

Module - 3

Controllers, Models and Views

Is MVC = ASP.NET MVC?

Before starting we should clarify the doubt most of us go through while studying .NET MVC framework i.e. are MVC and ASP.NET MVC same? The answer is **NO**.

MVC is a concept while ASP.NET MVC is a framework which uses that concept.

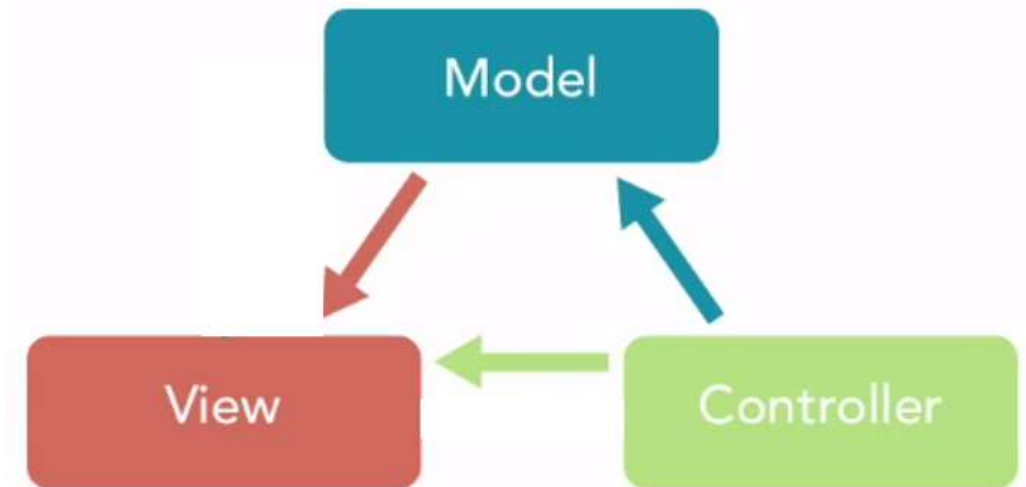
What is MVC?

- MVC is a design pattern
- MVC pattern is used for building web, mobile, desktop applications etc.
- All the major technologies are using MVC to design web application using MVC design pattern.
- MVC is used by all the technologies like Java, .NET, PHP, Angular etc.

What is ASP.NET MVC?

- ASP.NET MVC is a .NET framework which follows MVC design pattern.
- ASP.NET MVC is used for web development.
- ASP.NET MVC provides everything required to build a web application using MVC design pattern.

The primary purpose of the MVC is that each of these layers can be developed and tested independently and then combined together to form a robust application.



MVC stands for Model, View, and Controller.
MVC separates an application into three components - Model, View, and Controller.

Model -

Model represents the shape of the data. A class in C# is used to describe a model. Model objects store data retrieved from the database.

Model represents the data.

View -

View in MVC is a user interface. View display model data to the user and also enables them to modify them. View in ASP.NET MVC is HTML, CSS, and some special syntax (Razor syntax) that makes it easy to communicate with the model and the controller.

View is the User Interface.

Controller -

The controller handles the user request. Typically, the user uses the view and raises an HTTP request, which will be handled by the controller. The controller processes the request and returns the appropriate view as a response.

Controller is the request handler

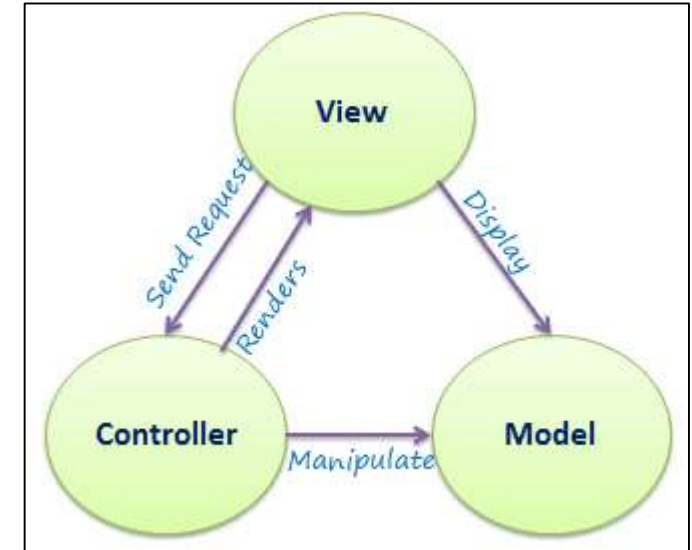
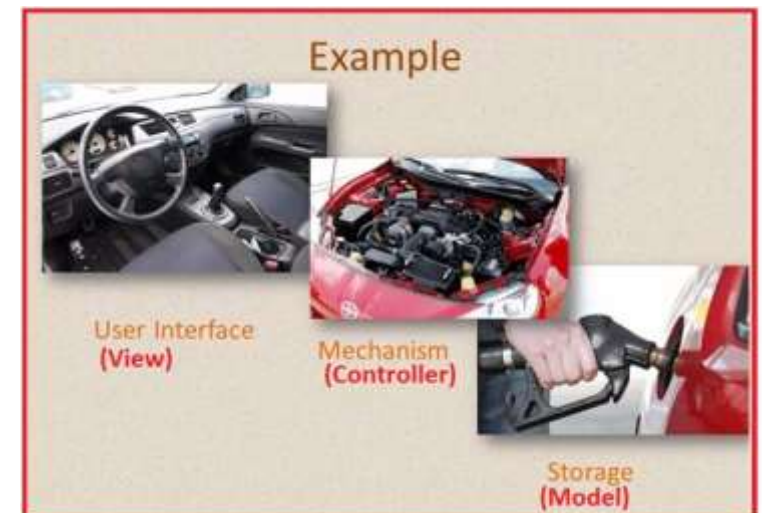


