**ASP.NET MVC 5**

**Developing an ASP.NET MVC 5 Application:** This is the last version of MVC from .NET Framework and the next version is first named as MVC 6 but later renamed as ASP.NET Core MVC and ported to .NET Core. So, first let’s learn working with ASP.NET MVC 5.

**MVC Project Structure:** when we create a new project either by choosing “Empty Project Template and select MVC CheckBox” or “MVC Project Template”, a set of files and folders are added in the project as following:

**1. Connected Services:** This is used for integrating Microsoft Azure Service into Visual Studio, so that we can easily host and manage our application to Azure Portal from Visual Studio it-self.

**2. App\_Data:** this folder contains local Data Source files like “.mdf” files, “.xml” files, “Excel” files, etc.

**3. App\_Start:** this folder contains a set of files which contains classes, and these classes get executed when the application starts execution. App\_Start folder can contain following files in it:

BundleConfig.cs => Contains BundleConfig Class

FilterConfig.cs => Contains FilterConfig Class

IdentityConfig.cs => Contains IdentityConfig Class

RouteConfig.cs => Contains RouteConfig Class

Startup.Auth.cs => Contains Startup Class

**Note:** if we opened an “Empty Project Template” then “App\_Start” folder contains only “RouteConfig.cs” file, whereas if we opened an “MVC Project Template” then “App\_Start” folder contains “BundleConfig.cs”, “FilterConfig.cs” and “RouteConfig.cs” files.

**4. Content:** this folder contains static files like “.css” files, “image” files, etc.

**Note:** if we open an “Empty Project Template” then Content folder will not be existing, whereas if we open an “MVC Project Template” then this folder exists with a set of “.css” (Bootstrap) files in it.

**5. Controllers:** this folder contains Controller classes where each Controller should be defined in a separate file.

**Note:** if we open an “Empty Project Template” then Controllers folder is empty, whereas if we open an “MVC Project Template” then this folder will contain 1 default Controller with the name “HomeController” under the file “HomeController.cs”.

**6. Fonts:** this folder contains any custom fonts that are being used in the application.

**Note:** if we open an “Empty Project Template” then Font’s folder will not exist, whereas if we open an “MVC Project Template” then this folder will exist with a set of fonts in it.

**7. Models:** this folder contains Model classes i.e., classes representing the Entities and Properties representing the Attributes of Entities as well as all the Methods to manipulate the data.

**Note:** if we open an “Empty Project Template” or “MVC Project Template” also this folder will exist but will be empty only.

**8. Scripts:** this folder contains Java Script or jQuery files that are used for development of the application.

**Note:** if we open an “Empty Project Template” then Scripts folder will not exist, whereas if we open an “MVC Project Template” then this folder will exist with a set of “.js” (jQuery) files in it.

**9. Views:** this folder contains all the View files (UI) that are required for this application and the extension of these files will be “.cshtml” in case we are working with “C#” Language or else if we are working with “VB” Language then extension of the files will be “.vbhtml” and these files contains both “C# / VB” and Html code in them, and we call these files as “Razor Pages”.

**Note:** if we open an “Empty Project Template” then Views folder comes with a file in it i.e., “Web.config”, whereas if we open an “MVC Project Template” then this folder will contain 2 sub folder in it with the names “Home” and “Shared” and these folders will contain a set of “.cshtml” files, and apart from that “Views” folder also contain “\_ViewStart.cshtml” and “Web.config” files.

Under the Views folder, for every Controller it will create a folder to store all the Views that are associated with that Controller, for example if there is a Controller with the name “HomeController” then there will be a “Home” folder created under Views folder to store all the Views associated with the “HomeController” class.

Under the Views folder, we can also have Shared folder containing all the Views that are common for all the Controllers in the application, for example Layout View, Error View, etc.

**10. Global.asax:** this is a file that contains a Global Configuration class, and, in this class, we define a set of methods: “Application\_Start”, “Application\_End”, “Application\_Error” that executes for application-level actions and other methods like: “Session\_Start” and “Session\_End” that executes for session level actions.

**11. Packages.config:** this contains information of packages that are used under this project, so that anyone can easily understand if there are any 3rd party packages consumed for the development of this application.

**12. Web.config:** this is a configuration file for the whole application which contains configuration settings like “App Settings”, “Connection Strings”, “Network Settings”, “Compiler Settings”, etc.

