TASK-2

Introduction

In this article we are going to create a web application using ASP.NET Core MVC with the help of Visual Studio Code and ADO.NET. We will be creating a sample Employee Record Management System and performing CRUD operations on it.

We will use VS Code and SQL Server for our demo.

Prerequisites

- Install .NET Core 2.0.0 or above SDK from here
- Download and install Visual Studio Code from here
- SQL Server 2008 or above

Source Code

Before proceeding further, I would recommend that you download the source code from GitHub.

Creating the Table and Stored Procedures

We will be using a DB table to store all the records of the employees.

Open SQL Server and use the following script to create **tblEmployee** table.

```
Create table tblEmployee(
    EmployeeId int IDENTITY(1,1) NOT NULL,
    Name varchar(20) NOT NULL,
    City varchar(20) NOT NULL,
    Department varchar(20) NOT NULL,
    Gender varchar(6) NOT NULL
)
```

Now, we will create stored procedures to add, delete, update, and get employee data.

To Insert an Employee Record

```
Create procedure spAddEmployee
(
    @Name VARCHAR(20),
    @City VARCHAR(20),
    @Department VARCHAR(20),
    @Gender VARCHAR(6)
)
as
Begin
    Insert into tblEmployee (Name,City,Department, Gender)
    Values (@Name,@City,@Department, @Gender)
End
```

To Update an Employee Record

```
Create procedure spUpdateEmployee
(
   @EmpId INTEGER ,
   @Name VARCHAR(20),
   @City VARCHAR(20),
   @Department VARCHAR(20),
   @Gender VARCHAR(6)
)
as
```

```
begin
   Update tblEmployee
   set Name=@Name,
   City=@City,
   Department=@Department,
   Gender=@Gender
   where EmployeeId=@EmpId
End
```

To Delete an Employee Record

```
Create procedure spDeleteEmployee
(
   @EmpId int
)
as
begin
   Delete from tblEmployee where EmployeeId=@EmpId
End
```

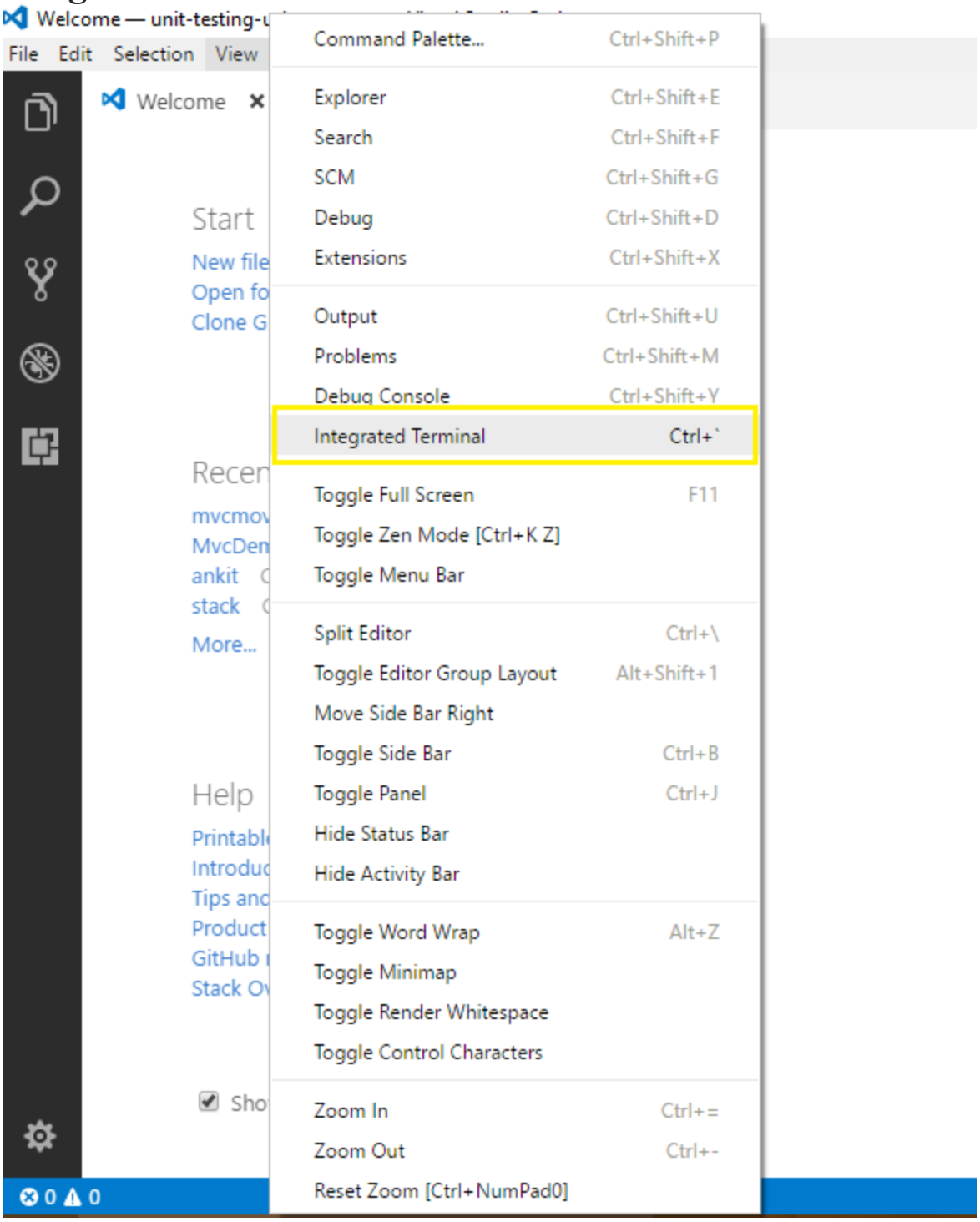
To View all Employee Records

```
Create procedure spGetAllEmployees
as
Begin
    select *
    from tblEmployee
    order by EmployeeId
End
```

Now, our Database part has been completed. So we will proceed to create the MVC application using Visual Studio code.

Create the MVC Web Application

We will be creating a source project from the terminal window in Visual Studio Code. Open VS code and navigate to view >> Integrated Terminal.



