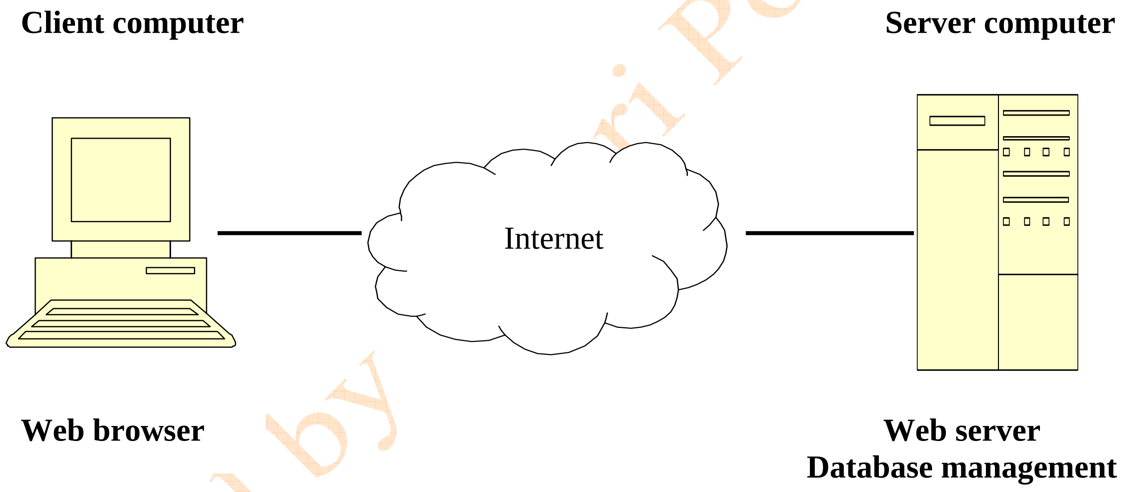
An introduction to Web application

A ***Web Application*** consists of a set of ***Web Pages*** that are generated in response to user request. The Internet has many different types of web applications, such as search engines, online stores, auctions, new sites, discussions group, games, and so on.

An important point to notice about web application that it consist of web pages which contain controls that let the user interact with the page. *A page that contains controls like these is called a **Web Form***.

Hardware and software components for web applications

Figure below shows the basic hardware and software components that are required for a web application.



A Web application is a type of ***client/server application***, which means that the functions of the application are split between a client computer and a server computer. The client and server computers are connected to one another via the Internet, and communicate with each other using ***HTTP***, or ***Hypertext Transfer Protocol***.

To access a web application, you use a ***web browser*** that runs on the client computer.

The web application itself is stored on the server computer. This computer runs special ***web server*** software that enables it to send web pages to web browser. Although many web servers are available, the two most popular are Microsoft's ***Internet Information Services*** (or ***IIS***) and The Apache Software Foundation's ***Apache HTTP Server***, usually just called ***Apache***. For ASP.Net applications, the server computer must run IIS. In addition, it must have Microsoft's .Net Framework software installed.

Because most web applications work with data that's stored in a database, most server computers also run a database management system (or ***DBMS***). Although ASP.Net applications require Microsoft's web server software, you can use any vendor's DBMS. Two popular database management systems for ASP.Net development are Microsoft SQL Server and Oracle. The database server software doesn't have to run on the same server as the web server software. In fact, a separate database server is often used to improve an application's overall performance.

Although the picture above shows the client and server computers connected via the Internet, this is not only one way a client can connect to a server in a web application. If the client and the server are on the same local area network, they can connect via an ***Intranet***. Since an Intranet uses the same protocol as the Internet, a web application works the same on an Intranet as it does on the Internet.

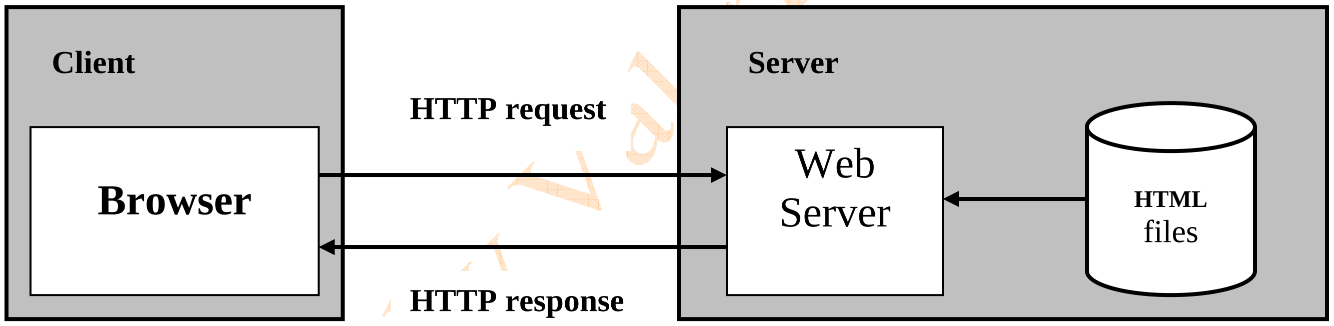
You can also run the web browser and web server software on the same computer so that one computer functions as both the client and server. This configuration is typical for the labs in the university for the students and for Internet developers.