Fig: Interaction between Model, View, and Controller



Explanation

User will make request for the page which user would like to retrieve. Requested page will go to controller and on controller Route.Config will be checked. Requested page will get transfer to Model from Controller.

On Model there will be 2 steps,

1. Page initialization will get started.
2. Then result set will be generated.

After these operation result set will get unload to View through view engine.

Responsibility of View Engine

- View Engine get the request from Controller
- The requested page will get executed
- The result got by the execution process will get deliver to View.

The Properties of MVC is ,

- Loose Coupling
- Lightweight code
- Effective look
- Testing is Very Easy
- Rapid application integrated development.

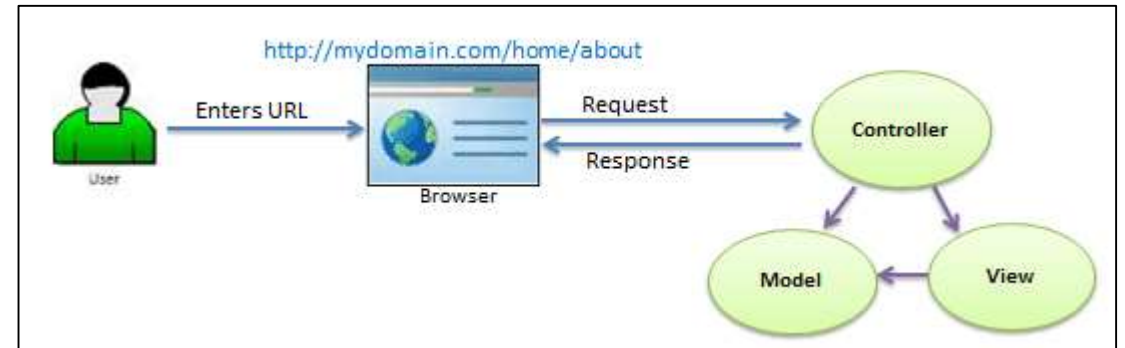


Fig: Request Flow in MVC Architecture

Controller

- The Controller in MVC architecture handles any incoming URL request. The Controller is a class, derived from the base class `System.Web.Mvc.Controller`. Controller class contains public methods called Action methods. Controller and its action method handles incoming browser requests, retrieves necessary model data and returns appropriate responses.
- In ASP.NET MVC, every controller class name must end with a word "Controller". For example, the home page controller name must be HomeController, and for the employee page, it must be the EmployeeController. Also, every controller class must be located in the Controller folder of the MVC folder structure.