**Controller**

It is a class that handles user requests i.e., this class is responsible for taking all the incoming requests for an MVC Application.

The parent class for all Controllers we define should be the class “Controller” which is in turn a child of class “ControllerBase” and both the classes are defined in “System.Web.MVC” namespace.

**Adding a Controller to our MVC Project:** We can add a controller to an MVC Project in 2 different ways:

Manually defining Controller class.

Using scaffolding to define Controller class.

**What is Routing?**

**Ans:** Routing enables you to use URL’s that do not have to map to specific files in a Web Site.

By using routing, we can define URL patterns that map to request-handler files, but that do not necessarily include the names of those files in URL. In addition, we can also include placeholders in a URL pattern so that variable data can be passed to the request handler without requiring a query string.

**Can we define multiple routes in RouteConfig class?**

**Ans:** Yes, we can define multiple routes in the RouteConfig class, so that MVC Framework evaluates each route in sequence. It starts with the first configured route, and if the incoming URL doesn’t satisfy the URL pattern of the route, then it will evaluate the second route and so on. To understand this open “RouteConfig.cs” file and add the below code above “default” route:

routes.MapRoute(

name: "Student",

url: "NIT/Students",

defaults: new { controller = "Student", action = "Index" }

);

**Action Methods:** The methods that we defined under the Controller class for performing user interactions are known as Action methods i.e., users will directly call these methods for performing actions.

**To define Action Methods, we need to follow a set of rules:**

Action methods must be public, so every public method in a Controller class is an Action method only.

Action methods cannot be static because behind the screen instance of the Controller class is used for calling the Action methods.

It is not suggested to overload Action methods but if required we can still do that by decorating the method with “ActionName” attribute.

[ActionName("SayHello1")]

public string SayHello()

{

return "Hello how are you?";

}

[ActionName("SayHello2")]

public string SayHello(string Name)

{

return "Hello " + Name + " how are you?";

}

**Note:** in the above case we need to call the method with the “ActionName” we have defined by but not with the original method name, to test this define the above methods inside of “ParamsController” class and then we need to call them as following:

**We can execute the above methods following:**

**IIS Express:**

<http://localhost:port/Params/SayHello1>

<http://localhost:port/Params/SayHello2?Name=Raju>

If we want to define any non-action methods in a controller class, make sure they are not public or else decorate them with “NonAction” attribute and in this case when we try to access those methods from browser we get “404 Not Found” error.

< private or internal or protected or private protected or protected internal > string Display()

{

return "Non-Action Method";

}

Or

[NonAction]

public string Display()

{

return "Non-Action Method";

}

**Views**

View is the second important component in an MVC Application which acts as a UI (User Interface) for presenting the data or results to end users as well as for accepting data from users.

**What does a View contain in it?**

**Ans:** A View contains code for presentation or presentation logic which is a combination of “C# or VB” and HTML (CSS and Java Script also). When a request is sent for a View by the client, the logic implemented in the View gets processed and finally everything gets converted into Text (HTML) and we call this process as “Rendering”.

**What is rendering?**

**Ans:** Unfortunately, Internet still has bandwidth limitations and not every person is running on the same OS, same Web Browser or same Device, and these issues make it necessary to stick with HTML (Text Format) as our mark-up language of choice. So, in all the Server-Side technologies including ASP.NET; Web Server will process all the logic implemented by us using any language and converts the result into Text (HTML) which we call it as “**Rendering”** and then that HTML will be sent to clients as response. Views are processed by “View Engines” to render the results and MVC by default supports 2 different View Engines, those are:

Web Forms Engine

Razor Engine

**Razor Engine:** this is introduced in MVC 3.0 and in this case, View Pages will be having an extension of “.cshtml” or “.vbhtml” based on the language we use for developing the Views.

**Note:** Razor Engine is the most advanced View Engine and the most recommended also. Pages that are created for Razor Engine are known as Razor Pages and these Pages can contain either “HTML and C# or VB” code in them with an easy syntax.

**Sample Web Form page with a for loop:**

<%

for(int i=1;i<=10;i++)

{

%>

<h3>Hello World</h3>

<%

}

%>

**Sample Razor page with a for loop:**

@{

for(int i=1;i<=10;i++)

{

<h3>Hello World</h3>

}

}

**Note:** apart from the above 2 View Engines, ASP.NET MVC also supports many other third-party View Engines also like “NHaml”, “Brail”, “NDjango”, “Spark”, “Hasic”, etc.