- Client-side scripting,
- Building own components,
- Automating active server pages,
- Efficiency and scalability.

What is Microsoft .Net ?

.Net is a programming framework developed by Microsoft, which can be used to build different types of applications like – Console, Windows, Web application and Mobile based applications. It provides controlled environment with built-in tools for developing, installing and executing different types of applications.

The main two components of .Net Framework are Common Language Runtime (CLR) and .Net Framework Class Library (FCL).

Why we choose .Net ?

Developing applications using .Net framework is very robust and highly secure with great quality. .Net platform reduces development time, creates quality, reliable, and scalable applications that ensure smooth functioning of complex business applications. Hence it helps customer to improve their business easily.

.Net job opportunities ?

This is a very interesting question, Microsoft being a market leader in Software industry, requirements for .Net developers are always in demand.

Beneficial use of .NET framework:

- It significantly decreases the quantity of code necessary in large web applications which are developed in .NET framework.
- Web applications developed in ASP.NET are secure as Windows confirmation and configuration can be attained for every application.
- This development provides WYSIWYG (What You See Is What You Get).
- It provides server controls and blueprints with capability of drag and drop and involuntary operation.
- HTML code and source code are separated so changes can be done easily in ASP.NET framework.
- ASP.NET platform is language independent.

What types of applications can we develop using .Net?

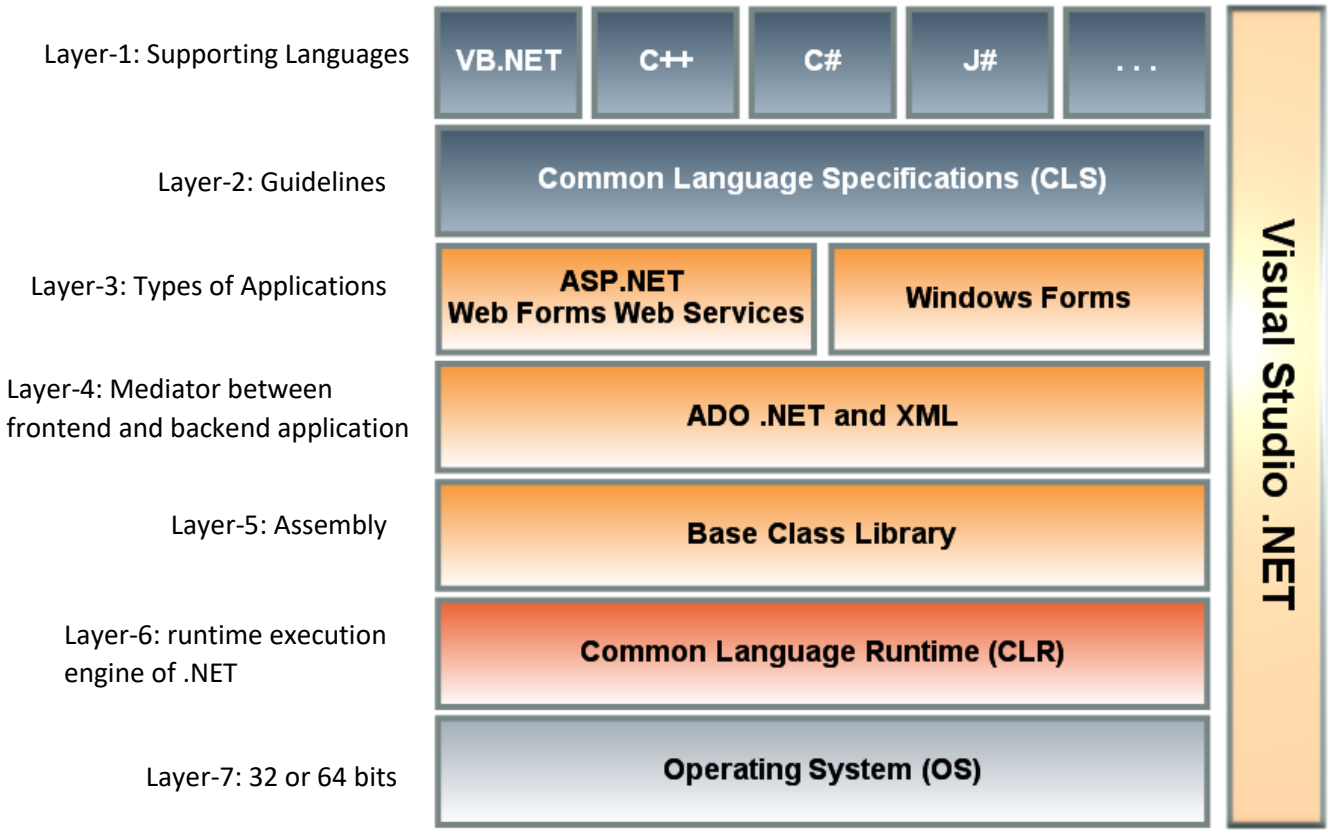
When you hear .NET, the first idea that comes to your mind will probably be internet or networked applications. Although this is absolutely true, there are many more types of applications to create with .NET.

So here is a more or less full list of various types of application that we can develop on .NET.

- **ASP.Net Web applications** are programs that used to run inside some web server to fulfill the user requests over the http. ASP.NET Web applications can range from simple Web sites that consist of HTML pages to advanced enterprise applications that run on local and remote networks. These enterprise applications also provide components for exchanging data using XML. This type includes dynamic and data driven browser based applications. (Ex: Hotmail and Google).
- **Web services** are “web callable” functionality available via industry standards like HTTP, XML and SOAP.
- **Windows applications** are form based standard Windows desktop applications for common day to day tasks. (Ex: Microsoft word). Run only under Windows environment. These applications consume the services provided by the Windows operating system.

