ASP.NET MVC Interview Questions

C# Curator Updated date Jan 28 2020

ASP.NET MVC Interview Questions and Answers

If you're planning to attend a .NET Interview, you may also be prepared for ASP.NET MVC interview questions. ASP.NET MVC is the framework used to build Web applications for .NET and C#. In this article, I list the top 50 ASP.NET MVC questions and their answers. The answers are code examples written by authors of C# Corner.

Question 1 - What is MVC (Model view controller)?

Answer

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 an MVC (Model View Controller) design:

- The Model represents the application core (for instance a list of database records).
- The View displays the data (the database records).
- The Controller handles the input (to the database records).

The MVC model also provides full control over HTML, CSS, and JavaScript.

The MVC model defines web applications with 3 logic layers,

- The business layer (Model logic)
- The display layer (View logic)
- The input control (Controller logic)

The Model is the part of the application that handles the logic for the application data.

Often model objects retrieve data (and store data) from a database.

The View is the part of the application that handles the display of the data.

Most often the views are created from the model data.

The Controller is the part of the application that handles user interaction.

Typically controllers read data from a view, control user input, and send input data to the model.

The MVC separation helps you manage complex applications because you can focus on one aspect a time. For example, you can focus on the view without depending on the business logic. It also makes it easier to test an application.

The MVC separation also simplifies group development. Different developers can work on the view, the controller logic, and the business logic in parallel.

Learn more about ASP.NET MVC here: Overview Of ASP.NET MVC

Question 2 - What are the advantages of MVC?

Answer - Benefits 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 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 the ASP.NET framework and hence most of the features of the ASP.NET like membership providers, roles, etc can still be used.

Here is a detailed article on Creating a Simple Application Using MVC 4.0.

Question 3 - Explain MVC application life cycle?

Answer

Any web application has two main execution steps, first understanding the request and depending on the type of the request sending out an appropriate response. MVC application life cycle is not different it has two main phases, first creating the request object and second sending our response to the browser.

Creating the request object,

The request object creation has four major steps. The following is a detailed explanation of the same.

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 rout table with routes collection. This filling of the route table happens 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 a Response object

This phase has two steps executing the action and finally sending the response as a result to the view.

Here is a complete article on ASP.Net MVC Life Cycle.

Question 4 - List out different return types of a controller action method?

Answer

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

The base type of all these result types is ActionResult.

1. *ViewResult (View)*

   This return type is used to return a webpage from an action method.

2. *PartialviewResult (Partialview)*

   This return type is used to send a part of a view that will be rendered in another view.

3. *RedirectResult (Redirect)*

   This return type is used to redirect to any other controller and action method depending on the URL.

4. *RedirectToRouteResult (RedirectToAction, RedirectToRoute)*

   This return type is used when we want to redirect to any other action method.

5. *ContentResult (Content)*

   This return type is used to return HTTP content type like text/plain as the result of the action.

6. *jsonResult (json)*

   This return type is used when we want to return a JSON message.

7. *javascriptResult (javascript)*

   This return type is used to return JavaScript code that will run in the browser.

8. *FileResult (File)*

   This return type is used to send binary output in response.

9. *EmptyResult*

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

Here is a complete article on Various Return Types From MVC Controller.

Question 5 - What are the Filters in MVC?

Answer

In MVC, controllers define action methods and these action methods generally have a one-to-one relationship with UI controls such as clicking a button or a link, etc. For example, in one of our previous examples, the UserController class contained methods UserAdd, UserDelete, etc.

But many times we would like to perform some action before or after a particular operation. For achieving this functionality, ASP.NET MVC provides a feature to add pre and post-action behaviors on the controller's action methods.

Types of Filters

ASP.NET MVC framework supports the following action filters,

- *Action Filters*

  Action filters are used to implement logic that gets executed before and after a controller action executes. We will look at Action Filters in detail in this chapter.

- *Authorization Filters*

  Authorization filters are used to implement authentication and authorization for controller actions.

- *Result Filters*

  Result filters contain logic that is executed before and after a view result is executed. For example, you might want to modify a view result right before the view is rendered to the browser.

- *Exception Filters*

  Exception filters are the last type of filter to run. You can use an exception filter to handle errors raised by either your controller actions or controller action results. You can also use exception filters to log errors.

Action filters are one of the most commonly used filters to perform additional data processing, or manipulating the return values or canceling the execution of an action or modifying the view structure at run time.

Here is a complete article on Understanding Filters in MVC.

Question 6 - What are Action Filters in MVC?

Answer - Action Filters

Action Filters are additional attributes that can be applied to either a controller section or the entire controller to modify the way in which action is executed. These attributes are special .NET classes derived from System.Attribute which can be attached to classes, methods, properties, and fields.

ASP.NET MVC provides the following action filters,

- *Output Cache*

  This action filter caches the output of a controller action for a specified amount of time.

- *Handle Error*

  This action filter handles errors raised when a controller action executes.

- *Authorize*

  This action filter enables you to restrict access to a particular user or role.

Now we will see the code example to apply these filters on an example controller ActionFilterDemoController.

(ActionFilterDemoController is just used as an example. You can use these filters on any of your controllers.)

Output Cache

Code Example

Specifies the return value to be cached for 10 seconds.

```
publicclassActionFilterDemoController: Controller
{
    [HttpGet]
    OutputCache(Duration = 10)]
publicstringIndex()
{
    returnDateTime.Now.ToString("T");

}
}
```

Learn more here: ASP.NET MVC with Action Filters

Question 7 - Explain what is routing in MVC? What are the three segments for routing important?

Answer

Routing is a mechanism to process the incoming URL that is more descriptive and gives the desired response. In this case, URL is not mapped to specific files or folder as was the case of earlier days web sites.

There are two types of routing (after the introduction of ASP.NET MVC 5).

1. Convention-based routing - to define this type of routing, we call MapRoute method and set its unique name, URL pattern and specify some default values.

2. Attribute-based routing - to define this type of routing, we specify the Route attribute in the action method of the controller.

Routing is the URL pattern that is mapped together to a handler,routing is responsible for incoming browser request for particular MVC controller. In other ways let us say routing help you to define a URL structure and map the URL with controller. There are three segments for routing that are important,

1. ControllerName
2. ActionMethodName
3. Parammeter

Code Example

ControllerName/ActionMethodName/{ParamerName} and also route map coding written in a Global.asax file.

Learn more here - Routing in MVC.

Question 8 - What is Route in MVC? What is Default Route in MVC?

Answer

A route is a URL pattern that is mapped to a handler. The handler can be a physical file, such as a .aspx file in a Web Forms application. A handler can also be a class that processes the request, such as a controller in an MVC application. To define a route, you create an instance of the Route class by specifying the URL pattern, the handler, and optionally a name for the route.

