Title

MSBuild Compilation & Automatic Deployment using Visual Studio for ASP.NET MVC using C#

Problem Statement

MVC stands for Model View Controller, its a framework or a design pattern used for the development of Web Application, nowadays its very high in demand for the development of an application with loosely coupled code.

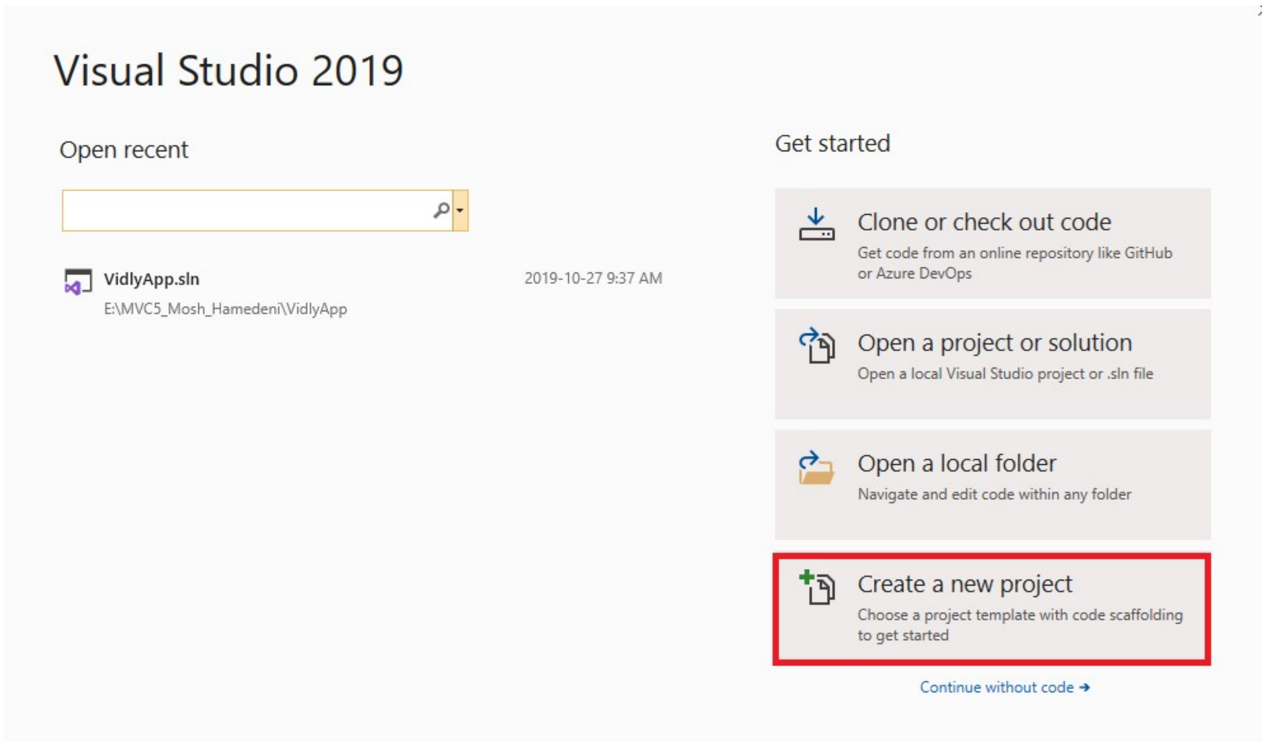
In MVC we have a separation of code that can be developed, tested & maintained easily. In MVC, the application can be separated using three different folders such as Models, Views, & Controller, where the Models folder contains all the Models or data, Controllers folder have all the controllers and Views folder contains all the views or .cshtml files related to the particular application.

Whenever a user requests a webpage, the controller is responsible for handling the HTTP request, the controller interacts with the appropriate model by interacting with the database by fetching the data and stores in the model and return back to the controller. The Controller in turns returns the data to the view for displaying the required data to the user.

In this article we will see the autogenerated folder structure of MVC, the autogenerated folder structure is created whenever we create a new MVC application. Here I am using visual studio 2019 community edition, you can use any version of your choice, in my case I am going with visual studio 2019.