- **Windows services** are long-running executable applications that run on the system as a background process. These applications do not interfere with the working of the other processes that run on the same computer. Windows services execute within separate Windows sessions created specifically for each Windows service. These services do not have a graphic user interface and are ideal for running on the server. Windows services were earlier called NT services.
- **Console applications** are light weight programs run inside the command prompt (DOS) window. They are commonly used for test applications.
- **Mobile applications** can run on multiple mobile devices, such as Pocket PCs, mobile phones, or personal digital assistants. These applications provide ubiquitous access to data from mobile devices. The .NET Framework automatically makes changes to these applications to enable them to run on multiple browsers, depending on the mobile device.
- **Class libraries** are components that you create once and reuse a number of times in multiple applications. Class libraries allow you to define several classes, along with their methods and interfaces, in one file. These libraries compile to .dll files and facilitate rapid development of new applications because of reusability of code. To access the functionality of the classes in a class library from your application, you need to include a reference to that library in your program.

Architecture of .NET Framework



- **Common Type System** (CTS)

  It describes set of data types that can be used in different .Net languages in common. (i.e), CTS ensures that objects written in different .Net languages can interact with each other. For Communicating between programs written in any .NET complaint language, the types have to be compatible on the basic level.

  The common type system supports two general categories of types:

  Value types:

Value types directly contain their data, and instances of value types are either allocated on the stack or allocated inline in a structure. Value types can be built-in (implemented by the runtime), user-defined, or enumerations.

Reference types:

Reference types store a reference to the value's memory address, and are allocated on the heap. Reference types can be self-describing types, pointer types, or interface types. The type of a reference type can be determined from values of self-describing types. Self-describing types are further split into arrays and class types. The class types are user-defined classes, boxed value types, and delegates.

- **CLS (Common Language Specification):**

  It is a sub set of CTS and it specifies a set of rules that needs to be adhered or satisfied by all language compilers targeting CLR. It helps in cross language inheritance and cross language debugging.

  This is a subset of the CTS which all .NET languages are expected to support. Microsoft has defined CLS which are nothing but guidelines that language to follow so that it can communicate with other .NET languages in a seamless manner.

- **ADO.NET:** ADO.NET provides support for data access in Microsoft .NET and its features include:
  - Support for disconnected data access model
  - Integration with the .Net framework
  - XML support

  This model can, however, only be used from the managed code environment. This implies that there is no COM interoperability allowed for ADO.NET.

- **The Base Class Library (BCL)** includes a small subset of the entire class library and is the core set of classes that serve as the basic API of the Common Language Runtime.

  For example: The classes in mscorlib.dll and some of the classes in System.dll and System.core.dll are considered to be a part of the BCL. The BCL classes are available in both .NET Framework as well as its alternative implementations including .NET Compact Framework, Microsoft Silverlight and Mono.

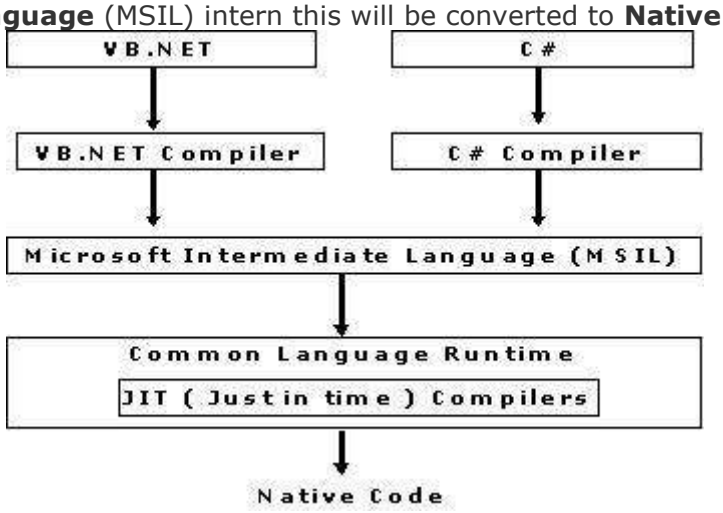
Namespaces in the BCL
System
System. CodeDom System. Collections System. Diagnostics System. Globalization System. IO System. Resources System. Text
System. Text.RegularExpressions

- **The Framework Class Library (FCL)** is a superset of the BCL classes and refers to the entire class library that ships with .NET Framework. It includes an expanded set of libraries, including Windows Forms, ADO.NET, ASP.NET, Language Integrated Query, Windows Presentation Foundation, Windows Communication Foundation among others. The FCL is much larger in scope than standard libraries for languages like C++, and comparable in scope to the standard libraries of Java.
- **Common Language Runtime (CLR)**

  .Net Framework provides runtime environment called **Common Language Runtime** (CLR).It provides an environment to run all the .Net Programs. 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 as it provides memory management and thread management.

  Programmatically, when our program needs memory, CLR allocates the memory for scope and de-allocates the memory if the scope is completed.

  Language Compilers (e.g. C#, VB.Net, J#) will convert the Code/Program to **Microsoft Intermediate Language** (MSIL) intern this will be converted to **Native Code**



- **CLR Services:**

  The CLR provides a lot of services for applications that run in the .NET environment. These include:
  - Just In Time Compilation
  - Memory management and isolation of the application memory
  - Garbage Collection
  - Code Verification
  - Code Access Security
  - Verification of type safety
  - Exception handling
  - Conversion of the MSIL to the native code
  - Access to metadata
- **The JIT compiler:** The MSIL code and metadata are loaded into the memory with the help of Common Language Runtime (CLR). The JIT compiler then compiles the MSIL code to native code and executes it at runtime.

