

.NET TECHNOLOGIES

UNIT 1

The .NET Framework

- The .NET Framework is a comprehensive software platform developed by Microsoft that provides tools, libraries, and runtime environments for building and running various types of applications.
- The .NET Framework was primarily used for Windows applications, although more versions like .NET Core (now .NET 5+ /6+/7+/8+) offer cross-platform capabilities.
- .NET Framework consists of the common language runtime (CLR) and the .NET Framework class library.

.NET Languages

- .NET Languages are computer programming languages that are used to produce programs that execute within the Microsoft .NET Framework.
- Microsoft provides several such languages, including C#, Visual Basic .NET, and C++/CLI
- .NET Framework supports more than 60 programming languages in which 11 programming languages are designed and developed by Microsoft.
- The remaining Non-Microsoft Languages which are supported by .NET Framework but not designed and developed by Microsoft.

.NET Languages

- The types of applications that can be built in the .Net framework are classified broadly into the following categories:
 - **WinForms** – This is used for developing Forms-based applications, which would run on an end-user machine. Notepad is an example of a client-based application.
 - **ASP.Net** – This is used for developing web-based applications, which are made to run on any browser such as Internet Explorer, Chrome, or Firefox. The Web application would be processed on a server, which would have Internet Information Services Installed.
 - **ADO.Net** – This technology is used to develop applications to interact with The .Net Framework Databases such as Oracle or Microsoft SQL Server.

Few examples of Microsoft .NET languages

C# - Microsoft's flagship .NET Framework language which bears similarities to the C++ and Java languages.

Visual Basic .NET - A completely redesigned version of the Visual Basic language for the .NET Framework. This also includes Visual Basic 2005 (v8.0).

VBx, a dynamic version of Visual Basic .NET that runs on top of the Dynamic Language Runtime.

C++/CLI and the deprecated Managed C++ - A managed version of the C++ language.

J# - A Java and J++ .NET transitional language.

JScript .NET - A compiled version of the JScript language.

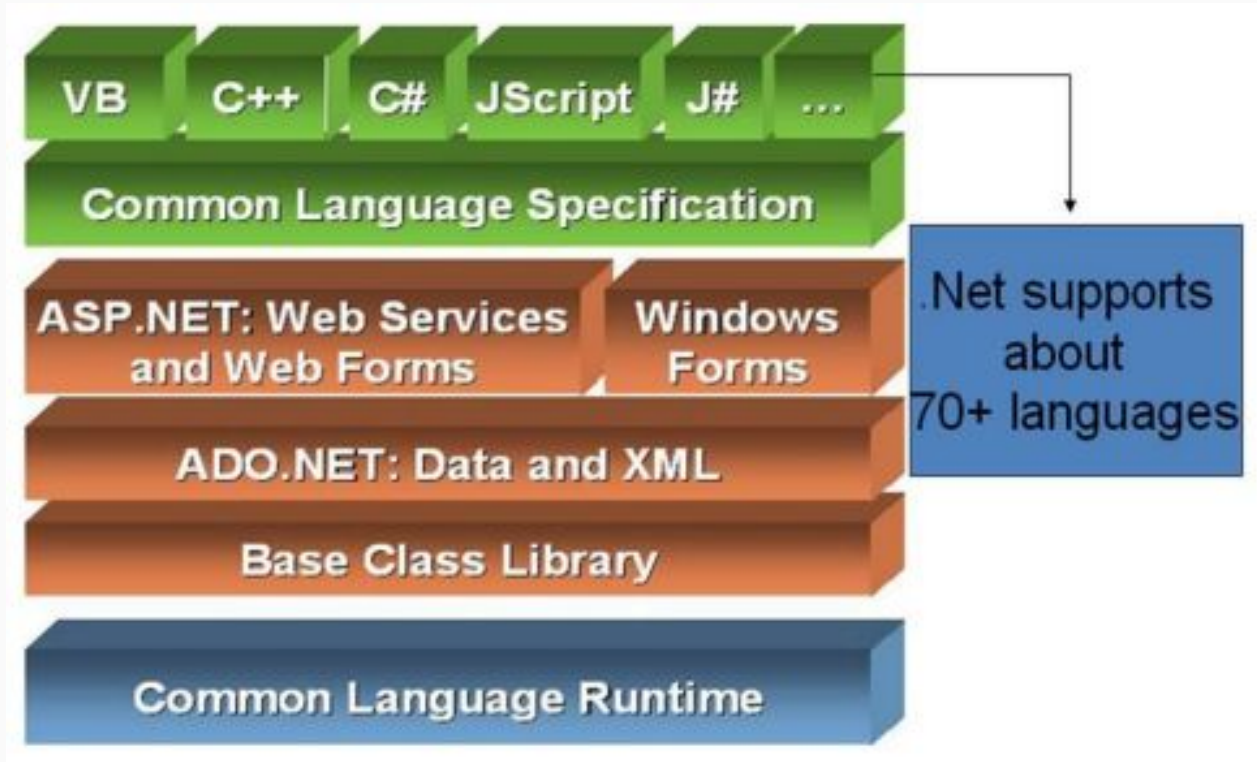
Windows PowerShell - An interactive command line shell/scripting language that provides full access to the .NET Framework.

IronPython - A .NET implementation of the Python programming language developed by Jim Hugunin at Microsoft.

IronRuby - A dynamically compiled version of the Ruby programming language targeting the .NET Framework.

F#, a member of the ML programming language family.

Components of .NET Framework



The .Net framework mainly contains two components :

1. Common Language Runtime(CLR)

2. .Net Framework Class Library (FCL)

Components of .NET Framework

1. Common Language Runtime(CLR)

- .Net Framework provides runtime environment called Common Language Runtime (CLR).
- It runs all the .Net programs.
- CLR provides memory management and thread management.
- It allocates the memory for scope and deallocates the memory.
- The code which runs under the CLR is called as Managed Code.
- Programmers need not to worry on managing the memory if the programs are running under the CLR. (memory management and thread management)

Components of .NET Framework

1. Common Language Runtime(CLR)

- Language Compilers (e.g. C#, VB.Net, J#) will convert the Code/Program to Microsoft Intermediate Language-MSIL/Common Intermediate Language(CIL) inturn this will be converted to Native Code by CLR.
- There are currently over 15 language compilers being built by Microsoft and other companies also producing the code that will execute under CLR.



Components of .NET Framework

2. .Net Framework Class Library (FCL)

- It accesses the library classes and methods.
- It is also called as Base Class Library.
- It is common for all types of application.
- Following are the applications in .Net Class Library:
 1. XML web services
 2. Windows services
 3. Windows application
 4. Web applications
 5. Console application

C# Language Basics

- C# (C-Sharp) is a object oriented programming language developed by Microsoft that runs on the .NET Framework.
- C# is used to develop web apps, desktop apps, mobile apps, games, and much more. C# is an object-oriented programming language.
- In Object-Oriented Programming methodology, a program consists of various objects that interact with each other by means of actions.
- The actions that an object may take are called methods.
- Objects of the same kind are said to have the same type or, are said to be in the same class.

C# Syntax

`using System;` → **use classes from the System namespace**

`namespace HelloWorld` → **namespace is used to organize your code, and it is a container for classes and other namespaces**

`{`
`class Program`
`{`
`static void Main(string[] args)`

class is a container for data and methods
Main method- Any code inside its curly brackets {} will be executed

`{`
`Console.WriteLine("Hello World!");` → **Console is a class of the System namespace, which has a WriteLine() method that is used to output/print text.**
`}`
`}`

If you omit the using System line, you would have to write System.Console.WriteLine() to print/output text.

Casting objects

- In C#, there are two types of casting:
- **Implicit Casting (automatically)** - converting a smaller type to a larger type size
char -> int -> long -> float -> double

Example:

```
int myInt = 9;
```

```
double myDouble = myInt;    // Automatic casting: int to double
```

```
Console.WriteLine(myInt);   // Outputs 9
```

```
Console.WriteLine(myDouble); // Outputs 9
```

Casting objects

- **Explicit Casting (manually)** - converting a larger type to a smaller size type
double -> float -> long -> int -> char

Example:

```
double myDouble = 9.78;
```

```
int myInt = (int) myDouble; // Manual casting: double to int
```

```
Console.WriteLine(myDouble); // Outputs 9.78
```

```
Console.WriteLine(myInt); // Outputs 9
```

It is also possible to convert data types explicitly by using built-in methods, such as

Convert.ToBoolean,
Convert.ToDouble,
Convert.ToString,

Convert.ToInt32 (int) and
Convert.ToInt64 (long)

Get User Input

- You have already learned that `Console.WriteLine()` is used to output (print) values. Now we will use `Console.ReadLine()` to get user input.
- In the following example, the user can input his or hers username, which is stored in the variable `userName`. Then we print the value of `userName`:

Example:

```
Console.WriteLine("Enter username:");  
string userName = Console.ReadLine();  
Console.WriteLine("Username is: " + userName);
```

- The `Console.ReadLine()` method returns a string. Therefore, you cannot get information from another data type, such as `int`. The following program will cause an error:

Example:

```
Console.WriteLine("Enter your age:");  
int age = Convert.ToInt32(Console.ReadLine());  
Console.WriteLine("Your age is: " + age);
```

OBJECT BASED MANIPULATION

A C# program consists of the following parts:

- Namespace declaration
- A class
- Class methods
- Class attributes
- A Main method
- Statements and Expressions
- Comments

```
using System;

namespace HelloWorld
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hello World!");
            // This is a comment
        }
    }
}
```


Classes

- Class is a blueprint for a data type.
- Objects are instances of a class.
- The methods and variables that constitute a class are called members of the class.

Defining a Class

A class definition starts with the keyword `class` followed by the class name; and the class body enclosed by a pair of curly braces.

```
<access specifier> class class_name {  
    // member variables  
    <access specifier> <data type> variable1;  
    <access specifier> <data type> variable2;  
    ...  
    <access specifier> <data type> variableN;  
    // member methods  
    ...  
    <access specifier> <return type> methodN(parameter_list) {  
        // method body  
    }  
}
```

Class methods

A method is a group of statements that together perform a task. Every C# program has at least one class with a method named Main.

Defining Methods in C#

Syntax

```
<Access Specifier><Return Type><Method Name>(Parameter List) {
```

```
Method Body
```

```
}
```

Modifier

Description

`public`

The code is accessible for all classes

`private`

The code is only accessible within the same class

`protected`

The code is accessible within the same class, or in a class that is inherited from that class. You will learn more about [inheritance](#) in a later chapter

`internal`

The code is only accessible within its own assembly, but not from another assembly. You will learn more about this in a later chapter

Calling Method:

```
using System;
namespace CalculatorApplication {
    class NumberManipulator {
        public int FindMax(int num1, int num2) {int result;
            if (num1 > num2)
                result = num1;
            else
                result = num2;
            return result;}
        static void Main(string[] args) {int a = 100;
            int b = 200;
            int ret;
            NumberManipulator n = new NumberManipulator();
            ret = n.FindMax(a, b);
            Console.WriteLine("Max value is : {0}", ret );
            Console.ReadLine();
        }
    }
}
```

Namespace and Assemblies

- A namespace is designed for providing a way to keep one set of names separate from another. The class names declared in one namespace does not conflict with the same class names declared in another.
- A namespace definition begins with the keyword namespace followed by the namespace name as follows –

```
namespace namespace_name {  
    // code declarations  
}
```

- To call the namespace-enabled version of either function or variable, prepend the namespace name as follows –

```
namespace_name.item_name;
```

```
using System;
namespace first_space {
    class namespace_cl {
        public void func() {Console.WriteLine("Inside first_space"); }
    }
}
namespace second_space {
    class namespace_cl {
        public void func() {Console.WriteLine("Inside second_space");}
    }
}
class TestClass {
    static void Main(string[] args) {
        first_space.namespace_cl fc = new first_space.namespace_cl();
        second_space.namespace_cl sc = new second_space.namespace_cl();
        fc.func();
        sc.func();
        Console.ReadKey();
    }
}
```


- An Assembly is a basic building block of .Net Framework applications. It is basically a compiled code that can be executed by the CLR. An assembly is a collection of types and resources that are built to work together and form a logical unit of functionality. An Assembly can be a DLL or exe depending upon the project that we choose.
- Assemblies are basically the following two types:

Private Assembly

Shared Assembly

1. Private Assembly

- It is an assembly that is being used by a single application only. Suppose we have a project in which we refer to a DLL(Dynamic Link Libraries) so when we build that project that DLL will be copied to the bin folder of our project. That DLL becomes a private assembly within our project. Generally, the DLLs that are meant for a specific project are private assemblies.

2. Shared Assembly

- Assemblies that can be used in more than one project are known to be a shared assembly. Shared assemblies are generally installed in the GAC(Global Assembly Cache). Assemblies that are installed in the GAC are made available to all the .Net applications on that machine.

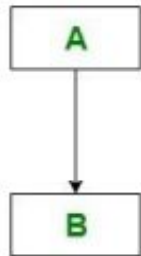
Inheritance

- Inheritance is an important pillar of OOP(Object Oriented Programming). It is the mechanism in C# by which one class is allowed to inherit the features(fields and methods) of another class.

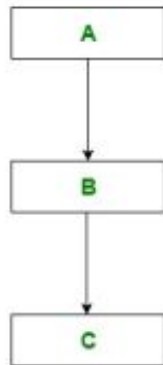
Important terminology:

- Super Class: The class whose features are inherited is known as super class(or a base class or a parent class).
- Sub Class: The class that inherits the other class is known as subclass(or a derived class, extended class, or child class). The subclass can add its own fields and methods in addition to the superclass fields and methods.
- Reusability: Inheritance supports the concept of “reusability”, i.e. when we want to create a new class and there is already a class that includes some of the code that we want, we can derive our new class from the existing class. By doing this, we are reusing the fields and methods of the existing class.

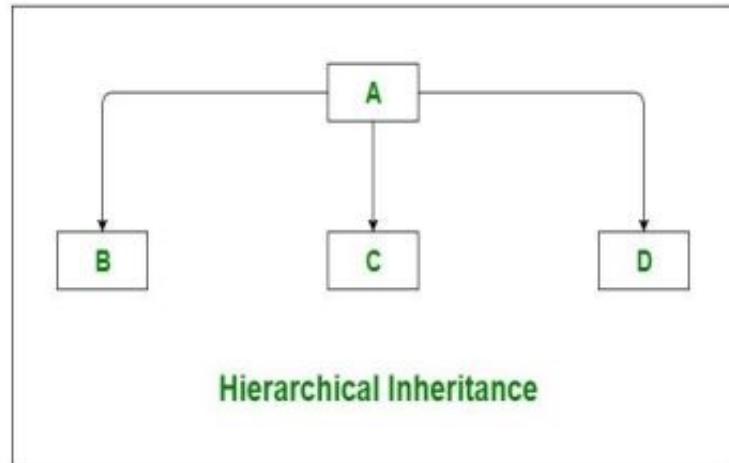
- **Single Inheritance:** In single inheritance, subclasses inherit the features of one superclass. In image below, the class A serves as a base class for the derived class B.
- **Multilevel Inheritance:** In Multilevel Inheritance, a derived class will be inheriting a base class and as well as the derived class also act as the base class to other class. In below image, class A serves as a base class for the derived class B, which in turn serves as a base class for the derived class C.
- **Hierarchical Inheritance:** In Hierarchical Inheritance, one class serves as a superclass (base class) for more than one subclass. In below image, class A serves as a base class for the derived class B, C, and D.



Single Inheritance

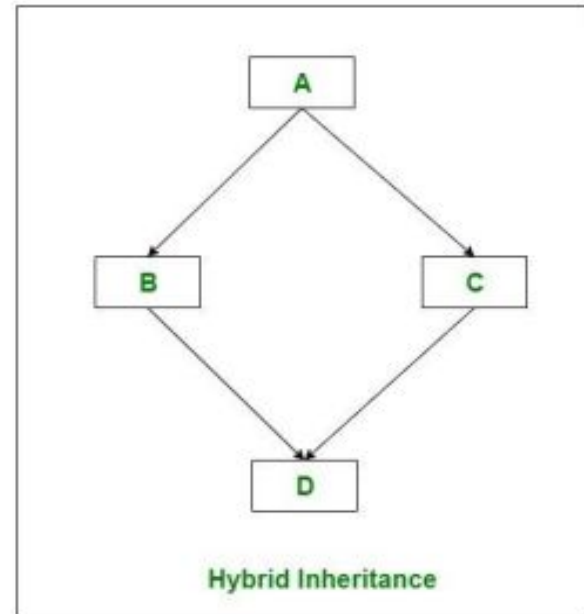
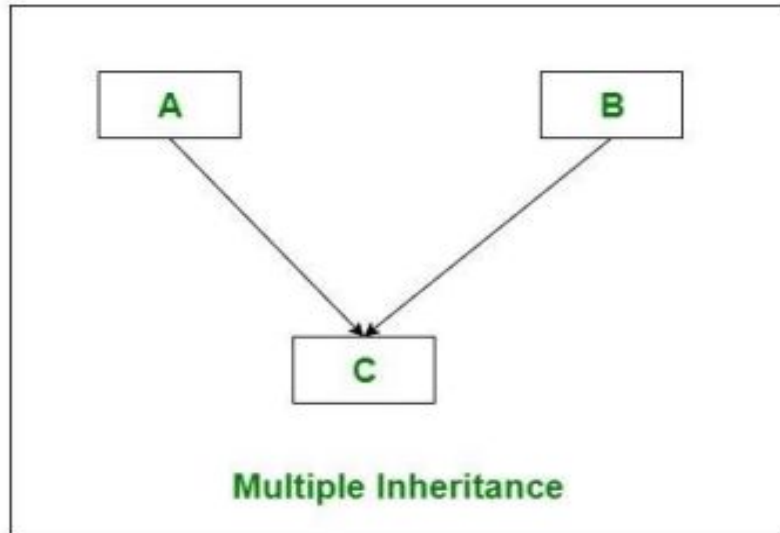


Multilevel Inheritance



Hierarchical Inheritance

- **Multiple Inheritance(Through Interfaces):**In Multiple inheritance, one class can have more than one superclass and inherit features from all parent classes. Please note that C# does not support multiple inheritance with classes. In C#, we can achieve multiple inheritance only through Interfaces. In the image below, Class C is derived from interface A and B.
- **Hybrid Inheritance(Through Interfaces):** It is a mix of two or more of the above types of inheritance. Since C# doesn't support multiple inheritance with classes, the hybrid inheritance is also not possible with classes. In C#, we can achieve hybrid inheritance only through Interfaces.



Static members

- Declaring a member of a class as static, it means no matter how many objects of the class are created, there is only one copy of the static member.
- The keyword static implies that only one instance of the member exists for a class.
- Static variables are used for defining constants because their values can be retrieved by invoking the class without creating an instance of it.
- Static variables can be initialized outside the member function or class definition.
- The static variables can also be initialized inside the class definition.

Static members

```
namespace StaticVarApplication {
    class StaticVar {
        public static int num;
        public void count() {
            num++;
        }
        public int getNum() {
            return num;
        }
    }
    class StaticTester {
        static void Main(string[] args) {
            StaticVar s1 = new StaticVar();
            StaticVar s2 = new StaticVar();
            s1.count();
            s1.count();
            s1.count();
            s2.count();
            s2.count();
            s2.count();
            Console.WriteLine("Variable num for s1: {0}", s1.getNum());
            Console.WriteLine("Variable num for s2: {0}", s2.getNum());
            Console.ReadKey();
        }
    }
}
```

Partial class

- A partial class splits the definition of a class over two or more source files. We can create a class definition in multiple files but it will be compiled as one class.
- Suppose we have a "Person" class. That definition is divided into the two source files "Person1.cs" and "Person2.cs". Then these two files have a class that is a partial class.