In order to create MVC 5 application follow the below steps:

Step 1: Open visual studio 2019 and click on Create a new project on the bottom right.


Visual Studio 2019

Open recent

VidlyApp.sln
E:\MVC5_Mosh_Hamedeni\VidlyApp
2019-10-27 9:37 AM

Get started

Clone or check out code
Get code from an online repository like GitHub or Azure DevOps

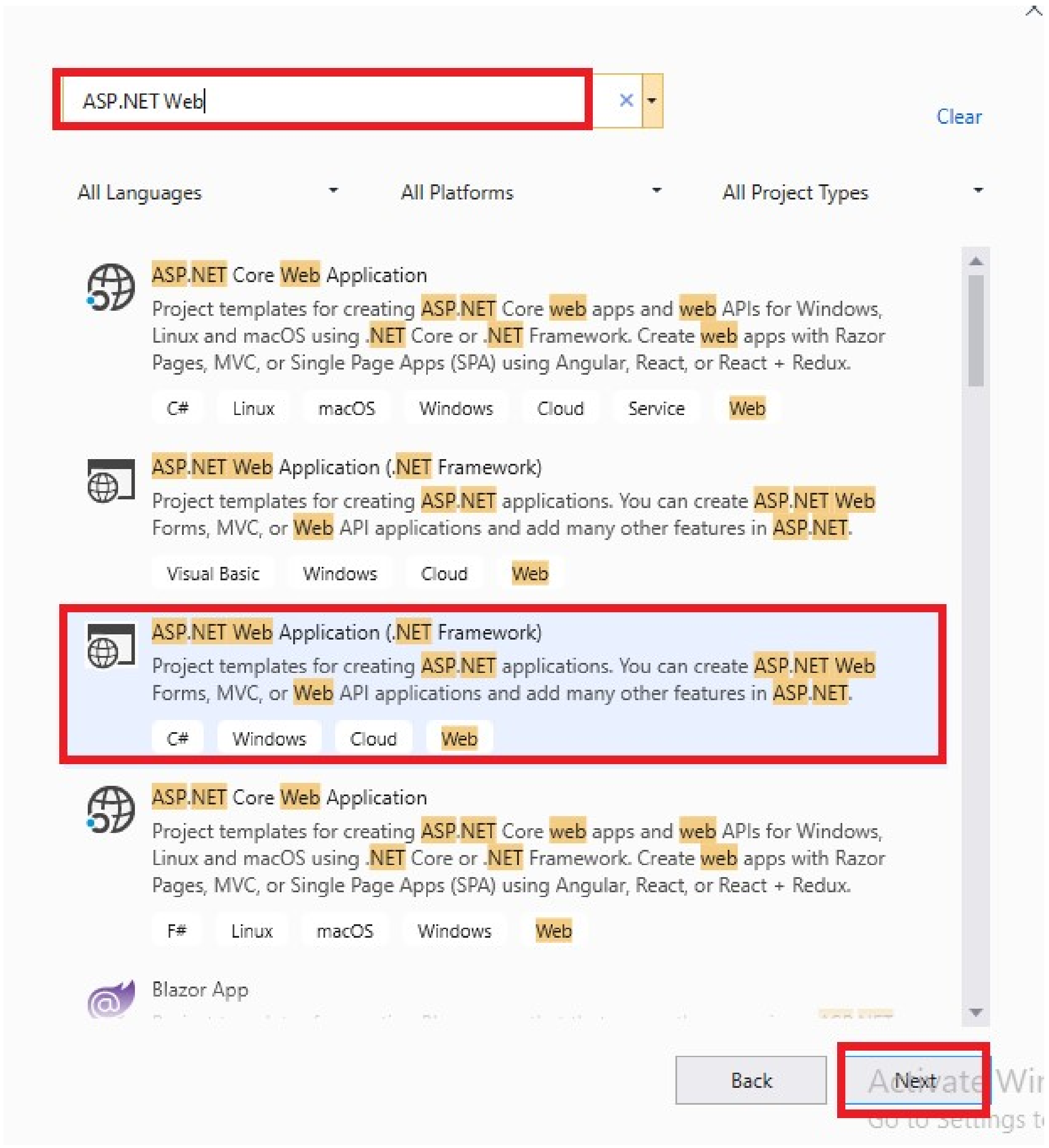
Open a project or solution
Open a local Visual Studio project or .sln file

Open a local folder
Navigate and edit code within any folder

Create a new project
Choose a project template with code scaffolding to get started

Continue without code →

Step 2: Search ASP.NET Web Application in the above-provided textbox and click on the template and click on the Next button as shown below.



