.NET Technology (PS02CDCA34)

Unit – 4 : ASP.NET

- Introduction to ASP.NET
- ASP.NET Web Application Project – introduction, creation
- ASP.NET Web form - introduction, creating web forms
- ASP.NET Page – layout, lifecycle
- ASP.NET Controls - adding server controls to a Web Form, adding event procedures to Web Server Controls, Implementing code-behind pages
- Creating Master Pages, themes and skins

Introduction to ASP.NET

- ASP.NET was released in 2002 as a successor to Classic ASP.
- ASP.NET pages have the extension **.aspx** and are normally written in C# (C sharp).
- **ASP.NET 4.6** is the latest official version of ASP.NET.
- Web Pages is models for creating ASP.NET web applications.
- **Web Pages combine HTML, CSS, and server code.**
- **ASP.NET code is executed on the server, you cannot view the code in your browser.**
- You will only see the output as plain HTML.

ASP.NET Web Forms Model

- ASP.NET web forms extend the event-driven model of interaction to the web applications. The browser submits a web form to the web server and the server returns a full markup page or HTML page in response.

- All client side user activities are forwarded to the server for stateful processing. The server processes the output of the client actions and triggers the reactions.

- Now, HTTP is a stateless protocol. ASP.NET framework helps in storing the information regarding the state of the application, which consists of:
    - Page state
    - Session state

ASP.NET Web Forms Model

- The **page state** is the state of the client, i.e., the content of various input fields in the web form.
- The **session state** is the collective information obtained from various pages the user visited and worked with, i.e., the overall session state.
- **Example of a shopping cart.**
- User adds items to a shopping cart. Items are selected from a page and the total collected items and price are shown on a different page, say the cart page.
- HTTP cannot keep track of all the information coming from various pages. ASP.NET session state and server side infrastructure keeps track of the information collected globally over a session.
- The ASP.NET runtime carries the page state to and from the server across page requests while generating ASP.NET runtime codes, and incorporates the state of the server side components in hidden fields.
- This way, the server becomes aware of the overall application state and operates in a two-tiered connected way.

ASP.NET Web Forms Model

The ASP.NET Component Model

- The ASP.NET component model provides various building blocks of ASP.NET pages.
- It describes:
- Server side counterparts of almost all HTML elements or tags, such as <form> and <input>.
- Server controls, which help in developing complex user-interface. For example, the Calendar control or the Gridview control.
- An ASP.NET web application is made of pages.
- When a user requests an ASP.NET page, the IIS delegates the processing of the page to the ASP.NET runtime system.
- The ASP.NET runtime transforms the **.aspx page** into an instance of a class, which inherits from the base class page of the .Net framework.
- Each ASP.NET page is an object and all its components i.e., the server-side controls are also objects.

ASP.NET Web Application Project

- When you start a new web site, ASP.NET provides the starting folders and files for the site, including two files for the first web form of the site.
- The file named **Default.aspx** contains the **HTML and asp code** that defines the form.

- The file named **Default.aspx.cs** (for C# coding) contains the code in the language you have chosen and this **code is responsible for the actions performed on a form.** Actions may be web form load , button click etc.

- The file name **Default.aspx.designer.cs** use to design a web form. It add **code for the control that we drag and drop from the toolbox**.

ASP.NET Web Application Project

Projects and Solutions

- A typical ASP.NET application consists of many items:
  - **web content files (Web forms) (.aspx),**
  - **source files (.cs files),**
  - **assemblies (.dll and .exe files),**
  - **data source files (.mdb files)**
  - **User control (.ascx)**
  - **Web service (.asmx)**
  - **Master page (.master)**
  - **Site map (.sitemap)**
  - **Website configuration file (.config) etc.**

  All these files that make up the website are contained in a Solution.

