## **Creating the First ASP.NET MVC Application using Visual Studio**

In this article, I am going to discuss how to create the first **ASP.NET MVC Application** step by step from scratch using Visual Studio 2015. You can use any version as per your choice but the step will remain the same. Please read our previous article before proceeding to this article where we gave a brief **introduction to ASP.NET MVC** Framework.

##### **Creating the First ASP.NET MVC Application:**

First of all, open the Visual Studio and then click on the **New Project** link which appeared on the **Startup** page. An alternative approach to creating an application is to select the **File =>** **New =>Project** option as shown in the below image.

Creating First ASP.NET MVC Application using Visual Studio

From the **New Project** dialog window, from the left pane select the **Web** option under the **Visual C#** which is under the “**Installed – Templates**” section. From the middle pane select **ASP.NET Web Application**. Name your project as **FirstMVCApplication** (you can give any name to your application). Change the location where you want to create the application by clicking on the **Browse** button. Finally, click on the **OK** button as shown in the image below**.**

**Creating the first ASP.NET MVC Application**

Once you click on the **OK** button, then a new dialog window will open with the name **New ASP.NET Web Application** for selecting the **Project Templates**. From this window select the **MVC** project template. Then Change the Authentication type to **No Authentication.** Finally, click on the **OK** button as shown in the below image.

Selecting MVC Template

To change the authentication, you just need to click on the **Change Authentication** button. Then it will open the following popup. From that popup select the appropriate authentication mode. As we don’t want to use any authentication for this demo, so select the **No Authentication** radio button and click on the **OK** button as shown below.

Selecting Authentication For ASP.NET MVC Application

Once you click on the OK button, it will take some time to create the project for us with the following default folder structure.

ASP.NET MVC Application Folder Structure

##### **Running the ASP.NET MVC Application:**

If you want to run the project **with debug** mode then just press **F5**. On the other hand, if you want to run the application **without debugging** then just Press **Ctrl + F5**. Once you run the application, it will open the following page in the browser.

Running the ASP.NET MVC Application

The ASP.NET MVC 5 framework includes the necessary JavaScript and CSS files that are required for Bootstrap by default to create a responsive web page. The responsive web page means the looks and feels of the web page will be changed based on the screen size of the devices on which we are running the application. For example, if you run the application on a mobile device, then the top menu bar will be changed as shown in the below image.

Responsive Web Design Support by Default

We have done creating our first ASP.NET MVC application from scratch using Visual Studio 2015.

## **ASP.NET MVC File and Folder Structure**

In this article, I am going to discuss the auto-generated **ASP.NET MVC File and File Structure** when we create a new ASP.NET MVC application. Please read our previous article before proceeding to this article where we discussed [**How to Create ASP.NET MVC 5 Application**](https://dotnettutorials.net/lesson/asp-dot-net-mvc-application/) step by step from scratch.

Nowadays, the ASP.NET MVC framework becomes more popular among developers because of the separation of concerns (codes) and folder structure. As a developer, it is very important for you to understand the need and use of each file and folder of an ASP.NET MVC application. When we create an ASP.NET MVC 5 application, Visual Studio by default creates the following files and for our application.

ASP.NET MVC Folder and File structure

Let us discuss the need and use of each File and Folder in detail one by one.

##### **App\_Data:**

The App\_Data folder of an ASP.NET MVC application contains application-related data files like .mdf files, LocalDB, XML files, etc. The most important point that you need to remember is IIS is never going to serve files from this App\_Data folder. In our example, this folder is empty as we don’t have any data files. In our upcoming articles, we will discuss how to store application-related data files in App\_Data Folder.

##### **App\_Start:**

The App\_Start folder of an ASP.NET MVC application contains configuration-related class files which are needed to be executed at the time of application starts. The classes like BundleConfig, FilterConfig, RouteConfig, IdentityConfig, etc are stored within this folder. We will discuss the use of each of these class files in detail in our upcoming articles.

