**What is MVC?**

MVC is a framework pattern that splits an application’s implementation logic into

three component roles: models, views, and controllers.

* **Model**: The business entity on which the overall application operates. Many applications use a persistent storage mechanism (such as a database) to store data. MVC does not specifically mention the data access layer because it is understood to be encapsulated by the Model.
* **View**: The user interface that renders the Model into a form of interaction.
* **Controller**: Handles a request from a View and updates the Model that results in a change of the Model's state.

To implement MVC in .NET we need mainly three classes (View, Controller and the Model).

**Explain MVC Architecture?**



The architecture is self explanatory. The browser (as usual) sends a request to IIS, IIS searches for the route defined in MVC application and passes the request to the controller as per route, the controller communicates with the model and passes the populated model (entity) to View (front end), Views are populated with model properties, and are rendered on the browser, passing the response to browser through IIS via controllers which invoked the particular View.

**What are the new features of MVC2?**

ASP.NET MVC 2 was released in March 2010. Its main features are:

* Introduction of UI helpers with automatic scaffolding with customizable templates.
* Attribute-based model validation on both client and server.
* Strongly typed HTML helpers.
* Improved Visual Studio tooling
* There were also lots of API enhancements and “pro” features, based on feedback from developers building a variety of applications on ASP.NET MVC 1, such as:
* Support for partitioning large applications into *areas.*
* Asynchronous controllers support.
* Support for rendering subsections of a page/site using Html.RenderAction.
* Lots of new helper functions, utilities, and API enhancements.

**What are the new features of MVC3?**

ASP.NET MVC 3 shipped just 10 months after MVC 2 in Jan 2011.Some of the top features in MVC 3 included:

* The Razor view engine.
* Support for .NET 4 Data Annotations.
* Improved model validation
* Greater control and flexibility with support for dependency resolution and global action filters.
* Better JavaScript support with unobtrusive JavaScript, jQuery Validation, and JSON binding.
* Use of NuGet to deliver software and manage dependencies throughout the platform.

**What are the new features of MVC4?**

Following are the top features of MVC4:

* ASP.NET Web API.
* Enhancements to default project templates.
* Mobile project template using jQuery Mobile.
* Display Modes.
* Task support for Asynchronous Controllers.
* Bundling and minification.

**Explain “page lifecycle” of an ASP.NET MVC?**

Following process are performed by ASP.Net MVC page:

1) App initialization

2) Routing

3) Instantiate and execute controller

4) Locate and invoke controller action

5) Instantiate and render view

**Advantages of MVC Framework?**

1. Provides a clean separation of concerns between UI (Presentation layer), model (Transfer objects/Domain Objects/Entities) and Business Logic (Controller).

2. Easy to UNIT Test.

3. Improved reusability of views/model. One can have multiple views which can point to

same model and vice versa.

4. Improved structuring of the code.

**What do you mean by Separation of Concerns?**

As per Wikipedia 'the process of breaking a computer program into distinct features that overlap in functionality as little as possible'. MVC design pattern aims to separate content from presentation and data-processing from content.

**Where do we see Separation of Concerns in MVC?**

Between the data-processing (*Model*) and the rest of the application.

When we talk about Views and Controllers, their ownership itself explains separation. The views are just the presentation form of an application, it does not have to know specifically about the requests coming from controller. The Model is independent of View and Controllers, it only holds business entities that can be passed to any View by the controller as required for exposing them to the end user. The controller is independent of Views and Models, its sole purpose is to handle requests and pass it on as per the routes defined and as per the need of rendering views. Thus our business entities (model), business logic (controllers) and presentation logic (views) lie in logical/physical layers independent of each other.

**What is Razor View Engine?**

Razor is the first major update to render HTML in MVC3. Razor was designed specifically as a view engine syntax. It has one main focus: *code*-*focused templating for HTML generation*. Here’s how that same markup would be generated using Razor:

@model MvcMusicStore.Models.Genre

@{ViewBag.Title = "Browse Albums";}

<div class="genre">

<h3><em>@Model.Name</em> Albums</h3>

<ul id="album-list">

@foreach (var album in Model.Albums)

{

<li>

<a href="@Url.Action("Details", new { id = album.AlbumId })">

<img alt="@album.Title" src="@album.AlbumArtUrl" />

<span>@album.Title</span>

</a>

</li>

}

</ul>

</div>

The Razor syntax is easier to type, and easier to read. Razor doesn’t have the XML-like heavy syntax.

of the Web Forms view engine.