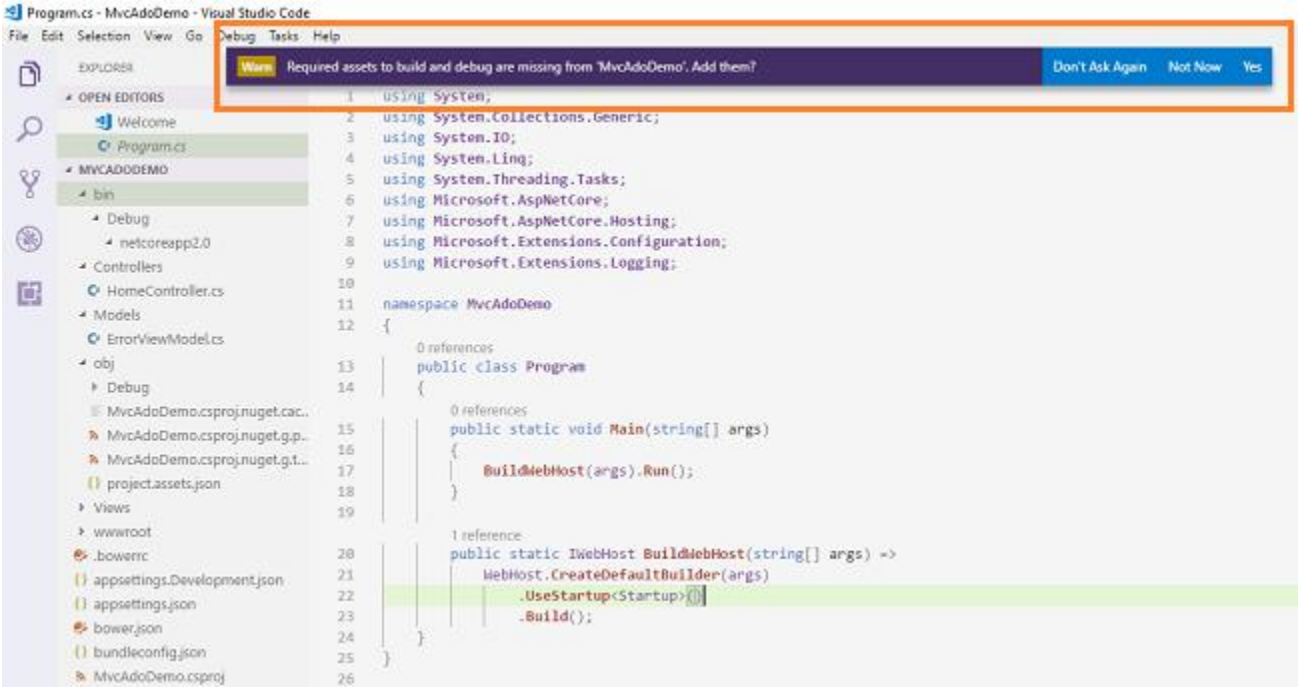
This will open the terminal window as shown in the image below.

Type the following sequence of commands in the terminal window. It will create our MVC application “MvcAdoDemo”.

- mkdir MvcAdoDemo
- cd MvcAdoDemo
- dotnet new mvc

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL

PS C:\Users\ankit> mkdir MvcAdoDemo


    Directory: C:\Users\ankit


Mode                LastWriteTime         Length Name
----                -------------         ------ ----
d-----        18-Nov-17   6:06 PM                MvcAdoDemo


PS C:\Users\ankit> cd .\MvcAdoDemo\
PS C:\Users\ankit\MvcAdoDemo> dotnet new mvc
The template "ASP.NET Core Web App (Model-View-Controller)" was created successfully.
This template contains technologies from parties other than Microsoft, see https://aka.ms/template-3pn for details.

Processing post-creation actions...
Running 'dotnet restore' on C:\Users\ankit\MvcAdoDemo\MvcAdoDemo.csproj...
  Restoring packages for C:\Users\ankit\MvcAdoDemo\MvcAdoDemo.csproj...
  Restore completed in 942.79 ms for C:\Users\ankit\MvcAdoDemo\MvcAdoDemo.csproj.
  Generating MSBuild file C:\Users\ankit\MvcAdoDemo\obj\MvcAdoDemo.csproj.nuget.g.props.
  Generating MSBuild file C:\Users\ankit\MvcAdoDemo\obj\MvcAdoDemo.csproj.nuget.g.targets.
  Restore completed in 31.82 sec for C:\Users\ankit\MvcAdoDemo\MvcAdoDemo.csproj.


Restore succeeded.
```

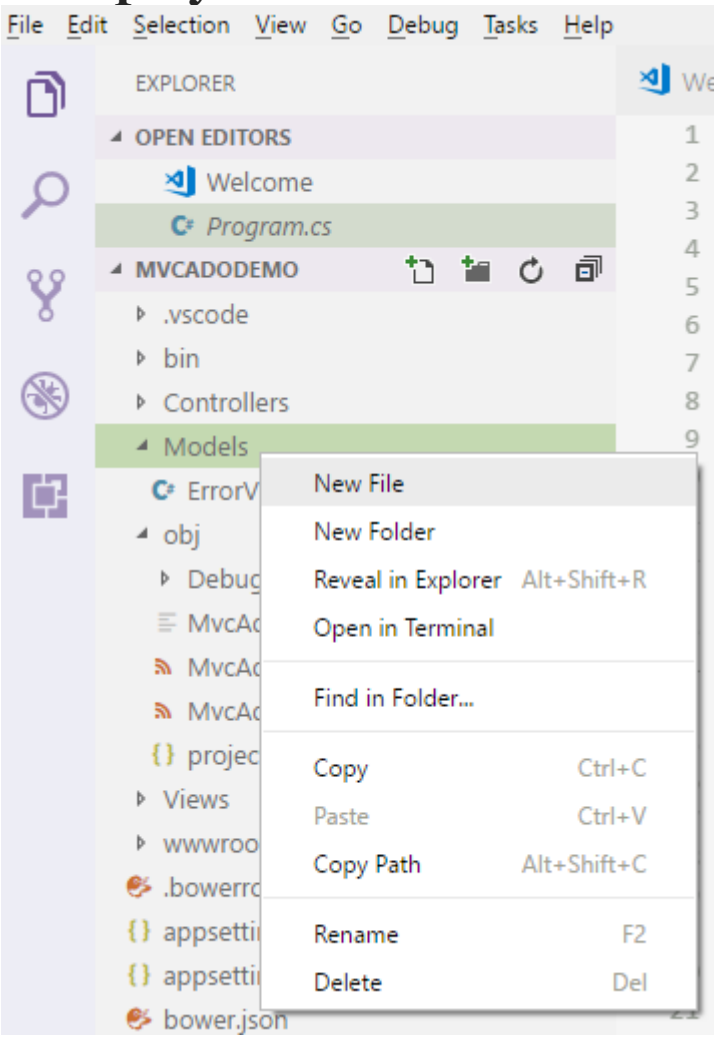
Now open this “MvcAdoDemo” project file using VS code. If it prompts the message “Required assets to build and debug are missing from MvcAdoDemo. Add them?”, select “Yes”.



You can observe in the solution explorer that we already have folders created with the names Controllers, Models, and Views. We will be adding our code files in these folders only.

Adding the Model to the Application

Right click on Models folder and select “New File”. Name it **Employee.cs**. It will create a file inside the Models folder.



Add one more file to the Models folder. Name it **EmployeeDataAccessLayer.cs**. This class will contain our Database-related operations.

Open **Employee.cs** and put following code inside it. Since we are adding the required validators to the fields of Employee class, we need to use System.ComponentModel.DataAnnotations at the top:

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

namespace MVCAdoDemo.Models
{
    public class Employee
    {
        public int ID { get; set; }
        [Required]
        public string Name { get; set; }
        [Required]
        public string Gender { get; set; }
        [Required]
        public string Department { get; set; }
        [Required]
        public string City { get; set; }
    }
}
```

