

## INTERVIEW QUESTION

### 1. Explain the joins?

Ans : A JOIN clause is used to combine rows from two or more tables, based on a related column between them.

Types of JOINS in SQL Server :

SQL Server mainly supports four types of JOINS, and each join type defines how two tables are related in a query. The following are types of join supports in SQL Server:

1. INNER JOIN
  2. SELF JOIN
  3. CROSS JOIN
  4. OUTER JOIN
- 

### 2. Difference between left and right outer join?

<b>Left Outer Join:</b>
-------------------------

- |   |
|---|
| <ul style="list-style-type: none"><li>• In a left outer join, all rows from the left table (the table mentioned first in the join clause) are returned, along with matching rows from the right table.</li><li>• If no matching rows are found in the right table, NULL values are returned for the columns of the right table.</li><li>• The left outer join ensures that all rows from the left table are preserved in the result set, even if there are no matches in the right table.</li></ul> |
|---|

<b>Right Outer Join:</b>
--------------------------

- |  |
|--|
| <ul style="list-style-type: none"><li>• In a right outer join, all rows from the right table (the table mentioned second in the join clause) are returned, along with matching rows from the left table.</li><li>• If no matching rows are found in the left table, NULL values are returned for the columns of the left table.</li><li>• The right outer join ensures that all rows from the right table are preserved in the result set, even if there are no matches in the left table.</li></ul> |
|--|

## Example :

Now, **1. Left Outer Join query** –

```
Select empid, ename, deptid, deptname  
from employee  
left outer join department  
on employee.empdept = department.deptname;
```

Output:

EMPID	ENAME	DEPTID	DEPTNAME
101	Amanda	10	Development
103	Bruce	11	Designing
104	Steve	12	Testing
102	Diana	null	null
105	Roger	null	null

**2. Right Outer Join query** –

```
Select empid, ename, deptid, deptname  
from employee right outer join department  
on employee.empdept = department.deptname;
```

EMPID	ENAME	DEPTID	DEPTNAME
101	Amanda	10	Development
103	Bruce	11	Designing
104	Steve	12	Testing
null	null	13	HelpDesk

**Example:** Consider following employee table,

EMPID	ENAME	EMPDEPT	SALARY
101	Amanda	Development	50000
102	Diana	HR	40000
103	Bruce	Designing	30000
104	Steve	Testing	35000
105	Roger	Analyst	10000

Department Table :

DEPTID	DEPTNAME	LOCATION
10	Development	New York
11	Designing	New York
12	Testing	Washington
13	HelpDesk	Los Angeles

---

### 3. What is Procedure and function

Ans : A Stored Procedure is a group of Transact-SQL statements. Stored procedures are precompiled SQL statements that are stored in a database and can be called from an application.

#### **Benefits of Using Stored Procedures in ASP.NET**

- **Improved Performance**
- **Better Security**
- **Easier Maintenance**

### Syntax For creating Stored Procedure :

```
CREATE PROCEDURE spGetDevelopers()  
BEGIN  
    select FullName,Gender from Developers;  
END
```

---

4. How many types of functions are there?

Ans : A function is a group of related instructions that performs a specific task. It can be a small or big task, but the function will perform that task completely.

Functions take some input as parameters and return the result as a return value.

#### Types of Functions in C#:

Basically, there are two types of Functions in C#. They are as follows:

1. Built-in Functions
  2. User-Defined Functions
- 

5. Is it possible to use stored procedure in functions?

Ans : **Stored Procedures can't be called inside a function** because functions would be called by a select command and Stored Procedures couldn't be called by select command. And Store Procedure only execute by using exec/execute.

Eg : `exec Procedure_Name // Execute Procedure_Name // Procedure_Name`

There are different ways to execute a stored procedure. The first and most common approach is for **an application or user to call the procedure**. Another approach is to set the stored procedure to run automatically when an instance of SQL Server starts

---

6. How many constraints are there in Sql?

Ans :- In SQL Server, there are typically six types of constraints that can be used to enforce data integrity rules. These constraints are as follows:

1. Primary Key Constraint: Ensures that each value in a column or a combination of columns is unique and not null. It is used to identify each row uniquely in a table.

2. Unique Constraint: Ensures that each value in a column or a combination of columns is unique. Unlike the primary key constraint, it allows null values (except for columns included in the constraint).

3. Foreign Key Constraint: Establishes a relationship between two tables by enforcing referential integrity. It ensures that values in a column (foreign key) of one table match the values in a primary key column (referenced key) of another table.

4. Check Constraint: Enforces that the values in a column meet specific conditions or expressions. It allows you to define custom rules to restrict the values stored in a column.

5. Default Constraint: Sets a default value for a column when no value is specified during an INSERT operation.

6. Not Null Constraint: Ensures that a column does not contain null values. It requires a value to be provided for the column during an INSERT operation.

---

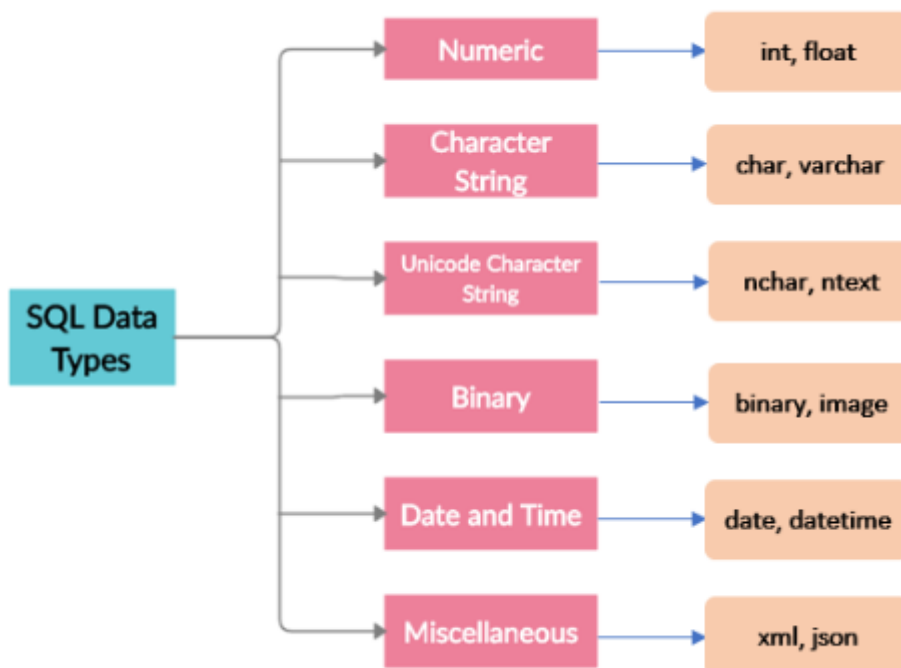
7. Difference between primary key and unique constraints?

Ans :

PRIMARY KEY	UNIQUE KEY
<ul style="list-style-type: none"><li>• Helps to identify a unique row from a table.</li><li>• Does not allow null values</li><li>• One per table</li><li>• Unique Index is created on the column where PK is defined.</li><li>• Foreign Key can refer to PK</li><li>• Column Level and Table Level</li></ul>	<ul style="list-style-type: none"><li>• Helps to maintains Unique data in a column of a table.</li><li>• Nulls are allowed</li><li>• Multiple per table</li><li>• Unique Index is created on the column where UK is defined.</li><li>• Foreign Key can refer to UK.</li><li>• Column Level and Table Level</li></ul>

8. How many datatypes in Sql?

Ans : A data type is an attribute that specifies the type of data that the object can hold: integer data, character data, monetary data, date and time data, binary strings, and so on.



## 9. Difference between clustered index and Non-Clustered Index?

Ans:

Clustered Index : A clustered index is used to define the order or to sort the table or arrange the data by alphabetical order just like a dictionary.

Non-Clustered Index : A non-clustered index collects the data at one place and records at another place. It is faster than a non-clustered index. It is slower than the clustered index.

	Clustered Index	Non-Clustered Index
Storage	The actual data rows are physically sorted and stored in the order of the clustered index key.	The index contains a copy of the indexed columns and a pointer to the actual data row.
Sorting	Only one clustered index can be created per table.	Multiple non-clustered indexes can be created per table.
Key Uniqueness	A clustered index key must be unique.	A non-clustered index key can have duplicate values.
Impact on Data Modification	Data modification operations (inserts, updates, deletes) can be slower since the physical order of the rows may need to be rearranged.	Data modification operations are generally faster as they only update the index, not the actual data rows.

---

## Candidate #2

### 1. Introduce Yourself

Ans:

First of all a very warm Good Evening/Morning/afternoon mam/sir so,Thankyou for giving me this opportunity to introduce myself in front of you ----

"my name is [Your Name]. I am a .NET developer with [X years] of experience in designing and implementing software solutions using the .NET framework.

Throughout my career, I have had the opportunity to work on various projects like Ecommerce using Asp.net webform, Hrms Management system using same also I had worked on some modules of Franchise Apk.

I have a strong foundation in C#, ASP.NET, .NET Core, Entity Framework, etc.], and I'm passionate about leveraging these technologies to build robust and scalable applications that meet the needs of users and stakeholders.

In addition to my technical skills, I also have experience collaborating with cross-functional teams, communicating complex technical concepts

I am excited about the opportunity to contribute my expertise and skills to [company name] and be a part of [specific project or team, if known].

That's all about myself.

---

### 2. Different technologies you used

Ans: We have worked/used on various technologies like C#(Console based E-commerce project), Asp.net webform, Asp.net core , Asp.net Mvc, Ajax , JQuery ,Bootstrap ,Angular etc during our internship period.

---

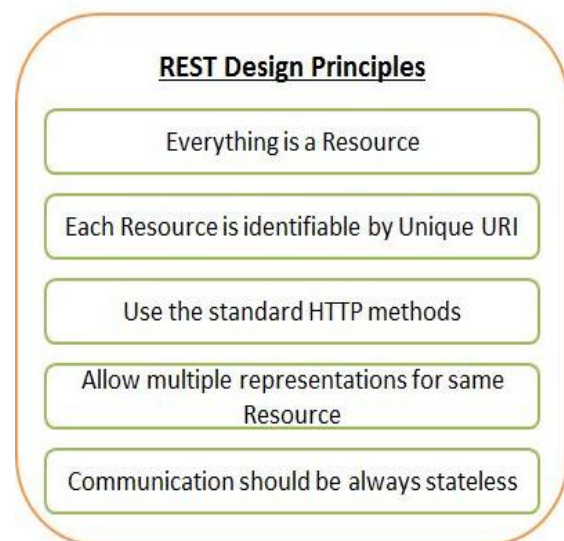
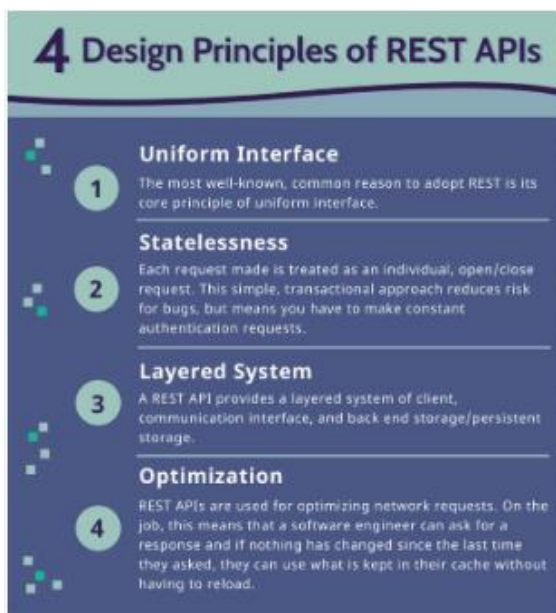


### 3. What are the design principles of Rest api's, Web api's ?

Ans:

1. **REST (Representational State Transfer)**: It is an architectural style for designing networked applications. RESTful systems typically communicate over HTTP using standard operations like GET, POST, PUT, DELETE, etc., and they use resources identified by URLs. RESTful APIs are designed to be stateless, meaning that each request from a client to the server must contain all the information necessary to understand and fulfill the request.
2. **Web API**: This term typically refers to a framework that allows you to build HTTP services that reach a broad range of clients, including browsers and mobile devices. In the context of .NET development, ASP.NET Web API is a framework for building RESTful HTTP services.