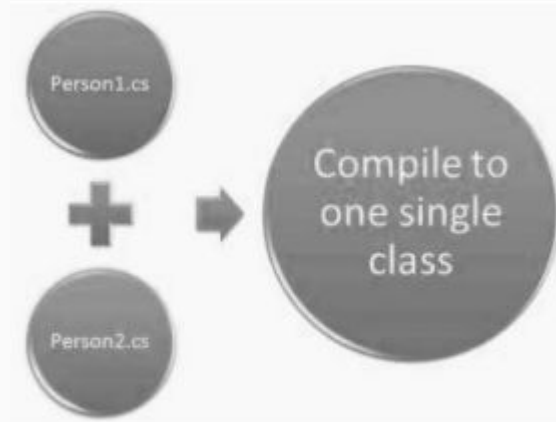


fig. 2.10 partial class

- ASP.NET is an open source web framework, created by Microsoft, for building modern web apps and services with .NET.
- ASP.NET is cross platform and runs on Windows, Linux, macOS, and Docker.
- ASP.NET is a web framework designed and developed by Microsoft.
- It is used to develop websites, web applications and web services.
- It provides fantastic integration of HTML, CSS and JavaScript.
- It is built on the Common Language Runtime (CLR) and allows programmers to write code using any supported .NET language.

ASP .NET

Anatomy of a Web Form

- An ASP.NET page is made up of a number of server controls along with HTML controls, text, and images.
- Sensitive data from the page and the states of different controls on the page are stored in hidden fields that form the context of that page request.
- An ASP.NET page is also a server side file saved with the .aspx extension.
- It is modular in nature and can be divided into the following core sections:
 1. **Page Directives:** set up the environment for the page to run
 2. **Code Section:** code section or the code behind file provides all these event handler routines, and other functions used by the developer.
 3. **Page Layout:** The page layout provides the interface of the page.

Anatomy of a Web Form

Webform.aspx

```
1  <%@ Page Language="C#" AutoEventWireup="true" CodeBehind="WebForm1.aspx.cs"
2     Inherits="WebApplication2.WebForm1" %>
3
4  <!DOCTYPE html>
5
6  <html xmlns="http://www.w3.org/1999/xhtml">
7  <head runat="server">
8     <title></title>
9  </head>
10 <body>
11     <h1>Welcome to my Page</h1>
12     <form id="form1" runat="server">
13         <div>
14             <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
15             <asp:Button ID="Button1" runat="server" Text="Button" />
16         </div>
17     </form>
18 </body>
19 </html>
```

Anatomy of a Web Form

Explanation of the elements:

1. **@ Page directive:** The `@ Page` directive specifies attributes of the page, such as the language, code-behind file, and class inheritance.
2. **Static HTML:** The `<h1>` tag and other HTML elements are rendered exactly as they are, displaying "Welcome to My ASP.NET Page".
3. **HTML and server-side comments:** Both HTML (`<!-- -->`) and server-side (`<%-- --%>`) comments can be used to add explanatory text to the page, though server-side comments won't be sent to the client.
4. **<form> and ASP.NET server controls:** Here, we have an ASP.NET form containing a TextBox control (`<asp:TextBox>`) and a Button control (`<asp:Button>`). These controls are server-side controls.

Page Directives

- The Page directive defines the attributes specific to the page file for the page parser and the compiler.
- The page directive gives ASP.NET basic information about how to compile the page.
- It indicates the language used for the code and the way to connect event handlers.
- If the code-behind approach is used, the page directive also indicates where the code file is located and the name of the custom page class.
- The basic syntax of the Page directive is:

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Default.aspx.cs" Inherits="_Default" Trace="true" %>
```



Some Page Directive Attributes

Attribute name	Description
AutoEventWireup	This attribute specifies whether the page parser should automatically bind page events to methods in the code-behind class. The default value is <code>true</code> .
Language	This attribute specifies the programming language for the code-behind file.
CodeBehind	The name of the code behind file.
Inherits	This attribute specifies the name of the code-behind or other class that inherits from the <code>Page</code> class.
ClientTarget	This attribute specifies the browser for which the server controls should render content.
ErrorPage	This attribute specifies the URL that the page should redirect to if an unhandled page exception occurs.

Code-behind Class

1. The code-behind class is an essential concept that separates the visual presentation (UI markup) of a web page from the code logic that controls its behavior and functionality
2. It is composed in a different class record that can have the extension of `.aspx.cs` or `.aspx.vb` relying upon the language used.
3. This relationship between your class and the web page is established by a Page directive at the top of the `.aspx` file using the `inherits` attribute.



Example in C#:

Markup (WebForm.aspx):

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

Code-Behind (WebForm.aspx.cs):

```
using System;
using System.Web.UI;

namespace MyWebApp
{
    public partial class WebForm : Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
            // Page load logic
        }

        protected void btnSubmit_Click(object sender, EventArgs e)
        {
            // Handle button click event
            lblMessage.Text = "Button clicked!";
        }
    }
}
```

Adding Event Handlers

- An event is an action or occurrence such as a mouse click, a key press, mouse movements, or any system-generated notification. A process communicates through events.
- For example, interrupts are system-generated events. When events occur, the application should be able to respond to it and manage it.
- Events in ASP.NET raised at the client machine, and handled at the server machine.
- For example, a user clicks a button displayed in the browser. A Click event is raised. The browser handles this client-side event by posting it to the server.
- The server has a subroutine describing what to do when the event is raised; it is called the event-handler.
- Therefore, when the event message is transmitted to the server, it checks whether the Click event has an associated event handler. If it has, the event handler is executed.

Adding Event Handlers

- Event Arguments
- ASP.NET event handlers generally take two parameters and return void. The first parameter represents the object raising the event and the second parameter is event argument.
- The general syntax of an event is:

```
private void EventName (object sender, EventArgs e);
```

Example in C#:

Markup (WebForm.aspx):

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

Code-Behind (WebForm.aspx.cs):

```
protected void btnSubmit_Click(object sender, EventArgs e)
{
    // Handle button click event
    lblMessage.Text = "Button clicked!";
}
```


Adding Event Handlers

- Application and Session Events

The most important application events are:

1. `Application_Start` - It is raised when the application/website is started.
2. `Application_End` - It is raised when the application/website is stopped.
3. Similarly, the most used Session events are:
4. `Session_Start` - It is raised when a user first requests a page from the application.
5. `Session_End` - It is raised when the session ends.

Adding Event Handlers

- Page and Control Events

Common page and control events are:

1. **DataBinding** - It is raised when a control binds to a data source.
2. **Disposed** - It is raised when the page or the control is released.
3. **Error** - It is a page event, occurs when an unhandled exception is thrown.
4. **Init** - It is raised when the page or the control is initialized.
5. **Load** - It is raised when the page or a control is loaded.
6. **PreRender** - It is raised when the page or the control is to be rendered.
7. **Unload** - It is raised when the page or control is unloaded from memory.

The common control events are:

Event	Attribute	Controls
Click	OnClick	Button, image button, link button, image map
Command	OnCommand	Button, image button, link button
TextChanged	OnTextChanged	Text box
SelectedIndexChanged	OnSelectedIndexChanged	Drop-down list, list box, radio button list, check box list.
CheckedChanged	OnCheckedChanged	Check box, radio button

ANATOMY OF AN ASP.NET APPLICATION

ASP.NET File Types

.asax	Typically a Global.asax file that represents the application class and contains event handlers that run at various points in the application life cycle.
.ascx	A Web user control file that defines a custom functionality that you can add to any ASP.NET Web Forms page.
.ashx	A handler file that is invoked in response to a Web request in order to generate dynamic content.
.asmx	An XML Web services file that contains classes and methods that can be invoked by other Web applications.
.aspx	An ASP.NET Web Forms page that can contain Web controls and presentation and business logic.

ASP.NET File Types

.cd	A class diagram file.
.config	A configuration file contains XML elements that represent settings for ASP.NET features.
.cs, .vb	Source code files (.cs or .vb files) that define code that can be shared between pages
.csproj, .vbproj	A project file for a Visual Studio Web-application project.
.dll	A compiled class library file (assembly).
.master	A master page that defines the layout for other Web pages in the application.

ASP.NET File Types

<code>.mdb</code>	An Access database file.
<code>.resources, .resx</code>	A resource file that contains resource strings that refer to images, localizable text, or other data.
<code>.sitemap</code>	A sitemap file that defines the logical structure of the Web application. ASP.NET includes a default sitemap provider that uses sitemap files to display a navigational control in a Web page.
<code>.sln</code>	A solution file for a Visual Studio project.

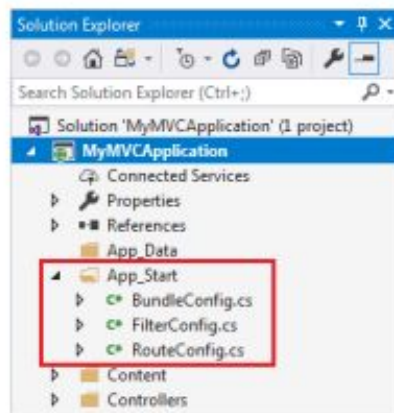
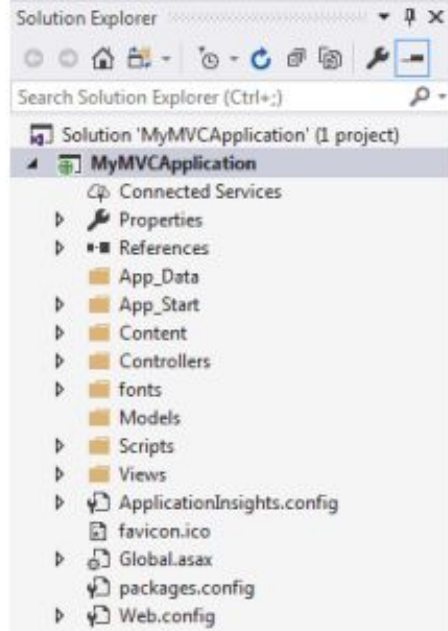
ASP.NET Web Folders

- App_Data

The App_Data folder can contain application data files like LocalDB, .mdf files, XML files, and other data related files. IIS will never serve files from App_Data folder.

- App_Start

The App_Start folder can contain class files that will be executed when the application starts. Typically, these would be config files like AuthConfig.cs, BundleConfig.cs, FilterConfig.cs, RouteConfig.cs etc. MVC 5 includes BundleConfig.cs, FilterConfig.cs and RouteConfig.cs by default. We will see the significance of these files later.



- Content

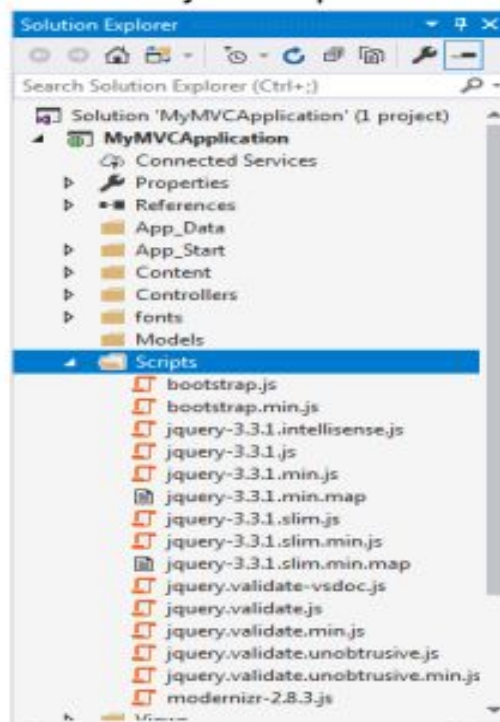
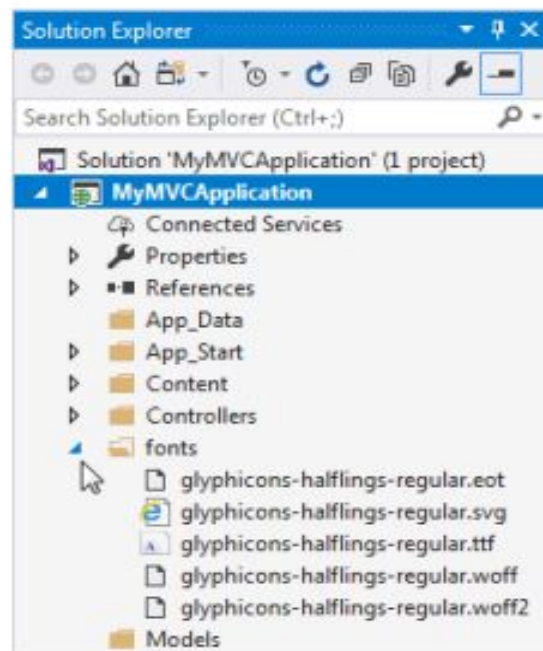
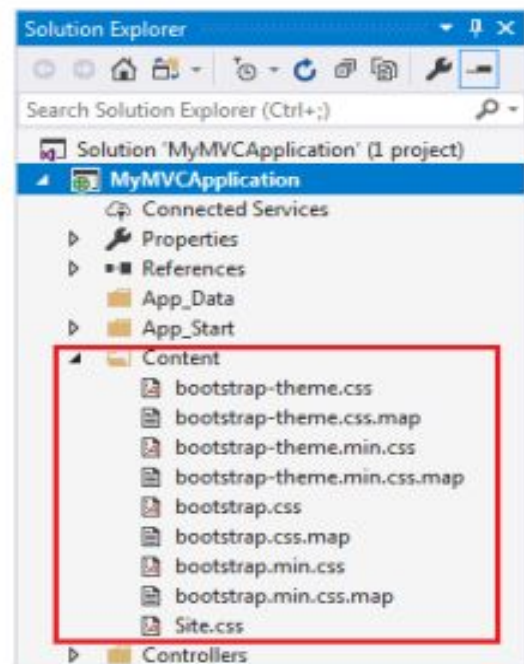
The Content folder contains static files like CSS files, images, and icons files. MVC 5 application includes bootstrap.css, bootstrap.min.css, and Site.css by default.

- fonts

The Fonts folder contains custom font files for your application.

- Scripts

The Scripts folder contains JavaScript or VBScript files for the application. MVC 5 includes javascript files for bootstrap, jquery 1.10, and modernizer by default.



- Global.asax

Global.asax file allows you to write code that runs in response to application-level events, such as `Application_BeginRequest`, `application_start`, `application_error`, `session_start`, `session_end`, etc.

- Packages.config

Packages.config file is managed by NuGet to track what packages and versions you have installed in the application.

- Web.config

Web.config file contains application-level configurations.

- The HTML server controls are basically the standard HTML controls enhanced to enable server side processing.
- By default, HTML elements(anchor tag, header tags ,input elements etc) on an ASP.NET Web page are not available to the server.
- These components are treated as simple text and pass through to the browser.
- HTML server controls provide automatic state management and server-side events.

HTML SERVER CONTROLS

CONVERT HTML ELEMENT TO SERVER CONTROL

- Adding the attribute **runat="server"** and adding an **id attribute** to make them available for serverside processing.

For example, consider the HTML input control:

```
<input type="text" size="40">
```

It could be converted to a server control, by adding the runat and id attribute:

```
<input type="text" id="testtext" size="40" runat="server">
```

- All the HTML Server controls can be accessed through the Request object.
- The HTML server controls have the same HTML output and the same properties as their corresponding HTML tags.

Example

```
<form id="form1" runat="server">
<div>
<input id="Text1" type="text" runat="server"/>
<asp:Button ID="Button1" runat="server" Text="Button" OnClick="Button1_Click"/>
</div>
</form>
```

