Name: Pranit Halady
Roll no. : 53
Division : A

Assignment - 1 Introduction to ASP.NET Controls & State Management

Module - 1

1. **A basic contact form where the user enters their name, and on button click, the name is displayed in a label.**

- **Aim :** To design a simple contact form in Visual Studio where a user enters their name in a textbox, and upon clicking a button, the entered name is displayed on a label.

- **Objective :** To understand the basic structure of a Windows Form Application.
  To learn how to use common controls like **Label, TextBox, and Button** in Visual Studio.
  To handle Button Click events in C#.
  To display user input dynamically in a label.

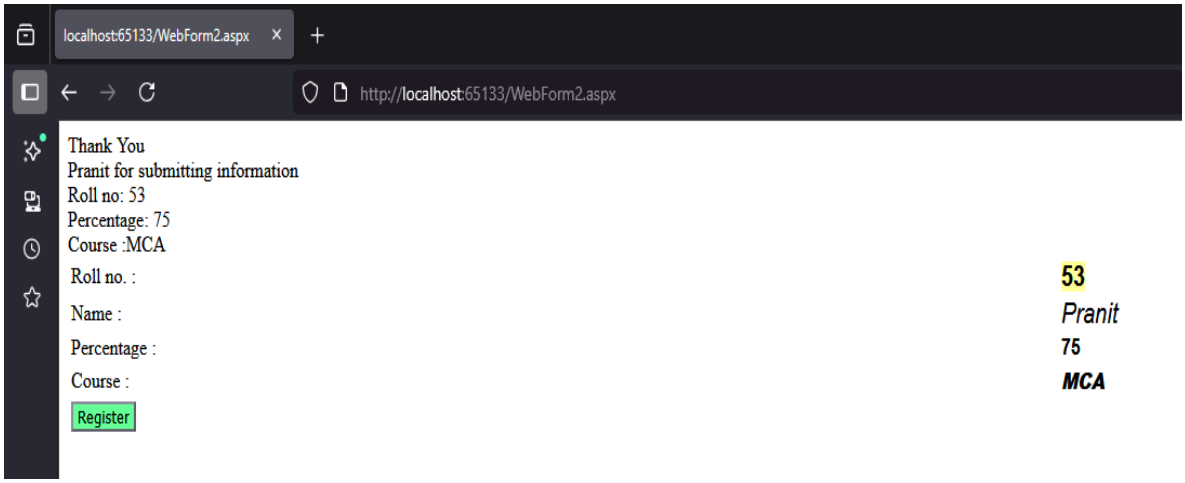
- **Theory :** A Windows Form Application in Visual Studio provides a graphical user interface (GUI) for user interaction.
  In this program, the following components are used:
  1. **TextBox Control:**
     Used to accept input from the user (in this case, the user's name).
  2. **Label Control:**
     Used to display static text or output messages to the user.
  3. **Button Control:**
     Triggers an event when clicked. In this application, it retrieves the text from the TextBox and displays it in the Label.

4. **Event Handling:**
   An event (like a button click) is handled using an event handler method. This demonstrates the concept of **event-driven programming**, where user actions control the flow of the application.

- **Code :**

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

namespace WebApplication1
{
   public partial class WebForm2 : System.Web.UI.Page
   {
      protected void Page_Load(object sender, EventArgs e)
      {

      }

      protected void Button1_Click(object sender, EventArgs e)
      {
         Response.Write("Thank You <br>"+ Label2.Text+
            " for submitting information "
            +"<br>Roll no: "+Label1.Text+
            "<br>Percentage: "+Label5.Text+
            "<br>Course :"+Label6.Text);
      }
   }
}
```

- **Output :**


localhost:65133/WebForm2.aspx
http://localhost:65133/WebForm2.aspx
Thank You
Pranit for submitting information
Roll no: 53
Percentage: 75
Course :MCA
Roll no. :
53
Name :
Pranit
Percentage :
75
Course :
MCA
Register

2. Take three Button controls, Red, Green, and Blue, and one Label control. Using CSS class, when a user presses any of the three buttons, the appearance of the label will change accordingly.

- **Aim :** To create a web application in Visual Studio that uses three buttons—**Red, Green, and Blue**—to change the appearance of a label dynamically using CSS classes when any button is clicked.

- **Objective :** To learn how to design a web page using **HTML, CSS, and ASP.NET** controls in Visual Studio.
  To apply **CSS styling** to web elements for visual changes.
  To understand how to dynamically change the **CSS class** of a label using **server-side code** or **JavaScript**.
  To explore **event-driven programming** and how button clicks affect UI behavior in web applications.

- **Theory :** In a **web-based application** developed using Visual Studio (ASP.NET or HTML + CSS + JavaScript), we can create interactive user interfaces that respond to user actions.
  This experiment demonstrates how **CSS classes** can control the appearance (color, font, background, etc.) of elements and how those styles can be switched dynamically.

  Concepts Involved:

  1. **CSS (Cascading Style Sheets):**
     CSS is used to style HTML elements.
     Example:

     ```
     .redLabel { color: red; }
     .greenLabel { color: green; }
     .blueLabel { color: blue; }
     ```

  2. **HTML Controls:**
     Buttons (<button>) and Labels (<label>) are basic elements for user interaction and text display.
  3. **Event Handling:**
     When a button is clicked, an event triggers a script or server-side code that changes the **class name** of the label.
     Example (JavaScript):