ASP.NET Web Application Project

Building and Running a Project

You can execute an application by:

- Selecting Start
- Selecting Start Without Debugging from the Debug menu,
- pressing F5
- Ctrl-F5

- Generally, **the contents of a project are compiled into an assembly as an executable file (.exe) or a dynamic link library (.dll) file.**

ASP.NET Web Pages - Page Layout

Web sites with a consistent look and feel:

- Every page have the same header
- Every page have the same footer
- Every page have the same style and layout

- With Web Pages this can be done very efficiently. You can have reusable blocks of content (**content blocks**), like **headers and footers**, in separate files.

- You can also define a consistent layout for all your pages, using a **layout template** (**layout file**).

ASP.NET Web Pages - Page Layout

Content Blocks

- Many websites have content that is displayed on every page (like headers and footers).
- With Web Pages you can use the **@RenderPage()** method to import content from separate files.
- Content block (from another file) can be imported anywhere in a web page, and can contain text, markup, and code, just like any regular web page.
- **Using common headers and footers as an example, this saves you a lot of work.**
- **You don't have to write the same content in every page, and when you change the header or footer files, the content is updated in all your pages.**

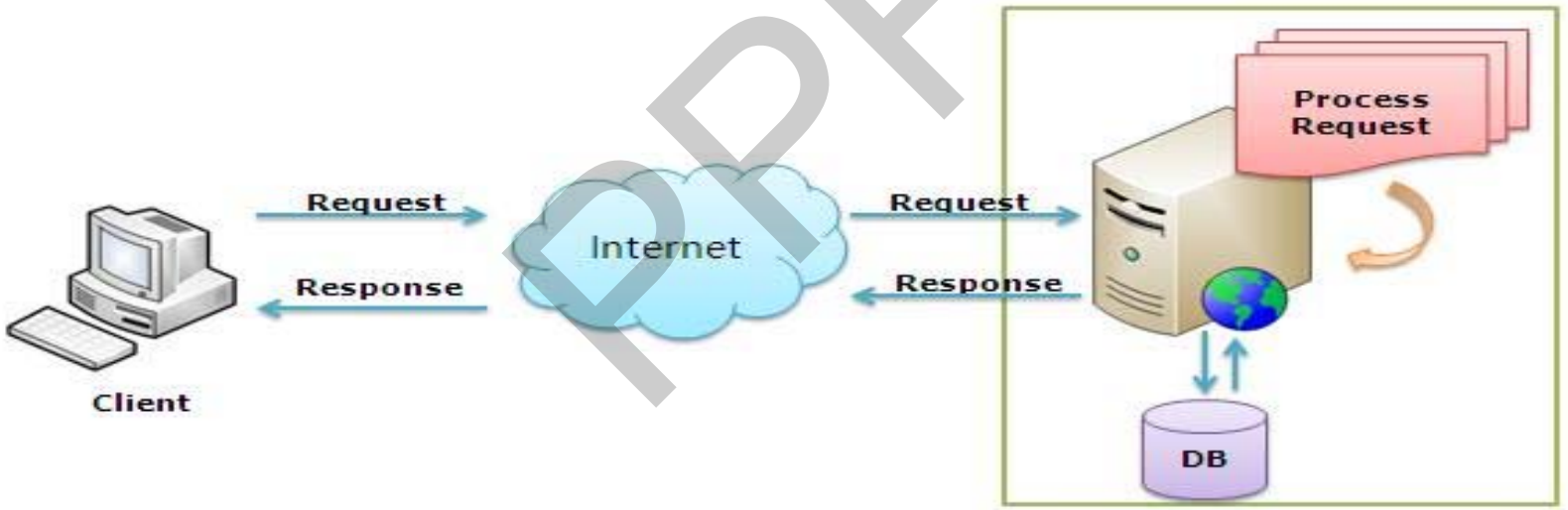
ASP.NET Web Pages - Page Layout

