**1) What is MVC (Model view controller)?**

Model–view–controller **(MVC) is a Software Architectural Pattern** for implementing user interfaces. It divides a given software application into three interconnected parts, so as to separate internal representation of information from the way that information is presented to or accepted from the user.

MVC is a framework for building web applications using a MVC (Model View Controller) design:

* The Model(Business Layer) represents the application core (for instance a list of database records).
* The View(Display Layer) displays the data (the database records).
* The Controller(Input Control) handles the input (to the database records).

**2) Explain MVC application life cycle?**

**Step 1 Fill route:** - MVC requests are mapped to route tables which in turn specify which controller and action to be invoked. So if the request is the first request the first thing is to fill the route table with routes collection. This filling of route table happens in the global.asax file.

**Step 2 Fetch route:-** Depending on the URL sent "UrlRoutingModule" searches the route table to create "RouteData" object which has the details of which controller and action to invoke.

**Step 3 Request context created: -** The "RouteData" object is used to create the "RequestContext" object.

**Step 4 Controller instance created: -** This request object is sent to "MvcHandler" instance to create the controller class instance. Once the controller class object is created it calls the "Execute" method of the controller class.

Creating Response object: - This phase has two steps executing the action and finally sending the response as a result to the view.

**Step 5 Execute Action: -** The "ControllerActionInvoker" determines which action to be executed and executes the action.

**Step 6 Result sent: -** The action method executes and creates the type of result which can be a view result, file result, JSON result etc.

**3) What are the advantages of MVC?**

**Multiple view support**

Due to the separation of the model from the view, the user interface can display multiple views of the same data at the same time.

**Change Accommodation**

User interfaces tend to change more frequently than business rules (different colors, fonts, screen layouts, and levels of support for new devices such as cell phones or PDAs) because the model does not depend on the views, adding new types of views to the system generally does not affect the model. As a result, the scope of change is confined to the view.

**SoC – Separation of Concerns**

Separation of Concerns is one of the core advantages of ASP.NET MVC . The MVC framework provides a clean separation of the UI, Business Logic, Model or Data.

**More Control**

The ASP.NET MVC framework provides more control over HTML, JavaScript and CSS than the traditional Web Forms.

**Testability**

ASP.NET MVC framework provides better testability of the Web Application and good support for the test driven development too.

**Lightweight**

ASP.NET MVC framework doesn’t use View State and thus reduces the bandwidth of the requests to an extent.

**Full features of ASP.NET**

One of the key advantages of using ASP.NET MVC is that it is built on top of ASP.NET framework and hence most of the features of the ASP.NET like membership providers, roles, etc can still be used.

**4) Filters in MVC**

There are five types of Filters in ASP.NET MVC 5:

**Authentication Filters:**

Authentication filter runs before any other filter or action method. **Authentication confirms that you are a valid or invalid user**. Action filters implements the **IAuthenticationFilter interface**.

**Authorization Filters:**

The AuthorizeAttribute and RequireHttpsAttribute are the examples of Authorization Filters. **Authorization Filters are responsible for checking User Access**; these implement the IAuthorizationFilterinterface in the framework. These filters used to implement authentication and authorization for controller actions. For example, the Authorize filter is an example of an Authorization filter.

**Action Filters:**

Action Filter is an attribute that you can apply to a controller action or an entire controller. This filter will be called before and after the action starts executing and after the action has executed.

Action filters implement the **IActionFilter interface** that have **two methods OnActionExecuting and OnActionExecuted**. OnActionExecuting runs before the Action and gives an opportunity to cancel the Action call. These filters contain logic that is executed before and after a controller action executes, you can use an action filter, for instance, to modify the view data that a controller action returns.

**Result Filters:**

The **OutputCacheAttribute class is an example of Result Filters**. These implement the **IResultFilter interface** which like the IActionFilter has **OnResultExecuting and OnResultExecuted**. These filters contains logic that is executed before and after a view result is executed. Like if you want to modify a view result right before the view is rendered to the browser.

**ExceptionFilters:**

The **HandleErrorAttribute class is an example of ExceptionFilters**. These implement the **IExceptionFilter interface** and **they execute if there are any unhandled exceptions thrown during the execution pipeline**. These filters can be used as an exception filter to handle errors raised by either your controller actions or controller action results.

**5) Attributes in MVC**

[Key] - to mark the property as primary key

[Display(Name = "Is Active?")]

public bool Active { get; set; }

[Required(ErrorMessage = "Please write your LastName")]

public string LastName{ get; set; }

[StringLength(20, MinimumLength = 4, ErrorMessage = "Must be at least 4 characters long.")]

public string FirstName { get; set; }

[DisplayFormat(DataFormatString = "{0:yyyy-MM-dd}", ApplyFormatInEditMode = true)]

// [DisplayFormat(NullDisplayText = "Not specified")]

public DateTime? BirthDate{ get; set; }

[Range(0, 18, ErrorMessage = "The value must be between 0 to 18")]

public int Age { get; set; }

[Compare("ConfirmPassword", ErrorMessage = "Sorry, both should match")]

public string Password { get; set; }

[RegularExpression(@"\w+([-+.']\w+)\*@\w+([-.]\w+)\*\.\w+([-.]\w+)\*")]

public string Email { get; set; }

**6) List out different return types of a controller action method?**

There are total nine return types we can use to return results from controller to view.