ASP.NET Web

Clear

All Languages | All Platforms | All Project Types

ASP.NET Core Web Application
Project templates for creating ASP.NET Core web apps and web APIs for Windows, Linux and macOS using .NET Core or .NET Framework. Create web apps with Razor Pages, MVC, or Single Page Apps (SPA) using Angular, React, or React + Redux.
C# | Linux | macOS | Windows | Cloud | Service | Web

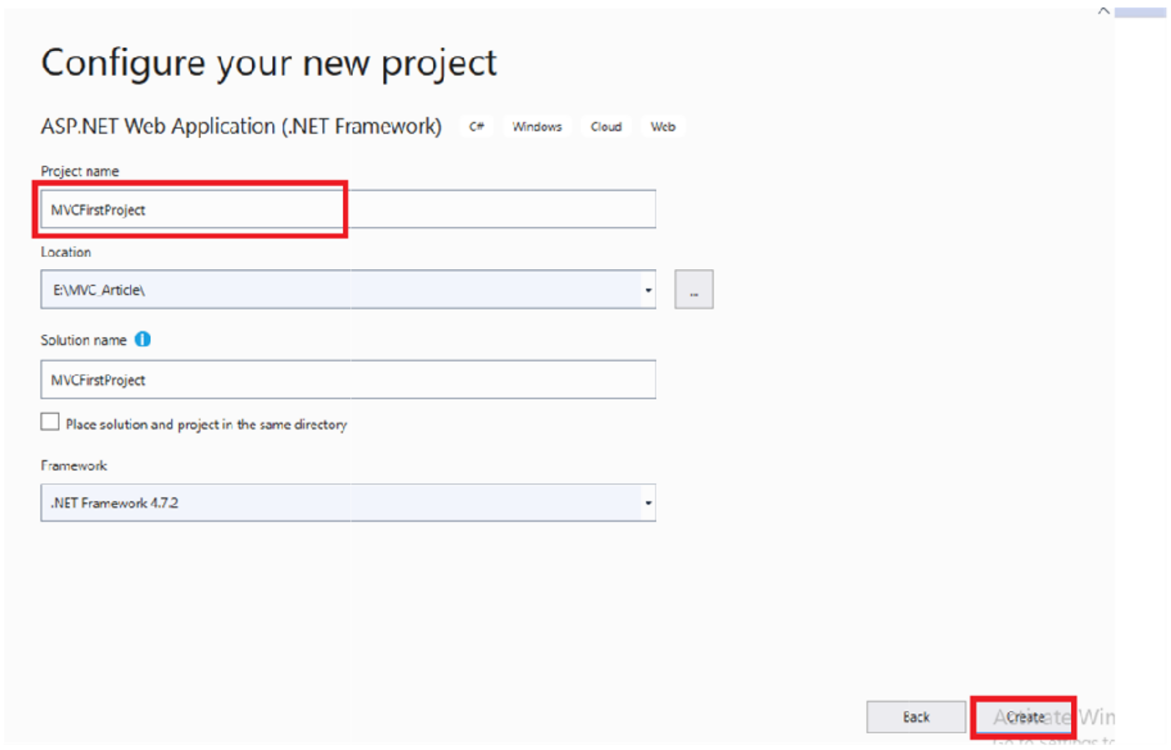
ASP.NET Web Application (.NET Framework)
Project templates for creating ASP.NET applications. You can create ASP.NET Web Forms, MVC, or Web API applications and add many other features in ASP.NET.
Visual Basic | Windows | Cloud | Web

