

# ASP.NET MVC Interview Questions & Answers



MVC



**By Shailendra Chauhan**

Microsoft MVP, Founder & CEO - Dot Net Tricks

 **DotNetTricks**

# ASP.NET MVC Interview Questions & Answers

All rights reserved. No part of this book can be reproduced or stored in any retrieval system or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, uploading on server and scanning without the prior written permission of the Dot Net Tricks Innovation Pvt. Ltd.

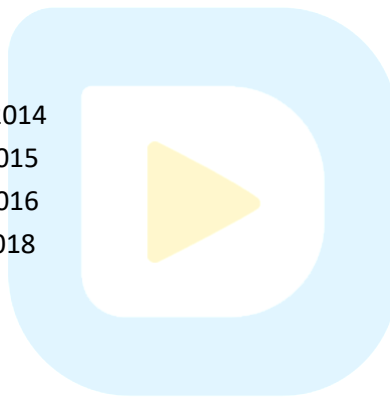
The author of this book has tried their best to ensure the accuracy of the information described in this book. However, the author cannot guarantee the accuracy of the information contained in this book. The author or Dot Net Tricks Innovation Pvt. Ltd. will not be liable for any damages, incidental or consequential caused directly or indirectly by this book.

Further, readers should be aware that the websites or reference links listed in this book may have changed or disappeared between when this book was written and when it is read.

All other trademarks referred to in this book are the property of their respective owners.

## Release History

- Initial Release 1.0.0 - 10<sup>th</sup> Mar 2014
- Second Release 1.0.1 - 1<sup>st</sup> Jan 2015
- Third Release 1.0.2 - 12<sup>th</sup> Aug 2016
- Fourth Release 1.1 - 19<sup>th</sup> Nov 2018
- Fifth Release 1.2 - 4<sup>th</sup> Jan 2019



# About Dot Net Tricks

Dot Net Tricks is founded by Shailendra Chauhan (Microsoft MVP), in Jan 2010. Dot Net Tricks came into existence in form of a blog post over various technologies including .NET, C#, SQL Server, ASP.NET, ASP.NET MVC, JavaScript, Angular, Node.js and Visual Studio etc.

The company which is currently registered by a name of Dot Net Tricks Innovation Pvt. Ltd. came into the shape in 2015. Dot Net Tricks website has an average footfall on the tune of 300k+ per month. The site has become a cornerstone when it comes to getting skilled-up on .NET technologies and we want to gain the same level of trust in other technologies. This is what we are striving for.

We have a very large number of trainees who have received training from our platforms and immediately got placement in some of the reputed firms testifying our claims of providing quality training. The website offers you a variety of free study material in form of articles.

## Dot Net Tricks Training Solutions

Dot Net Tricks provide you training in traditional as well as new age technologies, via various formats.

### **Master Courses (Instructor-led)**

For a beginner who needs regular guidance, we have a fully packed Master Courses. They are almost equal to semester courses taught in engineering colleges when it comes to length, breadth of content delivery, the only difference instead of 5-6 months, they take approx. 16-weekend classes (2 months).

The detail about Master courses can be found here: <https://www.dotnettricks.com/instructor-led-courses>

### **Hands-On Learning (Learn to code)**

Hands-On Learning courses give you the confidence to code and equally helpful to work in real-life scenarios. This course is composed of hands-on exercise using IDE or cloud labs so that you can practice each and everything by yourself. You can learn to code at your own pace, time and place.

The detail about Hands-On Learning courses can be found here: <https://www.scholarhat.com>

### **Skill Bootcamps (Instructor-led)**

Professionals who don't have two months' time and want to get skilled up in least possible time due to some new project that their company has to take in very near future, we have designed Skill Bootcamps Concept, where you will get trained on consecutive days in a fast-paced manner, where our full focus is going to be on hands-on delivery of technological exercises.

The detail about Skill Bootcamps can be found here: <https://www.dotnettricks.com/skill-bootcamp>

### **Self-paced Courses**

Self-paced courses give you the liberty to study at your own pace, time and place. We understand everyone has their own comfort zone, some of you can afford to dedicate 2 hours a day, some of you not. Keeping this thing in mind, we created these self-paced courses. While creating these courses we have ensured that quality of courses doesn't get compromise at any parameter, and they also will be able to produce the same results as our other course formats, given the fact you will be able to put your own honest effort.

The detail about Self-paced courses can be found here: <https://www.dotnettricks.com/self-paced-courses>

### **Corporate Training (Online and Classroom)**

Dot Net Tricks having a pool of mentors who help the corporate to enhance their employment skills as per changing technology landscape. Dot Net Tricks offers customized training programs for new hires and experienced employees through online and classroom mode. As a trusted and resourceful training partner, Dot Net Tricks helps the corporate to achieve success with its industry-leading instructional design and customer training initiatives.

The detail about Corporate Training can be found here: <https://www.dotnettricks.com/corporate-training>

### **Learning Platform**

We have a very robust technology platform to answer the needs of all our trainees, no matter which program they enrolled in. We have a very self-intuitive Learning Management System (LMS), which help you in remain focused and keeping an eye over your progress.

We offer two Learning Platforms as given below:

1. Dot Net Tricks: <https://www.dotnettricks.com>
2. Scholar Hat: <https://www.scholarhat.com>

Apart from these, we also provide on-demand Skill bootcamps and personalized project consultation.

# Dedication

*My mother Mrs Vriksha Devi and my wife Reshu Chauhan deserve to have their name on the cover as much as I do for all their support made this possible. I would like to say thanks to all my family members Virendra Singh(father), Jaishree and Jyoti(sisters), Saksham and Pranay(sons), friends, to you and to readers or followers of my blogs at <https://www.dotnettricks.com/mentors/shailendra-chauhan> to encourage me to write this book.*

**-Shailendra Chauhan**



# Introduction

Writing a book has never been an easy task. It takes a great effort, patience and consistency with strong determination to complete it. Also, one should have a depth knowledge over the subject is going to write.

**So, what where my qualification to write this book?** My qualification and inspiration come from my enthusiasm for and the experience with the technology and from my analytic and initiative nature. Being a trainer, analyst, consultant and blogger, I have thorough knowledge and understandings of .NET technologies. My inspiration and knowledge have also come from many years of my working experience and research over it.

**So, the next question is who this book is for?** This book covers useful Interview Questions and Answers on ASP.NET MVC. This book is appropriate for the novice as well as for senior-level professionals who want to strengthen their skills before appearing for an interview on ASP.NET MVC. This book is equally helpful to sharpen their programming skills and understanding ASP.NET MVC in a short time.

This book is not only the ASP.NET MVC interview book but it is more than that. This book helps you to get the depth knowledge of ASP.NET MVC with a simple and elegant way. This book is updated to the latest version of ASP.NET MVC5.

I hope you will enjoy this book and find it useful. At the same time, I also encourage you to become a continuing reader of my blog [www.dotnettricks.com](http://www.dotnettricks.com) and be the part of the discussion. But most importantly practice a lot and enjoy the technology. That's what it's all about.

To get the latest information on ASP.NET MVC, I encourage you to follow the official Microsoft ASP.NET community website at [www.asp.net](http://www.asp.net). I also encourage you to subscribe to my blogs at [www.dotnettricks.com](http://www.dotnettricks.com) that contains .NET, C#, ASP.NET MVC, EF, jQuery and many more tips, tricks and tutorials.

**All the best for your interview and happy programming!**

# About the Author

Shailendra Chauhan - An Entrepreneur, Author, Architect, Corporate Trainer, and Microsoft MVP



He is the **Founder and CEO** of DotNetTricks which is a brand when it comes to e-Learning. DotNetTricks provides training and consultation over an array of technologies like **Cloud, .NET, Angular, React, Node and Mobile Apps development**. He has been awarded as **Microsoft MVP** three times in a row (2016-2018).

He has changed many lives from his writings and unique training programs. He has a number of most sought-after books to his name which have helped job aspirants in **cracking tough interviews** with ease.

Moreover, and to his credit, he has delivered **1000+ training sessions** to professionals worldwide in Microsoft .NET technologies and other technologies including JavaScript, AngularJS, Node.js, React and NoSQL Databases. In addition, he provides **Instructor-led online training, hands-on workshop** and **corporate training** programs.

**Shailendra** has a strong combination of **technical skills and solution development for complex application architecture with proven leadership and motivational skills** have elevated him to a world-renowned status, placing him at the top of the list of most sought-after trainers.

**"I always keep up with new technologies and learning new skills to deliver the best to my students,"** says Shailendra Chauhan, he goes on to acknowledge that the betterment of his followers and enabling his students to realize their goals are his prime objective and a great source of motivation and satisfaction.

Shailendra Chauhan - **"Follow me and you too will have the key that opens the door to success"**

# How to Contact Us

Although the author of this book has tried to make this book as accurate as it possible but if there is something strikes you as odd, or you find an error in the book please drop a line via e-mail.

The e-mail addresses are listed as follows:

- [mentor@dotnettricks.com](mailto:mentor@dotnettricks.com)
- [info@dotnettricks.com](mailto:info@dotnettricks.com)

We always happy to hear from our readers. Please provide your valuable feedback and comments!

You can follow us on [YouTube](#), [Facebook](#), [Twitter](#), [LinkedIn](#) and [Google Plus](#) or subscribe to [RSS feed](#).





# Table of Contents

<b>ASP.NET MVC Interview Questions &amp; Answers .....</b>	<b>1</b>
Release History .....	1
About Dot Net Tricks .....	2
Dot Net Tricks Training Solutions .....	2
Dedication.....	4
Introduction.....	5
About the Author.....	6
How to Contact Us.....	7
<b>ASP.NET MVC Introduction .....</b>	<b>12</b>
Q1.    What is MVC? .....	12
Q2.    Explain MVC design pattern? .....	12
Q3.    What is Domain Driven Design and Development? .....	13
Q4.    What is the MVP pattern? .....	13
Q5.    What is the MVVM pattern? .....	14
Q6.    What is ASP.NET MVC? .....	15
Q7.    How does MVC pattern work in ASP.NET MVC? .....	15
Q8.    How Model, View and Controller communicate with each other in ASP.NET MVC? .....	16
Q9.    What are the advantages of ASP.NET MVC? .....	17
Q10.   Explain the brief history of ASP.NET MVC? .....	17
Q11.   What is the difference between 3-layer architecture and MVC architecture?.....	19
Q12.   What is the difference between ASP.NET WebForm and ASP.NET MVC? .....	19
Q13.   What is ViewModel in ASP.NET MVC? .....	20
Q14.   Explain ASP.NET MVC pipeline? .....	20
<b>Routing and View Engines.....</b>	<b>24</b>
Q1.    What is Routing in ASP.NET MVC? .....	24
Q2.    How to define a route in ASP.NET MVC? .....	25
Q3.    What is Attribute Routing and how to define it? .....	26
Q4.    When to use Attribute Routing? .....	27
Q5.    How to enable Attribute Routing in ASP.NET MVC? .....	27
Q6.    How to define Attribute Routing for Area in ASP.NET MVC? .....	28

Q7.	What is the difference between Routing and URL Rewriting? .....	29
Q8.	What are Route Constraints in ASP.NET MVC? .....	29
Q9.	How route table is created in ASP.NET MVC? .....	30
Q10.	What are important namespaces in ASP.NET MVC? .....	30
Q11.	What is View Engine? .....	30
Q12.	How does View Engine work? .....	30
Q13.	What is Razor View Engine? .....	30
Q14.	How to make a Custom View Engine? .....	31
Q15.	How to register Custom View Engine in ASP.NET MVC? .....	32
Q16.	Can you remove default View Engine in ASP.NET MVC? .....	32
Q17.	What is the difference between Razor and WebForm engine? .....	32
<b>Helpers and Views .....</b>		<b>34</b>
Q1.	What are HTML Helpers in ASP.NET MVC? .....	34
Q2.	What are different types of HTML Helpers? .....	34
Q3.	What are Url Helpers? .....	38
Q4.	What is Validation Summary? .....	38
Q5.	What are AJAX Helpers? .....	38
Q6.	What is unobtrusive AJAX? .....	39
Q7.	What are various configuration options for AJAX Helpers? .....	39
Q8.	What is Cross-Domain AJAX? .....	39
Q9.	What are Layouts in ASP.NET MVC? .....	40
Q10.	What are Sections in ASP.NET MVC? .....	40
Q11.	What are RenderBody and RenderPage in ASP.NET MVC? .....	41
Q12.	What are Styles.Render and Scripts.Render? .....	41
Q13.	How to enable and disable optimizations in ASP.NET MVC? .....	42
Q14.	What is ViewStart? .....	42
Q15.	When to use _ViewStart? .....	42
Q16.	What are different ways of rendering layout in ASP.NET MVC? .....	42
Q17.	What is the App_Start folder in ASP.NET MVC? .....	44
Q18.	What are different ways of returning/rendering a view in ASP.NET MVC? .....	44
Q19.	What are differences among ViewData, ViewBag, TempData and Session? .....	45

Q20.	How to persist data in TempData? .....	46
Q21.	How to control Session behaviour in ASP.NET MVC? .....	47
Q22.	How TempData is related to Session in ASP.NET MVC? .....	48
Q23.	What are Action methods in ASP.NET MVC? .....	48
Q24.	What is ActionResult and how is it different from others? .....	49
Q25.	How to make a Non-Action method in ASP.NET MVC? .....	49
Q26.	Can you change the action method name? .....	49
Q27.	How to restrict an action method to be invoked only by HTTP GET, POST, PUT or DELETE? .....	50
Q28.	How to determine an action method is invoked by HTTP GET or POST? .....	50
Q29.	How to determine an AJAX request? .....	51
<b>Forms and Validations .....</b>		<b>52</b>
Q1.	What are Data Annotations in ASP.NET MVC? .....	52
Q2.	How to apply Server-side validation in ASP.NET MVC? .....	52
Q3.	How to determine there is no error in Model State? .....	53
Q4.	How to enable and disable client-side validation in ASP.NET MVC? .....	54
Q5.	What is a CDN and advantages of CDN? .....	54
Q6.	What is jquery.validate.unobtrusive.js? .....	55
Q7.	What is Bundling and Minification in ASP.NET MVC? .....	55
Q8.	Can we use Bundling and Minification in ASP.NET MVC3 or ASP.NET4.0? .....	57
Q9.	How Bundling use browser Cache capability? .....	57
Q10.	What is Partial View in ASP.NET MVC? .....	57
Q11.	How do you return a partial view from the controller? .....	58
Q12.	What are different ways of rendering a Partial View in ASP.NET MVC? .....	58
Q13.	What is Area in ASP.NET MVC? .....	59
Q14.	How to register Area in ASP.NET MVC? .....	60
Q15.	What is Child action and how to invoke it? .....	60
Q16.	What is Scaffolding? .....	60
Q17.	How do Scaffold templates work in ASP.NET MVC? .....	60
<b>Security and Filters .....</b>		<b>63</b>
Q1.	What are ASP.NET MVC Filters and Attributes? .....	63
Q2.	What are different types of Filters in ASP.NET MVC? .....	63

Q3.	When Exception filters are executed in ASP.NET MVC? .....	65
Q4.	What is the order of execution of filters in ASP.NET MVC? .....	65
Q5.	How to configure filters in ASP.NET MVC? .....	65
Q6.	How do Authentication and Authorization work in ASP.NET MVC? .....	66
Q7.	How Forms Authentication and Authorization work in ASP.NET MVC? .....	66
Q8.	How to implement custom Forms Authentication and Authorization in MVC? .....	67
Q9.	How to allow HTML tags in ASP.NET MVC? .....	70
Q10.	What is caching and when to use it? .....	71
Q11.	What are the advantages of caching? .....	71
Q12.	What is output caching? .....	71
Q13.	What is Donut caching and Donut hole caching in ASP.NET MVC? .....	72
<b>Dependency Injection .....</b>		<b>74</b>
Q1.	What is loose coupling and how is it possible? .....	74
Q2.	What are the Dependency Inversion Principle (DIP) and IoC? .....	74
Q3.	What is Dependency Injection (DI)? .....	75
Q4.	What is Service Locator? .....	76
Q5.	What are different ways to implement Dependency Injection (DI)? .....	78
Q6.	What are the advantages of Dependency Injection (DI)? .....	81
Q7.	What is IoC or DI container? .....	81
Q8.	What are popular DI containers? .....	81
Q9.	What is Test Driven Development (TDD)? .....	82
Q10.	What are the commonly used tool for Unit Testing in ASP.NET MVC? .....	82
<b>References .....</b>		<b>83</b>

# ASP.NET MVC Introduction

## Q1. What is MVC?

**Ans.** MVC stands for Model-View-Controller. It is a software design pattern which was introduced in the 1970s. Also, MVC pattern forces a *separation of concerns*, it means domain model and controller logic are decoupled from the user interface (view). As a result, maintenance and testing of the application become simpler and easier.

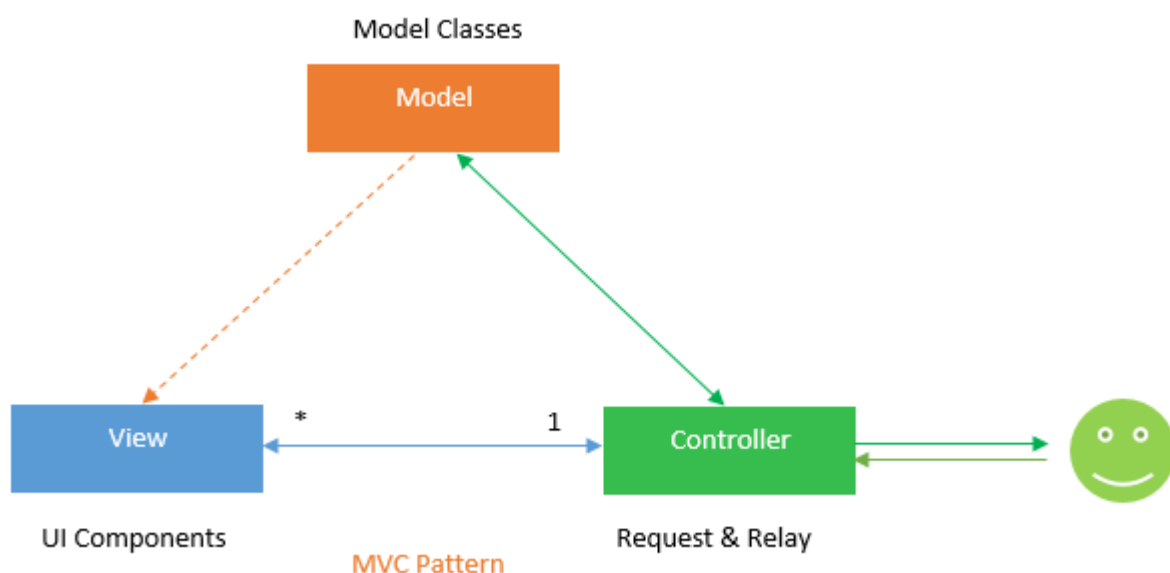
## Q2. Explain MVC design pattern?

**Ans.** MVC design pattern splits an application into three main aspects: Model, View and Controller

**Model** - The Model represents a set of classes that describe the business logic i.e. business model as well as data access operations i.e. data model. It also defines business rules for data means how the data can be changed and manipulated.

**View** - The View represents the UI components like CSS, jQuery, HTML etc. It is only responsible for displaying the data that is received from the controller as the result. This also transforms the model(s) into UI.

**Controller** - The Controller is responsible to process incoming requests. It receives input from users via the View, then processes the user's data with the help of Model and passing the results back to the View. Typically, it acts as the coordinator between the View and the Model.



Today, this pattern is used by many popular frameworks like as Ruby on Rails, Spring Framework, Apple iOS Development and ASP.NET MVC.

### Q3. What is Domain Driven Design and Development?

**Ans.** Domain-Driven Design (DDD) is a collection of principles and patterns that help developers to make design decisions to develop elegant systems for different domains. It is not a technology or methodology.

The main components of DDD are Entity, Value Object, Aggregate, Service and Repository.