```
function changeColor(color) {
        document.getElementById("myLabel").className = color +
    "Label";
    }
```

4. **ASP.NET Integration (if using Web Forms):**
   Visual Studio allows using server controls like <asp:Button> and <asp:Label>, where button click events can be handled in C# code-behind to change CSS classes dynamically.

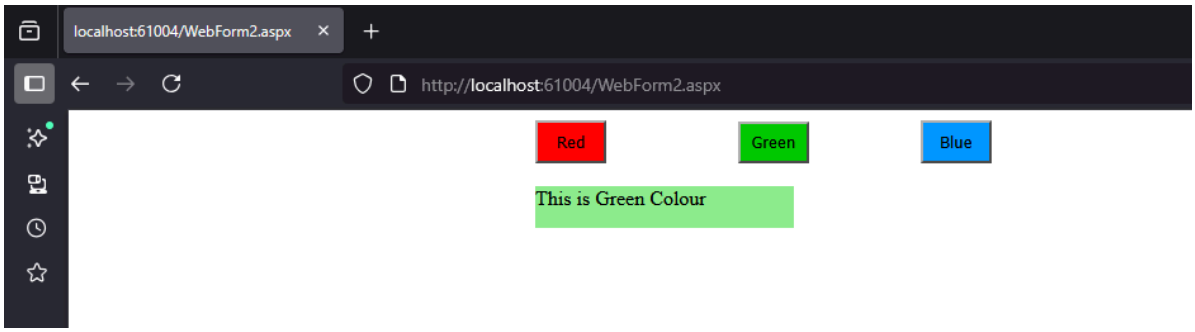
This demonstrates the principles of **event-driven web programming, DOM manipulation,** and **CSS-based styling**, all integrated within Visual Studio's web development environment.

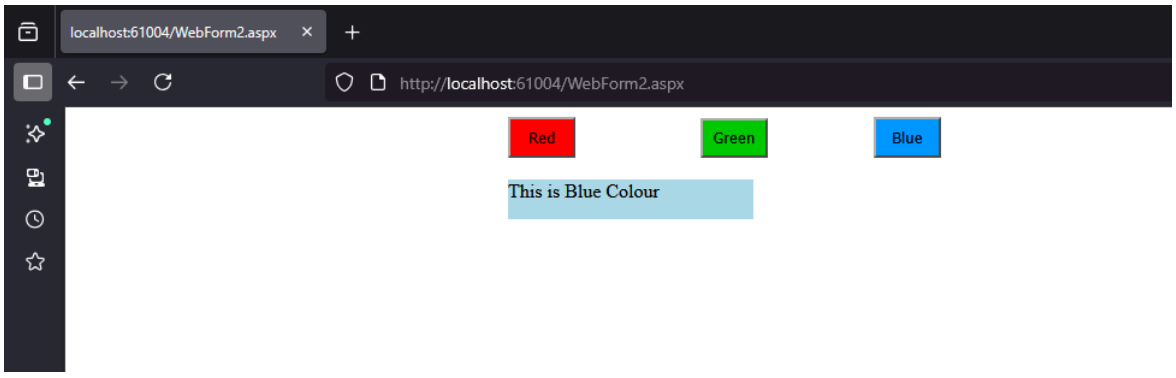
- **Code:**

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

namespace WebApplication2
{
  public partial class WebForm2 : System.Web.UI.Page
  {
    protected void Page_Load(object sender, EventArgs e)
    {

    }

    protected void Button1_Click(object sender, EventArgs e)
    {
      Label1.Text = "This is Red Colour";
      Label1.BackColor = System.Drawing.Color.Red;

    }
    protected void Button2_Click(object sender, EventArgs e)
```

```
    {
        Label1.Text = "This is Green Colour";
        Label1.BackColor = System.Drawing.Color.LightGreen;
    }

    protected void Button3_Click(object sender, EventArgs e)
    {
        Label1.Text = "This is Blue Colour";
        Label1.BackColor = System.Drawing.Color.LightBlue;
    }
  }
}
```

- **Output :**





3.Write a program to give font effects (name, size, effect) to the text (without using Button control).

- **Aim :** To develop a Visual Studio application that changes the **font name, size, and style (bold, italic, underline)** of text dynamically **without using a Button control.**

- **Objective :** To understand how to apply **font effects** to text in a Visual Studio application.
  To explore the use of **events** such as **ComboBox selection change** or **CheckBox state change** instead of button clicks.
  To dynamically modify text properties like **font family, font size,** and **font style** at runtime.
  To learn about e**vent-driven programming** and **control properties** in Visual Studio.

- **Theory :** In **Visual Studio**, GUI-based applications (like Windows Forms or Web Forms) are built using **event-driven programming**—where controls respond automatically to user actions such as typing, selecting, or checking options.