- **Garbage Collection:** CLR automatically manages memory thus eliminating memory leakes. When objects are not referred GC automatically releases those memory thus providing efficient memory management.
- **IL**(**Intermediate language**)-to-native translators and optimizer's : CLR uses JIT and compiles the IL code to machine code and then executes.
- **Code Verification:** This ensures proper code execution and type safety while the code runs.It prevents the source code to perform illegal operation such as accessing invalid memory locations etc.

.NET Framework Data Types

This section contains the information you need to translate the script-oriented Object Model Reference into Microsoft .NET Framework base data types. The Windows Media Player script reference has almost all the information you need to use the Windows Media Player control in a .NET Framework-based program, and in most cases, the syntax will be similar to that of other scripting languages such as Microsoft JScript.

The Windows Media Player reference provides the JScript data type and, if necessary, the C++ conversion. For example, a **Number**might be returned by a method. JScript treats all numbers in the same way, but other languages distinguish between different types of numbers (integer, floating point, and so on). The reference gives the C++ conversion for number data types because numbers can be processed differently by C++. For example, C++ uses the **int** data type for integer arithmetic and **float** for floating point.

The .NET Framework uses a slightly different system of base data types. The following table shows the differences in the common data types you are likely to use. For more information on .NET Framework base data types and the conversion to other data type systems, see the .NET Framework Developer's Guide discussion of System Namespace base data types.

This table gives the .NET Framework class name and the C# data type. Data types for other languages are defined for each language in their respective language references.

Script data type	C++ data type	.NET Framework class (C# data type)
Number	int	Int32 (int)
Number	long	Int32 (int)
Number	double	Double (double)
Number	float	Single (float)
String	BSTR	String (string)
Boolean	VARIANT_BOOL	Boolean (bool)
Object	Object	Object (object)

Input Validation Controls

ASP.NET validation controls validate the user input data to ensure that useless, unauthenticated, or contradictory data don't get stored.

ASP.NET provides the following validation controls:

- RequiredFieldValidator
- RangeValidator
- CompareValidator
- RegularExpressionValidator
- CustomValidator

- ValidationSummary

BaseValidator Class

The validation control classes are inherited from the BaseValidator class hence they inherit its properties and methods. Therefore, it would help to take a look at the properties and the methods of this base class, which are common for all the validation controls:

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

RequiredFieldValidator Control

The RequiredFieldValidator control ensures that the required field is not empty. It is generally tied to a text box to force input into the text box.

The syntax of the control is as given:

```
<asp:RequiredFieldValidator ID="rfvcandidate"  runat="server" ControlToValidate ="ddlcandidate"
 ErrorMessage="Please choose a candidate"  InitialValue="Please choose a candidate">
 </asp:RequiredFieldValidator>
```

RangeValidator Control

The RangeValidator control verifies that the input value falls within a predetermined range.

It has three specific properties:

Properties	Description

Type	It defines the type of the data. The available values are: Currency, Date, Double, Integer, and String.
MinimumValue	It specifies the minimum value of the range.
MaximumValue	It specifies the maximum value of the range.

The syntax of the control is as given:

```
<asp:RangeValidator ID="rvclass" runat="server" ControlToValidate="txtclass"    ErrorMessage="Enter your class (6 - 12)" MaximumValue="12"    MinimumValue="6" Type="Integer">

  </asp:RangeValidator>
```

CompareValidator Control

The CompareValidator control compares a value in one control with a fixed value or a value in another control.

It has the following specific properties:

Properties	Description
Type	It specifies the data type.
ControlToCompare	It specifies the value of the input control to compare with.
ValueToCompare	It specifies the constant value to compare with.
Operator	It specifies the comparison operator, the available values are: Equal, NotEqual, GreaterThan, GreaterThanEqual, LessThan, LessThanEqual, and DataTypeCheck.

The basic syntax of the control is as follows:

```
<asp:CompareValidatorID="CompareValidator1" runat="server"  ErrorMessage="CompareValidator">

  </asp:CompareValidator>
```

RegularExpressionValidator

The RegularExpressionValidator allows validating the input text by matching against a pattern of a regular expression. The regular expression is set in the ValidationExpression property.

The following table summarizes the commonly used syntax constructs for regular expressions:

Character Escapes	Description
\b	Matches a backspace.
\t	Matches a tab.

\r	Matches a carriage return.
\v	Matches a vertical tab.
\f	Matches a form feed.
\n	Matches a new line.
\	Escape character.

Apart from single character match, a class of characters could be specified that can be matched, called the metacharacters.

Metacharacters	Description
.	Matches any character except \n.
[abcd]	Matches any character in the set.
[^abcd]	Excludes any character in the set.
[2-7a-mA-M]	Matches any character specified in the range.
\w	Matches any alphanumeric character and underscore.
\W	Matches any non-word character.
\s	Matches whitespace characters like, space, tab, new line etc.
\S	Matches any non-whitespace character.
\d	Matches any decimal character.
\D	Matches any non-decimal character.

Quantifiers could be added to specify number of times a character could appear.

Quantifier	Description
*	Zero or more matches.
+	One or more matches.
?	Zero or one matches.
{N}	N matches.
{N,}	N or more matches.
{N,M}	Between N and M matches.