#### Design Principle of Rest Api:



#### 4. Difference between Post, Put and Patch ?

Ans:

PUT, POST, and PATCH are HTTP methods used to interact with web services.

##### 1. PUT (Create or Replace):

**Purpose:** The PUT method is used to create a new resource or replace an existing resource with a new representation.

**Usage:** When using PUT, the entire representation of the resource is sent in the request payload. If the resource already exists, it is replaced entirely with the new representation sent in the request.

##### 2. POST (Create or Append):

**Purpose:** The POST method is used to submit data to be processed by the resource identified by the URI. It can be used to create a new resource or append data to an existing resource.

**Usage:** When using POST, the request payload contains the data to be processed or appended.

##### 3. PATCH (Update):

**Purpose:** The PATCH method is used to partially update an existing resource. It applies modifications to the resource, rather than replacing it entirely.

**Usage:** When using PATCH, the request payload contains instructions on how the resource should be modified.

---

5.What is Restless and Restful services?

Ans:

- Restful services use rest architectural style
- restless services use soap protocol
- if we are creating an api we can create an api as a restful api or restless api as mention above.

**Guiding principle of rest :**

**1)Uniform interface**

(client(Http Request) -> Http methods(get,post,put,delete,patch) -> server(Httpresponse))

**2)client-server**

**3)stateless :** (server won't store any information about the client so server treats every request as a new request in stateless architecture) in this case we send a token or

every possible content in a url although we can use JWT i.e token based auth while implementing stateless services.

**4)Cacheable :** means(sometimes server send response from cache because server stores some information in the form of cache basically server stores cache info in the 'Redis' db i.e Nosql Database system)

**5)Layered System**

**6)Code on Demand**

### Data Formatting in Restful/Restless or Differences:

- Restful supports various Data Formats such as HTML,JSON,TEXT, etc.
- Restless Supports XML
- Restful is easier and flexible
- Restless is not easy and flexible
- Restful consumes less bandwidth and resource
- Restless consumes more bandwidth and resource

if we are sending a data in the form of Xml then we are using Soap api for sending data to server.

### Different types of architectural pattern for creating apis :

- **GraphQL** is not just an architectural style but also a query language, allowing clients to ask for specific data
- **SOAP** is a messaging protocol specification for exchanging structured information in the implementation of web services in computer networks.
- **REST** is a software architectural style that was created to guide the design and development of the architecture for the World Wide Web.
- **WebSocket** is a technology that revolutionizes communication on the web, enabling real-time and bidirectional connections between clients and server
- **gRPC** (Google Remote Procedure Call): **gRPC** is a modern and high-performance API architecture style
- **Webhooks** are commonly used in various scenarios where event-driven communication and asynchronous processing are desired.
- Remote Procedure Call. **RPC** is an architectural style for distributed systems.

- **Microservices** architecture pattern is seen as a viable alternative to monolithic applications
- 

6. Is it possible to create a resource using put

Ans:

The HTTP PUT method is used to create a new resource or replace a resource. It's similar to the POST method, in that it sends data to a server, but it's idempotent. This means that the effect of multiple PUT requests should be the same as one PUT request.

---

## 7. Different return types of web api's

Ans:

Return type	How Web API creates the response
void	Return empty 204 (No Content)
HttpResponseMessage	Convert directly to an HTTP response message.
IActionResult	Call <code>ExecuteAsync</code> to create an <code>HttpResponseMessage</code> , then convert to an HTTP response message.
Other type	Write the serialized return value into the response body; return 200 (OK).



## 8. How to validate a model on server side

Ans:

To Validate a Model on server side we will use Data Annotation

**Below is My Model Class :**

public class Product

{

public int Id { get; set; }

[Required(ErrorMessage = "Please enter a name.")]

public string Name { get; set; }

```
[Range(0, 100, ErrorMessage = "Price must be between 0 and 100.")]
public decimal Price { get; set; }
}
```

**Below is My Action:**

```
[HttpPost]

public IActionResult Create(Product product)
{
    if (ModelState.IsValid)
    {
        // Process the data

        // Save to database, etc.

        return RedirectToAction("Index");
    }

    return View(product); // If model state is not valid, return the view with
validation errors
}
```

**Below is My View :**

```
<form asp-action="Create">

    <div class="form-group">

        <label asp-for="Name"></label>

        <input asp-for="Name" class="form-control" />
```

```
<span asp-validation-for="Name" class="text-danger"></span>
```

```
</div>
```

```
<div class="form-group">
```

```
<label asp-for="Price"></label>
```

```
<input asp-for="Price" class="form-control" />
```

```
<span asp-validation-for="Price" class="text-danger"></span>
```

```
</div>
```

```
<button type="submit" class="btn btn-primary">Submit</button>
```

```
</form>
```

```
@section Scripts {
```

```
<partial name="_ValidationScriptsPartial"/>
```

"\_ValidationScriptsPartial" this is my inbuilt class in MVC Directory Structure.

```
}
```

---



## 9. What are filters in MVC

Ans:

ASP.NET MVC Filter is a custom class where you can write custom logic to execute before or after an action method executes.

MVC provides different types of filters.

Filter Type	Description	Built-in Filter	Interface
Authorization filters	Performs authentication and authorizes before executing an action method.	[Authorize], [RequireHttps]	IAuthorizationFilter
Action filters	Performs some operation before and after an action method executes.		IActionFilter
Result filters	Performs some operation before or after the execution of the view.	[OutputCache]	IResultFilter
Exception filters	Performs some operation if there is an unhandled exception thrown during the execution of the ASP.NET MVC pipeline.	[HandleError]	IExceptionHandler

---

## 10. How do you implement global exception handling in MVC or Web Api's

Ans:

In ASP.NET Core, global exception handling is typically implemented using middleware. Middleware is a pipeline component that can inspect, modify, or terminate incoming requests and outgoing responses. You can create custom middleware to handle exceptions and register it in the application pipeline.

**Key benefits of using global exception handling**

- Consistent Error Responses
- Improved User Experience
- Application Resilience
- Logging and Monitoring
- Security

## Install the required NuGet packages :

Microsoft.AspNetCore.Diagnostics.EntityFrameworkCore

Microsoft.AspNetCore.Mvc.NewtonsoftJson

---

### 11. What is difference between ViewData, ViewBag, and TempData ?

Ans:

**ViewBag :** ViewBag is a dynamic object to pass the data from the Controller to View. This will pass the data as a property of the object ViewBag.

Syntax :

```
Public ActionResult Index()  
{  
    ViewBag.Title = "Welcome";  
    return View();  
}
```

**ViewData :** ViewData is a dictionary object to pass the data from Controller to View, where data is passed in the form of a key-value pair. Typecasting is required to read the data in View if the data is complex, and we need to ensure a null check to avoid null exceptions.

Syntax :

```
public IActionResult Index()
```

```

{
    ViewData["Message"] = "Hello, world!";
    return View();
}

@{
    // Retrieve the value from ViewData and cast it to string
    string message = ViewData["Message"] as string;
}

<div>@message</div>

```

**TempData** : TempData is a feature that allows you to pass data from the current request to the next request. TempData is useful when you need to transfer data between actions, especially when using the Post-Redirect-Get (PRG) pattern.

TempData is similar to ViewData, but it has a lifespan that spans beyond a single request. It is typically used to store data for the duration of the user's session or until it is read.

Syntax :

```

public class HomeController : Controller
{
    public IActionResult Index()
    {
        TempData["Message"] = "Welcome to the homepage!";
        return RedirectToAction("About");
    }

    public IActionResult About()
    {
        // Retrieve data from TempData
    }
}

```

```
        string message = TempData["Message"] as string;
        ViewBag.Message = message;
        return View();
    }
}
```

---

## 12. What are partial views

Ans:

Partial views are a feature in ASP.NET MVC and ASP.NET Core MVC that allow us to break down complex views into smaller, reusable components.

Here are some key points about partial views:

- **Reusability**
- **Encapsulation**
- **Separation of concerns**
- **Rendered using `Html.Partial` or `Html.RenderPartial`**
- **Strongly typed or untyped**

Suppose you have a partial view named `_Navbar.cshtml` that contains navigation links:

```
<!-- _Navbar.cshtml -->
```

```
html

<!-- _Navbar.cshtml -->
<nav>
  <ul>
    <li><a href="/">Home</a></li>
    <li><a href="/about">About</a></li>
    <li><a href="/contact">Contact</a></li>
  </ul>
</nav>
```

We can then include this partial view in our main view (`Index.cshtml`, for example) using `@Html.Partial()`:

```
html

<!-- Index.cshtml -->
<!DOCTYPE html>
<html>
<head>
  <title>My Website</title>
</head>
<body>
  @Html.Partial("_Navbar")

  <div>
    <h1>Welcome to My Website</h1>
    <!-- Other content here -->
  </div>
</body>
</html>
```



## Strongly Type Views :

csharp

```
public class Product
{
    public int Id { get; set; }
    public string Name { get; set; }
    public decimal Price { get; set; }
}
```

In our controller action method, we can pass an instance of the `Product` class to the view:

csharp

```
public IActionResult Details()
{
    var product = new Product
    {
        Id = 1,
        Name = "Laptop",
        Price = 999.99m
    };

    return View(product);
}
```

In our strongly typed view (**Details.cshtml**), we can declare the model type at the top of the file and access properties of the model directly:

```
html

@model YourNamespace.Product

<!DOCTYPE html>
<html>
<head>
    <title>Product Details</title>
</head>
<body>
    <h1>@Model.Name</h1>
    <p>Price: $@Model.Price</p>
</body>
</html>
```

This is my model product and this is the way to create strongly typed views

---

### 13. Difference between User control and Custom control in Asp.net?

Ans:

In ASP.NET, both user controls and custom controls are ways to encapsulate reusable pieces of UI functionality or content, but they differ in their implementation and usage.

**UserControl** : A control which can reuse the Components in the Applications. The control can be defined in both Xaml and Code-Behind.

**CustomControl** : An UserInterface element that have a distinct behavior which is said as CustomControl.

### User control

- 1) Reusability web page
- 2) We can't add to toolbox
- 3) Just drag and drop from solution explorer to page (aspx)
- 4) U can register user control to. Aspx page by Register tag
- 5) A separate copy of the control is required in each application
- 6) Good for static layout
- 7) Easier to create
- 8) Not compiled into DLL
- 9) Here page (user page) can be converted as control then  
We can use as control in aspx

### Custom controls

- 1) Reusability of control (or extend functionalities of existing control)
- 2) We can add to toolbox
- 3) Just drag and drop from toolbox
- 4) U can register user control to. Aspx page by Register tag
- 5) A single copy of the control is required in each application
- 6) Good for dynamic layout
- 7) Hard to create
- 8) Compiled into dll

---

## 14. How to create a Custom Control/file

Ans:

Creating a custom control in ASP.NET allows you to encapsulate functionality and reuse it across different pages or projects.

Steps For creating Custom Control File :

Step 1 : **Create a new Class Library project:**

Step 2 : **Define your custom control:**

Step 3 : **Build your project:**

Step 4 : **Use the custom control:**



## Step 5 : **Deploy your application:**

To use your custom control or file in an ASP.NET page or another control, you need to register it first. This involves adding a **Register** directive at the top of your ASPX page:

asp

Copy code

```
<%@ Register TagPrefix="Custom" TagName="MyControl" Src="~/PathToYourControl/YourCon
```

After registering, you can use your custom control in the page markup like any other ASP.NET control:

asp

Copy code

```
<Custom:MyControl ID="myCustomControl" runat="server" />
```

---

## 14. Class you should inherit to create custom controller

Ans:

In ASP.NET, to create a custom server control, you typically inherit from one of the base classes provided by the ASP.NET framework.