App_Start folder of MVC application

##### **Content:**

The Content Folder of an ASP.NET MVC application contains static files such as image files, CSS files, icons files, etc. When we create a new ASP.NET MVC 5 application, then by default bootstrap.css, Site.css, and bootstrap.min.css files are included by Visual Studio as shown in the image below.

Content folder in MVC Application

##### **Controllers:**

The Controllers Folder of an ASP.NET MVC application contains all the controllers of your application. The Controllers are nothing but classes that are inherited from the base Controller class. The name of the Controller should end with the word “Controller”. It is this class that actually handles the user’s request i.e. the incoming HTTP Requests and returns a response. In our upcoming articles, we will discuss the controllers in detail.

Controllers folder in MVC application

##### **Fonts:**

The Fonts folder of an ASP.NET MVC application contains the custom font files that are required for the application.

Fonts folder of an MVC application

##### **Models:**

The Models folder of an ASP.NET MVC application contains the class files which are used to store the domain data (you can also say business data) as well as business logic to manage the data. In our example, the Models folder is empty as we have not created any models for our application. In our upcoming articles, we will discuss Models in detail.

##### **Scripts:**

The Scripts Folder of an ASP.NET MVC application contains all the JavaScript files that are required for your application. When we create an ASP.NET MVC 5 application, by default the necessary javascript files for jquery and Bootstrap are included. If you want to create any custom javascript files then they should be created within this folder or any subfolder of this folder. This is not mandatory but it is a good programming practice as in later time you can easily find out the javascript files.

Scripts Folder of an MVC application

##### **Views:**

The Views Folder of an ASP.NET MVC application contains all the “.cshtml” files of your application. In MVC, the .cshtml file is a file where we need to write the HTML code along with the C# code. The Views folder also includes separate folders for each and every controller for your application. For example, all the .cshtml files of the HomeController will be in the View => Home folder. We also have the Shared folder under the Views folder. The Shared Folder contains all the views which are needed to be shared by different controllers e.g. error files, layout files, etc.

Views Folder of an MVC application

Now, let us discuss the configuration files which are created by the framework by default:

##### **Global.asax:**

The Global.asax file in an ASP.NET MVC application allows us to write the code that we want to run at the application level or you can say global level, such as Application\_BeginRequest, Application\_Error, Application\_Start, Session\_Start, Session\_End, etc. In our upcoming articles, we will discuss the use of these application-level events in detail.

##### **Packages.config:**

The Packages.config file in an ASP.NET MVC application is managed by NuGet Package Manager which will keep track of what packages and versions have been installed in your application.

##### **Web.config:**

The Web.config file of an ASP.NET MVC application is one of the most useful and important files which contains the application-level configurations such as connection strings, global variables, etc.

Here in this article, we discussed the ASP.NET MVC File and Folder structure. Once you understand the need and use of each folder and file of an MVC application, then it is easy for you to find, store and organize project-related files.

In the next article, we are going to discuss **Controllers in ASP.NET MVC** Applications. Here, in this article, I try to explain the need and use of ASP.NET MVC File and Folder structure which are auto-generated by Visual Studio.

##### **What are ASP.NET MVC Views?**

In the MVC pattern, the view component contains the logic to represent the model data as a user interface with which the end-user can interact. Typically, it creates the user interface with the data from the model provided to it by the controller. So you can consider the Views in ASP.NET MVC as HTML templates embedded with Razor syntax which generates HTML content that sends to the client.

##### **Where ASP.NET MVC View Files are Stored?**

In ASP.NET MVC, the views are having a “**.cshtml**” extension when we use C# as the programming language with Razor syntax. Usually, each controller will have its own folder in which the controller-specific .cshtml files (views) are going to be saved. The controller-specific folders are going to be created within the Views folder. The most important point that you need to keep in mind is the view file name and the controller action name is going to be the same.

##### **Example to Understand ASP.NET MVC Views:**

Let’s say, we created an ASP.NET MVC application with two controllers i.e. StudentController and HomeController. The HomeController that we created is having the following three action methods.