**ViewResult (View):** This return type is used to return a webpage from an action method.

**PartialviewResult (Partialview):** This return type is used to send a part of a view which will be rendered in another view.

**RedirectResult (Redirect):** This return type is used to redirect to any other controller and action method depending on the URL.

**RedirectToRouteResult (RedirectToAction, RedirectToRoute):** This return type is used when we want to redirect to any other action method.

**ContentResult (Content):** This return type is used to return HTTP content type like text/plain as the result of the action.

**jsonResult (json):** This return type is used when we want to return a JSON message.

**javascriptResult (javascript):** This return type is used to return JavaScript code that will run in browser.

**FileResult (File):** This return type is used to send binary output in response.

**EmptyResult:** This return type is used to return nothing (void) in the result.

**7) Explain attribute based routing in MVC?**

In ASP.NET MVC 5.0 we have a new attribute route, By using the "Route" attribute we can define the URL structure. For example in the below code we have decorated the "GotoAbout" action with the route attribute. The route attribute says that the "GotoAbout" can be invoked using the URL structure "Users/about".

public class HomeController: Controller

{

[Route("Users/about")]

Public ActionResult GotoAbout()

{

return View();

}

}

**8) If we have multiple filters, what’s the sequence for execution?**

Authentication Filters

Authorization filters

Action filters

Response filters

Exception filters

**9) Explain RenderSection in MVC?**

RenderSection() is a method of the WebPageBase class. Scott wrote at one point, The first parameter to the "RenderSection()" helper method specifies the name of the section we want to render at that location in the layout template. The second parameter is optional, and allows us to define whether the section we are rendering is required or not. If a section is "required", then Razor will throw an error at runtime if that section is not implemented within a view template that is based on the layout file (that can make it easier to track down content errors). It returns the HTML content to render.

<div id="body">

@RenderSection("featured", required: false)

<section class="content-wrapper main-content clear-fix">

@RenderBody()

</section>

</div>

**10) Explain what is connected and dis-connected data access in ADO.NET?**

**Connected data access**: Through the DataReader objects of data-provider you can have connected data access. It provides fast access to data, and it does not allow editing.

**Dis-connected data access:** Through the DataAdapter object, disconnected data access is achieved. The dataset works independently of the database, and the data is editable.

**11) What is the difference between each version of MVC 2, 3, 4, 5 and 6?**

**MVC 6**

1. **ASP.NET MVC and Web API has been merged in to one.**
2. Side by side - deploy the runtime and framework with your application
3. **No need to recompile for every change. Just hit save and refresh the browser.**
4. **Dependency injection is inbuilt and part of MVC.**
5. Everything packaged with NuGet, Including the .NET runtime itself.
6. New JSON based project structure.
7. Compilation done with the new Roslyn real-time compiler.

**MVC 5**

1. **Asp.Net Identity**
2. **Attribute based routing**
3. Bootstrap in the MVC template
4. Filter overrides
5. **Authentication Filters**

**MVC 4**

1. **ASP.NET Web API**
2. New mobile project template
3. Refreshed and modernized default project templates
4. Many new features to support mobile apps

**MVC 3**

1. **Razor**
2. HTML 5 enabled templates
3. JavaScript and Ajax
4. **Support for Multiple View Engines**
5. Model Validation Improvements

**MVC 2**

1. Templated Helpers
2. Client-Side Validation
3. Areas
4. Asynchronous Controllers
5. Html.ValidationSummary Helper Method
6. DefaultValueAttribute in Action-Method Parameters
7. Binding Binary Data with Model Binders
8. DataAnnotations Attributes
9. Model-Validator Providers
10. New RequireHttpsAttribute Action Filter

**12) What is the difference between “HTML.TextBox” and “HTML.TextBoxFor”?**

Both provide the same HTML output, **“HTML.TextBoxFor” is strongly typed while “HTML.TextBox” isn’t**. Below is a simple HTML code which just creates a simple textbox with “FirstName” as name.

**Html.TextBox("FirstName")**

Below is “Html.TextBoxFor” code which creates HTML textbox using the property name ‘FirstName” from object “m”.

**Html.TextBoxFor(m =>m.FirstName)**

**13) Where is the route mapping code written?**

The route mapping code is written in "RouteConfig.cs" file and registered using "Global.asax" application start event.

**14) What is the difference between Temp data, View, and View Bag?**

In ASP.NET MVC there are three ways to pass/store data between the controllers and views.

**ViewData**

* ViewData is used to pass data from controller to view.
* It is derived from ViewDataDictionary class.
* It is available for the current request only.
* **Requires typecasting for complex data type and checks for null values to avoid error**.
* If redirection occurs, then its value becomes null.

**ViewBag**

* ViewBag is also used to pass data from the controller to the respective view.
* ViewBag is a dynamic property that takes advantage of the new dynamic features in C# 4.0
* It is also available for the current request only.
* If redirection occurs, then its value becomes null.
* **Doesn’t require typecasting for complex data type**.