ASP.NET Web Application (.NET Framework)
Project templates for creating ASP.NET applications. You can create ASP.NET Web Forms, MVC, or Web API applications and add many other features in ASP.NET.
C# | Windows | Cloud | Web

ASP.NET Core Web Application
Project templates for creating ASP.NET Core web apps and web APIs for Windows, Linux and macOS using .NET Core or .NET Framework. Create web apps with Razor Pages, MVC, or Single Page Apps (SPA) using Angular, React, or React + Redux.
F# | Linux | macOS | Windows | Web

Blazor App

Step 3: Provide the project name say MVCFirstProject and click on Create button.


Configure your new project

ASP.NET Web Application (.NET Framework)

Project name: MVCFirstProject

Location: E:\MVC Article\

Solution name: MVCFirstProject

☐ Place solution and project in the same directory

Framework: .NET Framework 4.7.2

Step 4: Select the MVC template and click on the Create button, with this a new MVC project is created. Let's see the MVC folder structure which is created when we create the above MVC application. Here we have different folders like

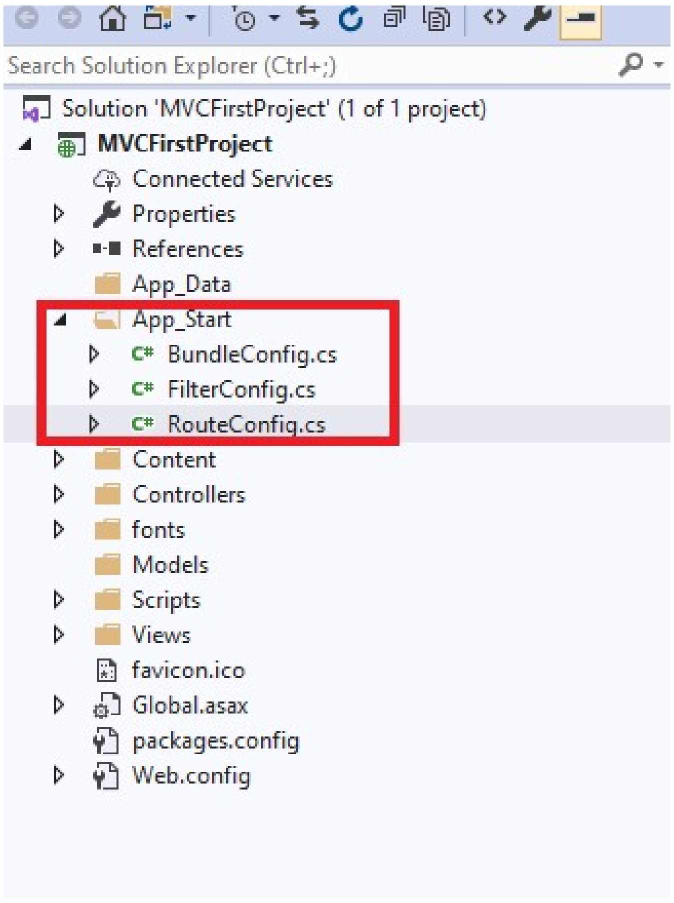
- App_Data
- App_Start
- Content
- Controllers
- fonts
- Models
- Scripts
- Views etc.

App_Data: The App_Data contains all the database related files.