1. **AboutUs()**
2. **ContactUs()**
3. **Index()**

Similarly, the StudentController is created with the following four action methods.

1. **Index()**
2. **Details()**
3. **Edit()**
4. **Delete()**

The views are going to be created and saved in the following order.

Views in ASP.NET MVC

As we have two controllers in our application, so there are two folders created with the Views folder one per Controller. The Home Folder is going to contain all the view files (i.e. cshtml files) which are specific to HomeController. Similarly, the Student Folder is going to contain all the .cshtml files which are specific to Student Controller. This is the reason why, the Home folder contains the Index, AboutUs, and ContactUs cshtml files. Similarly, the Student folder contains the Index, Details, Edit, and Delete view files.

##### **Understanding Views in ASP.NET MVC with Examples:**

To understand the views in the ASP.NET MVC application, let us first modify the HomeController as shown below.

**using** *System.Web.Mvc;*

**namespace** *FirstMVCDemo.Controllers*

**{**

**public** **class** HomeController : Controller

**{**

**public** ActionResult Index**()**

**{**

**return** View**()**;

**}**

**}**

**}**

In the above HomeController, we created an Action method that is going to return a view. In order to return a view from an action method in ASP.NET MVC Application, we need to use the **View()** extension method which is provided by **System.Web.Mvc.Controller** Base class. Now run the application and navigate to the **“/Home/Index”** URL and you will get the following error.

Views in ASP.NET MVC

##### **Let us understand why we got the above error.**

As we are returning a view from the Index action method of Home Controller, by default the MVC Framework will look for a file with the name **Index.aspx or Index.ascx** within the Home and Shared folder of the application if the view engine is APSX. If it is not found there then it will search for a view file with the name **Index.cshtml or Index.vbhtml** within the Home and Shared folder of your application.

If the requested view file is found in any of the above locations, then the View generates the necessary HTML and sends the generated HTML back to the client who initially made the request. On the other hand, if the requested view file is not found in any of the above locations, then we will get the above error.

##### **Adding Views in ASP.NET MVC Application**

In order to add the Index view, Right-click anywhere with the Index() function and then click on the “Add View” option which will open the following Add View dialog window. From the Add View window, provide the name for the view as Index, select Template as Empty, and uncheck the checkboxes for “create as a partial view” and “use a layout page” options. Finally, click on the Add button as shown below.

Adding Views in ASP.NET MVC

Once the Index view is created, then copy and paste the following into it.

@{

Layout = null;

}

<!DOCTYPE html>

**<html>**

**<head>**

**<meta** name="viewport" content="width=device-width" **/>**

**<title>**Index**</title>**

**</head>**

**<body>**

**<div>**

**<h1>**Index View Coming From Views/Home Folder**</h1>**

**</div>**

**</body>**

**</html>**

That’s it. Now run the application and navigates to the “**/Home/Index**” URL and you will see the output as expected. If you go to the definition of the **Controller** base class, then you will find there are eight overload versions of the View method which return a view as shown below.

Overloaded Versions of View in MVC

Each of the above-overloaded versions we will discuss as we progress through this course.

##### **Advantages of Using Views in ASP.NET MVC Application:**

The Views in ASP.NET MVC application provide the separation of concerns (codes). It means it separates the user interface from the rest of the application such as the business layer and data access layer. The ASP.NET MVC views use the advanced Razor syntax which makes it easy to switch between the HTML and C# code.

##### **What are the Models in ASP.NET MVC?**

The Models in ASP.NET MVC application are the component that contains a set of classes that are used to represent the business data (or domain data) as well as logic to manage the business data. So in simple words, we can say that the model in ASP.NET MVC is used to manage the domain data i.e. the state of the application in memory.

**Note:** It is not mandatory, but it is a good programming practice to store all model classes within the Models folder of an ASP.NET MVC application.

##### **Example to Understand Models in ASP.NET MVC.**

We need to display the employee information on a webpage as shown below.

Models in ASP.NET MVC

In order to store the employee data, we are going to use the Employee model class. To do so, right-click on the “Models” folder and then select Add => Class option. Provide the name as **Employee.cs** and finally click on the Add button as shown in the image below.