**TempData**

* TempData is derived from TempDataDictionary class
* TempData is used to pass data from the current request to the next request
* It keeps the information for the time of an HTTP Request. This means only from one page to another. It helps to maintain the data when we move from one controller to another controller or from one action to another action
* It requires typecasting for complex data type and checks for null values to avoid error. Generally, it is used to store only one time messages like the error messages and validation messages

**15) Explain attribute based routing in MVC?**

In ASP.NET MVC 5.0 we have a new attribute route, By using the "Route" attribute we can define the URL structure. For example in the below code we have decorated the "GotoAbout" action with the route attribute. The route attribute says that the "GotoAbout" can be invoked using the URL structure "Users/about".

public class HomeController: Controller

{

[Route("Users/about")]

Public ActionResult GotoAbout()

{

return View();

}

}

**Note:** You can make a URI parameter optional by adding a question mark to the route parameter.

// eg: /books

// eg: /books/1430210079

**[Route(“books/{isbn?}”)]**

public ActionResult View(string isbn)

{

if (!String.IsNullOrEmpty(isbn))

{

return View(“OneBook”, GetBook(isbn));

}

return View(“AllBooks”, GetBooks());

}

**Note:** You can also specify a default value by using the form parameter=value.

// eg: /books/lang

// eg: /books/lang/en

// eg: /books/lang/he

**[Route(“books/lang/{lang=en}”)]**

public ActionResult ViewByLanguage(string lang)

{

return View(“OneBook”, GetBooksByLanguage(lang));

}

**Note:** Use a tilde (~) on the method attribute to override the route prefix if needed:

[RoutePrefix(“reviews”)]

public class ReviewsController : Controller

{

// eg.: /spotlight-review

**[Route(“~/spotlight-review”)]**

public ActionResult ShowSpotlight() { … }

}

**16) How to implement Forms authentication in MVC?**

ASP.NET forms authentication occurs after IIS authentication is completed. You can configure forms authentication by using forms element with in web.config file of your application. The default attribute values for forms authentication are shown below:

<system.web>

<authenticationmode="Forms">

<formsloginUrl="Login.aspx" protection="All" timeout="30" name=".ASPXAUTH" path="/" requireSSL="false" slidingExpiration="true" defaultUrl="default.aspx" cookieless="UseDeviceProfile" enableCrossAppRedirects="false" />

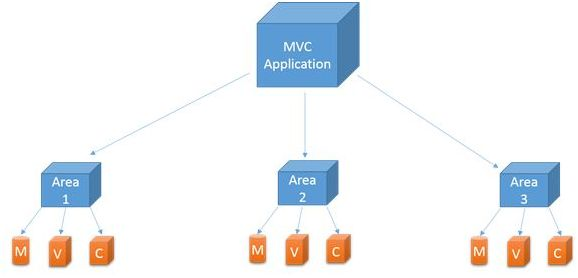
</authentication>

</system.web>

The FormsAuthentication class creates the authentication cookie automatically when SetAuthCookie() or RedirectFromLoginPage() methods are called. The value of authentication cookie contains a string representation of the encrypted and signed FormsAuthenticationTicket object.

**17) What is Areas in MVC?**

From ASP.Net MVC 2.0 Microsoft provided a new feature in MVC applications, Areas. **Areas are just a way to divide or “isolate” the modules of large applications in multiple or separated MVC.** like:



When you add an area to a project, a route for the area is defined in an AreaRegistration file. The route sends requests to the area based on the request URL. To register routes for areas, you add code to the Global.asax file that can automatically find the area routes in the AreaRegistration file.

**18) What is the difference between ActionResult and ViewResult?**

* ActionResult is an abstract class while ViewResult derives from the ActionResult class.
* ActionResult has several derived classes like ViewResult, JsonResult, FileStreamResult, and so on.
* ActionResult can be used to exploit polymorphism and dynamism. So if you are returning different types of views dynamically, ActionResult is the best thing. For example in the below code snippet, you can see we have a simple action called DynamicView. Depending on the flag (IsHtmlView) it will either return a ViewResult or JsonResult.

public ActionResult DynamicView()

{

if (IsHtmlView)

return View(); // returns simple ViewResult

else

return Json(); // returns JsonResult view

}

**19) What is the Html.AntiForgeryToken helper function for?**

Basically the antiforgery tokens stop anyone from submitting requests to your site that are generated by a malicious script not generated by the actual user. When you use it, your form will contain a hidden field and a corresponding cookie will also be set in the browser. Then, when the form is submitted, the hidden field is checked against the cookie value (assuming that ValidateAntiForgeryTokenAttribute is used): if the field and the cookie match then the form post is probably genuine; if they don't then it's probably not.

**20) Difference between static class and singleton?**

1- Static is a keyword and Singleton is a design pattern.

2- Singleton gives you an object, while static classes just provides static methods.

3- Singleton is an object creational pattern with one instance of the class.

4- Singleton can implement interfaces, inherit from other classes and it aligns with the OOPS concepts.

5- Singleton object can be passed as a reference whereas static object can't passed as a reference to other methods or objects.

6- Singleton supports object disposal.

7- Singleton object is stored on heap whereas static class is stored on stack

8- Singleton object can be cloned as well.

**21) Difference between @Html.Action and @Html.RenderAction**

**@Html.Action**

This Html.Action renders partial view as an HTML string so we can store it in another string variable. It is string return type method so first it returns result as a string then renders result to response.

**@Html.RenderAction**

This is also same as Html.Action but **main difference is that it renders result directly to response that’s why it is more efficient** if the action returns a large amount of HTML over @Html.Action.

