# Introduction to ASP.NET MVC Framework

**What is ASP.NET MVC?**

The ASP.NET MVC is a web application development framework provided by Microsoft which is built on top of the .NET Framework. We can use this ASP.NET MVC Framework to develop web applications that provide a clean separation of code. The ASP.NET MVC framework is the most extensible and customizable framework provided by Microsoft.

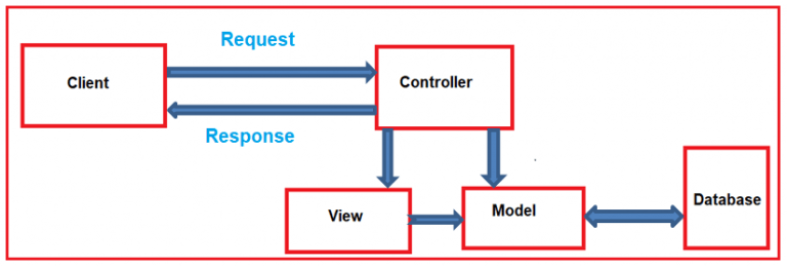
The **ASP.NET MVC Framework** is based on **MVC (Model-View-Controller) Design Pattern**. So the point that I need to highlight is **ASP.NET MVC** is a **Framework** whereas **MVC** is a **Design Pattern**.

The ASP.NET MVC Framework is not built from ground zero. You can consider it as an alternative approach to our traditional ASP.NET Web Forms Framework. As it is built on the top of the .NET Framework, developers enjoy almost all the ASP.NET features while working with the MVC application.

##### **How does the MVC Design Pattern work in ASP.NET MVC Application?**

Let us understand how does the MVC Design Pattern work in ASP.NET MVC application with an example. Let say to display the Employee details in a web page .

So, when the client (user) issues a request something like “**http://localhost:52230/Employee/details/2**” from a web browser then the request handled by the MVC framework as shown below.



The controller is the component who actually receives the incoming HTTP request and then handle that request. In order to handle the incoming HTTP request, the controller does several things are as follows.

The controller creates the model object if that is required by a view. The model is the component in MVC design which contains a set of classes to represent the domain data or business data as well as logic to manage the data.

The controller then selects a view to render the domain data or business data. The point that you need to remember is, while selecting a view, it is the responsibility of the controller to pass the model data.

In the MVC, the one and only responsibility of a view to display the model data. So, the responsibility of view is to generate the necessary HTML in order to render the model data. Once the HTML is generated by the view, then that HTML is then sent to the client who initially made the request.

##### **Model:**

The Model is the component in the MVC design pattern that manages that data i.e. state of the application in memory. The Model contains a set of classes that represent the data as well as logic to manage the data. So, in our example, the model is consists of Employee class to represent the Employee data as well as EmployeeBusinessLayer class to retrieve the Employee data from any persistent medium like a database.

public class EmpInfo

{

public int EmpId { get; set; }

public string Name { get; set; }

public string Dept { get; set; }

}

public class EmpBusinessLayer

{

public EmpInfo GetById(int empID)

{

EmpData objempdata = new EmpData();

objempdata.GetEmpDateaFromSQl();

EmpInfo \_empInfo = new EmpInfo()

{

EmpId = empID,

Name = "Tes123",

Dept = "IT",

};

return \_empInfo;

}

}

**So in short, a Model:**

1. In ASP.NET MVC is basically a C# or VB.net class to represent the data as well as to manage the data.
2. It is accessible by both controller and view.
3. It can be used to pass data from controller action methods to a view.
4. It can also be used by a view to display data in a page (HTML output).

##### **View:**

The view is the component in MVC Design Pattern which renders the model data as the user interface with which the end-user can interact. So, the View creates the user interface with data from the model. In our example, we want to display the Employee information in a web page. So here the Employee model carried the Employee data to the view. This is the Employee model which should be supplied by the controller to the view. The following code does the same thing.

