# [ASP.NET](http://asp.net/" \t "_blank) MVC

MVC- Basically a Software Architectural design Pattern divides the Software application in to three interconnected parts.

**MVC Architecture:**

MVC framework is defined in the System.Web.Mvc assembly. From its properties can see its version and Runtime version (.Net Framework/CLR which runs the MVC dll.)

**3 Components:** Model, View, Control

**Model**- A class which consists of the shape of the data and business logic for the application.

*From below UI, can see model needs to hold vendor payment components, subcomponents, cost and status and hence it’s Model.*

Model:

**View**- (.cshtml /.vbhtml/.aspx/.ascx) page which corresponds the display page for the particular action.

**Controller**- A class file which handles the user requests and serve the user with model data and the appropriate view.

**Home** – View; **List of House Hold items** – Model; **Servant to Arrange Households** – Action Method;

**You** – Controller has several action methods for different tasks;

**MVC Versions:** [source from <http://www.tutorialsteacher.com/mvc/asp.net-mvc-version-history> ]

| **MVC Version** | **Visual Studio** | **.Net Version (Runtime Version)** | **Release date** | **Features** |
| --- | --- | --- | --- | --- |
| MVC 1.0 | VS2008 | .Net 3.5 | 13-Mar-2009 | * MVC architecture with webform engine * Routing * HTML Helpers * Ajax Helpers * Auto binding |
| MVC 2.0 | VS 2008, | .Net 3.5/4.0 | 10-Mar-2010 | * Area * Asynchronous controller * Html helper methods with lambda expression * DataAnnotations attributes * Client side validation * Custom template * Scaffolding |
| MVC 3.0 | VS 2010 | .Net 4.0 | 13-Jan-2011 | * Unobtrusive javascript validation * Razor view engine * Global filters * Remote validation * Dependency resolver for IoC * ViewBag |
| MVC 4.0 | VS 2010 SP1, VS 2012 | .NET 4.0/4.5 | 15-Aug-2012 | * Mobile project template * Bundling and minification * Support for Windows Azure SDK |
| MVC 5.0 | VS 2013 | .NET 4.5 | 17-oct-2013 | * Authentication filters * Bootstrap support * New scaffolding items * ASP.Net Identity |
| **MVC 5.2** – Current | VS 2013 | .NET 4.5 | 28-Aug-2014 | * Attribute based routing * bug fixes and minor features upate |

Version of the Existing MVC application can be found from **References->System.Web.Mvc->(Right Click)Properties->Version**

**MVC Folder Structure:**

**App\_Data:** Contains data related files like .mdf,.xml,local db files.

**App\_Start:** Contains class files like Authconfig.cs, BundleConfig.cs, Routeconfig.cs, Filterconfig.cs, WebApiConfig.cs

**Content**: Contains static files like images, css files. (See below it even contains **.xlsx** files templates)

**Controller**: Contains Controller classes which serve response for the user requests with appropriate view with data from model.

It should end with “Controller” name. Eg: Home**Controller**.cs

**Fonts**: custom font files for the application.

**Images**: contains image files.

**Models**: Model classes which contains the shape of the data for the application.

**Scripts**: Contains .js files and .vbscript files.

**Views**: Contains .cshtml/.vbhtml files inside the folder with controller name, because of each action method inside controller can correspond to different views.

**Shared**: Contains Layout.cshtml and other files which will be shared by all the controllers.

**Global.asax:** Contains events that raise in response to the application level, session level.

* Application\_Start()
* Application\_End()
* Session\_Start()
* Session\_End()

**Packages.Config:** Managed by Nuget packages, helps to track the packages installed and its version tracking.

**Web.Config:** Contains application level configurations

There are two Web.config files. 1. Root Folder 2.Views Folder

Root **Web.config** file consists of the application level configurations. Eg: below

<customErrors defaultRedirect="~/Info/Error" mode="On">

      <error statusCode="401" redirect="~/Account" />

    </customErrors>

<sessionState mode="InProc" timeout="20" cookieless="AutoDetect" />

    <pages>

      <namespaces>

        <add namespace="System.Web.Helpers" />

        <add namespace="System.Web.Mvc" />

        <add namespace="System.Web.Mvc. Ajax" />

        <add namespace="System.Web.Mvc.Html" />

        <add namespace="System.Web.Routing" />

        <add namespace="System.Web.WebPages" />

      </namespaces>

    </pages>

2. Web.config in Views folder

It blocks the direct access of the .cshtml page in the views folder.