**22) Difference between NonAction and ChildActionOnly Attribute**

1. An action method is a public method in a controller that can be invoked using a URL. So, by default, if you have any public method in a controller then it can be invoked using a URL request. ***To restrict access to public methods in a controller, NonAction attribute can be used****.*
2. Child action methods are different from NonAction methods, in that NonAction methods cannot be invoked using Action() or RenderAction() helpers.
3. ***Child action methods will not respond to URL requests***. If an attempt is made, a runtime error will be thrown stating - **Child action is accessible only by a child request.**
4. ***Child action methods can be invoked by making child request from a view using Action*() and RenderAction() Html helpers**.
5. An action method doesn’t need to have [ChildActionOnly] attribute to be used as a child action, but use this attribute to prevent if you want to prevent the action method from being invoked as a result of a user request.
6. Child actions are typically associated with partial views, although this is not compulsory.
7. **Using child action methods, it is possible to cache portions of a view. This is the main advantage of child action methods.**

[ChildActionOnly]

public string GetEmployeeCount()

{

return "Employee Count = " + db.employees.Count().ToString() + " @ " + DateTime.Now.ToString();

}

Calling above method using **@Html.Action("GetEmployeeCount")**  will not give error.

[NonAction]

public string Test()

{

return "Employee Count = " + db.employees.Count().ToString() + " @ " + DateTime.Now.ToString();

}

Calling above method using **@Html.Action("GetEmployeeCount")**  will give error.

**23) Benefits of Unit Tests**

1. Find bugs early
2. Significantly reduce production bugs
3. Unit Test make complex code easy to understand
4. Provide Documentation
5. Save Development Time
6. Easier to change and refactor code

**24) Dependency Injection**

With DI it's easy to share data and functionality as the angular Injector Provide Singleton i.e. single instance of the service

**25) What is Routing in MVC?**

Routing is a mechanism in MVC that decides which action method of a controller class to execute. Without routing there's no way an action method can be mapped to a request. Routing is a part of the MVC architecture so ASP.NET MVC supports routing by default.

**26) Difference between TextBoxFor and EditorFor**

**TextBoxFor:** It will render like text input html element corresponding to specified expression. In simple word it will always render like an input textbox irrespective datatype of the property which is getting bind with the control.

**EditorFor:** This control is bit smart. It renders HTML markup based on the datatype of the property. E.g. suppose there is a boolean property in model. To render this property in the view as a checkbox either we can use CheckBoxFor or EditorFor. Both will be generate the same markup.

What is the advantage of using EditorFor?

As we know, depending on the datatype of the property it generates the html markup. So suppose tomorrow if we change the datatype of property in the model, no need to change anything in the view. EditorFor control will change the html markup automatically.

**27) How route table is created in ASP.NET MVC?**

When an MVC application first starts, the Application\_Start() method is called. This

method, in turn, calls the RegisterRoutes() method. The RegisterRoutes() method

creates the route table.

**28) Ajax Helpers**

Install-Package **jquery.unobstrusive.ajax.js**

**Microsoft.jQuery.Unobtrusive.Ajax**

1. @Ajax.ActionLink("View  All Student Info", "AllStudent", "Home",
2. **new** AjaxOptions
3. {
4. UpdateTargetId = "divAllStudent",
5. OnBegin = "fnOnBegin",
6. InsertionMode = InsertionMode.Replace,
7. HttpMethod = "GET",
8. LoadingElementId = "imgloader",
9. OnSuccess= "fnSuccess",
10. Confirm="Do you want to get all student info ?????"
11. }, **new** { @**class** = "btn btn-default" })
12. Allow cache

**By Default it is false**, it specifies whether you want to cache the page requested by the browser or not by specifying its value as true or false.

1. Confirm

Like jQuery confirm dialog.

1. HttpMethod

Specifies the HTTP request method that is either GET or POST. **By default it is POST.**

1. InsertionMode

Specify the way of populating the target container. The possible values are InsertAfter, InsertBefore and **Replace (which is the default).**

1. LoadingElementDuration

Here you can specify the value in milliseconds that controls the duration of the animation loading element.

1. LoadingElementId

HTML element that will be displayed when AJAX call is in-progress.

Generally we used this property to display the loader element i.e.

<div id="imgloader" style="display:none;position:absolute;top:50%;left:50%;padding:2px;">

<img src="~/Content/loader.gif" />

</div>

1. OnBegin

This property specifies the name of the java script function that is called just before Ajax starts. So here you can perform some validation or other operation that is required before Ajax starts

1. OnComplete

Name of the java script function that will be called when response data has been represented by the AJAX call.

1. OnFailure

Name of the java script function to call when AJAX request returns error.