**Entity-** An object that has an identity- it is unique within the system, like Customer, Employee etc.

**Value Object-** An object that has no identity within the system like Rate, State etc.

**Note:** A value object can become an entity depending on the situation.

**Aggregate:** An aggregate root is a special kind of entity that consumers refer to directly. All consumers of the aggregate root are called as aggregate. The aggregate root guarantees the consistency of changes being made within the aggregate.

**Service-** A service is a way of dealing with actions, operations and activities within your application.

**Repository-** A repository is responsible to store and to retrieve your data. It is not a concern how and where data will be persisted. So, it can be SQL server, Oracle, XML, text file or anything else. The repository is not a Data Access Layer but it refers to a location for storage, often for safety or preservation.

For more info refer this link <http://msdn.microsoft.com/en-us/magazine/dd419654.aspx>

### Q4. What is the MVP pattern?

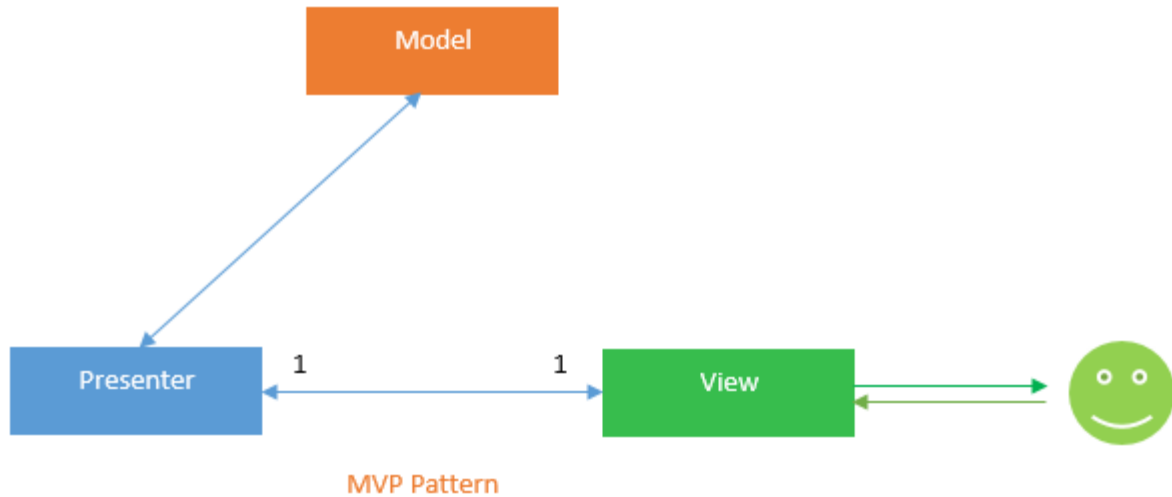
**Ans.** This pattern is similar to MVC pattern in which the controller has been replaced by the presenter. This design pattern splits an application into three main aspects: Model, View and Presenter.

**Model** - The Model represents a set of classes that describe the business logic and data. It also defines business rules for data means how the data can be changed and manipulated.

**View** - The View represents the UI components like CSS, jQuery, HTML etc. It is only responsible for displaying the data that is received from the presenter as the result. This also transforms the model(s) into UI.

**Presenter** - The Presenter is responsible for handling all UI events on behalf of the view. This receive input from users via the View, then process the user's data with the help of Model and passing the results back to the View. Unlike view and controller, view and presenter are completely decoupled from each other's and communicate to each others by an interface.

Also, the presenter does not manage the incoming request traffic as a controller.



This pattern is commonly used with ASP.NET Web Forms applications which require to create automated unit tests for their code-behind pages. This is also used with windows forms.

#### Key Points about MVP Pattern

1. The user interacts with the View.
2. There is a one-to-one relationship between View and Presenter means one View is mapped to only one Presenter.
3. The view has a reference to Presenter but View has no reference to Model.
4. Provides two-way communication between View and Presenter.

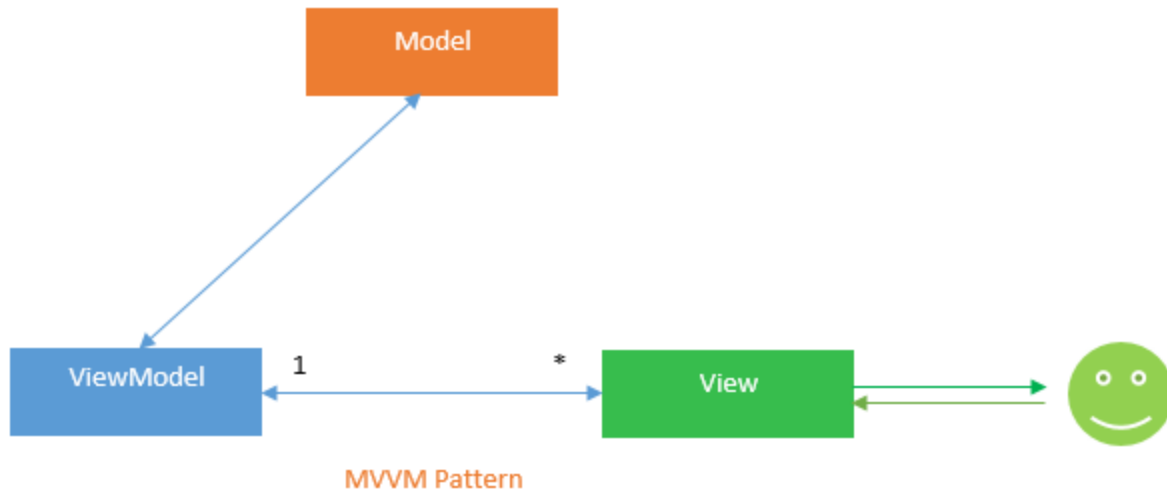
#### **Q5. What is the MVVM pattern?**

**Ans.** MVVM stands for Model-View-View Model. This pattern supports two-way data binding between view and View model. This enables automatic propagation of changes, within the state of view model to the View. Typically, the view model uses the observer pattern to notify changes in the view model to model.

**Model** - The Model represents a set of classes that describe the business logic and data. It also defines business rules for data means how the data can be changed and manipulated.

**View** - The View represents the UI components like CSS, jQuery, HTML etc. It is only responsible for displaying the data that is received from the controller as the result. This also transforms the model(s) into UI.

**View Model** - The View Model is responsible for exposing methods, commands, and other properties that help to maintain the state of the view, manipulate the model as the result of actions on the view, and trigger events in the view itself.



This pattern is commonly used by the WPF, Silverlight, Caliburn, nRoute etc.

#### Key Points about MVVM Pattern

1. The user interacts with the View.
2. There is a many-to-one relationship between View and ViewModel means many View can be mapped to one ViewModel.
3. The view has a reference to ViewModel but View Model has no information about the View.
4. Supports two-way data binding between View and ViewModel.

#### **Q6. What is ASP.NET MVC?**

**Ans.** ASP.NET MVC is an open source framework built on the top of Microsoft .NET Framework to develop a web application that enables a clean separation of code. ASP.NET MVC framework is the most customizable and extensible platform shipped by Microsoft.

#### **Q7. How does MVC pattern work in ASP.NET MVC?**

**Ans.** Working of MVC pattern in ASP.NET MVC is explained as below:

##### The Model in ASP.NET MVC

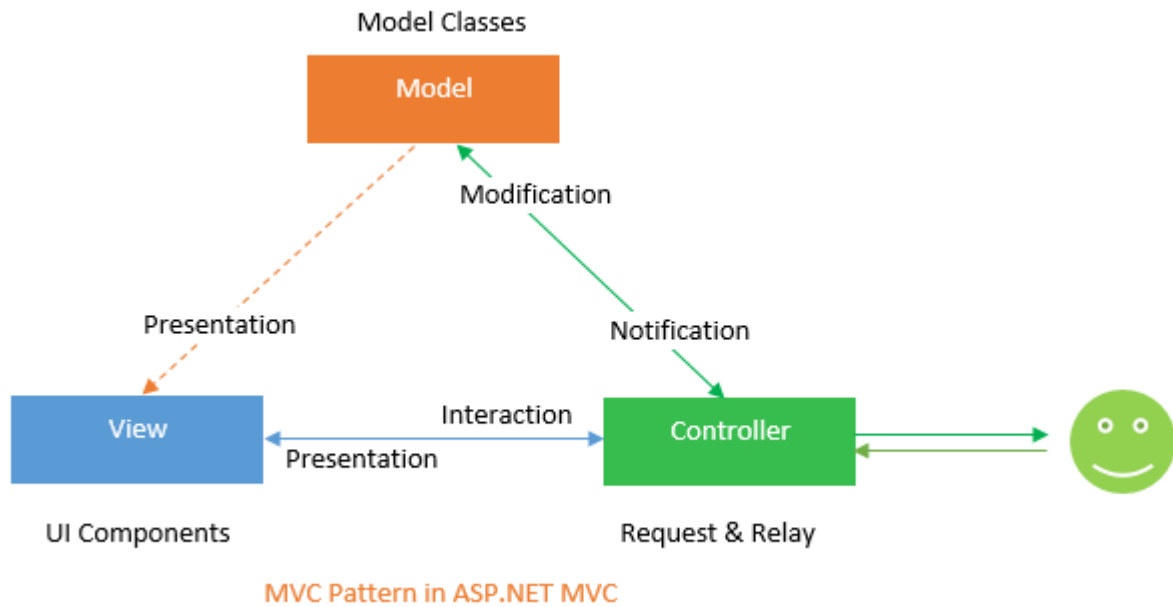
The Model in ASP.NET MVC can be broken down into several different layers as given below:

1. **Objects or ViewModel or Presentation Layer** - This layer contains simple objects or complex objects which are used to specify a strongly-typed view. These objects are used to pass data from controller to strongly-typed view and vice versa. The classes for these objects can have specific validation rules which are defined by using data annotations. Typically, these classes have those properties which you want to display on corresponding view/page.
2. **Business Layer** - This layer helps you to implement your business logic and validations for your application. This layer makes use of Data Access Layer for persisting data into the database. Also, this layer is directly invoked by the Controller to do processing on input data and sent back to view.



- 3. Data Access Layer** - This layer provides objects to access and manipulate the database of your application. Typically, this layer is made by using ORM tools like Entity Framework or NHibernate etc.

By default, models are stored in the Models folder of an ASP.NET MVC application.



### The View in ASP.NET MVC

The view is only responsible for displaying the data that is received from the controller as a result. It also responsible for transforming a model or models into UI which provide all the required business logic and validation to the view.

By default, views are stored in the Views folder of an ASP.NET MVC application.

### The Controller in ASP.NET MVC

The Controller in ASP.NET MVC, respond to HTTP requests and determine the action to take based upon the content of the incoming request. It receives input from users via the View, then processes the user's data with the help of Model and passing the results back to the View.

By default, controllers have stored in the Controllers folder an ASP.NET MVC application.

## **Q8. How Model, View and Controller communicate with each other in ASP.NET MVC?**

**Ans.** There are following rules for communication among Model, View and Controller:

1. The user interacts with the Controller.
2. There is a one-to-many relationship between Controller and View means one controller can be mapped to multiple views.
3. Controller and View can have a reference to the model.
4. Controller and View can talk to each other.

5. Model and View cannot talk to each other directly. They communicate with each other with the help of a controller.

### Q9. What are the advantages of ASP.NET MVC?

**Ans.** There are following advantages of ASP.NET MVC over Web Forms (ASP.NET):

- **Separation of concern** - MVC design pattern divides the ASP.NET MVC application into three main aspects Model, View and Controller which make it easier to manage the application complexity.
- **TDD** - The MVC framework brings better support to test-driven development.
- **Extensible and pluggable** - MVC framework components were designed to be pluggable and extensible and therefore can be replaced or customized easier than Web Forms.
- **Full control over application behaviour** - MVC framework doesn't use View State or server-based forms like Web Forms. This gives the application developer more control over the behaviours of the application and also reduces the bandwidth of requests to the server.
- **ASP.NET features are supported** - MVC framework is built on top of ASP.NET and therefore can use most of the features that ASP.NET include such as the provider's architecture, authentication and authorization scenarios, membership and roles, caching, session and more.
- **URL routing mechanism** - MVC framework supports a powerful URL routing mechanism that helps to build a more comprehensible and searchable URLs in your application. This mechanism helps the application to be more addressable from the eyes of search engines and clients and can help in search engine optimization.

### Q10. Explain the brief history of ASP.NET MVC?

**Ans.** Here is the list of the released version history of ASP.NET MVC Framework with their features.

#### ASP.NET MVC1

- Released on Mar 13, 2009
- Runs on .NET 3.5 and with Visual Studio 2008 & Visual Studio 2008 SP1
- MVC Pattern architecture with WebForm Engine
- Html Helpers
- Ajax helpers
- Routing
- Unit Testing

#### ASP.NET MVC2

- Released on Mar 10, 2010
- Runs on .NET 3.5, 4.0 and with Visual Studio 2008 & 2010
- Strongly typed HTML helpers means lambda expression based Html Helpers
- Templated Helpers
- UI helpers with automatic scaffolding & customizable templates
- Support for DataAnnotations Attributes to apply model validation on both client and server sides

- Overriding the HTTP Method Verb including GET, PUT, POST, and DELETE
- Areas for partitioning a large application into modules
- Asynchronous controllers

### **ASP.NET MVC3**

- Released on Jan 13, 2011
- Runs on .NET 4.0 and with Visual Studio 2010
- The Razor view engine
- Enhanced Data Annotations attributes for model validation on both client and server sides
- Remote Validation
- Compare Attribute
- Sessionless Controller
- Child Action Output Caching
- Dependency Resolver
- Entity Framework Code First support
- Partial-page output caching
- ViewBag dynamic property for passing data from the controller to view
- 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

### **ASP.NET MVC4**

- Released on Aug 15, 2012
- Runs on .NET 4.0, 4.5 and with Visual Studio 2010SP1 & Visual Studio 2012
- 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
- Support for the Windows Azure SDK

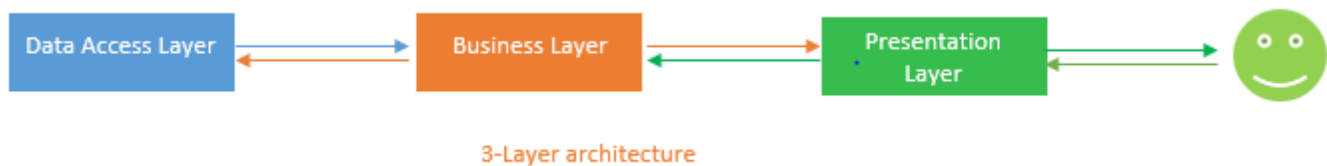
### **ASP.NET MVC5**

- Released on 17 October 2013
- Runs on .NET 4.5, 4.5.1 and with Visual Studio 2012 & Visual Studio 2013
- One ASP.NET
- ASP.NET Identity

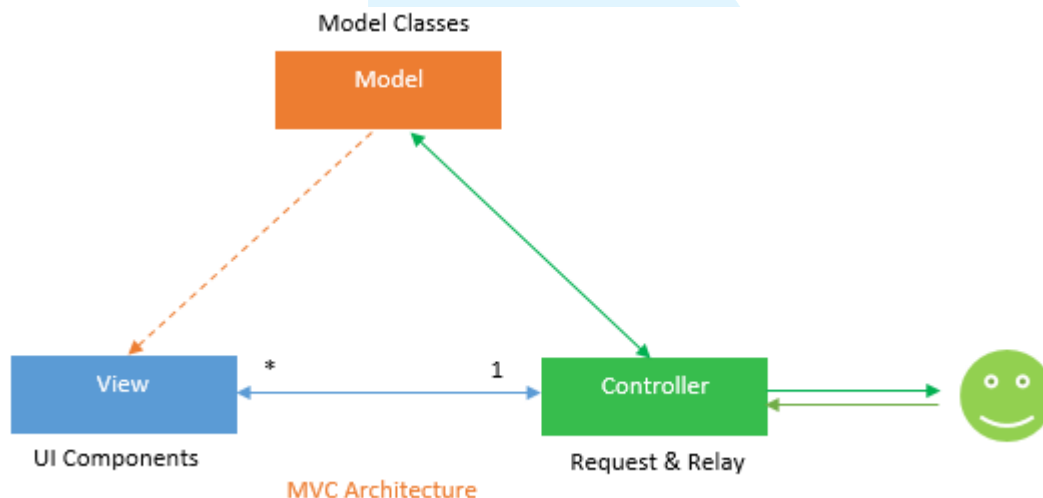
- ASP.NET Scaffolding
- Authentication filters - run prior to authorization filters in the ASP.NET MVC pipeline
- Bootstrap in the MVC template
- ASP.NET WEB API2

### Q11. What is the difference between 3-layer architecture and MVC architecture?

**Ans.** 3-layer architecture separates the application into 3 components which consist of Presentation Layer Business Layer and Data Access Layer. In 3-layer architecture, the user interacts with the Presentation layer. 3-layer is a linear architecture.



MVC architecture separates the application into three components which consist of Model, View and Controller. In MVC architecture, the user interacts with the controller with the help of view. MVC is a triangle architecture.



MVC does not replace 3-layer architecture. Typically, 3-layer and MVC are used together and MVC acts as the Presentation layer.

### Q12. What is the difference between ASP.NET WebForm and ASP.NET MVC?

**Ans.** The main differences between ASP.NET Web Form and ASP.NET MVC are given below:

ASP.NET Web Forms	ASP.NET MVC
ASP.NET Web Form follows a traditional event-driven development model.	ASP.NET MVC is a lightweight and follows MVC (Model, View, and Controller) pattern based development, model.
ASP.NET Web Form has server controls.	ASP.NET MVC has HTML helpers.
ASP.NET Web Form has state management (like as view state, session) techniques.	ASP.NET MVC has no automatic state management techniques.

ASP.NET Web Form has file-based URLs means file name exist in the URLs must have its physical existence.	ASP.NET MVC has route-based URLs means URLs are divided into controllers and actions and moreover it is based on controller not on physical file.
ASP.NET Web Form follows WebForm Syntax	ASP.NET MVC follow customizable syntax (Razor as default)
In ASP.NET Web Form, Web Forms (ASPX) i.e. views are tightly coupled to Code behind (ASPX.CS) i.e. logic.	In ASP.NET MVC, Views and logic are kept separately.
ASP.NET Web Form has Master Pages for a consistent look and feels.	ASP.NET MVC has Layouts for a consistent look and feels.
ASP.NET Web Form has User Controls for code re-usability.	ASP.NET MVC has Partial Views for code re-usability.
ASP.NET Web Form has built-in data controls and best for rapid development with powerful data access.	ASP.NET MVC is lightweight, provide full control over markup and support many features that allow fast & agile development. Hence it is best for developing interactive web application with the latest web standards.
ASP.NET Web Form is not Open Source.	ASP.NET Web MVC is an Open Source.

### Q13. What is ViewModel in ASP.NET MVC?

**Ans.** In ASP.NET MVC, ViewModel is a class that contains the fields which are represented in the strongly-typed view. It is used to pass data from the controller to view.