**What is Unobtrusive JavaScript?**

Unobtrusive JavaScript is a general term that conveys a general philosophy, similar to the term

REST (Representational State Transfer). The high-level description is that unobtrusive JavaScript doesn’t intermix JavaScript code in your page markup. For example, rather than hooking in via event attributes like onclick and onsubmit, the unobtrusive JavaScript attaches to elements by their ID or class, often based on the presence of other attributes (such as HTML5 data- attributes).

It’s got semantic meaning, and all of it — the tag structure, element attributes, and so on — should have a precise meaning. Strewing JavaScript gunk across the page to facilitate interaction (I’m looking at you, \_\_doPostBack!) harms the content of the document.

**What is JSON Binding?**

MVC 3 included JavaScript Object Notation (JSON) binding support via the new

JsonValueProviderFactory, enabling the action methods to accept and model-bind data in JSON format. This is especially useful in advanced Ajax scenarios like client templates and data binding that need to post data back to the server.

**What is Dependency Resolution?**

MVC 3 introduced a new concept called a *dependency resolver*, which greatly simplified the use of dependency injection in your applications. This made it easier to decouple application components, making them more configurable and easier to test.

Support was added for the following scenarios:

* Controllers (registering and injecting controller factories, injecting controllers)
* Views (registering and injecting view engines, injecting dependencies into view pages)
* Action fi lters (locating and injecting fi lters)
* Model binders (registering and injecting)
* Model validation providers (registering and injecting)
* Model metadata providers (registering and injecting)
* Value providers (registering and injecting)

**What are Display Modes in MVC4?**

Display modes use a convention-based approach to allow selecting different views based on the browser making the request. The default view engine fi rst looks for views with names ending with .Mobile.cshtml when the browser’s user agent indicates a known mobile device. For example, if we have a generic view titled Index.cshtml and a mobile view titled Index.Mobile.cshtml, MVC 4 will automatically use the mobile view when viewed in a mobile browser.

Additionally, we can register your own custom device modes that will be based on your own custom criteria — all in just one code statement. For example, to register a WinPhone device mode that would serve views ending with .WinPhone.cshtml to Windows Phone devices, you’d use the following code in the Application\_Start method of your Global.asax:

DisplayModeProvider.Instance.Modes.Insert(0, new DefaultDisplayMode("WinPhone")

{

ContextCondition = (context => context.GetOverriddenUserAgent().IndexOf

("Windows Phone OS", StringComparison.OrdinalIgnoreCase) >= 0)

});

**What is AuthConfig.cs in MVC4?**

AuthConfig.cs is used to configure security settings, including sites for OAuth login.

 **What is BundleConfig.cs in MVC4?**

 BundleConfig.cs in MVC4 is used to register bundles used by the bundling and minification

 system. Several bundles are added by default, including jQuery, jQueryUI, jQuery validation, Modernizr, and default CSS references.

 **What is FilterConfig.cs in MVC4?**

 This is used to register global MVC filters. The only filter registered by default is the HandleErrorAttribute, but this is a great place to put other filter registrations.

 **What is RouteConfig.cs in MVC4?**

 RouteConfig.cs holds the granddaddy of the MVC config statements, Route configuration.

 **What is WebApiConfig.cs in MVC4?**

 Used to register Web API routes, as well as set any additional Web API configuration settings.

**What’s new in adding controller in MVC4 application?**

Previously(in MVC3 and MVC2), the Visual Studio Add Controller menu item only displayed when we right-clicked on the Controllers folder. However, the use of the Controllers folder was purely for organization. (MVC will recognize any class that implements the IController interface as a Controller, regardless of its location in your application.) The MVC 4 Visual Studio tooling has been modified to display the Add Controller menu item for any folder in your MVC project. This allows us to organize your controllers however you would like, perhaps separating them into logical groups or separating MVC and Web API controllers.