Here are some common base classes for creating custom server controls in ASP.NET:

**System.Web.UI.Control:**

**System.Web.UI.WebControls.WebControl:**

**System.Web.UI.WebControls.BaseDataBoundControl:**

**System.Web.UI.WebControls.CompositeControl:**

**System.Web.UI.WebControls.Adapters.ControlAdapter:**

---

## 15. Difference between GridView and DataList

Ans:

The GridView and DataList are both data-bound controls in ASP.NET used for displaying data from a data source, but they have some differences in terms of features, flexibility, and layout

GridView	DataList
<b>Layout:</b> The GridView control provides a tabular layout, similar to a traditional HTML table. It is suitable for displaying data in a row-column format where each record occupies a separate row, and each field occupies a separate column.	<b>Layout:</b> The DataList control provides more flexibility in terms of layout. It allows you to define the layout template using templates such as ItemTemplate, AlternatingItemTemplate, HeaderTemplate, FooterTemplate, etc. This makes it more versatile for creating custom layouts compared to the GridView.
<b>Customization:</b> While the GridView control provides some level of customization through properties and styles, it has a predefined structure with limited flexibility in terms of layout customization.	<b>Customization:</b> The DataList control offers more customization options through its templated structure. You can define custom templates for items, alternating items, headers, footers, etc., allowing for more control over the layout and appearance of the data.
<b>Performance:</b> Due to its tabular structure and limited customization options, the GridView control may offer better performance in scenarios where a simple tabular layout is sufficient.	<b>Performance:</b> The DataList control, with its templated structure and flexibility, may incur slightly higher overhead compared to the GridView, especially if you're using complex templates and custom layouts.

---

## 16. Method signature while writing extension method

Ans:

the method signature typically consists of the following components:

1. **Access Modifier:**
2. **Static Modifier**
3. **Return Type:**
4. **This Keyword:**
5. **Method Name:**
6. **Parameters:**

### Advantages:

- The main advantage of the extension method is to add new methods in the existing class without using [inheritance](#).
- You can add new methods in the existing class without modifying the source code of the existing class.
- It can also work with [sealed class](#).

Syntax:

```
public static class MyExtensionMethods
{
    public static ReturnType MyExtensionMethod(this ExtendedType
extendedObject, parameters)
    {
        // Method implementation
    }
}
```

## 17. Difference between Finalize and Dispose method

Ans:

**Finalize()** and **Dispose()** are both methods used in .NET for resource cleanup.

BASIS FOR COMPARISON	DISPOSE( )	FINALIZE( )
Defined	Defined in the interface Disposable interface	Defined in the interface Disposable interface
Syntax	<pre>public void Dispose(){     // Dispose code here }</pre>	<pre>protected void finalize(){     // finalization code here }</pre>
Invoked	Invoked by the user	Invoked by the garbage collector
Purpose	Used to free unmanaged resources whenever it is invoked	Used to free unmanaged resources before the object is destroyed
Implementation	Implemented whenever there is a close( ) method	Implemented for unmanaged resources
Access specifier	Declared as public	Declared as private
Action	Faster in disposing objects	Slower compared to dispose
Performance	Performs an instantaneous action that doesn't affect the performance of websites	Affects the performance of the websites because of its slow pace

---

## 18. Difference between Array and ArrayList and List

Ans:

Array	ArrayList
An Array is strongly-typed. We can store only the same type of data.	ArrayList is a non-generic collection type. ArrayList's internal Array is of the object type. So, we can store multiple types of data in ArrayList.
Array stores a fixed number of elements.	ArrayList is dynamic in term of capacity. If the number of element exceeds, ArrayList will increase to double its current size.
Array provides better performance than ArrayList.	If we are using a large number of ArrayList then it degrades performance because of boxing and unboxing.
Array uses static helper class Array which belongs to system namespace	ArrayList implements an IList interface so, it provides a method that we can use for easy implementation.
Array belongs to namespace System	ArrayList belongs to namespace System.Collection
The Array cannot accept null.	An Array can accept null.
<b>Example:</b> string[] array1=new string[5]; array1[0]="Hello"; array1[1]="Bye";	<b>Example:</b> ArrayList a1=new ArrayList(); a1.add(null); a1.insert(1,"hi"); a1.add(3); a1.add(8.23);

---

## 19. What are CTE's in sql server

Ans:

CTE stands for Common Table Expression in SQL Server. It's a temporary result set that you can reference within a SELECT, INSERT, UPDATE, or DELETE statement. CTEs are defined within the execution scope of a single SELECT, INSERT, UPDATE, or DELETE statement.

### 1. Syntax:

```
sql                                                                    Copy code
WITH cte_name (column1, column2, ...)
AS
(
    -- CTE query definition here
)
```

Example of a simple CTE in SQL Server:

```
sql Copy code  
  
WITH EmployeesCTE (EmployeeID, FirstName, LastName) AS  
(  
    SELECT EmployeeID, FirstName, LastName  
    FROM Employees  
    WHERE DepartmentID = 1  
)  
SELECT *  
FROM EmployeesCTE;
```

---

20. It is possible to write insert statement in sql server function?

Ans:

Yes, it is possible to use 'Insert' statement in a SQL Server function, but with some limitations in a function we can't use output clause and the function must return a result.

```
CREATE FUNCTION dbo.InsertData(@Name VARCHAR(50))  
RETURNS INT  
AS  
BEGIN  
    DECLARE @ID INT  
  
    INSERT INTO MyTable (Name) VALUES (@Name)  
    SET @ID = SCOPE_IDENTITY() -- get the ID of the inserted row  
  
    RETURN @ID  
END
```

```
SELECT dbo.InsertData('John Doe')
```

---

## 21. Difference between Delete and Truncate, which is Faster?

Ans:

In database management systems, "DELETE" and "TRUNCATE" are two distinct commands used to remove data from a table.

### **DELETE:**

- The DELETE command removes one or more rows from a table based on a specified condition.
- It is a DML (Data Manipulation Language) operation.
- DELETE command operations are logged in the transaction log, making it possible to roll back changes if necessary (if used within a transaction).
- It causes triggers defined on the table to be fired.
- DELETE allows you to specify conditions for removing specific rows, making it more flexible than TRUNCATE.

### **TRUNCATE:**

- The TRUNCATE command removes all rows from a table.
- It is a DDL (Data Definition Language) operation.
- TRUNCATE removes the data by deallocating the data pages used by the table, which is much faster than deleting rows one by one.
- TRUNCATE operations are not logged in the transaction log (or the log entries are minimal), which makes it non-recoverable.
- TRUNCATE resets any identity columns back to their seed value.
- TRUNCATE cannot be used when a table is referenced by a foreign key constraint.

In terms of speed, TRUNCATE is generally faster than DELETE, especially for large tables, because TRUNCATE doesn't need to log individual row deletions and it doesn't invoke triggers.

---

## 22. Difference between GroupBy and Having clauses?

Ans:

**GROUP BY** clause is used to group rows that have the same values in one or more columns

It is used in conjunction with aggregate functions such as SUM(), COUNT(), AVG(), MAX(), or MIN() to perform calculations on groups of rows.

**HAVING clause** is used to filter groups of rows returned by the GROUP BY clause.

It operates on the grouped data rather than individual rows.

HAVING is similar to the WHERE clause, but WHERE filters individual rows before they are grouped, whereas HAVING filters groups after they have been grouped.

Having Clause	GroupBy Clause
It is used for applying some extra condition to the query.	The groupby clause is used to group the data according to particular column or row.
Having cannot be used without groupby clause,in aggregate function,in that case it behaves like where clause.	groupby can be used without having clause with the select statement.
The having clause can contain aggregate functions.	It cannot contain aggregate functions.
It restrict the query output by using some conditions	It groups the output on basis of some rows or columns.

---



## Candidate #4

### 1. What is the ASP.NET Web.config file used for?

Ans:

A configuration file (web.config) is used to manage various settings that define a website. The settings are stored in XML files that are separate from your application code. In this way you can configure settings independently from your code. Generally a website contains a single Web.config file stored inside the application root directory.

#### **Benefits of XML-based Configuration files :**

- ASP.NET Configuration system is extensible and application specific information can be stored and retrieved easily. It is human readable.
- We don't want to restart the web server when the settings are changed in configuration file. ASP.NET automatically detects the changes and applies them to the running ASP.NET application. **(V-V-IMP POINT)**

```
<configuration>
  <connectionStrings>
    <add name="myCon" connectionString="server=MyServer;database=puran;uid=puranme" />
  </connectionStrings>
</configuration>
```

- Here are some common uses of the Web.config file:

Application Settings // Security Settings // Session State Management // Error Handling and Debugging // Http Handlers and Module // Compilation Setting // Caching Setting.

---

2. Explain the role of the App\_Code folder in an ASP.NET application.

Ans:

The App\_Code folder in an ASP.NET application serves as a central location for storing source code files that contain classes, business logic, data access logic, and other reusable components that are used throughout the application.

All items stored in App\_Code are automatically accessible throughout the application.

#### **Where do we see this Folder :**

Create New Project -> Asp.Net Web Application(.Net Framework) -> Select MVC ->

Now App\_Data is visible in Directory.

#### **Here are the key aspects and roles of the App\_Code folder:**

Organizing Code // Automatic Compilation // Global Accessibility // Namespace Consideration // Dynamic Compilation.

---

3. Differentiate between ViewState and Session State.

Ans:

ViewState and Session State are both state management techniques used in ASP.NET to persist data across multiple requests, but they serve different purposes and have distinct characteristics:

#### **Syntax For ViewState and SessionState:**

```
csharp Copy code
// Storing data in ViewState
ViewState["Key"] = "Value";

// Retrieving data from ViewState
string value = ViewState["Key"].ToString();
```

```
csharp Copy code
// Storing data in Session State
Session["Key"] = "Value";

// Retrieving data from Session State
string value = Session["Key"].ToString();
```

ViewState	SessionState
Maintained at page level only.	Maintained at session level.
View state can only be visible from a single page and not multiple pages.	Session state value availability is across all pages available in a user session.
It will retain values in the event of a postback operation occurring.	In session state, user data remains in the server. Data is available to user until the browser is closed or there is session expiration.
Information is stored on the client's end only.	Information is stored on the server.
used to allow the persistence of page-instance-specific data.	used for the persistence of user-specific data on the server's end.
ViewState values are lost/cleared when new page is loaded.	SessionState can be cleared by programmer or user or in case of timeouts.

---

#### 4. What is the purpose of the Globalization and Localization in ASP.NET?

Ans: (Definition)

Globalization is the process of designing the application in such a way that it can be used by users from across the globe (multiple cultures).

Localization, is the process of customize our application behave as per the current culture and locale.

##### Globalization:

- Globalization refers to the process of designing and developing software applications that can be adapted to various cultures and regions without code modification.
- In ASP.NET, globalization involves making applications culture-aware and language-independent, enabling them to display content in multiple languages and adapt to different cultural preferences.

##### Localization:

- Localization involves customizing an application's user interface (UI) and content to suit a specific culture, language, or region.
- In ASP.NET, localization typically involves creating resource files that contain localized versions of UI strings, messages, labels, and other text elements.

---

## 5. How does caching work in ASP.NET, and what are the different types of caching?

Ans:

Caching in ASP.NET involves storing frequently accessed data in memory to improve performance by reducing the need to recompute or retrieve the data from its original source repeatedly.

Simple Definition : "A cache simply stores the output generated by a page in the memory and this saved output (cache) will serve us (users) in the future.

Types of Cache in Asp.Net :

### Output Caching:

- Output caching stores the generated HTML output of a page or user control so that subsequent requests for the same content can be served directly from the cache without re-executing the page or control.

### Data Caching:

- Data caching allows you to cache application data such as database query results, objects, or any other data that is expensive to create or retrieve.

### Fragment Caching:

- Fragment caching allows you to cache portions of a page or user control, rather than caching the entire page.

### SQL Cache Dependency:

- SQL cache dependency allows you to cache data retrieved from a database and automatically invalidate the cache when the underlying data in the database changes.

### Custom Caching:

- ASP.NET allows you to implement custom caching mechanisms using the **Cache** object, which provides programmatic access to the ASP.NET caching system.

Syntax for Cache :

## Page Caching

```
01. <%@ OutputCache Duration = 5 VaryByParam = "*" VaryByCustom = "Browser" %>
```

## Data Caching

```
01. Cache["Website"] = "CSharpCorner";
```

Now, for inserting the cache into the objects, the insert method of the Cache class can be used. This insert method is used as follows:

```
01. Cache.Insert("Website", strName,  
02. new CacheDependency(Sever.MapPath("Website.txt")));
```

---

6. Explain the difference between Server.Transfer and Response.Redirect.

Ans:

Both `Server.Transfer` and `Response.Redirect` are mechanisms used in ASP.NET to navigate users to another page.

### **Server.Transfer:**

- `Server.Transfer` is a server-side method that transfers control from one ASP.NET page to another on the server.

### **Response.Redirect:**

- `Response.Redirect` is a client-side redirection method that sends a redirect response to the client's browser, instructing it to navigate to a new URL.