#### Key Points about ViewModel

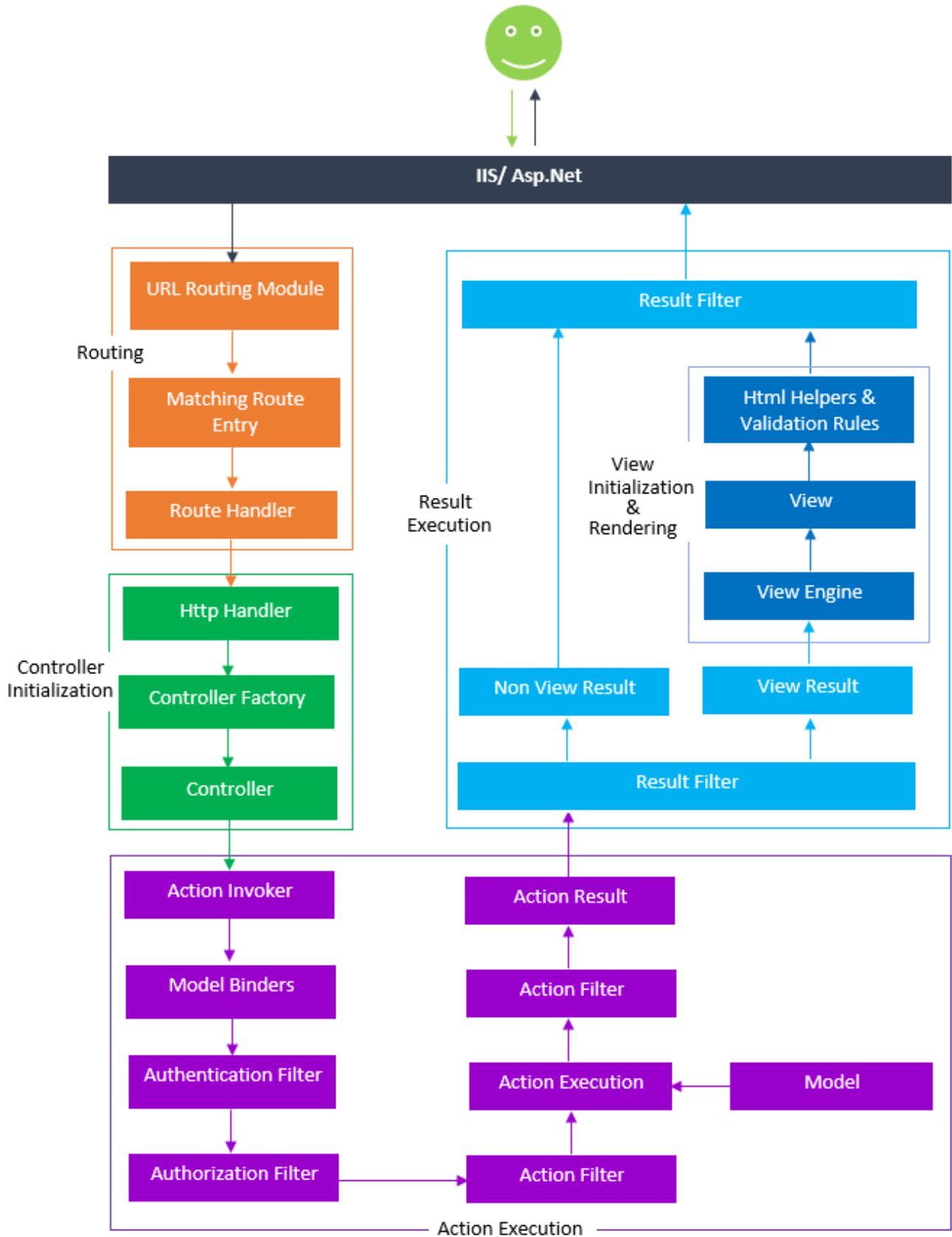
- ViewModel contain fields that are represented in the view (for LabelFor, EditorFor, DisplayFor helpers)
- ViewModel can have specific validation rules using data annotations.
- ViewModel can have multiple entities or objects from different data models or data source.

### Q14. Explain ASP.NET MVC pipeline?

**Ans.** The detail ASP.NET MVC pipeline is given below:

1. **Routing** - Routing is the first step in ASP.NET MVC pipeline. Typically, it is a pattern matching system that matches the incoming request to the registered URL patterns in the Route Table.

The `UrlRoutingModule(System.Web.Routing.UrlRoutingModule)` is a class which matches an incoming HTTP request to a registered route pattern in the `RouteTable(System.Web.Routing.RouteTable)`.



ASP.NET MVC Pipeline

- 2. Controller Initialization** - The MvcHandler initiates the real processing inside ASP.NET MVC pipeline by using the `ProcessRequest` method. This method uses the `IControllerFactory` instance (default is `System.Web.Mvc.DefaultControllerFactory`) to create corresponding controller.
- 3. Action Execution** – Action execution occurs in the following steps:
- When the controller is initialized, the controller calls its own `InvokeAction()` method by passing the details of the chosen action method. This is handled by the `IActionInvoker`.
  - After chosen of appropriate action method, model binders (default is `System.Web.Mvc.DefaultModelBinder`) retrieves the data from incoming HTTP request and do the data type conversion, data validation such as required or date format etc. and also take care of input values mapping to that action method parameters.
  - Authentication Filter was introduced with ASP.NET MVC5 that run prior to authorization filter. It is used to authenticate a user. Authentication filter process user credentials in the request and provide a corresponding principal. Prior to ASP.NET MVC5, you use authorization filter for authentication and authorization to a user.
  - By default, `Authorize` attribute is used to perform Authentication. You can easily create your own custom authentication filter by implementing `IAuthorizationFilter`.
  - Authorization filter allows you to perform the authorization process for an authenticated user. For example, Role based authorization for users to access resources.
  - By default, `Authorize` attribute is used to perform authorization. You can also make your own custom authorization filter by implementing `IAuthorizationFilter`.
  - Action filters are executed before (`OnActionExecuting`) and after (`OnActionExecuted`) an action is executed. `IActionFilter` interface provides you two methods `OnActionExecuting` and `OnActionExecuted` methods which will be executed before and after an action gets executed respectively. You can also make your own custom ActionFilters filter by implementing `IActionFilter`. For more about filters refer this article [Understanding ASP.NET MVC Filters and Attributes](#)
  - When the action is executed, it processes the user inputs with the help of model (Business Model or Data Model) and prepare Action Result.
- 4. Result Execution** - Result execution occurs in the following steps:
- Result filters are executed before (`OnResultExecuting`) and after (`OnResultExecuted`) the `ActionResult` is executed. `IResultFilter` interface provides you two methods `OnResultExecuting` and `OnResultExecuted` methods which will be executed before and after an `ActionResult` gets executed respectively. You can also make your own custom ResultFilters filter by implementing `IResultFilter`.
  - Action Result is prepared by performing operations on user inputs with the help of BAL or DAL. The Action Result type can be `ViewResult`, `PartialViewResult`, `RedirectToRouteResult`, `RedirectResult`, `ContentResult`, `JsonResult`, `FileResult` and `EmptyResult`.
  - Various Result type provided by the ASP.NET MVC can be categorized into two category- `ViewResult` type and `NonViewResult` type. The Result type which renders and returns an HTML page to the browser falls into `ViewResult` category and another result type which returns only data either in text format, binary format or a JSON format, falls into `NonViewResult` category.

#### 4.1 View Initialization and Rendering - View Initialization and Rendering execution occur in the following steps:

- ViewResult type i.e. view and partial view are represented by IView (System.Web.Mvc.IView) interface and rendered by the appropriate View Engine.
- This process is handled by the IViewEngine (System.Web.Mvc.IViewEngine) interface of the view engine. By default, ASP.NET MVC provides WebForm and Razor view engines. You can also create your custom engine by using IViewEngine interface and can register your custom view engine into your ASP.NET MVC application as shown below:
- Html Helpers are used to writing input fields, create links based on the routes, AJAX-enabled forms, links and much more. Html Helpers are extension methods of the HtmlHelper class and can be further extended very easily. In a more complex scenario, it might render a form with client-side validation with the help of JavaScript or jQuery.



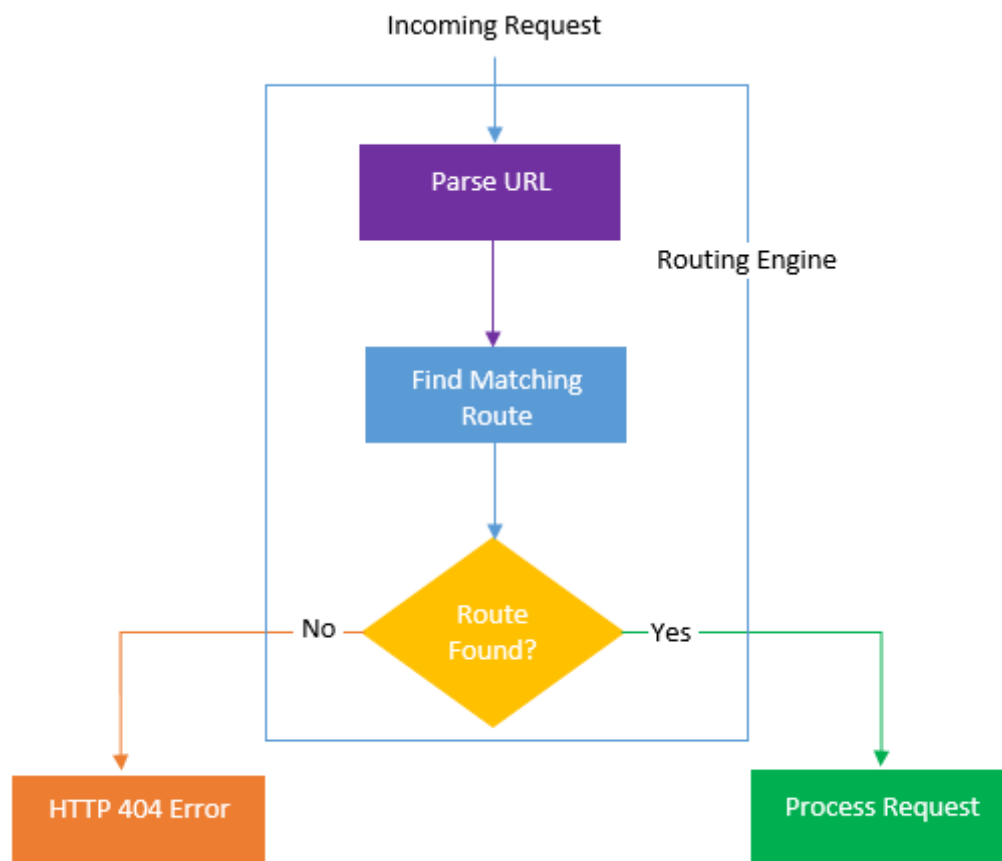


# 2

## Routing and View Engines

### Q1. What is Routing in ASP.NET MVC?

**Ans.** Routing is a pattern matching system that monitors the incoming request and figure out what to do with that request. At runtime, the Routing engine uses the Route table for matching the incoming request's URL pattern against the URL patterns defined in the Route table. You can register one or more URL patterns to the Route table at *Application\_Start* event.



Incoming Request Processing by Routing System

When the routing engine finds a match in the route table for the incoming request's URL, it forwards the request to the appropriate controller and action. If there is no match in the route table for the incoming request's URL, it returns a 404 HTTP status code.

## Q2. How to define a route in ASP.NET MVC?

Ans. You can define a route in ASP.NET MVC as given below:

```
public static void RegisterRoutes(RouteCollection routes)
{
    routes.MapRoute(
        "Default", // Route name
        "{controller}/{action}/{id}", // Route Pattern
        new
        {
            controller = "Home",
            action = "Index",
            id = UrlParameter.Optional
        } // Default values for above defined parameters
    );
}
protected void Application_Start()
{
    RegisterRoutes(RouteTable.Routes);
    //TODO:
}
```

Always remember route name should be unique across the entire application. Route name can't be duplicate.

In the above example we have defined the Route Pattern {controller}/{action}/{id} and also provide the default values for controller, action and id parameters. Default values mean if you will not provide the values for controller or action or id defined in the pattern then these values will be served by the routing system.

Suppose your web application is running on [www.example.com](http://www.example.com) then the URL pattern for your application will be [www.example.com/{controller}/{action}/{id}](http://www.example.com/{controller}/{action}/{id}). Hence you need to provide the controller name followed by action name and id if it is required. If you will not provide any of the value then the default values of these parameters will be provided by the routing system. Here is a list of URLs that match and don't match this route pattern.

Request URL	Parameters
<a href="http://example.com/">http://example.com/</a>	controller=Home, action=Index, id=none, since default value of controller and action are Home and Index respectively.
<a href="http://example.com/Admin">http://example.com/Admin</a>	controller=Admin, action=Index, id=none, since default value of action is Index
<a href="http://example.com/Admin/Product">http://example.com/Admin/Product</a>	controller=Admin, action=Product, id=none
<a href="http://example.com/Admin/Product/1">http://example.com/Admin/Product/1</a>	controller=Admin, action=Product, id=1
<a href="http://example.com/Admin/Product/SubAdmin/1">http://example.com/Admin/Product/SubAdmin/1</a>	No Match Found
<a href="http://example.com/Admin/Product/SubAdmin/Add/1">http://example.com/Admin/Product/SubAdmin/Add/1</a>	No Match Found

**Note:** Always put the more specific route on the top order while defining the routes, since routing system checks the incoming URL pattern from the top and as it gets the matched route it will consider that. It will not be checked further routes after matching pattern.

### Q3. What is Attribute Routing and how to define it?

**Ans.** ASP.NET MVC5 and WEB API 2 supports a new type of routing, called attribute routing. In this routing, attributes are used to define routes. Attribute routing provides you with more control over the URIs by defining routes directly on actions and controllers in your ASP.NET MVC application and WEB API.

1. **Controller level routing** – You can define routes at controller level which apply to all actions within the controller unless a specific route is added to an action.

```
[RoutePrefix("MyHome")]
[Route("{action=index}")] //default action
public class HomeController: Controller
{
    //new route: /MyHome/Index
    public ActionResult Index()
    {
        return View();
    }

    //new route: /MyHome/About
    public ActionResult About()
    {
        ViewBag.Message = "Your application description page.";
        return View();
    }

    //new route: /MyHome/Contact
    public ActionResult Contact()
    {
        ViewBag.Message = "Your contact page.";
        return View();
    }
}
```

2. **Action level routing** – You can define routes at action level which apply to a specific action within the controller.

```
public class HomeController: Controller
{
    [Route("users/{id:int:min(100)}")] //route: /users/100
    public ActionResult Index(int id)
    {
        //TO DO:
        return View();
    }

    [Route("users/about")] //route" /users/about
    public ActionResult About()
    {
        ViewBag.Message = "Your application description page.";
    }
}
```

```

        return View();
    }

    //route: /Home/Contact
    public ActionResult Contact()
    {
        ViewBag.Message = "Your contact page.";
        return View();
    }
}

```

#### Note:

- Attribute routing should configure before the convention-based routing.
- When you combine attribute routing with convention-based routing, actions which do not have **Route** attribute for defining **attribute-based** routing will work according to **convention-based** routing. In the above example, **Contact** action will work according to **convention-based** routing.
- When you have only attribute routing, actions which do not have **Route** attribute for defining **attribute-based** routing will not be the part of attribute routing. In this way, they can't be accessed from outside as a URI.

#### Q4. When to use Attribute Routing?

**Ans.** The convention-based routing is complex to support certain URI patterns that are common in RESTful APIs. But by using attribute routing you can define these URI patterns very easily.

**For example,** resources often contain child resources like Clients have ordered, movies have actors, books have authors and so on. It's natural to create URIs that reflects these relations like as */clients/1/orders*

This type of URI is difficult to create using convention-based routing. Although it can be done, the results don't scale well if you have many controllers or resource types.

With attribute routing, it's pretty much easy to define a route for this URI. You simply add an attribute to the controller action as:

```

[Route("clients/{clientId}/orders")]
public IEnumerable<Order> GetOrdersByClient(int clientId)
{
    //TO DO
}

```

#### Q5. How to enable Attribute Routing in ASP.NET MVC?

**Ans.** Enabling attribute routing in your ASP.NET MVC5 application is simple, just add a call to *routes.MapMvcAttributeRoutes()* method within *RegisterRoutes()* method of *RouteConfig.cs* file.

```

public class RouteConfig
{
    public static void RegisterRoutes(RouteCollection routes)
    {
        routes.IgnoreRoute("{resource}.axd/{*pathInfo}");
    }
}

```

```

        //enabling attribute routing
        routes.MapMvcAttributeRoutes();
    }
}

```

You can also combine attribute routing with convention-based routing.

```

public class RouteConfig
{
    public static void RegisterRoutes(RouteCollection routes)
    {
        routes.IgnoreRoute("{resource}.axd/{*pathInfo}");

        //enabling attribute routing
        routes.MapMvcAttributeRoutes();

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

```

## Q6. How to define Attribute Routing for Area in ASP.NET MVC?

**Ans.** You can also define attribute routing for a controller that belongs to an area by using the *RouteArea* attribute. When you define attribute routing for all controllers within an area, you can safely remove the *AreaRegistration* class for that area.

```

[RouteArea("Admin")]
[RoutePrefix("menu")]
[Route("{action}")]
public class MenuController: Controller
{
    // route: /admin/menu/login
    public ActionResult Login()
    {
        return View();
    }

    // route: /admin/menu/products
    [Route("products")]
    public ActionResult GetProducts()
    {
        return View();
    }

    // route: /categories

```

```
[Route("~/categories")]
public ActionResult Categories()
{
    return View();
}
```

## Q7. What is the difference between Routing and URL Rewriting?

**Ans.** Many developers compare routing to URL rewriting since both look similar and can be used to make SEO friendly URLs. But both the approaches are very much different. The main difference between routing and URL rewriting is given below:

- URL rewriting is focused on mapping one URL (new URL) to another URL (old URL) while routing is focused on mapping a URL to a resource.
- URL rewriting rewrites your old URL to a new one while routing never rewrite your old URL to the new one but it maps to the original route.

## Q8. What are Route Constraints in ASP.NET MVC?

**Ans.** Route constraints are a way to put some validation around the defined route.

### Creating Route Constraints

Suppose we have defined the following route in our application and you want to restrict the incoming request URL with numeric id only. Now let's see how to do it with the help of regular expression.

```
public static void RegisterRoutes(RouteCollection routes)
{
    routes.MapRoute(
        "Default", // Route name
        "{controller}/{action}/{id}", // Route Pattern
        new
        {
            controller = "Home",
            action = "Index",
            id = UrlParameter.Optional
        } // Default values for parameters
    );
}
```

### Restrict to numeric id only

```
public static void RegisterRoutes(RouteCollection routes)
{
    routes.MapRoute(
        "Default", // Route name
        "{controller}/{action}/{id}", // Route Pattern
        new
        {
            controller = "Home",
```

```

        action = "Index",
        id = UrlParameter.Optional
    }, // Default values for parameters
    new { id = @"\d+" } //Restriction for id
);
}

```

Now for this route, routing engine will consider only those URLs which have only numeric id like as `http://example.com/Admin/Product/1` else it will consider that URL is not matched with this route.

### Q9. How route table is created in ASP.NET MVC?

**Ans.** 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.

### Q10. What are important namespaces in ASP.NET MVC?

**Ans.** There are some important namespaces as given below:

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

### Q11. What is View Engine?

**Ans.** A View Engine is an MVC subsystem which has its own markup syntax. It is responsible for converting the server-side template into HTML markup and rendering it to the browser. Initially, ASP.NET MVC ships with one view engine, web forms (ASPX) and from ASP.NET MVC3 a new view engine, Razor is introduced. With ASP.NET MVC, you can also use other view engines like Spark, NHaml etc.

### Q12. How does View Engine work?

**Ans.** Each view engine has the following three main components:

1. **ViewEngine class** - This class implements the `IViewEngine` interface and responsible for locating view templates.
2. **View class** - This class implements the `IView` interface and responsible for combining the template with data from the current context and convert it to output HTML markup.
3. **Template parsing engine** - This parses the template and compiles the view into executable code.