**What are the software requirements of ASP.NET MVC4 application?**

MVC 4 runs on the following Windows client operating systems:

* Windows XP
* Windows Vista
* Windows 7
* Windows 8

It runs on the following server operating systems:

* Windows Server 2003
* Windows Server 2008
* Windows Server 2008 R2

MVC 4 development tooling is included with Visual Studio 2012 and can be installed on Visual

Studio 2010 SP1/Visual Web Developer 2010 Express SP1.

**What are the various types of Application Templates used to create an MVC application?**

The various templates are as follows,

**1.** **The Internet Application template:** This contains the beginnings of an MVC web

application — enough so that you can run the application immediately after creating it

and see a few pages. This template also includes some basic account management functions which run against the ASP.NET Membership .

**2.** **The Intranet Application template:** The Intranet Application template was added as part of

the ASP.NET MVC 3 Tools Update. It is similar to the Internet Application template,but the account management functions run against Windows accounts rather than the ASP.NET Membership system.

**3.** **The Basic template:** This template is pretty minimal. It still has the basic folders, CSS, and

MVC application infrastructure in place, but no more. Running an application created using

the Empty template just gives you an error message.

**Why use Basic template?** The Basic template is intended for experienced MVC developers

who want to set up and configure things exactly how they want them.

**4.The Empty template:** The Basic template used to be called the Empty template, but developers complained that it wasn’t quite empty enough. With MVC 4, the previous Empty

template was renamed Basic, and the new Empty template is about as empty as we can get.

It has the assemblies and basic folder structure in place, but that’s about it.

**5.** **The Mobile Application template:** The Mobile Application template is preconfigured with jQuery Mobile to jump-start creating a mobile only website. It includes mobile visual themes, a touch-optimized UI, and support for Ajax navigation.

**6.** **The Web API template:** ASP.NET Web API is a framework for creating HTTP services.

The Web API template is similar to the Internet Application template but is streamlined for Web API development. For instance, there is no user account management functionality, as Web API account management is often signify-cantly different from standard MVC account management. Web API functionality is also available in the other MVC project templates, and even in non-MVC project types.

**What are the default Top level directories created when adding MVC4 application?**

Default Top level Directories are:

**DIRECTORY** **PURPOSE**

/Controllers To put Controller classes that handle URL requests

/Models To put classes that represent and manipulate data and business objects

/Views To put UI template files that are responsible for rendering output like HTML.

/Scripts To put JavaScript library files and scripts (.js)

/Images To put images used in your site

/Content To put CSS and other site content, other than scripts and images

/Filters To put filter code.

/App\_Data To store data files you want to read/write

/App\_Start To put configuration code for features like Routing, Bundling, Web API.

**What is namespace of asp.net mvc?**

ASP.NET MVC namespaces as well as classes are located in assembly System.Web.Mvc.

**Note**: Some of the content has been taken from various books/articles.

**What is System.Web.Mvc namespace?**

This namespace contains classes and interfaces that support the MVC pattern for ASP.NET Web applications. This namespace includes classes that represent controllers, controller

factories, action results, views, partial views, and model binders.

**What is System.Web.Mvc.Ajax namespace?**

System.Web.Mvc.Ajax namespace contains classes that supports Ajax scripting in an ASP.NET MVC application. The namespace includes support for Ajax scripts and Ajax option settings as well.

**What is System.Web.Mvc.Async namespace?**

System.Web.Mvc.Async namespace contains classes and interfaces that support asynchronous actions in an ASP.NET MVC application.

**What is System.Web.Mvc.Html namespace?**

System.Web.Mvc.Html namespace contains classes that help render HTML controls in an MVC application. This namespace includes classes that support forms, input controls, links, partial views, and validation.

**What is ViewData, ViewBag and TempData?**

MVC provides us ViewData, ViewBag and TempData for passing data from controller, view and in next requests as well. ViewData and ViewBag are similar to some extent but TempData performs additional roles.

**What are the roles and similarities between ViewData and ViewBag?**

* Maintains data when move from controller to view.
* Passes data from controller to respective view.
* Their value becomes null when any redirection occurs, because their role is to provide a way to communicate between controllers and views. It’s a communication mechanism within the server call.

**What are the differences between ViewData and ViewBag?**(taken from a blog)

* 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.