You add the route to the application by adding the Route object to the static Routes property of the RouteTable class. The Routesproperty is a RouteCollection object that stores all the routes for the application.

You typically do not have to write code to add routes in an MVC application. Visual Studio project templates for MVC include preconfigured URL routes. These are defined in the MVC Application class, which is defined in the Global.asax file.

Route definition	Example of matching URL
{controller}/{action}/{id}	/Products/show/beverages
{table}/Details.aspx	/Products/Details.aspx
blog/{action}/{entry}	/blog/show/123
{reporttype}/{year}/{month}/{day}	/sales/2008/1/5
{locale}/{action}	/US/show
{language}-{country}/{action}	/en-US/show

Default Route

The default ASP.NET MVC project templates add a generic route that uses the following URL convention to break the URL for a given request into three named segments.

URL: "{controller}/{action}/{id}"

This route pattern is registered via a call to the MapRoute() extension method of RouteCollection.

Question 9 - Mention what is the difference between Temp data, View, and View Bag?

Answer

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

ViewData

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

ViewBag

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

TempData

1. TempData is derived from TempDataDictionary class
2. TempData is used to pass data from the current request to the next request
3. 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
4. It requires typecasting for complex data types and checks for null values to avoid an error. Generally, it is used to store only one time messages like the error messages and validation messages

Learn more here: Difference Between ViewData, ViewBag, and TempData

Question 10 - What is Partial View in MVC?

Answer

A partial view is a chunk of HTML that can be safely inserted into an existing DOM. Most commonly, partial views are used to componentize Razor views and make them easier to build and update. Partial views can also be returned directly from controller methods. In this case, the browser still receives text/html content but not necessarily HTML content that makes up an entire page. As a result, if a URL that returns a partial view is directly invoked from the address bar of a browser, an incomplete page may be displayed. This may be something like a page that misses title, script and style sheets. However, when the same URL is invoked via a script, and the response is used to insert HTML within the existing DOM, then the net effect for the end-user may be much

better and nicer.

Partial view is a reusable view (like a user control) which can be embedded inside another view. For example, let's say all the pages of your site have a standard structure with left menu, header, and footer as in the following image,

Learn more here - Partial View in MVC

Question 11 - Explain what is the difference between View and Partial View?

Answer

View

- It contains the layout page.
- Before any view is rendered, viewstart page is rendered.
- A view might have markup tags like body, HTML, head, title, meta etc.
- The view is not lightweight as compare to Partial View.

Partial View

- It does not contain the layout page.
- Partial view does not verify for a viewstart.cshtml.We cannot put common code for a partial view within the viewStart.cshtml.page.
- Partial view is designed specially to render within the view and just because of that it does not consist any mark up.
- We can pass a regular view to the RenderPartial method.

Learn more here - Partial View in MVC

Question 12 - What are HTML helpers in MVC?

Answer

With MVC, HTML helpers are much like traditional ASP.NET Web Form controls.

Just like web form controls in ASP.NET, HTML helpers are used to modify HTML. But HTML helpers are more lightweight. Unlike Web Form controls, an HTML helper does not have an event model and a view state.

In most cases, an HTML helper is just a method that returns a string.

With MVC, you can create your own helpers, or use the built in HTML helpers.

Standard HTML Helpers

HTML Links

The easiest way to render an HTML link in is to use the HTML.ActionLink() helper.With MVC, the Html.ActionLink() does not link to a view. It creates a link to a controller action.

ASP Syntax

```
01.  <%=Html.ActionLink("About this Website", "About")%>
```

The first parameter is the link text, and the second parameter is the name of the controller action.

The Html.ActionLink() helper above, outputs the following HTML:

```
01.  <a href="/Home/About">About this Website</a>
```

The Html.ActionLink() helper has several properties:

- Property Description.
- .linkText The link text (label).
- .actionName The target action.
- .routeValues The values passed to the action.
- .controllerName The target controller.
- .htmlAttributes The set of attributes to the link.
- .protocol The link protocol.
- .hostname The host name for the link.
- .fragment The anchor target for the link.

HTML Form Elements

There following HTML helpers can be used to render (modify and output) HTML form elements:

- BeginForm()
- EndForm()
- TextArea()
- TextBox()
- CheckBox()
- RadioButton()
- ListBox()
- DropDownList()
- Hidden()
- Password()

ASP.NET Syntax C#

```
<%= Html.ValidationSummary("Create was unsuccessful. Please correct the errors and try a
    <% using (Html.BeginForm()){%>
        <p>
            <label for="FirstName">First Name:</label>
            <%= Html.TextBox("FirstName") %>
                <%= Html.ValidationMessage("FirstName", "*") %>
        </p>
        <p>
            <label for="LastName">Last Name:</label>
            <%= Html.TextBox("LastName") %>
                <%= Html.ValidationMessage("LastName", "*") %>
        </p>
        <p>
            <label for="Password">Password:</label>
            <%= Html.Password("Password") %>
                <%= Html.ValidationMessage("Password", "*") %>
        </p>
        <p>
            <label for="Password">Confirm Password:</label>
            <%= Html.Password("ConfirmPassword") %>
                <%= Html.ValidationMessage("ConfirmPassword", "*") %>
        </p>
        <p>
            <label for="Profile">Profile:</label>
            <%= Html.TextArea("Profile", new {cols=60, rows=10})%>
        </p>
        <p>
            <%= Html.CheckBox("ReceiveNewsletter") %>
                <label for="ReceiveNewsletter" style="display:inline">Receive Newsletter
</label>
        </p>
        <p>
            <input type="submit" value="Register" />
        </p>
        <%}%>
```

Learn more here - HTML Helpers in MVC: Part 1

Question 13 - Explain attribute based routing in MVC?

Answer

In ASP.NET MVC 5.0 we have a new attribute route,cBy 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".

Hide Copy Code

```
public class HomeController: Controller
{
    [Route("Users/about")]
    publicActionResultGotoAbout()
    {
        return View();
    }
}
```

Learn more here - Attribute Based Routing in ASP.Net MVC 5

Question 14 - What is TempData in MVC?

Answer

TempData is a dictionary object to store data temporarily. It is a TempDataDictionary class type and instance property of the Controller base class.

TempData is able to keep data for the duration of a HTP request, in other words it can keep live data between two consecutive HTTP requests. It will help us to pass the state between action methods. TempData only works with the current and subsequent request. TempData uses a session variable to store the data. TempData Requires type casting when used to retrieve data.

TempDataDictionary is inherited from the IDictionary<string, object>, ICollection<KeyValuePair<string, object>>, IEnumerable<KeyValuePair<string, object>> and IEnumerable interfaces.