### Q13. What is Razor View Engine?

**Ans.** Razor Engine is an advanced view engine that was introduced with MVC3. This is not a new language but it is a new markup syntax. Razor has new and advances syntax that is compact, expressive and reduces typing. Razor syntax is easy to learn and much cleaner than Web Form syntax. Razor uses `@` symbol to write markup as:

```
@Html.ActionLink("SignUp", "SignUp")
```

#### Q14. How to make a Custom View Engine?

**Ans.** ASP.NET MVC is an open source and highly extensible framework. You can create your own View engine by Implementing the *IViewEngine* interface or by inheriting *VirtualPathProviderViewEngine* abstract class.

```
public class CustomViewEngine : VirtualPathProviderViewEngine
{
    public CustomViewEngine()
    {
        // Define the location of the View and Partial View
        this.ViewLocationFormats = new string[] { "~/Views/{1}/{0}.html",
        "~/Views/Shared/{0}.html" };
        this.PartialViewLocationFormats = new string[] { "~/Views/{1}/{0}.html",
        "~/Views/Shared/{0}.html" };
    }

    protected override IView CreatePartialView(ControllerContext
controllerContext, string partialPath)
    {
        var physicalpath =
controllerContext.HttpContext.Server.MapPath(partialPath);
        return new CustomView(physicalpath);
    }

    protected override IView CreateView(ControllerContext controllerContext,
string viewPath, string masterPath)
    {
        var physicalpath =
controllerContext.HttpContext.Server.MapPath(viewPath);
        return new CustomView(physicalpath);
    }
}

public class CustomView: IView
{
    private string _viewPhysicalPath;

    public CustomView(string ViewPhysicalPath)
    {
        _viewPhysicalPath = ViewPhysicalPath;
    }

    public void Render(ViewContext viewContext, System.IO.TextWriter writer)
    {
        //Load File
        string rawcontents = File.ReadAllText(_viewPhysicalPath);

        //Perform Replacements
    }
}
```



```

        string parsedcontents = Parse(rawcontents, viewContext.ViewData);

        writer.Write(parsedcontents);
    }

    public string Parse(string contents, ViewDataDictionary viewdata)
    {
        return Regex.Replace(contents, "\\{(.+)\\}", m => GetMatch(m, viewdata));
    }

    public virtual string GetMatch(Match m, ViewDataDictionary viewdata)
    {
        if (m.Success)
        {
            string key = m.Result("$1");
            if (viewdata.ContainsKey(key))
            {
                return viewdata[key].ToString();
            }
        }
        return string.Empty;
    }
}

```

#### Q15. How to register Custom View Engine in ASP.NET MVC?

**Ans.** To use your custom View Engine, you need to register it by using global.asax.cs file Application\_Start() method, so that the framework will use your custom View Engine instead of the default one.

```

protected void Application_Start()
{
    //Register Custom View Engine
    ViewEngines.Engines.Add(new CustomViewEngine());

    //other code is removed for clarity
}

```

#### Q16. Can you remove default View Engine in ASP.NET MVC?

**Ans.** Yes, you can remove default view engines (Razor and WebForm) provided by ASP.NET MVC.

```

protected void Application_Start()
{
    //Remove All View Engine including Webform and Razor
    ViewEngines.Engines.Clear();
}

```

#### Q17. What is the difference between Razor and WebForm engine?

**Ans.** The main differences between ASP.NET Web Form and ASP.NET MVC are given below:

Razor View Engine	WebForm View Engine
Razor Engine is an advanced view engine that was introduced with MVC3. This is not a new language but it is a new markup syntax.	WebForm Engine is the default view engine for the Asp.net MVC that is included with Asp.net MVC from the beginning.
The namespace for Razor Engine is System.Web.Razor.	The namespace for WebForm Engine is System.Web.Mvc.WebFormViewEngine.
The file extensions used with Razor Engine are different from Web Form Engine. It has .cshtml (Razor with C#) or .vbhtml (Razor with VB) extension for views, partial views, editor templates and for layout pages.	The file extensions used with Web Form Engine are also like Asp.net Web Forms. It has .aspx extension for views, .ascx extension for partial views & editor templates and .master extension for layout/master pages.
Razor has new and advances syntax that is compact, expressive and reduces typing.	WebForm Engine has the same syntax as Asp.net Web Forms uses for .aspx pages.
Razor syntax is easy to learn and much cleaner than Web Form syntax. Razor uses @ symbol to make the code like as: @Html.ActionLink("SignUp", "SignUp")	WebForm syntax is borrowed from Asp.net Web Forms syntax that is mixed with HTML and sometimes makes a view messy. Webform uses <% and %> delimiters to make the code like as: <%: Html.ActionLink("SignUp", "SignUp") %>
By default, Razor Engine prevents XSS attacks (Cross-Site Scripting Attacks) means it encodes the script or HTML tags like <, > before rendering to view.	WebForm Engine does not prevent XSS attacks means any script saved in the database will be fired while rendering the page
Razor Engine is a little bit slow as compared to WebForm Engine.	WebForm Engine is faster than Razor Engine.
Razor Engine doesn't support design mode in the visual studio means you cannot see your page look and feel.	WebForm engine support design mode in the visual studio means you can see your page look and feel without running the application.
Razor Engine support TDD (Test Driven Development) since it does not depend on System.Web.UI.Page class.	Web Form Engine doesn't support TDD (Test Driven Development) since it depends on System.Web.UI.Page class which makes the testing complex.

# Helpers and Views

## Q1. What are HTML Helpers in ASP.NET MVC?

**Ans.** An HTML Helper is just a method that returns an HTML string. The string can represent any type of content that you want. **For example**, you can use HTML Helpers to render standard HTML tags like HTML <input>, <button> and <img> tags etc.

You can also create your own HTML Helpers to render more complex content such as a menu strip or an HTML table for displaying database data.

## Q2. What are different types of HTML Helpers?

**Ans.** There are three types of HTML helpers as given below:

1. **Inline Html Helpers** - These are created in the same view by using the Razor @helper tag. These helpers can be reused only on the same view.

```
@helper ListingItems(string[] items)
{
    <ol>
        @foreach (string item in items)
        {
            <li>@item</li>
        }
    </ol>
}

<h3>Programming Languages:</h3>

@ListingItems(new string[] { "C", "C++", "C#" })

<h3>Book List:</h3>

@ListingItems(new string[] { "How to C", "how to C++", "how to C#" })
```

2. **Built-In Html Helpers** - Built-In Html Helpers are extension methods on the HtmlHelper class. The Built-In Html helpers can be divided into three categories-
  - **Standard Html Helpers** - These helpers are used to render the most common types of HTML elements like as HTML text boxes, checkboxes etc. A list of most common standard HTML helpers is given below:

HTML Element	Example
TextBox	<pre>@Html.TextBox("Textbox1", "val")</pre> Output: <pre>&lt;input id="Textbox1" name="Textbox1" type="text" value="val" /&gt;</pre>
TextArea	<pre>@Html.TextArea("Textarea1", "val", 5, 15, null)</pre> Output: <pre>&lt;textarea cols="15" id="Textarea1" name="Textarea1" rows="5"&gt;val&lt;/textarea&gt;</pre>
Password	<pre>@Html.Password("Password1", "val")</pre> Output: <pre>&lt;input id="Password1" name="Password1" type="password" value="val" /&gt;</pre>
Hidden Field	<pre>@Html.Hidden("Hidden1", "val")</pre> Output: <pre>&lt;input id="Hidden1" name="Hidden1" type="hidden" value="val" /&gt;</pre>
CheckBox	<pre>@Html.CheckBox("Checkbox1", false)</pre> Output: <pre>&lt;input id="Checkbox1" name="Checkbox1" type="checkbox" value="true" /&gt;</pre> <pre>&lt;input name="myCheckbox" type="hidden" value="false" /&gt;</pre>
RadioButton	<pre>@Html.RadioButton("Radiobutton1", "val", true)</pre> Output: <pre>&lt;input checked="checked" id="Radiobutton1" name="Radiobutton1" type="radio" value="val" /&gt;</pre>
Drop-down list	<pre>@Html.DropDownList("DropDownList1", new SelectList(new [] {"Male", "Female"}))</pre> Output: <pre>&lt;select id="DropDownList1" name="DropDownList1"&gt;</pre> <pre>&lt;option&gt;M&lt;/option&gt;</pre> <pre>&lt;option&gt;F&lt;/option&gt;</pre> <pre>&lt;/select&gt;</pre>
Multiple-select	<pre>Html.ListBox("ListBox1", new MultiSelectList(new [] {"Cricket", "Chess"}))</pre> Output: <pre>&lt;select id="ListBox1" multiple="multiple" name="ListBox1"&gt;</pre> <pre>&lt;option&gt;Cricket&lt;/option&gt;</pre> <pre>&lt;option&gt;Chess&lt;/option&gt;</pre> <pre>&lt;/select&gt;</pre>

- **Strongly Typed HTML Helpers** - These helpers are used to render the most common types of HTML elements in strongly typed view like as HTML text boxes, checkboxes etc. The HTML elements are created based on model properties.

The strongly typed HTML helpers work on lambda expression. The model object is passed as a value to a lambda expression, and you can select the field or property from model object to be used to set the id, name and value attributes of the HTML helper. A list of most common strongly-typed HTML helpers is given below:

HTML Element	Example
TextBox	<pre>@Html.TextBoxFor(m=&gt;m.Name) Output: &lt;input id="Name" name="Name" type="text" value="Name-val" /&gt;</pre>
TextArea	<pre>@Html.TextArea(m=&gt;m.Address , 5, 15, new{}}) Output: &lt;textarea cols="15" id="Address" name=" Address " rows="5"&gt;Addressvalue&lt;/textarea&gt;</pre>
Password	<pre>@Html.PasswordFor(m=&gt;m.Password) Output: &lt;input id="Password" name="Password" type="password"/&gt;</pre>
Hidden Field	<pre>@Html.HiddenFor(m=&gt;m.UserId) Output: &lt;input id=" UserId" name=" UserId" type="hidden" value="UserId-val" /&gt;</pre>
CheckBox	<pre>@Html.CheckBoxFor(m=&gt;m.IsApproved) Output: &lt;input id="Checkbox1" name="Checkbox1" type="checkbox" value="true" /&gt; &lt;input name="myCheckbox" type="hidden" value="false" /&gt;</pre>
RadioButton	<pre>@Html.RadioButtonFor(m=&gt;m.IsApproved, "val") Output: &lt;input checked="checked" id="Radiobutton1" name="Radiobutton1" type="radio" value="val" /&gt;</pre>
Drop-down list	<pre>@Html.DropDownListFor(m =&gt; m.Gender, new SelectList(new [] {"Male", "Female"})) Output: &lt;select id="Gender" name="Gender"&gt; &lt;option&gt;Male&lt;/option&gt; &lt;option&gt;Female&lt;/option&gt; &lt;/select&gt;</pre>
Multiple-select	<pre>Html.ListBoxFor(m =&gt; m.Hobbies, new MultiSelectList(new [] {"Cricket", "Chess"})) Output: &lt;select id="Hobbies" multiple="multiple" name="Hobbies"&gt; &lt;option&gt;Cricket&lt;/option&gt; &lt;option&gt;Chess&lt;/option&gt; &lt;/select&gt;</pre>

- **Templated HTML Helpers** - These helpers figure out what HTML elements are required to render based on the properties of your model class. This is a very flexible approach for displaying data to the user, although it requires some initial care and attention to set up. To set up proper HTML element with Templated HTML Helper, make use of DataType attribute of DataAnnotation class.

**For example**, when you use DataType as Password, A templated helper automatically render a Password type HTML input element.

Templated Helper	Example
Display	Renders a read-only view of the specified model property and selects an appropriate HTML element based on property's data type and metadata. Html.Display("Name")
DisplayFor	Strongly typed version of the previous helper Html.DisplayFor(m => m. Name)
Editor	Renders an editor for the specified model property and selects an appropriate HTML element based on property's data type and metadata. Html.Editor("Name")
EditorFor	Strongly typed version of the previous helper Html.EditorFor(m => m. Name)

3. **Custom Html Helpers** - You can also create your own custom helper methods by creating an extension method on the HtmlHelper class or by creating static methods within a utility class.

```
public static class CustomHelpers
{
    //Submit Button Helper
    public static MvcHtmlString SubmitButton(this HtmlHelper helper, string
buttonText)
    {
        string str = "<input type=\"submit\" value=\"" + buttonText + "\"
/>";
        return new MvcHtmlString(str);
    }
    //Readonly Strongly-Typed TextBox Helper
    public static MvcHtmlString TextBoxFor<TModel, TValue>(this
HtmlHelper<TModel> htmlHelper, Expression<Func<TModel,
TValue>>expression, bool isReadonly)
    {
        MvcHtmlString html = default(MvcHtmlString);

        if (isReadonly)
        {
            html = System.Web.Mvc.Html.InputExtensions.TextBoxFor(htmlHelper,
expression, new { @class = "readOnly",
@readonly = "read-only" });
        }
        else
        {
            html = System.Web.Mvc.Html.InputExtensions.TextBoxFor(htmlHelper,
expression);
        }
        return html;
    }
}
```

### Q3. What are Url Helpers?

**Ans.** Url helpers allow you to render HTML links and raw URLs. The output of these helpers is dependent on the routing configuration of your ASP.NET MVC application.

HTML Element	Example
Relative URL	@Url.Content("~/Files/asp.netmvc.pdf") Output: /Files/asp.netmvc.pdf
Based on action/controller	@Html.ActionLink("About Us", "About", "Home") Output: <a href="/Home/About">About Us</a>
	@Html.ActionLink("About Me", "About", "Home", "http", "www.dotnet-tricks.com", null, null, null) Output: <a href="https://www.dotnet-tricks.com/Home/About ">About Me</a>
Raw URL for Action	Url.Action("About", "Home") Output: /Home/About

### Q4. What is Validation Summary?

**Ans.** The ValidationSummary helper displays an unordered list of all validation errors in the ModelState dictionary. It accepts a *boolean* value (i.e. *true* or *false*) and based on the boolean value it displays the errors. When a boolean parameter value is true, it shows only model-level errors and excludes model property-level errors (i.e any errors that are associated with a specific model property). When a Boolean value is false, it shows both model-level and property-level errors.

Suppose, you have the following lines of code somewhere in the controller action rendering a view:

```
ModelState.AddModelError("", "This is Model-level error!");  
ModelState.AddModelError("Name", "This Model property-level error!");
```

In the first error, there is no key to associate this error with a specific property. In the second error, there is a key named as "Title" to associate this error for model property Title.

```
@Html.ValidationSummary(true) @*//shows model-level errors*@  
@Html.ValidationSummary(false) @*//shows model-level and property-level errors*@
```

Hence, when the boolean type parameter value is true then *ValidationSummary* will display only model-level errors and exclude property-level errors. It will display Model-level and property-level errors when the boolean type parameter value is false.

### Q5. What are AJAX Helpers?

**Ans.** AJAX Helpers are used to creating AJAX-enabled elements like as AJAX-enabled forms and links which performs request asynchronously. AJAX Helpers are extension methods of *AJAXHelper* class which exist in *System.Web.Mvc* namespace.

AJAX HTML Element	Example
AJAX-enabled link based on action/controller	<pre>@Ajax.ActionLink("Load Products", "GetProducts", new AjaxOptions {UpdateTargetId = "Products-container", HttpMethod = "GET" })</pre> <p>Output: <code>&lt;a data-ajax="true" data-ajax-method="GET" data-ajax-mode="replace" data-ajax-update="#Products-container" href="/Home/GetProducts"&gt;Load Products&lt;/a&gt;</code></p>

## Q6. What is unobtrusive AJAX?

**Ans.** ASP.NET MVC supports unobtrusive Ajax which is based on jQuery. The unobtrusive Ajax means that you use helper methods to define your Ajax features, rather than adding blocks of code throughout your views.

## Q7. What are various configuration options for AJAX Helpers?

**Ans.** The *AjaxOptions* class defines properties that allow you to specify callbacks for different stages in the AJAX request life cycle. There are following properties provided by *AjaxOptions* class for AJAX helpers:

Property	Description
Url	Specify the URL that will be requested from the server.
Confirm	Specify a message that will be displayed in a confirm dialog to the end user. When the user clicks on the OK button in the confirmation dialog, the Ajax call performs.
OnBegin	Specify a JavaScript function name which is called at the beginning of the Ajax request.
OnComplete	Specify a JavaScript function name which is called at the end of the Ajax request.
OnSuccess	Specify a JavaScript function name which is called when the Ajax request is successful.
OnFailure	Specify a JavaScript function name which is called if the Ajax request fails.
LoadingElementId	Specify progress message container's Id to display a progress message or animation to the end user while an Ajax request is being made.
LoadingElementDuration	Specify a time duration in milliseconds that controls the duration of the progress message or animation.
UpdateTargetId	Specify the target container's Id that will be populated with the HTML returned by the action method.
InsertionMode	Specify the way of populating the target container. The possible values are <i>InsertAfter</i> , <i>InsertBefore</i> and <i>Replace</i> (which is the default).

## Q8. What is Cross-Domain AJAX?

**Ans.** By default, web browsers allow AJAX calls only to your web application's site of origin i.e. site hosted server. This restriction helps us to prevent various security issues like cross-site scripting (XSS) attacks. But, sometimes you need to interact with externally hosted API(s) like Twitter or Google. Hence to interact with these external API(s) or services your web application must support JSONP requests or Cross-Origin Resource Sharing (CORS). By default, ASP.NET MVC does not support JSONP or Cross-Origin Resource Sharing. For this, you need to do a little bit of coding and configuration.



## Q9. What are Layouts in ASP.NET MVC?

**Ans.** Layouts are used to maintain a consistent look and feel across multiple views within ASP.NET MVC application. As compared to Web Forms, layouts serve the same purpose as master pages but offer a simple syntax and greater flexibility. A basic structure of layout is given below:

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width" />
  <title>@ViewBag.Title</title>
  @Styles.Render("~/Content/css")
  @Scripts.Render("~/bundles/modernizr")
</head>
<body>
  @RenderBody()
  @Scripts.Render("~/bundles/jquery")
  @RenderSection("scripts", required: false)
</body>
</html>
```

You can use a layout to define a common template for your site. A layout can be declared at the top of view as:

```
@{
    Layout = "~/Views/Shared/SiteLayout.cshtml";
}
```

## Q10. What are Sections in ASP.NET MVC?

**Ans.** A section allows you to specify a region of content within a layout. It expects one parameter which is the name of the section. If you don't provide that, an exception will be thrown. A section in a layout page can be defined by using the following code.

```
@section header{
<h1>Header Content</h1>
}
```

You can render above defined section header on the content page as given below:

```
@RenderSection("header")
```

By default, sections are mandatory. To make sections optional just provides the second parameter value as false, which is a Boolean value.

```
@RenderSection("header", false)
```

**Note:** A view can define only those sections that are referred to in the layout page otherwise an exception will be thrown.

### Q11. What are RenderBody and RenderPage in ASP.NET MVC?

**Ans.** RenderBody method exists in the Layout page to render child page/view. It is just like the ContentPlaceHolder on master page. A layout page can have only one RenderBody method.

```
<body>
@RenderBody()
@RenderPage("~/Views/Shared/_Header.cshtml")
@RenderPage("~/Views/Shared/_Footer.cshtml")

@RenderSection("scripts",false)

@section scripts{
    <script src="~/Scripts/jquery-1.7.1.min.js"></script>
}
</body>
```

The renderpage method also exists in the Layout page to render another page exists in your application. A layout page can have multiple RenderPage methods.

```
@RenderPage("~/Views/Shared/_Header.cshtml")
```

### Q12. What are Styles.Render and Scripts.Render?

**Ans.** Style.Render is used to render a bundle of CSS files defined within BundleConfig.cs files. Styles.Render create style tag(s) for the CSS bundle. Like Style.Render, Scripts.Render is also used to render a bundle of Script files by rendering script tag(s) for the Script bundle.

```
public class BundleConfig
{
    public static void RegisterBundles(BundleCollection bundles)
    {
        bundles.Add(new ScriptBundle("~/bundles/jqueryval").Include(
            "~/Scripts/jquery.unobtrusive*",
            "~/Scripts/jquery.validate*"));

        bundles.Add(new StyleBundle("~/Content/themes/base/css").Include(
            "~/Content/themes/base/jquery.ui.core.css",
            "~/Content/themes/base/jquery.ui.resizable.css",
            "~/Content/themes/base/jquery.ui.selectable.css",
            "~/Content/themes/base/jquery.ui.button.css",
            "~/Content/themes/base/jquery.ui.dialog.css",
            "~/Content/themes/base/jquery.ui.theme.css"));
    }
}
```

Styles.Render and Scripts.Render generate multiple style and script tags for each item in the CSS bundle and Script bundle when optimizations are disabled. When optimizations are enabled, Styles.Render and Scripts.Render generate a single style and script tag to a version-stamped URL which represents the entire bundle for CSS and Scripts.

### Q13. How to enable and disable optimizations in ASP.NET MVC?

**Ans.** You can enable and disable optimizations by setting EnableOptimizations property of BundleTable class to true or false within Global.asax.cs file as shown below.