You should realize that the web pages that make up a web application are defined using ***HTML***, or ***Hypertext Markup Language***.

How static web pages work

Many of the web pages on the Internet are ***static web pages***. Static pages are HTML documents that are the same each time they are viewed – they don't change in response to user input.

Figure below shows a web server handles static web pages.



The process begins when a user at a web browser requests a web page. This occur when the user enters at a web address, called a ***URL*** (***Uniform resource Locator***), into the browser's address box or when the user clicks a link that leads to another page. In either case, the web browser uses HTTP to send an ***HTTP request*** to the web server.

When the web server receives an HTTP request from the browser, the server retrieves the requested HTML file from the disk and sends the file back to the browser in the form of an ***HTTP response***.

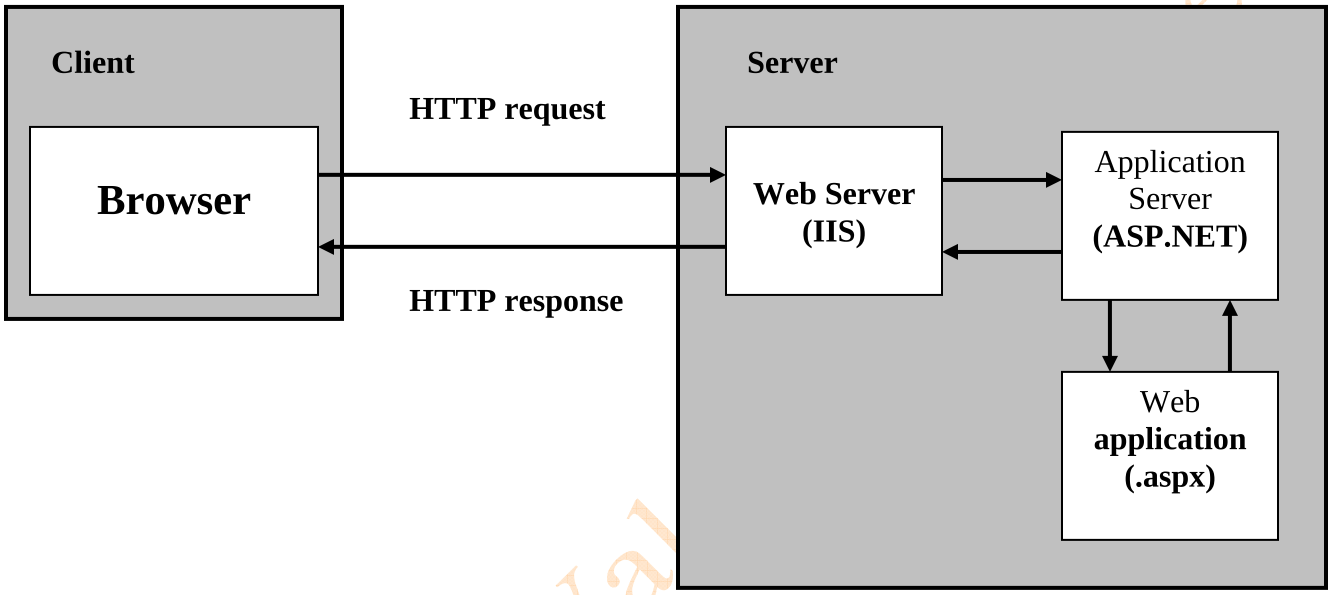
When the browser receives the HTTP response, it formats and displays the HTML document. Then, the user can view the content.

How dynamic web pages work

A web application consists of one or more web pages that are not static, but that can change in some way each time the page is displayed. Instead of being stored on the disk in the form of HTML files, the pages of a web application are generated dynamically by the application. As a result, the generated web pages are referred to as ***dynamic web pages***.

One of the key differences between static web pages and dynamic web pages is that dynamic web pages are defined by web forms, which contain a collections of a web controls, such as labels, text boxes, and buttons. Users work with these controls to interact with the application.

Figure below shows the basic processing for a web application.