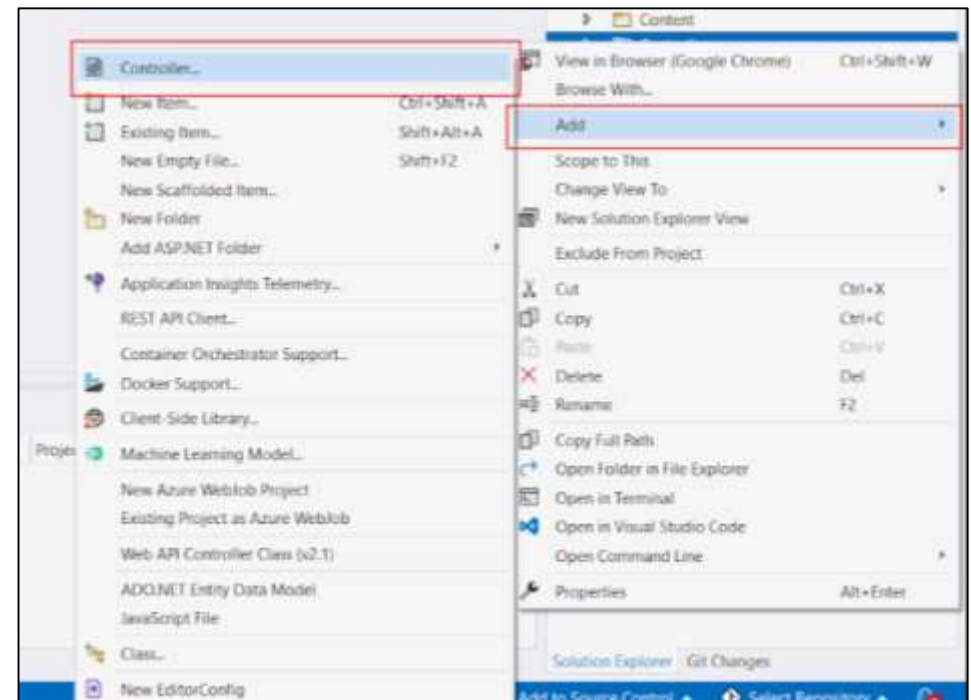
Adding new controller –

Lets add empty controller in our MVC application.

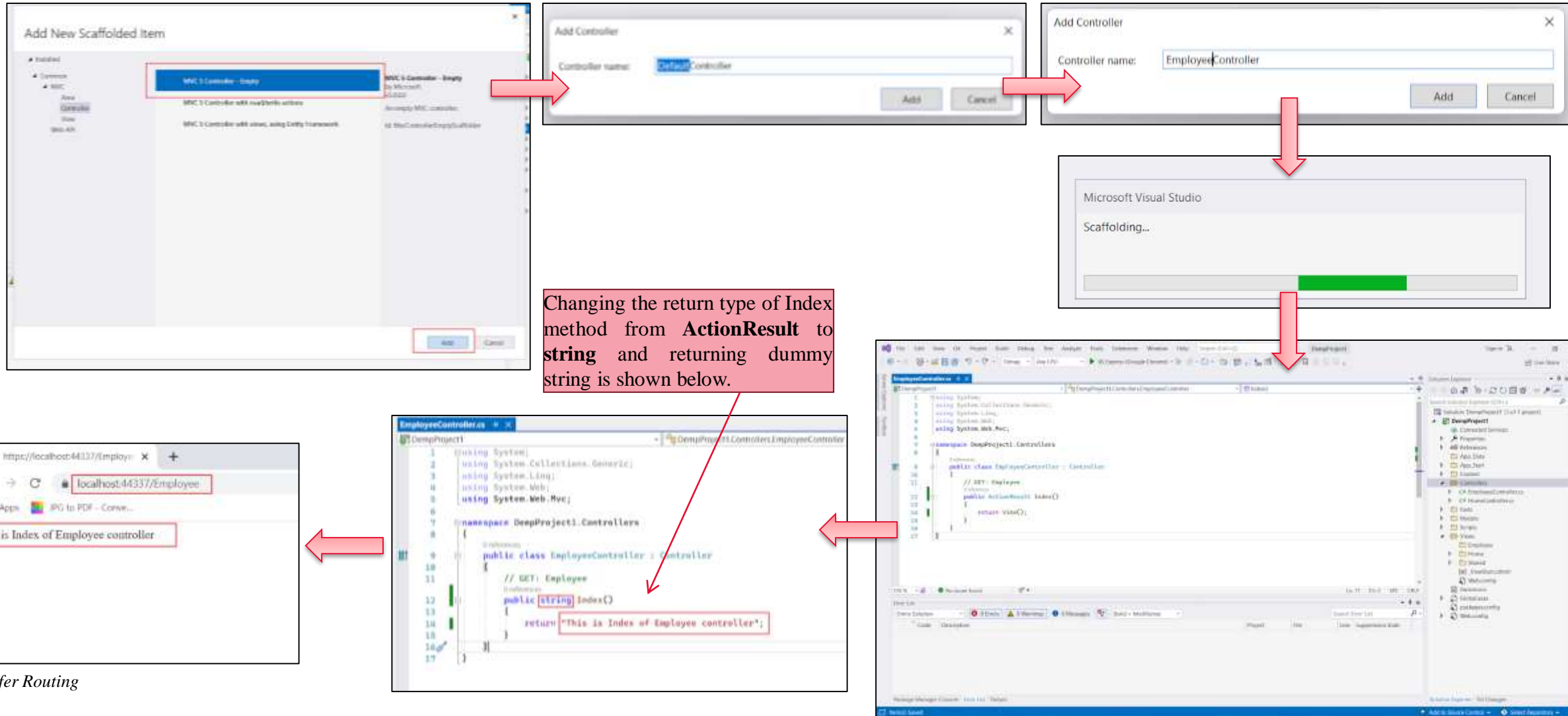
In the Visual Studio, right click on the Controller folder -> select **Add** -> click on **Controller..**

