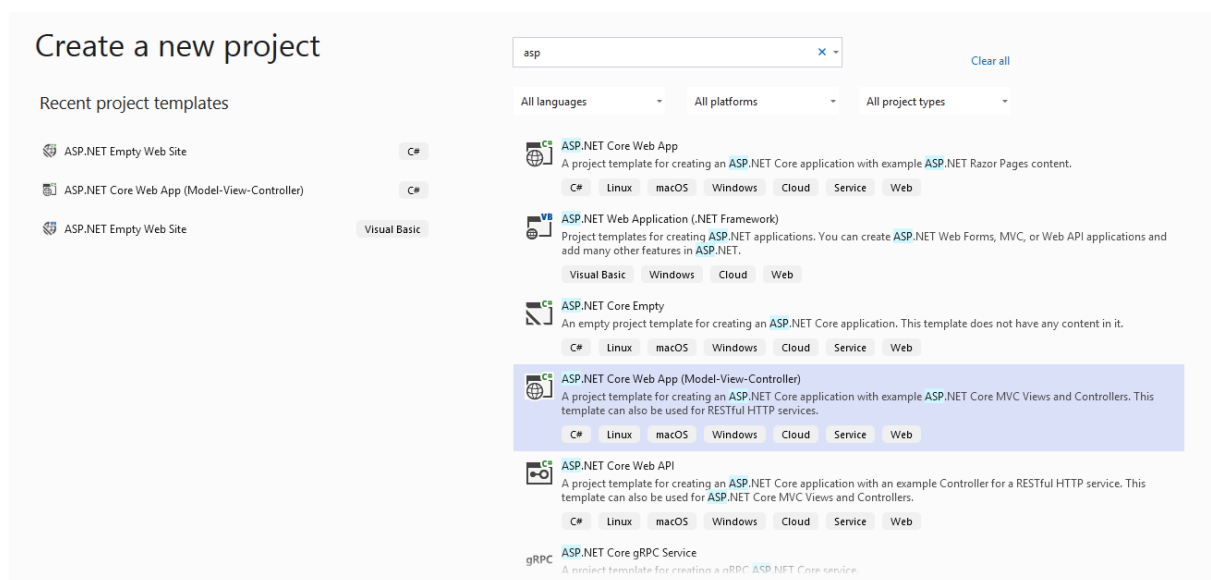
```
1 CREATE TABLE [dbo].[tbl_stud] (  
2     [Id] INT NOT NULL,  
3     [Name] VARCHAR (50) NULL,  
4     [Email] VARCHAR (50) NULL,  
5     [Age] INT NULL,  
6     [Sem] INT NULL,  
7     PRIMARY KEY CLUSTERED ([Id] ASC)  
8 );  
9
```

Step : 4 : Create following page in

Open Visual Studio and Select Create New Project:



➔ Select ASP.NET Core Web App (Model – View – Controller)



➔ Write project name and choose suitable path :

# Configure your new project

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

C#

Linux

macOS

Windows

Cloud

Service

Web

Project name

CMS

Location

D:\ASPNET\

Solution name

CMS

☐ Place solution and project in the same directory

→ Choose proper framework and press Create

## 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

None

☒ Configure for HTTPS

☐ Enable Docker

Docker OS

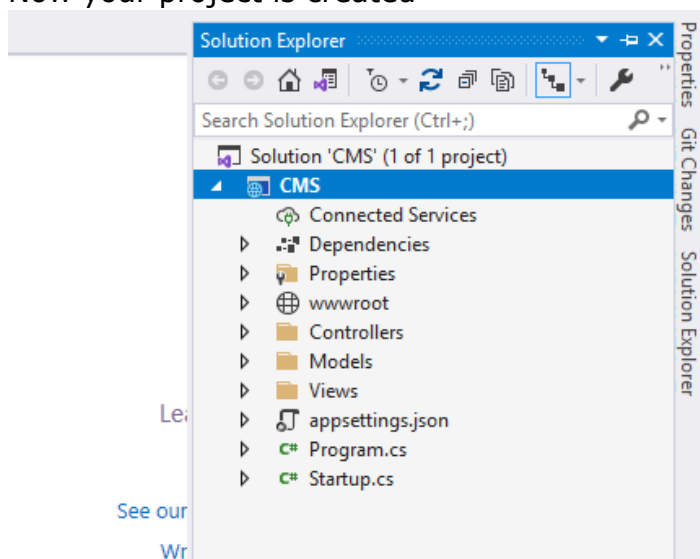
Linux

☐ Enable Razor runtime compilation

Back

Create

→ Now your project is created



You can find here three different folders :

## Models :

Model is a part of the application which implements the logic for the data domain of the application. It is used to retrieve and store model state in a database such as SQL Server database. It also used for business logic separation from the data in the application.