```
protected void Button1_Click(object sender, EventArgs e)
{
string a = Request.Form["Text1"];
Response.Write(a);
}
```

The following are HTML server controls that are available in ASP.NET:-

- HtmlButton Control
- HtmlForm Control
- HtmlImage Control
- HtmlInputButton Control
- HtmlInputCheckBox Control
- HtmlInputFile Control
- HtmlInputHidden Control
- HtmlInputImage Control
- HtmlInputRadioButton Control
- HtmlInputText Control
- HtmlSelect Control
- HtmlTable Control
- HtmlTableCell Control
- HtmlTableCell Control
- HtmlTextArea Control

VIEW STATE

- ViewState is a built-in feature in ASP.NET that allows you to preserve and maintain the state of controls and data between postbacks.
- Postbacks occur when a page is submitted to the server for processing, typically triggered by user actions like button clicks.
- It is a Page-Level State Management technique.

Advantages of View State

- **Simplicity and Ease of Use** - There is no need for complex codes and logical thinking to implement ViewState. It is simple and makes storing form data between page submissions easy.
- **Flexibility** - It is easy to enable, configure, and disable View State properties on a control-by-control basis.
- **Server-Independent** - There are absolutely no server resources required to use View State. It is contained in a structure within the page load.
- **Enhanced Security** - The View State values go through hashing, encoding, and compression for Unicode implementation.

VIEW STATE

Disadvantages of View State

- **Security Risk:** The Information of View State can be seen in the page output source directly. You can manually encrypt and decrypt the contents of a Hidden Field, but It requires extra coding. If security is a concern then consider using a Server-Based state Mechanism so that no sensitive information is sent to the client.
- **Performance:** Performance is not good if we use a large amount of data because View State is stored in the page itself and storing a large value can cause the page to be slow.
- **Device limitation:** Mobile Devices might not have the memory capacity to store a large amount of View State data.
- It can store values for the same page only

THE HTMLCONTROL CLASS

- The `HtmlControl` object is pretty important to HTML server controls. Because every property and method it has is inherited by every HTML server control
- The HTML server controls have their own properties and methods, as well, but they all have the properties and methods contained in the `HtmlControl` object.

Property	Description
<code>Attribute</code>	Returns the object's attributes collection.
<code>Disabled</code>	A Boolean (true or false) value that you can get or set that indicates whether a control is disabled.
<code>EnableViewState</code>	A Boolean (true or false) value that you can get or set that indicates whether a control should maintain its viewstate.
<code>ID</code>	A string that you can get or set that defines the Identifier for the control.
<code>Style</code>	Returns the <code>CSSStyleCollection</code> for a control.
<code>TagName</code>	Returns the tag name of an element such as <code>input</code> or <code>div</code> .
<code>Visible</code>	A Boolean (true or false) value that you can get or set that indicates whether a control is rendered to HTML for delivery to the client's browser.

HtmlContainerControl Class

- HtmlContainerControl is used as the parent for any HTML control that requires a closing tag, such as div, form, or select.
- This class actually inherits all its properties and methods from the HtmlControl class and adds a few of its own. So to clarify, any object that is a container-type object doesn't directly inherit from the HtmlControl class.
- The HtmlContainerControl actually inherits the HtmlControl, and then when a container-type object uses the HtmlContainerControl it gets the HtmlControl class's objects that way.

Html InputControl Class

- Just like the HtmlContainerControl, the HtmlInput inherits from the HtmlControl and adds a few properties of its own for the different object that live off it.
- It brings three additional properties to the table

Property	Description
Name	Gets or sets the unique name for the <code>HtmlInput</code> control.
Type	Determines what kind of <code>Input</code> element the <code>HtmlInput</code> control is.
Value	Gets or sets the value of the content of the <code>HtmlInput</code> object.

The following controls, which are based on the HTML INPUT element, are available on the HTML tab of the Toolbox:

- **Input (Button)** control: INPUT type="button" element
- **Input (Checkbox)** control: INPUT type="checkbox" element
- **Input (File)** control: INPUT type="file" element
- **Input (Hidden)** control: INPUT type="hidden" element
- **Input (Password)** control: INPUT type="password" element
- **Input (Radio)** control: INPUT type="radio" element
- **Input (Reset)** control: INPUT type="reset" element
- **Input (Submit)** control: INPUT type="submit" element
- **Input (Text)** control: INPUT type="text" element

Page Class

- Page class represents an .aspx file, also known as a Web Forms page, requested from a server that hosts an ASP.NET Web application.
- Every web page is a custom class that inherits from the `system.web.UI.Page` control.
- Page class acquires a number of properties that our code can use:
 - **IsPostBack** :- This Boolean property indicates whether this is the first time the page is being run (False) or whether the page is being resubmitted in response to a control event, typically with stored view state information (True).
 - **EnableViewState** :- When set to False, this overrides the `EnableViewState` property of the contained controls, thereby ensuring that no controls will maintain state information.

Page Class

- **Application** :- This collection holds information that's shared between all users in the website. For example, we can use the Application collection to count the number of times a page has been visited.
- **Session** :- holds information for a single user, so it can be used in different pages. For example, we can use the Session collection to store the items in the current user's shopping basket on an e-commerce website.

Page Class

- **Cache** :- allows us to store objects that are time-consuming to create so they can be reused in other pages or for other clients. Improves performance of the web pages.
- **Request** :-HttpRequest object that contains information about the current web request.
- **Response** :-HttpResponse object that represents the response ASP.NET will send to the user's browser.

global.asax File

global.asax allows you to write code that runs in response to "system level" events, such as the application starting, a session ending, an application error occurring, without having to try and shoe-horn that code into each and every page of your site.

1. **Application_Start:** Fired when the first instance of the `HttpApplication` class is created. It allows you to create objects that are accessible by all `HttpApplication` instances.
2. **Application_End:** Fired when the last instance of an `HttpApplication` class is destroyed. It's fired only once during an application's lifetime.
3. **Session_Start:** Fired when a new user visits the application Web site.
4. **Session_End:** Fired when a user's session times out, ends, or they leave the application Web site.

global.asax File

5. **Application_BeginRequest:** This event is triggered at the beginning of each HTTP request made to the application. It's useful for performing preprocessing tasks before the actual request processing begins.
6. **Application_AuthenticateRequest:** Fired when the security module has established the current user's identity as valid. At this point, the user's credentials have been validated.
7. **Application_Error:** Fired when an unhandled exception is encountered within the application.

WEB.CONFIG

- Every web application includes a web.config file that configures fundamental settings—everything from the way error messages are shown to the security settings that lock out unwanted visitors.
- 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.
- Visual Studio generates a default web.config file for each project.
- An application can run without a web.config file, however, we cannot debug an application without a web.config file.

.NET TECHNOLOGIES

UNIT 2

- Web Controls are small building blocks of the GUI (Graphical User Interface), which include labels, text box, buttons, etc which provide rich functionality in your pages.
- Allow developers to create dynamic and interactive web pages without writing extensive HTML or JavaScript code.

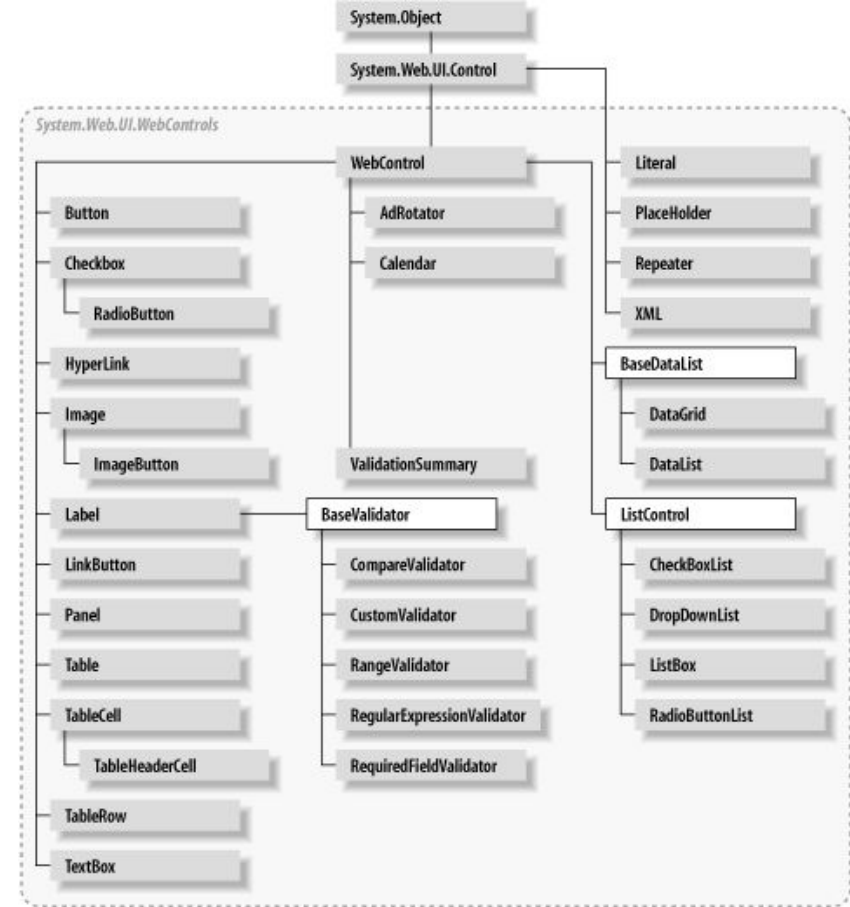
WEB CONTROLS

Advantages of Web Controls over HTML Elements:

1. Abstraction for consistent UI across browsers.
2. Server-side events for easy code execution.
3. Automatic ViewState management.
4. Built-in rich functionality and customization options.
5. Simplified development with reduced reliance on JavaScript

WEB CONTROL CLASSES

- Web control classes are defined in the `System.Web.UI.WebControls` namespace.



KEY

Concrete class

Abstract class

WEB CONTROL BASE CLASS

- Web controls provides the properties, methods, and events that are common to all Web server controls.
- Most web controls begin by inheriting from the `WebControl` base class.
- This class defines the essential functionality for tasks such as data binding and includes some basic properties that you can use with almost any web control.

Properties of WebControl Base Class:

Property	Description
AccessKey	Sets focus on the web control.
BackColor	Sets the background color.
ForeColor	Sets the foreground color (text color).
BorderColor	Sets the border color.
BorderWidth	Specifies the control border width.
BorderStyle	Sets the border style (e.g., Dotted, Solid).
Enabled	Enables or disables user interaction.
EnableViewState	Manages the automatic state management.
Font	Specifies the font for text in the control.
Height/Width	Sets the dimensions of the control.
ID	Specifies the identifier for code interaction.

Properties of WebControl Base Class:

Page	References the containing web page as a Page object.
Parent	Sets the parent control.
TabIndex	Controls the tab order of the control.
ToolTip	Sets the tooltip text displayed on hover.
Visible	Controls visibility; not rendered if false.

Methods of WebControl Base Class:

Method	Description
AddAttributesToRender	Adds HTML attributes and styles for rendering.
ClearChildState	Clears view-state and control-state of child controls.
ClearChildViewState	Clears view-state of child controls.
CreateChildControls	Creates child controls.
DataBind	Binds a data source to control and child controls.
Dispose	Cleans up before releasing from memory.
Focus	Sets input focus to the control.
GetType	Retrieves the Type of the current instance.
OnLoad	Raises the Load event.
OnPreRender	Raises the PreRender event.
OnUnload	Raises the Unload event.
ToString	Returns a string representation of the object.

Events of WebControl Base Class:

Event	Description
DataBinding	Occurs when the control binds to data.
Disposed	Occurs when the control is disposed.
Init	Occurs when the control is initialized.
Load	Occurs when the control is loaded.
PreRender	Occurs before rendering.
Unload	Occurs when the control is unloaded.

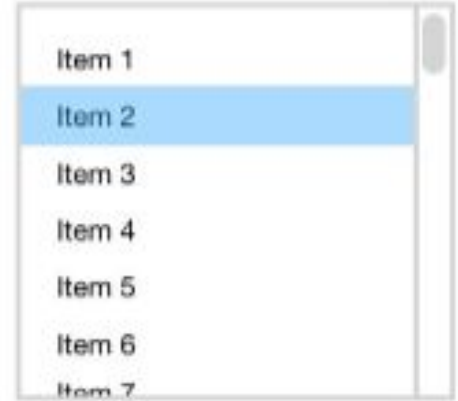
1. The list controls include the
 - ListBox
 - DropDownList
 - CheckBoxList
 - RadioButtonList
 - BulletedList
2. Work same way but are rendered differently
3. All the selectable list controls provide a SelectedIndex property that indicates the selected row as a zero-based index.
4. For example, if the first item in the list is selected, the SelectedIndex will be 0.

LIST CONTROLS

LISTBOX CONTROL

- Represents a user interface element for displaying a list of items, allowing single or multiple item selection.
- The SelectionMode property determines whether single or multiple items can be selected.
- In single selection mode, only one item can be selected.
- In multiple selection mode, users can select multiple items by pressing Ctrl or Shift key.
- Example of creating a ListBox:

Select an item:



Scrollable, Single-Select Listbox

Available items:



Scrollable, Multiselect Listbox

```
<asp:ListBox id="ListBox1" SelectionMode="Single" runat="server">
  <asp:ListItem>Item 1</asp:ListItem>
  <asp:ListItem>Item 2</asp:ListItem>
  <asp:ListItem>Item 3</asp:ListItem>
</asp:ListBox>
```

```
protected void lb1_SelectedIndexChanged(object sender, EventArgs e)
{
    string msg = "";
    foreach (ListItem li in lb1.Items)
    {
        if (li.Selected == true)
        {
            msg += "<BR>" + li.Text + " is selected.";
        }
    }
    Label1.Text = msg;
}
```

LISTBOX CONTROL

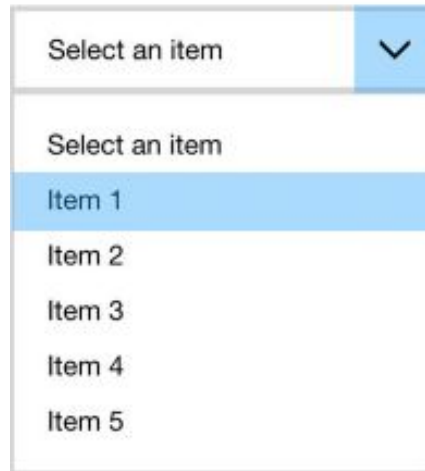
Common Properties of ListBox

Property	Description
Items	Gets the collection of items in the list control.
SelectionMode	This property will set the selection mode as single selection or multiple selection
Rows	This will determine the number of items shows in the list box.
SelectedIndex	Gets or sets the lowest ordinal index of the selected items in the list.
SelectedValue	Gets the value of the selected item in the list control, or selects the item in the list control that contains the specified value.

DROP DOWN LIST

- Allows users to select an item from a predefined list.
- Supports selecting only one item at a time.
- Example of creating a DropDownList:

Select an item:



Select an item

Select an item

Item 1

Item 2

Item 3

Item 4

Item 5


```
<asp:DropDownList ID="DropDownList1" runat="server" >
    <asp:ListItem Value="">Please Select</asp:ListItem>
    <asp:ListItem>item1 </asp:ListItem>
    <asp:ListItem> item2</asp:ListItem>
    <asp:ListItem> item3</asp:ListItem>
    <asp:ListItem> item4</asp:ListItem>
</asp:DropDownList>
```

```
protected void DropDownList1_SelectedIndexChanged(object sender, EventArgs e)
{
    Label1.Text = ddl1.SelectedValue;
}
```

CHECKBOX LIST

- Used when selecting one or more options from several choices.
- Supports selecting multiple items.
- Example of creating a CheckBoxList:

```
<asp:CheckBoxList id="checkboxlist1" runat="server">  
  <asp:ListItem>Item 1</asp:ListItem>  
  <asp:ListItem>Item 2</asp:ListItem>  
  <asp:ListItem>Item 3</asp:ListItem>  
</asp:CheckBoxList>
```

```
protected void CheckBoxList1_SelectedIndexChanged(object sender, EventArgs e)
{
    Label1.Text = "Selected Item(s):<br /><br />";
    for (int i = 0; i < ck1.Items.Count; i++)
    {
        if (ck1.Items[i].Selected)
        {
            Label1.Text += ck1.Items[i].Text + "<br />";
        }
    }
}
```

RADIO BUTTON LIST

- Displays a list of radio buttons.
- Allows selecting only one item.
- Example of creating a RadioButtonList:

```
<asp:RadioButtonList id="RadioButtonList1" runat="server">  
  <asp:ListItem>Item 1</asp:ListItem>  
  <asp:ListItem>Item 2</asp:ListItem>  
  <asp:ListItem>Item 3</asp:ListItem>  
</asp:RadioButtonList>
```

- Dansk
- Nederlands
- English
- Suomi
- Français

```
protected void RadioButtonList1_SelectedIndexChanged(object sender, EventArgs e)  
{  
    Label1.Text = rbl.SelectedValue;  
}
```

RADIO BUTTON LIST

Common properties:

Property	Description
Repeat Layout	Determines whether the radio buttons display in an HTML table.
RepeatColumns	It displays the number of columns of radio buttons.
RepeatDirection	The direction that the radio buttons repeat. By default RepeatDirection value is vertical. Possible values are Horizontal and Vertical.

BULLETED LIST

- Displays items in different styles (unordered or ordered list).
- Supports various bullet styles like Circle, Disc, Square, etc.
- Example:

```
<asp:BulletedList ID="BulletedList1" runat="server"></asp:BulletedList>
```

Property	Description
BulletStyle	To set the style and looks of the bullet list this property is used.
FirstBulletNumber	In an ordered list, this sets the first value. Ex. If you set FirstBulletNumber to 3, the list might read 3,4,5 for Numbered.
DisplayMode	Determines whether the text of each item is rendered as text or a hyperlink.

- Table class is used to build an HTML table.
- Table class is included in `System.Web.UI.Controls` namespace.
- Essentially, the Table control is built out of a hierarchy of objects.
- Each Table object contains one or more TableRow objects.
- In turn, each TableRow object contains one or more TableCell objects.
- Each TableCell object contains other ASP.NET controls or HTML content that displays information.

TABLE CONTROL

**A Sample Table object
(2 Rows, 3 Columns)**

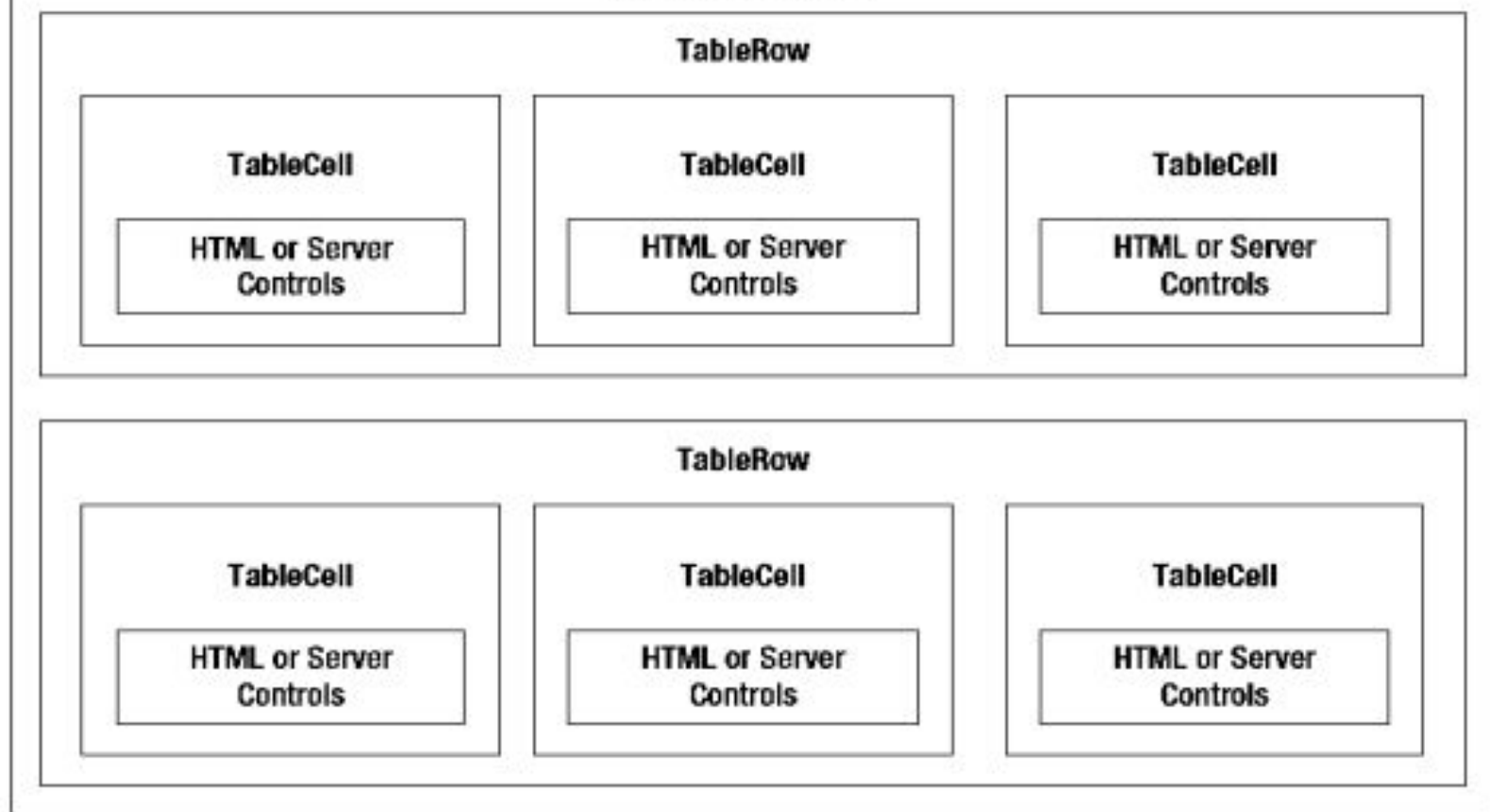


Figure 5.2: Table object

Properties of Table class

Property	Description
Runat	This property state that the web control is a server control. For this purpose, we have to set value of runat to “server”
Rows	This property state group of rows in the table
Caption	This property is used to set title to table
CaptionAlign	This property is used to set alignment of the caption text.
CellPadding	This property is used to state the space between the cell walls and controls in table.
CellSpacing	This property is used to determines distance between cell of tables.
GridLines	This property is used to set gridline format in the table.
HorizontalAlign	Gets or sets the horizontal alignment of the Table control on the page.

Creating table at design time:

```
<asp:Table id="Table1" runat="server" CellPadding="10" GridLines="Both" HorizontalAlign="Center">
  <asp:TableRow>
    <asp:TableCell>
      Row 0, Col 0
    </asp:TableCell>
    <asp:TableCell>
      Row 0, Col 1
    </asp:TableCell>
  </asp:TableRow>
  <asp:TableRow>
    <asp:TableCell>
      Row 1, Col 0
    </asp:TableCell>
    <asp:TableCell>
      Row 1, Col 1
    </asp:TableCell>
  </asp:TableRow>
</asp:Table>
```

WEB CONTROL EVENTS AND AUTOPOSTBACK

- One of the main limitations of HTML server controls is their limited set of useful events—they have exactly two.
- HTML controls that trigger a postback, such as buttons, raise a `ServerClick` event.
- Input controls provide a `ServerChange` event that doesn't actually fire until the page is posted back.

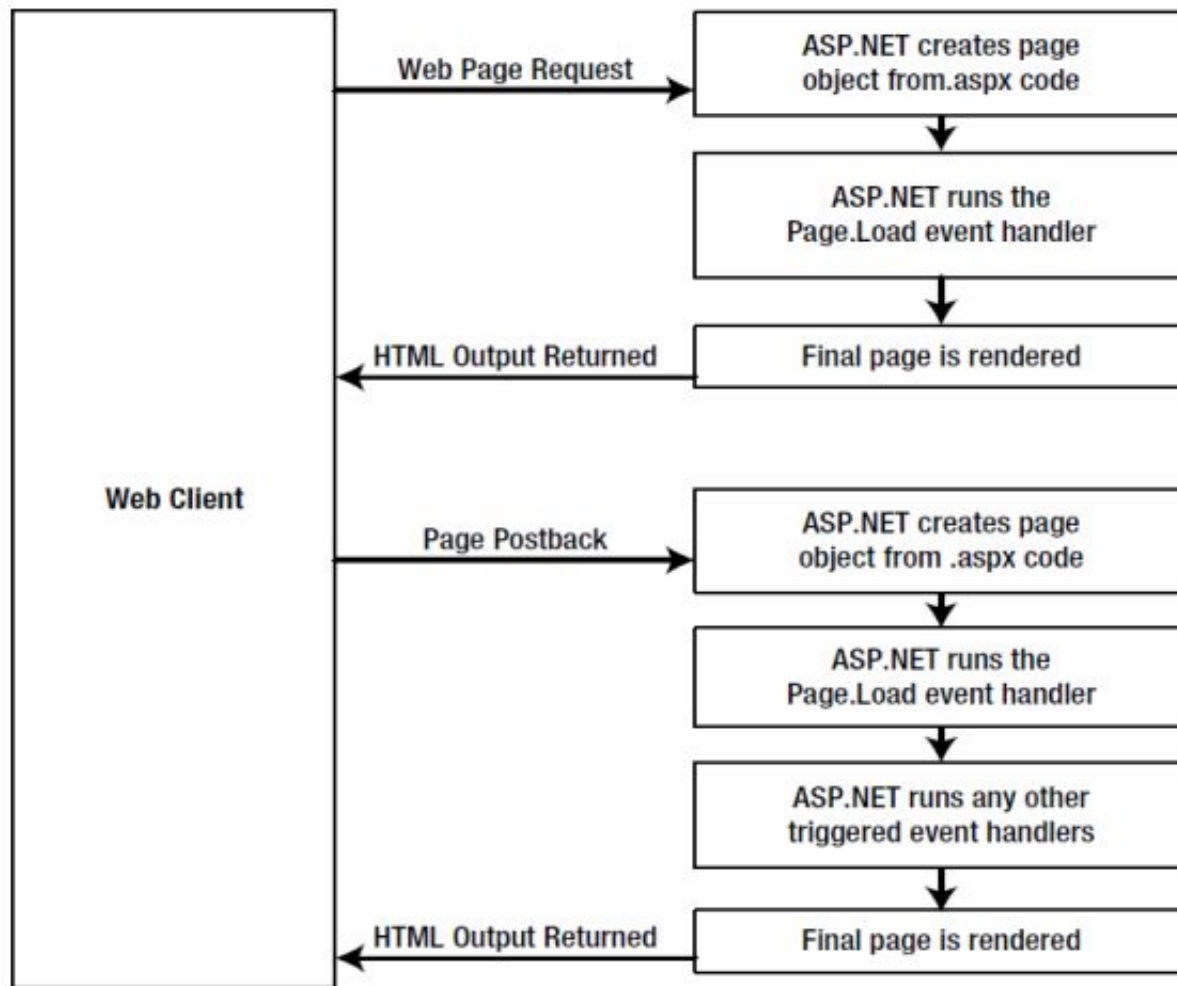


Figure 5.3: The page-processing sequence

The page-processing sequence

- When event occurs on client side some events such as Click event of a button take place immediately, because when clicked, the button post back the page.
- However, other actions do cause events but don't trigger a postback.
- For example, when user chooses a new item in a list or changes the text in text box.
- In these cases, without postback your code has no way to run.

ASP.NET handles this by giving you two options:

1. Wait until the next postback to react to the event. Like to react to SelectedIndexChanged event in a list, when user selects an item in a list, nothing happens immediately. But if user clicks a button to postback the page, two events fire: ButtonClick followed by ListBox.SelectedIndexChanged.

2. To use the automatic postback feature to force a control to post back the page immediately when it detects a specific user action. In this, when the user clicks a new item in the list, the page is posted back, your code executes, and a new version of the page is returned.

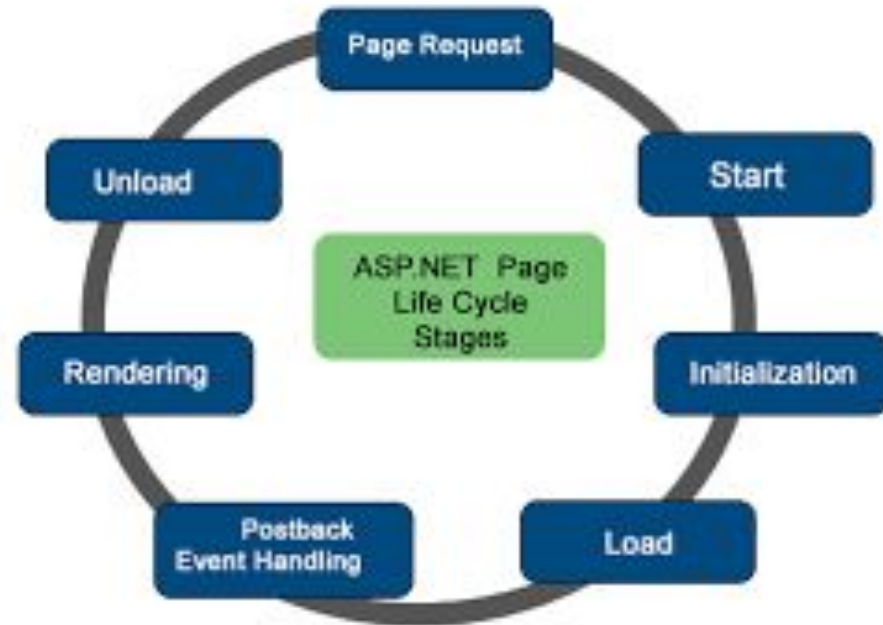
Event	Web Controls That Provide It	Always Posts Back
Click	Button, ImageButton	True
TextChanged	TextBox (fires only after the user changes the focus to another control)	False
CheckedChanged	CheckBox, RadioButton	False
SelectedIndexChanged	DropDownList, ListBox, CheckBoxList, RadioButtonList	False

Table 5.9: Web Control Events

- If you want to capture a change event (such as `TextChanged`, `CheckedChanged`, or `SelectedIndexChanged`) immediately, you need to set the control's `AutoPostBack` property to `true`.
- Depending on the result you want, you could have a page that has some controls that post back automatically and others that don't.

ASP.NET PAGE LIFE CYCLE

1. **Page Request:** The cycle begins when a user requests an ASP.NET page by entering its URL or interacting with a control that triggers a postback.
2. **Start and Initialization:** The page initialization process takes place, including creating controls and loading ViewState. This step occurs on every page request.
3. **Page_Init:** Developers can utilize this event to initialize controls and set their initial properties. However, ViewState is not yet fully loaded at this point.



ASP.NET PAGE LIFE CYCLE

4. Page_Load: Code within this event is executed, often involving data loading, validation, and populating controls. It's a critical point to distinguish between the initial load and postbacks.

5. Control Events: If any controls triggered postbacks, their associated event handlers are executed. This includes actions like button clicks or selection changes.

6. PreRender: The Page.PreRender event fires, and the page is rendered (transformed from a set of objects to an HTML page).

7. Unload: After rendering, cleanup tasks and resource release are performed. This is the last opportunity to interact with controls and the page. The Page_Unload is final event in the cycle, used for cleanup and finalization. The page instance is released after this point.

STATE MANAGEMENT

- State management in ASP.NET refers to the mechanisms and techniques used to maintain the state (data and values) of a web application across multiple requests and page transitions.
- ASP.NET offers various ways to manage state, allowing developers to preserve user input, control states, and other relevant data between different interactions with the application.

VIEW STATE

- ViewState is a important client side state management technique.
- ViewState is used to store user data on page at the time of post back of web page.
- ViewState does not hold the controls, it holds the values of controls.
- It does not restore the value to control after page post back.
- ViewState can hold the value on single web page, if we go to other page using response.redirect then ViewState will be null.
- When we require value of page variable to be maintained during page postback, we can use View state to store those value.
- “EnableViewState” property is used for both Page Level and Server contact level to manage the view state.

```
<asp:TextBox runat="server" ID="NameField" />
<asp:Button runat="server" ID="SubmitForm" OnClick="SubmitForm_Click" Text="Submit" />
<asp:Label runat="server" ID="NameLabel" />
```

```
protected void Page_Load(object sender, EventArgs e)
{
    if (ViewState["NameOfUser"] != null)
        NameLabel.Text = ViewState["NameOfUser"].ToString();
    else
        NameLabel.Text = "Not set yet...";
}
```

```
protected void SubmitForm_Click(object sender, EventArgs e)
{
    ViewState["NameOfUser"] = NameField.Text;
    NameLabel.Text = NameField.Text;
}
```

VIEW STATE

Limitation of view state

1. It increases the page size
2. Can impact performance
3. Can potentially expose data in the rendered HTML.
4. Viewstate can be used only with single page.
5. It is storing the information of an hidden field, so it can be seen in source code in browser, hence it is not secure way.

QUERY STRING

1. A common approach is to pass information by using a query string in the URL.
2. This approach is commonly found in search engines.
3. For example, if you perform a search on the Google website, you'll be redirected to a new URL that incorporates your search parameters
4. There is a limitation of length of query string. So query string cannot be used to send very large data.
5. Query string are visible to the user, so it should not be used to send sensitive information such as username, password without encryption.
6. Request object of QueryString property is used to retrieve the query string.

QUERY STRING

Usage and Structure:

- The query string is a part of the URL that comes after the question mark (?).
- It's composed of one or more key-value pairs separated by ampersands (&).
- Query strings are commonly used to transmit data from one page to another.

Example:

<http://www.google.ca/search?q=organic+gardening>

- The query string is the portion of the URL after the question mark.
- In this case, it defines a single variable named q, which contains the string organic+gardening.

QUERY STRING

- **Advantages:**

- Simple and easy to use.
- Works across different platforms and technologies.

- **Disadvantages:**

- Limited data capacity due to URL length restrictions.
- Exposes data in the URL, potentially compromising sensitive information.

QUERY STRING

```
// Forward the user to the information page,  
// with the query string data.  
string url = "QueryStringRecipient.aspx?";  
url += "Item=" + lstItems.SelectedItem.Text + "&";  
url += "Mode=" + chkDetails.Checked.ToString();  
Response.Redirect(url);
```

SenderPage



ReceiverPage



```
protected void Page_Load(Object sender, EventArgs e)  
{  
    lblInfo.Text = "Item: " + Request.QueryString["Item"];  
    lblInfo.Text+="Show Full Record: ";  
    lblInfo.Text += Request.QueryString["Mode"];  
}
```


COOKIES

1. Cookies are small pieces of data that are stored on the client's machine to retain information between requests.
2. They are often used to store user preferences, login sessions, and other stateful information.
3. In ASP.NET, cookies can be managed using the `HttpCookie` class.
4. Cookies have limitations, including a size limit and the fact that they are stored on the client's machine, making them less secure for sensitive information.

COOKIES

1. Create a webpage with language selection options (Default.aspx):

```
<!DOCTYPE html>
<html xmlns="<http://www.w3.org/1999/xhtml>">
<head runat="server">
  <title>Language Preference</title>
</head>
<body>
  <form id="form1" runat="server">
    <div>
      <asp:DropDownList ID="ddlLanguages" runat="server">
        <asp:ListItem Text="English" Value="en"></asp:ListItem>
        <asp:ListItem Text="French" Value="fr"></asp:ListItem>
        <asp:ListItem Text="Spanish" Value="es"></asp:ListItem>
      </asp:DropDownList>
      <asp:Button ID="btnSetLanguage" runat="server" Text="Set Language" OnClick="btnSetLanguage_Click" />
    </div>
  </form>
</body>
</html>
```

COOKIES

1. Add code-behind logic to set and read cookies (Default.aspx.cs):

```
using System;

public partial class Default : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        if (!IsPostBack)
        {
            if (Request.Cookies["LanguagePreference"] != null)
            {
                string language = Request.Cookies["LanguagePreference"].Value;
                ddlLanguages.SelectedValue = language;
            }
        }

        protected void btnSetLanguage_Click(object sender, EventArgs e)
        {
            HttpCookie cookie = new HttpCookie("LanguagePreference");
            cookie.Value = ddlLanguages.SelectedValue;
            cookie.Expires = DateTime.Now.AddDays(30); // Cookie will expire in 30 days
            Response.Cookies.Add(cookie);
        }
    }
}
```

SESSION STATE

1. Session state is a server-side state management technique in ASP.NET that allows you to store and retrieve user-specific data across multiple requests.
2. It provides a way to maintain state information for individual users during their entire session on a website.
3. This is particularly useful for scenarios where you need to store user specific data like shopping cart contents, user preferences, or login sessions.
4. Session state uses a unique session identifier (usually stored in a cookie or as part of the URL) to associate a user with their session data on the server.
5. The data is stored on the server's memory, a database, or an external state server, depending on the session state mode you choose.

SESSION STATE

How It Works:

1. **Session Tracking:** When a user starts using your website, ASP.NET gives them a unique session ID. It's like a special number that's unique to them. This ID can be stored as a cookie in their web browser or managed in the URL.
2. **Using Session State:** Think of session state like a locker on the server. You can put things inside (like data) and take them out later. You can use the `Session` object to do this. For example, you can put some data like a user's name in the session like this:

```
Session["UserName"] = "John";
```

And later, you can get it back:

```
string userName = (string)Session["UserName"];
```

SESSION STATE

3. Benefits:

- **Remembering Data:** You can remember data for a user across different pages. For instance, you can remember their preferences or shopping cart items.
- **Unique to Users:** Each user gets their own session, so the data you put in there is separate for each user.
- **Security:** Since session data stays on the server, it's more secure than keeping it on the user's side.

Limitations:

- **Memory Usage:** If you put too much stuff in session state, it can use up a lot of server memory.
- **Expiry:** Session data doesn't last forever. It has a timeout, so if a user doesn't do anything on your website for a while, their session might expire and they lose the data you stored.

SESSION STATE

These are properties and methods provided by ASP.NET's `HttpSessionState` class, which is used for managing session state in web applications. They allow you to interact with the session data stored on the server for each user

1. **Count:** Gives you the number of items currently stored in the session for a specific user's session.
2. **IsCookieless:** Tells you whether the session is being tracked using cookies or modified URLs.
3. **IsNewSession:** Indicates whether the session was newly created for the current request. If there's no session data, a new session is created.
4. **Keys:** Provides a collection of keys (names) used to access session items.

SESSION STATE

5. **Mode**: Specifies how session state information is stored, based on web.config settings.
6. **SessionID**: Unique identifier for the current user's session.
7. **Timeout**: Determines how long the session will last without any activity before it's abandoned.
8. **Abandon()**: Ends the current session and releases associated resources.
9. **Clear()**: Removes all session items but doesn't end the session itself.

Provide Following Details

Email

Password

Login

```
protected void login_Click(object sender, EventArgs e)
{
    if (password.Text == "qwe123")
    {
        // Storing email to Session variable
        Session["email"] = email.Text;
    }
    // Checking Session variable is not empty
    if (Session["email"] != null)
    {
        // Displaying stored email
        Label3.Text = "This email is stored to the session.";
        Label4.Text = Session["email"].ToString();
    }
}
```

CONFIGURING SESSION STATE

You configure session state through the web.config file for your current application.

```
<configuration>
...
<system.web>
...
<sessionState
  cookieless="UseCookies"
  cookieName="ASP.NET_SessionId"
  regenerateExpiredSessionId="false"
  timeout="20"
  mode="InProc"
  stateConnectionString="tcpip=127.0.0.1:42424"
  stateNetworkTimeout="10"
  sqlConnectionString="data source=127.0.0.1;Integrated Security=SSPI"
  sqlCommandTimeout="30"
  allowCustomSqlDatabase="false"
  customProvider=""
  compressionEnabled="false"
/>
</system.web>
</configuration>
```

APPLICATION STATE

1. Application state is a concept in ASP.NET that allows you to store data that is shared across all users and sessions of an application.
2. Unlike session state which is user-specific, application state is shared among all users and is accessible throughout the entire application.
3. Example:

```
// Storing a value in application state
Application["TotalVisitors"] = 0;

// Incrementing the value on each page visit
int currentVisitors = (int)Application["TotalVisitors"];
currentVisitors++;
Application["TotalVisitors"] = currentVisitors;

// Displaying the total number of visitors
LabelTotalVisitors.Text = "Total Visitors: " + Application["TotalVisitors"].ToString();
```

VALIDATIONS

- Validation controls in ASP.NET are used to validate user input on web forms to ensure that the data entered by users meets specific criteria or formats.
- Validation controls can be applied to various types of user input controls such as textboxes, dropdown lists, checkboxes, and more.
- 2 types of validations:
 - Client side validation
 - Server side validation

- **Server Side Validation**

Server-side validation is the process of validating user input and data on the server after it has been submitted by the user. Unlike client-side validation, which occurs in the user's browser using JavaScript, server-side validation takes place on the web server after the data has been posted back from the client.

- **Client Side Validation**

Client-side validation is the process of validating user input and data directly within the user's web browser using client-side scripting languages like JavaScript. This type of validation occurs before the data is submitted to the server, providing immediate feedback to users and reducing the need for round-trips to the server for validation.

VALIDATION CONTROLS

The `BaseValidator` class is the foundation for validation controls in ASP.NET. Some key properties of the `BaseValidator` class

Members	Description
<code>ControlToValidate</code>	Indicates the input control to validate.
<code>Display</code>	Indicates how the error message is shown.
<code>EnableClientScript</code>	Indicates whether client side validation will take.
<code>Enabled</code>	Enables or disables the validator.
<code>ErrorMessage</code>	Indicates error string.
<code>Text</code>	Error text to be shown if validation fails.
<code>IsValid</code>	Indicates whether the value of the control is valid.
<code>SetFocusOnError</code>	It indicates whether in case of an invalid control, the focus should switch to the related input control.
<code>ValidationGroup</code>	The logical group of multiple validators, where this control belongs.
<code>Validate()</code>	This method revalidates the control and updates the <code>IsValid</code> property.

VALIDATION CONTROL

Here are some common validation controls and their purposes:

1. **RequiredFieldValidator**: Ensures that a user provides input in a specific control, preventing the submission of empty or blank values.

```
<asp:RequiredFieldValidator ID="rf1" runat="server" ErrorMessage="RequiredFieldValidator" ControlToValidate="lbl1">  
</asp:RequiredFieldValidator>
```

2. **RegularExpressionValidator**: Validates input against a specified regular expression pattern. Useful for enforcing specific formats like email addresses, phone numbers, or zip codes.

```
<asp:RegularExpressionValidator ID="re1" runat="server" ErrorMessage="RegularExpressionValidator"  
    ControlToValidate="lb1" ValidationExpression=".+@.+">  
</asp:RegularExpressionValidator>
```

3. **RangeValidator**: Checks whether the input falls within a specified range of values (numeric or date range).

```
<asp:RangeValidator ID="rv1" runat="server" ErrorMessage="RangeValidator" ControlToValidate="txt1"  
    MaximumValue="1" MinimumValue="10">  
</asp:RangeValidator>
```

VALIDATION CONTROL

- CompareValidator:** Compares the input in one control to the input in another control or a constant value. Useful for password confirmation or checking if two fields have the same value.

```
<asp:CompareValidator ID="cv1" runat="server" ErrorMessage="CompareValidator" ControlToValidate="txtpass"
    ValueToCompare="txtconf_pass">
</asp:CompareValidator>
```

- CustomValidator:** Allows you to define custom validation logic by specifying a client-side and server-side validation function.