Open **EmployeeDataAccessLayer.cs** and put in the following code to handle database operations. Make sure to put in your own connection string.

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

namespace MVCAdoDemo.Models
{
    public class EmployeeDataAccessLayer
    {
        string connectionString = "Your Connection String
here";

        //To View all employees details
        public IEnumerable<Employee> GetAllEmployees()
        {
            List<Employee> lstemployee = new
List<Employee>();

            using (SqlConnection con = new
SqlConnection(connectionString))
            {
                SqlCommand cmd = new
SqlCommand("spGetAllEmployees", con);
                cmd.CommandType =
CommandType.StoredProcedure;

                con.Open();
                SqlDataReader rdr = cmd.ExecuteReader();

                while (rdr.Read())
                {
                    Employee employee = new Employee();

                    employee.ID =
Convert.ToInt32(rdr["EmployeeID"]);
                    employee.Name = rdr["Name"].ToString();
                    employee.Gender =
rdr["Gender"].ToString();
                    employee.Department =
rdr["Department"].ToString();
```

```
                    employee.City = rdr["City"].ToString();

                    lstemployee.Add(employee);
                }
                con.Close();
            }
            return lstemployee;
        }

        //To Add new employee record
        public void AddEmployee(Employee employee)
        {
            using (SqlConnection con = new
SqlConnection(connectionString))
            {
                SqlCommand cmd = new
SqlCommand("spAddEmployee", con);
                cmd.CommandType =
CommandType.StoredProcedure;

                cmd.Parameters.AddWithValue("@Name",
employee.Name);
                cmd.Parameters.AddWithValue("@Gender",
employee.Gender);
                cmd.Parameters.AddWithValue("@Department",
employee.Department);
                cmd.Parameters.AddWithValue("@City",
employee.City);

                con.Open();
                cmd.ExecuteNonQuery();
                con.Close();
            }
        }

        //To Update the records of a particluar employee
        public void UpdateEmployee(Employee employee)
        {
            using (SqlConnection con = new
SqlConnection(connectionString))
            {
                SqlCommand cmd = new
```

```
SqlCommand("spUpdateEmployee", con);
                cmd.CommandType =
CommandType.StoredProcedure;

                cmd.Parameters.AddWithValue("@EmpId",
employee.ID);
                cmd.Parameters.AddWithValue("@Name",
employee.Name);
                cmd.Parameters.AddWithValue("@Gender",
employee.Gender);
                cmd.Parameters.AddWithValue("@Department",
employee.Department);
                cmd.Parameters.AddWithValue("@City",
employee.City);

                con.Open();
                cmd.ExecuteNonQuery();
                con.Close();
            }
        }

        //Get the details of a particular employee
        public Employee GetEmployeeData(int? id)
        {
            Employee employee = new Employee();

            using (SqlConnection con = new
SqlConnection(connectionString))
            {
                string sqlQuery = "SELECT * FROM
tblEmployee WHERE EmployeeID= " + id;
                SqlCommand cmd = new SqlCommand(sqlQuery,
con);

                con.Open();
                SqlDataReader rdr = cmd.ExecuteReader();

                while (rdr.Read())
                {
                    employee.ID =
Convert.ToInt32(rdr["EmployeeID"]);
                    employee.Name = rdr["Name"].ToString();
```

```
                    employee.Gender =
rdr["Gender"].ToString();
                    employee.Department =
rdr["Department"].ToString();
                    employee.City = rdr["City"].ToString();
                }
            }
            return employee;
        }

        //To Delete the record on a particular employee
        public void DeleteEmployee(int? id)
        {

            using (SqlConnection con = new
SqlConnection(connectionString))
            {
                SqlCommand cmd = new
SqlCommand("spDeleteEmployee", con);
                cmd.CommandType =
CommandType.StoredProcedure;

                cmd.Parameters.AddWithValue("@EmpId", id);

                con.Open();
                cmd.ExecuteNonQuery();
                con.Close();
            }
        }
    }
}
```

To use ADO.NET functionalities in VS code, we need to add the nuget package reference to **System.Data.SqlClient**. Open the **MvcAdoDemo.csproj** file and put the following code into it.

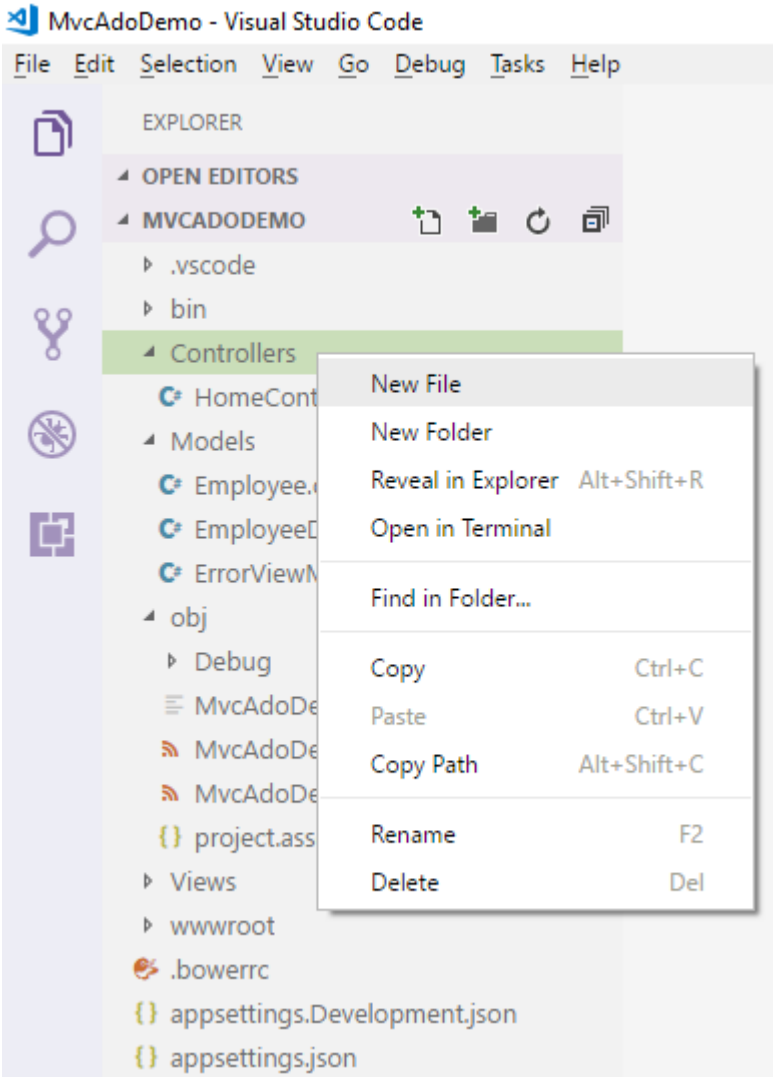
```
<PackageReference Include="System.Data.SqlClient"
Version="4.4.0" />
```

Put this code in the location highlighted in the image below.