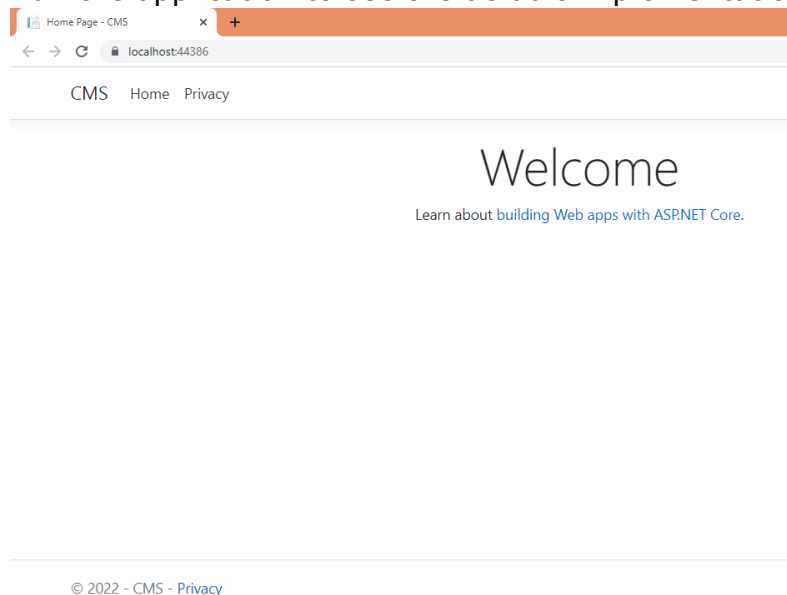
## Views:

View is a component that forms the application's user interface. It is used to create web pages for the application. An example would be an edit view of a Products table that displays text boxes, drop-down lists and checkboxes based on the current state of a Product object.

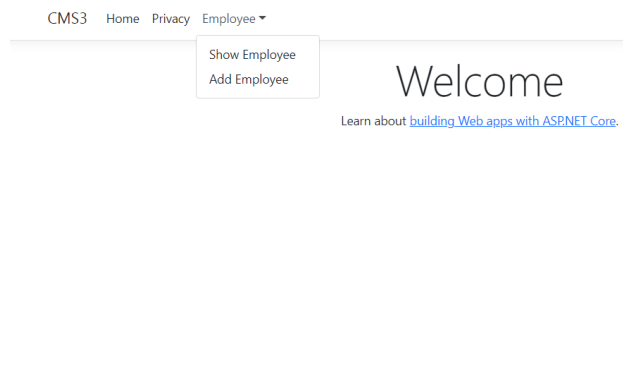
## Controllers:

Controller is the component which handles user interaction. It works with the model and selects the view to render the web page. In an MVC application, the view only displays information whereas the controller handles and responds to the user input and requests.

➔ Run the application to see the default implementation



## Exercise : 2 : Add Employee menu (Add Employee, Show Employees) into layout page (master page)



### Solution :

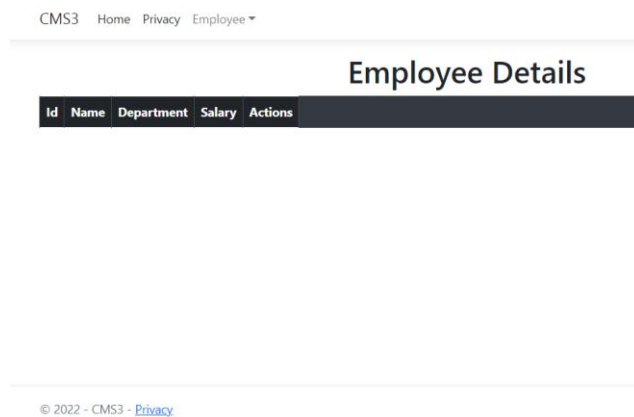
Change “\_Layout.cshtml” by adding yellow colour highlighted text into it.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <title>@ViewData["Title"] - CMS</title>
  <link rel="stylesheet" href="~/lib/bootstrap/dist/css/bootstrap.min.css" />
  <link rel="stylesheet" href="~/lib/bootstrap/dist/css/bootstrap.css" />
  <link rel="stylesheet" href="~/css/site.css" />
  <script src="~/lib/bootstrap/dist/js/bootstrap.js"></script>
  <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.1/dist/css/bootstrap.min.css" rel="stylesheet">
  <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.1/dist/js/bootstrap.bundle.min.js"></script>
</head>
<body>
  <header>
    <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="Home" asp-action="Index">CMS</a>
        <button class="navbar-toggler" type="button" data-toggle="collapse" data-target=".navbar-collapse" aria-controls="navbarSupportedContent"
          aria-expanded="false" aria-label="Toggle navigation">
          <span class="navbar-toggler-icon"></span>
        </button>
        <div class="navbar-collapse collapse d-sm-inline-flex justify-content-between">
          <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="Home" asp-action="Privacy">Privacy</a>
            </li>
            <li class="nav-item dropdown">
              <a class="nav-link dropdown-toggle" href="#" role="button" data-bs-toggle="dropdown">Employee</a>
              <ul class="dropdown-menu">
                <li><a class="dropdown-item" asp-action="Index" asp-controller="Employee">Show Employee</a></li>
                <li><a class="dropdown-item" asp-action="AddEmployee" asp-controller="Employee">Add Employee</a></li>
              </ul>
            </li>
          </ul>
        </div>
      </div>
    </nav>
  </header>
  <div class="container">
    <main role="main" class="pb-3">
      @RenderBody()
    </main>
  </div>