In this program, no **Button** control is used. Instead, font effects are applied when the user interacts directly with controls such as:

1. **ComboBox (for Font Name):**
   Lists available font families (e.g., Arial, Times New Roman, Verdana).
   When the user selects a font, the text's font changes immediately.

2. **NumericUpDown or ComboBox (for Font Size):**
   Allows selection of font size.
   Changing the value updates the text size instantly.

3. **CheckBoxes (for Font Effects):**
   Options like Bold, Italic, and Underline modify the text style dynamically.

4. **Label or TextBox (Display Area):**
   Displays the sample text whose font properties are updated based on user selection.

   Concepts Involved:

   - Event Handling:
     Controls like ComboBox and CheckBox trigger events such as:

```
private void comboBoxFont_SelectedIndexChanged(object sender, EventArgs e)
{
        label1.Font = new Font(comboBoxFont.Text, label1.Font.Size,
label1.Font.Style);
}
```

- Font Class in C#:
  The System.Drawing.Font class represents fonts used to display text in Windows Forms.
  Syntax example:

```
label1.Font = new Font("Arial", 14, FontStyle.Bold | FontStyle.Italic);
```

- No Button Usage:
  Instead of a button click, the real-time property change occurs whenever the user changes a selection or checks/unchecks a box.

This experiment demonstrates the concept of interactive font styling using real-time control events, enhancing user experience without explicit button interaction.

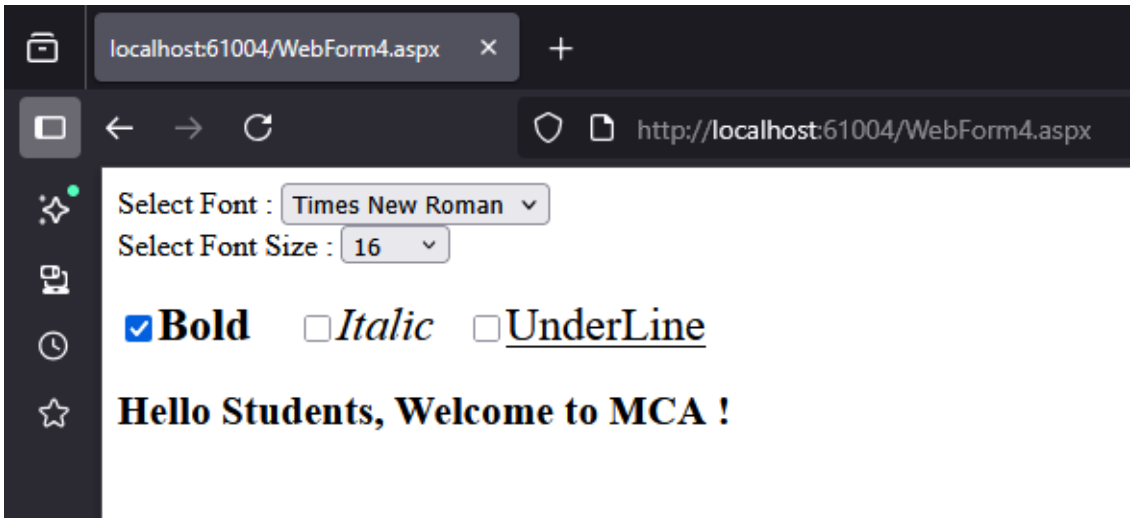
- **Code:**

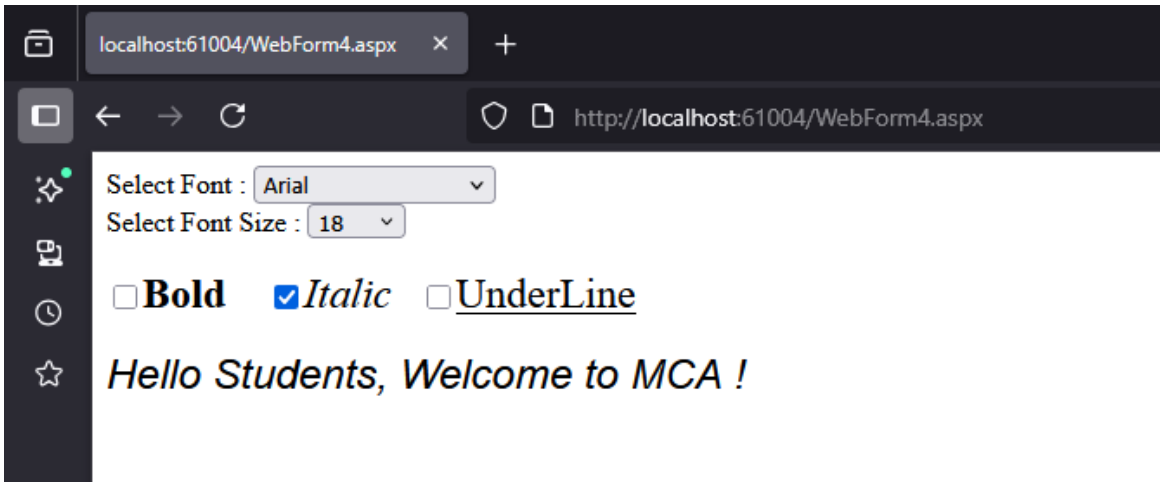
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