```
protected void Application_Start()
{
    //other code has been removed for clarity
    //disable optimization
    System.Web.Optimization.BundleTable.EnableOptimizations = false;
}
```

### Q14. What is ViewStart?

**Ans.** The \_ViewStart.cshtml page is used to serve the common layout page(s) for a group of views. The code within this file is executed before the code in any view placed in the same directory. This file is also recursively applied to any view within a subdirectory.

By default, ASP.NET MVC project has a \_ViewStart.cshtml file in the Views directory and it specifies a default layout for your ASP.NET MVC application as shown below:

```
@{
    Layout = "~/Views/Shared/Layout.cshtml";
}
```

Since this code runs before any view, hence a view can override the *Layout* property and choose a different layout.

### Q15. When to use \_ViewStart?

**Ans.** When a set of views shares common settings, the \_ViewStart.cshtml file is a great place to put these common view settings. If any view needs to override any of the common settings then that view can set new values in common settings.

### Q16. What are different ways of rendering layout in ASP.NET MVC?

**Ans.** There are following four different ways of rendering layout in ASP.NET MVC:

1. **Using \_ViewStart file in the root directory of the Views folder:** The \_ViewStart file within Views folder is used to server the default Layout page for your ASP.NET MVC application. You can also change the default rendering of layouts within \_ViewStart file based on the controller as shown below:

```
@{
    var controller =
HttpContext.Current.Request.RequestContext.RouteData.Values["Controller"].ToString();

    string layout = "";
    if (controller == "Admin")
    {
        layout = "~/Views/Shared/_AdminLayout.cshtml";
    }
    else
```

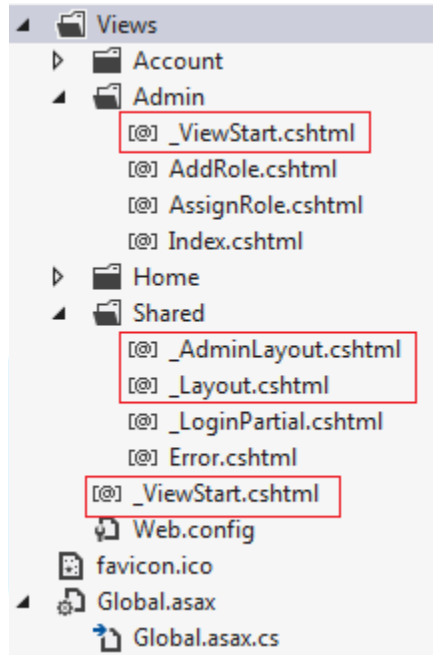
```

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

```

## 2. Adding a \_ViewStart file in each of the directories

You can also set the default layout for a particular directory by putting the \_ViewStart file in each of the directories with the required Layout information as shown below:



## 3. Defining Layout within each view on the top

```

@{
    Layout = "~/Views/Shared/_AdminLayout.cshtml";
}

```

## 4. Returning Layout from ActionResult

```

public ActionResult Index()
{
    RegisterModel model = new RegisterModel();
    //TO DO:
    return View("Index", "_AdminLayout", model);
}

```

### Q17. What is the App\_Start folder in ASP.NET MVC?

**Ans.** App\_Start folder has been introduced in MVC4. It contains various configurations files like as BundleConfig.cs, FilterConfig.cs, RouteConfig.cs, WebApiConfig.cs for your application. All these settings are registered within the Application\_Start method of Global.asax.cs file.

- **BundleConfig.cs** - This is used to create and register bundles for CSS and JS files. By default, various bundles are added in these files including jQuery, jQueryUI, jQuery Validation, Modernizr, and Site CSS.
- **FilterConfig.cs** - This is used to register global MVC filters like error filters, actions filters etc. By default, it contains a HandleErrorAttribute filter.
- **RouteConfig.cs** - This is used to register various route patterns for your ASP.NET MVC application. By default, one route is registered here named as Default Route.
- **WebApiConfig.cs** - This is used to register various WEB API routes like as ASP.NET MVC, as well as set any additional WEB API configuration settings.

### Q18. What are different ways of returning/rendering a view in ASP.NET MVC?

**Ans.** There are four different ways for returning/rendering a view in ASP.NET MVC as given below:

1. **Return View()** - This tells MVC to generate HTML to be displayed for the specified view and sends it to the browser. This acts like as Server.Transfer() in ASP.NET WebForm.
2. **Return RedirectToAction()** - This tells MVC to redirect to specified action instead of rendering HTML. In this case, the browser receives the redirect notification and make a new request for the specified action. This acts like as Response.Redirect() in ASP.NET WebForm.

Moreover, RedirectToAction constructs a redirect URL to a specific action/controller in your application and use the route table to generate the correct URL. RedirectToAction causes the browser to receive a 302 redirect within your application and gives you an easier way to work with your route table.

3. **Return Redirect()** - This tells MVC to redirect to specified URL instead of rendering HTML. In this case, the browser receives the redirect notification and make a new request for the specified URL. This also acts like as Response.Redirect() in ASP.NET WebForm. In this case, you have to specify the full URL to redirect.

Moreover, redirect also cause the browser to receive a 302 redirect within your application, but you have to construct the URLs yourself.

4. **Return RedirectToRoute()** - This tells MVC to look up the specifies route into the Route table that is defined in global.asax and then redirect to that controller/action defined in that route. This also make a new request like RedirectToAction().

#### Note:

1. Return View doesn't make a new request, it just renders the view without changing URLs in the browser's address bar.
2. Return RedirectToAction makes a new request and URL in the browser's address bar is updated with the generated URL by MVC.

3. Return Redirect also makes a new request and URL in the browser's address bar is updated, but you have to specify the full URL to redirect
4. Between RedirectToAction and Redirect, best practice is to use RedirectToAction for anything dealing with your application actions/controllers. If you use Redirect and provide the URL, you'll need to modify those URLs manually when you change the route table.
5. RedirectToRoute redirects to a specific route defined in the Route table.

### Q19. What are differences among ViewData, ViewBag, TempData and Session?

**Ans.** In ASP.NET MVC there are three ways - ViewData, ViewBag and TempData to pass data from the controller to view and in next request. Like WebForm, you can also use Session to persist data during a user session.

#### ViewData

- ViewData is a dictionary object that is derived from ViewDataDictionary class.

```
public ViewDataDictionary ViewData { get; set; }
```

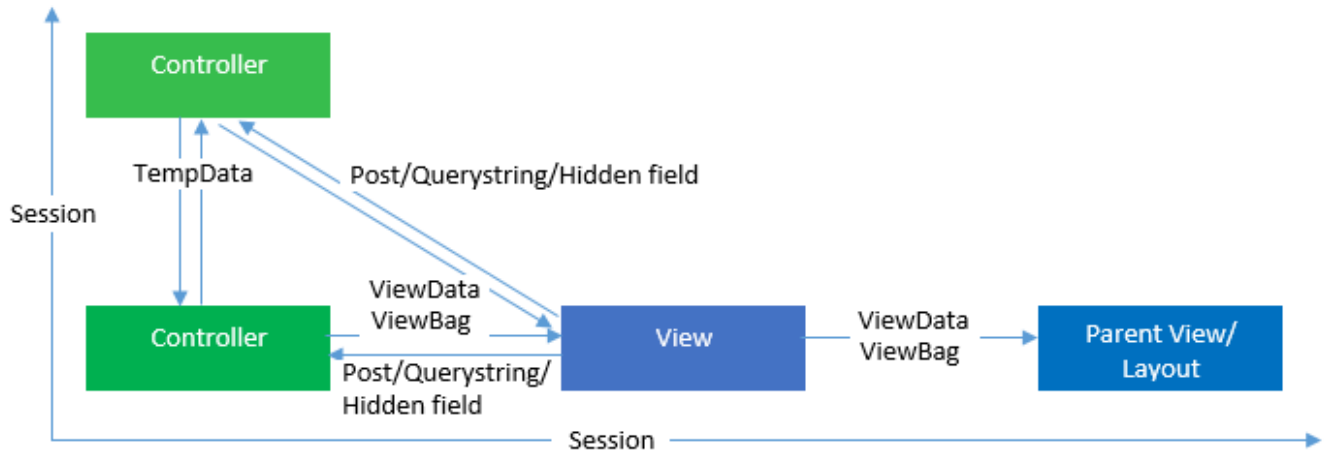
- ViewData is used to pass data from controller to the corresponding view.
- Its life lies only during the current request.
- If redirection occurs then its value becomes null.
- It's required typecasting for getting data and check for null values to avoid error.

#### ViewBag

- ViewBag is a dynamic property that takes advantage of the new dynamic features in C# 4.0.

```
public Object ViewBag { get; }
```

- Basically, it is a wrapper around the ViewData and also used to pass data from controller to the corresponding view.
- Its life also lies only during the current request.
- If redirection occurs then its value becomes null.
- It doesn't require typecasting for getting data.



### TempData

- TempData is a dictionary object that is derived from TempDataDictionary class and stored in short lives session.

```
public TempDataDictionary TempData { get; set; }
```

- TempData is used to pass data from current request to subsequent request (means redirecting from one page to another).
- Its life is very short and lies only till the target view is fully loaded.
- It's required typecasting for getting data and check for null values to avoid error.
- It's used to store only one-time messages like error messages, validation messages.

### Session

- In ASP.NET MVC, Session is a property of Controller class whose type is HttpSessionStateBase.

```
public HttpSessionStateBase Session { get; }
```

- The session is also used to pass data within the ASP.NET MVC application and Unlike TempData, it persists data for a user session until it is timeout (by default session timeout is 20 minutes).
- The session is valid for all requests, not for a single redirect.
- It's also required typecasting for getting data and check for null values to avoid error.

## Q20. How to persist data in TempData?

**Ans.** The life of TempData is very short and lies only till the target view is fully loaded. But you can persist data in TempData by calling Keep() method after request completion

- void Keep()** - Calling this method within the current action ensures that all the items in TempData are not removed at the end of the current request.

```
public ActionResult Index()
{
    ViewBag.Message = TempData["Message"];
    Employee emp = TempData["emp"] as Employee; //need type casting
    TempData.Keep(); //persist all strings values
    return View();
}
```

- **void Keep(string key)** - Calling this method within the current action ensures that specific item in TempData is not removed at the end of the current request.

```
public ActionResult Index()
{
    ViewBag.Message = TempData["Message"];
    Employee emp = TempData["emp"] as Employee; //need type casting
    //persist only data for emp key and Message key will be destroy
    TempData.Keep("emp");
    return View();
}
```

## Q21. How to control Session behaviour in ASP.NET MVC?

**Ans.** By default, ASP.NET MVC support session state. The session is used to store data values across requests. Whether you store some data values within the session or not ASP.NET MVC must manage the session state for all the controllers in your application that is time-consuming. Since, the session is stored on the server side and consumes server memory, hence it also affects your application performance.

If some of the controllers of your ASP.NET MVC application are not using session state features, you can disable session for that controller and can gain slight performance improvement of your application. You can simplify session state for your application by using available options for session state.

In ASP.NET MVC4, *SessionState* attribute provides you more control over the behaviour of session-state by specifying the value of *SessionStateBehavior* enumeration as shown below:

Value	Description
Default	The default ASP.NET behaviour is used to determine the session state behaviour.
Disabled	Session state is disabled entirely.
ReadOnly	Read-only session state behaviour is enabled.
Required	Full read-write session state behaviour is enabled.



```

1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Web;
5 using System.Web.Mvc;
6 using System.Web.SessionState; — Required for controlling session state
7 namespace Mvc4_SessionState.Controllers
8 {
9     [SessionState(SessionStateBehavior.Disabled)]
10    public class HomeController : Controller
11    {
12        public ActionResult Index()
13        {
14            TempData["Message"] = "DotNet Tricks";
15            return View();
16        }
17    }
18 }

```

controlling Session in Asp.Net MVC4";

## Q22. How TempData is related to Session in ASP.NET MVC?

**Ans.** In ASP.NET MVC, TempData uses session state for storing the data values across requests. Hence, when you will be disabled the session state for the controller, it will throw the exception as shown below:

### Server Error in '/' Application.

*The SessionStateTempDataProvider class requires session state to be enabled.*

**Description:** An unhandled exception occurred during the execution of the current web request. Please review the stack trace for more information about the error and where it originated in the code.

**Exception Details:** System.InvalidOperationException: The SessionStateTempDataProvider class requires session state to be enabled.

## Q23. What are Action methods in ASP.NET MVC?

**Ans.** Controller actions are methods defined in the controller class and responsible to perform required operations on the user's inputs like as form values, query strings values etc. with the help of Model and passing the results back to the View. Asp.net MVC has the following built-in ActionResult Type and Helper methods:

1. **ViewResult** - Returns a ViewResult which renders the specified or default view by using controller View() helper method.
2. **PartialViewResult** - Returns a PartialViewResult which renders the specified or default partial view (means a view without its layout) by using controller PartialView() helper method.
3. **RedirectResult** - Returns a RedirectResult which Issues an HTTP 301 or 302 redirection to a specific URL by using controller Redirect() helper method.
4. **RedirectToRouteResult** - Returns a RedirectToRouteResult which Issues an HTTP 301 or 302 redirection to an action method or specific route entry by using controller RedirectToAction(), RedirectToActionPermanent(), RedirectToRoute(), RedirectToRoutePermanent() helper methods.
5. **ContentResult** - Returns a ContentResult which renders raw text like as "Hello, DotNet Tricks!" by using controller Content() helper method.

6. **JsonResult** - Returns a JsonResult which serializes an object in JSON format (like as "{ \"Message\": Hello, World! }") and renders it by using controller Json() helper method.
7. **JavaScriptResult** - Returns a JavaScriptResult which renders a snippet of JavaScript code like as "function hello() { alert(Hello, World!); }" by using controller JavaScript() helper method. This is used only in AJAX scenarios.
8. **FileResult** - Returns a FileResult which renders the contents of a file like as PDF, DOC, Excel etc. by using controller File() helper method.
9. **EmptyResult** - Returns no result returned by an action. This has no controller helper method.
10. **HttpNotFoundResult** - Returns an HttpNotFoundResult which renders a 404 HTTP Status Code response by using controller HttpNotFound() helper method.
11. **HttpUnauthorizedResult** - Returns an HttpUnauthorizedResult which renders a 401 HTTP Status Code (means "not authorized") response. This has no controller helper method. This is used for authentication (forms authentication or Windows authentication) to ask the user to log in.
12. **HttpStatusCodeResult** - Returns an HttpStatusCodeResult which renders a specified HTTP code response. This has no controller helper method.

#### Q24. What is ActionResult and how is it different from others?

**Ans.** The ActionResult class is the base class for all action results. An action result can be of type ViewResult, JsonResult, RedirectResult and so on. Hence, when your action method returns multiple results based on different conditions, ActionResult is the best choice. Since it can return any type of result.

```
public ActionResult Index(int id)
{
    if (id == 1)
        return View(); // returns simple ViewResult
    else if (id == 2)
        return Json(new { result = "1" }, JsonRequestBehavior.AllowGet); //
    returns JsonResult
    else
        return RedirectToAction("Login"); // returns to Login Page
}
```

#### Q25. How to make a Non-Action method in ASP.NET MVC?

**Ans.** By default, the ASP.NET MVC framework treats all public methods of a controller class as action methods. If you do not want a public method to be an action method, you must mark that method with the *NonActionAttribute* attribute.

```
[NonAction]
public void DoSomething()
{
    // Method logic
}
```

#### Q26. Can you change the action method name?

**Ans.** You can also change the action method name by using *ActionName* attribute. Now the action method will be called by the name defined by the *ActionName* attribute.

```
[ActionName("DoAction")]
public ActionResult DoSomething()
{
    //TODO:
    return View();
}
```

Now, *DoSomething* action will be identified and called by the name *DoAction*.

## Q27. How to restrict an action method to be invoked only by HTTP GET, POST, PUT or DELETE?

**Ans.** By default, each and every action method can be invoked by an HTTP request (i.e. GET, PUT, POST, and DELETE). But you can restrict an action to be invoked only by a specific HTTP request by applying `HttpGet` or `HttpPost` or `HttpPut` or `HttpDelete` attribute.

If you want to restrict an action method for HTTP Get request only then decorate it with `HttpGet` action method selector attribute as given below:

```
[HttpGet]
public ActionResult Index()
{
    //TODO:
    return View();
}
```

## Q28. How to determine an action method is invoked by HTTP GET or POST?

**Ans.** By using `HttpMethod` property of `HttpRequestBase` class, you can find out whether an action is invoked by HTTP GET or POST.

```
public ActionResult Index(int? id)
{
    if (Request.HttpMethod == "GET")
    {
        //TODO:
    }
    else if (Request.HttpMethod == "POST")
    {
        //TODO:
    }
    else
    {
        //TODO:
    }
    return View();
}
```

### Q29. How to determine an AJAX request?

**Ans.** You can determine an AJAX request by using *Request.IsAjaxRequest()* method. It will return true, if the request is an AJAX request else returns false.

```
public ActionResult DoSomething()
{
    if (Request.IsAjaxRequest())
    {
        //TODO:
    }
    return View();
}
```



# Forms and Validations

## Q1. What are Data Annotations in ASP.NET MVC?

**Ans.** Data validation is a key aspect for developing a web application. In Asp.net MVC, we can easily apply validation to the web application by using Data Annotation attribute classes to the model class. Data Annotation attribute classes are present in System.ComponentModel.DataAnnotations namespace and are available to Asp.net projects like Asp.net web application & website, Asp.net MVC, Web forms and also to Entity framework ORM models.

Data Annotations help us to define the rules to the model classes or properties for data validation and displaying suitable messages to end users.

### Data Annotation Validator Attributes

- **DataType** - Specify the datatype of a property
- **DisplayName** - specify the display name for a property.
- **DisplayFormat** - specify the display format for a property like the different format for Date property.
- **Required** - Specify a property as required.
- **RegularExpression** - validate the value of a property by specified regular expression pattern.
- **Range** - validate the value of a property within a specified range of values.
- **StringLength** - specify min and max length for a string property.
- **MaxLength** - specify max length for a string property.
- **Bind** - specify fields to include or exclude when adding parameter or form values to model properties.
- **ScaffoldColumn** - specify fields for hiding from editor forms.

## Q2. How to apply Server-side validation in ASP.NET MVC?

**Ans.** Server-side validations are very important before playing with sensitive information of a user. Server-side validation must be done whether we validate the received data on the client side. The user could disable script in his browser or do something else to bypass client-side validation. In this case, server-side validation must require to protect our data from dirty input.

In ASP.NET MVC, there are two ways to validate a model on the server side:

1. **Explicit Model Validation** – This is the traditional way to validate the model data by using If...else...If statement. In this way, you need to check your model property values one by one for your desired result. If model property values are unexpected, inject error messages within ModelState.

```

class HomeController : Controller
{
    [HttpPost]
    public ActionResult ExplicitServer(UserViewModel model)
    {
        //Write custom logic to validate UserViewModel
        if (string.IsNullOrEmpty(model.UserName))
        {
            ModelState.AddModelError("UserName", "Please enter your
name");
        }
        if (!string.IsNullOrEmpty(model.UserName))
        {
            Regex emailRegex = new Regex(".*@.*\\.?.*");
            if (!emailRegex.IsMatch(model.UserName))
                ModelState.AddModelError("UserName", "Please enter correct
email address");
        }

        if (ModelState.IsValid) //Check model state
        {
            //TO DO:
        }
    }
}

```

2. **Model Validation with Data Annotations** - Data Annotations was introduced with .NET 3.5 SP1. It has a set of attributes and classes defined in the System.ComponentModel.DataAnnotations assembly. Data Annotations allow us to decorate model classes with metadata. This metadata describes a set of rules that are used to validate a property.