1. OnSuccess

Name of the java script function to call after the AJAX request returns successfully

1. UpdateTargetId

This property is used to specify the ID of HTML element that is updated by using the response;

1. Url

Specify the URL to make call i.e. External URL link

**29) Send data from View to Action**

1. Using Traditional approach (**Request["txtAmount"].ToString()**)
2. Using the FormCollection Object
3. Using the Parameters
4. Strongly type model binding to view

**30) Bundling and Minification**

The ASP.NETMVC offers bundling and minification technique by **System.Web.Optimization** class, which exists under the **System.Web.Optimization dll.**

Both bundling and minification are the two separate techniques to reduce the load time. The **bundling reduces the number of requests to the Server**, while the **minification reduces the size of the requested assets.**

**Bundling**

The bundle is a logical group of physical files, which loads in a single HTTP request. We have separate CSS files, which can be loaded in a single request with the help of bundling. The bundling also can create for JavaScript files separately. A bundle can’t contain both CSS and JavaScript files. We need to create a separate bundle for CSS and JavaScript files.

public class BundleConfig

{

//For more information on bundling, visit https://go.microsoft.com/fwlink/?LinkId=301862

public static void RegisterBundles(BundleCollection bundles)

{

bundles.Add(new StyleBundle("~/Content/css").Include(

"~/Content/bootstrap.css",

"~/Content/site.css"));

bundles.Add(new ScriptBundle("~/bundles/bootstrap").Include(

"~/Scripts/bootstrap.js"));

}

}

Afterwards, we need to register this bundle in the Application. We call this method in the Application\_Start method of the global.asax.cs file and register this bundle module to bundle config.

public class MvcApplication : System.Web.HttpApplication

{

protected void Application\_Start()

{

AreaRegistration.RegisterAllAreas();

FilterConfig.RegisterGlobalFilters(GlobalFilters.Filters);

RouteConfig.RegisterRoutes(RouteTable.Routes);

BundleConfig.RegisterBundles(BundleTable.Bundles);

}

}

Now, call this style bundle on the view, as per the code snippet, given below:

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

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

The bundle doesn’t work in the debug mode. Thus, we set the debug value false in web.config file, as shown in the snippet, given below:

<system.web>

<compilation debug="false" targetFramework="4.7.2" />

<httpRuntime targetFramework="4.7.2" />

</system.web>

**Minification**  
  
The Minification is a technique for removing unnecessary characters (white space, newline, tab), comments and short variable names from the text based files such as JavaScript and CSS files without expecting alter functionality to reduce the size, which causes improved load times of a Webpage.

**31) Busting Browser's Cache by Bundling**

As we upload the changes in the static resources such as CSS and JS files on the live server, the resources changes but it does not update on the Browser, because the Browser's cache resources are based on URLs automatically. Thus, when a Web page requests a resource, it checks in cache first. If the resource is found in cache, use cached copy rather than retrieving the resources from the Server. Hence, whenever you change the content of CSS and JS, files will not reflect on the Browser. For this, you need to force the Browser for refreshing/reloading.

**The bundles set the HTTP expires header**, one year from when the bundle is created. As we have a CSS bundle resource, which loads on the Browser with the following link.

<link href="/Content/css?v=Bz3KZjU\_pdOm2wAVr7z\_ylCuQzQDs1O8N6pV4cvXc\_Q1" rel="stylesheet"/>

The /Content/css style bundle contains the query string pair v=Bz3KZjU\_pdOm2wAVr7z\_ylCuQzQDs1O8N6pV4cvXc\_Q1. The query string v has a value token. This token is a unique identifier, which is used for caching. As long as the bundle /Content/css dosen’t change, the request for this bundle uses its same token. If any file in the bundle changes, ASP.NET optimization framework will generate a new token, guaranteeing the browser requests for the bundle will get the latest bundle.

**32) DLL Hell**

Before some time, if we install an application the .dll of that application get stored in the registry, then if we install other application that has same name .dll  that means previously installed .dll get overwrite by the same name new .dll. Ok for newly installed application but previously installed application can’t get execute further. This is big problem in context of version of same application. This is Dell-Hell problem.

**Solution of Dll-Hell Problem**

This problem of dynamic link library (.dll) is resolved through Versioning.

**Versioning:**

Versioning is the technique to provide version to the .dll to prevent them from replacement. GAC (Global assembly cache) is the separate memory like cache, it is used to remove load form operating system.

To add version in assembly we just write in Program:

[assembly: AssemblyVersion("1.0.0.0")]

MajorVersion**.**MinorVersion**.**BuildNumber**.**Revision

We can do versioning only with shared assembly because to install .dll in GAC, so we need to have strong key name.

**33) GAC**

The Global Assembly Cache (GAC) is a folder in Windows directory to store the .NET assemblies that are specifically designated to be shared by all applications executed on a system.

**GAC path**

1) C:\Windows\Assembly (for .NET 2.0 ~ 3.5)

2) C:\Windows\Microsoft.NET\assembly (for .NET 4.0)

**How to install an assembly into GAC (as Administrator)**

1) Drag and Drop

2) Use GacUtil.exe with Visual Studio Command Prompt

**gacutil -i [Path][Assembly Name].dll**

**Note:** To install an assembly into the GAC, the assembly **must be strongly named**. Otherwise you get an error like this: Failure adding assembly to the cache: Attempt to install an assembly without a strong name.

**How to uninstall an assembly from GAC (as Administrator)**

gacutil -u [Assembly Name], Version=1.0.0.0, PublickeyToken=7896a3567gh

**34) Different type of return**

**return View() :** It tells MVC to generate an HTML template to be displayed and sends it to the browser without making a new request.

**return Redirect() :** Makes a new request for the new action but in order to use this method we have to **specify the full URL**.