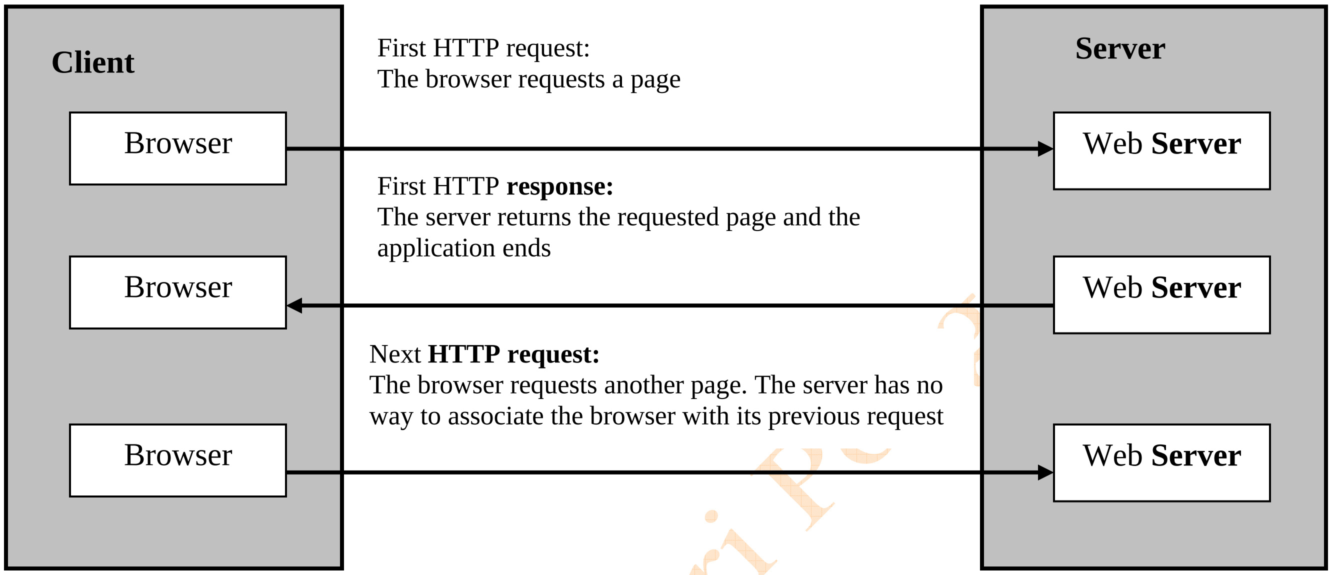
Typically, the users enter information into one or more form controls and then click on a button. That causes the browser to send an HTTP request to the server that contains the address of the web page being requested, along with the information the user entered into the form. When the web server receives this request, it determines that it is a request for a web application rather than for a static web page. As a result, the web server passes the request on to an ***application server*** for processing. The application server, in turn, manages the execution of the web application.

To determine if the request is for a static page or a dynamic page generated by a web application, the web server looks up the extension of the requested page in a list of ***application mappings***. These mapping indicate what program a file extension of *htm* or *html*. In contrast, a dynamic page created by an ASP.NET application has an extension of aspx. As a result, when the web server receives an HTTP request for an `aspx` file, the server knows to pass this request along to ASP.NET.

When the application is executed, it processes the information the user entered and generates an HTML document. The actual content of the HTML document depends on the application. If the application displays data from a database, for example, it queries the database to obtain the requested information. Then, it generates a page with that information, which is requested by the application server to the web browser. The web server, in turn, sends the page back to the browser in the form of an HTTP response, and the browser displays the page. This entire process that begins with the browser requesting a web page and ends with the page being sent back to the client is called a ***round trip***.

The importance of state in Web applications

After an application generates a web page, the application ends. That means that the current status of any data maintained by the application, such as variables or control properties, is lost. It means that HTTP doesn't maintain the state of the application. You can see this in figure below:


Client
Server
First HTTP request:
The browser requests a page
Browser
Web Server
First HTTP response:
The server returns the requested page and the application ends
Browser
Web Server
Next HTTP request:
The browser requests another page. The server has no way to associate the browser with its previous request
Browser
Web Server

Concepts

- ***State*** refers to the status of the properties, variables, and other data maintained by an application for a single user. The application must maintain a separate state for each user currently accessing the application
- HTTP is a ***stateless*** protocol. That means that it doesn't keep track of state between round trips. Once a browser makes a request and receives a response, the application terminates and its state is lost.

Although HTTP doesn't maintain state, ASP.Net provides several ways to do that. There are three of the most common techniques:

1. You can use ***view state*** to maintain the values of form control properties. For example, you can use it to preserve the Text property of label controls that change as the program executes or to maintain a list of items in a list box or a drop-down list. Because ASP.Net implements view state by default, you don't need to write any special code to use it.
2. You can also use ***session state*** to maintain data between executions of an application. When a user starts a session, ASP.Net creates a ***session state object*** that's sent back and forth between the server and the browser. This object contains a session ID that the server can use to identify the session. In addition, you can add your own items to this object so that their previous values are available each time the program is executed.