**NOTE** *Although there might not be a technical advantage to choosing one format over the other, there are some critical differences to be aware of between the two syntaxes.*

*One obvious difference is that* ViewBag *works only when the key being accessed is a valid C# identifier. For example, if you place a value in* ViewData["KeyWith Spaces"]*, you can’t access that value using* ViewBag *because the codewon’t compile.*

*Another key issue to be aware of is that dynamic values cannot be passed in as parameters to extension methods. The C# compiler must know the real type of every parameter at compile time in order for it to choose the correct extension method.*

*If any parameter is dynamic, compilation will fail. For example, this code will always fail:* @Html.TextBox("name", ViewBag.Name)*. To work around this,either use* ViewData["Name"] *or cast the value to a specifi c type:* (string) ViewBag.Name*.*

**What is TempData?**

TempData is a dictionary derived from the **TempDataDictionary** class and stored in short lives session. It is a string key and object value.

It keep the information for the time of an HTTP Request. This means only from one page to another. It helps to maintain data when we move from one controller to another controller or from one action to other action. In other words, when we redirect Tempdata helps to maintain data between those redirects. It internally uses session variables. Temp data use during the current and subsequent request only means it is use when we are sure that next request will be redirecting to next view. It requires typecasting for complex data type and check for null values to avoid error. Generally it is used to store only one time messages like error messages, validation messages.

**How can you define a dynamic property with the help of viewbag in ASP.NET MVC?**

Assign a key name with syntax,

**ViewBag.[Key]=[ Value]** and value using equal to operator.

For example, you need to assign list of students to the dynamic Students property

of ViewBag.

List<string> students = new List<string>();

countries.Add("Akhil");

countries.Add("Ekta");

ViewBag.Students = students;

//Students is a dynamic property associated with ViewBag.

**Note**: Some of the content has been taken from various books/articles.

**What is ViewModel(taken from stackoverflow)?**

|  |  |
| --- | --- |
| accepted | A view model represents data that you want to have displayed on your view/page.Lets say that you have an Employee class that represents your employee domain model and it contains the following 4 properties:public class Employee : IEntity{ public int Id { get; set; } // Employee's unique identifier public string FirstName { get; set; } // Employee's first name public string LastName { get; set; } // Employee's last name public DateTime DateCreated { get; set; } // Date when employee was created}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. For example, lets say that you want to add a new employee record, your view model might look like this:public class CreateEmployeeViewModel{ public string FirstName { get; set; } public string LastName { get; set; }}As you can see it only contains 2 of the properties of the employee domain model. Why is this you may ask? Id might not be set from the view, it might be auto generated by the Employee table. AndDateCreated might also be set in the stored procedure or in the service layer of your application. So Id and DateCreated is not need in the view model.When loading the view/page, the create action method in your employee controller will create an instance of this view model, populate any fields if required, and then pass this view model to the view:public class EmployeeController : Controller{ private readonly IEmployeeService employeeService; public EmployeeController(IEmployeeService employeeService) { this.employeeService = employeeService; } public ActionResult Create() { CreateEmployeeViewModel viewModel = new CreateEmployeeViewModel(); return View(viewModel); } public ActionResult Create(CreateEmployeeViewModel viewModel) { // Do what ever needs to be done before adding the employee to the database }}Your view might look like this (assuming you are using ASP.NET MVC3 and razor):@model MyProject.Web.ViewModels.ProductCreateViewModel<table> <tr> <td><b>First Name:</b></td> <td>@Html.TextBoxFor(x => x.FirstName, new { maxlength = "50", size = "50" }) @Html.ValidationMessageFor(x => x.FirstName) </td> </tr> <tr> <td><b>Last Name:</b></td> <td>@Html.TextBoxFor(x => x.LastName, new { maxlength = "50", size = "50" }) @Html.ValidationMessageFor(x => x.LastName) </td> </tr></table>Validation would thus be done only on FirstName and LastName. Using [Fluent Validation](http://fluentvalidation.codeplex.com/) you might have validation like this:public class CreateEmployeeViewModelValidator : AbstractValidator<CreateEmployeeViewModel>{ public CreateEmployeeViewModelValidator() { RuleFor(x => x.FirstName) .NotEmpty() .WithMessage("First name required") .Length(1, 50) .WithMessage("First name must not be greater than 50 characters"); RuleFor(x => x.LastName) .NotEmpty() .WithMessage("Last name required") .Length(1, 50) .WithMessage("Last name must not be greater than 50 characters"); }}The key thing to remember is that the view model only represents the data that you want use. You can imagine all the uneccessary code and validation if you have a domain model with 30 properties and you only want to update a single value. Given this scenario you would only have this one value/property in the view model and not the whole domain object. |