```
MvcAdoDemo.csproj ●
1   <Project Sdk="Microsoft.NET.Sdk.Web">
2
3     <PropertyGroup>
4       <TargetFramework>netcoreapp2.0</TargetFramework>
5     </PropertyGroup>
6
7     <ItemGroup>
8       <PackageReference Include="Microsoft.AspNetCore.All" Version="2.0.0" />
9       <PackageReference Include="System.Data.SqlClient" Version="4.4.0" />
10    </ItemGroup>
11
12    <ItemGroup>
13      <DotNetCliToolReference Include="Microsoft.VisualStudio.Web.CodeGeneration.Tools" Version="2.0.0" />
14    </ItemGroup>
15
16  </Project>
```

Adding the Controller to the Application

Right click on the Controllers folder and select “New File”. Name it **EmployeeController.cs**. It will create a new file inside the Controllers folder.

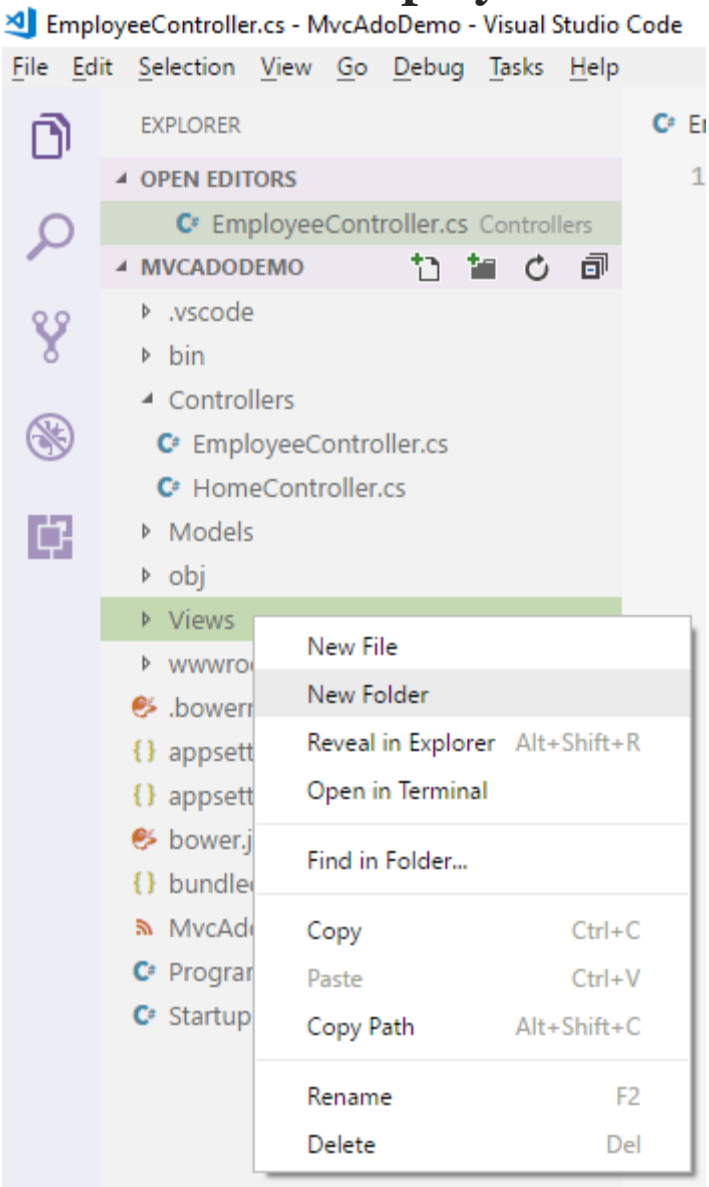


Now our **EmployeeController** has been created. We will put all our business logic in this controller.

Adding Views to the Application

To add views for our controller class, we need to create a folder inside the **Views** folder with the same name as our controller and then add our views to that folder.

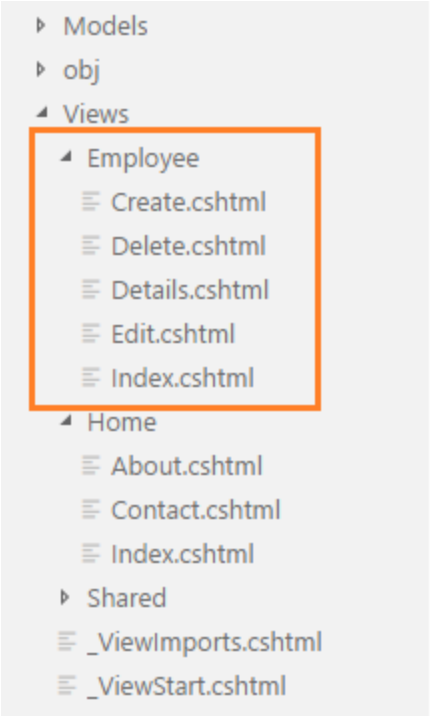
Right-click on the Views folder, and select “New Folder” and name the folder **Employee**.



To add view files, right click on the Employee folder inside the Views folder and select “New File”. Name it **Index.cshtml**. This will create a view file inside Employee folder. Thus, we have created our first view. Similarly add 4

more views in the Views/Employee folder: **Create.cshtml, Delete.cshtml, Details.cshtml,** and **andEdit.cshtml.**

Now our Views folder will look like this:



Since all our Views have been created, we will put some code in View and Controller for performing CRUD operations.

Index View

This view will display all the employee records present in the database. Additionally, we will also provide the action methods Edit, Details, and Delete on each record.

Open **Index.cshtml** and put the following code in it.