This open scaffold dialog box

NOTE: ASP.NET Scaffolding is a code generation framework for ASP.NET Web applications. Using scaffolding can reduce the amount of time to develop standard data operations in your project. You can develop a customized scaffolding template using T4 templates as per your architecture and coding standards.



- Add New Scaffold dialog contains different templates to create a new controller.



*Refer Routing

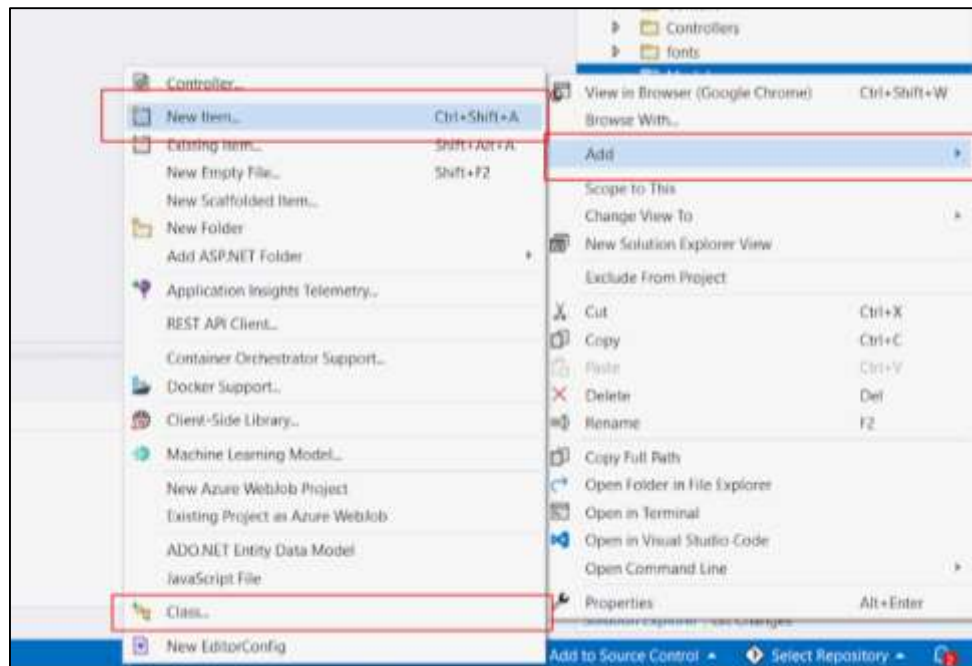
Model

- The model classes represents domain-specific data and business logic in the MVC application. It represents the shape of the data as public properties and business logic as methods.
- In the ASP.NET MVC Application, all the Model classes must be created in the Model folder.