namespace WebApplication2
{
  public partial class WebForm4 : System.Web.UI.Page
  {
    protected void Page_Load(object sender, EventArgs e)
    {
    }
    protected void DropDownList1_SelectedIndexChanged(object sender,
EventArgs e)
    {
      Label1.Font.Name = DropDownList1.SelectedItem.Text.ToString();
```

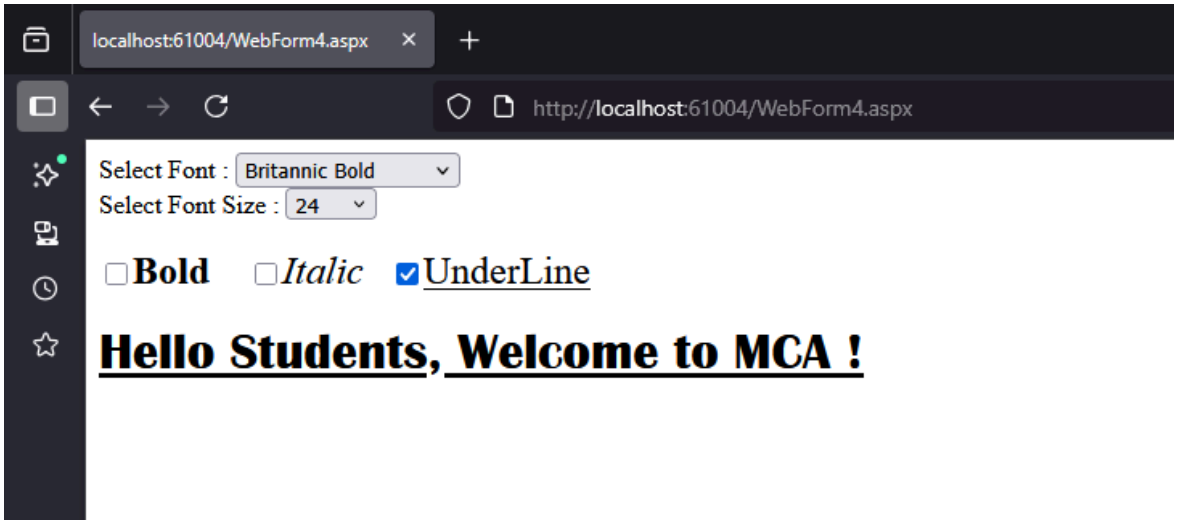
```
    }

    protected void DropDownList2_SelectedIndexChanged(object sender,
EventArgs e)
    {
      Label1.Font.Size =
Convert.ToInt32(DropDownList2.SelectedItem.Text.ToString());
    }

    protected void CheckBox1_CheckedChanged(object sender, EventArgs e)
    {
      if (CheckBox1.Checked)
      {
        Label1.Font.Bold = true;
      }
      else
      {
        Label1.Font.Bold = false;
      }
    }
    protected void CheckBox2_CheckedChanged(object sender, EventArgs e)
    {
      if (CheckBox2.Checked)
      {
        Label1.Font.Italic = true;
      }
      else
      {
        Label1.Font.Italic= false;
      }
    }

    protected void CheckBox3_CheckedChanged(object sender, EventArgs e)
    {
      if (CheckBox3.Checked)
      {
        Label1.Font.Underline = true;

      }
      else
```

```
            {
                Label1.Font.Underline = false;
            }

        }
    }
}
```

- **Output :**






localhost:61004/WebForm4.aspx
http://localhost:61004/WebForm4.aspx
Select Font : Britannic Bold
Select Font Size : 24
Bold
Italic
UnderLine
Hello Students, Welcome to MCA !

4. Design a web application as follows. On submitting the form data confirm the selection made in the following format on one LABEL Control.

Thank you very much _____.
You have chosen _____ for breakfast. I will prepare it for you _____.

- **Aim :** To design and develop a web application in Visual Studio that accepts user input through a form and, upon submission, displays a confirmation message in a Label control showing the entered details in the specified format.

- **Objective :** To understand how to create and manage **web forms** using **ASP.NET** in Visual Studio.
  To learn how to collect and process user input from different form controls (such as **TextBox, DropDownList**, etc.).
  To display customized output messages dynamically using a **Label control**.
  To implement **event-driven programming** through the use of the **Button Click event** to handle form submission.

- **Theory :** A **Web Form Application** in Visual Studio allows users to interact with a web page using server-side controls that can process input and display output dynamically.

  In this experiment, the user fills in a form—such as entering their name, selecting a breakfast item, and specifying a time. When the **Submit** button is clicked, the input values are processed and displayed in a confirmation message using a **Label control**.

Key Concepts:

1. **ASP.NET Web Forms:**
   Web Forms are part of the ASP.NET framework that allows building interactive and dynamic web pages. Controls like TextBox, DropDownList, and Label make it easy to create structured forms.

2. **Controls Used:**

   - TextBox: For entering the user's name.
   - DropDownList / RadioButtonList: For choosing a breakfast item.
   - TextBox / TimePicker: For specifying the preparation time.
   - Button: To submit the form.
   - Label: To display the formatted confirmation message.

3. **Event Handling:**

When the submit button is clicked, the **Button Click** Event is triggered. The event handler reads the input data from the form controls and constructs a message as follows:

```
lblMessage.Text = "Thank you very much " + txtName.Text +
". You have chosen " + ddlBreakfast.SelectedItem.Text +" for breakfast. I will
prepare it for you " + txtTime.Text + ".";
```

4. Label Control:

The Label control displays text dynamically on the web page after the event executes. It is ideal for showing confirmation or output messages.

5. Event-Driven Programming:

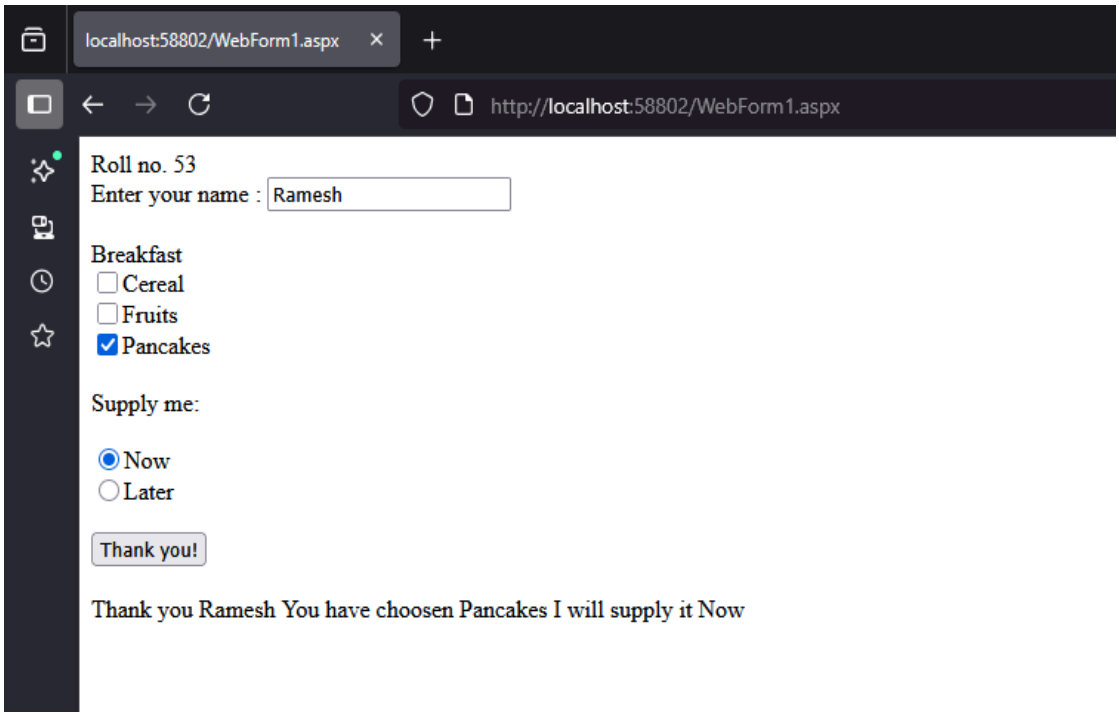
The logic of the application depends on user-triggered events (like form submission). Each event runs specific code in response to user interaction.

- **Code :**

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

namespace WebApplication3
{
   public partial class WebForm1 : System.Web.UI.Page
   {
      protected void Page_Load(object sender, EventArgs e)
      {

      }
      protected void Button1_Click(object sender, EventArgs e)
      {
         string name = TextBox1.Text;
         string breakfast = null;
          string time = null;
         if (CheckBox1.Checked)
```

```
            {
               breakfast += CheckBox1.Text;
            }
            if (CheckBox2.Checked)
            {
               breakfast += CheckBox2.Text;
            }
            if(CheckBox3.Checked)
            {
               breakfast += CheckBox3.Text;
            }
            if(RadioButton1.Checked)
            {
               time += RadioButton1.Text;

            }
            if (RadioButton2.Checked)
            {
               time += RadioButton2.Text;
            }
         Label1.Text = "Thank you " + name + " You have choosen "+ breakfast +"
for breakfast“+”<br>I will prepare it "+time;
        }
    }
}
```

- **Output :**


localhost:58802/WebForm1.aspx
http://localhost:58802/WebForm1.aspx
Roll no. 53
Enter your name : Ramesh
Breakfast
Cereal
Fruits
Pancakes
Supply me:
Now
Later
Thank you!
Thank you Ramesh You have choosen Pancakes I will supply it Now

5. Write a program to demonstrate Postback.

Enable auto post back true in first

- **Aim :** To create a web application in Visual Studio that demonstrates the concept of Postback — where a web page sends data back to the same page on the server for processing and updates dynamically without navigating away.