```
@model IEnumerable<MVCAdoDemo.Models.Employee>

@{
        ViewData["Title"] = "Index";
    }
<h2>Index</h2>
<p>
    <a asp-action="Create">Create New</a>
</p>
<table class="table">
    <thead>
        <tr>
            <th>
                @Html.DisplayNameFor(model => model.Name)
            </th>
            <th>
                @Html.DisplayNameFor(model => model.Gender)
            </th>
            <th>
                @Html.DisplayNameFor(model =>
model.Department)
            </th>
            <th>
                @Html.DisplayNameFor(model => model.City)
            </th>
            <th></th>
        </tr>
    </thead>
    <tbody>
        @foreach (var item in Model)
{
            <tr>
                <td>
                    @Html.DisplayFor(modelItem =>
item.Name)
                </td>
                <td>
                    @Html.DisplayFor(modelItem =>
item.Gender)
                </td>
```

```
                <td>
                    @Html.DisplayFor(modelItem =>
item.Department)
                </td>
                <td>
                    @Html.DisplayFor(modelItem =>
item.City)
                </td>
                <td>
                    <a asp-action="Edit" asp-route-
id="@item.ID">Edit</a> |
                    <a asp-action="Details" asp-route-
id="@item.ID">Details</a> |
                    <a asp-action="Delete" asp-route-
id="@item.ID">Delete</a>
                </td>
            </tr>
        }
    </tbody>
</table>
```

Open your **EmployeeController.cs** file. You'll see that it is empty. Put the following code into it.

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

namespace MVCAdoDemo.Controllers
{
    public class EmployeeController : Controller
    {
        EmployeeDataAccessLayer objemployee = new
EmployeeDataAccessLayer();

        public IActionResult Index()
        {
```

```
            List<Employee> lstEmployee = new
List<Employee>();
            lstEmployee =
objemployee.GetAllEmployees().ToList();

            return View(lstEmployee);
        }
    }
}
```

To handle database operations, we have created an object of **EmployeeDataAccessLayer** class inside the **EmployeeController** class.

Create View

This view will be used to Add new employee data to the database.

Open **Create.cshtml** and put the following code into it.

```
@model MVCAdoDemo.Models.Employee

@{
    ViewData["Title"] = "Create";
}
<h2>Create</h2>
<h4>Employees</h4>
<hr />
<div class="row">
    <div class="col-md-4">
        <form asp-action="Create">
            <div asp-validation-summary="ModelOnly"
class="text-danger"></div>
            <div class="form-group">
                <label asp-for="Name" class="control-
```

```
label"></label>
                <input asp-for="Name" class="form-control"
/>
                <span asp-validation-for="Name"
class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="Gender" class="control-
label"></label>
                <select asp-for="Gender" class="form-
control">
                    <option value="">-- Select Gender --
</option>
                    <option value="Male">Male</option>
                    <option value="Female">Female</option>
                </select>
                <span asp-validation-for="Gender"
class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="Department" class="control-
label"></label>
                <input asp-for="Department" class="form-
control" />
                <span asp-validation-for="Department"
class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="City" class="control-
label"></label>
                <input asp-for="City" class="form-control"
/>
                <span asp-validation-for="City"
class="text-danger"></span>
            </div>
            <div class="form-group">
                <input type="submit" value="Create"
class="btn btn-default" />
            </div>
        </form>
    </div>
</div>
```

```
<div>
    <a asp-action="Index">Back to List</a>
</div>
@section Scripts {
    @{await
Html.RenderPartialAsync("_ValidationScriptsPartial");}
}
```

To handle the business logic of **create**, open **EmployeeController.cs** and put the following code into it.

```
[HttpGet]
public IActionResult Create()
{
    return View();
}

[HttpPost]
[ValidateAntiForgeryToken]
public IActionResult Create([Bind] Employee employee)
{
    if (ModelState.IsValid)
    {
        objemployee.AddEmployee(employee);
        return RedirectToAction("Index");
    }
    return View(employee);
}
```

The [Bind] attribute is used with parameter “employee” to protect against over-posting. To learn more about over-posting, visit this link.

Edit View

This view will enable us to edit an existing employee's data.

Open **Edit.cshtml** and put the following code into it.

```
@model MVCAdoDemo.Models.Employee

@{
    ViewData["Title"] = "Edit";
}
<h2>Edit</h2>
<h4>Employees</h4>
<hr />
<div class="row">
    <div class="col-md-4">
        <form asp-action="Edit">
            <div asp-validation-summary="ModelOnly"
class="text-danger"></div>
            <input type="hidden" asp-for="ID" />
            <div class="form-group">
                <label asp-for="Name" class="control-
label"></label>
                <input asp-for="Name" class="form-control"
/>
                <span asp-validation-for="Name"
class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="Gender" class="control-
label"></label>
                <select asp-for="Gender" class="form-
control">
                    <option value="">-- Select Gender --
</option>
                    <option value="Male">Male</option>
                    <option value="Female">Female</option>
                </select>
                <span asp-validation-for="Gender"
class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="Department" class="control-
```

```
label"></label>
                <input asp-for="Department" class="form-
control" />
                <span asp-validation-for="Department"
class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="City" class="control-
label"></label>
                <input asp-for="City" class="form-control"
/>
                <span asp-validation-for="City"
class="text-danger"></span>
            </div>
            <div class="form-group">
                <input type="submit" value="Save"
class="btn btn-default" />
            </div>
        </form>
    </div>
</div>
<div>
    <a asp-action="Index">Back to List</a>
</div>
@section Scripts {
    @{await
Html.RenderPartialAsync("_ValidationScriptsPartial");}
}
```

To handle the business logic of the **Edit** view, open **EmployeeController.cs** and add the following code to it.

```
[HttpGet]
public IActionResult Edit(int? id)
{
    if (id == null)
    {
        return NotFound();
    }
```

```
    Employee employee = objemployee.GetEmployeeData(id);

    if (employee == null)
    {
        return NotFound();
    }
    return View(employee);
}

[HttpPost]
[ValidateAntiForgeryToken]
public IActionResult Edit(int id, [Bind]Employee employee)
{
    if (id != employee.ID)
    {
        return NotFound();
    }
    if (ModelState.IsValid)
    {
        objemployee.UpdateEmployee(employee);
        return RedirectToAction("Index");
    }
    return View(employee);
}
```

You'll see that we have two Edit action methods: one for HttpGet and another for HttpPost.The HttpGet Edit action method fetches the employee data and populates the fields of edit view. Once the user clicks on the Save button after editing the record, a Post request will be generated which is handled by the HttpPost Edit action method.