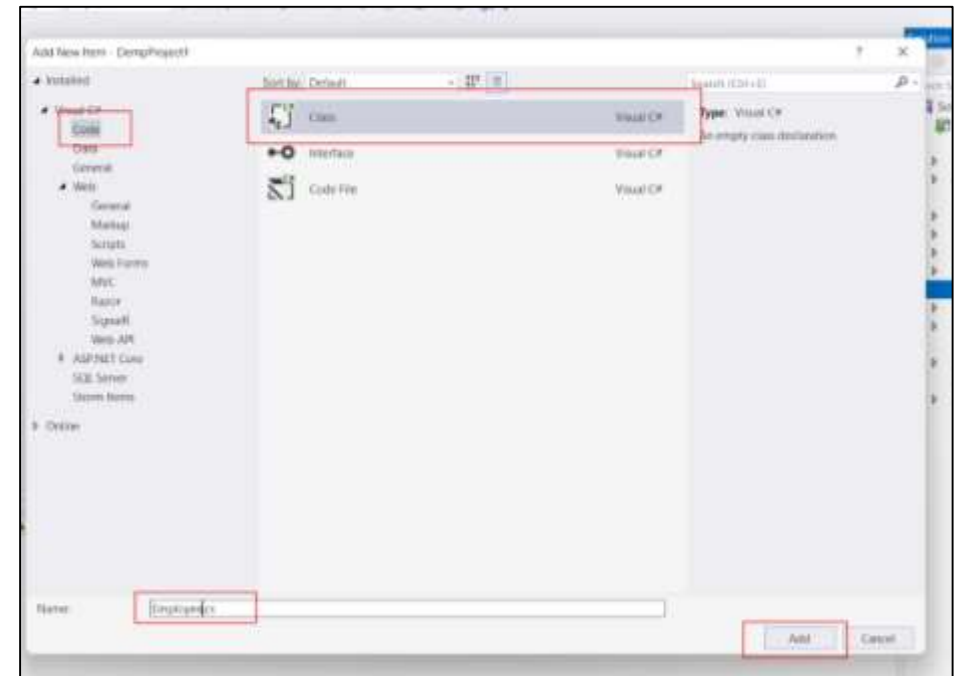
Adding Models –

Lets create a model class which has the properties of **Employee** entity.

- In the MVC application in Visual Studio, and right-click on the Model folder, select **Add ->** and click on **Class...** It will open the **Add New Item** dialog box.



- In the Add New Item dialog box, enter the class name Student and click **Add**.

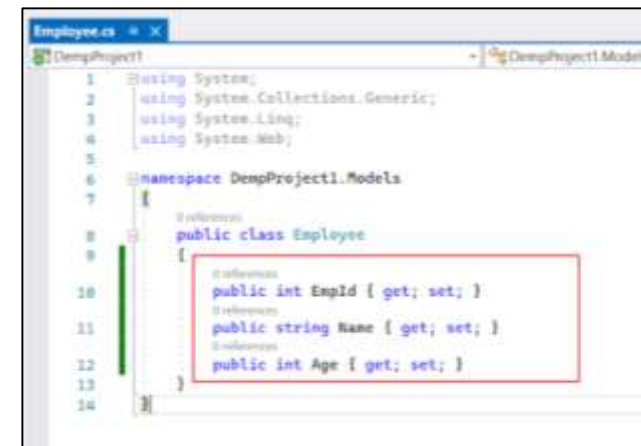


This will add a new Employee class in model folder.



We want this model class to store id, name, and age of the Employee. So, we will have to add public properties for Id, Name, and Age, as shown below.

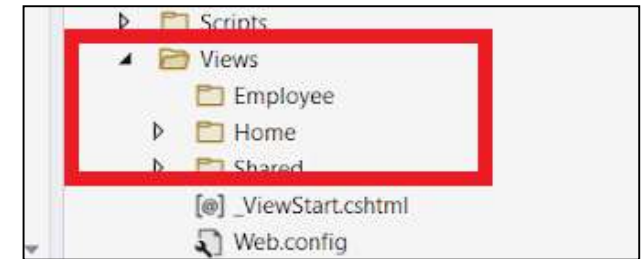
The model class can be used in the view to populate the data, as well as sending data to the controller.



Let's create a view and use this model

View

- A view is used to display data using the model class object. The Views folder contains all the view files in the ASP.NET MVC application.
- A controller can have one or more action methods, and each action method can return a different view. In short, a controller can render one or more views. So, for easy maintenance, the MVC framework requires a separate sub-folder for each controller with the same name as a controller, under the Views folder.
- For example, all the views rendered from the HomeController will reside in the Views > Home folder. In the same way, views for EmployeeController will reside in Views > Employee folder, as shown below.



Razor view engine

Razor view engine in ASP.NET MVC is the syntax that allows us to write server-side code on view. It helps to combine code and HTML in a fluid manner. Razor is not a new programming language.

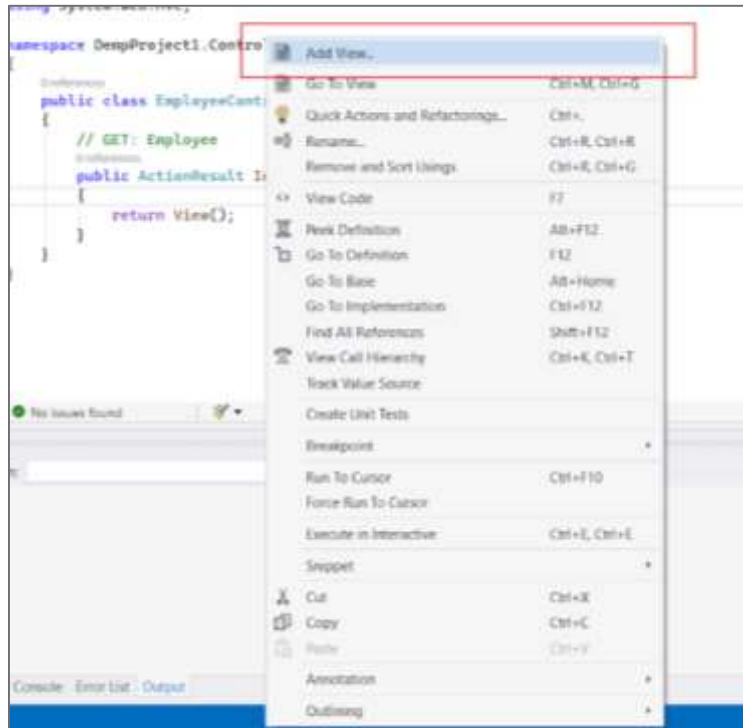
The razor view uses @ character to include the server-side code instead of the traditional <% %> of ASP. We can use C# or Visual Basic syntax to write server-side code inside the razor view.(we will use it later)