Syntax :

```
Server.Transfer("NewPage.aspx");
```

```
Response.Redirect("NewPage.aspx");
```

Now, Basic Difference Between Server.Transfer and Response.Redirect is,

- `Server.Transfer` occurs on the server-side and does not change the client's browser URL, while `Response.Redirect` occurs on the client-side and changes the browser URL.

- **Server.Transfer** is suitable for transferring control within the same application, while **Response.Redirect** is suitable for redirecting users to another page, possibly on a different server or domain.
- 

7. What is the purpose of the <compilation> section in the Web.config file?

Ans:

The **<compilation>** section in the Web.config file of an ASP.NET application is used to configure settings related to the compilation of ASP.NET code. It allows developers to specify various compilation options and settings that affect how ASP.NET code is compiled and executed.

Here are some of the purposes and configuration options available in the **<compilation>** section:

- **Debugging and Error Handling:** `debug="true" / debug="true"`
  - **Target Framework Version:** The `targetFramework` attribute allows developers to specify the version of the .NET Framework
  - **Code Optimization:** `<Compilers>`
  - **Batch Compilation:**
  - **Language Support:** `<assemblies>`
  - **Code Access Security:** `<trust>`
-

8. How can you handle errors and exceptions in an ASP.NET application?

Ans:

In ASP.NET applications, error and exception handling is crucial for providing a smooth and user-friendly experience, as well as for logging and troubleshooting issues.

---

#### Try-Catch Blocks:


- Use try-catch blocks to catch exceptions in code where you anticipate errors might occur.
- Wrap code that may throw exceptions inside a try block and catch specific types of exceptions in catch blocks.

#### Global Exception Handling:

- Implement a global exception handler to catch unhandled exceptions that occur during the processing of requests.
- In ASP.NET, you can use the **Application\_Error** event in the **Global.asax** file to handle unhandled exceptions.

Syntax :

csharp


 Copy code

```
protected void Application_Error(object sender, EventArgs e)
{
    // Log the error
    Exception exception = Server.GetLastError();
    // Display a user-friendly error page
    Server.ClearError();
}
```

### Custom Error Pages:

- Configure custom error pages in the Web.config file to provide a better user experience when errors occur.

xml

 Copy code

```
<customErrors mode="On" defaultRedirect="Error.aspx">
  <error statusCode="404" redirect="NotFound.aspx" />
</customErrors>
```

---

### Logging:

- Implement logging mechanisms to record details about exceptions, including the exception message, stack trace, and context information.

---

## 9. What is State Management in Asp.net?

Ans:

In an ASP NET application, state management in ASP NET is an object and preserves type state control.

In ASP NET, the information of users is stored and maintained till the user session ends.

### Types of State Management Techniques in ASP.NET :

There are two types of State management in ASP net. They are :

- Server-side
- Client-side



These are further subdivided into the following –

### **Server-Side**

- Session
- Application
- Cache

### **Client-Side**

- Cookies
  - Viewstate
  - Control state
  - Query String
  - Hidden Field
-

## Security in ASP.NET:

1. What is Cross-Site Scripting (XSS), and how can you prevent it in ASP.NET applications?

Ans:

- Cross-Site Scripting (XSS) is a type of security vulnerability commonly found in web applications. It occurs when a web application allows users to inject malicious scripts into web pages viewed by other users.
  - These scripts are usually written in JavaScript and can be executed in the context of other users' sessions, leading to various malicious actions such as stealing cookies, session tokens, or other sensitive information.
  - In ASP.NET applications, we can prevent Cross-Site Scripting attacks by following these best practices:
    1. **Input Validation**: Validate and sanitize all user inputs before accepting them
    2. **Output Encoding**: Encode all output that originates from user input, including data retrieved from databases, files, or other sources.
    3. **Use Anti-XSS Libraries**: Utilize Anti-XSS libraries provided by Microsoft or other reputable sources.
    4. **HTTPOnly Cookies**: Set the HTTPOnly flag on cookies to prevent client-side scripts from accessing them.
    5. **Regular Security Updates**: Keep your ASP.NET framework, libraries, and dependencies up to date to ensure that you have the latest security patches and fixes for known vulnerabilities.
    6. **Security Code Reviews**: Regularly conduct security code reviews and audits
-

## 2. Explain the role of Anti-Forgery Tokens in ASP.NET MVC.

Ans:

- Anti-Forgery Tokens play a crucial role in preventing Cross-Site Request Forgery (CSRF) attacks in ASP.NET MVC applications.
- CSRF attacks occur when an attacker tricks a user into unknowingly performing actions on a web application
- Here's how Anti-Forgery Tokens work and their role in ASP.NET MVC:
- **Token Generation**: When a user requests a form from an ASP.NET MVC application, the server generates a unique anti-forgery token (also known as a CSRF token) and includes it in the form as a hidden field.
- **Token Validation**: When the user submits the form, the ASP.NET MVC framework verifies that the anti-forgery token submitted with the form matches the one generated for the user's session.
- **Protection against CSRF Attacks**: By including an anti-forgery token in forms, ASP.NET MVC applications can protect against CSRF attacks.
- **AntiForgeryToken Attribute**: In ASP.NET MVC, developers can easily incorporate anti-forgery tokens into their forms by using the **@Html.AntiForgeryToken()** helper method within the form. Additionally, developers can apply the **[ValidateAntiForgeryToken]** attribute to controller actions that require protection against CSRF attacks.
- **Secure by Default**: ASP.NET MVC automatically includes anti-forgery tokens in forms generated by HTML helper methods such as **Html.BeginForm()** and **Ajax.BeginForm()**.

## Syntax For Using this token :

### In MVC:

```
namespace BankingApplication.Controllers
{
    public class HomeController : Controller
    {
        [HttpGet]
        public IActionResult Index()
        {
            return View();
        }

        [HttpPost]
        [ValidateAntiForgeryToken]
        public string Index(string AccountNumber, string Pin)
        {
            // Process the data
            // ...

            return $"AccountNumber: {AccountNumber} Pin Changed to: {Pin}";
        }
    }
}
```

### IN HTML FORM:

```
@{
    ViewData["Title"] = "Home Page";
}

<div class="text-center">
    <h1>Pin Change</h1>
    @using (Html.BeginForm("Index", "Home", FormMethod.Post))
    {
        @Html.AntiForgeryToken()

        <input type="text" name="AccountNumber" id="AccountNumber" />
        <input type="text" name="Pin" id="Pin" />
        <input type="submit" value="Change Pin" />
    }
</div>
```

---

3. How does ASP.NET protect against Cross-Site Request Forgery (CSRF) attacks?

Ans:

ASP.NET provides several mechanisms to protect against Cross-Site Request Forgery (CSRF) attacks, including Anti-Forgery Tokens, also known as CSRF tokens.

Here's how ASP.NET protects against CSRF attacks:

- **Anti-Forgery Tokens (CSRF Tokens)**
  - **AntiForgeryToken Attribute**
  - **Secure by Default**
  - **SameSite Cookies**
  - **AntiForgeryToken Verification**
- 

4. What is SQL injection, and how can it be prevented in ASP.NET applications?

Ans:

- SQL injection is a type of security vulnerability that occurs when malicious SQL code is inserted into input fields or parameters used by an application to interact with a database.
- SQL injection attacks can result in unauthorized access to sensitive data, data manipulation, or even data deletion.
- A Successful SQL-Injection attack can lead to:
  - 1) Access/Steal Sensitive information
  - 2) change or delete data from the database
- In ASP.NET applications, SQL injection vulnerabilities can be prevented by following these best practices:
  - I. **Parameterized Queries:** Instead of concatenating user input directly into SQL queries, developers should use parameterized queries or prepared statements.

- II. **Stored Procedures**: Utilize stored procedures whenever possible to encapsulate SQL logic within the database.
- III. **Input Validation and Sanitization**: Validate and sanitize all user inputs before using them in SQL queries.
- IV. **ORMs (Object-Relational Mapping)**: Consider using Object-Relational Mapping frameworks such as Entity Framework or Dapper, which provide built-in protection against SQL injection
- V. **Input and Output Encoding**: Encode user input and output data appropriately to prevent malicious script injection attacks such as Cross-Site Scripting (XSS).

A hacker might get access to user names and passwords in a database by simply inserting " OR ""=" into the use name or password text box:

User Name:

Password:

The code at the server will create a valid SQL statement like this:

```
Result
SELECT * FROM Users WHERE Name ="" or ""="" AND Pass ="" or ""=""
```

---

5. Describe the role of the authentication cookie in ASP.NET.

Ans:

- In ASP.NET, the authentication cookie plays a vital role in managing user authentication and maintaining session state.
- Here's a breakdown of its role and functionality:

- a) **User Authentication**: When a user successfully logs in to an ASP.NET application, the server creates an authentication cookie and sends it to the client's browser.
- b) **Session Persistence**: The authentication cookie persists on the client's browser across subsequent requests.
- c) **Authentication Ticket**: The authentication cookie typically contains an encrypted authentication ticket that includes information such as the user's identity, roles, expiration time, and any additional custom data relevant to the user's session.
- d) **Secure Communication**: Authentication cookies can be configured to use secure transmission mechanisms such as HTTPS to encrypt data transmitted between the client and the server.
- e) **Authentication Middleware**: In ASP.NET, authentication middleware components, such as the Forms Authentication middleware or ASP.NET Core Identity, handle the creation, validation, and management of authentication cookies. (Done in .NET CORE IDENTITY)

**Implement/set the cookie before user LoggedIN:**

```
services.AddControllersWithViews();  
services.AddAuthentication(CookieAuthenticationDefaults.AuthenticationScheme)  
    .AddCookie(x=>x.LoginPath="/account/login");
```

**Now add two middlewares in program.cs file:**

```
app.UseAuthentication();  
app.UseAuthorization();
```

Now add Data annotation to the action to prevent user from direct access to the action:

```
public class HomeController : Controller
{
    References
    public IActionResult Index()
    {
        return View();
    }
    [Authorize]
    References
    public IActionResult ConfidentialData()
    {
        return View();
    }
}
```

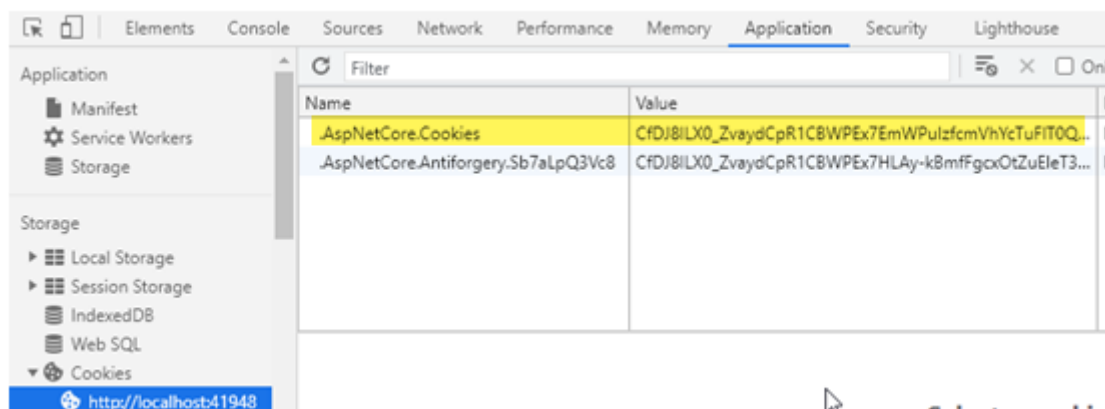
Now get the username through cookie on Home Page:

```
@{
    ViewData["Title"] = "Confidential Data";
    Layout = "~/Views/Shared/_Layout.cshtml";
}

<h2>Confidential Data</h2>

@if (User.Identity.IsAuthenticated)
{
    <table class="table table-bordered">
        @foreach (var claim in User.Claims) {
            <tr><td>@claim.Type</td><td>@claim.Value</td></tr>
        }
    </table>
}
```

Now Check in browser Network tab whether the cookie has been set or not:





## 6. How can you implement role-based authentication in ASP.NET?

Ans:

- Role-based authentication in ASP.NET can be implemented using the ASP.NET Identity framework.
- ASP.NET Identity allows you to manage user authentication, including roles, claims, and more.

- Here's how you can implement role-based authentication in ASP.NET:

1. **Create an ASP.NET Web Application:** Start by creating a new ASP.NET web application project in Visual Studio.
2. **Set up ASP.NET Identity:** ASP.NET Identity is typically included by default in new ASP.NET projects.
3. **Define Roles:** Define the roles that you want to use in your application. Roles represent different levels of access or permissions that users can have.