```
<asp:CustomValidator ID="cust" runat="server" ErrorMessage="CustomValidator" ControlToValidate="txt"
    OnServerValidate="cust_ServerValidate">
</asp:CustomValidator>
```

- ValidationSummary:** Provides a summary of all validation errors on the page. It's often placed near the top of the form to give users an overview of any issues.

```
<asp:ValidationSummary ID="vs1" runat="server" ValidationGroup="txtrange" />
```

MANUAL VALIDATION

1. Manual validation in ASP.NET refers to the process of validating user input without relying on built-in validation controls.
2. Instead of using validation controls like `RequiredFieldValidator`, `RegularExpressionValidator`, etc., you perform validation using code-behind logic.
3. This gives you more control over the validation process and allows you to customize the validation logic as needed.

```
<asp:TextBox ID="txtNumber" runat="server" />  
<br />  
<asp:Button ID="btnValidate" runat="server" Text="Validate" OnClick="btnValidate_Click" />  
<br />  
<asp:Label ID="lblResult" runat="server" />
```

MANUAL VALIDATION

```
protected void btnValidate_Click(object sender, EventArgs e)
{
    string input = txtNumber.Text;
    if (IsValidNumber(input))
    {
        lblResult.Text = "Valid number entered!";
        lblResult.ForeColor = System.Drawing.Color.Green;
    }
    else
    {
        lblResult.Text = "Invalid number format!";
        lblResult.ForeColor = System.Drawing.Color.Red;
    }
}

private bool IsValidNumber(string input)
{
    int number;
    return int.TryParse(input, out number);
}
```

VALIDATION WITH REGULAR EXPRESSIONS

1. Validation with regular expressions in ASP.NET allows you to define complex patterns that user input must match.
2. Regular expression validation is useful when you need to ensure that input follows a specific format, such as email addresses, phone numbers, passwords, and more.
3. A phone number must be a set number of digits, an e-mail address must include exactly one @ character (with text on either side)
4. ASP.NET provides the `RegularExpressionValidator` control for this purpose.

VALIDATION WITH REGULAR EXPRESSIONS

Character	Description
*	Zero or more occurrences of the previous character or subexpression. For example, 7^*8 matches 7778 or just 8.
+	One or more occurrences of the previous character or subexpression. For example, 7^+8 matches 7778 but not 8.
()	Groups a subexpression that will be treated as a single element. For example, $(78)^+$ matches 78 and 787878.
{m,n}	The previous character (or subexpression) can occur from m to n times. For example, $A\{1,3\}$ matches A, AA, or AAA.
	Either of two matches. For example, $8 6$ matches 8 or 6
[]	Matches one character in a range of valid characters. For example, $[A-C]$ matches A, B, or C.

VALIDATION WITH REGULAR EXPRESSIONS

<code>[^]</code>	Matches one character in a range of valid characters. For example, <code>[A-C]</code> matches A, B, or C.
<code>.</code>	Any character except a newline. For example, <code>.here</code> matches where and there.
<code>\s</code>	Any whitespace character (such as a tab or space).
<code>\S</code>	Any nonwhitespace character
<code>\d</code>	Any digit character
<code>\D</code>	Any character that isn't a digit.
<code>\w</code>	Any "word" character (letter, number, or underscore).
<code>\W</code>	Any character that isn't a "word" character (letter, number, or underscore).

VALIDATION WITH REGULAR EXPRESSIONS

Commonly Used Regular Expressions

Content	Regular Expression	Description
E-mail address*	<code>\S+@\S+\.\S+</code>	Check for an at (@) sign and dot (.) and allow nonwhitespace characters only.
Password	<code>\w+</code>	Any sequence of one or more word characters (letter, space, or underscore).
Specific-length password	<code>\w{4,10}</code>	A password that must be at least four characters long but no longer than ten characters.

VALIDATION WITH REGULAR EXPRESSIONS

Advanced password	<code>[a-zA-Z]\w{3,9}</code>	As with the specific-length password, this regular expression will allow four to ten total characters. The twist is that the first character must fall in the range of a–z or A–Z
Another advanced password	<code>[a-zA-Z]\w*\d+\w*</code>	This password starts with a letter character, followed by zero or more word characters, one or more digits, and then zero or more word characters
Limited-length field	<code>\S{4,10}</code>	Like the password example, this allows four to ten characters, but it allows special characters

VALIDATION WITH REGULAR EXPRESSIONS

<p>US Social Security number</p>	<pre>\d{3}-\d{2}-\d{4}</pre>	<p>A sequence of three, two, and then four digits, with each group separated by a dash. You could use a similar pattern when requiring a phone number.</p>
----------------------------------	------------------------------	--

RICH CONTROLS

- Rich controls are web controls that model complex user interface elements.
- Provides object model that has complex HTML representation and also client-side JavaScript
- A typical rich control can be programmed as a single object but renders itself using a complex sequence of HTML elements.
- Rich controls can also react to user actions (such as a mouse click on a specific region of the control) and raise more-meaningful events that your code can respond to on the web server.
- Rich controls, also known as Web server controls or ASP.NET controls

CALENDAR CONTROL

1. The Calendar control in ASP.NET is a rich control that provides an interactive calendar interface for selecting dates.
2. The Calendar control is part of the `System.Web.UI.WebControls` namespace and offers various features for date selection and customization.
3. To use the Calendar control, you can simply add it to your ASP.NET page using the `<asp:Calendar>` tag.
4. For example:

```
<asp:Calendar ID="Calendar1" runat="server"></asp:Calendar>
```

CALENDAR CONTROL - FORMATTING THE CALENDAR

Property	Description
DayHeaderStyle	The style for the section of the Calendar that displays the days of the week (as column headers).
DayStyle	The default style for the dates in the current month.
NextPrevStyle	The style for the navigation controls in the title section that move from month to month.
OtherMonthDayStyle	The style for the dates that aren't in the currently displayed month.
SelectedDayStyle	The style for the selected dates on the calendar.
SelectorStyle	The style for the week and month date selection controls.
TitleStyle	The style for the title section.
TodayDayStyle	The style for the date designated as today (TodaysDate property of Calendar control).
WeekendDayStyle	The style for dates that fall on the weekend.

CALENDAR CONTROL - CALENDAR DAY PROPERTIES

Property	Description
Date	The DateTime object representing this date.
IsWeekend	True if this date falls on a Saturday or Sunday.
IsToday	True if this value matches Calendar.TodaysDate property.
IsOtherMonth	True if this date isn't in the current month but displayed.
IsSelectable	Configure whether the user can select this day.

ADROTATOR CONTROL

1. The AdRotator control in ASP.NET is used for displaying a sequence of advertisements (ads) on a webpage.
2. It is commonly used to rotate different banner ads, images, or HTML content in a cyclic manner.
3. The control makes it easier to manage and display advertisements on a website without manually updating the content each time.
4. Here's how the AdRotator control works:

```
<asp:AdRotator ID="adRotator" runat="server"
  AdvertisementFile="~/App_Data/Ads.xml"
  Target="_blank"
  Width="300"
  Height="200" />
```

web1.aspx

In this example, the AdvertisementFile property specifies the XML file containing the ad information. The Target property indicates that clicking on an ad will open the link in a new browser tab. The Width and Height properties set the dimensions of the displayed ad.

Target	Description
_blank	The link opens a new unframed window.
_parent	The link opens in the parent of the current frame.
_self	The link opens in the current frame.
_top	The link opens in the topmost frame of the current window

```
<Advertisements>
  <Ad>
    <ImageUrl>~/Images/ad1.jpg</ImageUrl>
    <NavigateUrl><https://www.example.com/ad1></NavigateUrl>
    <AlternateText>Ad 1</AlternateText>
  </Ad>
  <Ad>
    <ImageUrl>~/Images/ad2.jpg</ImageUrl>
    <NavigateUrl><https://www.example.com/ad2></NavigateUrl>
    <AlternateText>Ad 2</AlternateText>
  </Ad>
</Advertisements>
```

Ads.xml

In this XML, each <Ad> element represents an advertisement with properties such as ImageUrl , NavigateUrl , and AlternateText .

The AdRotator control helps streamline the process of displaying rotating ads on a webpage, making it easier to manage and update advertisement content without modifying the HTML code.

- **ImageUrl:** This property specifies the image that will be displayed as an advertisement. It can be either a relative link to a local file (such as "~/Images/ad1.jpg") or a fully qualified Internet URL.
- **NavigateUrl:** This property defines the URL to which the user will be redirected if they click on the advertisement. Similar to the `ImageUrl`, this can be a relative or fully qualified URL.
- **AlternateText:** The text provided in this property will be displayed if the image cannot be loaded, acting as an alternative description. It is also used as a tooltip in some modern browsers when the user hovers over the image.
- **Impressions:** This numeric property determines the frequency at which an advertisement will appear relative to other ads. For example, if you set an advertisement's Impressions to 10 and another ad's Impressions to 5, the first ad will be shown twice as often as the second on average.
- **Keyword:** This property allows you to assign a keyword to advertisements. It can be used for filtering purposes, allowing you to group and display ads based on certain keywords. For instance, you might have different sets of ads for categories like "Retail," "Technology," etc.

MULTIVIEW CONTROL

1. The MultiView control in ASP.NET provides a way to display different sets of content in separate views and allows users to switch between these views without requiring separate postbacks or page reloads.
2. It's commonly used to create tabbed interfaces or step-by-step wizards where each step is presented in a different view

3. Syntax

```
<asp:MultiView ID="MultiView1" runat="server">  
    <asp:View ID="View1" runat="server">...</asp:View>  
    <asp:View ID="View2" runat="server">...</asp:View>  
    <asp:View ID="View3" runat="server">...</asp:View>  
</asp:MultiView>
```

Key Features of the `MultiView` Control:

- **Views:** The `MultiView` control contains a collection of `View` controls. Each `View` represents a different content segment that can be displayed.
- **Active View:** Only one `View` can be active at a time. The content of the active view is shown, while the content of other views remains hidden.
- **Switching Views:** The active view can be changed programmatically in response to user actions, such as clicking buttons or links. This allows you to create interactive user interfaces.
- **User-Friendly:** The `MultiView` control provides a smooth and user-friendly way to present different sections of content without causing full page reloads.


```
<asp:MultiView ID="multiView" runat="server">
  <asp:View ID="view1" runat="server">
    <!-- Content for View 1 -->
    <h2>Welcome to View 1</h2>
  </asp:View>
  <asp:View ID="view2" runat="server">
    <!-- Content for View 2 -->
    <h2>Welcome to View 2</h2>
    <p>This is some content in View 2.</p>
  </asp:View>
</asp:MultiView>
```

```
<asp:Button ID="btnSwitchToView1" runat="server" Text="Switch to View 1" OnClick="btnSwitchToView1_Click" />
<asp:Button ID="btnSwitchToView2" runat="server" Text="Switch to View 2" OnClick="btnSwitchToView2_Click" />
```

In this example, the MultiView control contains two views. The user can switch between these views by clicking the buttons. The code-behind sets the initial view and handles the button clicks to change the active view accordingly

Code-Behind (C#):

```
protected void Page_Load(object sender, EventArgs e)
{
    if (!IsPostBack)
    {
        // Set initial view
        multiView.ActiveViewIndex = 0; // Show View 1 by default
    }
}

protected void btnSwitchToView1_Click(object sender, EventArgs e)
{
    multiView.ActiveViewIndex = 0; // Switch to View 1
}

protected void btnSwitchToView2_Click(object sender, EventArgs e)
{
    multiView.ActiveViewIndex = 1; // Switch to View 2
}
```

The MultiView control allows you to switch between different views on a page. To control this switching, you can use command names with buttons. Here are the command names and their actions:

- **PrevView:** Moves to the previous view.
- **NextView:** Moves to the next view.
- **SwitchViewByID:** Moves to a specific view based on its ID.
- **SwitchViewByIndex:** Moves to a specific view based on its index.

MASTERPAGES IN ASP.NET

1. What are MasterPages?

- MasterPages in ASP.NET serve as templates defining the common structure and layout of a web application.
- They provide uniformity across multiple pages while accommodating varied content.
- Master pages have a different file extension (.master instead of .aspx) and they can't be viewed directly by a browser.
- Instead, master pages must be used by other pages, which are known as content pages. Essentially, the master page defines the page structure and the common ingredients.
- A single master page might define the layout for the entire site.

MASTERPAGES IN ASP.NET

2. **Benefits of MasterPages:**

1. **Consistency:** Maintain a consistent look and feel throughout the application.
2. **Ease of Maintenance:** Changes in the MasterPage propagate to all associated pages.
3. **Separation of Concerns:** Designers and developers work independently on layout and content.
4. **Navigation:** Shared elements like headers and menus can be placed in the MasterPage.

How to Use MasterPages:

Step 1: Create a MasterPage

1. Add a new Master Page to your project.
2. Design the layout with HTML and controls, using ContentPlaceHolder to indicate dynamic content areas.

Example - `SiteTemplate.master`:

```
<%@ Master Language="C#" CodeFile="SiteTemplate.master.cs" Inherits="SiteTemplate" %>
<html>
<head runat="server">
    <title>My Master Page</title>
</head>
<body>
    <div id="header">Header Content</div>
    <asp:ContentPlaceHolder ID="MainContent" runat="server"></asp:ContentPlaceHolder>
    <div id="footer">Footer Content</div>
</body>
</html>
```


How to Use MasterPages:

Step 2: Create a Content Page

1. Add a new Web Form (Content Page).
2. Set `MasterPageFile` in the Page directive to link to the MasterPage.

Example - `ContentPage.aspx` :

```
<%@ Page Language="C#" MasterPageFile="~/SiteTemplate.master" Inherits="ContentPage" %>
<asp:Content ID="Content1" ContentPlaceHolderID="MainContent" runat="server">
    <h1>Welcome to My Content Page</h1>
    <p>This is a sample content.</p>
</asp:Content>
```

- When you create a master page, you're building something that looks much like an ordinary ASP.NET web form.
- The key difference is that, although web forms start with the Page directive, a master page starts with a Master directive that specifies the same information.
- The ContentPlaceHolder is the portion of the master page that a content page can change. Or, to look at it another way, everything else that's set in the master page is unchangeable in a content page.
- When you create a content page, ASP.NET links your page to the master page by adding an attribute to the Page directive.
- This attribute, named MasterPageFile, indicates the associated master page.

- ASP.NET provides various site-navigation features which gives a consistent way for visitors to navigate the site.
- These features are
 - Site Maps
 - URL Mapping and Routing
 - SiteMapPath.

WEBSITE NAVIGATION

SITEMAPS

Site maps are used to define a website's navigation structure, making it easier for users to navigate between pages.

Components of ASP.NET Navigation

1. **Site Map Definition:** Site maps define the organization of web pages using XML elements.
2. **Data Source Control (SiteMapDataSource):** This control reads the site map file and converts it into an object model.
3. **Navigation Controls:** These controls utilize the site map information to create navigation menus, breadcrumb links, and more

SITEMAPS

Creating Site Maps - Key Rules

Rule 1: Site Maps Begin with the <siteMap> Element

- Every Web.sitemap file starts with the <siteMap> element.
- The xmlns attribute is essential for ASP.NET recognition.

```
<siteMap xmlns="http://schemas.microsoft.com/AspNet/SiteMap-File-1.0" >
```

Rule 2: Each Page Is Represented by a <siteMapNode> Element

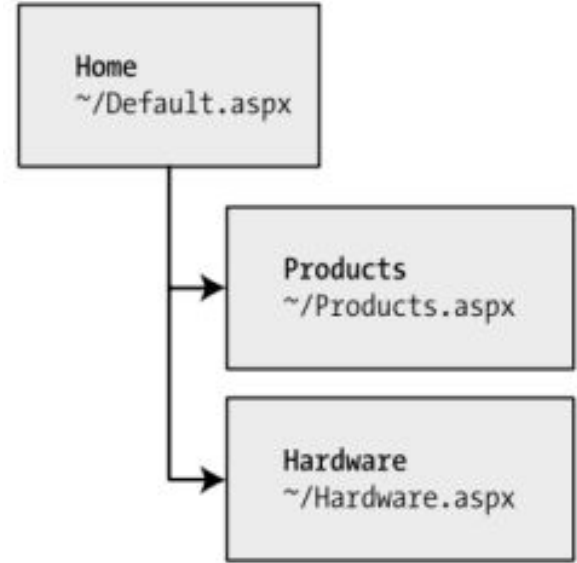
- Pages are represented using the element, containing attributes for title, description, and URL

```
<siteMapNode url="~/Home.aspx" title="Home" description="Home" />
```

Creating Site Maps - Key Rules

Rule 3: A <siteMapNode> Element Can Contain Other <siteMapNode> Elements

- Nodes can be nested to create page groups and subgroups within the site map.



```
<siteMapNode url="~/Home.aspx" title="Home" description="Home">
  <siteMapNode url="~/Products.aspx" title="Products" description="Products" />
  <siteMapNode url="~/Hardware.aspx" title="Hardware" description="Hardware" />
</siteMapNode>
```


Creating Site Maps - Key Rules

Rule 4: Every Site Map Begins with a Single <siteMapNode>

- A site map must have a single root <siteMapNode> that contains all other nodes.
- That means the following is not a valid site map, because it contains two top-level nodes:

```
<siteMapNode url="~/Products.aspx" title="Products" description="Products" />  
<siteMapNode url="~/Hardware.aspx" title="Hardware" description="Hardware" />
```

- The following site map is valid, because it has a single top-level node (Home), which contains two more nodes:

```
<siteMapNode url="~/Home.aspx" title="Home" description="Home">  
  <siteMapNode url="~/Products.aspx" title="Products" description="Products" />  
  <siteMapNode url="~/Hardware.aspx" title="Hardware" description="Hardware" />  
</siteMapNode>
```

Creating Site Maps - Key Rules

Rule 5: Duplicate URLs Are Not Allowed

- Each site map node must have a unique URL.
- Similar URLs with different query string arguments are allowed.
- These two nodes are acceptable, even though they lead to the same page (products.aspx), because the two URLs have different query string arguments at the end.

```
<siteMapNode url="" title="Products" description="Products">  
  <siteMapNode url="~/Products.aspx?stock=1" title="In Stock" description="Products that are available" />  
  <siteMapNode url="~/Products.aspx?stock=0" title="Not In Stock" description="Products that are on order" />  
</siteMapNode>
```

URL MAPPING

URL mapping in ASP.NET allows you to associate multiple URLs with the same page, offering a way to simplify and enhance the user experience. Here's a breakdown of how URL mapping works:

Purpose of URL Mapping:

- Use multiple URLs to lead to a single page.
- Provide shorter, user-friendly, or meaningful URLs while maintaining a single-page logic.

URL MAPPING

Concept of URL Mapping:

- Map a request URL to a different URL using rules defined in the web.config file.
- Mapping rules are applied before any other processing occurs.
- Requested URLs must use a file type extension that's mapped to ASP.NET for the mapping to work

URL Mapping Configuration:

- URL mappings are defined in the <urlMappings> section of the web.config file.
- Each mapping includes the original request URL and the mapped destination URL.

URL MAPPING

```
<configuration>
  <system.web>
    <urlMappings enabled="true">
      <add url="-/category.aspx" mappedUrl="-/default.aspx?category=default" />
      <add url="-/software.aspx" mappedUrl="-/default.aspx?category=software" />
    </urlMappings>
    <!-- Other configuration settings -->
  </system.web>
</configuration>
```

Matching and Redirection:

- For a match to occur, the submitted browser URL must closely match the URL specified in the web.config file.
- Redirection takes place in a way similar to `Server.Transfer()` , without a round-trip.

URL MAPPING

- The URL in the browser will still show the original request URL, not the redirected page's URL.

Benefits:

- Simplifies URLs and makes them more user-friendly.
- Enables you to organize and structure URLs logically.
- Provides a cleaner and consistent user experience.

URL ROUTING

- Unlike URL mapping, URL routing doesn't take place in the `web.config` file. Instead, it's implemented using code.

1. Purpose of URL Routing:

- Create user-friendly and search engine optimized URLs.
- Organize URLs based on a logical and hierarchical structure.
- Improve the overall user experience and navigation.

2. Implementing URL Routing:

- URL routing is commonly set up in the `Application_Start` event of the `Global.asax` file.
- The `RouteTable.Routes` collection is used to define routes using the `MapPageRoute` method.
- A route consists of a unique name, a URL pattern with placeholders, and the target page or handler.