File Extension	Description
.cshtml	C# Razor view. Supports C# code with html tags
.vbhtml	Visual Basic Razor view. Supports visual basic code with html tags
.aspx	ASP.NET web form
.ascx	ASP.NET web control

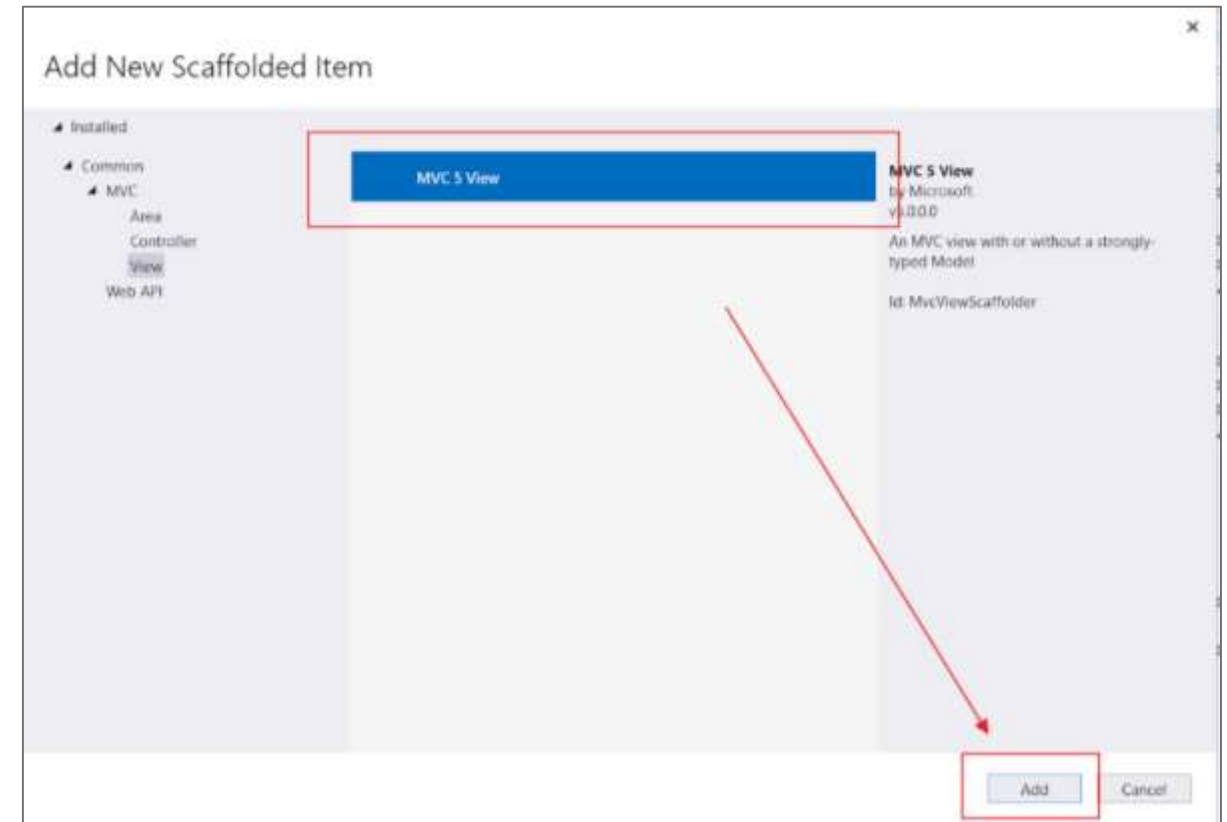
Table: ASP.NET MVC supports the above types of razor view files

■ Creating View –

We can create directly by right clicking on index action method of the controller and then clicking Add View.



To create an empty view without any model to it we can follow below steps –



This will create an empty index file without any model and we can add a table manually and then display it in the page.