  <footer class="border-top footer text-muted">
    <div class="container">
      &copy; 2022 - CMS - <a asp-area="" asp-controller="Home" asp-action="Privacy">Privacy</a>
    </div>
  </footer>
  <script src="~/lib/jquery/dist/jquery.min.js"></script>
  <script src="~/lib/bootstrap/dist/js/bootstrap.bundle.min.js"></script>
  <script src="~/js/site.js" asp-append-version="true"></script>
  @await RenderSectionAsync("Scripts", required: false)
</body>
</html>
```



## Exercise : 3 : Display Show Employee page as below



### Solution:

Right click on Controllers → Add → Controller → Select "MVC Controller – Empty" → Add

Give the name as "EmployeeController.cs"

Default code for EmployeeController.cs will be as following :

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

namespace CMS.Controllers
{
    public class EmployeeController : Controller
    {
        public IActionResult Index()
        {
            return View();
        }
    }
}
```

Open the controller file. Right click on "Index" Action → Select Add View → Select "Razor View " → Press Add

×

Add Razor View

View name

Index

Template

Empty (without model)

Model class

Options

☐ Create as a partial view

☐ Reference script libraries

☒ Use a layout page

~/Views/Shared/\_Layout.cshtml

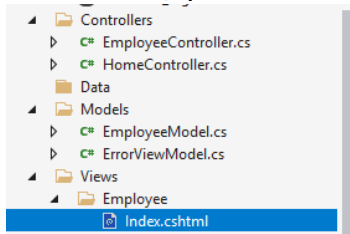
...

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

Add

Cancel

Choose layout page as above and Press Add  
As a result, you will find following in the views



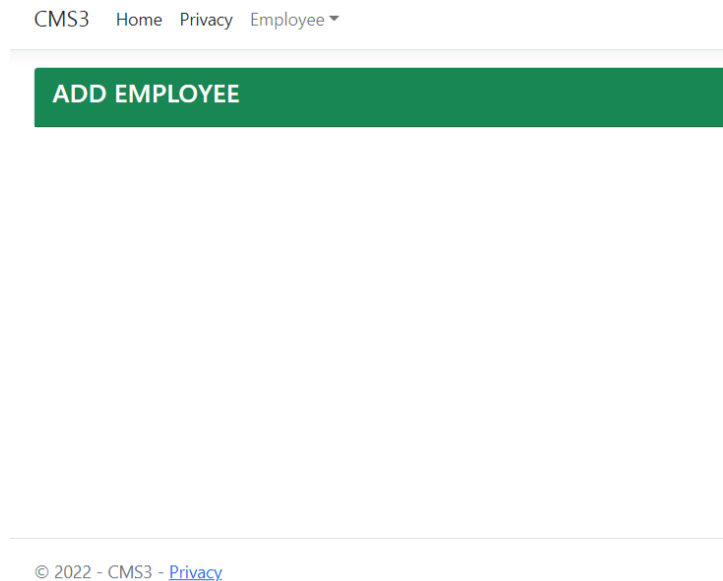
Open index.cshtml file and change content to following:

Change the "index.cshtml" file to the following:

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

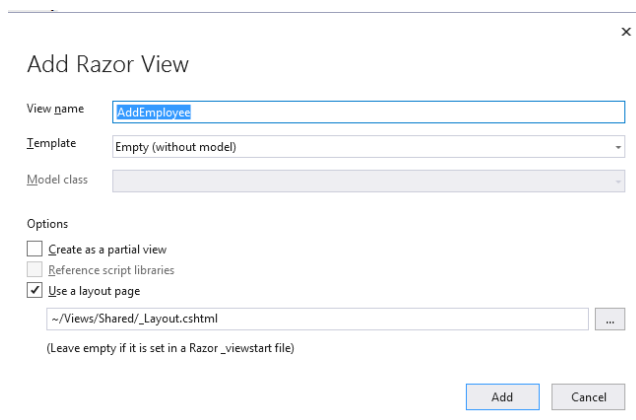
<h1 style="text-align:center;">Employee Details</h1>
```

## Exercise : 4 : Display Add Employee page as below

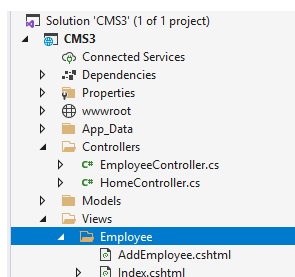


### Solution :

Right click on "Views->Employee" → Select Add View →  
Select "Razor View " → Press Add



Change the name and Choose layout page as above and Press Add  
As a result, you will find following in the views



Open AddEmployee.cshtml file and change content to following:  
Change the "AddEmployee.cshtml" file to the following:

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

```
<div class="card">
    <div class="card-header bg-success text-white text-uppercase">
        <h4>Add Employee</h4>
    </div>
</div>
```

Open the "EmployeeController.cs" file and change the content as below:

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

namespace CMS.Controllers
{
    public class EmployeeController : Controller
    {
        public IActionResult Index()
        {
            return View();
        }

        public IActionResult AddEmployee()
        {
            return View();
        }
    }
}
```

## Exercise : 5 : Display Employee data by clicking on Show Employee option

CMS3   Home   Privacy   Employee ▼

Employee Details

Id	Name	Department	Salary	Actions
2	Rakesh	ce	150000	<a href="#">Edit</a> <a href="#">Delete</a>

© 2022 - CMS3 - [Privacy](#)