```
protected void Application_Start(object sender, EventArgs e)
{
    RouteTable.Routes.MapPageRoute("product-details",
        "product/{productID}", "~/productInfo.aspx");

    RouteTable.Routes.MapPageRoute("products-in-category",
        "products/category/{categoryID}", "~/products.aspx");
}
```

In this example, two routes are defined: one for displaying product details and another for listing products in a specific category

Benefits of URL Routing:

- **Improved SEO:** Search engines favor descriptive URLs, resulting in better search engine rankings.
- **User-Friendly:** Users find it easier to understand and remember meaningful URLs.
- **Flexibility:** Dynamic placeholders in URLs allow for parameterized pages.
- **Centralized Configuration:** Routes can be managed in one place, improving maintenance.

SITEMAPPATH CONTROL

- The SiteMapPath control is an ASP.NET control that displays a hierarchical representation of a sitemap.
- The sitemap is a collection of pages that are organized in a tree-like structure. The SiteMapPath control
- can be used to display the sitemap in a variety of ways, including as a list, a table, or a navigation bar.
- The SiteMapPath control is a server control, which means that it is processed on the server before it is rendered to the client. This allows the SiteMapPath control to access the sitemap and dynamically generate the navigation links.

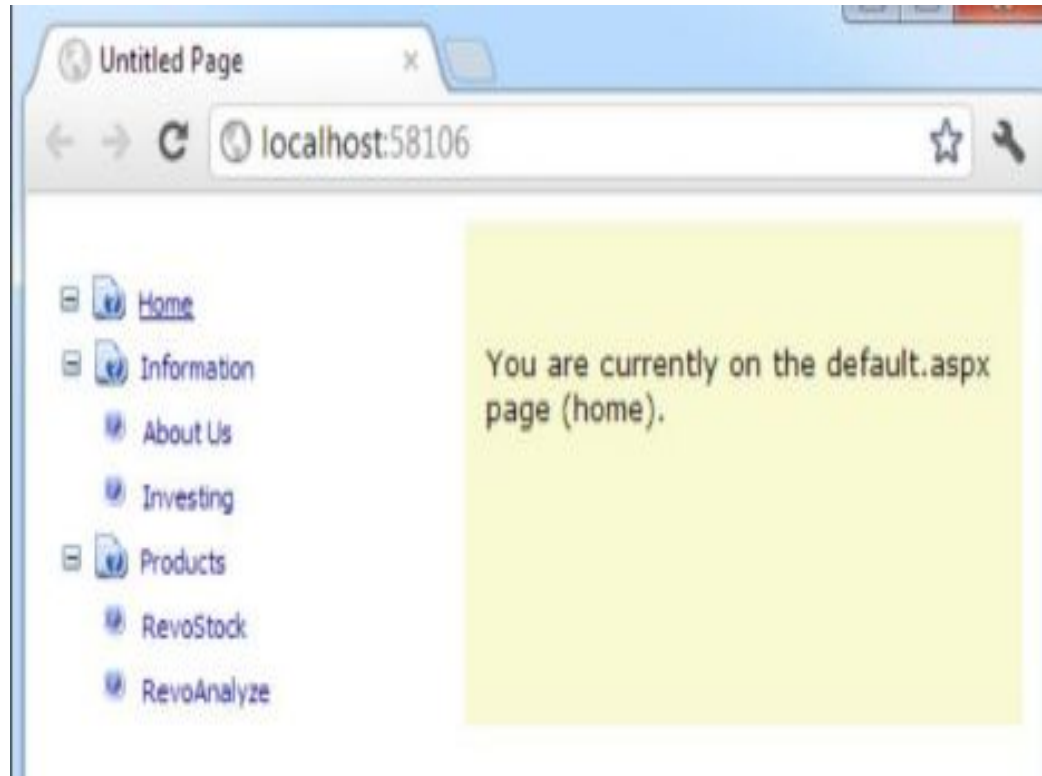
- The SiteMapPath control has a number of properties that can be used to customize its appearance and behavior. These properties include:
 - **SiteMapDataSource:** This property specifies the data source that is used to populate the sitemap.
 - **RootNodeID:** This property specifies the ID of the root node in the sitemap.
 - **PathSeparator:** This property specifies the character that is used to separate the nodes in the sitemap path.
 - **CurrentNodeStyle:** This property specifies the style that is used to render the current node in the sitemap path.
 - **OtherNodeStyle:** This property specifies the style that is used to render the other nodes in the sitemap path.

- The SiteMapPath control can be used to display the sitemap in a variety of ways. Here are a few examples:
- **As a list:** The SiteMapPath control can be used to display the sitemap as a list of links. This can be done by setting the PathSeparator property to a space character.
- **As a table:** The SiteMapPath control can be used to display the sitemap as a table. This can be done by setting the PathSeparator property to a table break character.
- **As a navigation bar:** The SiteMapPath control can be used to display the sitemap as a navigation bar. This can be done by setting the CurrentNodeStyle property to a style that highlights the current node and the OtherNodeStyle property to a style that is used to render the other nodes.

```
<asp:SiteMapPath ID="SiteMapPath1" runat="server" RenderMode="Tree">
  <SiteMapDataSource ID="SiteMapDataSource1" runat="server">
    <SiteMapFile Path="~/sitemap.xml" />
  </SiteMapDataSource>
  <PathSeparator> </PathSeparator>
  <CurrentNodeStyle>
    <Style BackColor="Blue" ForeColor="White" />
  </CurrentNodeStyle>
  <OtherNodeStyle>
    <Style BackColor="Gray" ForeColor="Black" />
  </OtherNodeStyle>
</asp:SiteMapPath>
```

TREEVIEW CONTROL

- The TreeView has a slew of properties that let you change how it's displayed on the page. One of the most important properties is ImageSet, which lets you choose a predefined set of node icons.
- The TreeView offers 16 possible ImageSet values, which are represented by the TreeViewImageSet enumeration.



```
<asp:TreeView ID="TreeView1" runat="server" ImageSet="Faq">  
</asp:TreeView>
```

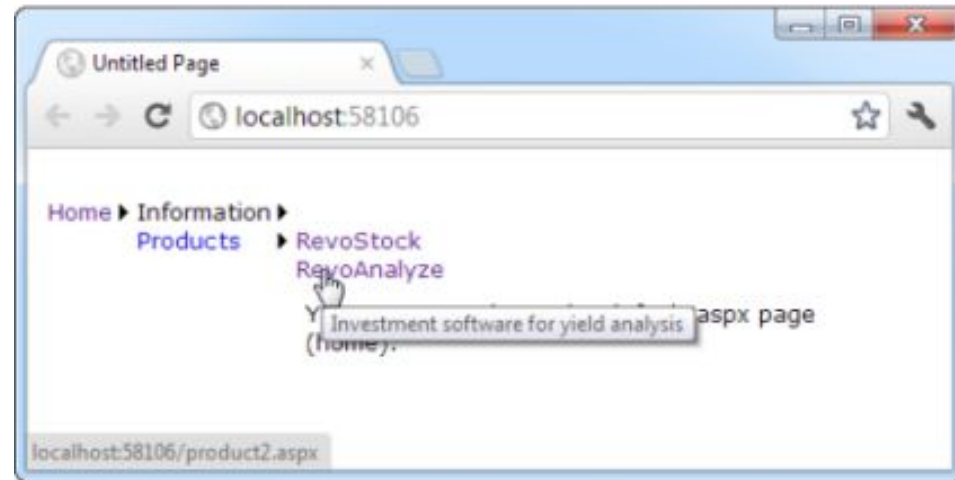
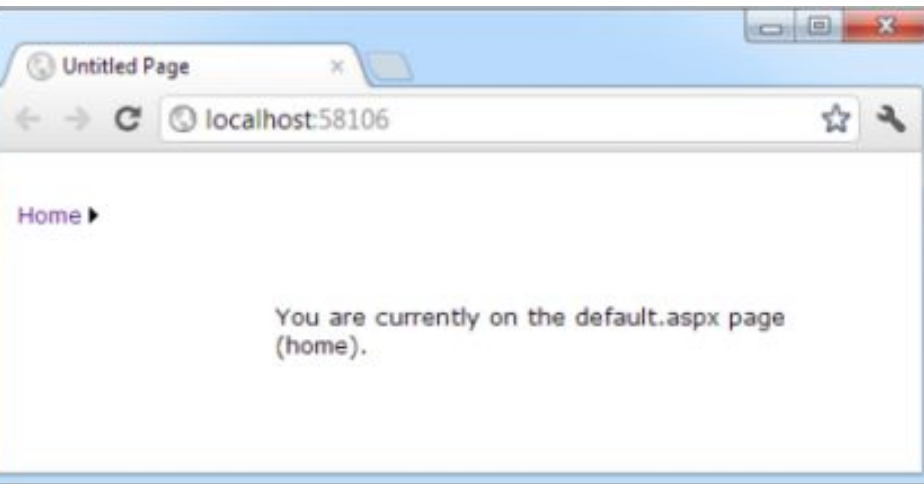

Property	Description
MaxDataBindDepth	Determines how many levels the TreeView will show. By default, MaxDataBindDepth is -1, and you'll see the entire tree.
ExpandDepth	Lets you specify how many levels of nodes will be visible at first. If you use 0, the TreeView begins completely closed.
NodeIndent	Sets the number of pixels between each level of nodes in the TreeView. Set this to 0 to create a nonindented TreeView, which saves space.
ImageSet	Lets you use a predefined collection of node images for collapsed, expanded, and nonexpandable nodes.
ShowLines	Adds lines that connect every node when set to true.
NodeWrap	Lets a node text-wrap over more than one line when set to true.
ShowCheckBoxes	Shows a check box next to every node when set to true. This isn't terribly useful for site maps, but it is useful with other types of trees.

TreeNodeStyle-Added Properties

Property	Description
ImageUrl	The URL for the image shown next to the node.
NodeSpacing	The space (in pixels) between the current node and the node above and below.
VerticalPadding	The space (in pixels) between the top and bottom of the node text and border around the text.
HorizontalPadding	The space (in pixels) between the left and right of the node text and border around the text.
ChildNodesPadding	The space (in pixels) between the last child node of an expanded parent node and the following node.

MENU CONTROL

- The Menu control in ASP.NET is used to create hierarchical navigation menus. It allows you to define menu items and submenus in a structured manner. Here are some key points about the Menu control:



1. Binding to Data Source:

You can bind the Menu control to a data source using the `DataSourceID` property. This allows you to populate the menu dynamically from a data source such as a sitemap.

2. MenuItem Objects:

Alternatively, you can manually add menu items using `MenuItem` objects. Each `MenuItem` can have submenus created by adding child `MenuItem` objects.

3. Rendering:

The Menu control renders as a list of links. The root-level menu items are displayed horizontally, and submenus are displayed as fly-out menus that appear on hovering over a menu item.

4. Styling:

The Menu control provides various style properties to control the appearance of different aspects:

- `StaticMenuStyle` : Sets the appearance of the overall menu box.
- `StaticMenuItemStyle` : Sets the appearance of individual menu items.
- `StaticSelectedStyle` : Sets the appearance of the selected menu item.
- `StaticHoverStyle` : Sets the appearance of the menu item that the user is hovering over.

5. Orientation:

The `Orientation` property allows you to control whether the menu is displayed horizontally or vertically.

6. Templates:

The Menu control supports templates that allow you to customize the rendering of menu items. This is useful when you need more control over the HTML markup and styling of menu items.

7. Hierarchical Navigation:

The Menu control supports hierarchical navigation, allowing you to create multi-level menus with submenus.

```
<asp:Menu ID="Menu1" runat="server" DataSourceID="SiteMapDataSource1" Orientation="Horizontal">
</asp:Menu>
```

```
<asp:SiteMapDataSource ID="SiteMapDataSource1" runat="server" ShowStartingNode="false" />
```

In this example, the Menu control is bound to a sitemap data source using the SiteMapDataSource control. The Orientation property is set to "Horizontal" to display the root-level menu items horizontally.

ADO.NET

(ActiveX Data Object .NET)

UNIT 3

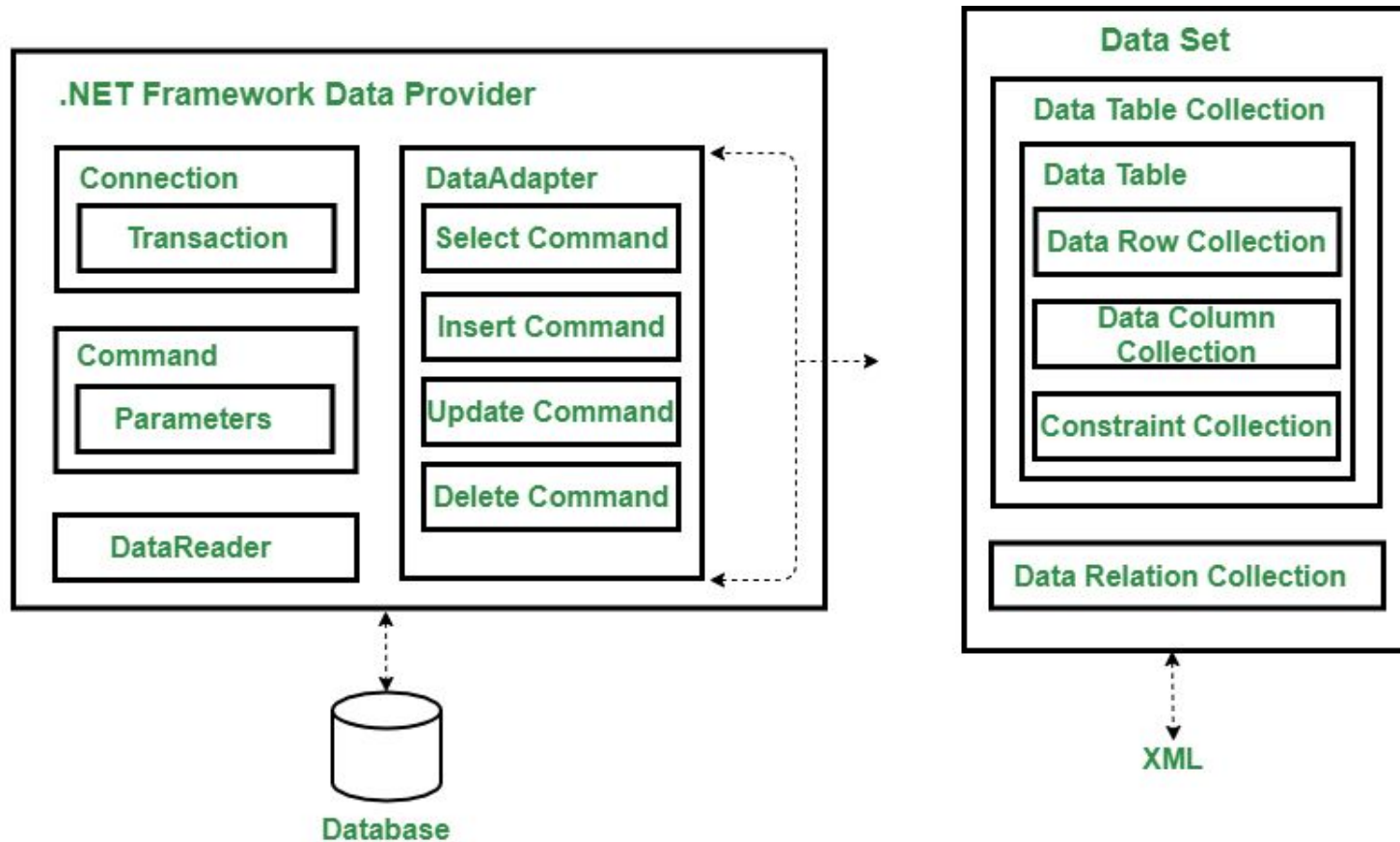
- Database access technology developed by Microsoft as part of the .NET framework known as ActiveX Data Object or ADO.NET.
- ADO.NET consists of managed classes that allow .NET applications to connect to data sources (relational databases), execute commands, and manage disconnected data.
- ADO is automatically installed with Microsoft IIS

What is ADO?

DATA PROVIDER MODEL / ADO.NET ARCHITECTURE

- ADO.NET uses a multilayered architecture that revolves around a few key concepts, such as Connection, Command, and DataSet objects.
- In many past database technologies, such as classic ADO, programmers use a generic set of objects no matter what the underlying data source is.
- For example, if you want to retrieve a record from an Oracle database using ADO code, we use the same Connection class you would use to tackle the task with SQL Server.
- ADO.NET revolves around Data Providers and is built upon the System.Data namespace.
- This namespace contains classes shared across different data providers, ensuring consistency in data access

DATA PROVIDER MODEL / ADO.NET ARCHITECTURE



ADO.NET Data Providers:

- A data provider is a set of ADO.NET classes that allows you to access a specific database, execute SQL commands, and retrieve data.
- A data provider is a bridge between your application and a data source. The classes that make up a data provider include the following:
 1. **Connection:** You use this object to establish a connection to a data source.
 2. **Command:** You use this object to execute SQL commands and stored procedures.
 3. **DataReader:** This object provides fast read-only, forward-only access to the data retrieved from a query.
 4. **DataAdapter:** This object performs two tasks. First, you can use it to fill a DataSet with information extracted from a data source. Second, you can use it to apply changes to a data source, according to the modifications made in a DataSet.

ADO.NET Data Providers:

- Instead, it includes different data providers specifically designed for different types of data sources.
- Each data provider has a specific implementation of the Connection, Command, DataReader, and DataAdapter classes that's optimized for a specific RDBMS.
- For example, if you need to create a connection to a SQL Server database, use a connection class named SqlConnection.
- You can easily create custom ADO.NET providers to wrap non relational data stores, such as the file system or a directory service.
- Some third-party vendors also sell custom providers for .NET

ADO.NET Data Providers:

- The .NET Framework is bundled with a small set of four providers: ADO.Net
 1. **SQL Server provider:** Provides optimized access to a SQL Server database. Uses the System.Data.SqlClient namespace.
 2. **OLE DB provider:** Provides access to any data source that has an OLE DB driver. Uses the System.Data.OleDb namespace.
 3. **Oracle provider:** Provides optimized access to an Oracle database. The .NET Framework Data Provider for Oracle supports Oracle client software version 8.1.7 and later, and uses the System.Data.OracleClient namespace.
 4. **ODBC provider:** Provides access to any data source that has an ODBC driver. Uses the System.Data.Odbc namespace.

DataSet:

- The DataSet object stores data retrieved from a data source in-memory.
- It operates independently of the data source, supporting disconnected data management.
- DataSet holds multiple DataTables and supports relationships, constraints, and more

Direct Data Access

- ADO.NET does not provide a single set of objects that communicate with multiple database management systems (DBMSs).
- ADO.NET supports multiple data providers, each of which is optimized to interact with a specific DBMS.
- The first benefit of this approach is that you can program a specific data provider to access any unique features of a particular DBMS.
- The second benefit is that a specific data provider can connect directly to the underlying engine of the DBMS in question without an intermediate mapping layer standing between the tiers to communicate with a specific type of data source.

Type of Object	Base Class	Relevant Interfaces	Meaning in Life
Connection	DbConnection	IDbConnection	Provides the ability to connect to and disconnect from the data store. Connection objects also provide access to a related transaction object.
Command	DbCommand	IDbCommand	Represents a SQL query or a stored procedure. Command objects also provide access to the provider's data reader object.

Type of Object	Base Class	Relevant Interfaces	Meaning in Life
DataReader	DbDataReader	IDataReader, IDataRecord	Provides forward-only, read-only access to data using a server-side cursor.
DataAdapter	DbDataAdapter	IDataAdapter, IDbDataAdapter	Transfers DataSets between the caller and the data store. Data adapters contain a connection and a set of four internal command objects used to select, insert, update, and delete information from the data store.

Type of Object	Base Class	Relevant Interfaces	Meaning in Life
Parameter	DbParameter	IDataParameter, IDbDataParameter	Represents a named parameter within a parameterized query.
Transaction	DbTransaction	IDbTransaction	Encapsulates a database transaction

.NET Framework Data Provider for SQL Server

- Uses its own protocol for communicating with SQL Server.
- Optimized for direct access to SQL Server, avoiding OLE DB or ODBC layers.
- Classes are located in the System.Data.SqlClient namespace.
- Supports local and distributed transactions.
- To connect to Microsoft SQL Server, use the SqlConnection object of the .NET framework Data Provider for SQL Server
- ADO.NET SqlConnection Class- It is used to establish an open connection to the SQL Server database. It is a sealed class so that cannot be inherited.
- SqlConnection class uses SqlDataAdapter and SqlCommand classes together to increase performance when connecting to a Microsoft SQL Server database.
- Connection does not close explicitly even it goes out of scope. Therefore, you must explicitly close the connection by calling Close() method.

Creating a Connection

- Example code:

1. Import Required Namespaces:

First, you need to import the necessary namespaces for ADO.NET classes. For example, if you're using the SQL Server data provider, you'll import the `System.Data.SqlClient` namespace.