Add View

View name: Index

Template: Empty (without model)

Model class:

Options:

☐ Create as a partial view

☒ Reference script libraries

☒ Use a layout page:

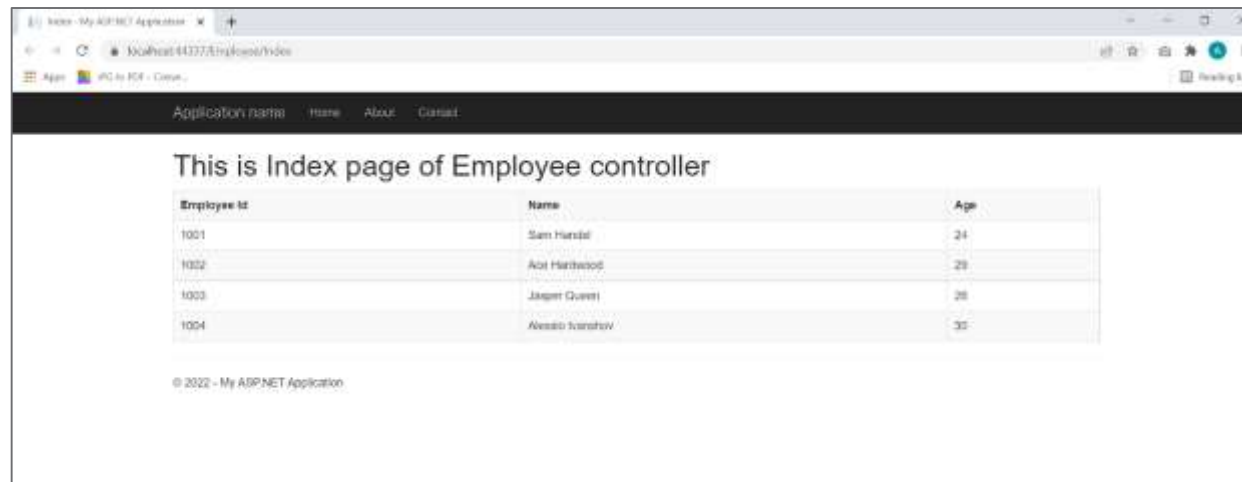
~/Views/Shared/_Layout.cshtml

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

Add Cancel



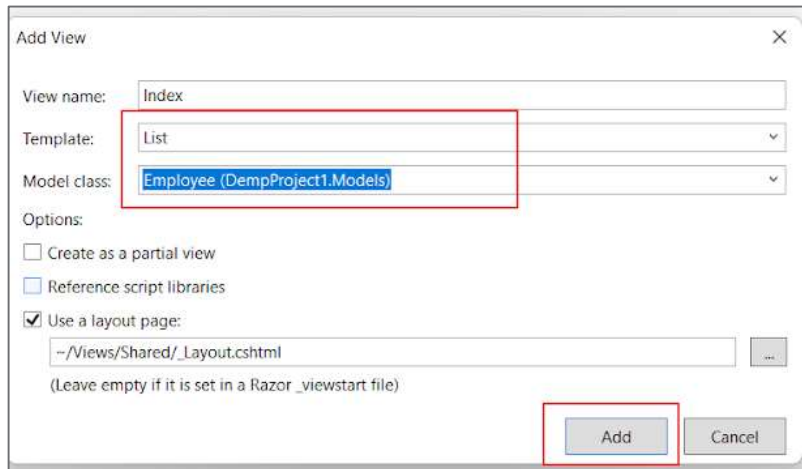
```
1
2
3  @{
4      ViewBag.Title = "Index";
5      Layout = "~/Views/Shared/_Layout.cshtml";
6  }
7
8  <h1>This is Index page of Employee controller</h1>
9  <table class="table table-bordered table-striped">
10     <tr>
11         <th>Employee Id</th>
12         <th>Name</th>
13         <th>Age</th>
14     </tr>
15     <tr>
16         <td>1001</td>
17         <td>Sam Handel</td>
18         <td>24</td>
19     </tr>
20     <tr>
21         <td>1002</td>
22         <td>Ace Hardwood</td>
23         <td>29</td>
24     </tr>
25     <tr>
26         <td>1003</td>
27         <td>Jasper Queen</td>
28         <td>26</td>
29     </tr>
30     <tr>
31         <td>1004</td>
32         <td>Alessia Ivanshov</td>
33         <td>30</td>
34     </tr>
35 </table>
```



**Manually created table*

Integrate Model, View and Controller

- We have already created model, view and controller in previous section but they all were separate. Now we will integrate all three of them together.
- First add a list type of index file instead of without model type view. It will automatically add a basic create, edit, delete and update options and a table to be created.