Details View

This view will display the details of a particular employee.

Open **Details.cshtml** and put the following code into it.

```
@model MVCAdoDemo.Models.Employee

@{
    ViewData["Title"] = "Details";
}
<h2>Details</h2>
<div>
    <h4>Employees</h4>
    <hr />
    <dl class="dl-horizontal">
        <dt>
            @Html.DisplayNameFor(model => model.Name)
        </dt>
        <dd>
            @Html.DisplayFor(model => model.Name)
        </dd>
        <dt>
            @Html.DisplayNameFor(model => model.Gender)
        </dt>
        <dd>
            @Html.DisplayFor(model => model.Gender)
        </dd>
        <dt>
            @Html.DisplayNameFor(model => model.Department)
        </dt>
        <dd>
            @Html.DisplayFor(model => model.Department)
        </dd>
        <dt>
            @Html.DisplayNameFor(model => model.City)
        </dt>
        <dd>
            @Html.DisplayFor(model => model.City)
        </dd>
    </dl>
</div>
<div>
    <a asp-action="Edit" asp-route-id="@Model.ID">Edit</a>
|
```

```
    <a asp-action="Index">Back to List</a>
</div>
```

To handle the business logic of the **Details** view, open **EmployeeController.cs** and add the following code to it.

```
[HttpGet]
public IActionResult Details(int? id)
{
    if (id == null)
    {
        return NotFound();
    }
    Employee employee = objemployee.GetEmployeeData(id);

    if (employee == null)
    {
        return NotFound();
    }
    return View(employee);
}
```

Delete View

This view will help us to remove employee data.

Open **Delete.cshtml** and put the following code into it.

```
@model MVCAdoDemo.Models.Employee

@{
    ViewData["Title"] = "Delete";
}
<h2>Delete</h2>
<h3>Are you sure you want to delete this?</h3>
<div>
    <h4>Employees</h4>
```

```
    <hr />
    <dl class="dl-horizontal">
        <dt>
            @Html.DisplayNameFor(model => model.Name)
        </dt>
        <dd>
            @Html.DisplayFor(model => model.Name)
        </dd>
        <dt>
            @Html.DisplayNameFor(model => model.Gender)
        </dt>
        <dd>
            @Html.DisplayFor(model => model.Gender)
        </dd>
        <dt>
            @Html.DisplayNameFor(model => model.Department)
        </dt>
        <dd>
            @Html.DisplayFor(model => model.Department)
        </dd>
        <dt>
            @Html.DisplayNameFor(model => model.City)
        </dt>
        <dd>
            @Html.DisplayFor(model => model.City)
        </dd>
    </dl>

    <form asp-action="Delete">
        <input type="hidden" asp-for="ID" />
        <input type="submit" value="Delete" class="btn btn-
default" /> |
        <a asp-action="Index">Back to List</a>
    </form>
</div>
```

To handle the business logic of the **Delete** view, open **EmployeeController.cs** and add the following code to it.

```
[HttpGet]
public IActionResult Delete(int? id)
{
    if (id == null)
    {
        return NotFound();
    }
    Employee employee = objemployee.GetEmployeeData(id);

    if (employee == null)
    {
        return NotFound();
    }
    return View(employee);
}

[HttpPost, ActionName("Delete")]
[ValidateAntiForgeryToken]
public IActionResult DeleteConfirmed(int? id)
{
    objemployee.DeleteEmployee(id);
    return RedirectToAction("Index");
}
```

To complete the Delete operation, we need two Delete methods that accept the same parameter (Employee Id). But two methods with same name and method signature will create a compile time error. And if we rename the Delete method, then routing won't be able to find it, as asp.net maps URL segments to action methods by name.

So, to resolve this issue, we add the ActionName("Delete") attribute to the DeleteConfirmed method. This attribute performs mapping for the routing system so that a URL that

includes /Delete/ for a POST request will find the DeleteConfirmed method.

When we click on the Delete link on the Index page, it will send a Get request and return a View of the employee using the HttpGet Delete method. When we click on the Delete button on this view, it will send a Post request to delete the record which is handled by the HttpPost DeleteConfirmed method.

Performing a delete operation in response to a Get request (or, for that matter, performing an edit operation, create operation, or any other operation that changes data) opens up a security hole. Therefore, we have two separate methods.

Configure the route URL

Before launching the application, we will configure the route URLs. Open the **Startup.cs** file to set the format for routing. Scroll down to the **app.UseMvc** method, where you can set the route URL.

Make sure that your route URL is set like this:

```
app.UseMvc(routes =>
 {
     routes.MapRoute(
```

```
        name: "default",
        template:
"{controller=Home}/{action=Index}/{id?}");
 });
```

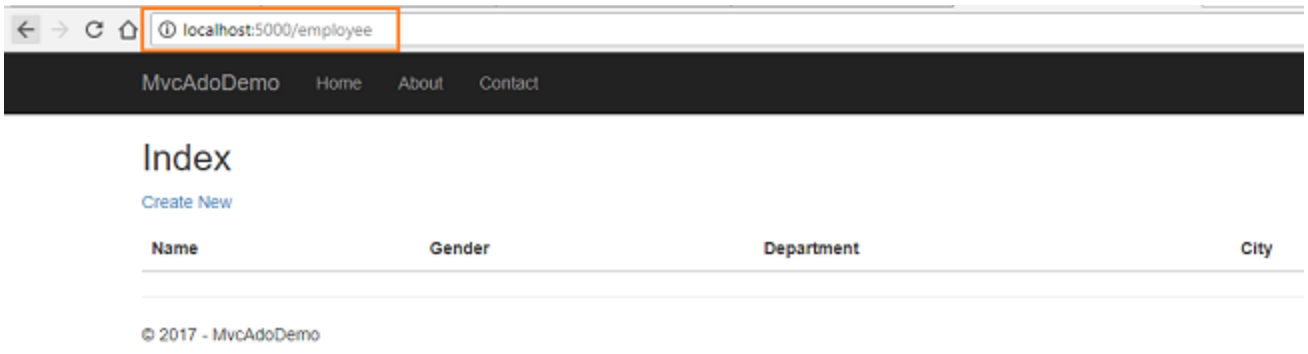
This URL pattern sets HomeController as the default controller and Index method as the default action method (whereas the Id parameter is optional). Default and optional route parameters need not be present in the URL path for a match.