```

public class UserViewModel
{
    [Required(ErrorMessage = "Please Enter Email Address")]
    [RegularExpression(".*@.*\\.?.*", ErrorMessage = "Please Enter Correct
Email Address")]
    public string UserName { get; set; }

    [Required(ErrorMessage = "Please Enter Password")]
    [StringLength(50, ErrorMessage = "The {0} must be at least {2}
characters long.", MinimumLength = 6)]
    public string Password { get; set; }
}

```

### Q3. How to determine there is no error in Model State?

**Ans.** When server-side model validation fails, errors are included in the ModelState. Hence, by using *ModelState.IsValid* property you can verify model state. It returns true if there is no error in ModelState else returns false.

```
[HttpPost]
public ActionResult DoSomething(UserViewModel model)
{
    if (ModelState.IsValid)
    {
        //TODO:
    }
    return View();
}
```

#### Q4. How to enable and disable client-side validation in ASP.NET MVC?

**Ans.** We can enable and disable the client-side validation by setting the values of ClientValidationEnabled & UnobtrusiveJavaScriptEnabled keys true or false. This setting will be applied to application level.

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

For client-side validation, the values of above both the keys must be true. When we create a new project using Visual Studio in MVC3 or MVC4, by default the values of both the keys are set to true.

We can also enable the client-side validation programmatically. For this we need to do code within the Application\_Start() event of the Global.asax, as shown below.

```
protected void Application_Start()
{
    //Enable or Disable Client Side Validation at Application Level
    HtmlHelper.ClientValidationEnabled = true;
    HtmlHelper.UnobtrusiveJavaScriptEnabled = true;
}
```

We can also enable or disable client-side validation for a specific view. For this, we required to enable or disable client-side validation inside a Razor code block as shown below. This option will override the application level settings for that specific view.

```
@using MvcApp.Models
@{
    ViewBag.Title = "About";
    HtmlHelper.ClientValidationEnabled = false;
}
```

#### Q5. What is a CDN and advantages of CDN?

**Ans.** CDN stands for content delivery network or content distribution network (CDN) which is a large distributed system of servers deployed in multiple data centres across the Internet. The goal of a CDN is to serve the content (like the jQuery library and other open source libraries) to end-users with high availability and high performance.

There are three popular CDN – Google, Microsoft and jQuery.

```
// Google CDN
<scripttype="text/javascript"src="http://ajax.googleapis.com/ajax/libs/jquery/1.9.1/jquery.min.js"></script>

// Microsoft CDN
<scripttype="text/javascript"src="http://ajax.microsoft.com/ajax/jquery/jquery-1.9.1.min.js"></script>

// JQuery CDN
<scripttype="text/javascript"src="http://code.jquery.com/jquery-1.9.1.min.js"></script>
```

### Advantages

1. It reduces the load from your application server.
2. It saves bandwidth since jQuery and another open libraries/framework will load faster from these CDN.
3. The most important benefit is it will be cached means if a user has visited any site which is using jQuery framework from any of these CDN and your web application is also using the same CDN for serving the jQuery then for your application, it will not request the jQuery from CDN.

### Q6. What is jquery.validate.unobtrusive.js?

Or

#### What is jQuery Validation Unobtrusive plugin?

**Ans.** Microsoft introduced jquery.validate.unobtrusive.js plugin with ASP.NET MVC3 to apply data model validations to the client side using a combination of jQuery Validation and HTML 5 data attributes.

### Q7. What is Bundling and Minification in ASP.NET MVC?

**Ans.** ASP.NET MVC4 and .NET Framework 4.5 offer bundling and minification techniques that reduce the number of request to the server and size of requested CSS and JavaScript, which improve page loading time.

A **bundle** is a logical group of files that are loaded with a single HTTP request. You can create style and script bundle for CSS and Java Scripts respectively by calling *BundleCollection* class Add() method. All bundles are created within BundleConfig.cs file.

```
public class BundleConfig
{
    public static void RegisterBundles(BundleCollection bundles)
    {
        bundles.Add(new
        StyleBundle("~/Content/css").Include("~/Content/site.min.css",
        "~/Content/mystyle.min.css"));

        bundles.Add(new ScriptBundle("~/bundles/jqueryval").Include(
        "~/Scripts/jquery-1.7.1.min.js",
        "~/Scripts/jquery.validate.min.js",
        "~/Scripts/jquery.validate.unobtrusive.min.js"));
    }
}
```



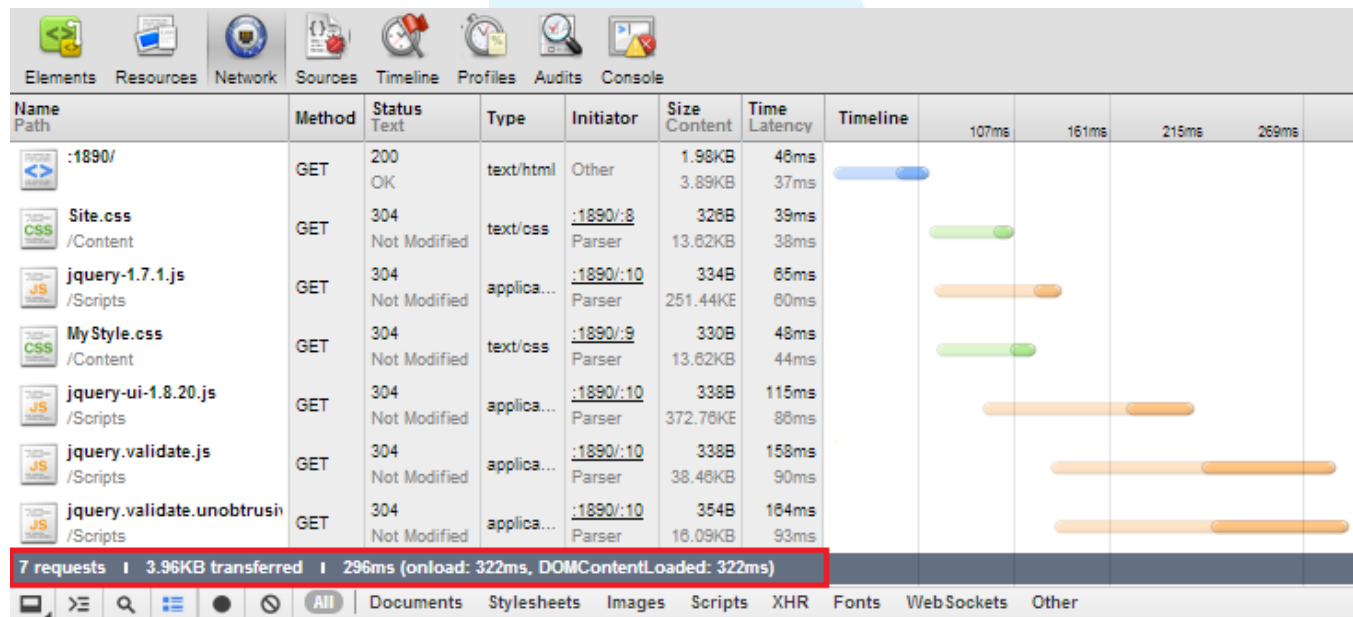
```
}
```

**Minification** is a technique for removing unnecessary characters (like white space, newline, tab) and comments from the JavaScript and CSS files to reduce the size which cause improved load times of a webpage. There are so many tools for minifying the js and CSS files. *JSMIn* and *YUI Compressor* are the two most popular tools for minifying js and CSS files.

### CSS and JS files Without Bundling and Minification

Suppose you have below CSS and JS files on the layout page and run the application in chrome browser and test no of request and loading time using chrome developer tools as shown below.

```
<link href="/Content/Site.css" rel="stylesheet"/>
<link href="/Content/MyStyle.css" rel="stylesheet"/>
<script src="/Scripts/jquery-1.7.1.js"></script>
<script src="/Scripts/jquery-ui-1.8.20.js"></script>
<script src="/Scripts/jquery.validate.js"></script>
<script src="/Scripts/jquery.validate.unobtrusive.js"></script>
```

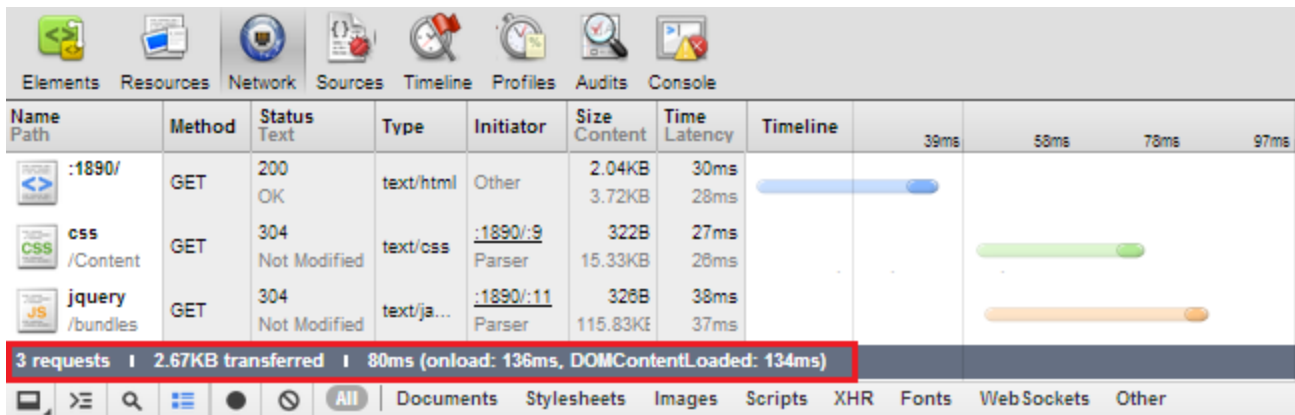


In this test, there are 7 requests, total data size is 3.96KB and loading time is approximate 296ms.

### CSS and JS files with Bundling and Minification

When you will run the above application with Bundling and Minification of CSS and js files and test no of request and loading time using chrome developer tools as shown below.

```
@Styles.Render("~/Content/CSS")
@Scripts.Render("~/bundles/jquery")
```



Name Path	Method	Status Text	Type	Initiator	Size Content	Time Latency	Timeline
:1890/	GET	200 OK	text/html	Other	2.04KB 3.72KB	30ms 28ms	
css /Content	GET	304 Not Modified	text/css	:1890/:9 Parser	322B 15.33KB	27ms 26ms	
jquery /bundles	GET	304 Not Modified	text/ja...	:1890/:11 Parser	326B 115.83KB	38ms 37ms	

3 requests | 2.67KB transferred | 80ms (onload: 136ms, DOMContentLoaded: 134ms)

In this test, there are only 3 requests, total data size is 2.67KB and loading time is approximate 80ms. In this way, by using bundling and minification you have reduced the total no of the request, size and loading time.

## Q8. Can we use Bundling and Minification in ASP.NET MVC3 or ASP.NET4.0?

**Ans.** **System.Web.Optimization** class offers the bundling and minification techniques that are exist within the **Microsoft.Web.Optimization.dll**. Using this dll you can also use this technique with ASP.NET MVC3 and .NET Framework 4.0.

## Q9. How Bundling use browser Cache capability?

**Ans.** Browsers cache resources based on URLs. When a web page requests a resource, the browser first checks its cache to see if there is a resource with the matched URL. If yes, then it simply uses the cached copy instead of fetching a new one from the server. Hence whenever you change the content of CSS and JS files will not reflect on the browser. For this, you need to force the browser for refreshing/reloading.

```
<link href="/Content/css?v=iJLo_MK-rZGssJwIhsd0GRIRUYoG7m0bm72SoF5rthY1" rel="stylesheet"/>
<script src="/bundles/jquery?v=BxdNbDh1jyvLVc0nUvn8CE1ECM7Yjo33KPcEmsDrOWQ1"></script>
```

But bundles automatically takes care of this problem by adding a hash code to each bundle as a query parameter to the URL as shown below. Whenever you change the content of CSS and JS files then a new hash code will be generated and rendered to the page automatically. In this way, the browser will see a different Url and will fetch the new copy of CSS and JS.

## Q10. What is Partial View in ASP.NET MVC?

**Ans.** A partial view is like as user control in ASP.NET Web forms that are used for code re-usability. Partial views help us to reduce code duplication. Hence partial views are reusable views like as Header and Footer views.

We can use a partial view to display blog comments, product category, social bookmarks buttons, a dynamic ticker, calendar etc.

It is best practice to create a partial view in the shared folder and partial view name is preceded by "\_", but it is not mandatory. The "\_" before view name specify that it is a reusable component i.e. partial view.

### Q11. How do you return a partial view from the controller?

Ans. return PartialView(options); where options could be a Model or a View name

### Q12. What are different ways of rendering a Partial View in ASP.NET MVC?

Ans. There are four methods for rendering a partial view in ASP.NET MVC These are RenderPartial, RenderAction, Partial and Action helper methods.

#### Html.RenderPartial

- This method result will be directly written to the HTTP response stream means it used the same TextWriter object as used in the current webpage/template.
- This method returns void.
- Simple to use and no need to create any action.
- RenderPartial method is useful when the displaying data in the partial view is already in the corresponding view model. For example: In a blog to show comments of an article, you can use RenderPartial method since an article information with comments is already populated in the view model.

```
@{Html.RenderPartial("_Comments");}
```

- This method is faster than Partial method since its result is directly written to the response stream which makes it fast.

#### Html.RenderAction

- This method result will be directly written to the HTTP response stream means it used the same TextWriter object as used in the current webpage/template.
- For this method, we need to create a child action for the rendering the partial view.
- RenderAction method is useful when the displaying data in the partial view is independent of the corresponding view model. For example: In a blog to show category list on each and every page, we would like to use RenderAction method since the list of the category is populated by the different model.

```
@{Html.RenderAction("Category", "Home");}
```

- This method is the best choice when you want to cache a partial view.
- This method is faster than Action method since its result is directly written to the HTTP response stream which makes it fast.

#### Html.Partial

- Renders the partial view as an HTML-encoded string.
- This method result can be stored in a variable since it returns string type value.
- Simple to use and no need to create any action.
- Like a RenderPartial method, a Partial method is also useful when the displaying data in the partial view is already in the corresponding view model. For example: In a blog to show comments of an article, you

can use Partial method since an article information with comments is already populated in the view model.

```
@Html.Partial("_Comments")
```

### Html.Action

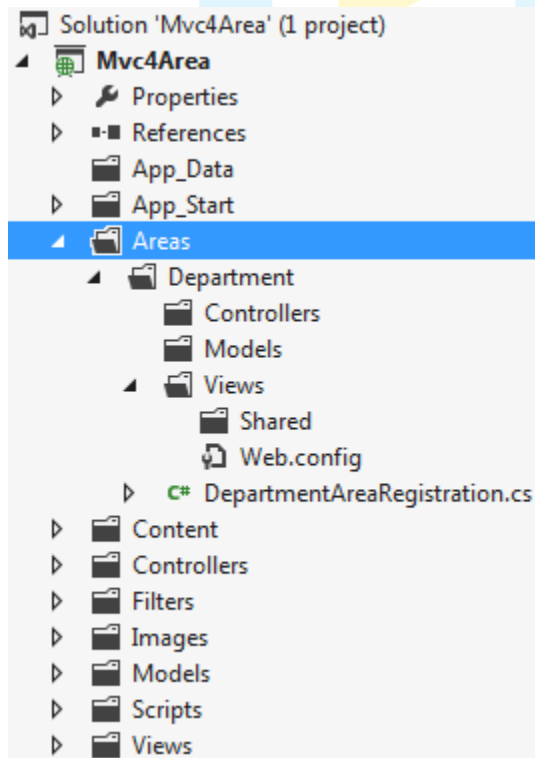
- Renders the partial view as an HtmlString.
- For this method, we need to create a child action for the rendering the partial view.
- This method result can be stored in a variable since it returns string type value.
- The action method is useful when the displaying data in the partial view is independent of the corresponding view model. For example: In a blog to show category list on each and every page, we would like to use Action method since the list of categories is populated by the different model.

```
@{Html.Action("Category", "Home");}
```

- This method is also the best choice when you want to cache a partial view.

### Q13. What is Area in ASP.NET MVC?

**Ans.** Areas were introduced in Asp.net MVC2 which allow us to organize models, views, and controllers into separate functional sections of the application, such as administration, billing, customer support, and so on. This is very helpful in a large web application, where all the controllers, views, and models have a single set of folders and that becomes difficult to manage.



Each MVC area has its own folder structure which allows us to keep separate controllers, views, and models. This also helps the multiple developers to work on the same web application without interfere with one another.

#### Q14. How to register Area in ASP.NET MVC?

**Ans.** Before working with area, make sure you have registered your area within the `Application_Start` method in `Global.asax` as shown below.

```
protected void Application_Start()
{
    //Register all application Areas
    AreaRegistration.RegisterAllAreas();
}
```

Always remember the order of registering the Areas must be on top so that all of the settings, filters and routes registered for the applications will also apply on the Areas.

#### Q15. What is Child action and how to invoke it?

**Ans.** Child actions are useful for creating reusable widgets which could be embedded into your views. In ASP.NET MVC partial views are used to create reusable widgets and a partial can be rendered by an action method. This action method can have child attribute and has its independent MVC lifecycle from parent view. Also, an action which has a child attribute cannot be called independently. It always will be called within a parent view otherwise it would give an error.