Adding Employee Models in ASP.NET MVC Application

**Now open the Employee.cs class file and then copy and paste the following code.**

**namespace** *FirstMVCDemo.Models*

**{**

**public** **class** Employee

**{**

**public** **int** EmployeeId **{** **get**; **set**; **}**

**public** **string** Name **{** **get**; **set**; **}**

**public** **string** Address **{** **get**; **set**; **}**

**public** **string** City **{** **get**; **set**; **}**

**public** **string** Gender **{** **get**; **set**; **}**

**public** **decimal** Salary **{** **get**; **set**; **}**

**}**

**}**

This is our Employee model which is going to hold the employee data in memory. As we already discussed, the Models in ASP.NET MVC Framework also contain the business logic to manage the business data. So in our example, in order to manage the employee data i.e. to perform the CRUD operation on the employee data, we are going to use the following EmployeeBusinessLayer model.

##### **Creating EmployeeBusinessLayer Model:**

Right-click on the Models folder and then add a new class file with the name EmployeeBusinessLayer.cs. Once you create the EmployeeBusinessLayer class file, then copy and paste the following code into it.

**namespace** *FirstMVCDemo.Models*

**{**

**public** **class** EmployeeBusinessLayer

**{**

**public** Employee GetEmployeeDetails**(int** EmployeeId**)**

**{**

//Here we hardcoded the data

//later we will discuss how to retrieve

//the data from a database

Employee employee = new Employee**()**

**{**

EmployeeId = EmployeeId,

Name = "Pranaya",

Gender = "Male",

City = "Mumbai",

Salary = 1000,

Address = "Andheri"

**}**;

**return** employee;

**}**

**}**

**}**

Once you created the required models for your application, then the model folder structure should look like below.Models in MVC

##### **Modifying the HomeController**

Now let us modify the HomeController class as shown below to use the Employee and EmployeeBusinessLayer model to retrieve the employee data.

**using** *FirstMVCDemo.Models;*

**using** *System.Web.Mvc;*

**namespace** *FirstMVCDemo.Controllers*

**{**

**public** **class** HomeController : Controller

**{**

**public** ActionResult Index**(int** id**)**

**{**

EmployeeBusinessLayer employeeBL = new EmployeeBusinessLayer**()**;

Employee employee = employeeBL.GetEmployeeDetails**(**id**)**;

**return** View**()**;

**}**

**}**

**}**

That’s it. In the next article, we will discuss how to pass the employee model data to a view, so that the view generates the necessary HTML.

In the ASP.NET MVC application, we can pass the model data from a controller to a view in many ways such as by using ViewBag, ViewData, TempData, Session, and Application as well as you can also use strongly typed views. You can also use the Session and Application State variable as we use in our traditional Web Forms to manage the data during a user session or throughout the application.

Now the most important question that comes to your mind is when to use ViewData, ViewBag, TempData, Session, and Application as each one has its own advantages and disadvantages. As we progress through this course you will come to know when to use one over another. Here in this article, I will show you how to use ViewData to pass the data from a controller action method to a view.

##### **What is ViewData in ASP.NET MVC?**

The ViewData in ASP.NET MVC Framework is a mechanism to pass the data from a controller action method to a view. If you go to the definition of ViewData by right-clicking on it and selecting go to definition, then you will see that ViewData is defined as a property in the ConstrollerBase class and its type is ViewDataDictionary as shown in the below image.

signature of the ViewData in MVC

As you can see in the above image, the return type of ViewData is ViewDataDictionary. Let’s have a look at the definition of the ViewDataDictionary class.

Definition of ViewDataDictionary class in ASP.NET MVC

As you can see, the ViewDataDictionary class implements the IDictionary interface. So we can say that the **ViewData in ASP.NET MVC** Framework is a dictionary object. As it is a dictionary object, it is going to store the data in the form of **key-value pairs** where each **key must be a string** and the value that we are passing to the dictionary is going to be stored in the form of an **object type**.