If we do not append any controller name in the URL, then it will take HomeController as the default controller and the Index method of HomeController as the default action method. Similarly, if we append only the Controller name in the URL, it will navigate to the Index action method of that controller.

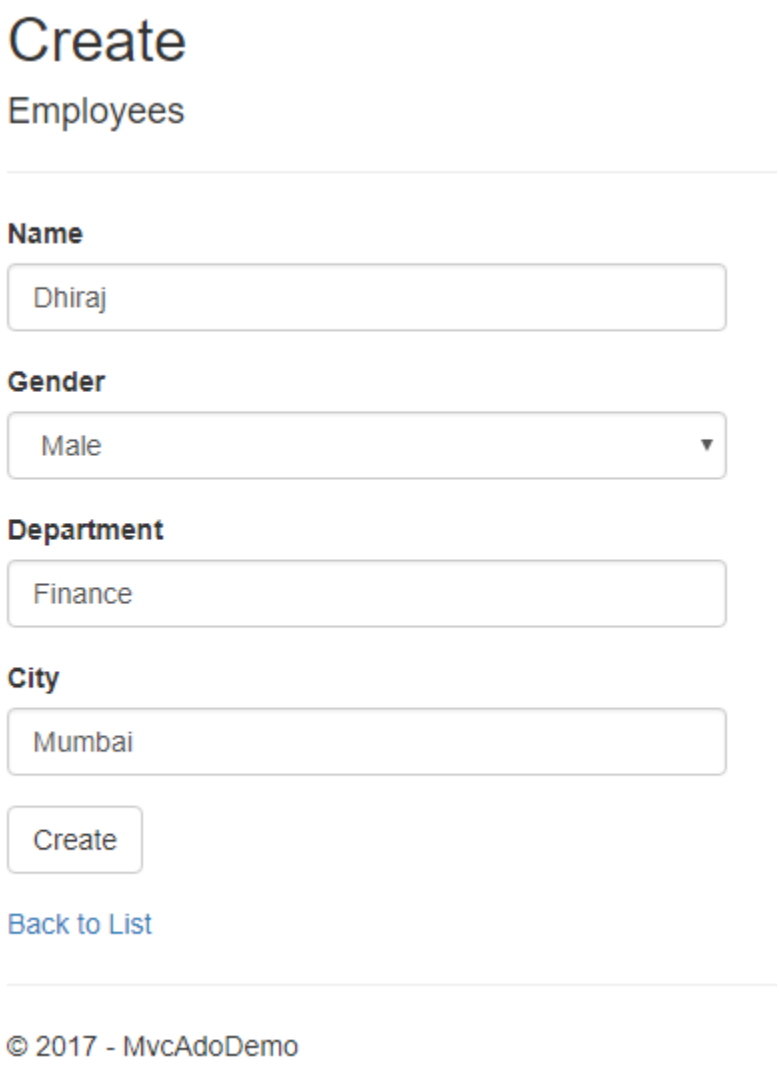
Execution Demo

Now press F5 to launch the application and navigate to the Employee controller by appending **/Employee** to the URL.

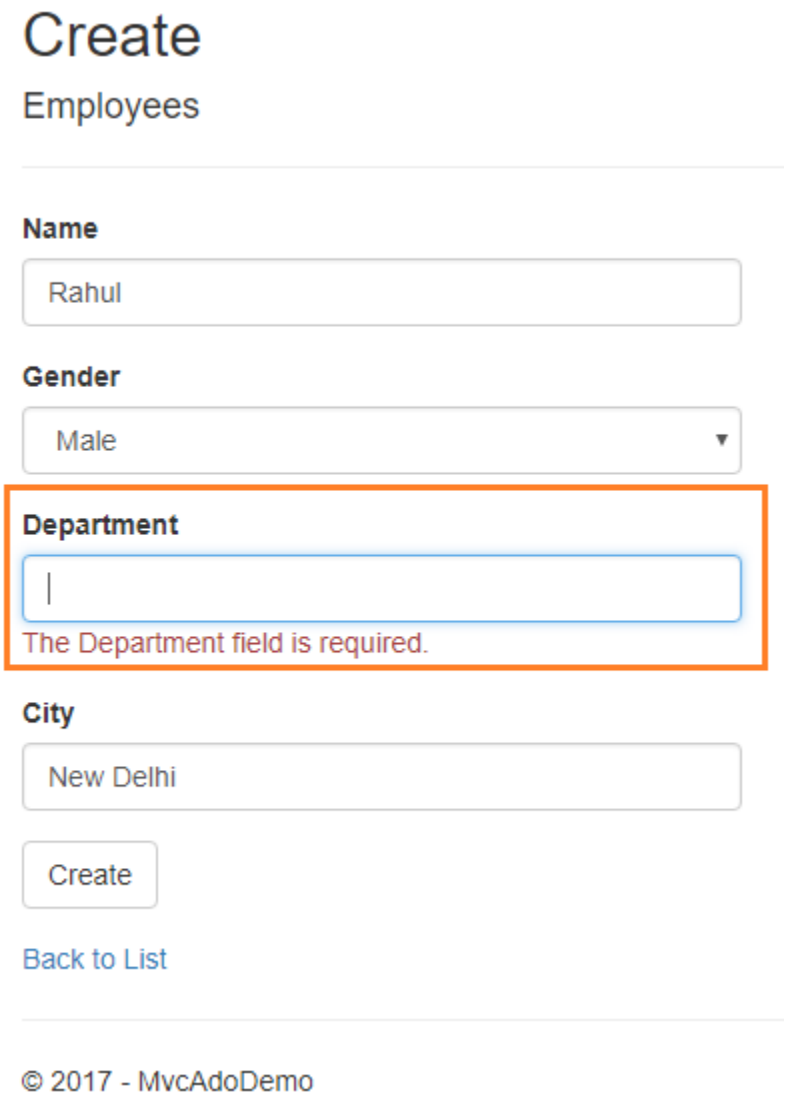
You can see the page as shown below.



Click on **CreateNew** to navigate to the **Create** view. Add a new Employee record as shown in the image below.



If we miss the data in any field while creating the employee record, we will get a required field validation error message.