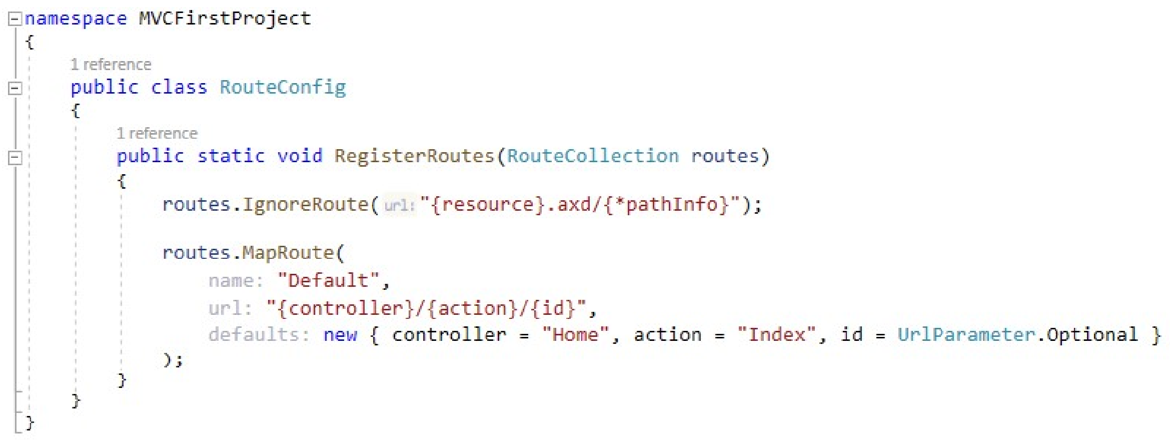
App_Start: This folder contains all the configuration files for the application.

For Example:

BundleConfig.cs
FilterConfig.cs &
RouteConfig.cs



The RouteConfig.cs contains the routing configuration defined by the RouteConfig class and RegisterRoutes static method.



```
namespace MVCFirstProject
{
    1 reference
    public class RouteConfig
    {
        1 reference
        public static void RegisterRoutes(RouteCollection routes)
        {
            routes.IgnoreRoute(url: "{resource}.axd/{*pathInfo}");

            routes.MapRoute(
                name: "Default",
                url: "{controller}/{action}/{id}",
                defaults: new { controller = "Home", action = "Index", id = UrlParameter.Optional }
            );
        }
    }
}
```

In the above Url we have a pattern with {Controller}/{action}/{id}. And default Controller is the Home and Action method is Index.

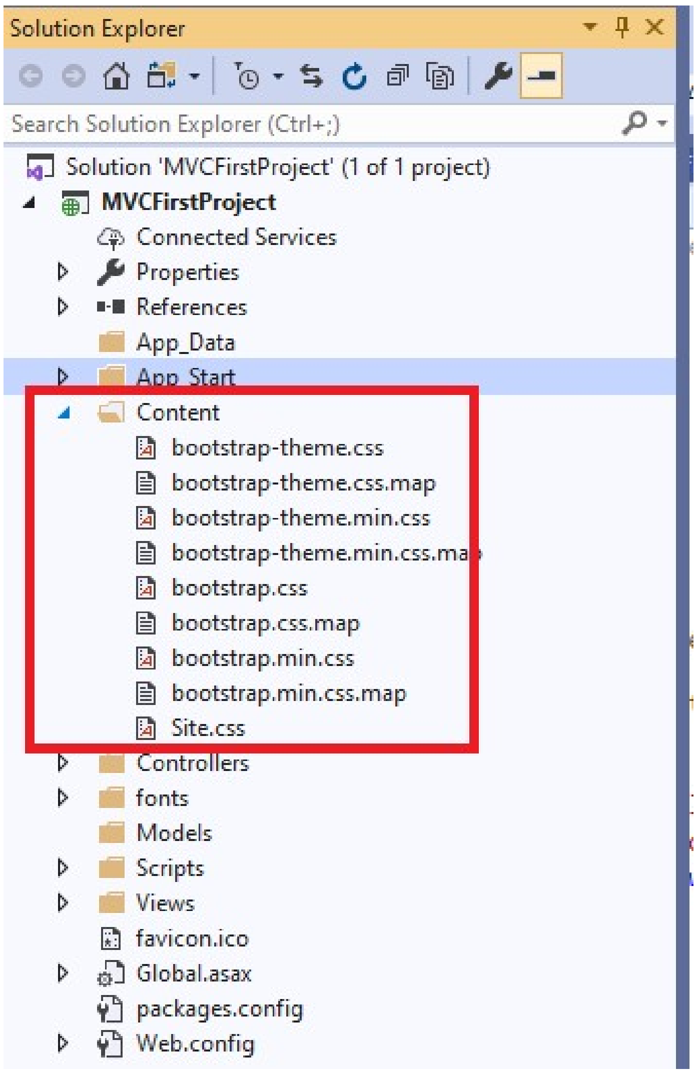
Whenever user request for a webpage by providing the Url say, localhost:40366/Employee/EmployeeInfo/1

The request will be handled by the EmployeeController and its EmployeeInfo action method by passing the value 1 to the EmployeeInfo method's parameter,

Suppose in a controller's EmployeeInfo action method, if we provide a parametername which is not matching with the Url pattern, then MVC is unable to handle the request by the Employee Controller as there is a mismatch of a parameter name, which gives an error message.