Example

```
public ActionResult FirstRequest()
{
    List < string > TempDataTest = new List < string > ();
    TempDataTest.Add("Tejas");
    TempDataTest.Add("Jignesh");
    TempDataTest.Add("Rakesh");
    TempData["EmpName"] = TempDataTest;
    return View();
}
public ActionResult ConsecutiveRequest()
{
    List < string > modelData = TempData["EmpName"] as List < string > ;
    TempData.Keep();
    return View(modelData);
}
```

Learn more here - All About the TempData in MVC

Question 15 - What is Razor in MVC?

Answer

ASP.NET MVC has always supported the concept of "view engines" - which are the pluggable modules that implement different template syntax options. The "default" view engine for ASP.NET MVC uses the same .aspx/.ascx/. master file templates as ASP.NET Web Forms. Other popular ASP.NET MVC view engines are Spart&Nhaml.

MVC 3 has introduced a new view engine called Razor.

Why is Razor?

1. Compact & Expressive.
2. Razor minimizes the number of characters and keystrokes required in a file, and enables a fast coding workflow. Unlike most template syntaxes, you do not need to interrupt your coding to explicitly denote server blocks within your HTML. The parser is smart enough to infer this from your code. This enables a really compact and expressive syntax which is clean, fast and fun to type.
3. Easy to Learn: Razor is easy to learn and enables you to quickly be productive with a minimum of effort. We can use all your existing language and HTML skills.
4. Works with any Text Editor: Razor doesn't require a specific tool and enables you to be productive in any plain old text editor (notepad works great).
5. Has great Intellisense:
6. Unit Testable: The new view engine implementation will support the ability to unit test views (without requiring a controller or web-server, and can be hosted in any unit test project - no special app-domain required).

Learn more here - Brief Introduction to MVC3

Question 16 - Differences between Razor and ASPX View Engine in MVC?

Answer - Razor View Engine VS ASPX View Engine

Razor View Engine	ASPX View Engine (Web form view engine)
The namespace used by the Razor View Engine is System.Web.Razor	The namespace used by the ASPX View Engine is System.Web.Mvc.WebFormViewEngine
The file extensions used by the Razor View Engine are different from a web form view engine. It uses cshtml with C# and vbhtml with vb for views, partial view, templates and layout pages.	The file extensions used by the Web Form View Engines are like ASP.Net web forms. It uses the ASPX extension to view the aspc extension for partial views or User Controls or templates and master extensions for layout/master pages.
The Razor View Engine is an advanced view engine that was introduced with MVC 3.0. This is not a new language but it is markup.	A web form view engine is the default view engine and available from the beginning of MVC
Razor has a syntax that is very compact and helps us to reduce typing.	The web form view engine has syntax that is the same as an ASP.Net forms application.
The Razor View Engine uses @ to render server-side content.	The ASPX/web form view engine uses "<%= %>" or "<%: %>" to render server-side content.
By default all text from an @ expression is HTML encoded.	There is a different syntax ("<%: %>") to make text HTML encoded.
Razor does not require the code block to be closed, the Razor View Engine parses itself and it is able to decide at runtime which is a content element and which is a code element.	A web form view engine requires the code block to be closed properly otherwise it throws a runtime exception.
The Razor View Engine prevents Cross-Site Scripting (XSS) attacks by encoding the script or HTML tags before rendering to the view.	A web form View engine does not prevent Cross-Site Scripting (XSS) attacks.
The Razor Engine supports Test Driven Development (TDD).	Web Form view engine does not support Test Driven Development (TDD) because it depends on the System.Web.UI.Page class to make the testing complex.
Razor uses "@* â€¦ *@" for multiline comments.	The ASPX View Engine uses "<!--...-->" for markup and "/* â€¦ */" for C# code.
There are only three transition characters with the Razor View Engine.	There are only three transition characters with the Razor View Engine.

The Razor View Engine is a bit slower than the ASPX View Engine.

Conclusion

Razor provides a new view engine with a streamlined code for focused templating. Razor's syntax is very compact and improves the readability of the markup and code. By default, MVC supports ASPX (web forms) and Razor View Engine. MVC also supports third-party view engines like Spark, Nhaml, NDjango, SharpDOM and so on. ASP.NET MVC is open source.

Learn more here - ASPX View Engine VS Razor View Engine

Question 17 - What are the Main Razor Syntax Rules?

Answer

- Razor code blocks are enclosed in @{ ... }
- Inline expressions (variables and functions) start with @
- Code statements end with semicolon
- Variables are declared with the var keyword
- Strings are enclosed with quotation marks
- C# code is case sensitive
- C# files have the extension .cshtml

C# Example

```
<!-- Single statement block -->
@ {
    varmyMessage = "Hello World";
}
<!-- Inline expression or variable -->
< p > The value of myMessage is: @myMessage < /p>
    <!-- Multi-statement block -->
@ {
    var greeting = "Welcome to our site!";
    varweekDay = DateTime.Now.DayOfWeek;
    vargreetingMessage = greeting + " Here in Huston it is: " + weekDay;
} < p > The greeting is: @greetingMessage < /p>
```

Learn more here - Introduction to Microsoft ASP.NET MVC 3 Razor View Engine

Question18 - How do you implement Forms authentication in MVC?

Answer

Authentication is giving access to the user for a specific service by verifying his/her identity using his/her credentials like username and password or email and password. It assures that the correct user is authenticated or logged in for a specific service and the right service has been provided to the specific user based on their role that is nothing but authorization.

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=
        </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.

Learn more here - Form Authentication in MVC 5: Part 1

Question 19 - Explain Areas in MVC?

Answer

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 theGlobal.asax file that can automatically find the area routes in the AreaRegistration file.

AreaRegistration.RegisterAllAreas();

Benefits of Area in MVC

1. Allows us to organize models, views and controllers into separate functional sections of the application, such as administration, billing, customer support and much more.
2. Easy to integrate with other Areas created by another.
3. Easy for unit testing.

Learn more here - What Are Areas in ASP.Net MVC - Part 6

Question 20 - Explain the need of display mode in MVC?

Answer

DisplayModes give you another level of flexibility on top of the default capabilities we saw in the last section. DisplayModes can also be used along with the previous feature so we will simply build off of the site we just created.

Using display modes involves in 2 steps

1. We should register Display Mode with a suffix for particular browser using “DefaultDisplayMode”e class inApplication_Start() method in the Global.asax file.
2. View name for particular browser should be appended with suffix mentioned in first step.

1. Desktop browsers (without any suffix. e.g.: Index.cshtml, _Layout.cshtml).
2. Mobile browsers (with a suffix “Mobile”. e.g.: Index.Mobile.cshtml,Layout.Mobile.cshtml)

   If you want design different pages for different mobile device browsers (any different browsers) and render them depending on the browser requesting. To handle these requests you can register custom display modes. We can do that using DisplayModeProvider.Instance.Modes.Insert(int index, IDisplayMode item) method.