After inserting the data in all the fields, click on the “Create” button. The new employee record will be created and you will be redirected to the Index view, which displays records of all the employees. Here, we can also see the action methods Edit, Details, and Delete.

Index

Create New

Name	Gender	Department	City	
Dhiraj	Male	Finance	Mumbai	Edit | Details | Delete
Rahul	Male	HR	New Delhi	Edit | Details | Delete
Swati	Female	Accounts	Chennai	Edit | Details | Delete

© 2017 - MvcAdoDemo

If we want to edit an existing employee record, then click the Edit action link. It will open the Edit View as below where we can change the employee's data.

Edit

Employees

Name

Dhiraj

Gender

Male

Department

Finance

City

New Delhi

Save

Back to List

© 2017 - MvcAdoDemo

Here we have changed the City of employee Dhiraj from Mumbai to New Delhi. Click on “Save” to return to the Index view to see the updated changes as highlighted in the image below.

Index

Create New

Name	Gender	Department	City	
Dhiraj	Male	Finance	New Delhi	Edit | Details | Delete
Rahul	Male	HR	New Delhi	Edit | Details | Delete
Swati	Female	Accounts	Chennai	Edit | Details | Delete

© 2017 - MvcAdoDemo

If we miss any fields while editing the employee's record, then the Edit view will also throw the required field validation error message.

Edit

Employees

Name

Dhiraj

Gender

Male

Department

The Department field is required.

City

New Delhi

Save

Back to List

© 2017 - MvcAdoDemo

If you want to see the details of any Employee, then click on the Details action link, which will open the Details view, as shown in the image below.

Details

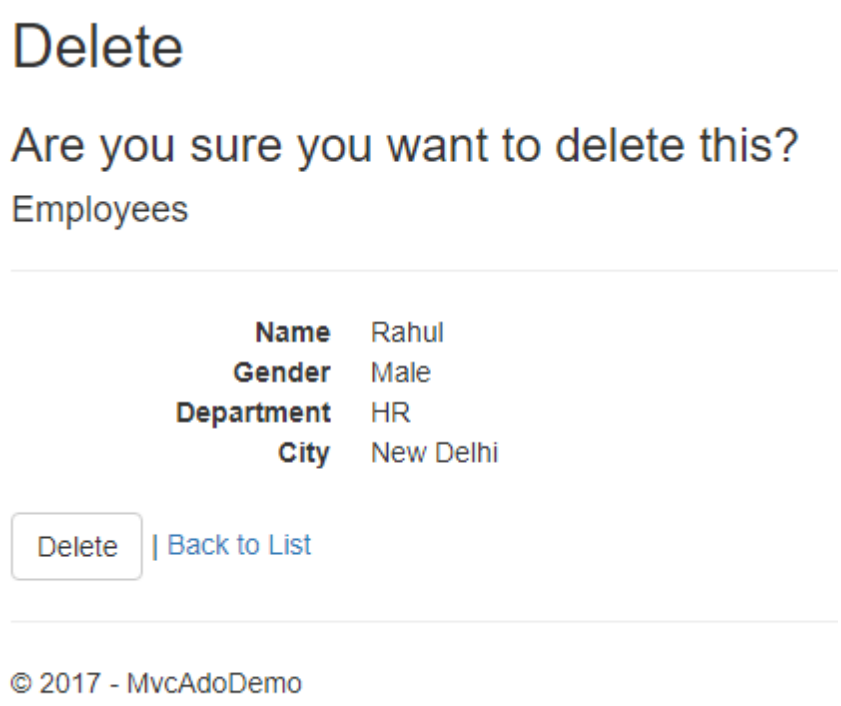
Employees

Name	Swati
Gender	Female
Department	Accounts
City	Chennai

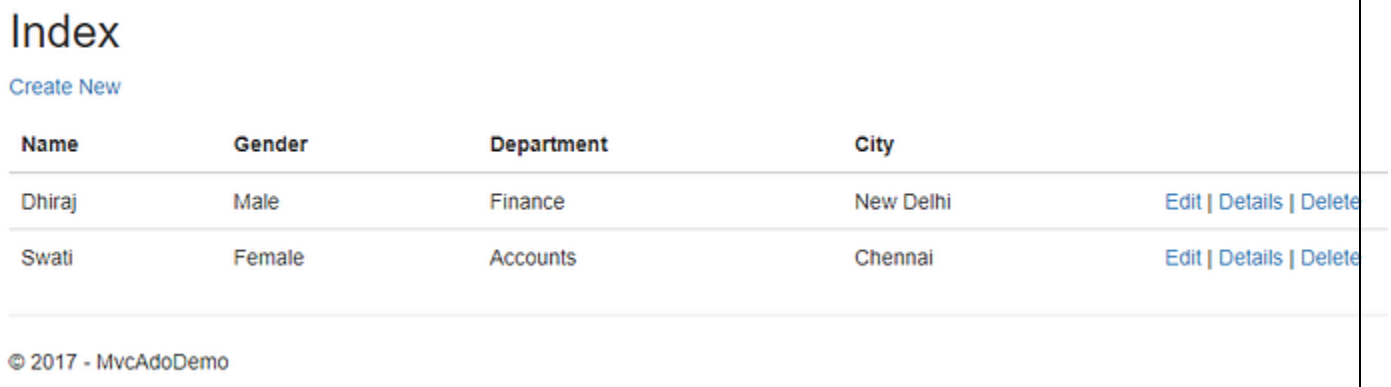
Edit | Back to List

© 2017 - MvcAdoDemo

Click on “Back to List” to go back to Index view. Now, we will perform a Delete operation on an employee named Rahul. Click on the Delete action link which will open the Delete view asking for a confirmation to delete.



Once we click on the Delete button, it will send an HttpPost request to delete the employee's record, and we will be redirected to the Index view. Here, we can see that the employee with the name Rahul has been removed from our record.



Conclusion

We have learned about creating a sample MVC web application using ASP.Net Core 2.0, ADO.NET, and a SQL server with the help of Visual Studio Code.

Download the source code from GitHub and play around to get a better understanding.

You can check my other articles on ASP .NET Core here

Preparing for interviews? Read my article on C# Coding Questions For Technical Interviews.

See Also

- CRUD Operation With ASP.NET Core MVC Using ADO.NET and Visual Studio 2017
- CRUD Operation With ASP.NET Core MVC Using Visual Studio Code and EF
- ASP.NET Core — CRUD With React.js And Entity Framework Core
- CRUD Operations With ASP.NET Core Using Angular 5 and ADO.NET
- ASP.NET Core — Getting Started With Blazor

- Cookie Authentication With ASP.NET Core 2.0