## Solution :

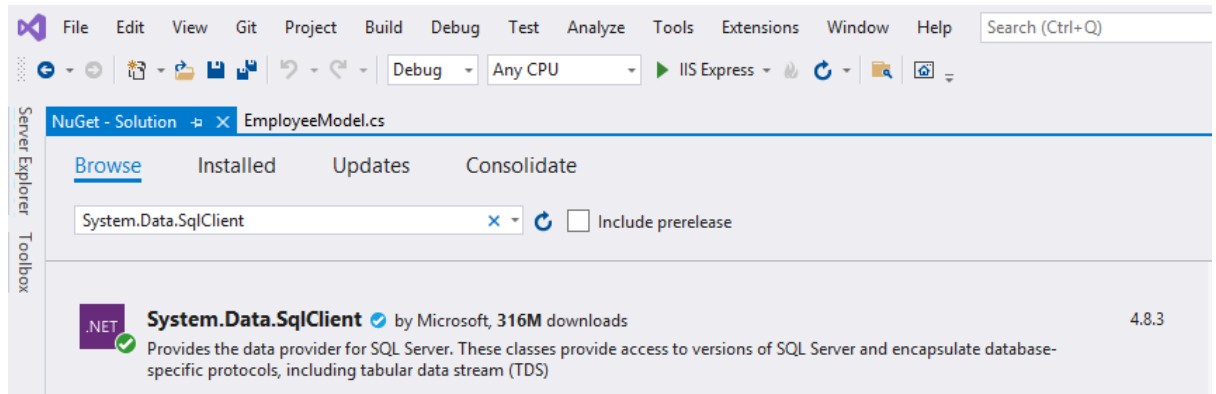
Right click on Models folder -> select Add -> New Item -> Class -> Give the filename as "EmployeeModel.cs" -> press Add button

You will get the following code by default.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace CMS.Models
{
    public class EmployeeModel
    {
    }
}
```

- ➔ Update EmployeeModel.cs file to the following :
  - 1<sup>st</sup> Add field details (Note : must have same name like in table)
  - 2<sup>nd</sup> Add methods for insert update, delete, getData (single record), getData(all records)
- ➔ Install System.Data.SqlClient using NuGet



Finally file will be as following:

# EmployeeModel.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.Data;
using System.Data.SqlClient;
using System.Linq;
using System.Threading.Tasks;

namespace CMS.Models
{
    public class EmployeeModel
    {
        //SQL Server Database Connection
        //SqlConnection con = new SqlConnection(@"Data Source =
        PARESHSIR\MSSQLSERVER2014;Initial Catalog = employee; Integrated Security = True");

        //MySQL Connection
        //SqlConnection con = new SqlConnection(@"Server =
        localhost;Database=employee;user=root");

        //Database MDF file connection
        SqlConnection con = new SqlConnection(@"Data Source =
        (LocalDB)\MSSQLLocalDB;AttachDbFilename=D:\ASPNET\CMS\CMS\CoreWebApplication1\Ap
        p_Data\Database.mdf;Integrated Security = True");
        //SqlConnection con = new SqlConnection(@"Data Source =
        (LocalDB)\MSSQLLocalDB;AttachDbFilename=|DataDirectory|\Database.mdf;Integrated
        Security = True");

        public int Id { get; set; }

        [Required(ErrorMessage = "Please enter name")]
        public string Name { get; set; }

        [Required(ErrorMessage = "Please enter Dept")]
        public string Department { get; set; }

        [Required(ErrorMessage = "please enter salary")]
        [Range(5000, 50000, ErrorMessage = "Value should be between 5k to 50k")]
        public int Salary { get; set; }

        //Retrieve all records from a table
        public List<EmployeeModel> getData()
        {
            List<EmployeeModel> lstEmp = new List<EmployeeModel>();

            SqlDataAdapter apt = new SqlDataAdapter("select * from tbl_emp", con);
            DataSet ds = new DataSet();
            apt.Fill(ds);

            foreach (DataRow dr in ds.Tables[0].Rows)
            {
                lstEmp.Add(new EmployeeModel { Id = Convert.ToInt32(dr["Id"].ToString()),
                Name = dr["Name"].ToString(), Department = dr["Dept"].ToString(), Salary =
                Convert.ToInt32(dr["Salary"].ToString()) });
            }

            return lstEmp;
        }

        //Retrieve single record from a table
    }
}
```



```

public EmployeeModel getData(string Id)
{
    EmployeeModel emp = new EmployeeModel();
    SqlCommand cmd = new SqlCommand("select * from tbl_emp where id=" + Id +
    "", con);

    con.Open();
    SqlDataReader dr = cmd.ExecuteReader();
    if (dr.HasRows)
    {
        while (dr.Read())
        {
            emp.Id = Convert.ToInt32(dr["Id"].ToString());
            emp.Name = dr["Name"].ToString();
            emp.Department = dr["Dept"].ToString();
            emp.Salary = Convert.ToInt32(dr["Salary"].ToString());
        }
    }
    con.Close();
    return emp;
}

//Insert a record into a database table
public bool insert(EmployeeModel Emp)
{
    SqlCommand cmd = new SqlCommand("insert into tbl_emp
values(@name,@dept,@salary)", con);
    cmd.Parameters.AddWithValue("@name", Emp.Name);
    cmd.Parameters.AddWithValue("@dept", Emp.Department);
    cmd.Parameters.AddWithValue("@salary", Emp.Salary);
    con.Open();
    int i = cmd.ExecuteNonQuery();
    if (i >= 1)
    {
        return true;
    }

    return false;
}

//Update a record into a database table
public bool update(EmployeeModel Emp)
{
    SqlCommand cmd = new SqlCommand("update tbl_emp set Name=@name,
Dept=@dept, Salary=@salary where Id = @id", con);
    cmd.Parameters.AddWithValue("@name", Emp.Name);
    cmd.Parameters.AddWithValue("@dept", Emp.Department);
    cmd.Parameters.AddWithValue("@salary", Emp.Salary);
    cmd.Parameters.AddWithValue("@id", Emp.Id);
    con.Open();
    int i = cmd.ExecuteNonQuery();
    if (i >= 1)
    {
        return true;
    }

    return false;
}

//delete a record from a database table