Application state is an ASP.Net feature that lets you save state information so it can be shared by all the users of an application. For example, you can use application state to maintain global counters or to maintain a list of the users who are currently loggod on to the application.

ASP.NET 2.0 can also maintain a ***profile*** for each user of an application. Because profiles are stored in a database, the profile data is maintained from one user session to another. This makes it easier to personalize an application.

An introduction to ASP.NET application development

The software you need

Software requirements for ASP.NET application development

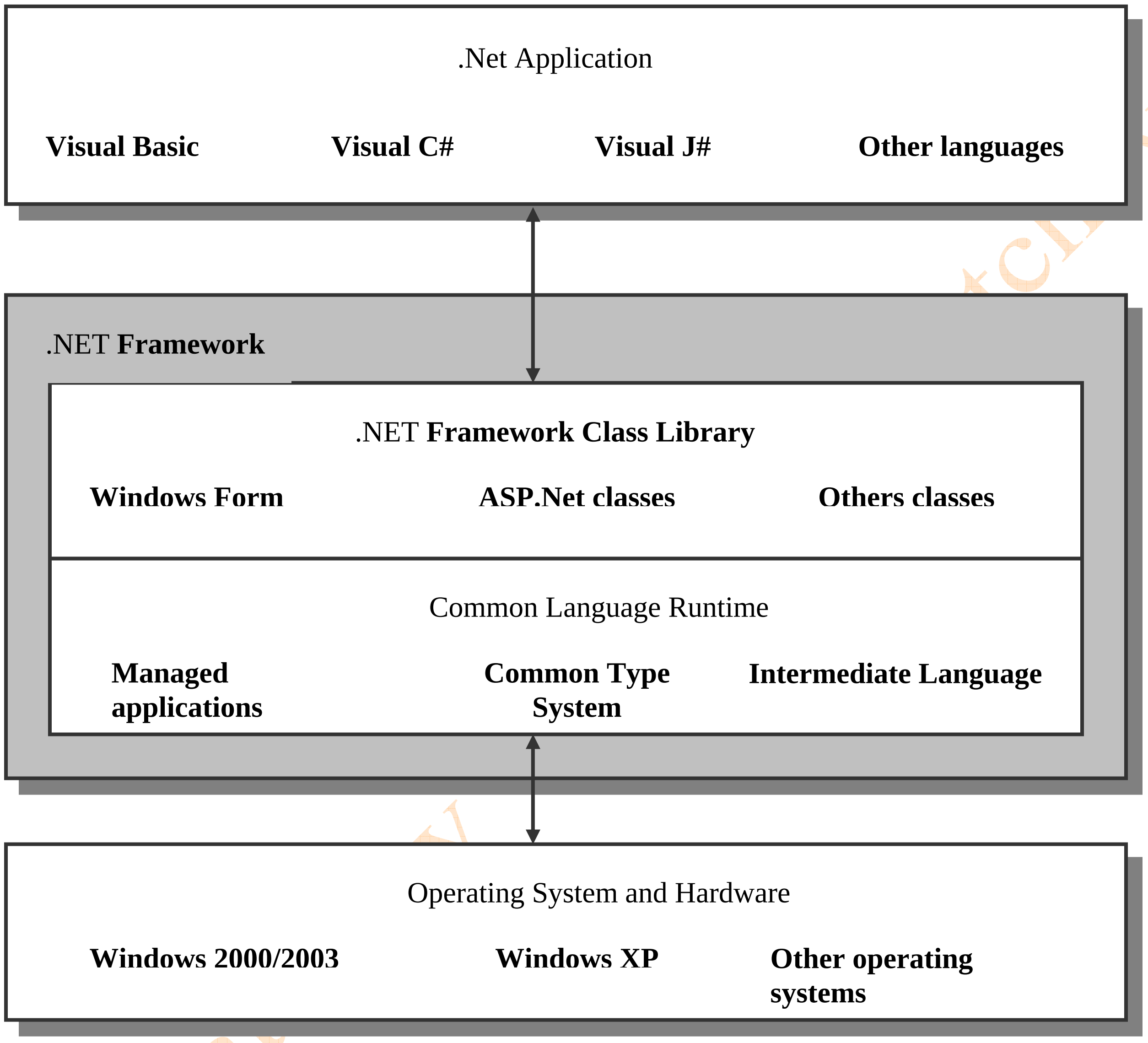
Client	Server
Windows 2000 or later	Windows 2000 or later
Microsoft .NET Framework 2.0	Microsoft .NET Framework 2.0
A browser like Internet Explorer (6.0 or later)	Internet Information Services 5.0 or later (6.0 recommended)
Visual Studio 2005	Microsoft SQL Server or equivalent database. FrontPage Server Extensions (for remote development only)