The syntax of the control is as given:

```
<asp:RegularExpressionValidator ID="string" runat="server" ErrorMessage="string"
ValidationExpression="string" ValidationGroup="string">

  </asp:RegularExpressionValidator>
```

CustomValidator

The CustomValidator control allows writing application specific custom validation routines for both the client side and the server side validation.

The client side validation is accomplished through the ClientValidationFunction property. The client side validation routine should be written in a scripting language, such as JavaScript or VBScript, which the browser can understand.

The server side validation routine must be called from the control's ServerValidate event handler. The server side validation routine should be written in any .Net language, like C# or VB.Net.

The basic syntax for the control is as given:

```
<asp:CustomValidator ID="CustomValidator1" runat="server" ClientValidationFunction=.cvf_func.
ErrorMessage="CustomValidator">

  </asp:CustomValidator>
```

ValidationSummary

The ValidationSummary control does not perform any validation but shows a summary of all errors in the page. The summary displays the values of the ErrorMessage property of all validation controls that failed validation.

The following two mutually inclusive properties list out the error message:

- **ShowSummary** : shows the error messages in specified format.
- **ShowMessageBox** : shows the error messages in a separate window.

The syntax for the control is as given:

```
<asp:ValidationSummary ID="ValidationSummary1" runat="server" DisplayMode = "BulletList"
ShowSummary = "true" HeaderText="Errors:" />
```

Validation Groups

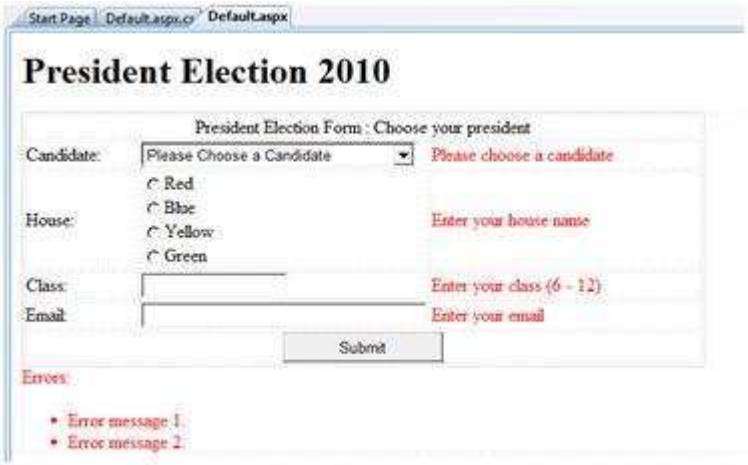
Complex pages have different groups of information provided in different panels. In such situation, a need might arise for performing validation separately for separate group. This kind of situation is handled using validation groups.

To create a validation group, you should put the input controls and the validation controls into the same logical group by setting their *ValidationGroup*property.

Example

The following example describes a form to be filled up by all the students of a school, divided into four houses, for electing the school president. Here, we use the validation controls to validate the user input.

This is the form in design view:



The content file code is as given:

```
<form id="form1" runat="server">

  <table style="width: 66%;">

    <tr>
      <td class="style1" colspan="3" align="center">
      <asp:Label ID="lblmsg"
        Text="President Election Form : Choose your president"
        runat="server" />
      </td>
    </tr>

    <tr>
      <td class="style3">
        Candidate:
      </td>
```

```
    <td class="style2">
      <asp:DropDownList ID="ddlcandidate" runat="server"  style="width:239px">
        <asp:ListItem>Please Choose a Candidate</asp:ListItem>
        <asp:ListItem>M H Kabir</asp:ListItem>
        <asp:ListItem>Steve Taylor</asp:ListItem>
        <asp:ListItem>John Abraham</asp:ListItem>
        <asp:ListItem>Venus Williams</asp:ListItem>
      </asp:DropDownList>
    </td>
    <td>
      <asp:RequiredFieldValidator ID="rfvcandidate"
        runat="server" ControlToValidate ="ddlcandidate"
        ErrorMessage="Please choose a candidate"
        InitialValue="Please choose a candidate">
      </asp:RequiredFieldValidator>
    </td>
  </tr>

  <tr>
    <td class="style3">
      House:
    </td>
    <td class="style2">
      <asp:RadioButtonList ID="rblhouse" runat="server" RepeatLayout="Flow">
        <asp:ListItem>Red</asp:ListItem>
        <asp:ListItem>Blue</asp:ListItem>
        <asp:ListItem>Yellow</asp:ListItem>
        <asp:ListItem>Green</asp:ListItem>
```

```
      </asp:RadioButtonList>
    </td>
    <td>
      <asp:RequiredFieldValidator ID="rfvhouse" runat="server"
        ControlToValidate="rblhouse" ErrorMessage="Enter your house name" >
      </asp:RequiredFieldValidator>
      <br />
    </td>
  </tr>

  <tr>
    <td class="style3">
      Class:
    </td>
    <td class="style2">
      <asp:TextBox ID="txtclass" runat="server"></asp:TextBox>
    </td>
    <td>
      <asp:RangeValidator ID="rvclass"
        runat="server" ControlToValidate="txtclass"
        ErrorMessage="Enter your class (6 - 12)" MaximumValue="12"
        MinimumValue="6" Type="Integer">
      </asp:RangeValidator>
    </td>
  </tr>
  <tr>
    <td class="style3">
      Email:
    </td>
```

```
      <td class="style2">
        <asp:TextBox ID="txtemail" runat="server" style="width:250px">
        </asp:TextBox>
      </td>
      <td>
        <asp:RegularExpressionValidator ID="remail" runat="server"
          ControlToValidate="txtemail" ErrorMessage="Enter your email"
          ValidationExpression="\w+([-+.']\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*">
        </asp:RegularExpressionValidator>
      </td>
    </tr>
    <tr>
      <td class="style3" align="center" colspan="3">
        <asp:Button ID="btnsubmit" runat="server" onclick="btnsubmit_Click"
          style="text-align: center" Text="Submit" style="width:140px" />
      </td>
    </tr>
  </table>
  <asp:ValidationSummary ID="ValidationSummary1" runat="server"
    DisplayMode ="BulletList" ShowSummary ="true" HeaderText="Errors:" />
</form>
```