Example

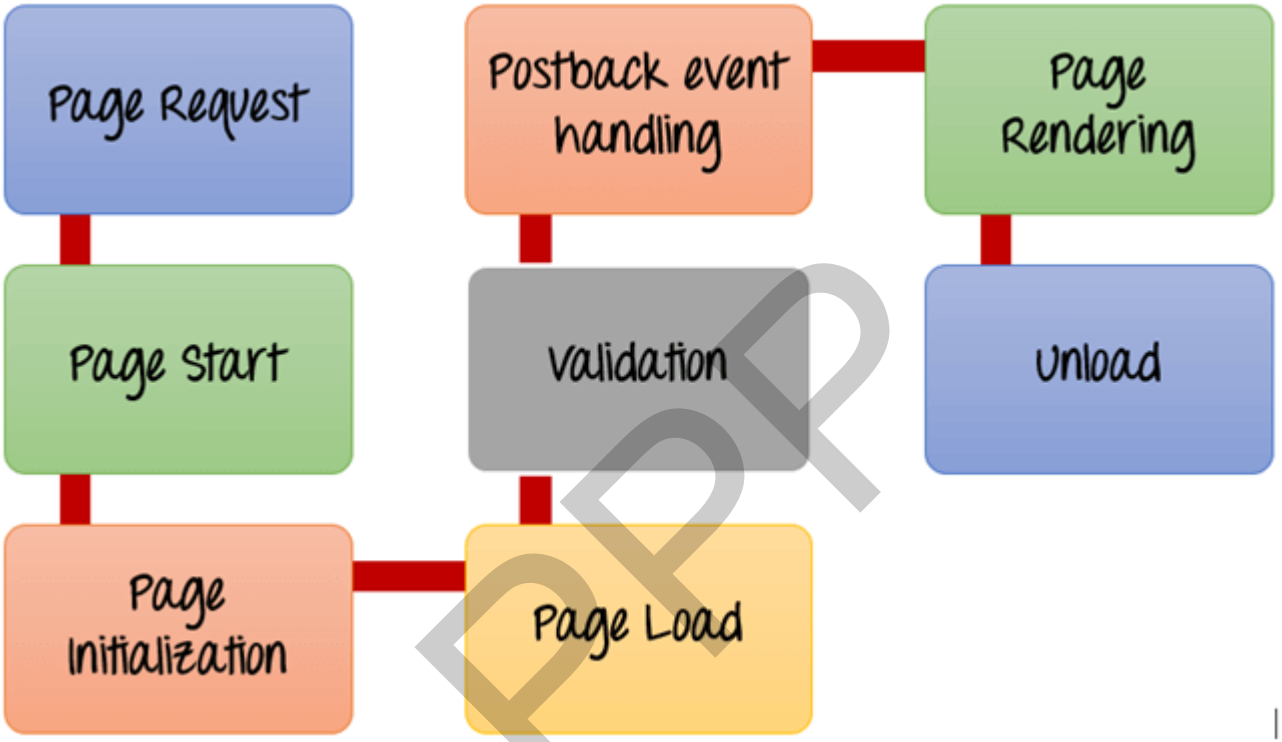
```
<html>
<body>
@RenderPage("header.cshtml")
<h1>Hello Web Pages</h1>
<p>This is a paragraph</p>
@RenderPage("footer.cshtml")
</body>
</html>
```

ASP.NET Page Lifecycle

1. Client Request For Information
2. Request comes to Server
3. Server Process the request
4. Send the response back to Client



ASP.NET Page Lifecycle



- **Page Request-**

–This is when the page is first requested from the server.

–When the page is requested, the server checks if it is requested for the first time. If so, then it needs to compile the page, parse the response and send it across to the user.

–If it is not the first time, the cache is checked to see if the page output exists. If so, that response is sent to the user

ASP.NET Page Lifecycle

- **Page Start** –
  - During this time, 2 objects, known as the **Request and Response object are created**.
  - The Request object is used to hold all the information which was sent when the page was requested.
  - The Response object is used to hold the information which is sent back to the user.
- **Page Initialization**
  - During this time, **all the controls on a web page is initialized.**
  - if you have any label, textbox or any other controls on the web form, they are all initialized.
- **Page Load**
  - This is when the **page is actually loaded with all the default values.**
  - So if a textbox is supposed to have a default value, that value is loaded during the page load time.