Learn more here - Display Mode Provider in MVC 5 Application

Question 21 - Explain the concept of MVC Scaffolding?

Answer

ASP.NET Scaffolding is a code generation framework for ASP.NET Web applications. Visual Studio 2013 includes pre-installed code generators for MVC and Web API projects. You add scaffolding to your project when you want to quickly add code that interacts with data models. Using scaffolding can reduce the amount of time to develop standard data operations in your project.

Scaffolding consists of page templates, entity page templates, field page templates, and filter templates. These templates are called Scaffold templates and allow you to quickly build a functional data-driven Website.

Scaffolding Templates

Create

It creates a View that helps in creating a new record for the Model. It automatically generates a label and input field for each property in the Model.

Delete

It creates a list of records from the model collection along with the delete link with delete record.

Details

It generates a view that displays the label and an input field of the each property of the Model in the MVC framework.

Edit

It creates a View with a form that helps in editing the current Model. It also generates a form with label and field for each property of the model.

List

It generally creates a View with the help of a HTML table that lists the Models from the Model Collection. It also generates a HTML table column for each property of the Model.

Learn more here - Terminologies in MVC: Part 3 (Scaffolding)

Question 22 - What is Route Constraints in MVC?

Answer

Routing is a great feature of MVC, it provides a REST based URL that is very easy to remember and improves page ranking in search engines.

This article is not an introduction to Routing in MVC, but we will learn a few features of routing and by implementing them we can develop a very flexible and user-friendly application. So, let's start without wasting valuable time.

Add constraint to URL

This is very necessary for when we want to add a specific constraint to our URL. Say, for example we want a URL.

So, we want to set some constraint string after our host name. Fine, let's see how to implement it.

It's very simple to implement, just open the RouteConfig.cs file and you will find the routing definition in that. And modify the routing entry as in the following. We will see that we have added “abc” before.

Controller name, now when we browse we need to specify the string in the URL, as in the following:

Learn more here - Route Constraints in MVC

Question 23 - What is Razor View Engine in MVC?

Answer

ASP.NET MVC has always supported the concept of "view engines" that are the pluggable modules that implement various template syntax options. The "default" view engine for ASP.NET MVC uses the same .aspx/.ascx/.master file templates as ASP.NET Web Forms. In this article I go through the Razor View Engine to create a view of an application. "Razor" was in development beginning in June 2010 and was released for Microsoft Visual Studio in January 2011.

Razor is not a new programming language itself, but uses C# syntax for embedding code in a page without the ASP.NET delimiters: <%= %>. It is a simple-syntax view engine and was released as part of ASP.NET MVC 3. The Razor file extension is "cshtml" for the C# language. It supports TDD (Test Driven Development) because it does not depend on the System.Web.UI.Page class.

Learn more here - Getting Started with Razor View Engine in MVC 3

Question 24 - What is Output Caching in MVC?

Answer

The main purpose of using Output Caching is to dramatically improve the performance of an ASP.NET MVC Application. It enables us to cache the content returned by any controller method so that the same content does not need to be generated each time the same controller method is invoked. Output Caching has huge advantages, such as it reduces server round trips, reduces database server round trips, reduces network traffic etc.

Keep the following in mind,

- Avoid caching contents that are unique per user.
- Avoid caching contents that are accessed rarely.
- Use caching for contents that are accessed frequently.

Let's take an example. My MVC application displays a list of database records on the view page so by default each time the user invokes the controller method to see records, the application loops through the entire process and executes the database query. And this can actually decrease the application performance. So, we can advantage of the "Output Caching" that avoids executing database queries each time the user invokes the controller method. Here the view page is retrieved from the cache instead of invoking the controller method and doing redundant work.

Cached Content Locations

In the above paragraph I said, in Output Caching the view page is retrieved from the cache, so where is the content cached/stored?

Please note, there is no guarantee that content will be cached for the amount of time that we specify. When memory resources become low, the cache starts evicting content automatically.

OutputCache label has a "Location" attribute and it is fully controllable. Its default value is "Any", however there are the following locations available; as of now, we can use any one.

1. Any
2. Client
3. Downstream
4. Server
5. None
6. ServerAndClient

With "Any", the output cache is stored on the server where the request was processed. The recommended store cache is always on the server very carefully. You will learn about some security related tips in the following "Don't use Output Cache".

Learn more here - Output Caching in MVC

Question 25 - What is Bundling and Minification in MVC?

Answer

Bundling and minification are two new techniques introduced to improve request load time. It 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

It lets us combine multiple JavaScript (.js) files or multiple cascading style sheet (.css) files so that they can be downloaded as a unit, rather than making individual HTTP requests.

Minification

It squeezes out whitespace and performs other types of compression to make the downloaded files as small as possible. At runtime, the process identifies the user agent, for example IE, Mozilla, etc. and then removes whatever is specific to Mozilla when the request comes from IE.

Learn more here - Bundling and Minification in Visual Studio 2012 or Migrate Existing Website

Question 26 - What is Validation Summary in MVC?

Answer

The ValidationSummary helper method generates an unordered list (ul element) of validation messages that are in the ModelStateDictionary object.

The ValidationSummary can be used to display all the error messages for all the fields. It can also be used to display custom error messages. The following figure shows how ValidationSummary displays the error messages.

ValidationSummary() Signature

MvcHtmlStringValidateMessage(bool excludePropertyErrors, string message, object htmlAttributes)

Display field level error messages using ValidationSummary

By default, ValidationSummary filters out field level error messages. If you want to display field level error messages as a summary then specify *excludePropertyErrors = false.*

Example - ValidationSummary to display field errors

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

So now, the following Edit view will display error messages as a summary at the top. Please make sure that you don't have a ValidationMessageFor method for each of the fields.

Learn more here - Understanding Validation In MVC - Part 4

Question 27 - What is Database First Approach in MVC using Entity Framework?

Answer

Database First Approach is an alternative to the Code First and Model First approaches to the Entity Data Model which creates model codes (classes,properties, DbContextetc) from the database in the project and that classes behaves as the link between database and controller.

There are the following approach which is used to connect with database to application.

- Database First
- Model First
- Code First

Database first is nothing but only a approach to create web application where database is available first and can interact with the database. In this database, database is created first and after that we manage the code. The Entity Framework is able to generate a business model based on the tables and columns in a relational database.

Learn more here - Database First Approach With ASP.NET MVC Step By Step Part 1

Question 28 - What are the Folders in MVC application solutions?

Answer - Understanding the folders

When you create a project a folder structure gets created by default under the name of your project which can be seen in solution explorer. Below i will give you a brief explanation of what these folders are for.

Model

This folder contains classes that is used to provide data. These classes can contain data that is retrived from the database or data inserted in the form by the user to update the database.