**Define Roles:** Define the roles that you want to use in your application. Roles represent groups of users with similar permissions or access levels.

```
public static class ApplicationRoles
{
    public const string Admin = "Admin";
    public const string User = "User";
}
```

**Authorize Access:** Use the **[Authorize]** attribute to restrict access to specific controllers or actions based on roles.

```
// Example of restricting access based on roles
[Authorize(Roles = "Admin")]
public class AdminController : Controller
{
    // Actions for admin-only access
}
```

**Check Roles in Code:** We can also check the roles of the current user programmatically within our application code.

```
// Example of checking roles in code
if (User.IsInRole(ApplicationRoles.Admin))
{
    // Perform admin-specific actions
}
```

## ASP.NET MVC

1) What is the Model-View-Controller (MVC) design pattern, and how does it apply to ASP.NET MVC?

Ans:

The **ASP.NET MVC** is an open-source **web application development framework** from Microsoft that is based on the **MVC (Model-View-Controller)** architectural design pattern which provides a clean separation of code.

### **Model in ASP.NET MVC:**

1. In MVC, the models are basically C#.NET or VB.NET classes.
2. The Models are basically used to manage the business data and business logic.
3. This is the component that can be accessed by both the controller and view.
4. The model component can be used by a controller to pass the data to a view.
5. It can also be used by a view, in order to display the data the in page (HTML output)

### **View in ASP.NET MVC:**

1. In the ASP.NET MVC application, the views are nothing but the cshtml or vbhtml pages without having a code-behind file.
2. All page specific HTML generation and formatting can be done inside view.
3. A request to view can only be made from a controller's action method.
4. The view is only responsible for displaying the data.
5. By default, views are stored in the Views folder of an ASP.NET MVC application.

### **controller in ASP.NET MVC:**

1. In ASP.NET MVC, the controller is basically a C# or VB.NET class that inherits from **System.Web.Mvc.Controller**.
2. This is the component that has access to both the Models and Views in order to control the flow of the application execution.
3. The Controller class contains action methods that are going to respond to the incoming URL.
4. A controller can access and use the model classes to pass the data to a view.
5. By default, controllers have stored in the Controllers folder an ASP.NET MVC application.

---

2) Explain the Role of Action Result?

Ans:

- The ActionResult class is the base class for all action results. An action result can be of type ViewResult, JsonResult, RedirectResult, **ContentResult**, **JavaScriptResult**, **FileResult**, **EmptyResult**, **HttpNotFoundResult**, **HttpUnauthorizedResult**, **HttpStatusCodeResult** and so on.
- An action method is responsible for processing a client request and returning an appropriate response.
- The Action Result encapsulates the data that needs to be sent back to the client in response to the request.

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

---

3) What is the difference between three layer architecture and MVC architecture?

Ans:

- The 3-layer architecture separates the application into 3 components which consist of the Presentation Layer, Business Layer, and Data Access Layer. In 3-layer architecture, the user is going to interact with the Presentation layer only. 3-layer is a linear architecture.
  - The MVC architecture separates the application into three major components such as Model, View, and Controller. In MVC architecture, the user is going to interact with the controller with the help of a view. MVC is a triangle architecture.
  - MVC does not replace 3-layer architecture. Typically MVC and 3-layer architecture are used together and the MVC Design Pattern acts as the Presentation layer of the application.
- 

4) what are action filters how are they used in ASP.NET MVC?

Ans:

- Action filter executes before and after an action method executes. Action filter attributes can be applied to an individual action method or to a controller. When an action filter is applied to a controller, it will be applied to all the controller's action methods.
- The `OutputCache` is a built-in action filter attribute that can be applied to an action method for which we want to cache the output. For example, the output of the following action method will be cached for 100 seconds.

```
[OutputCache(Duration=100)]  
public ActionResult Index()  
{  
    return View();  
}
```

---

- You can create a custom action filter in two ways, first, by implementing the `IActionFilter` interface and the `FilterAttribute` class. Second, by deriving the `ActionFilterAttribute` abstract class.

The **IActionFilter** interface include following methods to implement:

- `void OnActionExecuted(ActionExecutedContext filterContext)`
- `void OnActionExecuting(ActionExecutingContext filterContext)`

The **ActionFilterAttribute** abstract class includes the following methods to override:

- `void OnActionExecuted(ActionExecutedContext filterContext)`
- `void OnActionExecuting(ActionExecutingContext filterContext)`
- `void OnResultExecuted(ResultExecutedContext filterContext)`
- `void OnResultExecuting(ResultExecutingContext filterContext)`

```
public class LogAttribute : ActionFilterAttribute
{
    public override void OnActionExecuted(ActionExecutedContext filterContext)
    {
        Log("OnActionExecuted", filterContext.RouteData);
    }

    public override void OnActionExecuting(ActionExecutingContext filterContext)
    {
        Log("OnActionExecuting", filterContext.RouteData);
    }
}
```

---

5) Difference between TempData, ViewBag, and ViewData in Asp.Net MVC?

Ans:

ViewData, ViewBag, and TempData to pass data from the controller to view and in the next request.

#### **ViewData in ASP.NET MVC**

1. ViewData is a dictionary object that is derived from ViewDataDictionary class.
2. The ViewData is used to pass data from the controller to the corresponding view.
3. Its life lies only during the current request.
4. If redirection occurs then its value becomes null.
5. It's required typecasting for getting data and check for null values to avoid the error.

#### **ViewBag in ASP.NET MVC**

1. ViewBag is a dynamic property that takes advantage of the new dynamic features in C# 4.0.
2. Basically, it is a wrapper around the ViewData and also used to pass data from the controller to the corresponding view.
3. Its life also lies only during the current request.
4. If redirection occurs then its value becomes null.
5. It doesn't require typecasting for getting data.

#### **TempData in ASP.NET MVC**

1. TempData is a dictionary object that is derived from the TempDataDictionary class and stored in a short life session.
  2. TempData is used to pass data from current request to subsequent request (means redirecting from one page to another).
  3. Its life is very short and lies only until the target view is fully loaded.
  4. It's required typecasting for getting data and check for null values to avoid the error.
  5. It's used to store only one time messages like error messages, validation messages.
-

## 6) Purpose of Routing in ASP.NET MVC?

Ans:

In MVC, routing is a process of mapping the browser request to the controller action and return response back. Each MVC application has default routing for the default **HomeController**. We can set custom routing for newly created controller.

In ASP.NET MVC we have Two Types of Routing:

- 1) Conventional Based Routing
- 2) Attribute Based Routing

Conventional Based Routing is the default Routing comes in Program.cs

```
app.MapControllerRoute(
    name: "default",
    //pattern: "{controller=Employee}/{action=Index}/{id?}");
    pattern: "{controller=Ajax}/{action=Index}/{id?}");

app.Run();
```

Attribute Based Routing is the manual task can be implemented with the help of asp-controller, asp-action

```
<td><a asp-controller="Emp" asp-action="DeleteEmp1" asp-route-id="@data.Id" class="btn btn-sm btn-danger">Delete</a>
<a asp-controller="Emp" asp-action="EditEmp" asp-route-id="@data.Id" class="btn btn-sm btn-success">Edit</a>
```

---

## 7) Concept of Dependency Injection in ASP.NET MVC?

Ans:

- Dependency Injection (DI) is a design pattern widely used in ASP.NET MVC (Model-View-Controller) and other modern software development frameworks. The concept of Dependency Injection revolves around the principle of inverting the control of object creation and managing dependencies between objects.
- **Dependency injection** is one form of the broader (more general) technique of inversion of control.

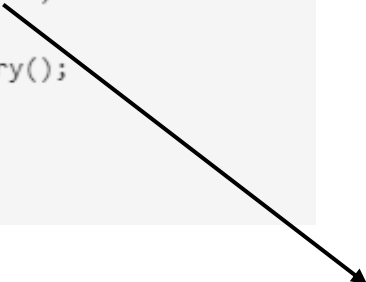
**Important Terms :**

- Controllers should have mostly action methods and no business logic



- For writing business logic we have to create Interface which contains (abstract method) for the implementation of that abstract method we want service class which provides the definition for that abstract method and we will call that method from controller.

```
public class UserController : Controller
{
    public ActionResult Create(User user)
    {
        var repository = new Repository();
        repository.Save(user);
    }
}
```



User is the modal class and we are injecting all the dependencies inside user references although we are saving result inside using save method.

2) Below are the use of method means how we are performing crud using DI(Dependency Injection).

```
public class UserController : Controller
{
    public ActionResult Create(User user)
    {
        var repository = new Repository();
        repository.Save(user);
    }

    public ActionResult Edit(User user)
    {
        var repository = new Repository();
        repository.Update(user);
    }
}
```

```
public class UserController : Controller
{
    private readonly Repository _repository;

    public UserController(Repository repository)
    {
        _repository = repository;
    }

    public ActionResult Create(User user)
    {
        _repository.Save(user);
    }

    public ActionResult Edit(User user)
    {
        _repository.Update(user);
    }
}
```

## Q.1 What is Web.Config file?

It is the main settings and configuration file for an ASP.NET web application. The file is an XML document that defines configuration information regarding the web application. The web.config file contains information that controls module loading, security configuration, session configuration, and application language and compilation settings.

Configuration file is nested in a root <configuration> element

Configuration element contains a <system.web> element, which is used for ASP.NET settings

Inside the <system.web> element are separate elements for each aspect of configuration.

<appSettings> element can store custom settings

A typical web.config file looks like this;

```
<?xml version="1.0"?>
  <configuration>
    <appSettings />
    <connectionStrings />
    <system.web>
      <!-- ASP.NET configuration sections go here. -->
    </system.web>
  </configuration>
```

We define our connection string in web.config file itself,

```
<configuration>
  <connectionStrings>
    <add name="myConnectionString"
      connectionString=
        "Data Source=localhost;Integrated Security=SSPI;Initial Catalog=Airtel;"
      providerName="System.Data.SqlClient" />
  </connectionStrings>
</configuration>
```

## Q.2 What is Route Config?

In MVC, routing is a process of mapping the browser request to the controller action and return response back. Each MVC application has default routing for

the default **HomeController**. We can set custom routing for newly created controller.

The **RouteConfig.cs** file is used to set routing for the application. Initially it contains the following code.

```
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.Routing;
7. namespace MvcApplicationDemo
8. {
9.     public class RouteConfig
10.    {
11.        public static void RegisterRoutes(RouteCollection routes)
12.        {
13.            routes.IgnoreRoute("{resource}.axd/{*pathInfo}");
14.            routes.MapRoute(
15.                name: "Default",
16.                url: "{controller}/{action}/{id}",
17.                defaults: new { controller = "Home", action = "Index", id = UrlPa
                    rameter.Optional }
18.            );
19.        }
20.    }
21.}
```

### Q.3 Types of routing in MVC?

There are two types of routing in MVC:

- 1) Convention Based Routing
- 2) Attribute Routing

Convention Based Routing:

It creates routes based on a series of conventions that represents all the possible routes in your system. Convention-based is defined in the Program.cs file.

Below is the Syntax to define Convention Based Routing.

```
app.MapControllerRoute(  
    name: "default",  
    pattern: "{controller=Home}/{action=Index}/{id?}");
```

### Attribute Routing:

It creates routes based on attributes placed on the controller or action level. Attribute routing provides us more control over the URL generation patterns which helps us in SEO.

We use [Route()] data annotation to give attribute routing to particular controller or action.

```
public class HomeController : Controller  
{  
    [Route("")]  
    [Route("Home")]  
    [Route("Home/Index")]  
    public string Index()  
    {  
        return "Index() Action Method of HomeController";  
    }  
    [Route("Home/Details/{id}")]  
    public string Details(int id)  
    {  
        return "Details() Action Method of HomeController, ID Value = " + id;  
    }  
}
```

### Mixed Routing:

You can use Convention-based Routing and Attribute routing together. Even you should use both together since it's not possible to define attribute route for each and every action or controller. In that case, Convention-based Routing will help you.

Note: If both Routings are present, then .NET Core overrides conventional routes. This is because .NET Core gives preference to attribute routes.