The code behind the submit button:

```
protected void btnsubmit_Click(object sender, EventArgs e)
{
  if (Page.IsValid)
  {
    lblmsg.Text = "Thank You";
  }
```

```
  else
  {
    lblmsg.Text = "Fill up all the fields";
  }
}
```

Improve - Efficiency and Scalability of ASP.NET Apps

In today's scenario developing applications is not a big deal, but increasing the performance-efficiency and scalability is the big challenge faced by the developers.

Consider the following sections present these tips for ensuring application efficiency and scalability.

- Move processing into Components
- Programming languages and web frameworks e.g. Spring, NHibernet, MVC
- Web applications and data access
- Software Architecture – 2-tier or 3 tier
- Avoid large pages
- Remove dead code.
- Avoid extra tips to the server.

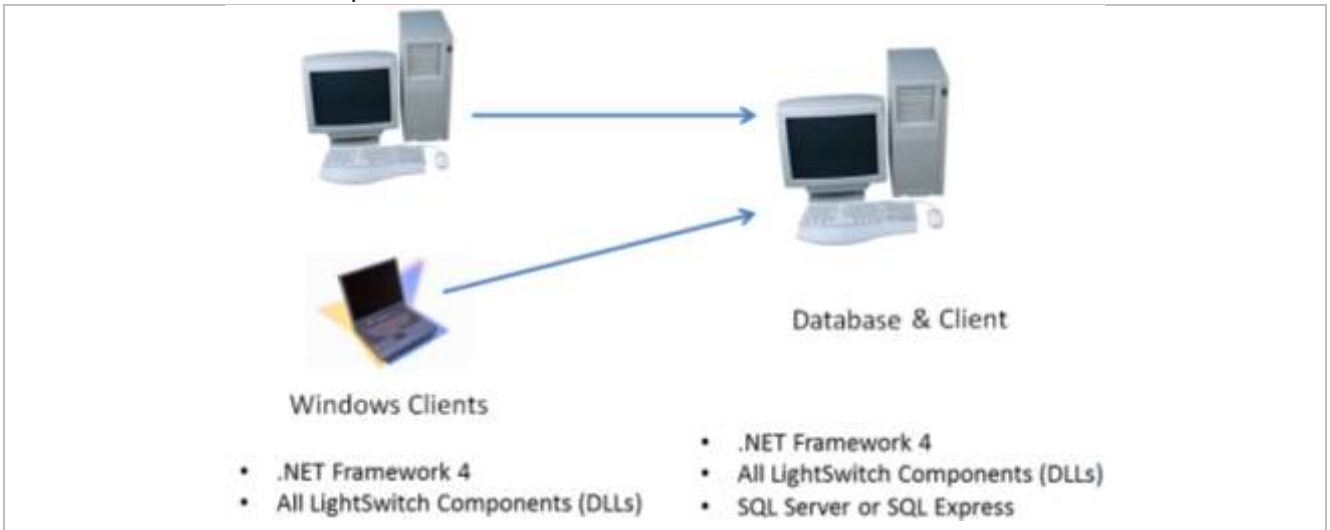


FIG. 2-TIER ARCHITECTURE

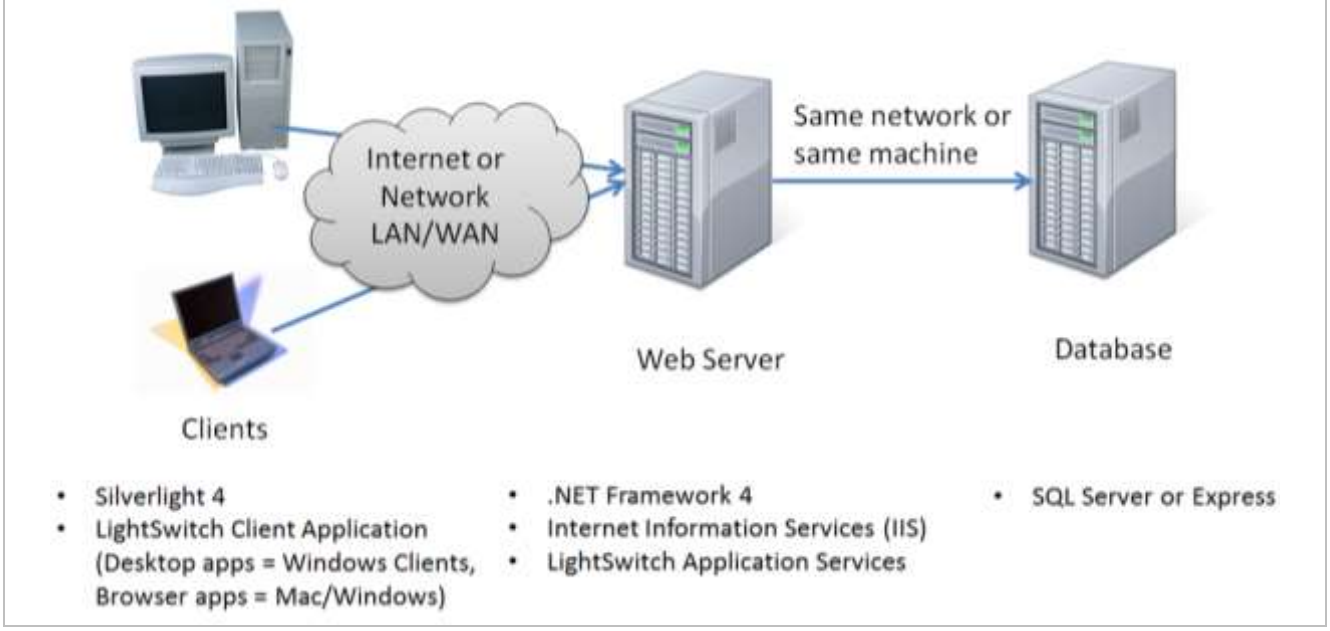


FIG. 3-TIER ARCHITECTURE		
FEATURE	**2-TIER**	**3-TIER**
ADMINISTRATION	Complex as business logic and database exist on one system	Simple as layers are separated
SECURITY	Less secured as client can talk to database directly	Highly secured as client is not allowed to talk to database directly
SCALABILITY	Poor	Excellent as requests can be load balanced between servers
SERVICES	Not required as business logic application and database are tight-coupled.	Application server may use services like RMI or JMS etc.
PERFORMANCE	Poor as second-tier should execute both business logic and database queries	Good as processing is distributed between 2nd and 3rd tiers.
DEPLOYMENT COSTS	High as no granularity exists	Less and ease due to builder tools
INFRASTRUCTURE	No extra infrastructure is required due to tight-coupling	Infrastructure of middleware services can be used
REUSABILITY	Mostly clients are monolothic and thereby reusability not possible	Reusability more with services implementation
LEGACY INTEGRATION	No	Yes, possible through gateways and business objects
HARDWARE FLEXIBILITY	Very limited as only one client and one server exist	Best with component-based environments to distribute services between 2nd and 3rd tiers
HETEROGENEOUS DATABASE SUPPORT	Only database should be used	Multiple databases can be used even within the same transaction
SUPPORT FOR INTERNET	Poor as bandwidth limitations may take more time for download fat client programs	Best to download client programs as services supporting downloading are distributed
AVAILABILITY	Can use backup servers	If one server fails, the services can be executed on other services
COMMUNICATION CHOICES	Only connection-oriented and synchronous methods calls are permitted	Can support connectionless messaging, publish-subscribe and broadcast messaging etc.

Move processing into Components

You can increase application efficiency by moving processing into components and running those components as COM+ applications in Microsoft Transaction Server (MTS).

The efficiency gain occurs because—unlike your Web Forms—MTS doesn't have to instantiate a new component for each request; it instantiates components on demand but then keeps them in memory, ready for use, for a defined period of time. With that said, you can find a greater speed increase by caching .NET classes in memory at Application scope than in building MTS components.