Visual Studio 2005 Editions

Edition	Description
Visual Studio 2005 Standard Edition	Supports Windows and Web development using Visual Basic, C#, C++, and J#
Visual Studio 2005 Professional Edition	Same as Standard Edition with several; additional features such an additional development options and integration with SQL Server 2005
Visual Studio 2005 Team System	The top-of-the-line version of Visual studio, with special features added to support large development team
Visual Web Developer 2005 Express Edition	Inexpensive web development in Visual Basic, C#, or J# for hobbyist and novices

The components of the .NET Framework

Figure below shows the major components that make up the .NET Framework.



The .NET Framework is divided into two main components:

- The .NET Framework Class Library
- The Common Language Runtime

The ***.NET Framework Class Library*** consists of classes that provide many of the functions that you need for developing .Net applications. For instance structure, the ASP.NET classes are used for developing ASP.NET web application. For standard Windows applications, you work with the Windows Forms classes. Other classes let you work with databases, manage security, access files, and perform many other functions.

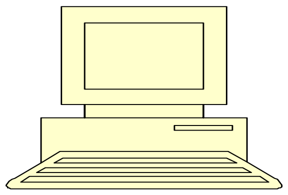
The classes in the .NET Framework Class Library are organized in a hierarchical structure. Within this structure, related classes are organized into groups called ***namespaces***. Each namespaces contains the classes used to support a particular function. For example, the System.Web namespace contains the classes used to create ASP.NET web applications, and the System.Data namespace contains the classes used to access data.

The ***Common Language Runtime,*** or CLR is the foundation of the .NET Framework. It provides the services that are needed for executing any applications that's developed with one of the .Net languages. This is possible because all of the .Net languages compile to a common ***Microsoft Intermediate Language (MSIL)*** or just ***Intermediate Language***, which is stored on disk in an assembly. The CLR also provides the ***Common Type System*** that defines the data types that are used by the .Net languages. That way, you can use the same data types regardless of what .NET languages you're using to develop your application. Unlike Windows applications, though, all of the forms in a web application must be developed using the same language.

To run an ASP.NET application, the web server must have the .NET Framework installed. However, the client computers that access the web server do not need the .NET Framework. Instead, the client computers can run any client operation system with a modern web browser.

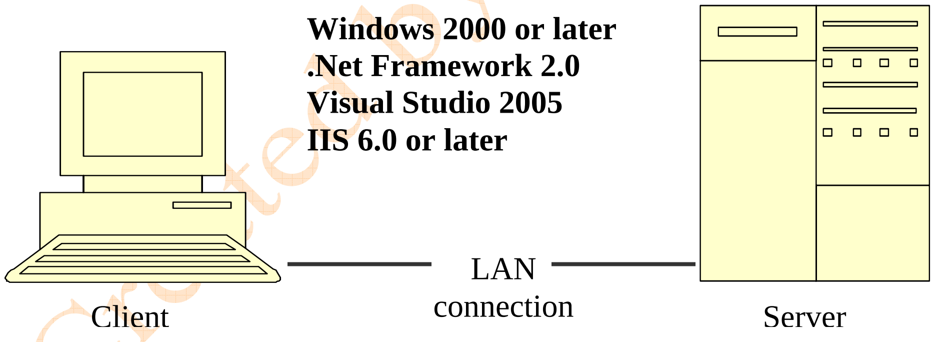
Three environments for developing ASP.Net applications

Standalone development



Windows 2000 or later
.Net Framework 2.0
Visual Studio 2005
Optional: IIS 6.0 or later, SQL Server 2000/2005

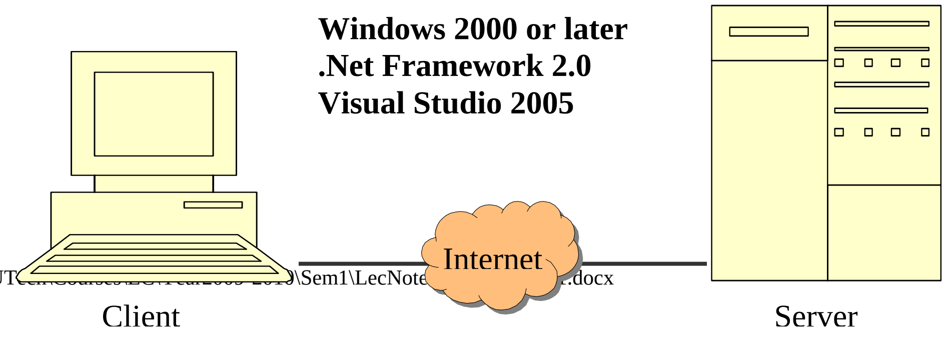
Local area network development



Windows 2000 or later
.Net Framework 2.0
Visual Studio 2005
IIS 6.0 or later

Windows 2000 or later
.Net Framework
Visual Studio 2005
IIS 6.0 or later
SQL Server,
FrontPage Extension

Internet development



Windows 2000 or later
.Net Framework 2.0
Visual Studio 2005

Windows 2000 or later
.Net Framework
IIS 6.0 or later
FTP Server, SQL Server

This picture shows three possible ways to set up a development environment for coding and testing ASP.NET applications.