@model MVCApplication.Models.EmpInfo

@{

ViewBag.Title = "Details";

}

<h2>Employee Info</h2>

<table>

<tr>

<td>

Emp. Id :

</td>

<td>

@Model.EmpId

</td>

</tr>

<tr>

<td>

Name :

</td>

<td>

@Model.Name

</td>

</tr>

<tr>

<td>

Dept :

</td>

<td>

@Model.Dept

</td>

</tr>

</table>

**So in short, a View**

1. In ASP.NET MVC is a cshtml page.
2. It contains all page specific HTML generation and formatting code.
3. A request to a view can only be made from a controller’s action method.
4. The one and only responsibility of a view to render the domain data.

##### **Controller:**

It contains the control flow logic. It is the one that will interact with both models and views to control the flow of application execution. The controller is the component in MVC Design Pattern which will handle the incoming HTTP Request. Based on the user actions, the respective controller will work with the model and view and then sends the response back to the user who initially made the request. In our example, when the client issued a request to the following URL

**http://localhost:52203/Employee/details/2**

Then that request is going to be mapped to the Details action method of the Employee Controller. Following is the code of our Controller class with the Details action method.

public ActionResult Index(int empId)

{

EmpBusinessLayer objemp\_BL = new EmpBusinessLayer();

EmpInfo empInfo=objemp\_BL.GetById(empId);

return View(empInfo);

}

**So, in short, a Controller**:

1. It is basically a C# or VB.NET class that is inherited from the **System.Web.Mvc.Controller**.
2. Is the component which will interact with both Models and views.
3. It contains action methods that are responsible for handling the incoming URL.
4. Can access and use the model class to pass the data to the views.

##### **Let’s Understand the above Code:**

As you can see in the above code, the Routing is configured using the **MapRoute()** extension method of **RouteCollection** class, where the Route name is “**Default**” and the URL pattern is “**{controller}/{action}/{id}**“. The Defaults value for the **controller** is **Home**, and the default **action** method is **Index** and the **id** parameter is **optional**.

**Note:** Always remember route name should be unique across the entire application. Route name can’t be duplicated.

##### **How to Register Routes in ASP.NET MVC?**

After configuring the routes in **RouteConfig** class, you need to register it in the **Application\_Start()** event in the **Global.asax** file. So that it includes all your routes into the RouteTable.

**Global.asax**

namespace FirstMVCDemo

{

public class MvcApplication : System.Web.HttpApplication

{

protected void Application\_Start()

{

AreaRegistration.RegisterAllAreas();

RouteConfig.RegisterRoutes(RouteTable.Routes);

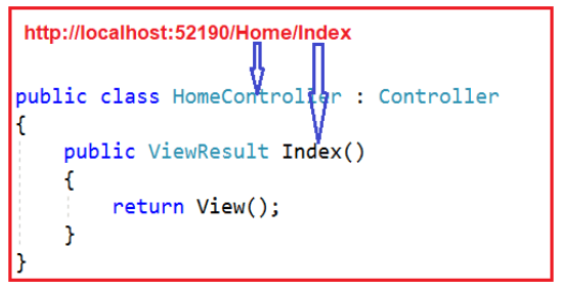
}

}

}

##### **What is Routing in MVC?**

The ASP.NET Routing module is responsible for mapping the incoming browser requests (i.e. the incoming URL) to a particular MVC controller action method. This mapping is done by the routing rules defined for your application. For example, if we issue a request to the **“/Home/Index**” URL, then it is the Index action method of Home Controller class which is going to handle the request as shown in the below image.



##### **How to Configure a Route in ASP.NET MVC?**

Every MVC application must configure (register) at least one route in the RouteConfig class and by default MVC Framework provide one default route. But you can configure as many as routes you want. You can register a route in the **RouteConfig** class, which is in RouteConfig.cs file under the **App\_Start** folder as shown below.

The following code illustrates how to configure a Route in the RouteConfig class.

namespace FirstMVCDemo

{

public class RouteConfig

{

public static void RegisterRoutes(RouteCollection routes)

{

routes.IgnoreRoute("{resource}.axd/{\*pathInfo}");

routes.MapRoute(

name: "Default", //Route Name

url: "{controller}/{action}/{id}", //Route Pattern

defaults: new

{

controller = "Home", //Controller Name

action = "Index", //Action method Name

id = UrlParameter.Optional //Defaut value for above defined parameter

}

);

}

}

}