Bear in mind that there's a performance penalty for running the components in a separate process, particularly the first time your application instantiates a component. For example, when your application uses a component only rarely, it may not be worth moving that component into MTS, because the performance and start up penalty may be greater than the delay associated with using the component to begin with.

Also, remember that an instantiated component uses server resources. If you only need the component periodically, you may not want to waste server resources on it during the bulk of the

application time, when the application isn't using the component anyway. You can set a timeout value for MTS applications. MTS destroys component instances when they haven't been used for a specified amount of time.

Avoid large Pages

Try to avoid creating large pages unless you can cache the entire page. Instead, break the page logic up into smaller chunks—particularly if you can reuse those chunks in other pages in your application.

Try to design pages that require user interaction so that they fit on a single screen without scrolling. Although most users have become inured to long pages for reading information, it's extremely irritating to fill out a form only to find that the Submit button—and perhaps a few "hidden" fields—can be found only by scrolling around the page.

Remove Dead Code

As applications grow, you'll often find that older, unused code remains in the application. Even though your application may never use the code, the server must still compile it. Find and remove unused methods, variable declarations, components, Web Forms, Web Services, and client-side code blocks.

Avoid extra trips to server

As a final note, the greatest performance bottleneck in a Web application is usually the Web itself. This isn't as great a problem for intranet applications as it is for Internet applications; nonetheless, performing an action on the local machine is almost always faster than making a call to the server. Therefore, try to plan your applications so that you perform as much processing as possible locally, between server round trips.

One way you can do that is by downloading the interface to the local machine in advance and then retrieving just the data from the server itself. Of course, that's what Web applications were all about; by using a browser to draw the UI, you could maintain the entire application centrally, with no deployment or installation required. But another model is equally powerful and even better suited for many types of applications—the Windows UI running in conjunction with Web Services. Like browser applications, .NET Windows applications can download a very small amount of user-interface code at runtime; in other words, you can deploy your application dynamically, on demand, via HTTP to remote clients. You can even deploy the .NET framework on demand, although that requires broadband clients because the framework itself is (in the first release) approximately 15 megabytes in size. However, with many if not most corporate applications, businesses can ensure that the client machines have the .NET framework installed in advance. With the framework in place, downloading the Windows Forms and logic required for an application's user interface is usually a very small download—in the thousands of bytes rather than the megabyte range.