A standalone environment is ideal if you're working alone on an application and you don't have an available server computer. However, you might also use this environment if two or more programmers are working on the same application. In this case you use a ***Local Area Network (LAN)*** and a client computer communicates with a server computer over the LAN. In that case, the programmers each work in a standalone environment, but master copies of the source files for the application are typically stored on a server and ***source control software*** is used to coordinate access to the master files. Note that the files don't need to be stored on a web server. They simply need to be stored on a server that's accessible over a network.

If your development team is scattered over several sites, you should opt for the third environment in this figure. Here, the clients are connected to the server via the Internet rather than a LAN. The main difference between this type of setup and the LAN setup is that the Internet development setup requires that you install ***FrontPage Server Extension (FPSE)*** on the server.

How an ASP.NET application works

The files used by the ASP.NET Application
For this part of lecturer we use a *Ch03Cart* application.

The aspx files for the *Order* form (Order.aspx)

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

*The first set of tags for each web form defines a **page derictive** that provides for **attributes**. Here is:*

- *Page Language attribute says that the language is C#*
- *CodeFile attribute says that code-behind file is named Oreder.aspx.cs*
- *Inherits attribute specifies the class named Order*

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
        Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

The second set of tags defines a DOCTYPE declaration, which tells the browser what version of HTML the HTML uses.

```
<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
   <title>Shopping Cart</title>
</head>
<body>
   <form id="form1" runat="server">
```

runat attribute with value "server" indicates that the form will be processed on the server by ASP.NET.

```
<div>
```

The content of the web page itself is defined within the div tags, which are within the body and form tags. Notice that form tag includes a runat attribute that's assigned a value of "server". That indicates that the form will be processed on the server by ASP.NET. This attribute is required for all ASP.NETweb pages and all web server controls.The tags within the div tags define the web server controls that appear on the page. Since these controls include the runtime attribute with a value of "server", they will be processed on the server by ASP.NET. The last phase of this processing is ***rendering*** *the controls, which means that the asp code is converted to standard HTML so the page can be displayed by a browser.*

```
<asp:Image ID="Image1" runat="server" ImageUrl="~/Images/banner.jpg" /><br /><br />
<asp:Label ID="Label1" runat="server" Text="Please select a product:"></asp:Label>
<asp:DropDownList ID="ddlProducts" runat="server" Width="150px"
    DataSourceID="AccessDataSource1" DataTextField="Name" DataValueField="ProductID"
    AutoPostBack="True">
```

AutoPostBack attributehas bee set to True. That means that a postback will occur whenever the user selects an item that list. Note that a postback will also occur whenever the user clicks the Add to Cart button. However, the AutoPostBack attribute isn't requiered for this control because performing a posback is the default operation of a button.

Although the asp code for the Add to Cart button doesn't include an AutoPostBack attribute, it does include an OnClick attribute. This attribute names the event handler that's executed when the user clicks the button and the form is posted back to the server.

```
</asp:DropDownList>
<asp:AccessDataSource ID="AccessDataSource1" runat="server"
   DataFile="~/App_Data/Halloween.mdb"
   SelectCommand="SELECT [ProductID], [Name], [ShortDescription], [LongDescription],
                         [ImageFile], [UnitPrice]
                  FROM [Products] ORDER BY [Name]">
</asp:AccessDataSource>
<br />
<table>
   <tr>
      <td style="width: 250px; height: 22px">
         <asp:Label ID="lblName" runat="server" Font-Bold="False" Font-Size="Larger">
         </asp:Label>
      </td>
      <td rowspan="4" style="width: 20px"></td>
      <td rowspan="4" valign="top"><asp:Image ID="imgProduct" runat="server" Height="200"
      /></td>
   </tr>
   <tr>
      <td style="width: 250px">
         <asp:Label ID="lblShortDescription" runat="server"></asp:Label>
      </td>
   </tr>
   <tr>
```

```
            <td style="width: 250px">
               <asp:Label ID="lblLongDescription" runat="server"></asp:Label>
            </td>
         </tr>
         <tr>
            <td style="width: 250px; height: 53px;">
               <asp:Label ID="lblUnitPrice" runat="server" Font-Bold="True" Font-Size="Larger">
               </asp:Label>
               <asp:Label ID="Label2" runat="server" Font-Bold="True"  Font-Size="Larger"
                          Text="each">
               </asp:Label>
            </td>
         </tr>
      </table>
      <br />
      <asp:Label ID="Label3" runat="server" Text="Quantity:" Width="80px"
                 BorderWidth="0px"></asp:Label>
      <asp:TextBox ID="txtQuantity" runat="server" Width="80px"></asp:TextBox>
      <asp:RequiredFieldValidator ID="RequiredFieldValidator1" runat="server"
         ControlToValidate="txtQuantity" Display="Dynamic"
         ErrorMessage="Quantity is a required field.">
      </asp:RequiredFieldValidator>
      <asp:RangeValidator ID="RangeValidator1" runat="server"  ControlToValidate="txtQuantity"
         Display="Dynamic" ErrorMessage="Quantity must range from 1 to 500."
         MaximumValue="500" MinimumValue="1" Type="Integer"></asp:RangeValidator>
         <br /><br />
      <asp:Button ID="btnAdd" runat="server" Text="Add to Cart"  OnClick="btnAdd_Click" />&nbsp;
```