ASP.NET Page Lifecycle

- **Validation**
  - Sometimes there can be some **validation set on the form.**
  - For example, there can be a validation which says that a list box should have a certain set of values. If the condition is false, then there should be an error in loading the page.
- **Postback event handling**
  - This event is triggered **if the same page is being loaded again. This happens in response to an earlier event.**
  - Sometimes there can be a situation that a user clicks on a submit button on the page. In this case, the same page is displayed again. In such a case, the Postback event handler is called.
- **Page Rendering**
  - This happens just before all the response information is sent to the user.
  - **All the information on the form is saved, and the result is sent to the user as a complete web page.**
- **Unload**
  - **Once the page output is sent to the user**, there is no need to keep the ASP.net web form objects in memory. So **removing objects from memory.**

ASP.NET 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.
- **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.

ASP.NET Controls

- **Validation controls** - These are used to validate user input and they work by running client-side script.

There are six types of validation controls in ASP.NET

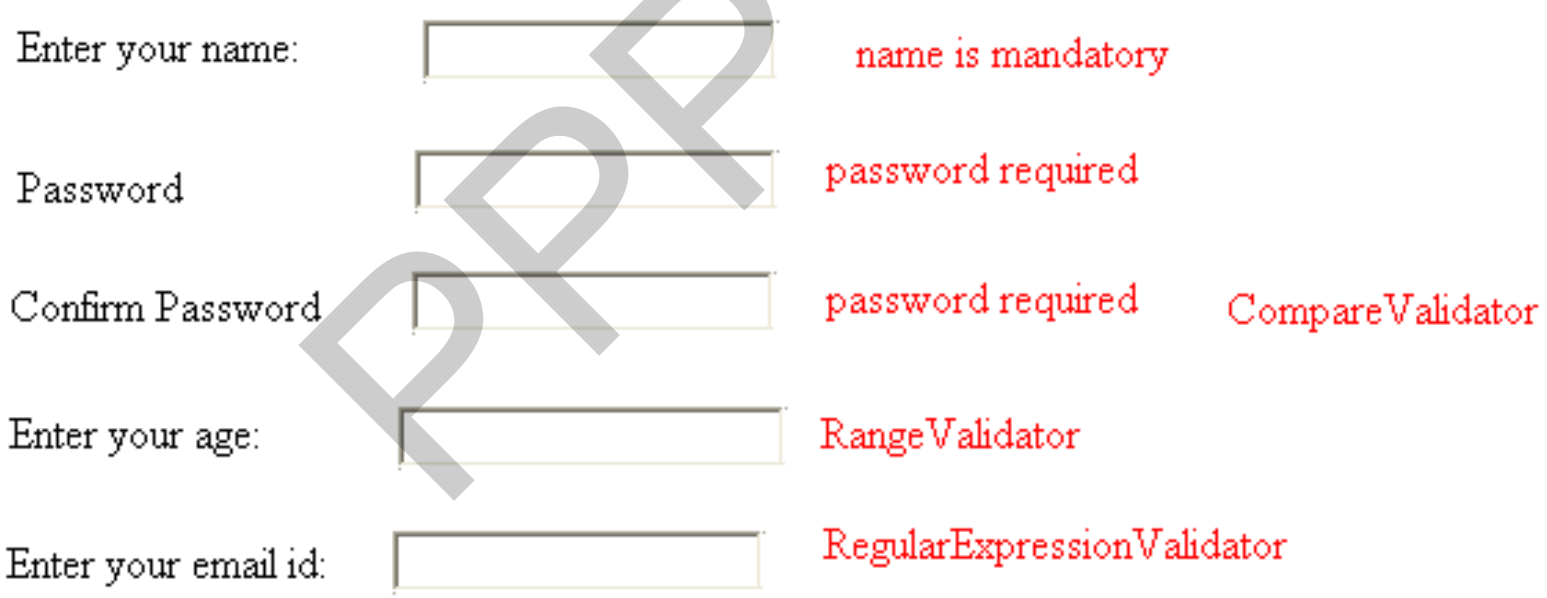
1. RequiredFieldValidation Control
2. CompareValidator Control
3. RangeValidator Control
4. RegularExpressionValidator Control
5. CustomValidator Control
6. ValidationSummary