```

```

public bool delete(EmployeeModel Emp)
{
    SqlCommand cmd = new SqlCommand("delete tbl_emp where Id = @id", con);
    cmd.Parameters.AddWithValue("@id", Emp.Id);
    con.Open();
    int i = cmd.ExecuteNonQuery();
    if (i >= 1)
    {
        return true;
    }
    return false;
}

//[Required(ErrorMessage = "Please enter Name")]
//[Display(Name = "Enter Name:")]
//public string Name { get; set; }
//[Required]
//public string Department { get; set; }
//[Required(ErrorMessage = "Please enter Salary")]
//[Range(2000, 10000, ErrorMessage = "Salary must be between 2k to 10K")]
//public int Salary { get; set; }
}
}

```

Open index.cshtml (from Employee view) file and change content to following:

```

@using CMS.Models
@model IEnumerable<EmployeeModel>
@{
    ViewData["Title"] = "Index";
    Layout = "~/Views/Shared/_Layout.cshtml";
}

<div>
    <h1 style="text-align:center;">Employee Details</h1>

    <table class="table table-bordered table-responsive table-dark table-hover">

        <tr>
            <th>Id</th>
            <th>Name</th>
            <th>Department</th>
            <th>Salary</th>
            <th>Actions</th>
        </tr>

        @foreach (EmployeeModel item in Model)
        {
            <tr>
                <td>@item.Id</td>
                <td>@item.Name</td>
                <td>@item.Department</td>
                <td>@item.Salary</td>
                <td>
                    <a asp-action="EditEmployee" asp-route-id="@item.Id">Edit</a>
                    <a asp-action="DeleteEmployee" asp-route-id="@item.Id">Delete</a>
                </td>
            </tr>
        }
    </table>

```

```

        </td>
    </tr>
}

</table>

</div>

```

Open "EmployeeController.cs" file and change the content to the following:

```

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

namespace CMS.Controllers
{
    public class EmployeeController : Controller
    {
        EmployeeModel empObj = new EmployeeModel();

        public IActionResult Index()
        {
            empObj = new EmployeeModel();
            List<EmployeeModel> lst = empObj.getData();

            return View(lst);
        }

        public IActionResult AddEmployee()
        {
            return View();
        }
    }
}

```

**Exercise : 6 : Show following form by clicking on “Add Employee” option. Also, when you click on submit button after entering data into fields, show successful message and then redirect to the show employees form.**

-----

CMS3 Home Privacy Employee ▾

ADD EMPLOYEE

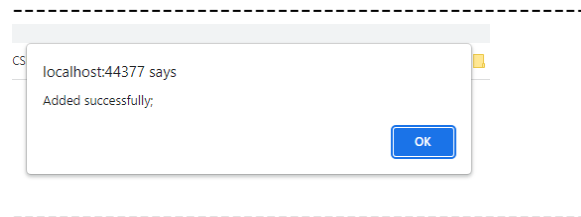
Name:

Department

Salary

Submit

© 2022 - CMS3 - [Privacy](#)



-----

CMS3 Home Privacy Employee \*

Employee Details

Id	Name	Department	Salary	Actions
2	Rakesh	ce	150000	<a href="#">Edit</a> <a href="#">Delete</a>
4	Tejas	IT	52000	<a href="#">Edit</a> <a href="#">Delete</a>

© 2022 - CMS3 - [Privacy](#)

-----

## Solution :

Open AddEmployee.cshtml (from Employee view) file and change content to following:

```
@model CMS.Models.EmployeeModel
@{
    ViewData["Title"] = "AddEmployee";
    Layout = "~/Views/Shared/_Layout.cshtml";
}

<div class="card">
```

```

<div class="card-header bg-success text-white text-uppercase">
  <h4>Add Employee</h4>
</div>
<div class="card-body">
  <form asp-action="AddEmployee">
    <div class="form-group">
      <label for="name">Name:</label>
      <input type="text" asp-for="Name" class="form-control" id="Name">
      <span asp-validation-for="Name" class="text-danger"></span>
    </div>
    <div class="form-group">
      <label for="Department">
        Department
      </label>
      <input type="text" asp-for="Department" class="form-control"
id="Department">
      <span asp-validation-for="Department" class="text-danger"></span>
    </div>
    <div class="form-group">
      <label for="Salary">
        Salary
      </label>
      <input type="text" asp-for="Salary" class="form-control" id="Salary">
      <span asp-validation-for="Salary" class="text-danger"></span>
    </div>
    <button type="submit" class="btn btn-success rounded-0">Submit</button>
  </form>

  @if (TempData["msg"] != null)
  {
    <script>
      alert('@TempData["msg"].ToString());';
      window.location.href = "@Url.Action("Index", "Employee")";
    </script>
  }
</div>
</div>