**e.g.** return Redirect("https://www.google.com/")

**return RedirectToRoute()** : Redirect to action from the specified URL defined in the route table that is defined in RouteConfig.cs file.

e.g.

routes.MapRoute(

name: "test",

url: "CascadeDropdown/Index",

defaults: new { controller = "CascadeDropdown", action = "Index", id = UrlParameter.Optional }

);

return RedirectToRoute("test");

**Note:**If you try to use in below format will give runtime error

return RedirectToRoute("https://www.google.com/");

Error: *A route named 'https://www.google.com/' could not be found in the route collection.*

**return RedirectToAction() :** To redirect to a different action which can be in the same or different controller.

e.g. return RedirectToAction("AddEmployee"); //Same Controller

return RedirectToAction("AddEmployee", "Employee"); //Different Controller

**return RedirectPermanent()** **:** Sends the browser an HTTP 301 (Moved Permanently) status code whereas Redirect will send an HTTP 302 status code.

**Example**

Let's say that you have users in your system. You also have an option to delete existing users. Your website has a resource /user/{userid} that displays the details of a given user. If the user has been deleted, you must redirect to the /user/does-not-exist page. In this case:

If the user will never be restored again, you should use RedirectPermanent so the browser can go directly to /user/does-not-exist in subsequent requests even if the URL points to /user/{userid}.

**35) Manage User Session**

**1) InProc**  
  
The InProc Session mode is the default Session Mode. Using this Session Mode the Session Mode is stored in the application worker process (**aspnet\_wp.exe**) in the application domain. The Worker Process is dependent on the IIS server version.

In Inproc Session mode the important point is:  
  
1. When we get the WebForm2 then if we end the task of aspnet\_wp.exe from the Task Manager and then again reload the WebForm2 then you will get no output, that means that all the sessions are stored in a worker process and after closing the task the session will be lost.  
  
**Advantages of the InProc mode:**

* 1. Inproc session mode is very easy to implement, the only thing that is required is <sessionState mode="InProc">.
  2. It will perform fast because the session is kept on the web server within the ASP.NET Worker Process.
  3. Data is stored separately and the data is secure so it is suitable for web applications.
  4. In this mode there is no need to serialize and deserialize the object for storage and retrieval of the data.

**Disadvantages of InProc mode:**

1. Session data is lost when the worker process or application process is recycled.
2. Increase the load of server: In this Mode the sessions are stored on the web server. If the number of sessions is increased then the load of the server is also increased and the scalability could be an issue.

**2) StateServer  
  
Also known as Out-Proc Session mode**. StateServer uses a stand-alone Windows Service that is independent of IIS and can also be run on a separate server. This session state is totally managed by **aspnet\_state.exe**. And the Session Variables are stored in an **ASP.NET State service** (Can be found in Services.msc).

<sessionState mode="StateServer" stateConnectionString="tcpip=localhost:42424">

The ASP.NET State Services can be present on a Web Server or a dedicated machine. So if we closed the worker Process (aspnet\_wp.exe) then it is also not affected.

**Advantage of State Service**

1. No issue about Worker Process because it's not dependent on Worker Process.
2. Scalability is also increased because it keeps data separate from IIS.

**Disadvantage of State Service**

1. Performance Decrease : When the request goes to the server the object is serialized and deserialized so for that the performance is decreased so it is slower than the InProc State Mode.
2. If the request goes to the server and for some reason the ASP.NET State Service is restarted then all the sessions will be destroyed.

**3) SQL Server**

In this mode the session data is stored inside the SQL Server database.

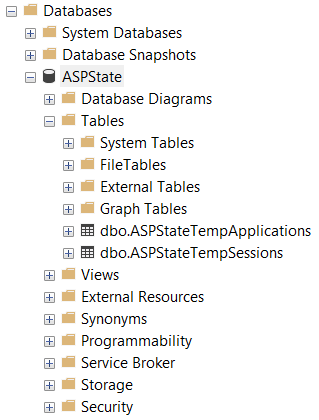
Step 1: From the command prompt, go to your Framework version directory, for example: C:\Windows\Microsoft.NET\Framework\v4.0.30319 and search for the **aspnet\_regsql.exe** that in which version these files are present and execute the file in the command prompt.

**C:\Windows\Microsoft.NET\Framework\v4.0.30319>aspnet\_regsql.exe –S DESKTOP-5RLLPLU –E –ssadd –sstype p**

**Parameter Description:**

1. ssadd: Add support for SQLServer mode session state.
2. sstype p: P stands for Persisted. It persists the session data on the server.
3. S : Server name.
4. U : User name.
5. P : Password.
6. E : Authentication using the windows credential of the currently logged on users.

Step 2: After executing the command, open the database:



Step 3: Now for configuration we need to write the connection string in the web.config file:

<sessionState mode="SQLServer" sqlConnectionString="Server= DESKTOP-5RLLPLU;Integrated Security=true">

Step 4: And the webfrom1.aspx and webform2.aspx code are the same.

Step 5: Now after sending a request to the server open the database and here the session id is stored and by default its expiration time is 20 minutes.

**Advantages of SQLServer mode:**

1. SQL Server is a more reliable and secure option.
2. It's not dependent on the Worker Process and ASP.NET State Service so if it restarts then it is also not affected by the session.
3. Scalability: The scalability is increased compared to InProc and State Server because the session is stored in the database, no matter how many requests on the web server.