Add View

View name: Index

Template: List

Model class: Employee (DempProject1.Models)

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

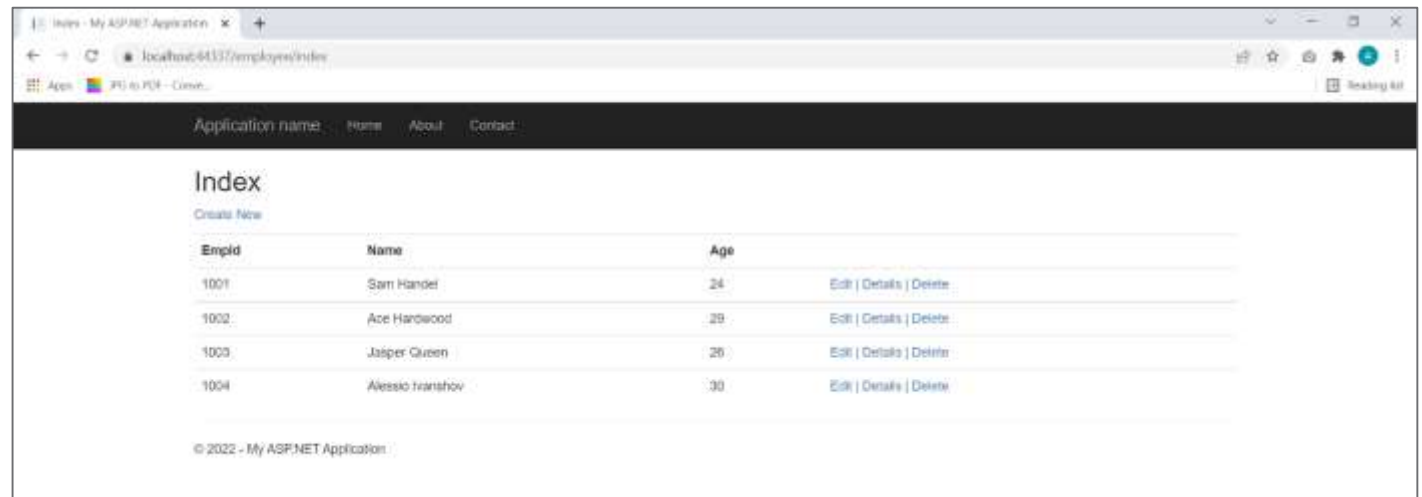


```
1 @model IEnumerable<DempProject1.Models.Employee>
2
3
4 ViewBag.Title = "Index";
5 Layout = "~/Views/Shared/_Layout.cshtml";
6
7
8 <h2>Index</h2>
9
10
11 @Html.ActionLink("Create New", "Create")
12
13 <table class="table">
14     <tr>
15         <th>
16             @Html.DisplayNameFor(model => model.Id)
17         </th>
18         <th>
19             @Html.DisplayNameFor(model => model.Name)
20         </th>
21         <th>
22             @Html.DisplayNameFor(model => model.Age)
23         </th>
24         <th></th>
25     </tr>
26
27     <tr>
28         <td>
29             @Html.DisplayFor(modelItem => item.Id)
30         </td>
31         <td>
32             @Html.DisplayFor(modelItem => item.Name)
33         </td>
34         <td>
35             @Html.DisplayFor(modelItem => item.Age)
36         </td>
37         <td>
38             @Html.ActionLink("Edit", "Edit", new { id=item.PrimaryKey }) |
39             @Html.ActionLink("Details", "Details", new { id=item.PrimaryKey }) |
40             @Html.ActionLink("Delete", "Delete", new { id=item.PrimaryKey })
41         </td>
42     </tr>
43 </table>
```

- Then add a list in controller file. Also add model to the controller

```
Index.cshtml EmployeeController.cs x DempProject1 DempProject1.Controllers.EmployeeController
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Web;
5 using System.Web.Mvc;
6 using DempProject1.Models;
7
8 namespace DempProject1.Controllers
9
10 {
11     public class EmployeeController : Controller
12     {
13         static IList<Employee> employeeList = new List<Employee>
14         {
15             new Employee() {EmpId = 1001, Name= "Sam Handel", Age=24 },
16             new Employee() {EmpId = 1002, Name= "Ace Hardwood", Age=29 },
17             new Employee() {EmpId = 1003, Name= "Jasper Queen", Age=26 },
18             new Employee() {EmpId = 1004, Name= "Alessio Ivanshov", Age=30 }
19         };
20         // GET: Employee
21         public ActionResult Index()
22         {
23             return View(employeeList);
24         }
25     }
26 }
```

- Save and run to get the following.



THANK YOU