### Route Constraints:

Route Constraints are used to restrict the type of passed value to an action. For example, if you expect an argument id as an integer type, then you have to restrict it to an integer type by using datatype in the curly brackets as {id:int}.

```
app.UseMvc(routes =>
{
    routes.MapRoute(
        template: "{controller}/{action}/{id:int?}");
});
```

### Q.4 What is Attribute Routing?

Attribute routing allows us to define routes directly on our controller and action methods using the **Route** Attributes. With the help of ASP.NET Core Attribute Routing, we can use the Route attribute to define routes for our application. We can use the Route attribute either at the Controller or Action Methods levels. Applying the Route attribute at the Controller level applies to all the controller's action methods.

we can apply Multiple Route Attributes to a Single Action Method in the ASP.NET Core MVC Application. Let us understand this with an example.

```
public class HomeController : Controller
{
    [Route("")]
    [Route("Home")]
```

```
[Route("Home/Index")]
public string Index()
{
    return "Index() Action Method of HomeController";
}
}
```

If you notice, here we have applied the Route() attribute 3 times on the Index() action method of the Home Controller class. Here, we specified a different route template with each instance of the Route attribute. With the above three Route attributes applied to the Index action method, we can now access the Index() action method of the HomeController using the following 3 URLs.

**<https://localhost:44359/>**

**<https://localhost:44359/home>**

**<https://localhost:44359/home/index>**

If you apply the same Route Attributes multiple times with an Action Method and if you apply the same Route Template to different action methods, then you will get **AmbiguousMatchException**.

With Attribute Routing in ASP.NET Core MVC Application, the Controller name and Action Method names do not play any role.

### **Attribute Routing with Parameters in ASP.NET Core MVC Application:**

As we already discussed, conventional-based routing can specify the route parameters as part of the route template. We can also do the same with Attribute Routing in ASP.NET Core. That means we can also define Route Attributes with Route Parameters. To understand this, modify the Home Controller as shown below.

```
[Route("Home/Details/{id}")]
public string Details(int id)
{
    return "Details() Action Method of HomeController, ID Value = " + id;
}
```

### **Attribute Routing with Optional Parameters in ASP.NET Core MVC Application:**

We can also make the Route Parameter optional in Attribute Routing like conventional-based routing. You can define a Route Parameter as optional by adding a **question mark (?)** to the route parameter. You can also specify the default value by using the **parameter = value**. If this is not clear at the moment, then don't worry; we will see both of these approaches with examples.

```
[Route("Home/Details/{id?}")]
public string Details(int id)
{
    return "Details() Action Method of HomeController, ID Value = " + id;
}
```

```
[Route("Home/Details/{id=10}")]
public string Details(int id)
{
    return "Details() Action Method of HomeController, ID Value = " + id;
}
```

Q.5 Difference between ViewBag and ViewData?

ViewBag:

- 1) ViewBag uses dynamic feature that was introduced in c# 4.0 it allows an object to have properties dynamically added to it
- 2) ViewBag doesn't require type casting,
- 3) ViewBag is used to pass data from controller to view.
- 4) View bag provides short life it means value becomes null when redirection occurs.
- 5) ViewBag doesn't provide compile time error checking
- 6) syntax: ViewBag.PropertyName=value;

ViewData:

- 1) ViewData is a dictionary of objects and it is accessible by using string as a key.
- 2) ViewData require type casting to avoid errors.



- 3) ViewData is used to pass data from controller to view.
- 4) ViewData provides short life it means value becomes null when redirection occurs.
- 5) ViewData doesn't provide compile time error checking
- 6) syntax: `ViewData["key"]=value;`

Q.6 Difference between TempData and Session.

#### TempData

- 1) TempData allow us to persisting data for the duration of single subsequent request.
- 2) ASP.net MVC will automatically expire the value of TempData once consecutive request returned the result (it means, it alive only till the target view is fully loaded).
- 3) It valid for only current and subsequent request only.
- 4) TempData has Keep method to retention the value of TempData.  
`TempData.Keep();`  
`TempData.Keep("msg");`
- 5) TempData internally stored the value in to Session variable.
- 6) It is used to stored only one time messages like validation messages, error messages etc.

#### Session

- 1) Session is able to store data much more long time, until user session is not expire.
- 2) Session will be expire after the session time out occurred
- 3) It is valid for all requests.
- 4) Session variable are stored in SessionStateItemCollection object (Which is exposed through the `HttpContext.Session` property of page).
- 5) It is used to stored long life data like user id, role id etc. which required throughout user session.

Q.7 what is Html helper?

[HTML](#) Helpers are managed within View to execute HTML content. We can use HTML Helpers to implement a method that returns a string.

This tutorial discusses how you can define or create custom HTML Helpers that you can use within your [MVC](#) views.

The [ASP.NET MVC](#) framework incorporates the below set of standard HTML Helpers.

- @Html.TextBox
- @Html.Password
- @Html.TextArea
- @Html.CheckBox
- @Html.RadioButton
- @Html.DropDownList
- @Html.ListBox
- @Html.Hidden
- @Html.Display
- @Html.Editor
- @Html.ActionLink
- @Html.BeginForm
- @Html.Label

### Q.8 extension of views?

.cshtml is file extension of views file in asp.net mvc. It means we can have both HTML and C sharp code inside this particular view file.

### Q.9 why we use Model?

A model is a class that contains the business logic of the application. It is also used for accessing data from the database. The model class does not handle directly input from the browser. It does not contain any HTML code as well.

Models are also refers as objects that are used to implement conceptual logic for the application. A controller interacts with the model, access the data, perform the logic and pass that data to the view.

### Q.10 Types of Models

In general, there are three distinct types of a model. With regard to ASP.NET MVC, we can call them domain model, view model and input model. Let's find out more.

The **domain model** describes the data you work with in the middle tier of the application. The domain model is expected to provide a faithful representation of the entities that populate the business domain. These entities are typically persisted by the data-access layer and consumed by services that implement business processes. The domain model is also often referred to as the entity model or simply as the data model. The domain model pushes a vision of data that is, in general, distinct from the vision of data you find in the presentation layer. (Note that the domain model "may" be the same model you work with in the presentation layer, but this should be taken as the exception rather than the rule.)

The **view model** describes the data being worked on in the presentation layer. Any data you present in the view (whether strings, collections, dates) finds its place in one of the properties of the view model classes. Any view model class represents the data that the controller transmits after processing operations for serving a response to the users. The view model results from the aggregation of multiple classes, one per view or page that you display.

Finally, the **input model** is the collection of classes that describe the input data flow that goes from the web page down to the controllers and backend. The input model faithfully represents the data being uploaded with individual HTTP requests. Usually, you manage this data as strings and read (serialized) values from collections such as **QueryString** and **Form**. Turning strings into strong types is entirely your responsibility in ASP.NET Web Forms. In ASP.NET MVC, the model-binding infrastructure does most of these chores for you and attempts to map incoming strings to matching properties of input model types. Let's find out about the details.

Q.11 What is INPUT model?

The **input model** is the collection of classes that describe the input data flow that goes from the web page down to the controllers and backend. The input model faithfully represents the data being uploaded with individual HTTP requests. Usually, you manage this data as strings and read (serialized) values from collections such as **QueryString** and **Form**. Turning strings into strong types is entirely your responsibility in ASP.NET Web Forms. In ASP.NET MVC, the model-binding infrastructure does most of these chores for you and attempts to map incoming strings to matching properties of input model types.

Q.12 What is the concept of filter in Asp. Net MVC?

Filters are used to execute custom logic before or after executing the action method. ASP.NET MVC provides filters for this purpose. ASP.NET MVC Filter is a custom class where we can write custom logic to execute that before or after an action method is executed.

### **Types of filters in MVC**

#### **Action Filter**

Action filters are used to implement logic that gets executed before and after a controller action executes. It implements the `IActionFilter` attribute.

## **Authentication Filter**

Authentication filter is introduced with ASP.NET MVC5. It authenticates the credentials of a user like a username and password. `IAuthenticationFilter` interface is used to create CustomAuthentication filter.

## **Authorization Filter**

Authorization filters are used to implement authentication and authorization for controller actions. It implements the `IAuthorizationFilter` attribute.

## **Result Filter**

Result filters contain logic that is executed before and after a View result is executed. For example, you might want to modify a View result right before the View is rendered to the browser. It implements the `IResultFilter` attribute.

## **Exception Filter**

Exception filter is used to handle errors raised by either your controller actions or controller action results. You also can use exception filters to log errors. It implements the `IExceptionFilter` attribute.

Q.13 Difference between asp.net web forms and asp.net MVC?

Asp.net web forms:

- 1) Asp.Net Web Form follows a traditional event-driven development model.
- 2) Asp.Net Web Form has server controls.
- 3) Asp.Net Web Form supports view state for state management at the client side.

- 4) Asp.Net Web Form has file-based URLs means the file name exist in the URLs must have its physical existence.
- 5) Asp.Net Web Form follows Web Forms Syntax
- 6) In Asp.Net Web Form, Web Forms(ASPX) i.e. views are tightly coupled to Code behind(ASPX.CS) i.e. logic.
- 7) Asp.Net Web Form has Master Pages for a consistent look and feel.
- 8) Asp.Net Web Form has User Controls for code re-usability.
- 9) Asp.Net Web Form has built-in data controls and is best for rapid development with powerful data access.

#### Asp.net MVC:

- 1) Asp.Net MVC is lightweight and follows the MVC (Model, View, Controller) pattern-based development, model.
- 2) Asp.Net MVC has HTML helpers.
- 3) Asp.Net MVC does not support view state.
- 4) Asp.Net MVC has route-based URLs means URLs are divided into controllers and actions and it is based on the controller not on the physical file.
- 5) Asp.Net MVC follows customizable syntax (Razor as default)
- 6) In Asp.Net MVC, Views and logic are kept separately.
- 7) Asp.Net MVC has Layouts for a consistent look and feel.
- 8) Asp.Net MVC has Partial Views for code re-usability
- 9) Asp.Net MVC is lightweight, provides full control over markup, and supports many features that allow fast & agile development. Hence it is best for developing an interactive web application with the latest web standards.

Q.14 What is the role of the Global.asax file in an ASP.NET application?

The **Global.asax** file, also known as the ASP.NET application file, is a special file in ASP.NET applications that serves as the entry point for handling application-level events and customizing application behavior. It's essentially a global application file where you can write code to respond to various application and session-level events.

Here are some key aspects and uses of the **Global.asax** file:

1. **Application Events Handling:** It allows you to handle various application-level events such as **Application\_Start**, **Application\_End**, **Session\_Start**, **Session\_End**, etc. These events are fired by the ASP.NET runtime during the application lifecycle. For example, **Application\_Start** is fired when the application starts, and **Application\_End** is fired when the application shuts down.
2. **Session Events Handling:** In addition to application-level events, you can also handle session-level events such as **Session\_Start** and **Session\_End**. These events are triggered when a new session starts or ends, respectively.
3. **Global Exception Handling:** You can use the **Application\_Error** event to handle unhandled exceptions that occur during the application's execution. This allows you to log errors, display custom error pages, or perform other actions in response to errors.
4. **Application-Level Variables and Methods:** You can declare application-level variables and methods in the **Global.asax** file, which are accessible throughout the application. This can be useful for storing global configuration settings or implementing shared functionality.

5. **URL Routing:** With ASP.NET MVC or ASP.NET Core, you can define custom URL routing rules in the **Global.asax** file to map URLs to specific controller actions or pages.

Here's a simple example of a **Global.asax** file:

```
<%@ Application Language="C#" %>
<script runat="server">

    void Application_Start(object sender, EventArgs e)
    {
        // Code that runs on application startup
        // For example, initializing application-wide resources
    }

    void Application_End(object sender, EventArgs e)
    {
        // Code that runs on application shutdown
    }

    void Session_Start(object sender, EventArgs e)
    {
        // Code that runs when a new session starts
    }

    void Session_End(object sender, EventArgs e)
    {
        // Code that runs when a session ends
    }
}
```



```
void Application_Error(object sender, EventArgs e)
{
    // Code that runs when an unhandled error occurs
}
</script>
```

Q.15 How does ASP.NET handle session state?

As we all know, our web is "**Stateless**", in other words a new instance of a web page class, is recreated each time the page is posted to the server. HTTP is a stateless protocol and it can't hold the client information on the page. For example, if the user inserts some information on one page and then moves to the next page then that inserted data will be lost from the first page and moreover the user will not be able to retrieve that information.