If we provide a Url say, localhost:40366/Employee. Then with this Url, Employee Controller handles the request and the Index method is executed as provided action is Index in defaults field. If we provide a Url, localhost:40366/, without providing the controller name or action method name then by default HomeController and Index action method is called which is defined in defaults field which is defined in RegisterRoutes method.

Content: This folder contains all the CSS related files such as bootstrap.css, bootstrap.min.css, etc.

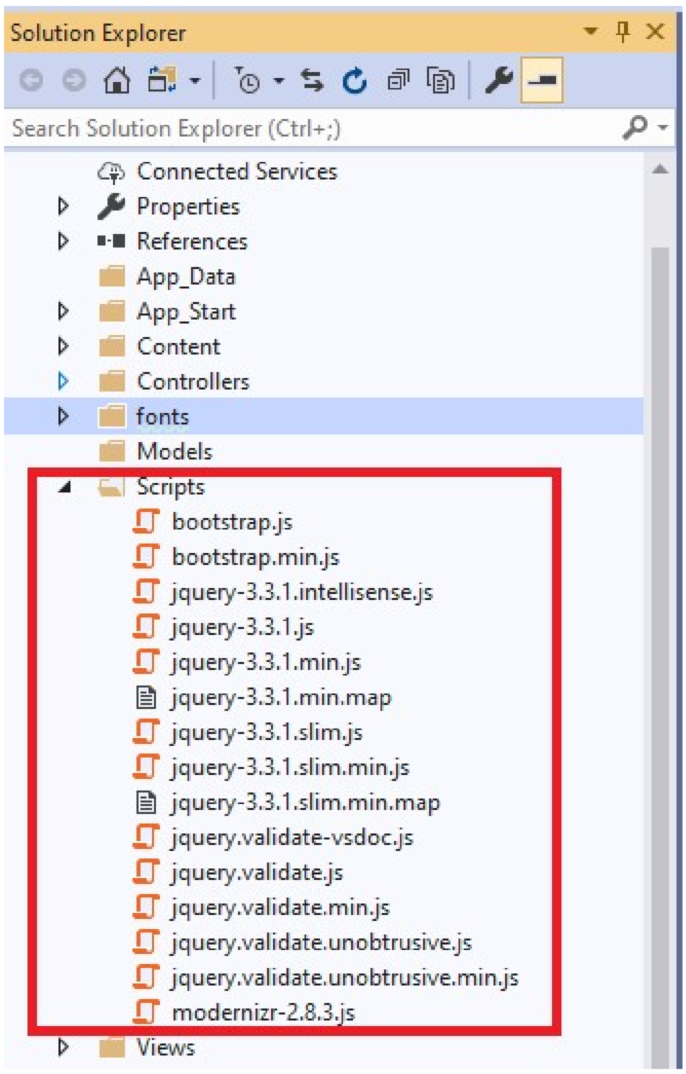


Controllers: This folder contains all the controller for the particular application, by default it will come with HomeController, each controller should be postfix with name Controller.

Fonts: This folder is used for storing images, icons, etc.

Models: Models folder contains all the Models related data for the application, a model is nothing but the properties for storing the data which can be used in the application for assigning the data and binding it to the view. Based on the functionality or requirement we create multiple Models.

Scripts: Scripts folder contains all the javascript files which are required for the particular application, by default MVC application provides various javascript files such as bootstrap.js, jquery-3.3.1.js, jquery-3.3.1.min.js, etc.