**How do you check for AJAX request with C# in MVC.NET?**

The solution is independed of MVC.NET framework and is global across server side

technologies. Most modern AJAX applications utilize XmlHTTPRequest to send

async request to the server. Such requests will have distinct request header:

X-Requested-With = XMLHTTPREQUEST



MVC.NET provides helper function to check for ajax requests which internally inspects

 X-Requested-With request header to set IsAjax flag.

**What are Scaffold template?**

These templates use the Visual Studio T4 templating system to generate a view based on the model type selected. *Scaffolding* in ASP.NET MVC can generate the boilerplate code we need for create, read, update,and delete (CRUD) functionality in an application. The scaffolding templates can examine the type definition for, and then generate a controller and the controller’s associated views. The scaffolding knows how to name controllers, how to name views, what code needs to go in each component, and where to place all these pieces in the project for the application to work.

**What are the types of Scaffolding Templates?**

Various types are as follows,

SCAFFOLD DESCRIPTION

Empty Creates empty view. Only the model type is specified using the model syntax.

Create Creates a view with a form for creating new instances of the model.

 Generates a label and input field for each property of the model type.

Delete Creates a view with a form for deleting existing instances of the model.

 Displays a label and the current value for each property of the model.

Details Creates a view that displays a label and the value for each property of the

 model type.

Edit Creates a view with a form for editing existing instances of the model.

 Generates a label and input fi eld for each property of the model type.

List Creates a view with a table of model instances. Generates a column

 for each property of the model type. Make sure to pass an IEnumerable<YourModelType> to this view from your action method.

 The view also contains links to actions for performing the create/edit/delete operations.

**Show an example of difference in syntax in Razor and WebForm View?**

Razor <span>@model.Message</span>

Web Forms <span><%: model.Message %></span>

Code expressions in Razor are always HTML encoded. This Web Forms syntax also automatically HTML encodes the value.

**What are Code Blocks in Views?**

Unlike code expressions, which are evaluated and outputted to the response, blocks of code are simply sections of code that are executed. They are useful for declaring variables that we may need to use later.

**Razor**

@{

int x = 123;

string y = ˝because.˝;

}

**Web Forms**

<%

int x = 123;

string y = "because.";

%>

**What is HelperPage.IsAjax Property?**

HelperPage.IsAjax gets a value that indicates whether Ajax is being used during the request of the Web page.

**Namespace**: System.Web.WebPages

**Assembly**: System.Web.WebPages.dll

However, same can be achieved by checking requests header directly:

Request["X-Requested-With"] == “XmlHttpRequest”.

**Explain combining text and markup in Views with the help of an example?**

This example shows what intermixing text and markup looks like using Razor as compared to Web Forms:

**Razor**

@foreach (var item in items) {

<span>Item @item.Name.</span>

}

**Web Forms**

<% foreach (var item in items) { %>

<span>Item <%: item.Name %>.</span>

<% } %>

**Explain Repository Pattern in ASP.NET MVC?**

In simple terms, a repository basically works as a mediator between our business logic layer and our data access layer of the application. Sometimes, it would be troublesome to expose the data access mechanism directly to business logic layer, it may result in redundant code for accessing data for similar entities or it may result in a code that is hard to test or understand. To overcome these kinds of issues, and to write an Interface driven and test driven code to access data, we use Repository Pattern. The repository makes queries to the data source for the data, thereafter maps the data from the data source to a business entity/domain object, finally and persists the changes in the business entity to the data source. According to MSDN, a repository separates the business logic from the interactions with the underlying data source or Web service. The separation between the data and business tiers has three benefits:

* It centralizes the data logic or Web service access logic.
* It provides a substitution point for the unit tests.
* It provides a flexible architecture that can be adapted as the overall design of the application evolves.

In Repository, we write our whole business logic of CRUD operations with the help of Entity Framework classes, that will not only result in meaningful test driven code but will also reduce our controller code of accessing data.