```

Open "EmployeeController.cs" file and change the content to the following:

```

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

namespace CMS.Controllers
{
  public class EmployeeController : Controller
  {
    EmployeeModel empObj = new EmployeeModel();

    public IActionResult Index()
    {
      empObj = new EmployeeModel();
      List<EmployeeModel> lst = empObj.getData();

      return View(lst);
    }

    public IActionResult AddEmployee()

```

```

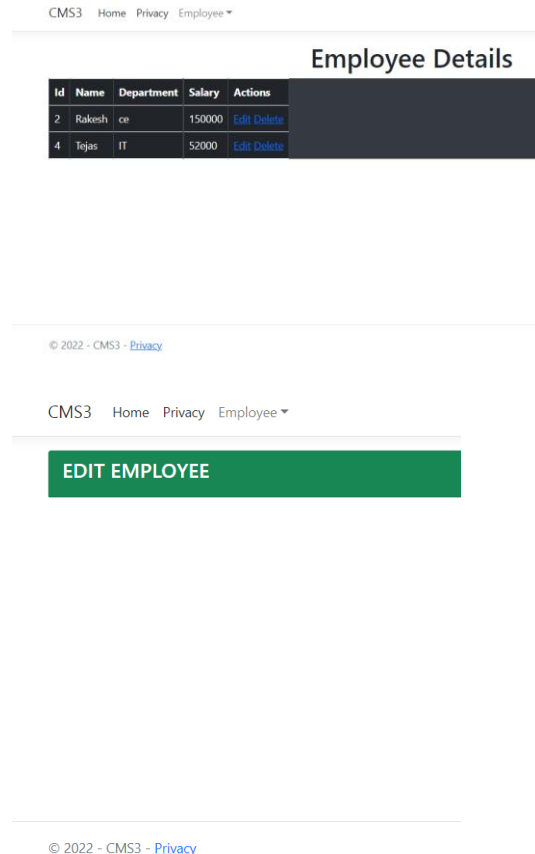
{
    return View();
}

[HttpPost]
public IActionResult AddEmployee(EmployeeModel emp)
{
    bool res;
    //if (ModelState.IsValid)
    //{
        empObj = new EmployeeModel();
        res = empObj.insert(emp);
        if (res)
        {
            TempData["msg"] = "Added successfully";
        }
        //}
    else
    {
        TempData["msg"] = "Not Added. something went wrong..!!";
    }

    return View();
}
}
}

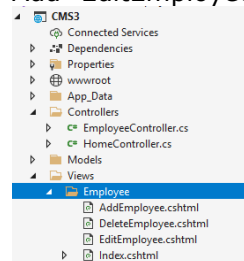
```

## Exercise : 7 : Show following form by clicking of “Edit” link into the Show Employee view.



## Solution :

Add “EditEmployee” view and change the content as following:



### EditEmployee.cshtml

```
@{  
    ViewData["Title"] = "EditEmployee";  
    Layout = "~/Views/Shared/_Layout.cshtml";  
}  
  
<div class="card">  
    <div class="card-header bg-success text-white text-uppercase">  
        <h4>Edit Employee</h4>  
    </div>  
</div>
```

Open “EmployeeController.cs” file and change the content to the following:



```

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

namespace CMS.Controllers
{
    public class EmployeeController : Controller
    {
        EmployeeModel empObj = new EmployeeModel();

        public IActionResult Index()
        {
            empObj = new EmployeeModel();
            List<EmployeeModel> lst = empObj.getData();

            return View(lst);
        }

        public IActionResult AddEmployee()
        {
            return View();
        }

        [HttpPost]
        public IActionResult AddEmployee(EmployeeModel emp)
        {
            bool res;
            //if (ModelState.IsValid)
            //{
                empObj = new EmployeeModel();
                res = empObj.insert(emp);
                if (res)
                {
                    TempData["msg"] = "Added successfully";
                }
            //}
            else
            {
                TempData["msg"] = "Not Added. something went wrong..!!";
            }

            return View();
        }

        [HttpGet]
        public IActionResult EditEmployee(string id)
        {
            EmployeeModel emp = empObj.getData(id);

            return View(emp);
        }
    }
}

```

## Exercise : 8 : Show following form by clicking of “Delete” link into the Show Employee view.

CMS3 Home Privacy Employee ▾

Id	Name	Department	Salary	Actions
2	Rakesh	ce	150000	<a href="#">Edit</a> <a href="#">Delete</a>
4	Tejas	IT	52000	<a href="#">Edit</a> <a href="#">Delete</a>

© 2022 - CMS3 - [Privacy](#)