Generally, this Web.config overrides the settings in the root Web.config settings.

<handlers>

      <remove name="BlockViewHandler"/>

      <add name="BlockViewHandler" path="\*" verb="\*" preCondition="integratedMode" type="System.Web.HttpNotFoundHandler" />

    </handlers>

**ROUTING:**

* Routing enables us to define a URL pattern (Eg: "{controller}/ {action}/ {id}") to map the user request to the appropriate request handler (i.e. Controller and action method in MVC).
* Routes can be configured in RouteConfig.cs file using Maproute method of RouteCollection class.
* Multiple routes can be configured and all configured routes will get stored in RouteTable class.
* Whenever request is made, the URL will compare with all the routes in RouteTable until the appropriate match occurs otherwise error will be thrown as no match found.
* Routes should be registered in Application\_Start event through RegisterRoutes method.
* Constraints can also be added with URL pattern.

**Registering Routes in Application\_Start () in Global.asax.cs file:**

 RouteConfig.RegisterRoutes(RouteTable.Routes);

**Configuring routes using route.MapRoute Method in RouteConfig.cs File:**

Two types: 1. Convention Based Routing. Below one.

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

2. Attributes Based Routing. [Could not test it. But understanding it a bit shows Routing will be performed based Attributes given over the action method to invoke it].

Below Eg: (Not tested)

[Route ("Pet/Breed/{petKey=123}")]

 Public ActionResult GetSpecificPet (string petKey)

 {

     return View ();

 }

**SCHAFFOLDING**:

* It is an automatic code generation framework introduced in order to reduce the time taken to develop code for view in MVC framework.
* The template (Create, Edit, List, Delete, Details) can be selected for code generation while creating view.

**ACTION METHOD:**

* All the public methods inside controller class are action methods.
* It should not be private/protected/static.
* Cannot be overloaded.
* Return type can be of ActionResult, ViewResult, ContentResult (string literal), any integral types including string.
* All the Contentresult, viewresult, redirecttoroute return types are derived type of ActionResult. Hence it is enough to use ActionResult for any of these types.
* Action methods can have parameters (including nullable types) as normal methods. MVC model binding automatically binds the parameters with query string values if matches with the name and Formcollection with the corresponding model class.

**ACTION SELECTORS: (Available in MVC 5) (BGV is coded in MVC 4.0)**

These are the attributes applied to action methods to help routing engine map to the appropriate request handler.

ActionName- [Eg: ActionName(“find”)]

* Helps to provide different name to action other than the action method name.

NonAction- [Eg: NonAction]

* Helps to define a method inside controller is not an action method.

ActionVerbs-

* [HttpPost], [HttpGet] or [AcceptVerbs(HttpVerbs.Get | HttpVerbs.Post)]

Helps to select the action method based on the Http request.

**RAZOR syntax:**

* Basically, there are two default view engines supported by MVC (razor and aspx).
* Razor is bit slower than aspx view engine.
* Razor is superior to .aspx due to its compactness (reduces the keystrokes than aspx view).
* Razor uses “@” symbol to specify server side code(c# or vb).
* (Eg: @if(condition){ -- code--}) Note: “@” end tag is not needed as razor’s parser is intelligent enough to find the end of server code and distinguish between content codes.

ASPX view uses <% …..code…..%> to write server code. (End tag is must).

@

{

for (int i=0; i<condition; i++)

{

……Logic code…..

If content need to be typed here use “<text>….content….</text>” or “@: textcontent”

}

}

**HTML helpers:**

HTML class resides in system.web.mvc.

HTML helpers are used to generate html elements using the model class object in **razor view.**It binds the model class properties values to appropriate html elements and also vice versa when posting the form to server.

Eg: @Html.RadioButton();

Here, Html is a property of type htmlhelpers included in base class razor view “System.Web.Mvc.WebViewPage”.