##### **How to Pass and Retrieve data From ViewData in ASP.NET MVC?**

The most important point that you need to remember is, as it stores the data in the form of an object while retrieving the data from ViewData type casting is required. If you are accessing string data from the ViewData, then it is not required to typecast the ViewData to string type. But it is mandatory to typecast explicitly to the actual type if you are accessing data other than the string type.

###### **ViewData in ASP.NET MVC with String Type:**

**ViewData in ASP.NET MVC with String values**

###### **ViewData in ASP.NET MVC with Complex Type:**

**ViewData in MVC with Complex Type:**

##### **Example of ViewData in ASP.NET MVC Application:**

Let us see an example to understand how to use the ViewData to pass data from a controller action method to a view. Please read our previous article as we are going to work with the same example. Let us first recap what we did in our previous article. First, we create the following Employee Model to hold the employee data in memory.

Employee Model to hold the employee data in memory.

Then we created the following EmployeeBusinessLayer model to manage the employee data. Here we created one method which will take the employee id as an input parameter and returns that employee information. As of now, we have hardcoded the employee data and in our upcoming article, we will discuss retrieving the employee data from a database like SQL Server, MySQL, Oracle, etc.

EmployeeBusinessLayer model to manage the employee data

Then we modify the Index action method of Home Controller as shown below to retrieve the employee data from EmployeeBusinesslayer and store it in the Employee model.

Index action method of Home Controller

##### **Passing ViewData From a Controller Action Method to a View:**

Now we will see, how to use the ViewData to pass the employee object to the Index view. Along with we are also going to pass the page Header using ViewData. So, modify the Index action method of the Home Controller class as shown below.

**using** *FirstMVCDemo.Models;*

**using** *System.Web.Mvc;*

**namespace** *FirstMVCDemo.Controllers*

**{**

**public** **class** HomeController : Controller

**{**

**public** ActionResult Index**()**

**{**

EmployeeBusinessLayer employeeBL = new EmployeeBusinessLayer**()**;

Employee employee = employeeBL.GetEmployeeDetails**(**102**)**;

ViewData**[**"Employee"**]** = employee;

ViewData**[**"Header"**]** = "Employee Details";

**return** View**()**;

**}**

**}**

**}**

##### **Accessing ViewData in a View:**

Now we will see how to access the ViewData within an ASP.NET MVC view. So, modify the Index Action method which is there within the Home folder in your application as shown below.

@{

Layout = null;

}

<!DOCTYPE html>

**<html>**

**<head>**

**<meta** name="viewport" content="width=device-width" **/>**

**<title>**Page Title**</title>**

**</head>**

**<body>**

@{

var employee = ViewData["Employee"]

as FirstMVCDemo.Models.Employee;

}

**<h2>**@ViewData["Header"]**</h2>**

**<table** style="font-family:Arial"**>**

**<tr>**

**<td>**Employee ID:**</td>**

**<td>**@employee.EmployeeId **</td>**

**</tr>**

**<tr>**

**<td>**Name:**</td>**

**<td>**@employee.Name**</td>**

**</tr>**

**<tr>**

**<td>**Gender:**</td>**

**<td>**@employee.Gender**</td>**

**</tr>**

**<tr>**

**<td>**City:**</td>**

**<td>**@employee.City**</td>**

**</tr>**

**<tr>**

**<td>**Salary:**</td>**

**<td>**@employee.Salary**</td>**

**</tr>**

**<tr>**

**<td>**Address:**</td>**

**<td>**@employee.Address**</td>**

**</tr>**

**</table>**

**</body>**

**</html>**

That’s it. Now run the application and you will see the employee details on the webpage as expected.

The ViewData in MVC is resolved dynamically at runtime. As a result, it does not provide compile-time error checking as well as we will not get the intelligence support. For example, if we miss-spell the key names then we wouldn’t get any compile-time error rather we came to know the error at runtime.

The ViewData in ASP.NET MVC can only transfer the data from a controller action method to a view. That means it is valid only during the current request.