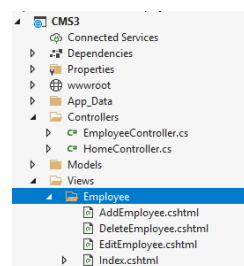
CMS3 Home Privacy Employee ▾

DELETE EMPLOYEE

© 2022 - CMS3 - [Privacy](#)

## Solution :

Add “DeleteEmployee” view and change the content as following:



### DeleteEmployee.cshtml

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

<div class="card">
    <div class="card-header bg-success text-white text-uppercase">
        <h4>Delete Employee</h4>
    </div>
```

</div>

Open "EmployeeController.cs" file and change the content to the following:

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

namespace CMS.Controllers
{
    public class EmployeeController : Controller
    {
        EmployeeModel empObj = new EmployeeModel();

        public IActionResult Index()
        {
            empObj = new EmployeeModel();
            List<EmployeeModel> lst = empObj.getData();

            return View(lst);
        }

        public IActionResult AddEmployee()
        {
            return View();
        }

        [HttpPost]
        public IActionResult AddEmployee(EmployeeModel emp)
        {
            bool res;
            //if (ModelState.IsValid)
            //{
                empObj = new EmployeeModel();
                res = empObj.insert(emp);
                if (res)
                {
                    TempData["msg"] = "Added successfully";
                }
            //}
            else
            {
                TempData["msg"] = "Not Added. something went wrong..!!";
            }

            return View();
        }

        [HttpGet]
        public IActionResult EditEmployee(string id)
        {
            EmployeeModel emp = empObj.getData(id);

            return View(emp);
        }

        [HttpGet]
        public IActionResult DeleteEmployee(string id)
        {
            EmployeeModel emp = empObj.getData(id);
```

```

    return View(emp);
}

}
}

```

**Exercise : 9 : Show following form by clicking of “Edit” link into the Show Employee view. Also, when you click on update button after changing data into fields, show successful message and then redirect to the show employees form.**

CMS3 Home Privacy Employee ▼

Employee Details				
Id	Name	Department	Salary	Actions
2	Rakesh	ce	150000	<a href="#">Edit</a> <a href="#">Delete</a>
4	Tejas	IT	52000	<a href="#">Edit</a> <a href="#">Delete</a>

© 2022 - CMS3 - [Privacy](#)

CMS3 Home Privacy Employee ▼

### EDIT EMPLOYEE

Name

Department

Salary

© 2022 - CMS3 - [Privacy](#)

localhost:44377 says  
Updated successfully;

## Solution :

Open “EditEmployee.cshtml” file and change the content as following:

```
<!--Fetches information for EmployeeModel object-->
```

```

@model CMS.Models.EmployeeModel
@{
    ViewData["Title"] = "EditEmployee";
    Layout = "~/Views/Shared/_Layout.cshtml";
}

```

```
<div class="card">
```

```

<div class="card-header bg-success text-white text-uppercase">
  <h4>Edit Employee</h4>
</div>

<div class="card-body">
  <form asp-action="EditEmployee" asp-controller="Employee" method="post">

    <input type="hidden" asp-for="Id" />

    <div class="row">
      <div class="col-md-6">
        <div class="form-group">
          <label asp-for="Name" class="label-control"></label>
          <input asp-for="Name" class="form-control" />
          <span asp-validation-for="Name" class="text-danger"></span>
        </div>
      </div>
    </div>

    <div class="row">
      <div class="col-md-6">
        <div class="form-group">
          <label asp-for="Department" class="label-control"></label>
          <input asp-for="Department" class="form-control" />
          <span asp-validation-for="Department" class="text-danger"></span>
        </div>
      </div>
    </div>

    <div class="row">
      <div class="col-md-6">
        <div class="form-group">
          <label asp-for="Salary" class="label-control"></label>
          <input asp-for="Salary" class="form-control" />
          <span asp-validation-for="Salary" class="text-danger"></span>
        </div>
      </div>
    </div>

    <div class="form-group">
      <button type="submit" value="Update" class="btn btn-success rounded-0">Update</button>
    </div>

  </form>
  @if (TempData["msg"] != null)
  {
    <script>
      alert('@TempData["msg"].ToString());';
      window.location.href = "@Url.Action("Index","Employee")";
    </script>
  }
</div>
</div>

```

Open "EmployeeController.cs" file and change the content to the following:

```

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

```

```

namespace CMS.Controllers
{
    public class EmployeeController : Controller
    {
        EmployeeModel empObj = new EmployeeModel();

        public IActionResult Index()
        {
            empObj = new EmployeeModel();
            List<EmployeeModel> lst = empObj.getData();

            return View(lst);
        }

        public IActionResult AddEmployee()
        {
            return View();
        }

        [HttpPost]
        public IActionResult AddEmployee(EmployeeModel emp)
        {
            bool res;
            //if (ModelState.IsValid)
            //{
                empObj = new EmployeeModel();
                res = empObj.insert(emp);
                if (res)
                {
                    TempData["msg"] = "Added successfully";
                }
            //}
            else
            {
                TempData["msg"] = "Not Added. something went wrong..!!";
            }

            return View();
        }

        [HttpGet]
        public IActionResult EditEmployee(string id)
        {
            EmployeeModel emp = empObj.getData(id);

            return View(emp);
        }

        [HttpGet]
        public IActionResult DeleteEmployee(string id)
        {
            EmployeeModel emp = empObj.getData(id);

            return View(emp);
        }

        [HttpPost]
        public IActionResult EditEmployee(EmployeeModel emp)
        {
            bool res;
            // if (ModelState.IsValid)
            // {
                empObj = new EmployeeModel();
                res = empObj.update(emp);
            }
        }
    }
}