OnClick attribute names the event handler that's executed when the user clicks the button and the form is posted back to the server.

```
      <asp:Button ID="btnCart" runat="server" CausesValidation="False"  PostBackUrl="~/Cart.aspx"
         Text="Go to Cart" />
```

The PostBackUrl attribute provides the path for the Cart.aspx file, which means that the Cart form will be requested when this button is clicked. That's one way to switch from one form to another as an application is run, and you'll see another in the next figure.

```
   </div>
   </form>
</body>
</html>
```

The *C#* code for the *Order* form

```
using System;
using System.Data;
using System.Configuration;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Web.UI.HtmlControls;
using System.Collections;

public partial class Order : System.Web.UI.Page
```

Because this declaration includes the partial keyword, this is a partial class that must be combined with another partial class when it's complied. The class declaration also indicates that the class inherits the System.Web.UI.Page class, which is the .NET class that provides all of the basic functionality of ASP.NET pages.

```
{
   private Product selectedProduct;

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

Each time the page is requested, ASP.NET initializes it and raises the Load event, which is handled by the Page_Load event handler. This event handler uses the IsPostBack property of the page to determine whether the page is being posted back to the server from the same page.

```
   {
      if (!IsPostBack)
         ddlProducts.DataBind();

      selectedProduct = this.GetSelectedProduct();
      lblName.Text = selectedProduct.Name;
      lblShortDescription.Text = selectedProduct.ShortDescription;
      lblLongDescription.Text = selectedProduct.LongDescription;
      lblUnitPrice.Text = selectedProduct.UnitPrice.ToString("c");
      imgProduct.ImageUrl = "Images/Products/" + selectedProduct.ImageFile;
   }

   private Product GetSelectedProduct()
   {
      DataView productsTable = (DataView)
         AccessDataSource1.Select(DataSourceSelectArguments.Empty);

      productsTable.RowFilter = "ProductID = '" + ddlProducts.SelectedValue + "'";
```

```
    DataRowView row = (DataRowView) productsTable[0];

    Product p = new Product();

    p.ProductID = row["ProductID"].ToString();
    p.Name = row["Name"].ToString();
    p.ShortDescription = row["ShortDescription"].ToString();
    p.LongDescription = row["LongDescription"].ToString();
    p.UnitPrice = (decimal) row["UnitPrice"];
    p.ImageFile = row["ImageFile"].ToString();

    return p;
}

protected void btnAdd_Click(object sender, EventArgs e)
{
    if (Page.IsValid)
    {
        CartItem item = new CartItem();

        item.Product = selectedProduct;
        item.Quantity = Convert.ToInt32(txtQuantity.Text);

        this.AddToCart(item);

        Response.Redirect("Cart.aspx");
    }
}

private void AddToCart(CartItem item)
{
    SortedList cart = this.GetCart();

    string productID = selectedProduct.ProductID;

    if (cart.ContainsKey(productID))
    {
        CartItem existingItem = (CartItem) cart[productID];

        existingItem.Quantity += item.Quantity;
    }
    else
        cart.Add(productID, item);
}

private SortedList GetCart()
```

```
    {
        if (Session["Cart"] == null) { Session.Add("Cart", new SortedList()); }

        return (SortedList) Session["Cart"];
    }
}
```

The *C#* code of the *Product.cs* class

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

/// <summary>
/// Summary description for Product
/// </summary>
public class Product
{
   public string ProductID;
   public string Name;
   public string ShortDescription;
   public string LongDescription;
   public decimal UnitPrice;
   public string ImageFile;
}
```