**How can you call a javascript function/method on the change of Dropdown List in MVC?**

**Create a java-script method:**

<script type="text/javascript">

function selectedIndexChanged() {

}

</script>

**Invoke the method:**

<%:Html.DropDownListFor(x => x.SelectedProduct,

new SelectList(Model.Users, "Value", "Text"),

"Please Select a User", new { id = "ddlUsers",

onchange="selectedIndexChanged()" })%>

**Explain Routing in MVC?**

A route is a URL pattern that is mapped to a handler. The handler can be a physical

file, such as an .aspx file in a Web Forms application. Routing module is responsible for mapping incoming browser requests to particular MVC controller actions.

Routing within the ASP.NET MVC framework serves two main purposes:

* It matches incoming requests that would not otherwise match a file on the file system and maps the requests to a controller action.
* It constructs outgoing URLs that correspond to controller actions.

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

When an MVC application first starts, the Application\_Start() method in global.asax is called. This method, calls the RegisterRoutes() method. The RegisterRoutes() method creates the route table for MVC application.

**What are Layouts in ASP.NET MVC Razor?**

Layouts in Razor help maintain a consistent look and feel across multiple views within our application.As compared to Web Forms Web Forms, layouts serve the same purpose as master pages, but offer both a simpler syntax and greater flexibility.

We can use a layout to define a common template for your site (or just part of it). This template contains one or more placeholders that the other views in your application provide content for. In some ways, it’s like an abstract base class for your views.

e.g. declared at the top of view as,

 @{

 Layout = "~/Views/Shared/SiteLayout.cshtml";

}

**What is ViewStart?**

For group of views that all use the same layout, this can get a bit redundant and harder to maintain.

The \_ViewStart.cshtml page can be used to remove this redundancy. The code within this file

is executed before the code in any view placed in the same directory. This fi le is also recursively applied to any view within a subdirectory.

When we create a default ASP.NET MVC project, we find there is already a \_ViewStart

.cshtml fi le in the Views directory. It specifi es a default layout:

@{

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

}

Because this code runs before any view, a view can override the Layout property and choose a different one. If a set of views shares common settings, the \_ViewStart.cshtml file is a useful place to consolidate these common view settings. If any view needs to override any of the common settings, the view can set those values to another value.

**Note**: Some of the content has been taken from various books/articles.

**What are HTML Helpers?**

HTML helpers are methods we can invoke on the Html property of a view. We also have

access to URL helpers (via the Url property), and AJAX helpers (via the Ajax property). All

these helpers have the same goal: to make views easy to author. The URL helper is also available from within the controller.

Most of the helpers, particularly the HTML helpers, output HTML markup. For example, the

BeginForm helper is a helper we can use to build a robust form tag for our search

form, but without using lines and lines of code:

@using (Html.BeginForm("Search", "Home", FormMethod.Get)) {

<input type="text" name="q" />

<input type="submit" value="Search" />

}

**What is Html.ValidationSummary?**

The ValidationSummary helper displays an unordered list of all validation errors in the ModelState dictionary. The Boolean parameter you are using (with a value of true) is telling the helper to exclude property-level errors. In other words, you are telling the summary to display only the errors in ModelState associated with the model itself, and exclude any errors associated with a specific model property. We will be displaying property-level errors separately.Assume you have the following code somewhere in the controller action rendering the edit view:

*ModelState.AddModelError("", "This is all wrong!");*

*ModelState.AddModelError("Title", "What a terrible name!");*

The first error is a model-level error, because you didn’t provide a key (or provided an empty key) to associate the error with a specifi c property. The second error you associated with the Title property, so in your view it will not display in the validation summary area (unless you remove the parameter to the helper method, or change the value to false). In this scenario, the helper renders the following HTML:

*<div class="validation-summary-errors">*

*<ul>*

*<li>This is all wrong!</li>*

*</ul>*

*</div>*

Other overloads of the ValidationSummary helper enable you to provide header text and set specific HTML attributes.