Controllers

These are the classes which will perform the action invoked by the user. These classes contains methods known as "Actions" which responds to the user action accordingly.

Views

These are simple pages which uses the model class data to populate the HTML controls and renders it to the client browser.

App_Start

Contains Classes such as FilterConfig, RoutesConfig, WebApiConfig. As of now we need to understand the RouteConfig class. This class contains the default format of the url that should be supplied in the browser to navigate to a specified page.

Question 29 - What is difference between MVC and Web Forms?

Answer - ASP.Net MVC / Web Forms difference

The following are some difference.

ASP.Net MVC	ASP.Net Web Forms
View and logic are separate, it has separation of concerns theory. MVC 3 onwards has .aspx page as .cshtml.	No separation of concerns; Views are tightly coupled with logic (.aspx.cs /.vb file).
Introduced concept of routing for route based URL. Routing is declared in Global.asax for example.	File-based routing .Redirection is based on pages.
Support Razor syntax as well as .aspx	Support web forms syntax only.
State management handled via Tempdata, ViewBag, and View Data. Since the controller and view are not dependent and also since there is no view state concept in ASP.NET, MVC keeps the pages lightweight.	State management handled via View State. Large viewstate, in other words increase in page size.
Partial Views	User Controls
HTML Helpers	Server Controls
Multiple pages can have the same controller to satisfy their requirements. A controller may have multiple Actions (method name inside the controller class).	Each page has its own code, in other words direct dependency on code. For example Sachin.aspx is dependent on Sachin.aspx.cs (code behind) file.
Unit Testing is quite easier than ASP.Net Web forms Since a web form and code are separate files.	Direct dependency, tight coupling raises issues in testing.
layouts	Master pages

Here are more Similarities and Dissimilarities Between MVC and Web Forms.

Question 30 - What are the methods of handling an Error in MVC?

Answer

Exception handling may be required in any application, whether it is a web application or a Windows Forms application.

ASP.Net MVC has an attribute called "HandleError" that provides built-in exception filters. The HandleError attribute in ASP.NET MVC can be applied over the action method as well as Controller or at the global level. The HandleError attribute is the default implementation of IExceptionFilter. When we create a MVC application, the HandleError attribute is added within the Global.asax.cs file and registered in the Application_Start event.

```
public static void RegisterGlobalFilters(GlobalFilterCollection filters)
{
    filters.Add(new HandleErrorAttribute());
}
protected void Application_Start()
{
    AreaRegistration.RegisterAllAreas();
    RegisterGlobalFilters(GlobalFilters.Filters);
    RegisterRoutes(RouteTable.Routes);
}
```

Important properties of HandleError attribute

The HandleError Error attribute has a couple for properties that are very useful in handling the exception.

ExceptionType

Type of exception to be catch. If this property is not specified then the HandleError filter handles all exceptions.

View

Name of the view page for displaying the exception information.

Master

Master View for displaying the exception.

Order

Order in which the action filters are executed. The Order property has an integer value and it specifies the priority from 1 to any positive integer value. 1 means highest priority and the greater the value of the integer is, the lower is the priority of the filter.

AllowMultiple

It indicates whether more than one instance of the error filter attribute can be specified.

Example

```
[HandleError(View = "Error")]
public class HomeController: Controller
{
    public ActionResult Index()
    {
        ViewBag.Message = "Welcome to ASP.NET MVC!";
        int u = Convert.ToInt32(""); // Error line
        return View();
    }
}
```

HandleError Attribute at Action Method Level,

```
[HandleError(View = "Error")]
public ActionResult Index()
{
    ViewBag.Message = "Welcome to ASP.NET MVC!";
    int u = Convert.ToInt32(""); // Error line
    return View();
}
```

Here are more details on Exception or Error Handling in ASP.Net MVC Using HandleError Attribute

Question 31 - How can we pass the data From Controller To View In MVC?

Answer

There are three options in Model View Controller (MVC) for passing data from controller to view. This article attempts to explain

the differences among ViewData, ViewBag and TempData with examples. ViewData and ViewBag are similar and TempData performs additional responsibility. The following are the key points on those three objects.

ViewData

- The ViewData is used to move data from controller to view.
- The ViewData is a dictionary of objects that are derived from the "ViewDataDictionary" class and it will be accessible using strings as keys.
- ViewData contains a null value when redirection occurs.
- ViewData requires typecasting for complex data types.

ViewBag

- ViewBag is just a dynamic wrapper around ViewData and exists only in ASP.NET MVC 3. ViewBag is a dynamic property that takes advantage of the new dynamic features in C# 4.0.
- ViewBag doesn't require typecasting for complex data types.
- ViewBag also contain a null value when redirection occurs.

TempData

- ViewData moves data from controller to view.
- Use TempData when you need data to be available for the next request, only. In the next request, it will be there but will be gone after that.
- TempData is used to pass data from the current request to the subsequent request, in other words in case of redirection. That means the value of TempData will not be null.

Learn more here - Various Ways to Pass Data From Controller to View in MVC 4

Question 32 - What is Scaffolding in MVC?

Answer

Scaffolding is a code generation framework for ASP.NET Web applications. Visual Studio 2013 includes pre-installed code generators for MVC and Web API projects. You add scaffolding to your project when you want to quickly add code that interacts with data models. Using scaffolding can reduce the amount of time to develop standard data operations in your project.

Prerequisites

To use ASP.NET Scaffolding, you must have,

- Microsoft Visual Studio 2013
- Web Developer Tools (part of default Visual Studio 2013 installation)
- ASP.NET Web Frameworks and Tools 2013 (part of default Visual Studio 2013 installation)

What are the Advantages of using Scaffolding ?

- Minimal or no code to create a data-driven Web applications.
- Quick development time.
- Pages that are fully functional and include display, insert, edit, delete, sorting, and paging functionalities.
- Built-in data validation that is based on the database schema.
- Filters that are created for each foreign key or Boolean fields.

Learn more here - Scaffolding In MVC 5

Question 33 - What is ViewStart?

Answer

Razor View Engine introduced a new layout named _ViewStart which is applied on all view automatically. Razor View Engine firstly executes the _ViewStart and then start rendering the other view and merges them.

Example of Viewstart

```
@ {
    Layout = "~/Views/Shared/_v1.cshtml";
} < !DOCTYPE html >
    < html >
    < head >
    < meta name = "viewport"
content = "width=device-width" / >
    < title > ViewStart < /title> < /head> < body >
    < /body> < /html>
```

Learn more here - Viewstart Page in ASP.NET MVC 3

Question 34 - What is JsonResultType in MVC?

Answer

Action methods on controllers return JsonResult (JavaScript Object Notation result) that can be used in an AJAX application. This class is inherited from the "ActionResult" abstract class. Here Json is provided one argument which must be serializable. The JSON result object that serializes the specified object to JSON format.

```
public JsonResult JsonResultTest()
{
    return Json("Hello My Friend!");
}
```

Learn more here - ActionResult Return Type in MVC 3.0

Question 35 - What is TempData?