Below Code uses some Html helpers to generate some basic html elements.

<table border="1">

            <tr>

                <td>@Html.DisplayNameFor(Model => Model.Name)</td>

                <td>@Html.TextBoxFor(Model => Model.Name)</td>

            </tr>

            <tr>

                <td>@Html.DisplayName("Gender")</td>

                <td>Male: @Html.RadioButton("Gender", "Male") Female: @Html.RadioButton("Gender", "Female")</td>

            </tr>

            <tr>

                @\*<td>@Html.DisplayName("Hobbies")</td>\*@

                    <td>Books: @Html.CheckBox("Books",false) Music: @Html.CheckBox("Hobbies", false) Sports: @Html.CheckBox("Hobbies", false)</td>

            </tr>

            <tr>

                <td>@Html.DisplayNameFor(Model => Model.City)</td>

                <td>

                    @Html.DropDownListFor(Model => Model.City, new List<SelectListItem> { new SelectListItem { Text="Chennai",Value="Chennai"},

                       new SelectListItem { Text="Bangalore",Value="Bangalore"},

                       new SelectListItem { Text="Mumbai",Value="Mumbai"} }, "Select City")

        </tr>

    </table>

Helpers with “For” suffix (**DisplayNameFor**) are **strongly typed helpers** used to build model specific controls which is advisable for compile time type checking.

**MODEL BINDING:**

Model binding in MVC framework converts the query string to Action method parameters primitive type or complex type.

Eg: Please see below code for reference.

 [HttpPost]

        public ActionResult Create(FormCollection formCollection) or (string Name,string Gender,Datetime DOB)

        {

            Student student = new Student();

            student.Name = formCollection["Name"];

            student.Gender = formCollection["Gender"];

            student.Class = Convert.ToInt32(formCollection["Class"]);

            student.DOB = Convert.ToDateTime(formCollection["DOB"]);

            StudentBusinessClass studentBusinessClass = new StudentBusinessClass();

            studentBusinessClass.AddStudent(student);

            return RedirectToAction("Index");

        }

**BIND ATTRIBUTE:**

* MVC framework enables us to specify which properties of the model class needs to be bound.
* This is used, when you suppose some property should not be edited/updated on post request by user by any means(Eg: from fiddler a post request can be created with editing all properties) so by using Bind attribute we can exclude or include only the properties to be updated.

[Bind(Include=”Studentname”)] or [Bind(Exclude=”Age”)]

[HttpPost]

public ActionResult Edit([Bind(Include = "StudentId, StudentName")] Student std)

{

    var name = std.StudentName;

    //write code to update student

    return RedirectToAction("Index");

}

**Model Binding is two-step process.**

1. Value collection from incoming http request. –VALUE PROVIDERS

2. Populating the primitive or complex type with data.-MODEL BINDERS

**DATA VALIDATIONS:**

MVC uses Data Annotations attributes to implement validations. DataAnnotations includes different attributes for different validation rules, which can be applied to model class properties which automatically enforces these validation rules and display validation error messages in the view.

DataAnnotations attributes is included in System.ComponentModel.DataAnnotations namespace.

Eg: Below code for reference.

[Display(Name = "Name")][Required][StringLength(10,ErrorMessage ="Length should not exceed 11 characters")]

public string StudentName { get; set; }

[Required][Range(5,10)]

public int Age { get; set; }

Attributes: Required,StringLength,RegularExpression,MaxLength,MinLength etc. (for full refer <http://www.tutorialsteacher.com/mvc/implement-validation-in-asp.net-mvc>)

**3 methods:** ValidationMessage() // @[Html.ValidationMessage](http://www.tutorialsteacher.com/mvc/htmlhelper-validationmessage)("StudentName", "", new { @class = "text-danger" })

                     ValidationMessageFor() // @Html.ValidationMessageFor(model => model.Password, "", new { @class = "text-danger" })

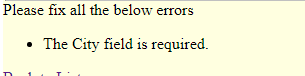
                      ValidationSummary() // @Html.ValidationSummary(false, "", new { @class = "text-danger" })

@Html.ValidationSummary – Used to display the summary of all the errors. It has several overloaded functions,

@Html.ValidationSummary(true) – will display only the field level errors, but does not display the summary.