That combination means that it's now both possible and efficient to deliver Web applications with Windows Forms front-ends that use Web Services to supply the data. By doing this, you gain more control over the application's interface, and the application looks, feels, and acts like the familiar applications people use on a daily basis. After all, there's little point in transmitting UI content over and over again via the browser if the user can run a similar but more powerful interface in native code.

Finally, as you've seen, planning your application around Web Services means you can get the customized UI available with Windows .NET-capable clients, take nearly instant advantage of your existing COM components, and *still* deliver the same application to browser-based clients or clients running customized user interfaces on other platforms. My advice is this: If you have any inkling that you may ever need or want to deliver an application to non-browser clients, plan the dynamic portions of your applications as Web Services whenever practicable, starting immediately.

ASP.NET Server Control

Controls are small building blocks of the graphical user interface, which include text boxes, buttons, check boxes, list boxes, labels, and numerous other tools. Using these tools, the users can enter data, make selections and indicate their preferences.

Controls are also used for structural jobs, like validation, data access, security, creating master pages, and data manipulation.

ASP.NET uses five types of web controls, which are:

- HTML controls
- HTML Server controls
- ASP.NET Server controls
- ASP.NET Ajax Server controls
- User controls and custom controls

ASP.NET server controls are the primary controls used in ASP.NET. These controls can be grouped into the following categories:

- **Validation controls** - These are used to validate user input and they work by running client-side script.
- **Data source controls** - These controls provides data binding to different data sources.
- **Data view controls** - These are various lists and tables, which can bind to data from data sources for displaying.
- **Personalization controls** - These are used for personalization of a page according to the user preferences, based on user information.
- **Login and security controls** - These controls provide user authentication.
- **Master pages** - These controls provide consistent layout and interface throughout the application.
- **Navigation controls** - These controls help in navigation. For example, menus, tree view etc.
- **Rich controls** - These controls implement special features. For example, AdRotator, FileUpload, and Calendar control.

The syntax for using server controls is:

```
<asp:controlType  ID ="ControlID" runat="server" Property1=value1  [Property2=value2] />
```

In addition, visual studio has the following features, to help produce in error-free coding:

- Dragging and dropping of controls in design view
- IntelliSense feature that displays and auto-completes the properties

- The properties window to set the property values directly

Properties of the Server Controls

ASP.NET server controls with a visual aspect are derived from the WebControl class and inherit all the properties, events, and methods of this class.

The WebControl class itself and some other server controls that are not visually rendered are derived from the System.Web.UI.Control class. For example, PlaceHolder control or XML control.

ASP.Net server controls inherit all properties, events, and methods of the WebControl and System.Web.UI.Control class.

The following table shows the inherited properties, common to all server controls:

Property	Description
AccessKey	Pressing this key with the Alt key moves focus to the control.
Attributes	It is the collection of arbitrary attributes (for rendering only) that do not correspond to properties on the control.
BackColor	Background color.
BindingContainer	The control that contains this control's data binding.
BorderColor	Border color.
BorderStyle	Border style.
BorderWidth	Border width.
CausesValidation	Indicates if it causes validation.
ChildControlCreated	It indicates whether the server control's child controls have been created.
ClientID	Control ID for HTML markup.
Context	The HttpContext object associated with the server control.
Controls	Collection of all controls contained within the control.
ControlStyle	The style of the Web server control.
CssClass	CSS class
DataItemContainer	Gets a reference to the naming container if the naming container implements IDataItemContainer.
DataKeysContainer	Gets a reference to the naming container if the naming container implements IDataKeysControl.

DesignMode	It indicates whether the control is being used on a design surface.
DisabledCssClass	Gets or sets the CSS class to apply to the rendered HTML element when the control is disabled.
Enabled	Indicates whether the control is grayed out.
EnableTheming	Indicates whether theming applies to the control.
EnableViewState	Indicates whether the view state of the control is maintained.
Events	Gets a list of event handler delegates for the control.
Font	Font.
Forecolor	Foreground color.
HasAttributes	Indicates whether the control has attributes set.
HasChildViewState	Indicates whether the current server control's child controls have any saved view-state settings.
Height	Height in pixels or %.
ID	Identifier for the control.
IsChildControlStateCleared	Indicates whether controls contained within this control have control state.
IsEnabled	Gets a value indicating whether the control is enabled.
IsTrackingViewState	It indicates whether the server control is saving changes to its view state.
IsViewStateEnabled	It indicates whether view state is enabled for this control.
LoadViewStateById	It indicates whether the control participates in loading its view state by ID instead of index.
Page	Page containing the control.
Parent	Parent control.
RenderingCompatibility	It specifies the ASP.NET version that the rendered HTML will be compatible with.
Site	The container that hosts the current control when rendered on a design surface.
SkinID	Gets or sets the skin to apply to the control.

Style	Gets a collection of text attributes that will be rendered as a style attribute on the outer tag of the Web server control.
TabIndex	Gets or sets the tab index of the Web server control.
TagKey	Gets the HtmlTextWriterTag value that corresponds to this Web server control.
TagName	Gets the name of the control tag.
TemplateControl	The template that contains this control.
TemplateSourceDirectory	Gets the virtual directory of the page or control containing this control.
ToolTip	Gets or sets the text displayed when the mouse pointer hovers over the web server control.
UniqueID	Unique identifier.
ViewState	Gets a dictionary of state information that saves and restores the view state of a server control across multiple requests for the same page.
ViewStateIgnoreCase	It indicates whether the StateBag object is case-insensitive.
ViewStateMode	Gets or sets the view-state mode of this control.
Visible	It indicates whether a server control is visible.
Width	Gets or sets the width of the Web server control.

