MVC stands for *model-view-controller*. MVC is a pattern for developing well architected, testable and easy to maintain. MVC-based applications.

* **M** odels: Represent the data of the application. uses validation logic to enforce business rules for that data.
* **V** iews: Template files that uses to dynamically generate HTML responses in application.
* **C** ontrollers: Classes that handle incoming browser requests, retrieve model data, and then specify view templates that return a response to the browser.

**@model directive : allows us to access the Model Object that the controller passed to the view (That's strongly typed view)**

**@RenderBody() :** RenderBody is a placeholder where all the view-specific pages are "wrapped" in the layout page.

**System.ComponentModel.DataAnnotations :**

**Model binders :** allows controller code to **remain cleanly separated** from the **dirtiness of interrogating** the request and its associated environment.

[Model binding](http://msdn.microsoft.com/en-us/library/dd410405(v=VS.90).aspx)  When action methods need data, and the incoming HTTP request carries the data and is embedded into POST-ed form values, and possibly the URL itself.

**View engine** is responsible for creating HTML from your views.

**Razor** view engine is an **advanced view engine** from Microsoft, packaged with **MVC 3. This is the default view engine in MVC 3 and MVC 4.**

**Passing Data from the Controller to the View.**

|  |  |  |
| --- | --- | --- |
| **ViewBag**is a **dynamic object** that provides a **convenient late-bound way** to pass information to a view. |  |  |

**Note\* : A best practice: A view template should never perform business logic or interact with a database directly. Instead, a view template should work only with the data that's provided to it by the controller. Maintaining this "separation of concerns" helps keep your code clean, testable and more maintainable.**

**Bundling and Minification**

Most of the current major browsers limit the each hostname to six [simultaneous connections](http://www.browserscope.org/?category=network) means that while six requests are being processed, additional requests for assets on a host will be queued by the browser. So, **Bundling,**

improves load time by reducing the number of requests to the server and reducing the size of requested assets (such as CSS and JavaScript.)

|  |  |
| --- | --- |
| **Bundling**   * Bundles multiple files into a single file. * We can create CSS, JavaScript and other bundles. * Fewer files means fewer HTTP requests and that can improve first page load performance. | **Minification**   * Performs code optimizations to scripts or css, such as * Removing unnecessary white space and comments and * Shortening variable names to one character. |

* **ViewResult** - Renders a specifed view to the response stream
* **PartialViewResult** - Renders a specifed partial view to the response stream
* **EmptyResult** - An empty response is returned
* **RedirectResult** - Performs an HTTP redirection to a specifed URL
* **RedirectToRouteResult** - Performs an HTTP redirection to a URL that is determined by the routing engine, based on given route data
* **JsonResult** - Serializes a given ViewData object to JSON format
* **JavaScriptResult** - Returns a piece of JavaScript code that can be executed on the client
* **ContentResult** - Writes content to the response stream without requiring a view
* **FileContentResult** - Returns a file to the client
* **FileStreamResult** - Returns a file to the client, which is provided by a Stream
* **FilePathResult** - Returns a file to the client

|  |  |  |  |
| --- | --- | --- | --- |
| **ViewData** is Dictionary object derived from ViewDataDictionary class.  ViewData is a property of **ControllerBase** class.  Used to pass data from controller to corresponding view.  life lies only during the current request.  If redirection occurs then it’s value becomes null.  Requires typecasting for getting data and check for null values to avoid error. | **ViewBag** is a dynamic property that takes advantage of the new dynamic features in C# 4.0.(so No typecasting Requires)  ViewBag is a property of **ControllerBase** class.  Used to pass data from controller to corresponding view.  It’s a wrapper around the ViewData.  It’s life also lies only during the current request.  Redirection occurs then it’s value becomes null. | **TempData** is a dictionary object that is derived from TempDataDictionary class and **stored in short lives session**.  TempData is a property of **ControllerBase** class.  TempData is used to pass data from **current request to subsequent request** (means redirecting from one page to another).  It’s life is very short and lies only till the target view is fully loaded.  It’s required typecasting for getting data and check for null values to avoid error.  It is used to store only one time messages like error messages, validation messages. To persist data with TempData | **Session I**n ASP.NET MVC, Session is a property of Controller class whose type is **HttpSessionStateBase**.  public HttpSessionStateBase Session { get; }  Session is also used to pass data within the ASP.NET MVC application and Unlike TempData,  It persists for its expiration time (by default session expiration time is 20 minutes but it can be increased).  **Session is valid for all requests, not for a single redirect.**  It’s also required typecasting for getting data and check for null values to avoid error. |

|  |  |
| --- | --- |
| **Strongly typed view :** View whichbinds to a **specific type of ViewModel.** | **loosely typed view :** Data binding done to a View is done using **ViewBag, TempData, or ViewData,** then that view. |

**Difference between textbox and TextboxFor in MVC?**

Ans : Main difference is that **Textbox is not strongly typed**. **TextboxFor** take a **lambda as a parameter that tell the helper** the with **element of the model** to use in a typed view.