- **Objective :** To understand the concept of Postback in ASP.NET Web Forms.
  To learn how server-side controls interact with the server during user actions.
  To identify which controls cause a Postback automatically and how to handle their events.
  To demonstrate how the state of the page and its controls are maintained using ViewState.

- **Theory :** In ASP.NET Web Forms, a Postback occurs when a user action on the client (such as clicking a button or selecting a value) sends the page data back to the same server page for processing. The server then executes the required logic and reloads the updated page for the user.

Key Concepts:

1. **Definition of Postback:**
   A Postback is a request sent from the client browser to the server from the same page.
   It differs from a non-postback request, where the page is being loaded for the first time.

   In ASP.NET, this can be checked using:

```
if (!IsPostBack)
{
   // Code runs only the first time the page loads
}
else
{
   // Code runs during postback (e.g., after button click)
}
```

2. **Server Controls and Events:**
   ASP.NET controls like Button, DropDownList, CheckBox, etc., can automatically cause a postback when interacted with.

Example:

- Clicking a Button sends the form data to the server.
- Selecting an item in a DropDownList (if AutoPostBack="true") triggers a postback event.

3. **ViewState:**
   ASP.NET maintains the state of controls between postbacks using ViewState, ensuring user inputs are not lost when the page reloads.
4. **Event Handling:**
   During postback, server-side event handlers are executed. Example:

```
protected void Button1_Click(object sender, EventArgs e)

{
   Label1.Text = "This message is displayed after postback!";
}
```

5. **Page Lifecycle and Postback Check:**
   ASP.NET distinguishes between the initial page load and subsequent postbacks using the IsPostBack property, which is part of the Page Lifecycle:

   IsPostBack = false → first time the page loads.

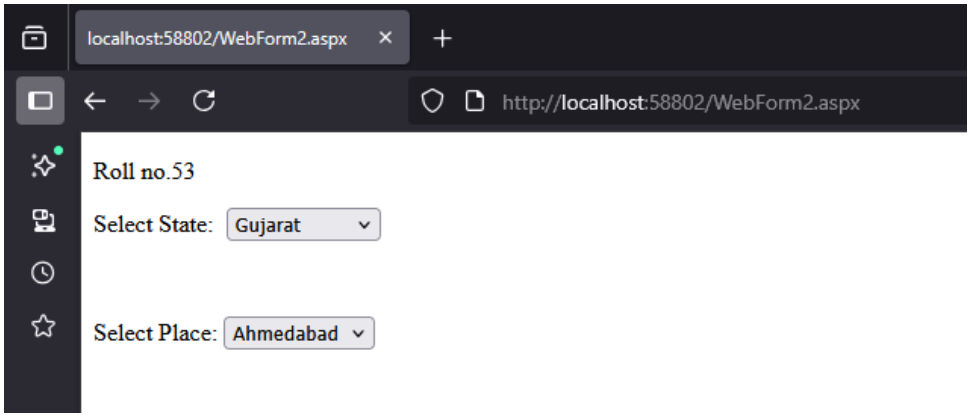
- **Code :**

.aspx file

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

namespace WebApplication3
{
   public partial class WebForm2 : System.Web.UI.Page
   {
      protected void Page_Load(object sender, EventArgs e)
      {
         if (!Page.IsPostBack)
         {
```

```
            DropDownList1.Items.Add("Maharashtra");
            DropDownList1.Items.Add("Gujarat");
            DropDownList1.Items.Add("Goa");
        }
    }
    protected void DropDownList1_SelectedIndexChanged(object sender, EventArgs e)
    {
        DropDownList2.Items.Clear();
        if(DropDownList1.SelectedItem.Text == "Maharashtra")
        {
            DropDownList2.Items.Add("Mumbai");
            DropDownList2.Items.Add("Pune");
            DropDownList2.Items.Add("Kolhapur");
        }
        else if (DropDownList1.SelectedItem.Text == "Gujarat")
        {
            DropDownList2.Items.Add("Ahmedabad");
            DropDownList2.Items.Add("Rajkot");
            DropDownList2.Items.Add("Gandhinagr");
        }
        else if (DropDownList1.SelectedItem.Text == "Goa")
        {
            DropDownList2.Items.Add("Mardol");
            DropDownList2.Items.Add("Thivim");
            DropDownList2.Items.Add("Madgaon");
        }
    }
}
```

- **Output :**


localhost:58802/WebForm2.aspx
http://localhost:58802/WebForm2.aspx
Roll no.53
Select State: Gujarat
Select Place: Ahmedabad

6. Write a program to upload your profile picture and display it in image control (file format should be .jpg, .jpeg or .png format).

- **Aim :** To design and develop a web application in Visual Studio that allows a user to **upload their profile picture** (in .jpg, .jpeg, or .png format) and display the uploaded image in an Image control on the same page.

- **Objective :** To understand how to use the FileUpload and Image controls in ASP.NET.
  To learn how to handle file uploads securely in a web application.
  To validate uploaded files based on their extensions and formats.
  To display the uploaded image dynamically in an Image control after submission.
  To explore how server-side event handling works in Visual Studio.

- **Theory :** In ASP.NET Web Forms, file upload and image display functionalities are implemented using server-side controls that interact with the server to process and manage user inputs.

When a user selects an image file and clicks the upload button, the selected file is sent to the server using a Postback. The server processes the uploaded file, verifies its format, saves it to a specific directory (like /Images), and displays it using an Image control.

Key Concepts:

1. **FileUpload Control:**
   - Allows users to browse and select a file from their computer.
   - Accessed in code using:
     FileUpload1.HasFile
   - The uploaded file can be saved using:
     FileUpload1.SaveAs(Server.MapPath("~/Images/"
   - FileUpload1.FileName));

2. **Image Control:**
   - Displays images on the web page.
   - After uploading, the file path is assigned to the Image control:
     Image1.ImageUrl = "~/Images/" + FileUpload1.FileName;

3. **File Validation:**
   - It is important to check the file extension before saving to ensure only .jpg, .jpeg, or .png files are accepted:

```
 string ext = System.IO.Path.GetExtension(FileUpload1.FileName).ToLower();
if (ext == ".jpg" || ext == ".jpeg" || ext == ".png")
{
   // Valid file type
}
```

4. **Event Handling:**
   - The process is triggered by a Button Click Event, which checks if the user has uploaded a valid file and then displays it.

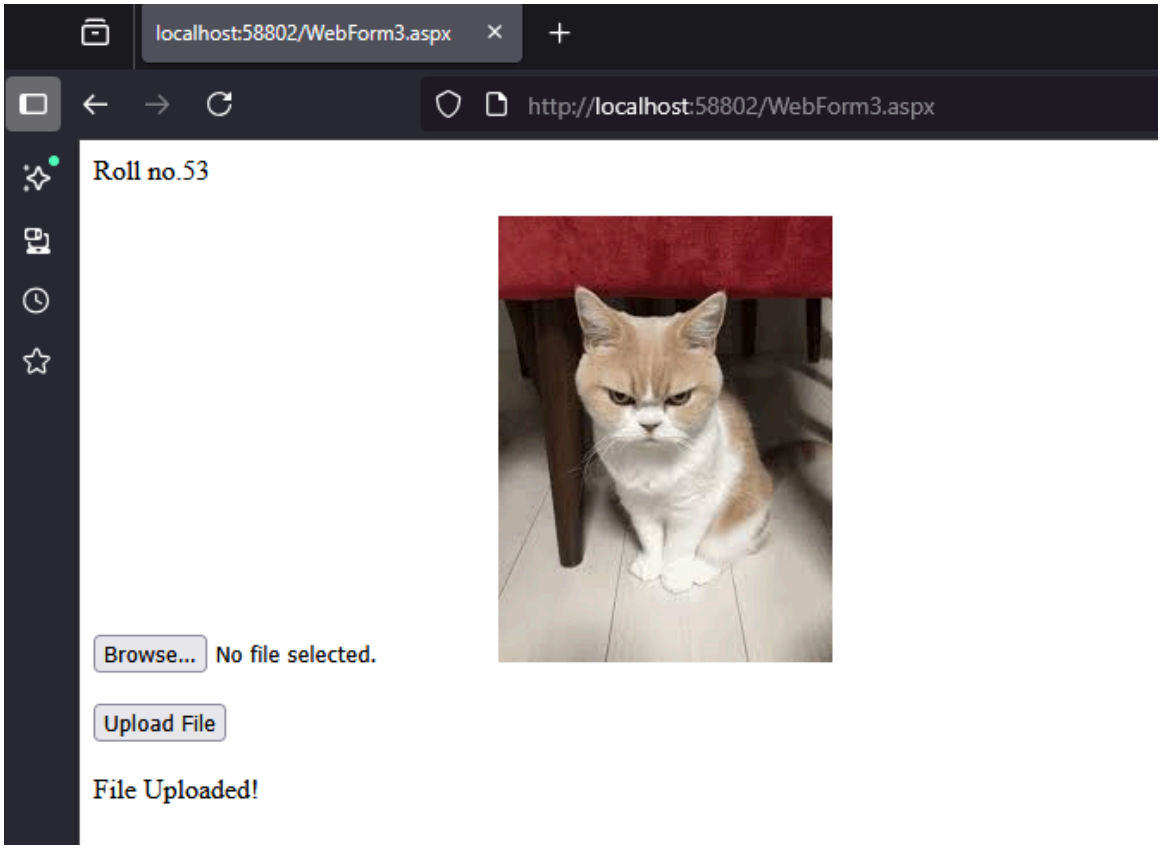
5. **Server-Side Processing:**
   - The uploaded image is stored in a server directory and accessed via relative path to display dynamically in the Image control.

   - **Code :**

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

namespace WebApplication3
{
   public partial class WebForm3 : System.Web.UI.Page
   {
      protected void Page_Load(object sender, EventArgs e)
      {

      }

      protected void Button1_Click(object sender, EventArgs e)
      {
         try
```

```
        {
            string filename = Path.GetFileName(FileUpload1.FileName);
            string ext = Path.GetExtension(filename);
            if (ext.ToLower() == ".jpg")
            {
                FileUpload1.SaveAs(Server.MapPath("~/Image/") + filename);
                Label1.Text = "File Uploaded!";
                Image1.ImageUrl = "~/Image/" + filename;
            }
            if (ext.ToLower() == ".png")
            {
                FileUpload1.SaveAs(Server.MapPath("~/Image/") + filename);
                Label1.Text = "File Uploaded!";
                Image1.ImageUrl = "~/Image/" + filename;
            }
            if (ext.ToLower() == ".jpeg")
            {
                FileUpload1.SaveAs(Server.MapPath("~/Image/") + filename);
                Label1.Text = "File Uploaded!";
                Image1.ImageUrl = "~/Image/" + filename;
            }
        }
        catch {
        }
    }
  }
}
```