@Html.ValidationSummary(false,"Please fix all the below errors") – Will display the summary along with the text we enter;

***Below Screenshot;***



**To Enable Client Side Validations using this Data Annotations, add the following in Web.config:**

<add key="ClientValidationEnabled" value="true" />

<add key="UnobtrusiveJavaScriptEnabled" value="true" />

**LAYOUT VIEW:**

* The [Layout view](http://www.tutorialsteacher.com/mvc/layout-view-in-asp.net-mvc) contains common parts of a UI. It is same like masterpage of [ASP.NET](http://asp.net/) webforms.
* It can be specified on the each view manually or by \_ViewStart.cshtml file in the views folder which has the following code.

@{

    Layout = "~/Views/Shared/\_Layout.cshtml";}

* We can set the Layout property in the individual view also, to override default layout page setting of \_ViewStart.cshtml
* 2 rendering methods: RenderBody () and RenderSection ().
* RenderBody – the method must be there in the Layout view to render the body (content) of the other views which are not in the named section of the Layout file. Should be mentioned only once in Layout file. -  @RenderBody()
* RenderSection can be configured as required or optional. If required, then all the child views must included that named section.- @RenderSection("featured", required: false)
* You can create own Layout view through Right Click on shared folder->Add->New Item->MVC 5 Layout page (Razor)

Example for named section:

The below section is defined in Employees.cshtml page, so when I call ***@RenderSection("TestSection", false)*** This Test section content will be shown wherever I have called it.

false is to mention it is optional for pages where this named section is not defined. Otherwise it will throw error.

@section TestSection

{

      @Html.Action ("Google")

}

**PARTIAL VIEW:**

* It is a Reusable View resides in Shared folder to make use of this contents in various views.
* Reduces the duplication of code. It is like user controls in [ASP.NET](http://asp.net/) web forms.
* Methods help to render partial views in a view are;

@Html.Partial() // Return type is **MvcHTMLString** and can be assigned to a variable, if needed can be manipulated before outputting to view.  
@Html.RenderPartial()// Return type is **VOID** and the result is directly sent to the output stream.

Below see the example implementation;

@Html.Partial ("\_LoginPartial")

@{

   Html.RenderPartial ("\_LoginPartial");

}

**From Performance perspective, “Html.RenderPartial” is bit a better due to its result is sent directly to output stream, but not “Html.Partial”. (*Not tested it*)**

**VIEWBAG and VIEWDATA:**

* **ViewBag and ViewData helps in transferring temporary data (Primitive or Complex) from controller to view page.** The value in Viewbag/View data property is alive only till current http request. It cannot help transferring data between action method / Controller. For that we need to use **TEMPDATA.**
* ViewBag is a wrapper around ViewData.
* ViewBag is dynamic type property of ControllerBase class which is the base class of all controllers.
* ViewData is a dictionary which contains name/value pairs to store data.

Eg: [ViewData ["students"] = studentList;

* Vice versa of transferring is not possible.

**Example:**

**How to Assign ViewBag/ViewData in Controller:**

public ActionResult Index()

        {

            //Basically using Business Logic - data layer method will be called to fetch the data, but in here we call data method here directly

            ViewBag.TestName = "asdmklhjasd";

            ViewData["TestName2"] = "testname 2 ";

            employees = Employees ();

            return View("Employees", employees);

        }

**How to Access it in View:**

@ViewBag.TestName <br /> [O/P: asdmklhjasd ]

@ViewData ["TestName2"]  [O/P: testname 2]

**TEMPDATA:**

* It is used to store temporary data which can be transferred to other action / controllers on the subsequent request.
* TempData is [TempDataDictionary](https://msdn.microsoft.com/en-us/library/system.web.mvc.tempdatadictionary(v=vs.118).aspx) type which stores values in name/value pairs.
* TempData["myData"]=”TestData”
* Tempdata value will be alive during subsequent request, on the third request the value will be lost. Call TempData.Keep() method to keep it for the third request.
* TempData internally use Session to store the data. So think of it as a short lived session.
* TempData value must be type cast before use. Check for null values to avoid runtime error.