```
using System.Data.SqlClient;
```

2. Create Connection String:

You need to prepare a connection string that contains information about the database server, credentials, and other settings. This string is used to establish the connection.

```
string connectionString = "Server=myServerAddress;Database=myDatabase;User Id=myUsername;Password=myPassword";
```

Replace `myServerAddress`, `myDatabase`, `myUsername`, and `myPassword` with your actual server and authentication details.

Creating a Connection

3. Create Connection Object:

Use the connection string to create a connection object from the appropriate class in the data provider. For SQL Server, you would use `SqlConnection`.

```
SqlConnection connection = new SqlConnection(connectionString);
```

4. Open the Connection:

After creating the connection object, you need to explicitly open the connection to the database.

```
connection.Open();
```

5. Use the Connection:

Once the connection is open, you can use it to execute SQL commands, queries, or other database operations.

Creating a Connection

6. Close the Connection:

After you're done using the connection, it's important to close it to release resources and free up connections in the pool.

```
connection.Close();
```

7. Dispose of Resources:

To ensure proper resource management, you should dispose of the connection object when you're done with it. This is especially important if you're using the `using` statement, which automatically disposes of the object when it goes out of scope.

```
using (SqlConnection connection = new SqlConnection(connectionString))
{
    connection.Open();

    // Perform database operations here

} // The connection will be automatically disposed here
```


Creating a Connection

It is not recommended to hardcode a connection string. Storing the connection string in the configuration file(web.config) is most suitable. So if any changes happen then they can be done in one place only.

```
<connectionStrings>
  <add name="ConnectionString"
        connectionString="data source=.; database=student; integrated security=SSPI"
        providerName="System.Data.SqlClient" />
</connectionStrings>
```

Note: You need to put the above connection string inside the configuration section of the configuration file.

Creating a Connection

How to read the connection string from the app.config file?

Accessing the connection string from web.config

```
        string ConString =  
ConfigurationManager.ConnectionStrings["ConnectionString"].ConnectionString;  
        using (SqlConnection connection = new SqlConnection(ConString))  
        {  
            connection.Open();  
            Console.WriteLine("Connection Established Successfully");  
        }
```

SELECT COMMAND

The Command Object is a fundamental component of ADO.NET.

- It uses a connection object to execute SQL queries.
- Queries can be in the form of inline text, stored procedures, or direct table access.
- The Command Object can execute queries and stored procedures with parameters.
- When a SELECT query is executed, the result set is usually stored in a DataSet or a DataReader object.

Properties of SqlCommand class

Property	Type of Access	Description
Connection	Read/Write	The SqlConnection object that is used by the command object to execute SQL queries or Stored Procedure.
CommandText	Read/Write	Represents the T-SQL Statement or the name of the Stored Procedure.
CommandType	Read/Write	This property indicates how the CommandText property should be interpreted. The possible values are: <ol style="list-style-type: none">1. Text (T-SQL Statement)2. StoredProcedure (Stored Procedure Name)3. TableDirect
CommandTimeout	Read/Write	This property indicates the time to wait when executing a particular command. Default Time for Execution of Command is 30 Seconds. The Command is aborted after it times out and an exception is thrown.

Properties of execute methods

Property	Description
ExecuteNonQuery	This method executes the command specified and returns the number of rows affected.
ExecuteReader	The ExecuteReader method executes the command specified and returns an instance of SqlDataReader class.
ExecuteScalar	This method executes the command specified and returns the first column of the first row of the result set. The remaining rows and columns are ignored.
ExecuteXMLReader	This method executes the command specified and returns an instance of XmlReader class. This method can be used to return the result set in the form of an XML document

DATA READER

- A data reader provides an easy way for the programmer to read data from a database as if it were coming from a stream.
- The data reader is also called a firehose cursor or forward read-only cursor because it moves forward through the data.
- The data reader not only allows you to move forward through each record of the database, but it also enables you to parse the data from each column.
- The `DataReader` class represents a data reader in ADO.NET
- Similar to other ADO.NET objects, each data provider has a data reader class for example; `OleDbDataReader` is the data reader class for OleDb data providers. Similarly, `SqlDataReader` and `OdbcDataReader` are data reader classes for SQL and ODBC data providers, respectively.

DATA READER

- The **IDataReader** interface defines the function of a data reader and works as the **base class for all data provider-specific data reader classes**

DataReader Properties

Property	Description
Depth	Indicates the depth of nesting for row
FieldCount	Returns number of columns in a row
IsClosed	Indicates whether a data reader is closed
Item	Gets the value of a column in native format
RecordsAffected	Number of row affected after a transaction

DATAREADER

DataReader methods

Method	Description
Close	Closes a DataRaeder object.
Read	Reads next record in the data reader.
NextResult	Advances the data reader to the next result during batch transactions.
Getxxx	There are dozens of Getxxx methods. These methods read a specific data type value from a column. For example, GetChar will return a column value as a character and GetString as a string.

DATAREADER

example

```
// Create a connection string
string connectionString = "Integrated Security=SSPI; " +
    "Initial Catalog=Northwind; " +
    "Data source=localhost; ";

string SQL = "SELECT * FROM Customers";

// Create a connection object
SqlConnection conn = new SqlConnection(connectionString);

// Create a command object
SqlCommand cmd = new SqlCommand(SQL, conn);
conn.Open();

// Call ExecuteReader to return a DataReader
SqlDataReader reader = cmd.ExecuteReader();

Console.WriteLine("Customer ID, Contact Name, Contact Title, Address ");
Console.WriteLine("=====");

while (reader.Read())
{
    Console.Write(reader["CustomerID"].ToString() + ", ");
    Console.Write(reader["ContactName"].ToString() + ", ");
    Console.Write(reader["ContactTitle"].ToString() + ", ");
    Console.WriteLine(reader["Address"].ToString() + ", ");
}

// Release resources
reader.Close();
conn.Close();
```

Connected V/s Disconnected Data Access

Connected Data Access	Disconnected Data Access
<p>In Connected Data Access, the application maintains an open and continuous connection to the database while interacting with it. This connection remains active as long as the application is performing database operations.</p>	<p>In Disconnected Data Access, the application connects to the database temporarily to retrieve data. Once the data is fetched, the database connection is closed. The application then works with the data in memory.</p>
<p>With a continuous connection, you can directly execute SQL queries against the database in real-time. This means that you can immediately access and manipulate data as needed.</p>	<p>The data retrieved from the database is stored in memory, typically within a DataSet or DataTable. The application can manipulate and work with this data without maintaining an active database connection.</p>
<p>Connected Data Access provides access to real-time data. Any changes made to the data in the database by other applications or users can be immediately reflected in your application.</p>	<p>Changes made to the in-memory data can be tracked, and batch updates can be sent back to the database when needed. This allows you to update the database efficiently</p>

Connected V/s Disconnected Data Access

Connected Data Access	Disconnected Data Access
<p>This approach is typically suitable for scenarios where real-time data access and immediate updates to the database are essential. For example, in applications requiring continuous monitoring or data input, such as online transaction processing (OLTP) systems.</p>	<p>Disconnected Data Access is well-suited for scenarios where you need to work with data flexibly and efficiently without the overhead of a continuous database connection. It's commonly used in applications that require reporting, data analysis, or scenarios where database connections are costly.</p>
<p>Key components include SqlConnection for managing the database connection, SqlCommand for executing SQL queries, and SqlDataReader for reading query results.</p>	<p>Key components include DataAdapter for fetching data from the database and updating it, DataSet or DataTable for storing data in memory, and DataRow for working with individual rows of data.</p>

DATA BINDING

- Data binding is an aspect that allows us to associate a data source with a control and have that control automatically display data.
- The main characteristic of data binding is that it's declarative, not programmatic. That means data binding is defined outside program, alongside the controls in the .aspx page.
- Types of data binding:
 - Single value
 - Multi value

Single-Value Data Binding

- Single-Value Data Binding allows you to bind individual values from a data source to specific properties of UI controls.
- Each data-bound control displays a single value from the data source. For example, you can bind a TextBox's Text property to a single value from the data source.
- Single-Value Data Binding is typically used when you want to display and edit individual data fields. It is suitable for scenarios where each control corresponds to a specific data attribute.
- Data binding expressions for single-value binding are enclosed within `<%# ... %>` delimiters in the .aspx markup

Single-Value Data Binding

- To evaluate a data binding expression, you must call the `Page.DataBind()` method in program.
- While calling `DataBind()`, ASP.NET will examine all the expressions on page and replace them with the corresponding value.
- If we forget to call the `DataBind()` method, the data binding expression won't be filled in instead, it just gets thrown away when page is rendered to HTML.
- In ASP.NET, most web controls (including `TextBox`, `LinkButton`, `Image`, and many more) support single-value data binding.
- Examples: Binding a label's text to a user's name, binding a textbox to a product's price, or binding an image control's source to an image URL are examples of single-value data binding.

Single-Value Data Binding

In this example, we'll bind a single value (a user's name) to a Label control's Text property.

In this case, the `GetUserName()` method retrieves the user's name from a data source (or any source you prefer), and this name is bound to the Text property of the `lblUserName` Label control using a data binding expression.

```
<asp:Label ID="Label1" runat="server" Text="<%=#uname %>"></asp:Label>
```

```
public partial class WebForm1 : System.Web.UI.Page
{
    public string uname;
    0 references
    protected void Page_Load(object sender, EventArgs e)
    {
        uname = "abc";
        this.DataBind();
    }
}
```

Multi-Value Data Binding/ Repeated- value data binding

- Multi-Value Data Binding allows you to bind collections of data, such as arrays, lists, or datasets, to controls that display multiple values.
- Controls supporting multi-value binding can display a collection of items from the data source. For example, a ListBox can display a list of items from a dataset.
- Multi-Value Data Binding is used when you need to display a list or grid of data, such as a list of products, a set of search results, or a table of records.
- Controls that support multi-value data binding include ListBox, DropDownList, CheckBoxList, RadioButtonList, GridView, and others.
- Multi-Value Data Binding typically requires a data source that contains multiple records or items, such as a dataset with multiple rows.

Multi-Value Data Binding/ Repeated- value data binding

- Multi-Value Data Binding involves setting the DataSource property of a control to the data source and calling the DataBind() method to populate the control with multiple values.
- Examples: Binding a dataset containing a list of products to a GridView, binding a collection of items to a ListBox, or displaying search results in a GridView are examples of multi-value data binding.
- In this example, we'll bind a collection of products to a DropDownList control for the user to select from.

```
<!-- .aspx Markup -->  
<asp:DropDownList ID="ddlProducts" runat="server"></asp:DropDownList>
```

Multi-Value Data Binding/ Repeated- value data binding

```
// Code-Behind (C#)
protected void Page_Load(object sender, EventArgs e)
{
    if (!IsPostBack)
    {
        // Simulated product data as a list
        List<string> products = new List<string>
        {
            "Product A", "Product B", "Product C", "Product D"
        };

        // Bind the list of products to the DropDownList
        ddlProducts.DataSource = products;
        ddlProducts.DataBind();
    }
}
```

Multi-Value Data Binding/ Repeated- value data binding

- Data Properties for List Controls Data Binding

1. DataSource

- Represents a data object containing a collection of data items to display.
- The data object must implement one of the supported interfaces, typically ICollection.

2. DataSourceID

- Instead of programmatically supplying the data object, this property links the list control to a data source control.
- The data source control generates the required data object automatically.
- Use either DataSource or DataSourceID, but not both simultaneously.

Multi-Value Data Binding/ Repeated- value data binding

3. **DataTextField**

- Specifies the field (for data sources with rows) or property (for data sources with objects) of the data item that contains the value to display.
- Each list item displays a single value from this field or property.

4. **DataTextFormatString**

- An optional property that defines a format string to format each DataTextValue before displaying it.
- Useful for specifying how data values should be presented, e.g., formatting a number as currency.

5. **DataValueField**

- Related to DataTextField but not displayed on the page.
- The value from this field or property is stored in the `value` attribute of the underlying HTML tag.
- Enables retrieval of the selected item's value in the program.
- Typically used to store a unique ID or primary key for further data retrieval.

DATA SOURCE CONTROLS – SQLDATASOURCE

- The SqlDataSource control represents a connection to a relational database such as SQL Server or Oracle database, or data accessible through OLEDB or Open Database Connectivity (ODBC).
- Connection to data is made through two important properties ConnectionString and ProviderName.

- The following code snippet provides the basic syntax of the control:

```
<asp:SqlDataSource runat="server" ID="MySqlSource"  
  ProviderName='<%= $ConnectionStrings:LocalNWind.ProviderName %>'  
  ConnectionString='<%= $ConnectionStrings:LocalNWind %>'  
  SelectionCommand= "SELECT * FROM EMPLOYEES" />
```

```
<asp:GridView ID="GridView1" runat="server" DataSourceID="MySqlSource" />
```


- The SqlDataSource command logic is supplied through four properties:
 - SelectCommand
 - InsertCommand
 - UpdateCommand
 - DeleteCommand,
- Each command takes a string. The string we supply can be inline SQL
- **Example:** SqlDataSource that defines a SELECT command for retrieving records.

```
<asp:SqlDataSource ID="sourceEmployees" runat="server"  
ProviderName="System.Data.SqlClient" ConnectionString="<%$  
ConnectionStrings:Northwind %>" SelectCommand="SELECT  
EmployeeID, FirstName, LastName, Title, City FROM Employees"/>
```

DATA CONTROLS

GRIDVIEW CONTROL

- The GridView is an extremely flexible grid control for showing data in a basic grid consisting of rows and columns.
- The GridView control is used to display the values of a data source in a table. Each column represents a field, while each row represents a record.
- The GridView control supports the following features:
 1. Binding to data source controls, such as SqlDataSource.
 2. Built-in sort capabilities.
 3. Built-in update and delete capabilities.
 4. Built-in paging capabilities.

GRIDVIEW CONTROL

5. Built-in row selection capabilities.
6. Programmatic access to the GridView object model to dynamically set properties, handle events, and so on.
7. Multiple key fields.
8. Multiple data fields for the hyperlink columns.
9. Customizable appearance through themes and styles.

GRIDVIEW CONTROL- Defining Columns

- Each column in the GridView control is represented by a DataControlField object. By default, the AutoGenerateColumns property is set to true, which creates an AutoGeneratedField object for each field in the data source.
- Each field is then rendered as a column in the GridView control in the order that each field appears in the data source.
- You can also manually control which column fields appear in the GridView control by setting the AutoGenerateColumns property to false and then defining your own column field collection.
- Different column field types determine the behavior of the columns in the control. The following table lists the different column field types that can be used.

GRIDVIEW CONTROL- Defining Columns

Column field type	Description
BoundField	Displays the value of a field in a data source. This is the default column type of the <code>GridView</code> control.
ButtonField	Displays a command button for each item in the <code>GridView</code> control. This enables you to create a column of custom button controls, such as the Add or the Remove button.
CheckBoxField	Displays a check box for each item in the <code>GridView</code> control. This column field type is commonly used to display fields with a Boolean value.

GRIDVIEW CONTROL- Defining Columns

CommandField

Displays predefined command buttons to perform select, edit, or delete operations.

HyperLinkField

Displays the value of a field in a data source as a hyperlink. This column field type enables you to bind a second field to the hyperlink's URL.

ImageField

Displays an image for each item in the [GridView](#) control.

TemplateField

Displays user-defined content for each item in the [GridView](#) control according to a specified template. This column field type enables you to create a custom column field.

GRIDVIEW CONTROL

```
<asp:GridView ID="GridView1" runat="server" AutoGenerateColumns="true" DataSourceID="SqlDataSource1">
</asp:GridView>
<asp:SqlDataSource ID="SqlDataSource1" runat="server" ConnectionString="<%= $ConnectionStrings:YourConnectionString
%>"
    SelectCommand="SELECT * FROM YourTable">
</asp:SqlDataSource>
```

DetailsView Control

- The DetailsView control is used to display a single record from a data source in a table, where each field of the record is displayed in a row of the table.
- It can be used in combination with a GridView control for master-detail scenarios. The DetailsView control supports the following features:
 1. Binding to data source controls, such as SqlDataSource.
 2. Built-in inserting capabilities.
 3. Built-in updating and deleting capabilities.
 4. Built-in paging capabilities.
 5. Programmatic access to the DetailsView object model to dynamically set properties, handle events, and so on.
 6. Customizable appearance through themes and styles.

DetailsView Control - Row field

- Each data row in the DetailsView control is created by declaring a field control. Different row field types determine the behavior of the rows in the control.
- Field controls derive from DataControlField. The following table lists the different row field types that can be used.