**Disadvantages of SQLServer mode:**

1. It's slower than StateServer and InProc Session mode.
2. It must be serialized and deserialized.

**The following are the reasons to use the SQLServer Session mode**:

1. When we need a session with more security, in other words we need that the data of the session is more secure.
2. If there happen to be frequent server restarts, this is an ideal choice.
3. We can use SQLServer session mode when we need to share sessions between two different applications.

According to performance and durability the difference between InProc, State Server and SQL Server is:

**Session mode Performance Durability**

InProc More (1 processor and 1 server) less

State Server Medium (n processor and 1 server) Medium

SQL Server Less More

**ValidateInput Attribute**

The ValidateInput attribute is used to allow sending the HTML content or codes to the server which, **by default, is disabled by ASP.NET MVC to avoid XSS (Cross-Site Scripting) attacks**. This attribute is used to enable or disable the request validation. By default, request validation is enabled in ASP.NET MVC

[ValidateInput(false)] – Allow HTML content to be posted.

**ValidateAntiForgeryToken**

The basic purpose of ValidateAntiForgeryToken attribute is to prevent cross-site request forgery attacks. A cross-site request forgery is an attack in which a harmful script element, malicious command, or code is sent from the browser of a trusted user.

**Using Attribute and Convention Based Routing Together**

We can combine attribute routing with convention-based routing as mentioned here: [Attribute Routing in ASP.NET MVC 5](https://blogs.msdn.microsoft.com/webdev/2013/10/17/attribute-routing-in-asp-net-mvc-5/#enabling-attribute-routing)  
And for using both we should use MapMvcAttributeRoutes method before convention routing implementations:

public static void RegisterRoutes(RouteCollection routes)

{

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

routes.MapMvcAttributeRoutes();

routes.MapRoute(

name: “Default”,

url: “{controller}/{action}/{id}”,

defaults: new { controller = “Home”, action = “Index”, id = UrlParameter.Optional }

);

}

But the MapMvcAttributeRoutes removes action methods from StandardRouteMethods which the convetion-based routing uses. So we can't use convetion-based routing if we define a routing like that:

[RoutePrefix("d")]

[Route("{action=index}")]

public class DefaultController : Controller

{

[Route]

public ActionResult Index()

{

return View();

}

[Route("f")]

public ActionResult Foo()

{

return View();

}

}

If we want to reach the url: "/Default/Foo" which is proper for "{controller}/{action}/{id}" convention route, an exception occures like that A public action method 'Foo' was not found on controller 'Namespace...DefaultController'..

There is an explanation in the source code of ActionMethodSelectorBase:

protected void Initialize(Type controllerType)

{

//////

// The attribute routing mapper will remove methods from this set as they are mapped.

// The lookup tables are initialized lazily to ensure that direct routing's changes are respected.

StandardRouteMethods = new HashSet<MethodInfo>(ActionMethods);

}

If we can combine both, why this is happening? Is this a bug or an intentionally behaviour? Are there any workaround to use both at same time

**Is the session on one server will be available on other server**

1. **StateServer mode**, which stores session state in a separate process called the ASP.NET state service. This ensures that session state is preserved if the Web application is restarted and also makes session state available to multiple Web servers in a Web farm.
2. **SQLServer mode** stores session state in a SQL Server database. This ensures that session state is preserved if the Web application is restarted and also makes session state available to multiple Web servers in a Web farm.

**36) What happened when remove the conventional routing?**

**routes.MapRoute(**

**name: "Default",**

**url: "{controller}/{action}/{id}",**

**defaults: new { controller = "CascadeDropdown", action = "Index", id = UrlParameter.Optional }**

**);**

If we removed the above part we get the following error at the time of loading the project.

### HTTP Error 403.14 - Forbidden

The Web server is configured to not list the contents of this directory.

**37) Advantage and Disadvantage of Hidden fields**

**Advantages:**

1. They are simple to implement.

2. As data is cached on client side they work with Web Farms.

3. All browsers support hidden field.

4. No server resources are required.

**Disadvantages:**

1. They can be tampered creating a security hole.

2. Page performance decreases if you store large data, as the data are stored in page itself.

3. Hidden fields do not support rich structures, as HTML hidden fields are only single valued. Then you have to work around with delimiters etc. to handle complex structures.

**Difference between Route and endpoints**

An **endpoint** performs a specific function by taking one or more parameters and returning the resulting data. A **route is** the name you use to access the available **endpoints**. In layman's terms, the **route is** the URI, the **endpoint is** the action performed on the URI.

**Viewmodel in MVC**

* In ASP.NET MVC, ViewModel is a class that contains the fields which are represented in the strongly-typed view. It is used to pass data from controller to strongly-typed view.
* View models differ from domain models in that view models only contain the data (represented by properties) that you want to use on your view.

**How may number of worker process can we have**

The IIS setting you're referring to creates multiple processes, each with its own address space.  If you're using in-process caching of any kind, including in-process session state, it can't be shared between the worker processes.  Having multiple worker processes is similar to have multiple load-balanced servers, and is useful for similar reasons: in case one worker fails, the others can pick up the load.

You don't need to have more than one worker process in order for IIS to be able to use all of your available CPUs.  Multiple threads within a single worker will do that for you.

If your system is slow with one worker process, it's probably because you have long-running synchronous tasks that are causing the worker threads to block.  Assuming you're running IIS 7, you should be able to improve the situation by making the following change to Aspnet.config in C:\Windows\Microsoft.NET\Framework\v2.0.50727:

<configuration>

. . .