How an ASP.NET application is compiled and run

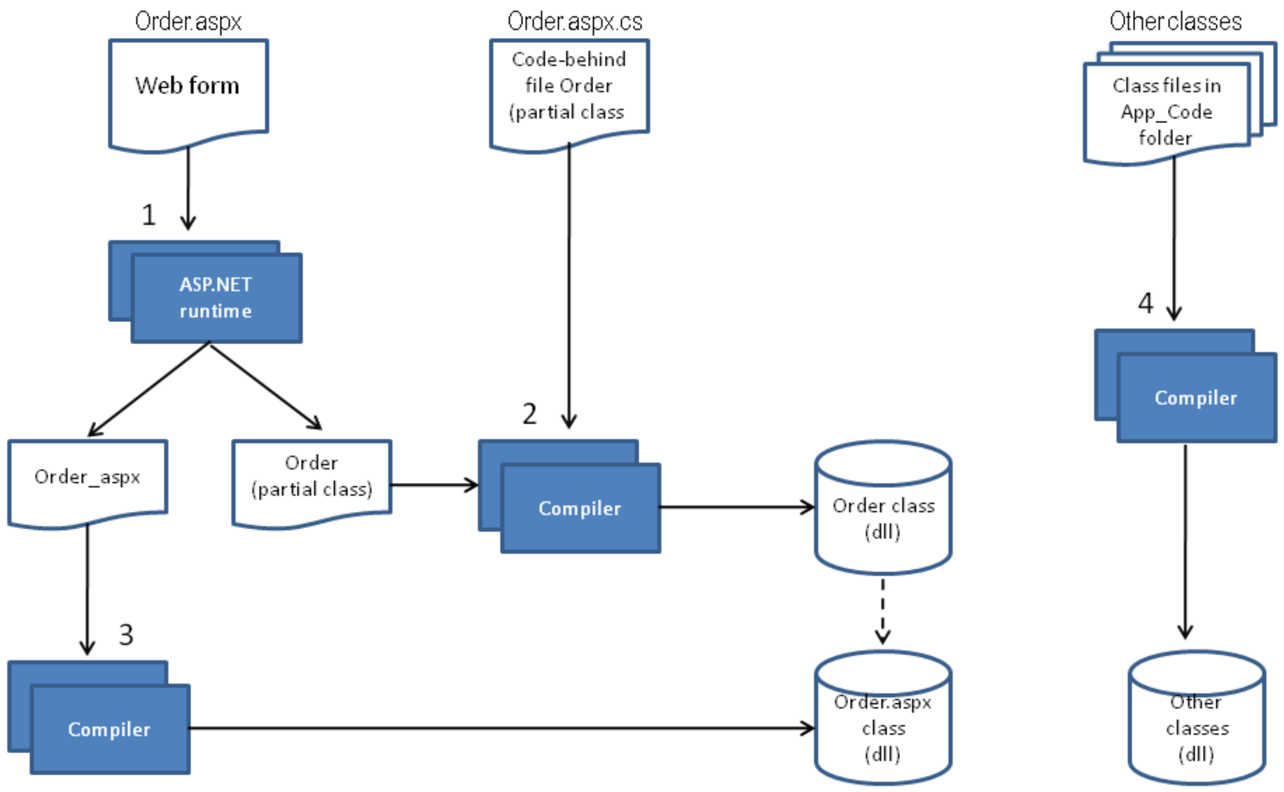
How an ASP.NET application is compiled

The diagram below shows what actually happens behind the scenes when a user requests a page of an ASP.NET 2.0 application.

What happened when an ASP.NET page is requested (see the diagram below**)**

1. The ASP.NET runtime processes the *.aspx* file and generates a partial class (*Order*) that contains the declarations for the form ant its controls, and a class (*Order_aspx*) that contains the code that will initialize the form, instantiates the controls, and generate the HTML for the web browser.
2. The C# compiler compiles the partial class that contains the control declarations with the partial class defined by the code-behind file for the form into an assembly (.dll) that provides the event-handling code for the requested page.

3. The C# compiler compiles the *Order_aspx* class, which inherits the first compiled class (*Order*), and saves the result as an assembly that's executed when the page is requested.
4. If necessary, the C# compiler compiles any other classes files that are stored in the application's *App_Code* folder. These classes are saved in a single assembly.
5. ASP.NET creates an instance of the page from the page's final assembly. Then, ASP.NET raised the appropriate events, which are processed by the event handler for the page, and the page generates the HTML that's passed back to IIS for the response.



The first four steps of this process are done only the first time the aspx page is requested. After that, the page is processed directly from the compiled assemblies. For the *Default* page, the name of the code-behind class is *_Default*.