- Connected Services
- Properties
- References
- App_Data
- App_Start
- Content
- Controllers
- fonts
- Models
- Scripts
  - bootstrap.js
  - bootstrap.min.js
  - jquery-3.3.1.intellisense.js
  - jquery-3.3.1.js
  - jquery-3.3.1.min.js
  - jquery-3.3.1.min.map
  - jquery-3.3.1.slim.js
  - jquery-3.3.1.slim.min.js
  - jquery-3.3.1.slim.min.map
  - jquery.validate-vsdoc.js
  - jquery.validate.js
  - jquery.validate.min.js
  - jquery.validate.unobtrusive.js
  - jquery.validate.unobtrusive.min.js
  - modernizr-2.8.3.js
- Views

Views: This folder contains all the views for the application, whenever we create a view, it creates a separate folder with the controller's name except the postfix name controller. View folder will have all the views depending on the Controller's name.

For Example: If we have 3 different Controllers say, AccountController, HomeController, LoginController then the views will have 3 different folder's with names as Account, Home and Login, etc.

Package.config: This is the configuration file that contains all the dependency files or references of the application. Whenever we add a new reference from NuGet Package Manager, one of the references will be added to the package.config file.

Web.config: This is the configuration file for the application which is used for defining the connectionStrings, AppSettings, etc.

In order to create a sample MVC project, we will add a model to the Models folder and add few properties like below.

Right-click on Models folder--> Add --> select class and give the classname as Employee.cs and click on Add. With this, a new Employee Model is created.

Open Employee.cs and add few properties like below

```csharp
namespace MVCFirstProject.Models
{
   public class Employee
   {
      public int EmployeeId { get; set; }
      public string EmployeeName { get; set; }
      public string EmployeeLocation { get; set; }
   }
}
```

Similarly to add a Controller called EmployeeController, right-click on Controllers folder, click on Add and select the Controller, In Add New Scaffolded Item Window box, select MVC5 Controller - Empty and click on Add, give the Controller name as EmployeeController and click on Add, with this a new controller called EmployeeController is added to Controllers folder.

Whenever we create a new Controller in the Controller's folder, a new folder will be created in views folder which has the same name as controller name except for the postfix Controller. In Employee Controller's EmployeeInfo action method, create the instance of the Employee model by assigning the value to the properties as shown below.

```
namespace MVCFirstProject.Controllers
{
    public class EmployeeController : Controller
    {
        // GET: Employee
        public ActionResult EmployeeInfo()
        {
            Employee employee = new Employee()
            {
                EmployeeId = 1001,
                EmployeeName = "Khaja Moiz",
                EmployeeLocation = "Hyderabad"
            };


            return View(employee);
        }
    }
}
```

Right-click on Employee folder in Views folder, click on Add and select Add View. In Add View window box, give the view name as the Action method name that is EmployeeInfo, template as Empty(without model) and click on Add. With this EmployeeInfo.cshtml view is added to the Employee View folder.

Open the EmployeeInfo.cshtml file and paste the below code.