Validation Control	Description
RequiredFieldValidation	Makes an input control a required field
CompareValidator	Compares the value of one input control to the value of another input control or to a fixed value
RangeValidator	Checks that the user enters a value that falls between two values
RegularExpressionValidator	Ensures that the value of an input control matches a specified pattern
CustomValidator	Allows you to write a method to handle the validation of the value entered
ValidationSummary	Displays a report of all validation errors occurred in a Web page

ASP.NET Controls

- **Example of Validation controls**

Default.aspx Design



Adding event procedures to Web Server Controls

- All ASP.NET controls are implemented as classes, and they have events which are fired when a user performs a certain action on them.
- For example, when a user clicks a button the 'Click' event is generated.
- For handling events, there are in-built attributes and event handlers.
- Event handler is coded to respond to an event, and take appropriate action on it.
- By default, Visual Studio creates an event handler by including a Handles clause on the Sub procedure. This clause names the control and event that the procedure handles.

- **The ASP tag for a button control:**
  <asp:Button ID="btnCancel" runat="server" Text="Cancel" />

Adding event procedures to Web Server Controls

The event handler for the Click event:

```
Protected Sub btnCancel_Click(ByVal sender As Object,
ByVal e As System.EventArgs)

        Handles btnCancel.Click


End Sub
```

Adding event procedures to Web Server Controls

- **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

Adding event procedures to Web Server Controls

- Some events cause the form to be posted back to the server immediately, these are called the **postback events**.
  - For example, the click event such as, Button.Click.

- Some events are not posted back to the server immediately, these are called **non-postback events**.
  - For example, the change events or selection events such as TextBox.TextChanged or CheckBox.CheckedChanged. The nonpostback events could be made to post back immediately by setting their AutoPostBack property to true.

Adding event procedures to Web Server Controls

Default Events

- The default event for the **Page object is Load event.**
- Similarly, every control has a default event. For example, default event for the **button control is the Click event**.
- The default event handler could be created in Visual Studio, just by double clicking the control in design view.
- The following table shows some of the **default events for common controls:**

Button	Click
Calender	SelectionChanged
CheckBox	CheckedChanged
ListBox	SelectedIndexChanged
Menu	MenuItemClick
RadioButton	CheckedChanged

Implementing code-behind pages

- Code-behind refers to code for your ASP.NET page that is contained within a separate class file.

- This allows a clean separation of your HTML from your presentation logic.

Implementing code-behind pages

Default.aspx.designer.cs

```
namespace WebApplication1 {
        public partial class _Default {
     protected global::System.Web.UI.WebControls.Label Label1;
protected global::System.Web.UI.WebControls.TextBox TextBox1;
    }
}
```

Default.aspx.cs

```
namespace WebApplication1{
    public partial class _Default : Page    {
        protected void Page_Load(object sender, EventArgs e)
        {
            Label1.Text = TextBox1.Text;
        }
    }
}
```

ASP.NET Master Pages

- ASP.NET master pages allow you to create a consistent layout for the pages in your application.
- A single master page defines the look and feel and standard behavior that you want for all of the pages (or a group of pages) in your application.
- You can then create individual content pages that contain the content you want to display. When users request the content pages, they merge with the master page to produce output that combines the layout of the master page with the content from the content page.
- A master page is an ASP.NET file with the **extension .master** (for example, MySite.master) with a predefined layout that can include static text, HTML elements, and server controls. The master page is identified by a special **@Master directive** that replaces the **@Page directive** that is used for ordinary .aspx pages. The directive looks like the following.
- **<%@ Master Language="C#" %>**