```

```
        if (res)
        {
            TempData["msg"] = "Updated successfully";
        }
        //}
        else
        {
            TempData["msg"] = "Not Updated. something went wrong..!";
        }

        return View();
    }
}
```

**Exercise : 10 : Show following form by clicking of "Delete" link into the Show Employee view. Also, when you click on delete button, show successful message and then redirect to the show employees form.**

CMS3 Home Privacy Employee ▾

### Employee Details

Id	Name	Department	Salary	Actions
2	Rakesh	ce	150000	<a href="#">Edit</a> <a href="#">Delete</a>
4	Tejas	IT	52000	<a href="#">Edit</a> <a href="#">Delete</a>

© 2022 - CMS3 - [Privacy](#)

CMS3 Home Privacy Employee ▾

### DELETE EMPLOYEE

Name

Department

Salary

© 2022 - CMS3 - [Privacy](#)

localhost:44377 says

Deleted successfully;

## Solution :

Open "DeleteEmployee.cshtml" file and change the content as following:

```
@model CMS.Models.EmployeeModel
@{
    ViewData["Title"] = "DeleteEmployee";
    Layout = "~/Views/Shared/_Layout.cshtml";
}
```

```
<div class="card">
    <div class="card-header bg-success text-white text-uppercase">
```



```

        <h4>Delete Employee</h4>
    </div>

    <div class="card-body">
        <form asp-action="DeleteEmployee" asp-controller="Employee" method="post">

            <input type="hidden" asp-for="Id" />

            <div class="row">
                <div class="col-md-6">
                    <div class="form-group">
                        <label asp-for="Name" class="label-control"></label>
                        <input asp-for="Name" class="form-control" />
                        <span asp-validation-for="Name" class="text-danger"></span>
                    </div>
                </div>
            </div>

            <div class="row">
                <div class="col-md-6">
                    <div class="form-group">
                        <label asp-for="Department" class="label-control"></label>
                        <input asp-for="Department" class="form-control" />
                        <span asp-validation-for="Department" class="text-danger"></span>
                    </div>
                </div>
            </div>

            <div class="row">
                <div class="col-md-6">
                    <div class="form-group">
                        <label asp-for="Salary" class="label-control"></label>
                        <input asp-for="Salary" class="form-control" />
                        <span asp-validation-for="Salary" class="text-danger"></span>
                    </div>
                </div>
            </div>

            <div class="form-group">
                <button type="submit" value="Delete" class="btn btn-success rounded-0">Delete</button>
            </div>

        </form>
        @if (TempData["msg"] != null)
        {
            <script>
                alert('@TempData["msg"].ToString());
                window.location.href = "@Url.Action("Index","Employee")";
            </script>
        }
    </div>
</div>

```

Open "EmployeeController.cs" file and change the content to the following:

```

using CMS.Models;
using Microsoft.AspNetCore.Mvc;
using System;
using System.Collections.Generic;
using System.Linq;

```

```

using System.Threading.Tasks;

namespace CMS.Controllers
{
    public class EmployeeController : Controller
    {
        EmployeeModel empObj = new EmployeeModel();

        public IActionResult Index()
        {
            empObj = new EmployeeModel();
            List<EmployeeModel> lst = empObj.getData();

            return View(lst);
        }

        public IActionResult AddEmployee()
        {
            return View();
        }

        [HttpPost]
        public IActionResult AddEmployee(EmployeeModel emp)
        {
            bool res;
            //if (ModelState.IsValid)
            //{
                empObj = new EmployeeModel();
                res = empObj.insert(emp);
                if (res)
                {
                    TempData["msg"] = "Added successfully";
                }
            //}
            else
            {
                TempData["msg"] = "Not Added. something went wrong..!!";
            }

            return View();
        }

        [HttpGet]
        public IActionResult EditEmployee(string id)
        {
            EmployeeModel emp = empObj.getData(id);

            return View(emp);
        }

        [HttpGet]
        public IActionResult DeleteEmployee(string id)
        {
            EmployeeModel emp = empObj.getData(id);

            return View(emp);
        }

        [HttpPost]
        public IActionResult EditEmployee(EmployeeModel emp)
        {
            bool res;
            // if (ModelState.IsValid)
            // {

```

```

empObj = new EmployeeModel();
res = empObj.update(emp);
if (res)
{
    TempData["msg"] = "Updated successfully";
}
//}
else
{
    TempData["msg"] = "Not Updated. something went wrong..!!";
}

return View();
}

[HttpPost]
public IActionResult DeleteEmployee(EmployeeModel emp)
{
    bool res;
    //if (ModelState.IsValid)
    //{
        empObj = new EmployeeModel();
        res = empObj.delete(emp);
        if (res)
        {
            TempData["msg"] = "Deleted successfully";
        }
        //}
        else
        {
            TempData["msg"] = "Not Deleted. something went wrong..!!";
        }
    }

    return View();
}
}
}

```