Answer - Tempdata

- TempData is a dictionary object derived from the TempDataDictionary class.
- TempData is used to pass data from the current request to a subsequent request, in other words in the case of redirection.
- The life of a TempData is very short and it retains its value for a short period of time.
- It requires typecasting for complex data type as I've used in my example:
- @foreach (var item in (List<MVCSample.Models.EmpRegistration>)TempData["EmployeeRegistration"])
- You can retain its value using the Keep method for subsequent requests.

Question 36 - How to use ViewBag?

Answer

ViewBag is dynamic property that takes advantage of new dynamic features in C# 4.0. It's also used to pass data from a controller to a view. In short, The ViewBag property is simply a wrapper around the ViewData that exposes the ViewData dictionary as a dynamic object. Now create an action method "StudentSummary" in the "DisplayDataController" controller that stores a Student class object in ViewBag.

```
public ActionResult StudentSummary()
{
    var student = new Student()
    {
        Name = "Sandeep Singh Shekhawat",
            Age = 24,
            City = "Jaipur"
    };
    ViewBag.Student = student;
    return View();
}
```

Thereafter create a view StudentSummary ("StudentSummary.cshtml") that shows student object data. ViewBag does not require typecasting for complex data type so you can directly access the data from ViewBag.

```
@ {
    ViewBag.Title = "Student Summary";
    var student = ViewBag.Student;
}
< table >
    < tr >
    < th > Name < /th> < th > Age < /th> < th > City < /th> < /tr> < tr >
    < td > @student.Name < /td> < td > @student.Age < /td> < td > @student.City < /td> <
< /table>
```

Here we used one more thing, "ViewBag.Title", that shows the title of the page.

Learn more here - Displaying Data on View From Controller

Question 37 - What are the Difference between ViewBag&ViewData?

Answer - Difference between ViewBag&ViewData?

- ViewData is a dictionary of objects that is derived from ViewDataDictionary class and accessible using strings as keys.
- ViewBag is a dynamic property that takes advantage of the new dynamic features in C# 4.0.
- ViewData requires typecasting for complex data type and check for null values to avoid error.
- ViewBag doesn't require typecasting for complex data type.
- Calling of ViewBag is:

  ViewBag.Name = "Yogesh";

  Calling of ViewDatais :

  ViewData["Name"] = "yogesh";

Learn more here - Difference Between ViewBag & ViewData in MVC

Question 38 - What is Data Annotation Validator Attributes in MVC?

Answer - Using the Data Annotation Validator Attributes

DataAnnotation plays a vital role in added validation to properties while designing the model itself. This validation can be added for both the client side and the server side.

You understand that decorating the properties in a model with an Attribute can make that property eligible for Validation.

Some of the DataAnnotation used for validation are given below,

1. *Required*

   Specify a property as required.

   1. [Required(ErrorMessage="CustomerName is mandatory")]

2. *RegularExpression*

   Specifies the regular expression to validate the value of the property.

   1. [RegularExpression("[a-z]", ErrorMessage = "Invalid character")]

3. *Range*

   Specifies the Range of values between which the property values are checked.

   1. [Range(1000,10000,ErrorMessage="Range should be between 1k & 10k")]

4. *StringLength*

   Specifies the Min & Max length for a string property.

   1. [StringLength(50, MinimumLength = 5, ErrorMessage = "Minimum char is 5 and maximum char is 10")]

5. *MaxLength*

   Specifies the Max length for the property value.

   1. [MaxLength(10,ErrorMessage="Customer Code is exceeding")]

6. *MinLength*

   It is used to check for minimum length.

   1. [MinLength(5, ErrorMessage = "Customer Code is too small")]

Learn more here - Adding Custom Validation in MVC

Question 39 - How can we done Custom Error Page in MVC?

Answer

The HandleErrorAttribute allows you to use a custom page for this error. First you need to update your web.config file to allow your application to handle custom errors.

```
<system.web>
    <customErrors mode="On">
</system.web>
```

Then, your action method needs to be marked with the atttribute.

```
[HandleError]
public class HomeController: Controller
{
    [HandleError]
    publicActionResultThrowException()
    {
        throw new ApplicationException();
    }
}
```

By calling the ThrowException action, this would then redirect the user to the default error page. In our case though, we want to use a custom error page and redirect the user there instead.So, let's create our new custom view page.

Next, we simply need to update the HandleErrorAttribute on the action method.

```
[HandleError]
public class HomeController: Controller
{
    [HandleError(View = "CustomErrorView")]
    publicActionResultThrowException()
    {
        throw new ApplicationException();
    }
}
```

Learn more here - Custom Error Page in ASP.NET MVC

Question 40 - Server Side Validation in MVC?

Answer

The ASP.NET MVC Framework validates any data passed to the controller action that is executing, It populates a ModelState object with any validation failures that it finds and passes that object to the controller. Then the controller actions can query the ModelState to discover whether the request is valid and react accordingly.

I will use two approaches in this article to validate a model data. One is to manually add an error to the ModelState object and another uses the Data Annotation API to validate the model data.

Approach 1 - Manually Add Error to ModelState object

I create a User class under the Models folder. The User class has two properties "Name" and "Email". The "Name" field has required field validations while the "Email" field has Email validation. So let's see the procedure to implement the validation. Create the User Model as in the following,

```
01. namespace ServerValidation.Models
02. {
03.     public class User
04.     {
05.         public string Name
06.       {
07.             get;
08.             set;
09.         }
10.         public string Email
11.         {
12.             get;
13.             set;
14.         }
15.     }
16. }
```

After that I create a controller action in User Controller (UserController.cs under Controllers folder). That action method has logic for the required validation for Name and Email validation on the Email field. I add an error message on ModelState with a key and that message will be shown on the view whenever the data is not to be validated in the model.

```
using System.Text.RegularExpressions;
using System.Web.Mvc;
namespace ServerValidation.Controllers
{
    public class UserController: Controller
    {
        public ActionResult Index()
        {
                return View();
            }
            [HttpPost]
        public ActionResult Index(ServerValidation.Models.User model)
        {

            if (string.IsNullOrEmpty(model.Name))
            {
                ModelState.AddModelError("Name", "Name is required");
            }
            if (!string.IsNullOrEmpty(model.Email))
            {
                string emailRegex = @ "^([a-zA-Z0-9 \-\.]+)@((\[[0-9]{1,3}" +
                    @ "\.[0-9]{1,3}\.[0-9]{1,3}\.)|(([a-zA-Z0-9\-]+\" +
                @ ".)+))([a-zA-Z]{2,4}|[0-9]{1,3})(\]?)$";
                Regex re = new Regex(emailRegex);
                if (!re.IsMatch(model.Email))
                {
                    ModelState.AddModelError("Email", "Email is not valid");
                }
            } else {
                ModelState.AddModelError("Email", "Email is required");
            }
            if (ModelState.IsValid)
            {
                ViewBag.Name = model.Name;
                ViewBag.Email = model.Email;
            }
            return View(model);
        }
    }
}
```