Methods of the Server Controls

The following table provides the methods of the server controls:

Method	Description
AddAttributesToRender	Adds HTML attributes and styles that need to be rendered to the specified HtmlTextWriterTag.
AddedControl	Called after a child control is added to the Controls collection of the control object.
AddParsedSubObject	Notifies the server control that an element, either XML or HTML, was parsed, and adds the element to the server control's control collection.
ApplyStyleSheetSkin	Applies the style properties defined in the page style sheet to the control.

ClearCachedClientID	Infrastructure. Sets the cached ClientID value to null.
ClearChildControlState	Deletes the control-state information for the server control's child controls.
ClearChildState	Deletes the view-state and control-state information for all the server control's child controls.
ClearChildViewState	Deletes the view-state information for all the server control's child controls.
CreateChildControls	Used in creating child controls.
CreateControlCollection	Creates a new ControlCollection object to hold the child controls.
CreateControlStyle	Creates the style object that is used to implement all style related properties.
DataBind	Binds a data source to the server control and all its child controls.
DataBind(Boolean)	Binds a data source to the server control and all its child controls with an option to raise the DataBinding event.
DataBindChildren	Binds a data source to the server control's child controls.
Dispose	Enables a server control to perform final clean up before it is released from memory.
EnsureChildControls	Determines whether the server control contains child controls. If it does not, it creates child controls.
EnsureID	Creates an identifier for controls that do not have an identifier.
Equals(Object)	Determines whether the specified object is equal to the current object.
Finalize	Allows an object to attempt to free resources and perform other cleanup operations before the object is reclaimed by garbage collection.
FindControl(String)	Searches the current naming container for a server control with the specified id parameter.
FindControl(String, Int32)	Searches the current naming container for a server control with the specified id and an integer.
Focus	Sets input focus to a control.
GetDesignModeState	Gets design-time data for a control.

GetType	Gets the type of the current instance.
GetUniqueIDRelativeTo	Returns the prefixed portion of the UniqueID property of the specified control.
HasControls	Determines if the server control contains any child controls.
HasEvents	Indicates whether events are registered for the control or any child controls.
IsLiteralContent	Determines if the server control holds only literal content.
LoadControlState	Restores control-state information.
LoadViewState	Restores view-state information.
MapPathSecure	Retrieves the physical path that a virtual path, either absolute or relative, maps to.
MemberwiseClone	Creates a shallow copy of the current object.
MergeStyle	Copies any nonblank elements of the specified style to the web control, but does not overwrite any existing style elements of the control.
OnBubbleEvent	Determines whether the event for the server control is passed up the page's UI server control hierarchy.
OnDataBinding	Raises the data binding event.
OnInit	Raises the Init event.
OnLoad	Raises the Load event.
OnPreRender	Raises the PreRender event.
OnUnload	Raises the Unload event.
OpenFile	Gets a Stream used to read a file.
RemovedControl	Called after a child control is removed from the controls collection of the control object.
Render	Renders the control to the specified HTML writer.
RenderBeginTag	Renders the HTML opening tag of the control to the specified writer.
RenderChildren	Outputs the contents of a server control's children to a provided HtmlTextWriter object, which writes the contents to be rendered on the client.

RenderContents	Renders the contents of the control to the specified writer.
RenderControl(HtmlTextWriter)	Outputs server control content to a provided HtmlTextWriter object and stores tracing information about the control if tracing is enabled.
RenderEndTag	Renders the HTML closing tag of the control into the specified writer.
ResolveAdapter	Gets the control adapter responsible for rendering the specified control.
SaveControlState	Saves any server control state changes that have occurred since the time the page was posted back to the server.
SaveViewState	Saves any state that was modified after the TrackViewState method was invoked.
SetDesignModeState	Sets design-time data for a control.
ToString	Returns a string that represents the current object.
TrackViewState	Causes the control to track changes to its view state so that they can be stored in the object's view state property.

Example

Let us look at a particular server control - a tree view control. A Tree view control comes under navigation controls. Other Navigation controls are: Menu control and SiteMapPath control.

Add a tree view control on the page. Select Edit Nodes... from the tasks. Edit each of the nodes using the Tree view node editor as shown:

Once you have created the nodes, it looks like the following in design view:

The AutoFormat... task allows you to format the tree view as shown:

Add a label control and a text box control on the page and name them lblmessage and txtmessage respectively.

Write a few lines of code to ensure that when a particular node is selected, the label control displays the node text and the text box displays all child nodes under it, if any. The code behind the file should look like this:

```
using System;

using System.Collections;

using System.Configuration;

using System.Data;

using System.Linq;
```

```
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.HtmlControls;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;

using System.Xml.Linq;

namespace eventdemo {
  public partial class treeviewdemo : System.Web.UI.Page {

    protected void Page_Load(object sender, EventArgs e) {
      txtmessage.Text = " ";
    }

    protected void TreeView1_SelectedNodeChanged(object sender, EventArgs e) {

      txtmessage.Text = " ";
      lblmessage.Text = "Selected node changed to: " + TreeView1.SelectedNode.Text;
      TreeNodeCollection childnodes = TreeView1.SelectedNode.ChildNodes;

      if(childnodes != null) {
        txtmessage.Text = " ";

        foreach (TreeNode t in childnodes) {
          txtmessage.Text += t.Value;
        }
```

```
      }
    }
  }
}
```

Execute the page to see the effects. You will be able to expand and collapse the nodes.