**Cross-Site Scripting (XSS)?**

**Ans:** By using the Cross-Site Scripting (XSS) technique, **users executed malicious scripts** (also called payloads) unintentionally by clicking on untrusted links, and **hence, these scripts pass cookies information to attackers**.

**information can an attacker steal using XSS :**session id of the genuine user,

**Other methods : Attackers** ***store malicious scripts*** (also called payloads) are discussion forums, the comment section of websites, and other similar platforms. Whenever the **user navigates those pages, payloads got executed,** and ***the user's cookies information automatically sends to an attacker***.

**How can we prevent XSS? Ans:**XSS can be prevented by **1).**sanitizing user input to the application.**2).**"http only" attribute can also be used to prevent XSS.

**Cross-Site Tracing (XST)?** By using XST technique, attackers are able to steal cookies by bypassing "http only" attribute.

**How can it be prevented? :**can be prevented by disabling the TRACE method on the webserver.

**\_ViewStart.cshtml** file is a special file. **The code present in this file is going to be executed before the code in an individual view is executed**.

**Basic structure of Layout Page :**

**Styles.Render** : is used to **render a bundle of CSS** files defined within BundleConfig.cs files.

**Scripts.Render** is also used to **render a bundle of Script** files by rendering script tag(s) for the Script bundle.

|  |  |  |  |
| --- | --- | --- | --- |
| **Sections :** Allow us to specify a region of content within a layout. It expects one parameter which is the name of the section. If you don’t provide that, an exception will be thrown. A section in a layout page can be defined by using the following code.[Note\* : By default, sections are mandatory. To make them optional, just provides the second parameter value as false]  **Layout view uses two rendering methods :** RenderBody() and RenderSection().   |  |  |  | | --- | --- | --- | | **RenderBody()**   * can be used **only once in the layout view**. * Renders all the content of view which is not wrapped in named section. * can be configured as required or optional. * Synatax : @RenderBody() | **RenderSection().**   * **Can be called multiple time with different name.** * Renders the content of a view which is wrapped in named section. * **Ex:** **@section header { <h1>Header Content</h1>}** * You can render above defined section header on the content page as given below: * **@RenderSection("header")** | **RenderPage :**   * method also exists in the Layout page to render other page exists in your application. * **Can be called multiple time with different name.** * **A layout page can have multiple RenderPage method.** * Synatax : @RenderPage("~/Views/Shared/\_Header.cshtml") | |

**NonActionAttribute** is used when we want a public method in a controller but do not want to treat it as an action method. An action method is a public method in a controller that can be invoked using a URL. So, by default, if we 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.

|  |  |
| --- | --- |
| **NonActionAttribute** is used   1. when we want a public method in a controller but do not want to treat it as an action method. 2. If we have any public method in a controller then it can be invoked using a URL request. 3. **To restrict access to public methods in a controller, NonAction attribute can be used**. | **ChildActionOnly**   1. Only accessible by a child request. 2. It will **not respond to the URL requests.** 3. If an attempt is made, a runtime error will be thrown. 4. Child action methods **can be invoked by making child request** from a view using **Action()** and **RenderAction()** html helpers. |

|  |  |  |  |
| --- | --- | --- | --- |
| **Partial** when you have a model to send to the view and there will be a little bit of text that needs to be stored in a variable. | **RenderPartial** when you have a model to send to the view and there will be a lot of html that doesn't need to be stored in a variable. | **Action** when you do not have a model to send to the view and have a little bit of text to bring back that needs to be stored in a variable. | **RenderAction** when you do not have a model to send to the view and have a lot of html to bring back that doesn't need to be stored in a variable. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Partial** | **Render Partial** | **Action** | **Render Action** |
| **Returns** | Object (MvcHtmlString) | void | Object (MvcHtmlString) | void |
| **Result** | Result can be stored in a variable | Result will be directly written to the HTTP response stream | Result can be stored in a variable | Result will be directly written to the HTTP response stream |
| **Action** | Not Applicable | Not Applicable | Must Be Exists | Must Be Exists |
| **Syntax** | @Html.Partial("ViewPath", new Partial1() { PageName = "ABC Partial1" }) | @{ Html.RenderPartial("ViewPath", new Partial1() { PageName = "ABC Partial1" });  } | @Html.Action("ActionMethod") | @{ Html.RenderAction("RenderActionMethod"); } |

|  |  |
| --- | --- |
| View() : **Check for** \_ViewStart.cshtml then \_Layout.cshtml (if exists) | PartialView() : **NOT** *check for* \_ViewStart.cshtml then \_Layout.cshtml (if exists) |

**What can be stored in Web.config file?**

There are number of important settings that can be stored in the configuration file. Here are some of the most frequently used configurations, stored conveniently inside Web.config file..

a.      Database connections.  
b.      Session States  
c.      Error Handling (CustomError Page Settings.)  
d.      Security (Authentication modes)

**Status Codes details:**

a. 400: Bad Request  
b. 401: Unauthorized access  
c. 404:  Not Found