```
<asp:DetailsView ID="detailsView" runat="server" DataSourceID="sqlDataSource" AutoGenerateRows="true">
  <EmptyDataTemplate>
    No data available.
  </EmptyDataTemplate>
</asp:DetailsView>
```

```
<asp:SqlDataSource ID="sqlDataSource" runat="server" ConnectionString="<%= $ConnectionStrings:YourConnectionString %>"
  SelectCommand="SELECT * FROM Employees" />
```

DetailsView Control - Row field

**Column field
type**

Description

BoundField

Displays the value of a field in a data source as text.

ButtonField

Displays a command button in the [DetailsView](#) control. This allows you to display a row with a custom button control, such as an Add or a Remove button.

CheckBoxField

Displays a check box in the [DetailsView](#) control. This row field type is commonly used to display fields with a Boolean value.

DetailsView Control - Row field

CommandField	Displays built-in command buttons to perform edit, insert, or delete operations in the DetailsView control.
HyperLinkField	Displays the value of a field in a data source as a hyperlink. This row field type allows you to bind a second field to the hyperlink's URL.
ImageField	Displays an image in the DetailsView control.
TemplateField	Displays user-defined content for a row in the DetailsView control according to a specified template. This row field type allows you to create a custom row field.

FormView Control

- The FormView control is used to display a single record from a data source. It is similar to the DetailsView control, except it displays user-defined templates instead of row fields.
- Creating your own templates gives you greater flexibility in controlling how the data is displayed.
- The FormView control supports the following features:
 1. Binding to data source controls, such as SqlDataSource and ObjectDataSource.
 2. Built-in inserting capabilities.
 3. Built-in updating and deleting capabilities.
 4. Built-in paging capabilities.
 5. Programmatic access to the FormView object model to dynamically set properties, handle events, and so on.
 6. Customizable appearance through user-defined templates, themes, and styles.

FormView Control- Templates

- For the FormView control to display content, you need to create templates for the different parts of the control.
- Most templates are optional; however, you must create a template for the mode in which the control is configured.
- For example, a FormView control that supports inserting records must have an insert item template defined. The following table lists the different templates that you can create.

FormView Control

Template type	Description
EditItemTemplate	Defines the content for the data row when the FormView control is in edit mode. This template usually contains input controls and command buttons with which the user can edit an existing record.
EmptyDataTemplate	Defines the content for the empty data row displayed when the FormView control is bound to a data source that does not contain any records. This template usually contains content to alert the user that the data source does not contain any records.
FooterTemplate	Defines the content for the footer row. This template usually contains any additional content you would like to display in the footer row. Note: As an alternative, you can simply specify text to display in the footer row by setting the FooterText property.

FormView Control

HeaderTemplate	Defines the content for the header row. This template usually contains any additional content you would like to display in the header row. Note: As an alternative, you can simply specify text to display in the header row by setting the HeaderText property.
ItemTemplate	Defines the content for the data row when the FormView control is in read-only mode. This template usually contains content to display the values of an existing record.
InsertItemTemplate	Defines the content for the data row when the FormView control is in insert mode. This template usually contains input controls and command buttons with which the user can add a new record.
PagerTemplate	Defines the content for the pager row displayed when the paging feature is enabled (when the AllowPaging property is set to <code>true</code>). This template usually contains controls with which the user can navigate to another record. Note: The FormView control has a built-in pager row user interface (UI). You need to create a pager template only if you want to create your own custom pager row.

FormView Control

```
<asp:FormView ID="formView" runat="server" DataSourceID="sqlDataSource" DefaultMode="ReadOnly">
  <EmptyDataTemplate>
    No data available.
  </EmptyDataTemplate>
  <ItemTemplate>
    <h3>Name: <%=# Eval("EmployeeName") %></h3>
    <p>Email: <%=# Eval("Email") %></p>
  </ItemTemplate>
  <EditItemTemplate>
    <h3>Edit Employee</h3>
    Name: <asp:TextBox ID="txtEditName" runat="server" Text='<%=# Bind("EmployeeName") %>' /><br />
    Email: <asp:TextBox ID="txtEditEmail" runat="server" Text='<%=# Bind("Email") %>' /><br />
    <asp:Button ID="btnUpdate" runat="server" Text="Update" CommandName="Update" />
    <asp:Button ID="btnCancel" runat="server" Text="Cancel" CommandName="Cancel" />
  </EditItemTemplate>
</asp:FormView>

<asp:SqlDataSource ID="sqlDataSource" runat="server" ConnectionString="<%= $ ConnectionStrings:YourConnectionString %>"
  SelectCommand="SELECT EmployeeName, Email FROM Employees" />
```

- XML stands for EXtensible Markup Language XML is a markup language much like HTML. XML was designed to describe data
- XML tags are not predefined. You must define your own tags
- XML uses a Document Type Definition (DTD) or an XML Schema to describe the data
- XML with a DTD or XML Schema is designed to be self descriptive

WORKING WITH XML

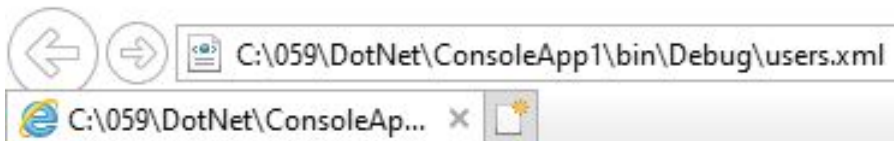
XML CLASSES

- The .NET framework contains a rich set of classes for working with XML data. Classes are divided into multiple namespaces.
- The main classes for working with XML data are as follows :
 1. XmlTextReader
 2. XmlTextWriter
 3. XmlDocument
 4. XmlDataDocument
 5. XmlNodeReader
 6. DocumentNavigator
 7. DataDocumentNavigator
 8. XslTransform

XmlTextWriter

1. The XmlTextWriter class is part of the System.Xml namespace and is used for writing XML data.
2. It provides a way to create and format XML documents by writing elements, attributes, and content to an output stream, such as a file or a memory buffer.
3. XmlTextWriter provides methods like WriteStartElement , WriteEndElement , and WriteElementString to create XML elements and their content.
4. You can add attributes to elements using WriteStartAttribute and WriteEndAttribute methods.
5. It can automatically format the XML data for human readability with indentation and line breaks.

XmlTextWriter



```
<?xml version="1.0"?>
- <Users>
    <user id="1">XYZ</user>
    <user id="2">ABC</user>
</Users>
```

```
public void writexml()
{
    XmlTextWriter wrt = new XmlTextWriter("users.xml", null);
    wrt.WriteStartElement("Users"); //rootelement

    wrt.WriteStartElement("user");
    wrt.WriteAttributeString("id", "1");
    wrt.WriteString("XYZ");
    wrt.WriteEndElement();

    wrt.WriteStartElement("user");
    wrt.WriteAttributeString("id", "2");
    wrt.WriteString("ABC");
    wrt.WriteEndElement();

    wrt.WriteEndElement();

    wrt.Close();
}
```


XmlTextWriter

6. So, we start by creating an instance of the `XmlTextWriter` class. It takes at least one parameter, in this case the path to where the XML should be written, but comes in many variations for various purposes.

7. The first thing we should do is to call the `WriteStartDocument()` method. After that, we write a start element called "users". The `XmlTextWriter` will translate this into `<users>`.

8. Before closing it, we write another start element, "user", which will then become a child of "users". We then proceed to adding an attribute (age) to the element, using the `WriteAttributeString()` method, and then we write the inner text of the element, by calling the `WriteString()` method.

XmlTextWriter

9. We then make sure to close the first "user" element with a call to the `WriteEndElement()` method.

10. This process is repeated to add another user, lastly we call the `WriteEndDocument()` method.

11. To get the `XmlWriter` to write our data to the disk, we call the `Close()` method. You can now open the file, "myfile.xml", in the same directory where the EXE file of your project is, usually in the `bin\debug` directory.

XmlTextReader

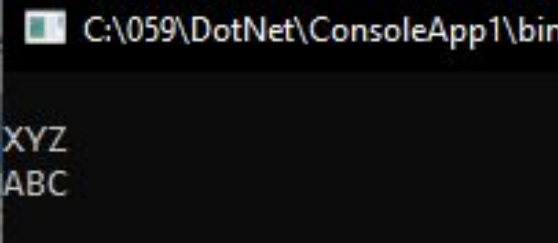
1. The XmlTextReader class, also part of the System.Xml namespace, is used for reading XML data.
2. It provides a forward-only, read-only way to parse and navigate through XML documents.
3. XmlTextReader provides forward-only, read-only access to a stream of XML data. The current node refers to the node on which the reader is positioned.
4. The reader is advanced using any of the read methods and properties reflect the value of the current node.

XmlTextReader

5. XmlTextReader reads XML data sequentially, from start to finish, in a forward-only manner, which minimizes memory consumption and improves performance.
6. XmlTextReader offers methods like Read , ReadStartElement , ReadEndElement , and ReadString to traverse XML elements and extract data.
7. It can perform validation against XML schemas (XSD) and report errors when the XML data doesn't conform to the schema.

XmlTextReader

```
public void readxmldata()  
{  
    XmlTextReader reader = new XmlTextReader("users.xml");  
    while (reader.Read())  
    {  
        Console.WriteLine(reader.ReadString());  
    }  
}
```



A screenshot of a Windows console application window. The title bar shows the file path "C:\059\DotNet\ConsoleApp1\bin". The console output displays two lines of text: "XYZ" on the first line and "ABC" on the second line.

XmlTextReader

```
using System.Xml;
namespace ConsoleApp1{
    0 references
    internal class Program{
        0 references
        static void Main(string[] args){
            XmlTextReader reader = new XmlTextReader("myfile.xml");
            while (reader.Read())
            {
                // Check the node type
                if (reader.NodeType == XmlNodeType.Element)
                {
                    // Read element name
                    string elementName = reader.Name;
                    Console.WriteLine("Root",elementName);
                    // Read element content
                    reader.ReadToFollowing("user");
                    string childname = reader.Name;
                    string elementContent = reader.ReadElementContentAsString("user","");
                    Console.WriteLine($"Element1: {childname}, Content: {elementContent}");
                    Console.ReadKey();
                }
            }
        }
    }
}
```

Root

Element1: user, Content: John Doe

CACHING

- Caching in .NET refers to the temporary storage of data in memory to improve application performance by reducing the need to fetch the same data repeatedly from the original source, such as a database or a web service.
- Unlike many other performance enhancing techniques, caching bolsters both performance and scalability.
- Performance is better because the time taken to retrieve the information is cut down dramatically.
- Scalability is improved because you work around bottlenecks such as database connections.

CACHING

- When we store information in a cache, we expect to find it there on a future request most of the time. However, the lifetime of that information is at the discretion of the server.
- If the cache becomes full or other applications consume a large amount of memory, information will be selectively evicted from the cache, ensuring that performance is maintained. It's this self sufficiency that makes caching so powerful.

TYPES OF CACHING

.NET provides several types of caching mechanisms to achieve this:

1. Output Caching
2. Data caching
3. Fragment caching
4. Data source caching

1. Output Caching

- Output caching is used to cache the entire HTML output of a web page.
- When a page is requested for the first time, ASP.NET processes the page and caches the resulting HTML.
- Subsequent requests for the same page are served directly from the cache without executing the page's code.
- Reduces server load and improves response time, especially for pages with content that doesn't change frequently.
- Output caching can be configured using the `@OutputCache` directive in ASP.NET Web Forms

- Different types of output caching are as follows:

A. Declarative Output Caching

- Output caching stores a copy of the final rendered HTML page to improve performance. You can add the OutputCache directive to your ASP.NET page to specify caching settings.

```
<%@ OutputCache Duration="20" VaryByParam="None" %>
```

B. Caching and the Query String

- Caching allows you to efficiently reuse slightly stale data while improving performance. You can use the VaryByParam attribute to cache separate copies of the page based on query string parameters.

```
<%@ OutputCache Duration="20" VaryByParam="*" %>
```

- Different types of output caching are as follows:

C. Caching with Specific Query String Parameters

- For more precise control, you can specify specific query string parameters to vary caching. This is useful when you want to cache pages with different query string values separately

```
<%@ OutputCache Duration="20" VaryByParam="ProductID" %>
```

2. Data caching

- Data caching is used to store data or objects in memory to avoid repeated expensive data retrievals from sources like databases
- In your code, you manually store and retrieve data from the cache using keys.
- When data is cached, it's stored in memory and can be quickly accessed without performing the original data retrieval operation.
- Reduces database or data source load, improving application performance
- You can set expiration policies on cached data, such as time-based expiration or dependency-based expiration.

- You can add items to the cache using the `Cache["key"] = item;` syntax.
- A better approach is to use the `Cache.Insert()` method, which allows you to specify various caching parameters, including dependencies, expiration, and priority.

```
Cache.Insert("MyItem", obj, null, DateTime.MaxValue, TimeSpan.FromMinutes(10));
```


3. Fragment caching

- **Purpose:** Fragment caching is a variation of output caching, where only a portion (or fragment) of a web page is cached.
- **How it Works:** You can cache the output of user controls or specific parts of a page, rather than the entire page. This allows you to selectively cache parts of a page that are more static while keeping dynamic content uncached.
- **Advantages:** Fine-grained control over caching for specific page sections, reducing processing time for dynamic content.
- **Configuration:** Typically implemented using user controls and the `<%@ OutputCache %>` directive.

4. Data source caching

- **Purpose:** Data source caching is specific to data source controls like `SqlDataSource`, `ObjectDataSource`, and `XmlDataSource`.
- **How it Works:** These data source controls have built-in caching mechanisms. You configure properties such as `CacheDuration` to specify how long data retrieved from these sources should be cached.
- **Advantages:** Simplifies caching for data-bound controls by providing declarative caching settings.
- **Configuration:** Configure caching properties directly in the data source control declarations.

```
<asp:SqlDataSource ID = "SqlDataSource1" runat = "server"  
    ConnectionString = "< '%$ ConnectionStrings: connectionstringID %>"  
    ProviderName = "< '%$ ConnectionStrings: SQLProviderName %>"  
    SelectCommand = "SELECT * FROM [dbTablename]"  
    EnableCaching = "true" CacheDuration = "60">  
</asp:SqlDataSource>
```

- LINQ in C# is used to work with data access from sources such as objects, data sets, SQL Server, and XML.
- LINQ stands for Language Integrated Query. LINQ is a data querying API with SQL like query syntaxes.
- LINQ provides functions to query cached data from all kinds of data sources.
- The data source could be a collection of objects, database or XML files.
- We can easily retrieve data from any object that implements the `IEnumerable<T>` interface.

LINQ

(Language Integrated Query)

Advantages of LINQ:

1. **Familiar Language:** No need to learn a new query language for each data source or format.
2. **Less Coding:** Reduces the amount of code compared to traditional approaches.
3. **Readable Code:** Enhances code readability for easier maintenance.
4. **Standardized Querying:** Same LINQ syntax for multiple data sources.
5. **Compile-Time Safety:** Provides type checking at compile time.
6. **IntelliSense Support:** Offers IntelliSense for generic collections.
7. **Data Shaping:** Retrieve data in various formats.

Disadvantages of LINQ:

1. **Complex Queries:** Writing complex queries can be challenging compared to SQL.
2. **Lack of Execution Plans:** Cannot leverage SQL's Cached Execution Plans.
3. **Performance Impact:** Poorly written queries can degrade performance.
4. **Recompilation:** Changes to queries may require application recompilation and redeployment.

Common LINQ Operators:

1. **Where:** Filters a collection based on a specified condition.
2. **Select:** Projects elements of a collection into a new form.
3. **OrderBy:** Sorts elements in ascending or descending order.
4. **GroupBy:** Groups elements in a collection based on a common key.
5. **Join:** Combines two collections based on a common key.
6. **Aggregate:** Performs an aggregation operation (e.g., sum, average) on a collection.
7. **Any:** Checks if any elements in a collection satisfy a condition.
8. **All:** Checks if all elements in a collection satisfy a condition.
9. **Distinct:** Removes duplicate elements from a collection.
10. **Take:** Returns a specified number of elements from the start of a collection.

```
List<Person> people = new List<Person>
{
    new Person { Name = "Alice", Age = 25 },
    new Person { Name = "Bob", Age = 32 },
    new Person { Name = "Charlie", Age = 28 },
    new Person { Name = "David", Age = 35 },
};

// LINQ query to filter people above 30 years old
var result = from person in people
              where person.Age > 30
              select person;

// Print the result
Console.WriteLine("People above 30 years old:");
foreach (var person in result)
{
    Console.WriteLine($"{person.Name}, {person.Age} years old");
}
```


ASP.NET AJAX

1. AJAX stands for Asynchronous JavaScript and XML. This is a cross platform technology which speeds up response time. The AJAX server controls add script to the page which is executed and processed by the browser.
2. However like other ASP.NET server controls, these AJAX server controls also can have methods and event handlers associated with them, which are processed on the server side.
3. The control toolbox in the Visual Studio IDE contains a group of controls called the 'AJAX Extensions'

ScriptManager Control

- The ScriptManager is an important component in ASP.NET AJAX that is used to manage client-script libraries, partial-page rendering, and other client-side functionalities. It is primarily associated with ASP.NET Web Forms applications
- It is both a server-side control (`<asp:ScriptManager>`) and a client-side script (`Sys.WebForms.PageRequestManager`) that enables AJAX functionality in ASP.NET web applications
- The ScriptManager control can be placed on your web page, typically in the `<form>` tag, to enable AJAX features.
- It is essential to have only one ScriptManager control per page
-

ScriptManager Control

- In this example, we have added a ScriptManager control to the web page. It enables AJAX functionality and handles JavaScript libraries and scripts.

```
<asp:ScriptManager ID="ScriptManager1" runat="server"></asp:ScriptManager>
```

Partial Refreshes

- Partial page refreshes, often referred to as partial rendering or AJAX updates, allow specific sections of a web page to be refreshed without reloading the entire page.
- This technique greatly improves the user experience by reducing flicker and improving response times.
- ASP.NET AJAX provides the UpdatePanel control to enable partial page refreshes. You wrap the content you want to update in an UpdatePanel .

- The UpdateMode property of the UpdatePanel determines when an update should occur. Options include "Always" (default) or "Conditional" (only when triggered explicitly).
- Partial refreshes are ideal for scenarios where you want to update specific parts of a page, such as a shopping cart, chat messages, or live search results, without affecting the rest of the page.

```
<asp:UpdatePanel ID="UpdatePanel1" runat="server">
  <ContentTemplate>
    <!-- Content you want to refresh -->
    <asp:Label ID="lblMessage" runat="server" Text="Initial Content"></asp:Label>
    <asp:Button ID="btnUpdate" runat="server" Text="Update Content" OnClick="btnUpdate_Click" />
  </ContentTemplate>
</asp:UpdatePanel>
```

In this example, we have an UpdatePanel containing a label and a button. When the button is clicked, only the content within the UpdatePanel will be refreshed without reloading the entire page. Here's the server-side code to handle the button click event:

```
protected void btnUpdate_Click(object sender, EventArgs e)
{
    lblMessage.Text = "Updated Content";
}
```


Progress Notification

- Progress notification is a feature in ASP.NET AJAX that provides feedback to users during asynchronous operations, indicating that something is happening in the background.
- It is often implemented using the UpdateProgress control.
- The UpdateProgress control allows you to define content that is displayed to the user while an asynchronous operation is in progress, typically accompanied by a loading animation (e.g., a spinning wheel or progress bar).
- Progress notification enhances user experience by letting users know that their request is being processed, especially in cases where operations may take some time to complete.

```
<asp:UpdateProgress ID="updateProgress1" runat="server">
  <ProgressTemplate>
    <div class="progress-container">
      
      <span>Loading...</span>
    </div>
  </ProgressTemplate>
</asp:UpdateProgress>
```

In this example, we use the UpdateProgress control to display a loading spinner and a "Loading..." message during an asynchronous operation within an UpdatePanel . When an operation is in progress, this content will be shown automatically, and it will disappear when the operation is completed.

Timed Refreshes

- Timed refreshes involve updating content on a web page at regular intervals, without requiring user interaction.
- ASP.NET AJAX provides the Timer control (`<asp:Timer>`) to facilitate timed refreshes.
- You set the Interval property of the Timer control to specify the time (in milliseconds) between each refresh.
- When the timer's interval elapses, it triggers a postback (either full or partial, depending on context) and invokes a server-side event (e.g., `OnTick`).

Timed Refreshes

- Timed refreshes are useful for scenarios like displaying real-time data (e.g., stock prices, weather updates), live notifications, or periodically updating content on a page (e.g., news feeds).

Timed Refreshes Example:

```
<asp:Timer ID="Timer1" runat="server" Interval="5000" OnTick="Timer1_Tick" />  
<asp:Label ID="lblTime" runat="server" Text="Current Time: " />
```

In this example, we have a Timer control set to refresh every 5000 milliseconds (5 seconds). The timer is linked to a label. Here's the server-side code to handle the timer tick event:

Timed Refreshes

```
protected void Timer1_Tick(object sender, EventArgs e)
{
    lblTime.Text = "Current Time: " + DateTime.Now.ToLongTimeString();
}
```

This code updates the label's text with the current time every 5 seconds without requiring any user interaction.

These examples demonstrate how to use ScriptManager, perform partial refreshes, provide progress notifications, and implement timed refreshes in ASP.NET AJAX applications.