Thereafter I create a view (Index.cshtml) for the user input under the User folder.

```
@model ServerValidation.Models.User
@ {
    ViewBag.Title = "Index";
}
@using(Html.BeginForm())
{
    if (@ViewData.ModelState.IsValid)
        {
        if (@ViewBag.Name != null)
    { < b >
                Name: @ViewBag.Name < br / >
                Email: @ViewBag.Email < /b>
        }
    } < fieldset >
        < legend > User < /legend>  < div class = "editor-label" >
        @Html.LabelFor(model => model.Name) < /div> < div class = "editor-field" >
        @Html.EditorFor(model => model.Name)
    @if(!ViewData.ModelState.IsValid)
{
< span class = "field-validation-
error" > @ViewData.ModelState["Name"].Errors[0].ErrorMessage < /span>

}
< /div>  < div class = "editor-label" >

        @Html.LabelFor(model => model.Email) < /div> < div class = "editor-field" >
        @Html.EditorFor(model => model.Email)
    @if(!ViewData.ModelState.IsValid)
 {
< span class = "field-validation-
error" > @ViewData.ModelState["Email"].Errors[0].ErrorMessage < /span>
  }
 < /div> < p >
        < input type = "submit"
    value = "Create" / >
        < /p> < /fieldset>
}
```

Question 41 - What is the use of remote validation in MVC?

Answer

Remote validation is the process where we validate specific data posting data to a server without posting the entire form data to the server. Let's see an actual scenario, in one of my projects I had a requirement to validate an email address, whetehr it already exists in the database. Remote validation was useful for that; without posting all the data we can validate only the email address supplied by the user.

Practical Explanation

Let's create a MVC project and name it accordingly, for me its “TestingRemoteValidation”. Once the project is created let's create a model named UserModel that will look like:

```
public class UserModel
{
    [Required]
    public string UserName
    {
        get;
        set;
    }
    [Remote("CheckExistingEmail", "Home", ErrorMessage = "Email already exists!")]
    public string UserEmailAddress
    {
        get;
        set;
    }
}
```

Let's get some understanding of the remote attribute used, so the very first parameter “CheckExistingEmail” is the the name of the action. The second parameter “Home” is referred to as controller so to validate the input for the UserEmailAddress the “CheckExistingEmail” action of the “Home” controller is called and the third parameter is the error message. Let's implement the

“CheckExistingEmail” action result in our home controller.

```
01.  public ActionResult CheckExistingEmail(string UserEmailAddress)
02.  {
03.      bool ifEmailExist = false;
04.      try
05.      {
06.          ifEmailExist = UserEmailAddress.Equals("mukeshknayak@gmail.com") ? true : false;
07.          return Json(!ifEmailExist, JsonRequestBehavior.AllowGet);
08.      } catch (Exception ex)
09.      {
10.          return Json(false, JsonRequestBehavior.AllowGet);
11.      }
12.  }
```

Learn more here - Remote Validation in MVC

Question 42 - What are the Exception filters in MVC?

Answer

Exception are part and parcel of an application. They are a boon and a ban for an application too. Isn't it? This would be controversial, for developers it helps them track minor and major defects in an application and sometimes they are frustrating when it lets users land on the Yellow screen of death each time. This would make the users mundane to the application. Thus to avoid this, developers handle the exceptions. But still sometimes there are a few unhandled exceptions.

Now what is to be done for them? MVC provides us with built-in "Exception Filters" about which we will explain here.

cc: Google

Let's start!

A Yellow screen of Death can be said is as a wardrobe malfunction of our application.

Get Started

Exception filters run when some of the exceptions are unhandled and thrown from an invoked action. The reason for the exception can be anything and so is the source of the exception.

Creating an Exception Filter

Custom Exception Filters must implement the builtinIExceptionFilter interface. The interface looks as in the following,

```
01.  public interface IExceptionFilter
02.  {
03.      void OnException(ExceptionContext filterContext)
04.  }
```

Whenever an unhandled exception is encountered, the OnException method gets invoked. The parameter as we can see, ExceptionContext is derived from the ControllerContext and has a number of built-in properties that can be used to get the information about the request causing the exception. Their property's ExceptionContextpassess are shown in the following table:

Name	Type	Detail
Result	ActionResult	The result returned by the action being invoked.
Exception	Exception	The unhandled exceptions caused from the actions in the applications.
ExceptionHandled	BOOL	This is a very handy property that returns a bool value (true/false) based on if the exception is handled by any of the filters in the applicaiton or not.

The exception being thrown from the action is detailed by the Exception property and once handled (if), then the property ExceptionHandled can be toggled, so that the other filters would know if the exception has been already handled and cancel the other filter requests to handle. The problem is that if the exceptions are not handled, then the default MVC behavior shows the dreaded yellow screen of death. To the users, that makes a very impression on the users and more importantly, it exposes the

application's handy and secure information to the outside world that may have hackers and then the application gets into the road to hell. Thus, the exceptions need to be dealt with very carefully. Let's show one small custom exception filter. This filter can be stored inside the Filters folder in the web project of the solution. Let's add a file/class called CustomExceptionFilter.cs.

```
01.  public class CustomExceptionFilter: FilterAttribute,
02.      IExceptionFilter
03.  {
04.          public void OnException(ExceptionContext filterContext)
05.          {
06.              if (!filterContext.ExceptionHandled && filterContext.Exception is NullRefere
07.              {
08.                  filterContext.Result = new RedirectResult("customErrorPage.html");
09.                  filterContext.ExceptionHandled = true;
10.              }
11.          }
12.      }
```

Learn more here - Exception Filters in MVC

Question 43 - What is MVC HTML- Helpers and its Methods?

Answer

Helper methods are used to render HTML in the view. Helper methods generates HTML output that is part of the view. They provide an advantage over using the HTML elements since they can be reused across the views and also requires less coding. There are several builtin helper methods that are used to generate the HTML for some commonly used HTML elements, like form, checkbox, dropdownlist etc. Also we can create our own helper methods to generate custom HTML. First we will see how to use the builtin helper methods and then we will see how to create custom helper methods.

Standard HtmlHelper methods

Some of the standard helper methods are,

- ActionLink: Renders an anchor.
- BeginForm: Renders HTML form tag
- CheckBox: Renders check box.
- DropDownList: Renders drop-down list.
- Hidden: Renders hidden field
- ListBox: Renders list box.
- Password: Renders TextBox for password input
- RadioButton: Renders radio button.
- TextArea: Renders text area.
- TextBox: Renders text box.