<system.web>

<applicationPool maxConcurrentRequestsPerCPU="24"

maxConcurrentThreadsPerCPU="0"

requestQueueLimit="5000" />

</system.web>

</configuration>

That tells the runtime to allow 24 concurrent requests per CPU, instead of the default of 12.

If this helps, then your site's performance could be improved further by switching to async pages -- which is something that I cover in detail in my book (see my signature).

**Model Binding**

<https://www.tutorialspoint.com/asp.net_mvc/asp.net_mvc_model_binding.htm#:~:text=ASP.NET%20MVC%20model%20binding,browser%20in%20an%20HTTP%20request.&text=Model%20binding%20is%20a%20well,and%20the%20C%23%20action%20methods>.

**ASP.NET MVC model binding allows you to map HTTP request data with a model**. It is the process of creating .NET objects using the data sent by the browser in an HTTP request. The ASP.NET Web Forms developers who are new to ASP.Net MVC are mostly confused how the values from View get converted to the Model class when it reaches the Action method of the Controller class, so this conversion is done by the Model binder.

**Model binding is a well-designed bridge between the HTTP request and the C# action methods. It makes it easy for developers to work with data on forms (views), because POST and GET is automatically transferred into a data model you specify. ASP.NET MVC uses default binders to complete this behind the scene.**

**Caching**

public class HomeController : Controller

{

[OutputCache(Duration = 10, VaryByParam = "none")]

public ActionResult Index()

{

return View();

}

}

**Where Content is Cached**

By default, when you use the [OutputCache] attribute, content is cached in three locations: the **web server, any proxy servers, and the web browser**. You can control exactly where content is cached by modifying the Location property of the [OutputCache] attribute.

You can set the Location property to any one of the following values:

· Any

· Client

· Downstream

· Server

· None

· ServerAndClient

By default, the Location property has the value Any. However, there are situations in which you might want to cache only on the browser or only on the server. For example, if you are caching information that is personalized for each user then you should not cache the information on the server. If you are displaying different information to different users then you should cache the information only on the client.

public class BadUserController : Controller

{

[OutputCache(Duration = 3600, VaryByParam = "none")]

public string GetName()

{

return "Hi " + User.Identity.Name;

}

}

The above code will display the same info to all the request once cached. To resolve this we need to change the cache location as below;

[OutputCache(Duration=3600, VaryByParam="none", Location=OutputCacheLocation.Client, NoStore=true)]

public string GetName()

{

return "Hi " + User.Identity.Name;

}

**The NoStore property is used to inform proxy servers and browser that they should not store a permanent copy of the cached content.**

The Master() action includes a VaryByParam property with the value "none". When the Master() action is invoked, the same cached version of the Master view is returned. Any form parameters or query string parameters are ignored

[OutputCache(Duration=int.MaxValue, VaryByParam="none")]

public ActionResult Master()

{

ViewData.Model = (from m in \_dataContext.Movies

select m).ToList();

return View();

}

The Details() action includes a VaryByParam property with the value "Id". When different values of the Id parameter are passed to the controller action, different cached versions of the Details view are generated.

[OutputCache(Duration = int.MaxValue, VaryByParam = "id")]

public ActionResult Details(int id)

{

ViewData.Model = \_dataContext.Movies.SingleOrDefault(m => m.Id == id);

return View();

}

It is important to understand that using the VaryByParam property results in more caching and not less. A different cached version of the Details view is created for each different version of the Id parameter.

You can set the VaryByParam property to the following values:

\* = Create a different cached version whenever a form or query string parameter varies.

none = Never create different cached versions

Semicolon list of parameters = Create different cached versions whenever any of the form or query string parameters in the list varies

**Creating a Cache Profile**

The <caching> section must appear within the <system.web> section of a web configuration file.

<system.web>

<caching>

<outputCacheSettings>

<outputCacheProfiles>

<add name="Cache1Hour" duration="3600" varyByParam="none"/>

</outputCacheProfiles>

</outputCacheSettings>

</caching>

</system.web>

**Applying cache profile to action**

public class ProfileController : Controller

{

[OutputCache(CacheProfile="Cache1Hour")]

public string Index()

{

return DateTime.Now.ToString("T");

}

}

protected void Page\_Load(object sender, EventArgs e)

{

Label1.Text = "Cache Refresh: " +

DateTime.Now.ToLongTimeString();

// Create a dependency connection to the database.

SqlDependency.Start(GetConnectionString());

using (SqlConnection connection =

new SqlConnection(GetConnectionString()))

{

using (SqlCommand command =

new SqlCommand(GetSQL(), connection))

{

SqlCacheDependency dependency =

new SqlCacheDependency(command);

// Refresh the cache after the number of minutes

// listed below if a change does not occur.

// This value could be stored in a configuration file.

int numberOfMinutes = 3;

DateTime expires =

DateTime.Now.AddMinutes(numberOfMinutes);

Response.Cache.SetExpires(expires);

Response.Cache.SetCacheability(HttpCacheability.Public);

Response.Cache.SetValidUntilExpires(true);

Response.AddCacheDependency(dependency);

connection.Open();

GridView1.DataSource = command.ExecuteReader();

GridView1.DataBind();

}

}

}