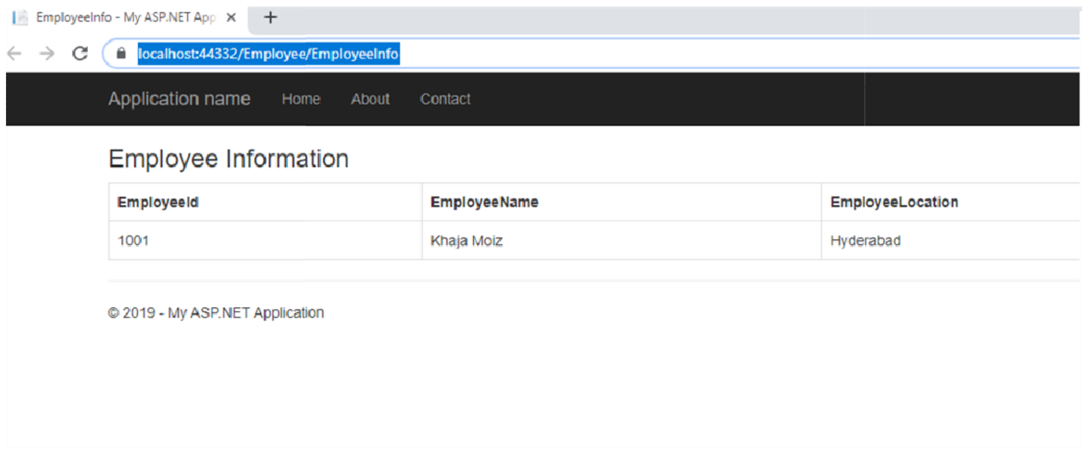
```
@model MVCFirstProject.Models.Employee

@{
    ViewBag.Title = "EmployeeInfo";
    Layout = "~/Views/Shared/_Layout.cshtml";
}

<style>
```

```
    table {
        font-family: arial, sans-serif;
        border-collapse: collapse;
        width: 100%;
    }

    td, th {
        border: 1px solid #dddddd;
        text-align: left;
        padding: 8px;
    }
</style>

<h3>Employee Information</h3>

<table>
    <tr>
        <th>EmployeeId</th>
        <th>EmployeeName</th>
        <th>EmployeeLocation</th>
    </tr>

    <tr>
        <td>@Model.EmployeeId</td>
        <td>@Model.EmployeeName</td>
        <td>@Model.EmployeeLocation</td>
    </tr>
</table>
```

Now build and run the solution by navigating to https://localhost:44332/Employee/EmployeeInfo. With the above request, EmployeeInfo of EmployeeController handles the request, In EmployeeInfo Action method we are creating the instance of the Employee model by assigning the hardcoded value and returning the Employee object through the view.

In EmployeeInfo.cshtml, the returned view will be binded to the HTML table and displayed to the user.

The output is shown below.


EmployeeInfo - My ASP.NET App
localhost:44332/Employee/EmployeeInfo
Application name
Home
About
Contact

Employee Information

EmployeeId	EmployeeName	EmployeeLocation
1001	Khaja Moiz	Hyderabad

© 2019 - My ASP.NET Application

Assignment

1. Open Command Prompt Window
2. Go to the MVC Project Root folder by using cd command
3. Run the following command:
    a. MSBuild.exe MVCFirstProject.csproj –t:rebuild –v:n
4. In Solution Explorer, click the project node MVCFirstProject.
5. Right-click the project node again, then click Edit MVCFirstProject.csproj.
6. Add a target and a task.
    ```
    <Target Name="HelloWorld">
    <Message Text="Hello"></Message> <Message Text="World"></Message>
    </Target>
    ```
7. Run the following command:
    a. MSBuild.exe MVCFirstProject.csproj –t:HelloWorld –v:n
8. Examine the output in the Command Prompt Window. You should see the two lines "Hello" and "World".
9. Add another target and a task.

```
<Target Name="BuildProjects" Condition=" '$(BuildingInTeamBuild)'! ='true'>
   <MSBuild Projects="@(ProjectsToBuild)"
         Properties="OutDir=$(OutputRoot);
                  Configuration=$(Configuration);
                  DeployOnBuild=true;
                  DeployTarget=Package"
         Targets="Build" />
    </Target>
```

10. Run the following command:

    a. MSBuild.exe MVCFirstProject.csproj –t:rebuild –v:n

11. Add another target and a task.

```
<Target Name="PublishWebPackages"
Outputs="%(PublishPackages.Identity)">
  <PropertyGroup>
   <_WhatIfSwitch>/Y</_WhatIfSwitch>
   <_WhatIfSwitch Condition=" '$(_WhatIf)'=='true' ">/T</_WhatIfSwitch>
   <_Cmd>
    %(PublishPackages.FullPath) $(_WhatifSwitch)
/M:$(MSDeployComputerName)
    /U:$(MSDeployUsername) /P:$(Password) /A:$(MSDeployAuth)
    %(PublishPackages.AdditionalMSDeployParameters)
   </_Cmd>
  </PropertyGroup>
  <Exec Command="$(_Cmd)"/>
</Target>
```

12. Run the following command:
    a. MSBuild.exe MVCFirstProject.csproj –t: PublishWebPackages –v:n