Interview Question:

How to transmit data between Controllers? **Ans:** **Session, TempData and Model Entity through RedirectToAction method.**

Example:

Assign Value in TempData and Session:

public ActionResult Index()

{

ViewBag.Message = "Modify this template to jump-start your ASP.NET MVC application.";

ViewBag.Session = Session.SessionID;

Session["sessionname"] = "Session Content";

TempData["TempDataContent"] = "Tempdata Content";

return RedirectToAction("TempAction", "ACcount");

}

Access TempData in AccountController:

public ActionResult TempAction()

{

ViewBag.Message = "View Bag - Message";

ViewBag.TempDataContent = TempData["TempdataContent"];

TempData.Keep();

return this.View();

}

**TempAction View:**

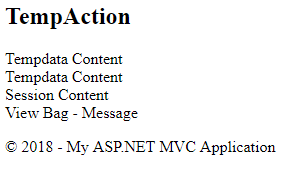
@ViewBag.TempDataContent <br />

@TempData ["TempDataContent"] <br /> // Valid access

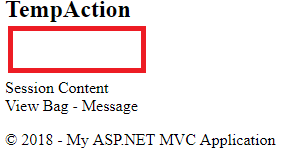
@Session ["sessionname"] <br />

@ViewBag.Message<br />

On First Load:



On Reloading the Page:



**Thing to be Noted here:**

For the first request TempData value was displaying, reloading the page it was not displayed. Hence, the TempData value exists only for subsequent request, for the third request need to include **TempData.keep();**

**FILTERS:**

* Filters are attributes applied to the action methods to perform certain action before or after the execution of the method. It can be done via programmatic way as well by implementing the corresponding interface.

*For Example: see below with Action Filters implementation and working explained;*

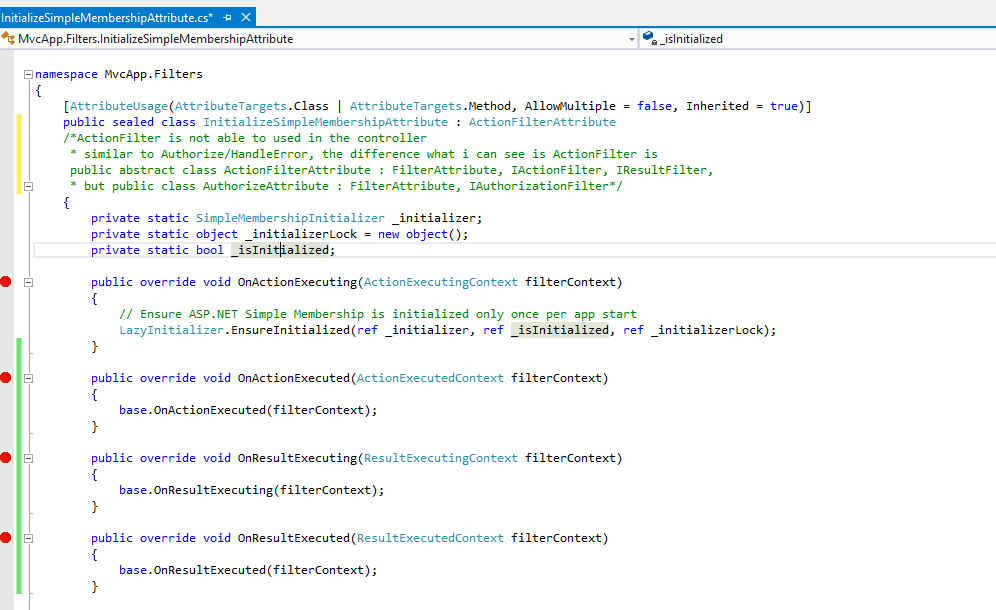
So here, whenever we add this filter “InitializeSimpleMembership” on to any controller/action method, the below mentioned 4 methods will be executed in order **“OnActionExecuting”, “OnActionExecuted”, “OnResultExecuting”, “OnResultExecuted”** based on our application needs, we can implement the logic in this override methods.