```
[ChildActionOnly]
public ActionResult MenuBar()
{
    //TODO:
    return PartialView();
}
```

A child action is invoked by using `@Html.RenderAction` or `@Html.Action` helper methods from inside of a view.

#### Q16. What is Scaffolding?

**Ans.** Scaffolding is a technique used by many MVC frameworks like ASP.NET MVC, Ruby on Rails, Cake PHP and Node.JS etc., to generate code for basic CRUD (create, read, update, and delete) operations against your database effectively. Further, you can edit or customize this auto-generated code according to your need.

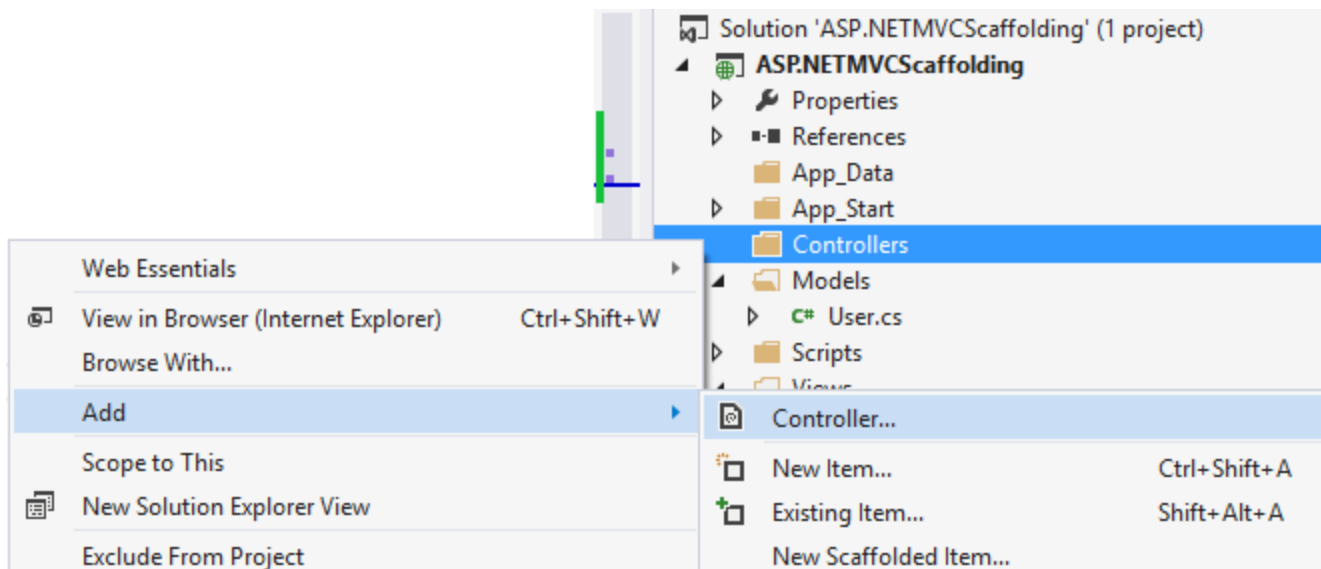
Scaffolding consists of page templates, entity page templates, field page templates, and filter templates. These templates are called Scaffold templates and allow you to quickly build a functional data-driven Web site.

#### Q17. How do Scaffold templates work in ASP.NET MVC?

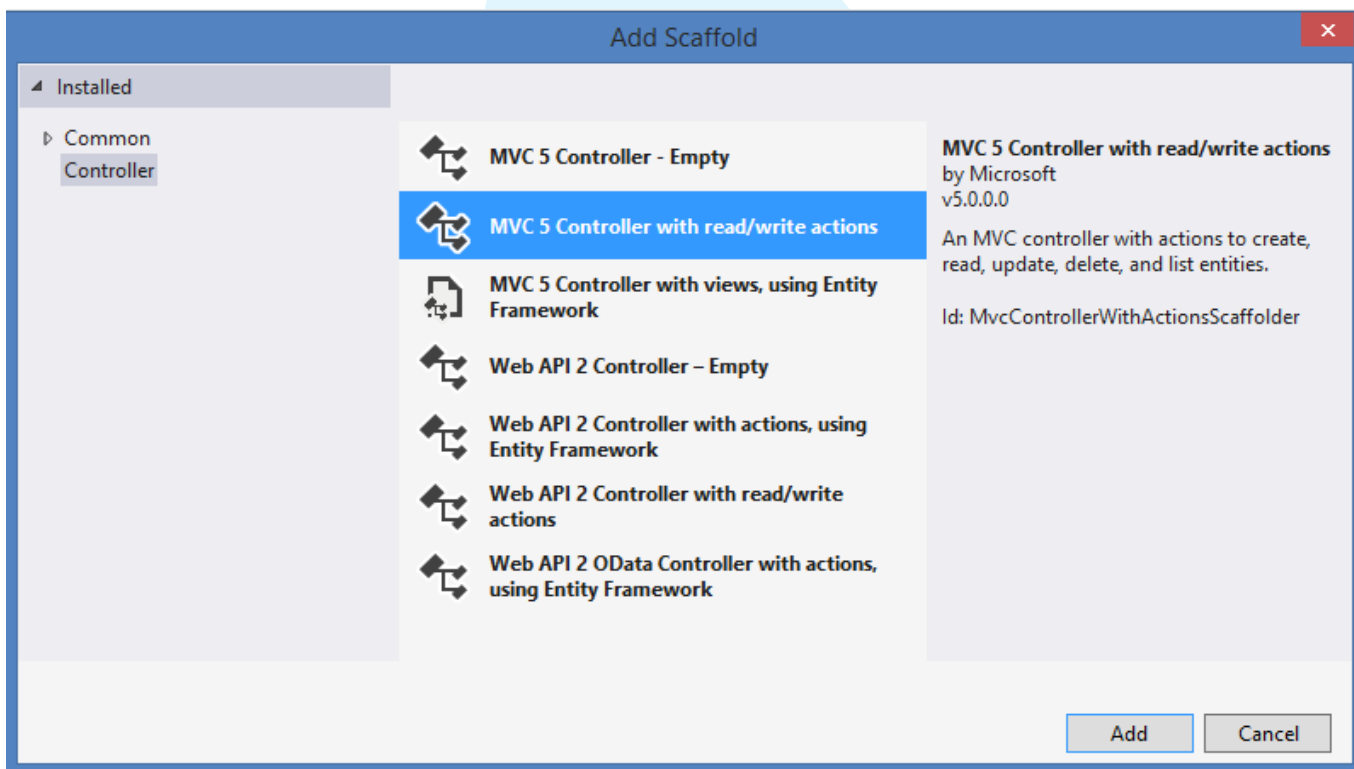
**Ans.** Scaffold templates are used to generate code for basic CRUD operations within your ASP.NET MVC applications against your database with the help Entity Framework. These templates use the Visual Studio T4 templating system to generate views for basic CRUD operations with the help of Entity Framework.

Steps to create ASP.NET MVC CRUD operations using scaffolding in ASP.NET MVC:

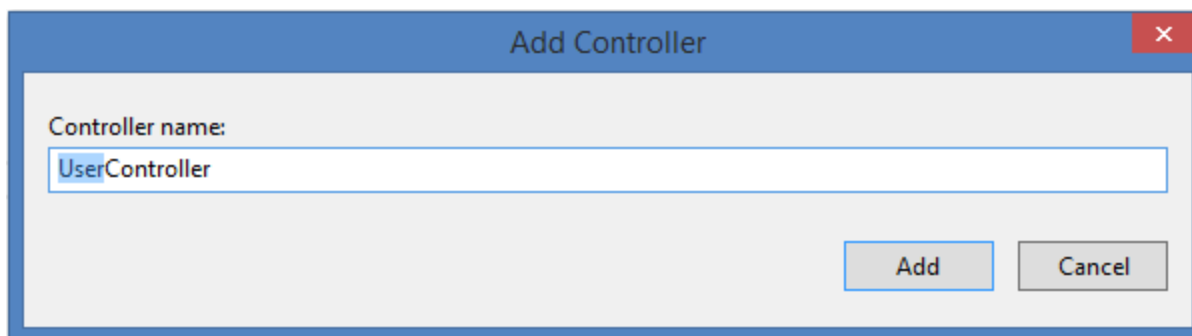
**Step1:** Adding a controller to your project



**Step2:** Choosing a scaffold template for creating CRUD operations



**Step3:** Provide a name to your controller



The following actions are created for insert, update and delete operations based on scaffold template within the User controller.

```
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Web;
5 using System.Web.Mvc;
6 namespace ASP.NETMVC Scaffold.Controllers
7 {
8     public class UserController : Controller
9     {
10         // GET: /User/
11         public ActionResult Index()
12         {
13             return View();
14         }
15         // GET: /User/Details/5
16         public ActionResult Details(int id)
17         {
18             return View();
19         }
20         // GET: /User/Create
21         public ActionResult Create()
22         {
23             return View();
24         }
25         // POST: /User/Create
26         [HttpPost]
27         public ActionResult Create(FormCollection collection)
28         {
```

# 5

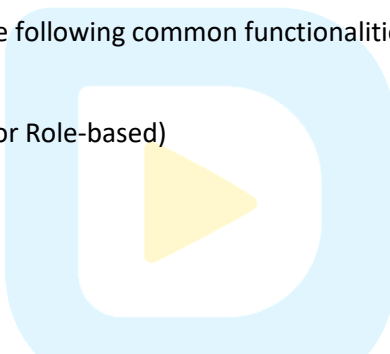
## Security and Filters

### Q1. What are ASP.NET MVC Filters and Attributes?

**Ans.** ASP.NET MVC provides a simple way to inject your piece of code or logic either before or after an action is executed. This is achieved by decorating the controllers or actions with ASP.NET MVC attributes or custom attributes. An attribute or custom attribute implements the ASP.NET MVC filters (filter interface) and can contain your piece of code or logic. You can make your own custom filters or attributes either by implementing ASP.NET MVC filter interface or by inheriting and overriding methods of ASP.NET MVC filter attribute class if available.

Typically, Filters are used to perform the following common functionalities in your ASP.NET MVC application.

1. Custom Authentication
2. Custom Authorization (User-based or Role-based)
3. Error handling or logging
4. User Activity Logging
5. Data Caching
6. Data Compression



### Q2. What are different types of Filters in ASP.NET MVC?

**Ans.** The ASP.NET MVC framework provides five types of filters.

1. **Authentication Filters** - This filter is introduced with ASP.NET MVC5. The `IAuthorizationFilter` interface is used to create CustomAuthentication filter. The definition of this interface is given below-

```
public interface IAuthenticationFilter
{
    void OnAuthentication(AuthenticationContext filterContext);

    void OnAuthenticationChallenge(AuthenticationChallengeContext
filterContext);
}
```

You can create your CustomAuthentication filter attribute by implementing `IAuthorizationFilter` as shown below-

```
public class CustomAuthenticationFilterAttribute : FilterAttribute,
IAuthorizationFilter
{
```



```

    public void OnAuthentication(AuthenticationContext filterContext)
    {
        filterContext.HttpContext.Response.Write("Authentication
Filter<br/>");
    }
    //Runs after the OnAuthentication method
    public void OnAuthenticationChallenge(AuthenticationChallengeContext
filterContext)
    {
        //TODO: Additional tasks on the request
    }
}

```

2. **Authorization Filters** - The ASP.NET MVC Authorize filter attribute implements the `IAuthorizationFilter` interface. The definition of this interface is given below-

```

public interface IAuthorizationFilter
{
    void OnAuthorization(AuthorizationContext filterContext);
}

```

The `AuthorizeAttribute` class provides the following methods to override in the CustomAuthorize attribute class.

```

public class AuthorizeAttribute : FilterAttribute, IAuthorizationFilter
{
    protected virtual bool AuthorizeCore(HttpContextBase httpContext);
    protected virtual void HandleUnauthorizedRequest(AuthorizationContext
filterContext);
    public virtual void OnAuthorization(AuthorizationContext filterContext);
    protected virtual HttpValidationStatus
OnCacheAuthorization(HttpContextBase httpContext);
}

```

In this way, you can make your CustomAuthorize filter attribute either by implementing an `IAuthorizationFilter` interface or by inheriting and overriding above methods of `AuthorizeAttribute` class.

3. **Action Filters** - Action filters are executed before or after an action is executed. The `IActionFilter` interface is used to create an Action Filter which provides two methods `OnActionExecuting` and `OnActionExecuted` which will be executed before or after an action is executed respectively.

```

public interface IActionFilter
{
    void OnActionExecuting(ActionExecutingContext filterContext);
    void OnActionExecuted(ActionExecutedContext filterContext);
}

```

4. **Result Filters** - Result filters are executed before or after generating the result for an action. The Action Result type can be `ViewResult`, `PartialViewResult`, `RedirectToRouteResult`, `RedirectResult`, `ContentResult`, `JsonResult`,

FileResult and EmptyResult which derives from the ActionResult class. Result filters are called after the Action filters. The IResultFilter interface is used to create a Result Filter which provides two methods OnResultExecuting and OnResultExecuted which will be executed before or after generating the result for an action respectively.

```
public interface IResultFilter
{
    void OnResultExecuted(ResultExecutedContext filterContext);
    void OnResultExecuting(ResultExecutingContext filterContext);
}
```

5. **Exception Filters** - Exception filters are executed when an exception occurs during the actions execution or filters execution. The IExceptionHandler interface is used to create an Exception Filter which provides OnException method which will be executed when an exception occurs during the actions execution or filters execution.

```
public interface IExceptionHandler
{
    void OnException(ExceptionContext filterContext);
}
```

The HandleErrorAttribute class is one example of an exception filter which implements IExceptionHandler. When HandleError filter receives the exception it returns an Error view located in the Views/Shared folder of your ASP.NET MVC application.

### Q3. When Exception filters are executed in ASP.NET MVC?

Ans. Exception filters are executed if there is an unhandled exception thrown during the execution of the ASP.NET MVC pipeline.

### Q4. What is the order of execution of filters in ASP.NET MVC?

Ans. All ASP.NET MVC filter are executed in an order. The correct order of execution is given below:

- Authentication filters
- Authorization filters
- Action filters
- Result filters

### Q5. How to configure filters in ASP.NET MVC?

Ans. You can configure your own custom filter into your application at following three levels:

1. **Global level** - By registering your filter into Application\_Start event of Global.asax.cs file with the help of FilterConfig class.

```
protected void Application_Start()
{
    FilterConfig.RegisterGlobalFilters(GlobalFilters.Filters);
}
```

```
}
```

2. **Controller level** - By putting your filter on the top of the controller name as shown below-

```
[Authorize(Roles = "Admin")]
public class AdminController: Controller
{
    //TODO:
}
```

3. **Action level** - By putting your filter on the top of the action name as shown below-

```
public class UserController : Controller
{
    [Authorize(Users = "User1,User2")]
    public ActionResult LinkLogin(string provider)
    {
        // TODO:
        return View();
    }
}
```

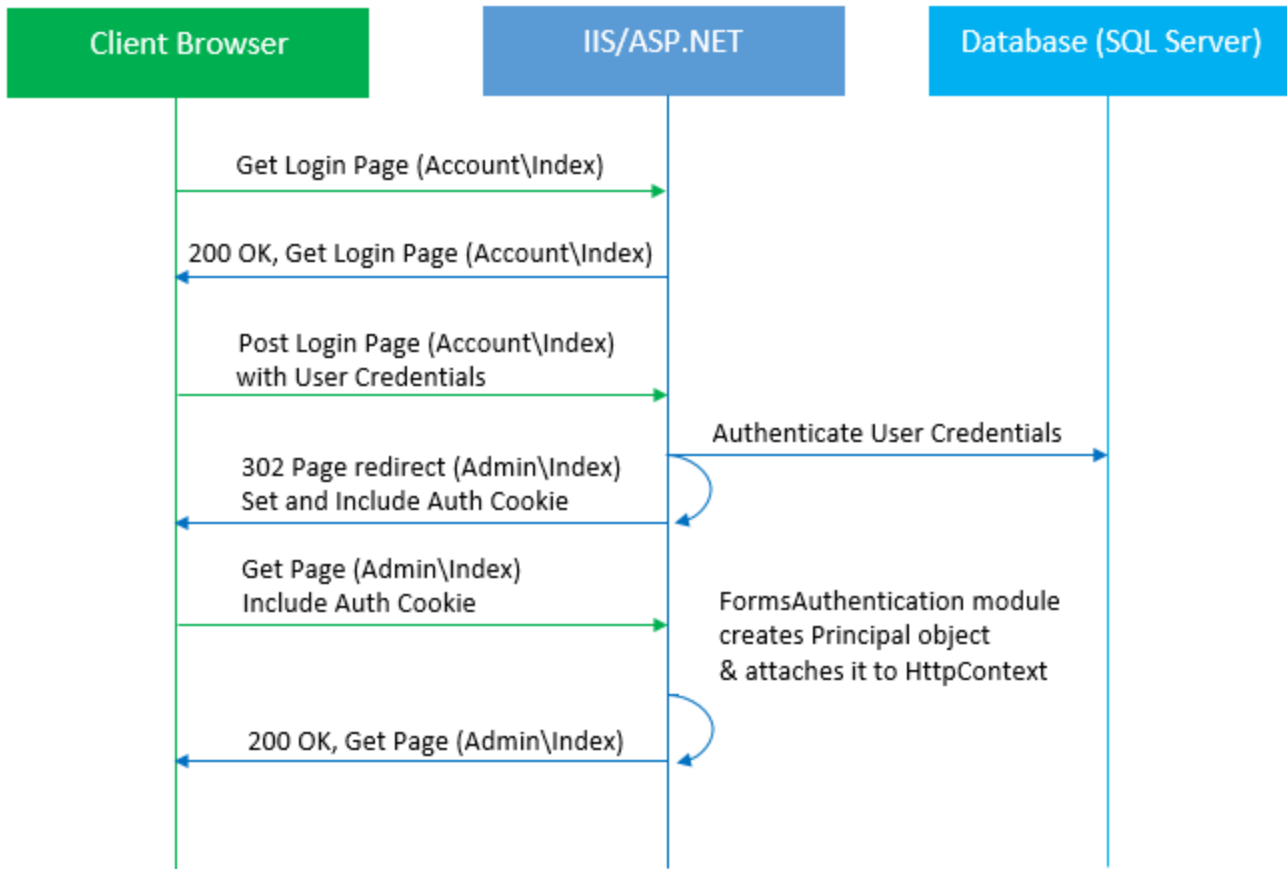
## Q6. How do Authentication and Authorization work in ASP.NET MVC?

**Ans.** Like ASP.NET, MVC also supports Windows and Forms authentication. You can configure both the authentications by using Web.config or doing some custom code.

## Q7. How Forms Authentication and Authorization work in ASP.NET MVC?

**Ans.** Like ASP.NET, MVC Forms authentication occurs after IIS authentication is completed. It can be configure by using forms element within Web.config file of your ASP.NET MVC application. The default attribute values for forms authentication are shown below:

```
<system.web>
<authentication mode="Forms">
  <forms loginUrl="Login.aspx"
  protection="All"
  timeout="30"
  name=".ASPXAUTH"
  path="/"
  requireSSL="false"
  slidingExpiration="true"
  defaultUrl="default.aspx"
  cookieless="UseDeviceProfile"
  enableCrossAppRedirects="false" />
</authentication>
</system.web>
```



Forms Authentication Control Flow

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.

You can create the *FormsAuthenticationTicket* object by specifying the cookie name, version of the cookie, directory path, issue date of the cookie, expiration date of the cookie, whether the cookie should be persisted, and optionally user-defined data as shown below:

```
FormsAuthenticationTicket ticket = new FormsAuthenticationTicket(1, "userName",
DateTime.Now,
DateTime.Now.AddMinutes(30), // value of time out property
false, // Value of IsPersistent property
String.Empty, FormsAuthentication.FormsCookiePath);
```

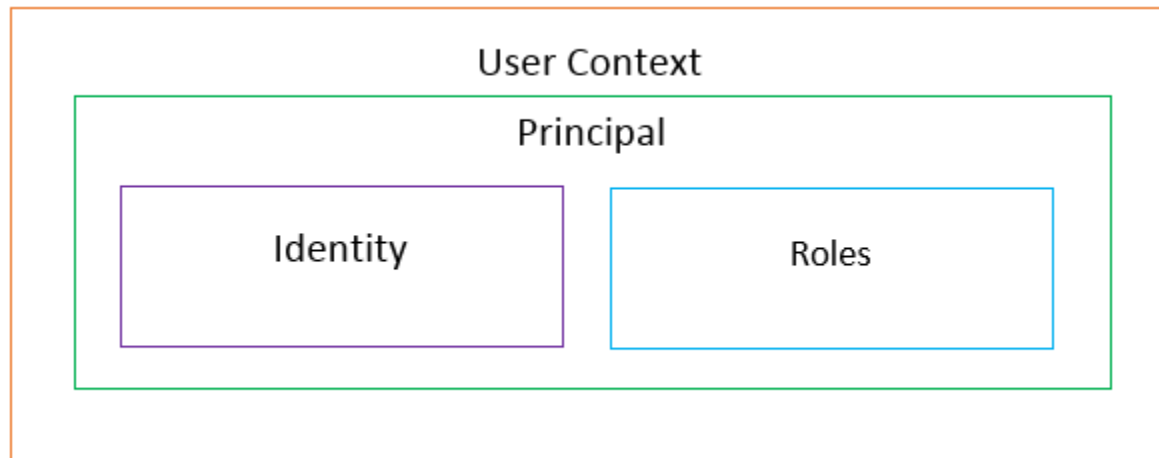
Now, you can encrypt this ticket by using the Encrypt method *FormsAuthentication* class as given below:

```
string encryptedTicket = FormsAuthentication.Encrypt(ticket);
```

## Q8. How to implement custom Forms Authentication and Authorization in MVC?

**Ans.** When standard types of authentication do not meet your requirements, you need to modify an authentication mechanism to create a custom solution. A user context has principal which represents the identity

and roles for that user. A user is authenticated by its identity and assigned roles to a user determine about authorization or permission to access resources.



ASP.NET provides `IPrincipal` and `Identity` interfaces to represents the identity and role for a user. You can create a custom solution by evaluating the `IPrincipal` and `Identity` interfaces which are bound to the `HttpContext` as well as the current thread.