So basically here we need someone to hold the state of our application. Here is the privilege role of our "**session state**". Basically a session is a variable used between the client and the server that is stored on the server side.

So a session helps to maintain the user state and data all over the application by storing the information on the server memory. Also a session can store any kind of information or object on the server side and is accessible in the entire website.

Steps to create session Variable:

Syntax:

```
Session["var_name"] = value;
```

We need type casting to retrieve our session variable

```
Session["var_name"].ToString();
```

Session Variable is accessible throughout our application.

#### Q.16 Describe the ASP.NET Page Life Cycle

The Page Life Cycle has certain phases that help in writing custom controls and initializing an application.

Following are the different phases of the Page Life Cycle:

##### 1. Page Request

Page Request is the first step of the page life cycle. When a user request is made, the server checks the request and compiles the pages. Once the compilation is done, the request is sent back to the user.

##### 2. Page Start

Page Start helps in creating two objects: request and response. The request holds all the information which the user sent, while the response contains all the information that is sent back to the user.

##### 3. Page Initialization

Page Initialization helps to set all the controls on the pages. It has a separate ID, and it applies themes to the pages in this step.

##### 4. Page Load

Page Load helps to load all the control properties of an application. It also helps to provide information using view state and control state.

##### 5. Validation

Validation happens when the execution of an application is successful. It returns two conditions: true and false. If execution is successful, it returns true, otherwise false.

## 6. Event Handling

Event Handling takes place when the same pages are loaded. It is a response for the validation. When the same page is loaded, a Postback event is called, which checks the page's credentials.

## 7. Rendering

Rendering happens before it sends all the information back to the user. And all this information is stored before being sent back.

## 8. Unload

Unload is a process that helps in deleting all unwanted information from the memory once the output is sent to the user.

Q.17 What is the purpose of the ViewState in ASP.NET?

- View state is a mechanism used in ASP.NET for preserving the state of server-side controls across postbacks.
- It is a hidden field on the page that contains the state of the page and its controls.
- View state is used to maintain the values of controls and other page state information between round trips to the server.

- It helps in preserving user input, control values, and other important data during postbacks without requiring additional server-side or client-side code.

Below is use case of ViewState:

```
<asp:TextBox runat="server" ID="txtUsername"></asp:TextBox>
```

```
<asp:Button runat="server" ID="btnSubmit" Text="Submit"  
OnClick="btnSubmit_Click"/>
```

```
protected void Page_Load(object sender, EventArgs e)
```

```
{  
    if (!IsPostBack)  
    {  
        // Set initial value in view state  
        ViewState["Counter"] = 0;  
    }  
}
```

```
protected void btnSubmit_Click(object sender, EventArgs e)
```

```
{  
    // Increment counter in view state  
    int counter = (int)ViewState["Counter"];  
    counter++;  
    ViewState["Counter"] = counter;  
  
    // Use the counter value as needed
```

```
Response.Write("Button clicked " + counter + " times.");  
}
```

Q.18 Explain the concept of Routing in asp.net mvc

In ASP.NET, routing is the process of directing the HTTP requests to the right controller. The MVC middleware must decide whether a request should go to the controller for processing or not. The middleware makes this decision based on the URL and some configuration information.

Q.19 What is difference the between authentication and authorization?

- 1) Authentication is the process of verifying the identity of a user or system, ensuring that the entity attempting to access a resource is who it claims to be. This typically involves providing credentials such as usernames, passwords, biometric data, or digital certificates.
- 2) Authorization, on the other hand, is the process of determining what actions an authenticated user or system is allowed to perform. It involves checking the permissions and privileges associated with the authenticated identity to determine whether they have the right to access a particular resource or perform a specific action.
- 3) summary, authentication verifies who you are, while authorization determines what you are allowed to do once your identity has been confirmed.
- 4) The key difference between authentication and authorization is that authentication confirms the identity of a user or system, while authorization controls what actions that authenticated entity is allowed to perform.

Q.20 How can you secure sensitive information, such as connection strings, in an ASP.NET application?

1. **Use Configuration Files:** Store sensitive information like connection strings in configuration files (**web.config** for web applications). ASP.NET provides configuration management features for securely storing and accessing such settings.
2. **Encrypt** your sensitive Information such as connection strings using base64 encoding methods or other similar methods.
3. **Connection String Encryption:** If you're using SQL Server, consider using Windows Authentication mode instead of storing plain text usernames and passwords in connection strings. This way, the connection is made using the identity of the application pool, which is more secure.
4. **Parameterized Queries:** When interacting with databases, use parameterized queries or stored procedures to prevent SQL injection attacks. Avoid constructing SQL queries by concatenating strings with user inputs.
5. **Secure Network Communication:** Ensure that sensitive data is transmitted securely over the network by using encryption protocols such as HTTPS for web applications and SSL/TLS for database connections.

By following these best practices, you can effectively secure sensitive information, including connection strings, in your ASP.NET C# applications and reduce the risk of unauthorized access or data breaches.

Q.21 What are the advantages of using Entity Framework for data access in ASP.NET?

Using Entity Framework (EF) for data access in ASP.NET offers several advantages:

1. **Increased Productivity:** EF enables developers to work with data in a more abstract and object-oriented way, reducing the amount of boilerplate code needed for data access. This leads to increased

productivity as developers can focus more on application logic rather than database interactions.

2. **Object-Relational Mapping (ORM):** EF provides a powerful ORM framework that allows developers to map database tables to .NET objects seamlessly. This simplifies data access by allowing developers to work with familiar object-oriented constructs rather than writing complex SQL queries.
3. **LINQ Support:** EF integrates with Language Integrated Query (LINQ), allowing developers to write expressive and type-safe queries directly in C# or VB.NET. This makes it easier to query and manipulate data without having to write raw SQL statements.
4. **Automatic Code Generation:** EF can automatically generate database schema and entity classes based on the underlying database structure. This saves developers time and effort by eliminating the need to manually create entity classes and mapping configurations.
5. **Change Tracking and State Management:** EF tracks changes made to entities and manages their state, making it easier to insert, update, and delete data. This helps ensure data consistency and integrity by automatically synchronizing changes with the database.
6. **Database Independence:** EF supports multiple database providers, allowing developers to work with different database systems (e.g., SQL Server, MySQL, SQLite) using a consistent API. This provides flexibility and allows applications to switch between database platforms without significant code changes.
7. **Query Optimization:** EF includes query optimization features such as query caching, lazy loading, and eager loading, which help improve performance by reducing the number of database round-trips and optimizing data retrieval.
8. **Integration with ASP.NET Features:** EF integrates seamlessly with other ASP.NET features such as data binding, model validation, and authentication. This simplifies development and enables developers to build robust, data-driven web applications more efficiently.

Q.22 How .Net core differs from traditional asp.net?

1. **Cross-platform Compatibility:** .NET Core is designed to be cross-platform, meaning it can run on Windows, Linux, and macOS. This is a departure from traditional ASP.NET, which primarily targeted Windows-based environments.
2. **Modular and Lightweight:** .NET Core is modular and lightweight compared to traditional ASP.NET. It allows developers to include only the necessary components in their applications, resulting in smaller deployment sizes and faster startup times.
3. **Open Source:** .NET Core is open source and developed in the open on GitHub. This allows for community contributions, transparency, and faster innovation compared to the closed development model of traditional ASP.NET.
4. **Performance:** .NET Core offers improved performance compared to traditional ASP.NET, especially in terms of throughput and scalability. It leverages new features such as the Kestrel web server and asynchronous programming models to achieve better performance.
5. **Cross-platform Development Tools:** .NET Core comes with cross-platform development tools, including the .NET CLI (Command Line Interface) and Visual Studio Code, which enable developers to build and debug applications on different operating systems.
6. **Support for Modern Development Practices:** .NET Core embraces modern development practices such as microservices architecture, containerization (e.g., Docker), and cloud-native development. It provides built-in support for Docker containers and Kubernetes orchestration, making it well-suited for cloud-based deployments.
7. **.NET Standard:** .NET Core introduces the concept of .NET Standard, which defines a common set of APIs that are available across all .NET implementations, including .NET Core, .NET Framework, and Xamarin. This allows developers to write code that can run on multiple platforms without modification.
8. **Support for Latest Language Features:** .NET Core supports the latest C# language features and runtime enhancements, enabling developers to take advantage of new language constructs, performance improvements, and productivity enhancements.



9. **Modular HTTP Pipeline:** .NET Core introduces a modular HTTP pipeline that allows developers to configure middleware components to handle HTTP requests and responses. This provides greater flexibility and control over the request processing pipeline compared to traditional ASP.NET.

Q.23 What are the main advantages of using ASP.NET Core over traditional ASP.NET?

ASP.NET Core is a high-performance, cross-platform, and open-source framework. It allows you to build modern, cloud-enabled, and Internet-connected apps.

Some of the significant benefits of the ASP.NET Core framework over the ASP.NET framework are:

- **High Performance:** ASP.NET Core framework is designed from scratch, keeping performance in mind. The ASP.NET team has focused on making the default web server, Kestrel, as fast as possible. TechEmpower, which has been running benchmarks on various frameworks, lists the ASP.NET Core with Kestrel as the fastest over 400 frameworks.
- **Cross-Platform:** ASP.NET Core runs on the cross-platform .NET 5.0 platform. It is not tied to a Windows operating system, like the legacy ASP.NET framework. You can develop and run production-ready ASP.NET Core apps on Linux or a Mac. If you decide to use Linux, you don't have to pay for Windows licenses, resulting in significant cost savings.
- **Open Source:** ASP.NET Core is open-source and actively developed on GitHub by thousands of developers all over the world, along with Microsoft. All the source code is hosted on GitHub for anyone to see, change and contribute back.

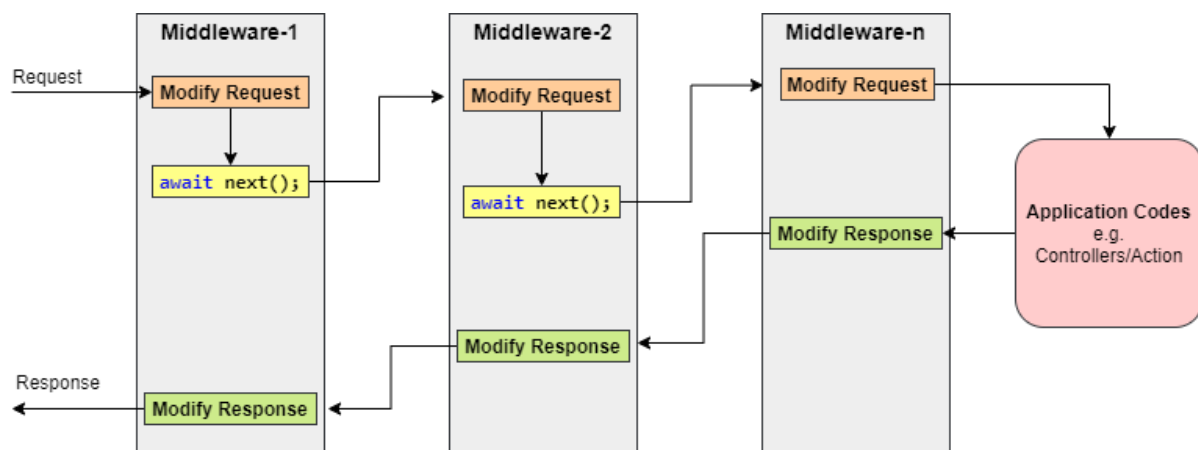
Q.24 Explain the concept of middleware in asp.net core?

Middleware is a piece of code in an application pipeline used to handle requests and responses.

For example, we may have a middleware component to authenticate a user, another piece of middleware to handle errors, and another middleware to serve static files such as JavaScript files, CSS files, images, etc.

Middleware can be built-in as part of the .NET Core framework, added via NuGet packages, or can be custom middleware. These middleware components are configured as part of the application startup class in the configure method. Configure methods set up a request processing pipeline for an ASP.NET Core application. It consists of a sequence of request delegates called one after the other.

The following figure illustrates how a request processes through middleware components.



Generally, each middleware may handle the incoming requests and pass execution to the next middleware for further processing.

We provide our middleware configurations in **Program.cs** File.

Q.25 How does Dependency Injection work in ASP.NET Core, and why is it beneficial?

Dependency injection (DI) is a software design pattern that allows us to separate the dependencies of a class from its implementation. This makes our code more loosely coupled and easier to test.