*Note: /\*ActionFilter is not able to used in the controller*

*\* similar to Authorize/HandleError, the difference what i can see is ActionFilter is*

*public abstract class ActionFilterAttribute : FilterAttribute, IActionFilter, IResultFilter,*

*\* but Authorize/HandleError has public class AuthorizeAttribute : FilterAttribute, IAuthorizationFilter\*/*



**Different types of filters:**

* **Authorization filter, Action filter, Result filter, Exception filter, ChildActionOnly**
* Filters can be applied globally in FilterConfig class, at controller level or action method level.
* Custom filter class can be created by implementing FilterAttribute class and corresponding interface.

Below code for the reference for few filter attributes usage:

 [HandleError]- //attribute for the HandleErrorAttribute Class

        public ActionResult SecureMethod()

        {

            throw new Exception("This is unhandled exception");

        }

        [Authorize] //attribute for the AuthorizeAttribute Class

        public ActionResult NonSecureMethod()

        {

            return View();

        }

 [OutputCache(Duration=10)]

        public ActionResult Index()

        {

            System.Threading.Thread.Sleep(3000);

            return View(db.tblEmployees.ToList());

        }

**BUNDLING** **and** **MINIFICATION**:

* Bundling is used to bundle the number of script or style files to help the server to download them on one http request which in turn improves the page load and applications performance.
* There are three types of Bundling technique;
* **ScriptBundle**- helps to bundle javascript files.
* **StyleBundle**- helps to bundle stylesheet files.
* **DynamicFolderBundle**: Represents a Bundle object that [ASP.NET](http://asp.net/) creates from a folder that contains files of the same type.

Bundling does **Minification** process which removes the white spaces, comments, new lines, rename the lengthy variables to single character to reduce the size of files which helps faster download for the server during http request on to client machine.

Bundles code should be added in Register bundles method of BundleConfig class file (App\_start).

Bundle class is inherited by StyleBundle and ScriptBundle classes.

Namespace:System.Web.Optimization.

Below see the usage:

bundles.Add(new StyleBundle("~/bundles/css").IncludeDirectory("~/Content/", "\*.css", true));

            bundles.Add(new ScriptBundle("~/bundles/script").IncludeDirectory("~/Scripts/", "\*.js", true));

You can also include individual files by method **Include(“filename”).**

**In view it is called by ;**

@Styles.Render("~/bundles/css")

        @Scripts.Render("~/bundles/script")

Using Wildcards for version changes;

bundles.Add(new ScriptBundle("~/bundles/jquery")

               .Include( "~/Scripts/jquery-{version}.js"));

From CDN as well we can add the files;

var cdnPath = "<http://ajax.aspnetcdn.com/ajax/jQuery/jquery-1.7.1.min.js>";

        bundles.Add(new ScriptBundle("~/bundles/jquery", cdnPath)

               .Include( "~/Scripts/jquery-{version}.js"));

**AREAS:**

Areas concept in MVC helps to partition the single large application to smaller units where each unit (i.e., Admin, Finance, HR etc.) will have separate MVC folder structure which makes easier to maintain the code for large applications.

If **“Admin”** module have **“Home”** controller and action method **“Index”-**It can be invoked by[**http://localhost/admin/home**](http://localhost/admin/home)

**Whenever an area is created (right clicking the project ->Add->Area) a (.CS) file will be created in which Register area method will be overridden for mapping the routes (route config).**

**These Areas will be registered Application\_start event as**AreaRegistration.RegisterAllAreas ();

**AUTHENTICATION in ASP.NET:**

<https://www.c-sharpcorner.com/article/authentication-and-authorization-in-Asp-Net/>

* IP Address restriction based on number of concurrent requests or over a period of time as well as configuring the behavior of IIS if IP is restricted – like what response to be given to the client in such case (Unauthorized/Forbidden/Not Found/Abort).

<https://docs.microsoft.com/en-us/iis/get-started/whats-new-in-iis-8/iis-80-dynamic-ip-address-restrictions>

<https://www.c-sharpcorner.com/article/authentication-filter-in-mvc-with-an-example/>