**NOTE** *By convention, the* ValidationSummary *helper renders the CSS class* validation-summary-errors *along with any specifi c CSS classes you provide.The default MVC project template includes some styling to display these items in red, which you can change in* styles.css*.*

**What are Validation Annotations?**

Data annotations are attributes you can find in S**ystem.ComponentModel.DataAnnotations**

namespace.These attributes provide server-side validation, and the framework also supports client-side validation when you use one of the attributes on a model property. You can use four attributes in the DataAnnotations namespace to cover common validation scenarios,

**Required, String Length, Regular Expression, Range.**

**What is Html.Partial?**

The Partial helper renders a partial view into a string. Typically, a partial view contains reusable markup you want to render from inside multiple different views. Partial has four overloads:

public void Partial(string partialViewName);

public void Partial(string partialViewName, object model);

public void Partial(string partialViewName, ViewDataDictionary viewData);

public void Partial(string partialViewName, object model,

ViewDataDictionary viewData);

**What is Html.RenderPartial?**

The RenderPartial helper is similar to Partial, but RenderPartial writes directly to the response output stream instead of returning a string. For this reason, you must place RenderPartial inside a code block instead of a code expression. To illustrate, the following two lines of code render the same output to the output stream:

@{Html.RenderPartial("AlbumDisplay "); }

@Html.Partial("AlbumDisplay ")

**If they are same then which one to use?**

In general, you should prefer Partial to RenderPartial because Partial is more convenient (you don’t have to wrap the call in a code block with curly braces). However, RenderPartial may result in better performance because it writes directly to the response stream, although it would require a lot of use (either high site traffic or repeated calls in a loop) before the difference would be noticeable.

**How do you return a partial view from controller?**

return PartialView(options); //options could be Model or View name

**What are different ways of returning a View?**

There are different ways for returning/rendering a view in MVC Razor.E.g. return View(), return RedirectToAction(), return Redirect() and return RedirectToRoute().

**Question 1: What does MVC stand for>**

**Answer:** Model-View-Controller.

**Question 2: What are three main components or aspects of MVC?**

**Answer:** Model-View-Controller

**Question 3: Which namespace is used for ASP.NET MVC? Or which assembly is used to define the MVC framework?**
**Answer:** System.Web.Mvc

**Question 4: What is the default view engine in ASP.NET MVC?**
**Answer:** Web Form(ASPX) and Razor View Engine.

**Question 5: Can we remove the default View Engine?**
**Answer:** Yes, by using the following code:

1. protected void Application\_Start()
2. {
3. ViewEngines.Engines.Clear();
4. }

**Question 6: Can we have a Custom View Engine?**
**Answer:** Yes, by implementing the IViewEngine interface or by inheriting from the VirtualPathProviderViewEngine abstract class.

**Question 7: Can we use a third-party View Engine?**
**Answer:** Yes, ASP.NET MVC can have Spark, NHaml, NDjango, Hasic, Brail, Bellevue, Sharp Tiles, String Template, Wing Beats, SharpDOM and so on third-party View Engine.

**Question 8: What are View Engines?**
**Answer:** View Engines are responsible for rendering the HTML from your views to the browser.

**Question 9: What is Razor Engine?**
**Answer:** The Razor view engine is an advanced view engine from Microsoft, packaged with MVC 3. Razor uses an @ character instead of aspx's <% %> and Razor does not require you to explicitly close the code-block.

**Question 10: What is scaffolding?**
**Answer:** Quickly generating a basic outline of your software that you can then edit and customize.

**Question 11: What is the name of Nuget scaffolding package for ASP.NET MVC3 scaffolding?**
**Answer:** MvcScaffolding

**Question 12: Can we share a view across multiple controllers?**
**Answer:** Yes, it is possible to share a view across multiple controllers by putting a view into the shared folder.

**Question 13: What is unit testing?**
**Answer:** The testing of every smallest testable block of code in an automated manner. Automation makes things more accurate, faster and reusable.

**Question 14: Is unit testing of MVC application possible without running the controller?**
**Answer:** Yes, by the preceding definition.

**Question 15: Can you change the action method name?**
**Answer:** Yes, we can change the action method name using the ActionName attribute. Now the action method will be called by the name defined by the ActionName attribute.

1. [ActionName("DoAction")]
2. public ActionResult DoSomething()
3. {
4. /TODO:
5. return View();
6. }

**Question 16: How to prevent a controller method from being accessed by an URL?**
**Answer:** By making the method private or protected but sometimes we need to keep this method public. This is where the NonAction attribute is relevant.