Code in master page

```
<div>
<table border="1">
<tr>
   <td colspan="2" style="text-align: center">
                        <h1>Header</h1>  </td>  </tr>
<tr>
   <td style="text-align: center; height: 480px; width: 250px">
                        <h1>Menu</h1>   </td>
   <td style="text-align: center; height: 480px; width: 700px">
      <asp:ContentPlaceHolder ID="MainContentPlaceHolder1" r
   unat="server"> <h1>you can change Content here</h1>
      </asp:ContentPlaceHolder>  </td>  </tr>
<tr>
   <td colspan="2" style="text-align: c enter">
                        <h1>Footer</h1></td>  </tr>
</table>
</div>
```

Themes and Skins

A **theme** is a collection of property settings that allow you to define the look of pages and controls, and then apply the look consistently across pages in a Web application, across an entire Web application, or across all Web applications on a server.

Themes are made up of a set of elements: skins, cascading style sheets (CSS), images, and other resources. At a minimum, a theme will contain skins.

,

A skin file has the **file name extension .skin** and **contains property settings for individual controls such as Button, Label, Textbox or Calendar controls.** Control skin settings are like the control markup itself, but contain only the properties you want to set as part of the theme.

How to Apply Theme

There are 3 different options to apply themes to our website:

[1]Setting the theme at the page level: the Theme attribute is added to the page directive of the page.

<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Default.aspx.cs" Inherits="Default" **Theme="Theme1"**%>

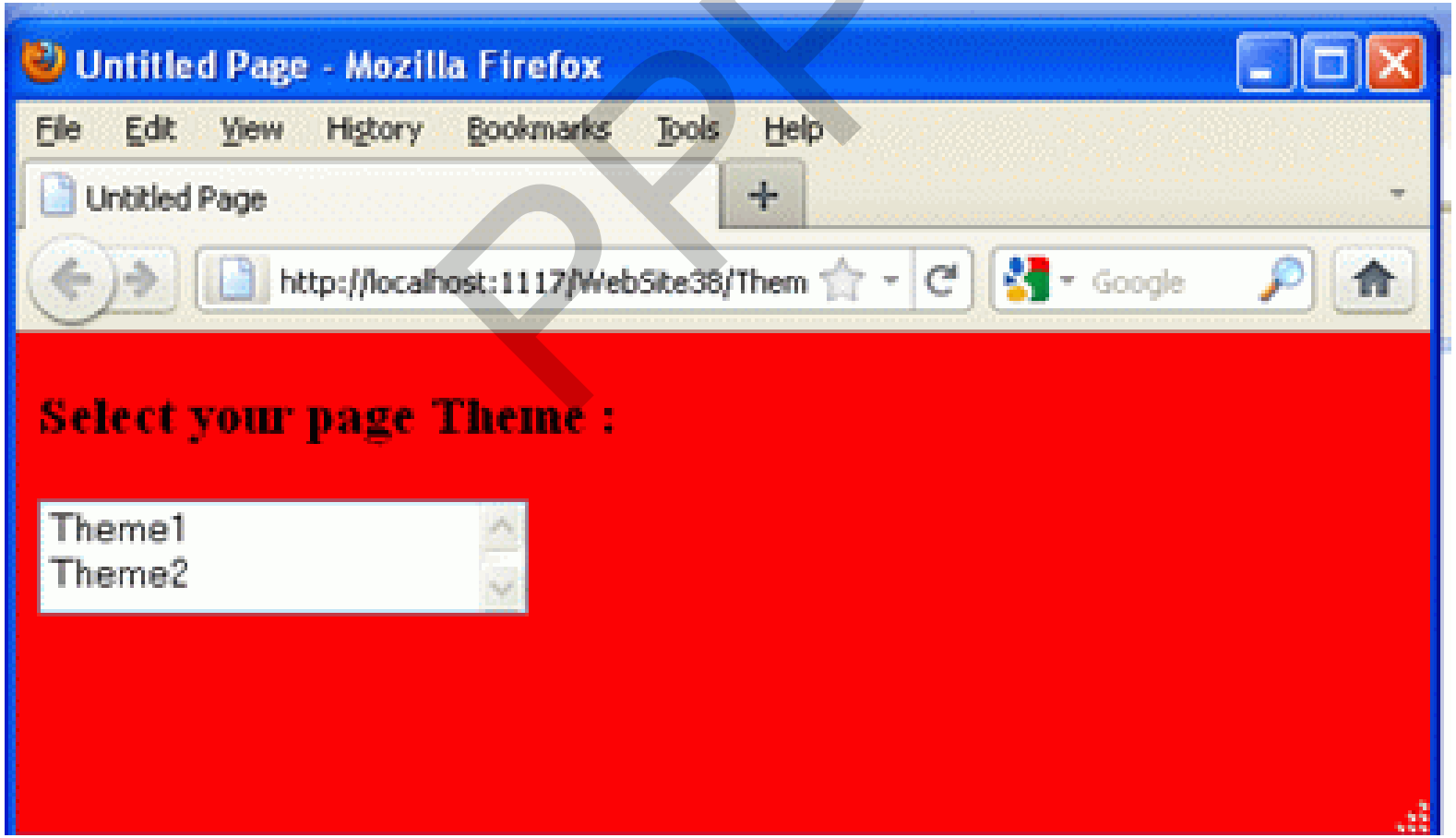
[2]Setting the theme at the site level: to set the theme for the entire website you can set the theme in the web.config of the website. Open the web.config file and locate the <pages> element and add the theme attribute to it:

```
<pages theme="Theme1">
    ....
    ....
</pages>
```