In .NET Core, dependency injection is implemented using the `IServiceProvider` interface. This interface provides a way to register services with the DI container and then inject those services into classes that need them.

To use dependency injection in .NET Core, we need to do the following:

1. Create an interface that defines the contract for our dependency.
2. Create a class that implements the interface.
3. Register the service with the DI container.
4. Inject the service into the class that needs it.

Q.26 What is the purpose of the wwwroot folder in an ASP.NET Core project?

By default, the **wwwroot** folder in the ASP.NET Core project is treated as a web root folder. Static files can be stored in any folder under the web root and accessed with a relative path to that root

Generally, there should be separate folders for the different types of static files such as JavaScript, CSS, Images, library scripts etc. in the wwwroot folder as shown below.

You can access static files with base URL and file name. For example, we can access above app.css file in the css folder by `http://localhost:<port>/css/app.css`.

Remember, you need to include a middleware for serving static files using `app.UseStaticFiles()` method in the `program.cs` file.

Q.27 Differentiate between Kestrel and IIS as hosting options for ASP.NET Core.

Kestrel is a lightweight, cross-platform, and open-source web server for ASP.NET Core that runs on Linux, Windows, and Mac. It is designed to be fast and scalable, and it is the preferred web server for all new ASP.NET applications.

IIS (Internet Information Services), on the other hand, is a web server that is developed and maintained only by Microsoft. It is a Windows-specific web server that is not cross-platform.

One of the main differences between Kestrel and IIS is that Kestrel is a cross-platform server that can run on Linux, Windows, and Mac, whereas IIS is Windows-specific. Another essential difference between the two is that Kestrel is fully open-source, whereas IIS is closed-source and developed and maintained only by Microsoft.

Q.28 What is the purpose of the wwwroot folder in an ASP.NET Core project, and what types of files are typically placed there?

All the static files that are required for our projects are stored under wwwroot Folder.

Generally CSS, javascript files, Images and other libraries such as bootstrap, JQuery are stored under wwwroot folder.

Q.29 What is the role of the appsettings.json file, and how can you access configuration values from it in an ASP.NET Core application?

In the Asp.Net Core application we may not find the web.config file. Instead we have a file named "appsettings.json". So to configure the settings like database connections, Mail settings or some other custom configuration settings we will use the "appsettings.json".

The appsettings.json file can be configured with Key and Value pair combinations. So, the Key will have single or multiple values.

```
{  
  
  "ConnectionStrings": {  
  
    "MyDatabase": "Server=localhost;Initial Catalog=MySampleDatabase;Trusted_Connection=Yes;MultipleActiveResultSets=true"  
  
  }  
  
}
```

#### Access configuration values:

You can access configuration values using the **IConfiguration** interface, which is provided by the **Microsoft.Extensions.Configuration** namespace. This interface allows you to access configuration values from various configuration sources, including **appsettings.json**.

In your code (such as in controllers, services, or middleware), inject **IConfiguration** into the constructor, and then use it to access configuration values.

Q.30 How does ASP.NET Core support cross-platform development, and what are the advantages of this approach?

ASP.NET Core is designed with cross-platform development in mind, allowing developers to build and run web applications on multiple operating systems, including Windows, macOS, and various Linux distributions. Here's how ASP.NET Core supports cross-platform development.

1. **Cross-Platform Runtime:** ASP.NET Core applications run on the cross-platform .NET Core runtime. Unlike the traditional .NET Framework, which is limited to Windows, .NET Core can run on various operating

systems. This enables developers to write ASP.NET Core applications once and run them on different platforms without modification.

2. **CLI Tooling:** ASP.NET Core includes a command-line interface (CLI) tooling that provides cross-platform support for various development tasks such as project scaffolding, building, testing, and publishing. Developers can use the CLI tools from their preferred terminal or command prompt, regardless of the operating system they are using.
3. **Cross-Platform Development Tools:** Popular integrated development environments (IDEs) like Visual Studio Code, Visual Studio for Mac, and JetBrains Rider provide excellent support for ASP.NET Core development on different platforms. These tools offer features such as code completion, debugging, and project management, making it easy for developers to work on ASP.NET Core projects irrespective of their operating system.
4. **Containerization:** ASP.NET Core applications can be containerized using technologies like Docker. Containerization enables developers to package their applications along with all dependencies into lightweight and portable containers, which can then be deployed and run consistently across different environments, regardless of the underlying operating system.
5. **Cloud Deployment:** ASP.NET Core applications can be easily deployed to various cloud platforms such as Microsoft Azure, Amazon Web Services (AWS), and Google Cloud Platform (GCP). Cloud providers offer support for ASP.NET Core applications on both Windows and Linux-based hosting environments, allowing developers to choose the platform that best suits their requirements.

6. **Open-Source Community Support:** ASP.NET Core has a vibrant open-source community that actively contributes to its development and provides support for cross-platform scenarios. This community-driven approach ensures that ASP.NET Core continues to improve and evolve as a cross-platform framework, with support for new operating systems, architectures, and development tools.

Advantages of Cross-Platform Development with ASP.NET Core:

**Increased Flexibility:** Developers can choose their preferred operating system for development, allowing them to work in environments they are most comfortable with.

**Cost Savings:** Cross-platform development reduces the need for expensive Windows licenses and proprietary development tools, making it more cost-effective for organizations.

**Broader Audience Reach:** Cross-platform applications can reach a wider audience by targeting users on different operating systems and devices.

**Improved Collaboration:** Cross-platform development encourages collaboration among developers with diverse backgrounds and preferences, fostering innovation and creativity within development teams.

Q.31 Explain the use of the `async` and `await` keywords in ASP.NET, and how they contribute to asynchronous programming.

In ASP.NET (including ASP.NET Core), the **`async`** and **`await`** keywords are used to implement asynchronous programming, which allows the server to handle multiple concurrent requests more efficiently. These keywords are essential for

writing asynchronous code that doesn't block the execution thread, enabling better scalability and responsiveness in web applications.

Here's how **async** and **await** contribute to asynchronous programming in ASP.NET:

1. **Asynchronous Operations:** By marking a method with the **async** keyword, you're indicating that the method contains asynchronous operations. Asynchronous operations typically involve I/O-bound tasks, such as making database queries, calling external APIs, or reading files. Instead of blocking the execution thread while waiting for these operations to complete, the **async** keyword allows the thread to be released to handle other tasks.
2. **Non-Blocking Execution:** The **await** keyword is used within an **async** method to asynchronously wait for the completion of another asynchronous operation. When you **await** an operation, the current method's execution is paused, and control is returned to the caller. This allows the execution thread to be freed up to handle other tasks while waiting for the awaited operation to complete. Once the awaited operation finishes, the execution resumes from the point of the **await** statement.
3. **Improved Scalability:** Asynchronous programming with **async** and **await** improves the scalability of ASP.NET applications by allowing the server to handle more concurrent requests with fewer resources. Instead of blocking threads while waiting for I/O operations to complete, asynchronous methods release the thread back to the thread pool, allowing it to be used to serve other requests. This improves the overall



throughput and responsiveness of the application, especially under heavy load.

4. **Avoiding Deadlocks:** Asynchronous programming helps prevent deadlocks by freeing up the execution thread to handle other tasks while waiting for asynchronous operations to complete. Deadlocks occur when threads are blocked waiting for each other, which can happen in synchronous code when multiple threads are waiting for I/O operations to finish simultaneously. By using asynchronous programming, you can avoid such deadlocks and improve the robustness of your ASP.NET applications.
5. **Better User Experience:** Asynchronous programming contributes to a better user experience by reducing the latency and improving the responsiveness of web applications. By freeing up the execution thread to handle other tasks while waiting for asynchronous operations, the server can respond to user requests more quickly, resulting in faster page loads and smoother interactions.

In summary, the **async** and **await** keywords in ASP.NET enable developers to write asynchronous code that improves scalability, responsiveness, and resource utilization in web applications. By allowing the server to handle multiple concurrent requests more efficiently, asynchronous programming contributes to better performance and a more responsive user experience.

Q.32 What is the role of the **IWebHostBuilder** in an ASP.NET Core application, and how is it used in the application startup process?

In ASP.NET Core, the **IWebHostBuilder** interface plays a crucial role in configuring and building the web host for an application. It is used in the application startup process to set up the web server, configure the application's services, and specify startup options. The **IWebHostBuilder** interface is typically used in the **Program.cs** file to create and configure the web host.

Here's how the **IWebHostBuilder** is used in the application startup process:

1. **Creating the Web Host:** The **IWebHostBuilder** interface is used to create an instance of the web host for the ASP.NET Core application. Typically, this is done in the **CreateWebHostBuilder** method within the **Program.cs** file. The **CreateWebHostBuilder** method returns an instance of **IWebHostBuilder**, which is used to configure the web host.
2. **Configuring the Web Host:** Once the **IWebHostBuilder** instance is created, you can use its methods to configure various aspects of the web host, such as the web server, logging, and application settings. Common methods used for configuration include **UseStartup**, **UseKestrel**, **ConfigureAppConfiguration**, and **ConfigureLogging**.
3. **Setting up Services:** The **ConfigureServices** method of the **IWebHostBuilder** interface is used to register the application's services with the dependency injection (DI) container. Services such as database contexts, repositories, authentication, and authorization are typically

configured in this method using the built-in DI container provided by ASP.NET Core.

4. **Configuring Middleware:** Middleware components are configured using the **Configure** method of the **IWebHostBuilder** interface. Middleware components handle requests and responses in the ASP.NET Core pipeline. Middleware components can be added, configured, and ordered within the **Configure** method to define the request processing pipeline for the application.
5. **Building the Web Host:** Once the **IWebHostBuilder** instance is fully configured, the **Build** method is called to create and initialize the web host. The **Build** method returns an instance of **IWebHost**, which represents the running web host for the ASP.NET Core application.

Here's a typical example of using **IWebHostBuilder** in the **Program.cs** file of an ASP.NET Core application.

```
using Microsoft.AspNetCore.Hosting;  
using Microsoft.Extensions.Hosting;
```

```
public class Program  
{  
    public static void Main(string[] args)  
    {  
        CreateHostBuilder(args).Build().Run();  
    }  
}
```

```
public static IHostBuilder CreateHostBuilder(string[] args) =>  
    Host.CreateDefaultBuilder(args)
```

```
.ConfigureWebHostDefaults(webBuilder =>
{
    webBuilder
        .UseStartup<Startup>()
        .UseKestrel(); // Example of configuring the web server
});
}
```

### Q.33 What are areas in asp.net MVC?

In ASP.NET MVC, an "Area" is a way to organize related functionality into a group within an MVC application. It provides a means to divide a large web application into smaller and more manageable sections, each with its own controllers, views, and models. Areas help maintain a clean and structured codebase, especially in larger applications with multiple modules or distinct functional areas.

Here are some key points about areas in ASP.NET MVC:

1. **Logical Segregation:** Areas allow you to logically segregate different parts of your application based on functionality or features. For example, you could have separate areas for user management, administration, reporting, or any other module within your application.
2. **Separate Folder Structure:** Each area has its own folder structure, typically containing subfolders for controllers, views, and models. This separation helps maintain a clear organization of files related to each area, making it easier to navigate and manage the codebase.
3. **Independent Routing:** Areas have their own route configuration, allowing you to define custom routes specific to each area. This means that controllers within an area can have their own route patterns without conflicting with routes defined in other areas.
4. **Shared Resources:** While areas promote separation of concerns, they also allow sharing resources between different areas when needed. For example, you can share common layouts, partial views, or reusable components across multiple areas to avoid code duplication.

5. **Controller Namespace:** Controllers within an area typically belong to a separate namespace, which helps prevent naming conflicts with controllers from other areas. This namespace isolation allows you to have controllers with the same name across different areas without conflict.
6. **Registration and Configuration:** Areas need to be registered and configured in the MVC application during startup. This involves adding routes for each area and specifying the namespace where controllers for that area can be found.

To create an area in an ASP.NET MVC application, you typically follow these steps:

1. Right-click on the project in Visual Studio.
2. Select "Add" > "Area" from the context menu.
3. Provide a name for the area and click "Add."
4. Visual Studio creates a new folder structure for the area and adds default folders for controllers, views, and models.
5. You can then add controllers, views, and models specific to that area within the respective folders.

Overall, areas in ASP.NET MVC provide a flexible and structured way to organize and manage large web applications, enabling better code organization, maintenance, and scalability.