**Question 17: What are the features of MVC5?**
**Answer:**

* Scaffolding
* ASP.NET Identity
* One ASP.NET
* Bootstrap
* Attribute Routing
* Filter Overrides

**Question 18: What are the various types of filters in an ASP.NET MVC application?**
**Answer:**

* Authorization filters
* Action filters
* Result filters
* Exception filters

**Question 19: If there is no match in the route table for the incoming request's URL, which error will result?**
**Answer:** 404 HTTP status code

**Question 20: How to enable Attribute Routing?**
**Answer:** By adding a Routes.MapMvcAttributeRoutes() method to the RegisterRoutes() method of the RouteConfig.cs file.

**Question 21:** Which page is used to serve the common layout page for a group of views?

**Answer:** \_ViewStart.cshml

**Question 22: How can be render layout in ASP.NET MVC?

Answer:**

* The \_ViewStart.cshtml file in the Views folder provides the default Layout page.
* We can also place the \_ViewStart.cshtml file in the SubFolders of Views for providing a default layout to a specific directory.
* Defining the Layout in a view on the top. @{ Layout =”~/Views/Shared/\_SchoolLayout.cshtml”;}
* Layout from ActionResult

**Question 23: In which version of MVC was the App\_Start folder introduced?

Answer:** MVC 4

**Question 24: What is the name of the configuration files that the App\_Start folder contains?

Answer:**

* BundleConfig.cs
* FilterConfig.cs
* RouteConfig.cs
* WebApiConfig.cs

**Question 25: Which does not require type casting among ViewData, ViewBag, TempData and Session?**
 **Answer:** ViewBag

**Question 26: What is the life of ViewData, ViewBag, TempData and Session in increasing order.

Answer:**

* TempData: only until the target view is fully loaded.
* ViewData: during the current request.
* ViewBag: during the current request.
* Session: for all requests.

**Question 27: From which class does ViewData, ViewBag, TempData and Session derive from?

Answer:**

* TempData: TempDataDictionary class
* ViewData: ViewDataDictionary class
* ViewBag: takes advantage of the new dynamic features in C# 4.0
* Session: HttpSessionState class

**Question 28: Which class is the base class of all action results?

Answer:** ActionResult class
 **Question 29: Why do we need server-side validation when we have client-side validation?

Answer:** The user can disable a script in his browser and bypass client-side validation. So, we need server-side validation to protect from such unvalidated data.
 **Question 30: Which property is used to determine an error in Model State?

Answer:** ModelState.IsValid property

If there is any error then ModelState.IsValid returns false. If there is no error it will return true.
 **Question 31: What is latency?

Answer:** The amount of time it takes for the host server to receive, process, and deliver on a request for a page resource (images, CSS files, and so on).
 **Question 32: Why do we use CDN?

Answer:**

* It solves latency problems.
* A CDN caches static resources in distributed servers across a region or worldwide.
* Thereby bringing resources closer to users and reducing round trip time.

**Question 33: In which version of ASP.NET MVC was bundling and minification techniques introduced?**
 **Answer:** ASP.NET MVC4 and .NET Framework 4.5
 **Question 34: What is bundling and minification in ASP.NET MVC?

Answer:**

Bundling: It bundles multiple files into a single file. You can create CSS, JavaScript and other bundles. Fewer files mean fewer HTTP requests that can improve first page load performance.
Minification: Minification does a variety of code optimizations for scripts or CSS, such as removing unnecessary white space and comments and shortening variable names to one character.
 **Question 34: Which filter is executed when there is an unhandled exception thrown during the execution of the ASP.NET pipeline?**
 **Answer:** Exception filters
 **Question 35: What is the order of execution of filters?

Answer:**

* Authentication filters
* Authorization filters
* Action filters
* Result filters

**Question 36: Various ways of implementing Dependency Injection.

Answer:**

* Constructor Injection
* Property Injection
* Method Injection

**Question 37: Advantage of Dependency Injection.

Answer:**

* Improves Application Testing
* Improves Code Maintainability
* Reduces Class Coupling
* Increases code reusing

**Question 38: What is Test Driven Development?

Answer:** TDD is a methodology in which we write tests first then write code.
 **Question 39: What is Area?

Answer:** Area allows us to organize models, views and controllers into separate functional sections of the application.
 **Question 40: Can be use bundling and minification in ASP.NET MVC 3.

Answer:** Yes, by using the System.Web.Optimization class.