[3]Setting the theme programmatically at runtime: here the theme is set at runtime through coding. It should be applied earlier in the page's life cycle i.e. **Page_PreInit event** should be handled for setting the theme. The better option is to apply this to the Base page class of the site as every page in the site inherits from this class.

Theme

Example : Create 2 themes for the page – one with red background (Theme1) and another with an image as a background (Theme2). When the user selects a particular theme from the ListBox then that theme should be applied dynamically for the page.



Theme

1. Solution Explorer -> Right click -> Add ASP.NET folder -> Themes.

   A new folder App_Themes is added to the Solution Explorer and a new folder Theme1 is added inside it.

2. Theme1 -> Right click -> Add new item -> Stylesheet -> name it as Theme1.css

3. Inside Theme1.css

```
body
{
        background-color:Red;
}
```

Theme

4. Again add a new file.

   App_Themes -> Right click -> Add ASP.NET folder -> Themes

   A new folder Themes2 is created.

5. Themes2 -> Right click -> Add new item -> Stylesheet -> name it as Theme2.css

6. Create an Images folder inside Theme2 and add a picture file 1.jpg inside this. File Theme2.css contains –

```
body
{
    background-color:Red;
    // background-image:url(Images/1.jpg);
}
```

Theme

7. Now create a default.aspx as follows. Add a heading and a list box having AutoPostBack to True.

```
<body>
    <form id="form1" runat="server">
    <div>
    <h3>Select your page Theme : </h3>
        <asp:ListBox ID="ListBox1" runat="server" AutoPostBack="Tr
    ue" Height="41px" onselectedindexchanged= "ListBox1_Select
    edIndexChanged" Width="175px">
                <asp:ListItem>Theme1</asp:ListItem>
                <asp:ListItem>Theme2</asp:ListItem>
            </asp:ListBox>
            <br />
      </div>
    </form>
</body>
```

Theme

8. Inside the default.aspx.cs file a static variable themeValue is defined which saves the value of current theme. In the Page_PreInit event of the page, selected theme from the ListBox is applied to the page and inside the constrctor of the page this PreInit event is provided a EventHandler.

```
//static string themeValue;
        static string themeValue = "Theme1";
private void Page_PreInit(object sender, EventArgs e)
        {
            Page.Theme = themeValue;
        }
    protected void ListBox1_SelectedIndexChanged(object sender,
    EventArgs e)
        {
            themeValue = ListBox1.SelectedItem.Value;
            Response.Redirect(Request.Url.ToString());
        }
```

Theme

9. Now run the page. When we select the Theme from the ListBox, immediately the page is automatically applied with the theme.