```
public class CustomPrincipal: IPrincipal
{
    public IIdentity Identity { get; private set; }
    public bool IsInRole(string role)
    {
        if (roles.Any(r => role.Contains(r)))
        {
            return true;
        }
        else
        {
            return false;
        }
    }

    public CustomPrincipal(string Username)
    {
        this.Identity = new GenericIdentity(Username);
    }

    public int UserId { get; set; }
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public string[] roles { get; set; }
}
```

Now, you can put this CustomPrincipal object into the thread's CurrentPrincipal property and into the HttpContext's User property to accomplish your custom authentication and authorization process.

A user will be authenticated if IsAuthenticated property returns true. For authenticating a user you can use one of the following two ways:

1. **Thread.CurrentPrincipal.Identity.IsAuthenticated**
2. **HttpContext.Current.User.Identity.IsAuthenticated**

ASP.NET MVC provides Authorization filter to authorize a user. This filter can be applied to an action, a controller, or even globally. This filter is based on *AuthorizeAttribute* class. You can customize this filter by overriding *OnAuthorization()* method as shown below:

```
public class CustomAuthorizeAttribute : AuthorizeAttribute
{
    protected virtual CustomPrincipal CurrentUser
    {
        get { return HttpContext.Current.User as CustomPrincipal; }
    }

    public override void OnAuthorization(AuthorizationContext filterContext)
    {
        if (filterContext.HttpContext.Request.IsAuthenticated)
        {
            if (!String.IsNullOrEmpty(Roles))
            {
                if (!CurrentUser.IsInRole(Roles))
                {
                    filterContext.Result = new RedirectToRouteResult(new
                        RouteValueDictionary(new { controller = "Error", action =
"AccessDenied" }));

                    // base.OnAuthorization(filterContext); //returns to login url
                }
            }

            if (!String.IsNullOrEmpty(Users))
            {
                if (!Users.Contains(CurrentUser.UserId.ToString()))
                {
                    filterContext.Result = new RedirectToRouteResult(new
                        RouteValueDictionary(new { controller = "Error", action =
"AccessDenied" }));

                    // base.OnAuthorization(filterContext); //returns to login url
                }
            }
        }
    }
}
```

```
}  
}
```

Now you can apply this custom authorization filter at controller or action level for authorization as shown below:

```
[CustomAuthorize(Roles= "Admin")]  
public class AdminController : BaseController  
{  
    public ActionResult Index()  
    {  
        return View();  
    }  
}
```

### Q9. How to allow HTML tags in ASP.NET MVC?

Ans. By default, ASP.NET MVC doesn't allow a user to submit HTML for avoiding Cross Site Scripting attack to your application. You can achieve it by using `ValidateInput` attribute and `AllowHtml` attribute.

**ValidateInput attribute** can enable or disable input validation at the controller level or at any action method.

```
[ValidateInput(false)]  
public class HomeController: Controller  
{  
    public ActionResult AddArticle()  
    {  
        return View();  
    }  
}
```

**ValidateInput attribute** allows the Html input for all the properties and that is unsafe. Since you have enabled Html input for only one-two properties then how to do this. To allow Html input for a single property, you should use `AllowHtml` attribute.

```
public class BlogModel  
{  
    [Required]  
    [Display(Name = "Title")]  
    public string Title { get; set; }  
  
    [AllowHtml]  
    [Required]  
    [Display(Name = "Description")]  
    public string Description { get; set; }  
}
```

## Q10. What is caching and when to use it?

**Ans.** Caching is a most important aspect of the high-performance web application. Caching provides a way of storing frequently accessed data and reusing that data. Practically, this is an effective way for improving web application's performance.

### When to use caching

- Use caching for contents that are accessed frequently.
- Avoid caching for contents that are unique per user.
- Avoid caching for contents that are accessed infrequently/rarely.
- Use the VaryByCustom function to cache multiple versions of a page based on customization aspects of the request such as cookies, role, theme, browser, and so on.
- For efficient caching use 64-bit version of Windows Server and SQL Server.
- For database caching make sure your database server has sufficient RAM otherwise, it may degrade the performance.
- For caching of dynamic contents that change frequently, define a short cache–expiration time rather than disabling caching.

## Q11. What are the advantages of caching?

**Ans.** There are following advantages of caching:

- Reduce hosting server round-trips
- When content is cached at the client or in proxies, it causes the minimum request to the server.
- Reduce database server round-trips
- When content is cached at the web server, it can eliminate the database request.
- Reduce network traffic
- When content is cached at the client side, it also reduces the network traffic.
- Avoid time-consumption for regenerating reusable content
- When reusable content is cached, it avoids the time consumption for regenerating reusable content.
- Improve performance
- Since cached content reduces round-trips, network traffic and avoid time consumption for regenerating reusable content which causes a boost in the performance.

## Q12. What is output caching?

**Ans.** The OutputCache filter allows you to cache the data that is the output of an action method. By default, this attribute filter cache the data till 60 seconds. After 60 sec, ASP.NET MVC will execute the action method again and cache the output again.

```
class HomeController : Controller
{
    [OutputCache(Duration = 20, VaryByParam = "none")]
    public ActionResult Index()
    {
        ViewBag.Message = DateTime.Now.ToString();
    }
}
```



```
        return View();  
    }  
}
```

The output of the Index() action method will be cached for 20 seconds. If you will not define the duration, it will cache it for by default cache duration 60 sec.

### Output Caching Location

By default, content is cached in three locations: the web server, any proxy servers, and the user's browser. You can control the content's cached location by changing the location parameter of the OutputCache attribute to any of the following values: **Any, Client, Downstream, Server, None, or ServerAndClient.**

By default, the location parameter has the value Any which is appropriate for most the scenarios. But sometimes there are scenarios when you required more control over the cached data.

## **Q13. What is Donut caching and Donut hole caching in ASP.NET MVC?**

**Ans.** **Donut caching** cache an entire web page except for one or more parts of the web page. Before Donut caching, we have Output Caching which cache the entire web page.

### When to use Donut caching

Suppose, you have a web application in which some pages like HomePage, Tools etc. are same for all the users excepts the user's logged in details like the username.

If you want to cache all these pages for all the users by using OutputCache with VaryByParam UserID, then the entire page would be cached every time for each user with a different username (or whatever your dynamic part of the page is). This is not a good practice since there will be 1000 cached pages if there are 1000 logged in user at a time.



To resolve this issue, Donut Caching was introduced which cached only one copy of the entire page for all the user except for a small part which remains dynamic. This small part act like as a hole in the cached content and much like a donut.

Donut caching is very useful in the scenarios where most of the elements in your page are rarely changed except the few sections that dynamically change, or changed based on a request parameter.

### Donut Hole caching

Donut Hole Caching is the inverse of Donut caching means while caching the entire page it cached only a small part of the page (the donut hole).

### **When to use Donut Hole caching**

Suppose, you have a web application in which ProductCategory is shown on each and every page so it makes sense to render all of the categories just once and cache the resulting HTML by using Donut Hole Caching.

Donut Hole caching is very useful in the scenarios where most of the elements in your page are dynamic except the few sections that rarely change, or changed based on a request parameter. ASP.NET MVC has great support for Donut Hole caching through the use of Child Actions.

```
class HomeController : Controller
{
    [ChildActionOnly]
    [OutputCache(Duration = 60)]
    public ActionResult CategoriesList()
    {
        // Get categories list from the database and
        // pass it to the child view
        ViewBag.Categories = GetCategories();
        return View();
    }
}
```

# 6

## Dependency Injection

### Q1. What is loose coupling and how is it possible?

**Ans.** One of the most important features of the MVC design pattern is that it enables the separation of concerns. Hence you can make your application's components independent as much as possible. This is known as loose coupling, and it makes testing and maintenance of our application easier. Using Dependency Injection, you can make your application's components more loosely coupled.

### Q2. What are the Dependency Inversion Principle (DIP) and IoC?

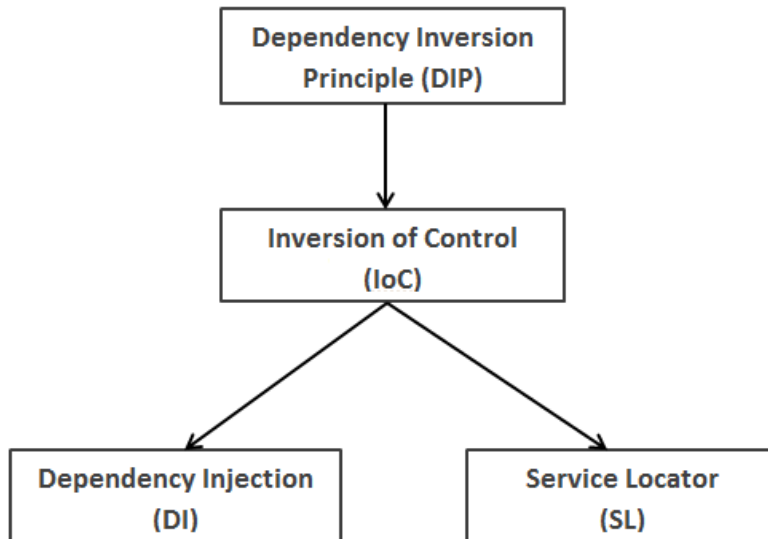
**Ans.** The Dependency Inversion Principle states that:

- High-level modules should not depend upon low-level modules. Both should depend upon abstractions.
- Abstractions should not depend upon details. Details should depend upon abstractions.

The Dependency Inversion Principle (DIP) helps us to develop loosely couple code by ensuring that high-level modules depend on abstractions rather than concrete implementations of lower-level modules. The Inversion of Control pattern is an implementation of this principle.

The term Inversion of Control (IoC) refers to a programming style where a framework or runtime, controls the program flow. Inversion of the control means we are changing the control from normal way. It works on the Dependency Inversion Principle. The most software developed on the .NET Framework uses IoC.

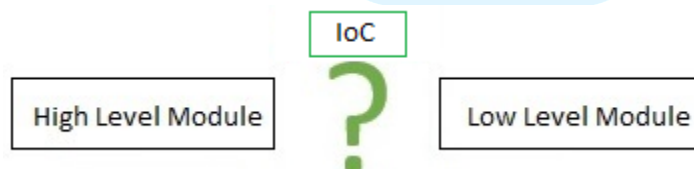
Moreover IoC is a generic term and it is not limited to DI. Actually, DI and Service Locator patterns are specialized versions of the IoC pattern or you can say DI and Service Locator are the ways of implementing IoC.



**For example**, suppose your Client class needs to use a Service class component, then the best you can do is to make your Client class aware of an IService interface rather than a Service class. In this way, you can change the implementation of the Service class at any time (and for how many times you want) without breaking the host code.

#### IoC and DIP

DIP says High-level module should not depend on the low level module and both should depend on abstraction. IoC is a way that provides abstraction. A way to change the control. IoC gives some ways to implement DIP. If you want to make an independent higher level module from the lower level module, then you have to invert the control so that low-level module does not control interface and creation of the object. Finally, IoC gives some way to invert the control.

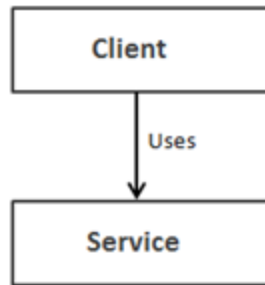


### Q3. What is Dependency Injection (DI)?

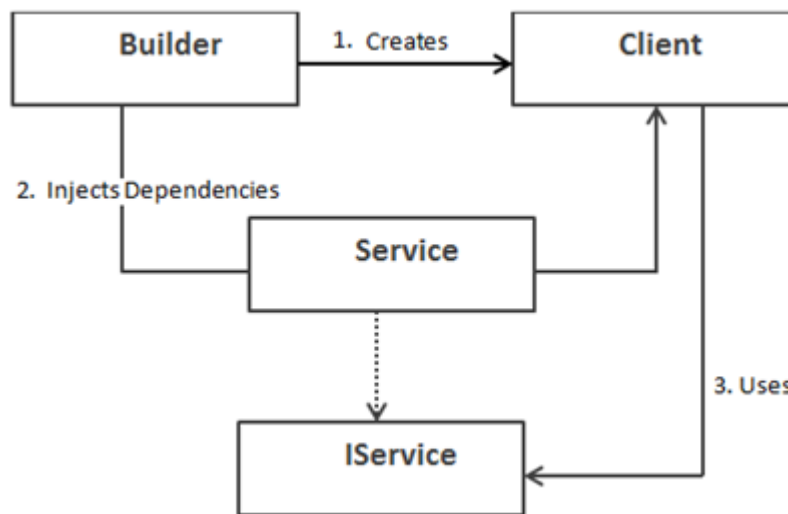
**Ans.** DI is a software design pattern that allows us to develop loosely coupled code. DI is a great way to reduce tight coupling between software components. DI also enables us to better manage future changes and other complexity in our software. The purpose of DI is to make code maintainable.

The Dependency Injection pattern uses a builder object to initialize objects and provide the required dependencies to the object means it allows you to "inject" a dependency from outside the class.

**For example**, suppose your Client class needs to use a Service class component, then the best you can do is to make your Client class aware of an IService interface rather than a Service class. In this way, you can change the implementation of the Service class at any time (and for how many times you want) without breaking the host code.



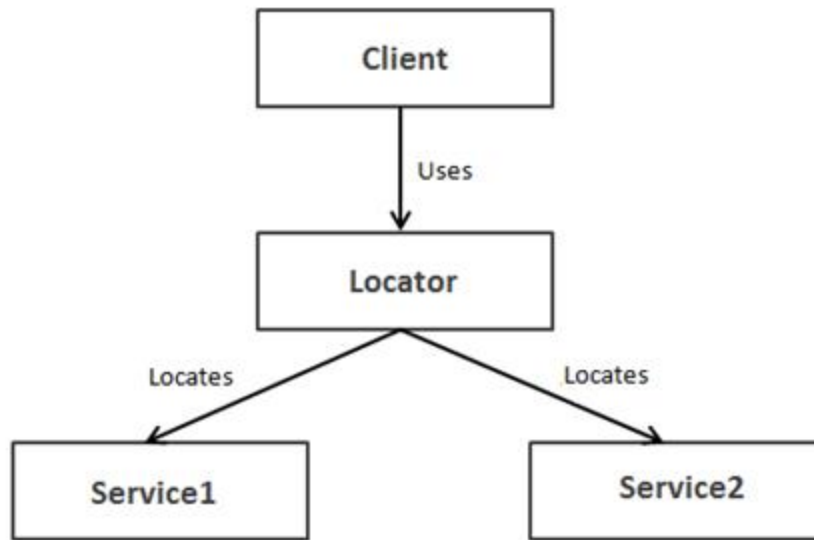
**DI Implementation:**



**Q4. What is Service Locator?**

**Ans.** Service Locator is a software design pattern that also allows us to develop loosely coupled code. It implements the DIP principle and easier to use with an existing codebase as it makes the overall design looser without forcing changes to the public interface.

The Service Locator pattern introduces a locator object that objects are used to resolve dependencies means it allows you to "resolve" a dependency within a class. Above example can be re-written as follows by using SL.



```
public interface IService
{
    void Serve();
}

public class Service: IService
{
    public void Serve()
    {
        Console.WriteLine("Service Called");
        //To Do: Some Stuff
    }
}

public static class LocateService
{
    public static IService _Service { get; set; }

    public static IService GetService()
    {
        if (_Service == null)
        {
            _Service = new Service();
        }

        return _Service;
    }
}

public class Client
{
    private IService _service;

    public Client()
    {

```

```

        this._service = LocateService.GetService();
    }

    public void Start()
    {
        Console.WriteLine("Service Started");
        this._service.Serve();

        //To Do: Some Stuff
    }
}
class Program
{
    static void Main(string[] args)
    {
        var client = new Client();
        client.Start();

        Console.ReadKey();
    }
}

```

## Q5. What are different ways to implement Dependency Injection (DI)?

Ans. There are three different ways to implement DI as given below:

- **Constructor Injection** - This is the most common DI. Dependency Injection is done by supplying the DEPENDENCY through the class's constructor when instantiating that class. The injected component can be used anywhere in the class. Should be used when the injected dependency is required for the class to function. It addresses the most common scenario where a class requires one or more dependencies.

```

public interface IService
{
    void Serve();
}

public class Service: IService
{
    public void Serve()
    {
        Console.WriteLine("Service Called");
        //To Do: Some Stuff
    }
}

public class Client
{
    private IService _service;

    public Client(IService service)

```

```

    {
        this._service = service;
    }

    public void Start()
    {
        Console.WriteLine("Service Started");
        this._service.Serve();
        //To Do: Some Stuff
    }
}
//Builder
class Program
{
    static void Main(string[] args)
    {
        Client client = new Client(new Service());
        client.Start();

        Console.ReadKey();
    }
}

```

- **Property Injection** – This is also called Setter injection. This is used when a class has optional dependencies, or where the implementations may need to be swapped. This is used by different logger implementations like Log4Net. It may require checking for a provided implementation throughout the class (need to check for null before using it). It does not require adding or modifying constructors.

```

public interface IService
{
    void Serve();
}

public class Service: IService
{
    public void Serve()
    {
        Console.WriteLine("Service Called");
        //To Do: Some Stuff
    }
}

public class Client
{
    private IService _service;

    public IService Service
    {
        set

```



```

        {
            this._service = value;
        }
    }

    public void Start()
    {
        Console.WriteLine("Service Started");
        this._service.Serve();
        //To Do: Some Stuff
    }
}
//Builder
class Program
{
    static void Main(string[] args)
    {
        Client client = new Client();
        client.Service = new Service();
        client.Start();

        Console.ReadKey();
    }
}

```

- **Method Injection** – This Inject the dependency into a single method, for use by that method only. It could be useful where the whole class does not need the dependency, just the one method.

```

public interface IService
{
    void Serve();
}

public class Service: IService
{
    public void Serve()
    {
        Console.WriteLine("Service Called");
        //To Do: Some Stuff
    }
}

public class Client
{
    private IService _service;

    public void Start(IService service)
    {
        this._service = service;
    }
}

```

```

        Console.WriteLine("Service Started");
        this._service.Serve();
        //To Do: Some Stuff
    }
}
//Builder
class Program
{
    static void Main(string[] args)
    {
        Client client = new Client();
        client.Start(new Service());

        Console.ReadKey();
    }
}

```

## Q6. What are the advantages of Dependency Injection (DI)?

**Ans.** There are following advantages of DI:

- Reduces class coupling
- Increases code reusing
- Improves code maintainability
- Improves application testing

## Q7. What is IoC or DI container?

**Ans.** The terms Dependency Injection (DI) & Inversion of Control (IoC) are generally used interchangeably to describe the same design pattern. Hence some people say IoC Container and some people says DI container but both terms indicate to the same thing. So, don't be confused by the terminology.

A DI Container is a framework to create dependencies and inject them automatically when required. It automatically creates objects based on the request and injects them when required. DI Container helps us to manage dependencies within the application in a simple and easy way.

We can also manage application dependencies without a DI Container, but it will be like as POOR MAN'S DI and we have to do more work, to make it configured and manageable.

## Q8. What are popular DI containers?

**Ans.** Today, there are a lot of excellent DI Containers that are available for .NET. The list of most useful DI container for the .NET framework is given below:

### Castle Windsor

- Based on the Castle MicroKernel.
- Well documented and used by many.
- Understands Decorator
- Typed factories

- Commercial support available

#### Spring.NET

- INTERCEPTION
- Comprehensive documentation
- Commercial support available

#### Autofac

- Easy to learn API
- second-generation DI Container
- Commercial support available

#### Unity

- INTERCEPTION
- Good documentation
- Consistent API

#### Ninject

- Easy to learn API
- Second-generation DI Container



### **Q9. What is Test Driven Development (TDD)?**

**Ans.** TDD is a methodology which says, write your tests first before you write your code. In TDD, tests drive your application design and development cycles. You do not do the check-in of your code into source control until all of your unit tests pass.

### **Q10. What are the commonly used tool for Unit Testing in ASP.NET MVC?**

**Ans.** ASP.NET MVC has been designed for testability without dependencies on the IIS server, on a database, or on external classes. There are following popular tools for ASP.NET MVC testing:

- **NUnit** - This is the most popular unit testing frameworks for Microsoft .NET. Its syntax is relatively simple and easy to use. It comes with a test runner GUI and a command-line utility. NUnit is also available as a NuGet package for download.
- **xUnit.NET** - This provides a way to run automated unit tests. It is simple, easily extended, and has a very clean syntax.
- **Ninject 2** - This provides a way to wire up classes in your application.
- **Moq** - This provides a framework for mocking interfaces and classes during testing.

# References

This book has been written by referring to the following sites:

1. <https://docs.microsoft.com/en-us/aspnet/mvc/> - Microsoft Docs - ASP.NET
2. <https://stackoverflow.com/questions/tagged/asp.net-mvc> - Stack Overflow - ASP.NET MVC
3. <https://www.dotnettricks.com/learn/mvc> - Dot Net Tricks – ASP.NET MVC