Learn more here - HtmlHelper Methods in ASP.NET MVC

Question 44 - Define Controller in MVC?

Answer

The controller provides model data to the view, and interprets user actions such as button clicks. The controller depends on the view and the model. In some cases, the controller and the view are the same object.

The Controllers Folder

The Controllers Folder contains the controller classes responsible for handling user input and responses. MVC requires the name of all controllers to end with "Controller".

In our example, Visual Web Developer has created the following files: HomeController.cs (for the Home and About pages) and AccountController.cs (For the Log On pages):

Learn more here - ASP.Net MVC Controller

Question 45 - Explain Model in MVC?

Answer

The model represents the data, and does nothing else. The model does NOT depend on the controller or the view. The MVC Model contains all application logic (business logic, validation logic, and data access logic), except pure view and controller logic. With MVC, models both hold and manipulate application data.

The Models Folde

The Models Folder contains the classes that represent the application model.

Visual Web Developer automatically creates an AccountModels.cs file that contains the models for application security.

Learn more here - Model in ASP.Net MVC : Part 1

Question 46 - Explain View in MVC?

Answer

A view is responsible for displaying all of, or a portion of, data for users. In simple terms, whatever we see on the output screen is a view.

The Views Folder

The Views folder stores the files (HTML files) related to the display of the application (the user interfaces). These files may have the extensions html, asp, aspx, cshtml, and vbhtml, depending on the language content.

The Views folder contains one folder for each controller. Visual Web Developer has created an Account folder, a Home folder, and a Shared folder (inside the Views folder). The Account folder contains pages for registering and logging in to user accounts. The Home folder is used for storing application pages like the home page and the about page. The Shared folder is used to store views shared between controllers (master pages and layout pages).

Learn more here - ASP.Net MVC View

Question 47 - What is Attribute Routing in MVC?

Answer

A route attribute is defined on top of an action method. The following is the example of a Route Attribute in which routing is defined where the action method is defined.

In the following example, I am defining the route attribute on top of the action method

```
01. public class HomeController: Controller
02. {
03.     //URL: /Mvctest
04.     [Route(“Mvctest”)]
05.     public ActionResult Index()
06.     ViewBag.Message = "Welcome to ASP.NET MVC!";
07.     return View();
08. }
09. }
```

Attribute Routing with Optional Parameter

We can also define an optional parameter in the URL pattern by defining a question mark ("?") to the route parameter. We can also define the default value by using parameter=value.

```
public class HomeController: Controller
{
        // Optional URI Parameter
        // URL: /Mvctest/
        // URL: /Mvctest/0023654
        [Route(“Mvctest /
        {
            customerName ?
        }”)]
        public ActionResult OtherTest(string customerName)
        ViewBag.Message = "Welcome to ASP.NET MVC!";
        return View();
    }
    // Optional URI Parameter with default value
    // URL: /Mvctest/
    // URL: /Mvctest/0023654
    [Route(“Mvctest /
    {
        customerName = 0036952
    }”)]
public ActionResult OtherTest(string customerName)
{
    ViewBag.Message = "Welcome to ASP.NET MVC!";
    return View();
}
}
```

Learn more here - Attribute Routing in ASP.Net MVC 5.0

Question 48 - Explain RenderSection in MVC?

Answer

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>
```

Learn more here - ASP.Net MVC 4 - Layout and Section in Razor

Question 49 - What is GET and POST Actions Types?

Answer

GET

GET is used to request data from a specified resource. With all the GET request we pass the URL which is compulsory, however it can take the following overloads.

.get(url [, data] [, success(data, textStatus, jqXHR)] [, dataType]).done/.fail

POST

POST is used to submit data to be processed to a specified resource. With all the POST requests we pass the URL which is compulsory and the data, however it can take the following overloads.

.post(url [, data] [, success(data, textStatus, jqXHR)] [, dataType])

Learn more here - GET and POST Calls to Controller's Method in MVC

Question 50 - What's new in MVC 6?

Answer

In MVC 6 Microsoft removed the dependency of System.Web.Dll from MVC6 because it's so expensive that typically it consume 30k of memory per request and response, whereas now MVC 6 only requires 2k of memory per request and the response is a very small memory consumtion.

The advantage of using the cloud-optimized framework is that we can include a copy of the mono CLR with your website. For the sake of one website we do not need to upgrade the .NET version on the entire machine. A different version of the CLR for a different website running side by side.

MVC 6 is a part of ASP.NET 5 that has been designed for cloud-optimized applications. The runtime automatically picks the correct version of the library when our MVC application is deployed to the cloud.

The Core CLR is also supposed to be tuned with a high resource-efficient optimization.

Microsoft has made many MVC, Web API, WebPage and SignalLrpeices we call MVC 6.

Most of the problems are solved using the Roslyn Compiler. In ASP.NET vNext uses the Roslyn Compiler. Using the Roslyn Compiler we do not need to compile the application, it automatically compiles the application code. You will edit a code file and can then see the changes by refreshing the browser without stopping or rebuilding the project.

Run on hosts other than IIS

Where we use MVC5 we can host it on an IIS server and we can also run it on top of an ASP. NET Pipeline, on the other hand MVC 6 has a feature that makes it better and that feature is itself hosted on an IIS server and a self-user pipeline.

Environment based configuration system

The configuration system provides an environment to easily deploy the application on the cloud. Our application works just like a configuration provider. It helps to retrieve the value from the various configuration sources like XML file.

MVC 6 includes a new environment-based configuration system. Unlike something else it depends on just the Web.Config file in the previous version.

Dependency injection

Using the IServiceProvider interface we can easily add our own dependency injection container. We can replace the default implementation with our own container.

Supports OWIN

We have complete control over the composable pipeline in MVC 6 applications. MVC 6 supports the OWIN abstraction.

Learn more here - ASP.Net MVC 6 New Features

More Interview Questions

- Azure Interview Questions

- C# interview questions
- ASP.NET interview questions
- Bootstrap interview questions
- Html 5 interview questions
- WCF interview questions and answers
- WPF interview questions and answers
- CSS interview questions and answers
- Angularjs interview questions
- SQL Server interview questions
- ADO.NET interview questions and answers
- Interview question on.NET framework or clr
- Javascript interview questions
- Ajax interview questions and answers
- Interview questions for 2 year experience in SQL and C#
- Important.NET interview questions and answers
- Software Testing Interview Questions/
- Dot.NET interview questions for experienced and fresher
- SQL interview questions
- C# interview questions and answers
- jQuery interview question and answer with practices part 2
- Python interview questions

ASP.NET Interview Questions Answers | ASP.NET MVC Interview Questions and Answers | MVC Interview Questions