- **Output :**


localhost:58802/WebForm3.aspx
http://localhost:58802/WebForm3.aspx
Roll no.53
Browse...
No file selected.
Upload File
File Uploaded!

7. Write a program to demonstrate Ad Rotator Control.

- **Aim :** To design and develop a web application in Visual Studio that demonstrates the use of the **AdRotator control** to display advertisements randomly or sequentially based on an external XML file.

- **Objective :** To understand the working of the **AdRotator control** in ASP.NET.
  To learn how to display different advertisements dynamically on a web page.
  To create and link an **XML advertisement file** with the AdRotator control.
  To explore how the AdRotator randomly selects and displays ads from the given list.
  To understand the use of **server controls** and **data binding** in ASP.NET Web Forms.

- **Theory :** The AdRotator control in ASP.NET is a server-side control used to display a series of advertisements (usually images) on a web page. Each time the page loads or refreshes, the AdRotator displays a different advertisement based on data stored in an XML file that defines the advertisement details.

  This helps developers easily manage and update ads without modifying the page design.

  Key Concepts:

1. **AdRotator Control:**

- It is an ASP.NET control that displays ad banners and automatically rotates them.
- It uses an external XML file that contains information about each advertisement, such as the image path, alternate text, navigation URL, and impression value.
- Example (in ASPX):

```
<asp:AdRotator ID="AdRotator1" runat="server" AdvertisementFile="~/Ads.xml" />
```

2. **Advertisement XML File:**
   The XML file defines a collection of advertisements using the <Advertisements> tag.Each ad is defined using an <Ad> element:

```
<Advertisements>

 <Ad>

  <ImageUrl>~/Images/ad1.jpg</ImageUrl>

        <NavigateUrl>https://www.example1.com</NavigateUrl>
        <AlternateText>Visit Example 1</AlternateText>
       <Impressions>3</Impressions>
       </Ad>
       <Ad>
       <ImageUrl>~/Images/ad2.jpg</ImageUrl>
       <NavigateUrl>https://www.example2.com</NavigateUrl>
       <AlternateText>Explore Example 2</AlternateText>
   <Impressions>2</Impressions>
  </Ad>
</Advertisements>
```

3. **Properties of AdRotator:**
   - AdvertisementFile – Specifies the path to the XML file containing ad details.
   - Target – Defines where the ad link opens (e.g., _blank for new tab).
   - KeywordFilter – Displays only ads with specific keywords.
   - AlternateTextField – Displays text if the image is not available.

4. **Working Principle:**
   - When the page is loaded or refreshed, the AdRotator reads the XML file.
   - It selects an ad based on the Impressions (probability weight) value.
   - It then displays the corresponding image as a hyperlink.

5. Advantages:
   - Easy to maintain and update ads via XML file.
   - No need to hardcode advertisements on the web page.
   - Supports dynamic ad display with controlled frequency.

- **Code :**

XML File Input:

```
<?xml version="1.0" encoding="utf-8" ?>
<Advertisements>
    <Ad>
        <ImageUrl>~/image/Amul.jpeg</ImageUrl>
        <Width>200</Width>
        <Height>100
        </Height>
        <NavigateUrl>https://amul.com/index.php</NavigateUrl>
        <AlternateText>AD 1</AlternateText>
        <Impression>50</Impression>
        <Keyword>Milk</Keyword>
    </Ad>
    <Ad>
        <ImageUrl>~/image/Dell.jpeg</ImageUrl>
        <Width>200</Width>
        <Height>
            100
        </Height>
        <NavigateUrl>https://www.dell.com/en-in/</NavigateUrl>
        <AlternateText>AD </AlternateText>
        <Impression>50</Impression>
        <Keyword>Laptop</Keyword>
    </Ad>
    <Ad>
        <ImageUrl>~/image/Haldiram.jpeg</ImageUrl>
        <Width>200</Width>
        <Height>
            100
        </Height>
        <NavigateUrl>https://www.haldirams.com/</NavigateUrl>
        <AlternateText>AD 3</AlternateText>
        <Impression>50</Impression>
        <Keyword>Farshan</Keyword>
    </Ad>
    <Ad>
        <ImageUrl>~/image/Harley_Davidson.jpeg</ImageUrl>
        <Width>200</Width>
```

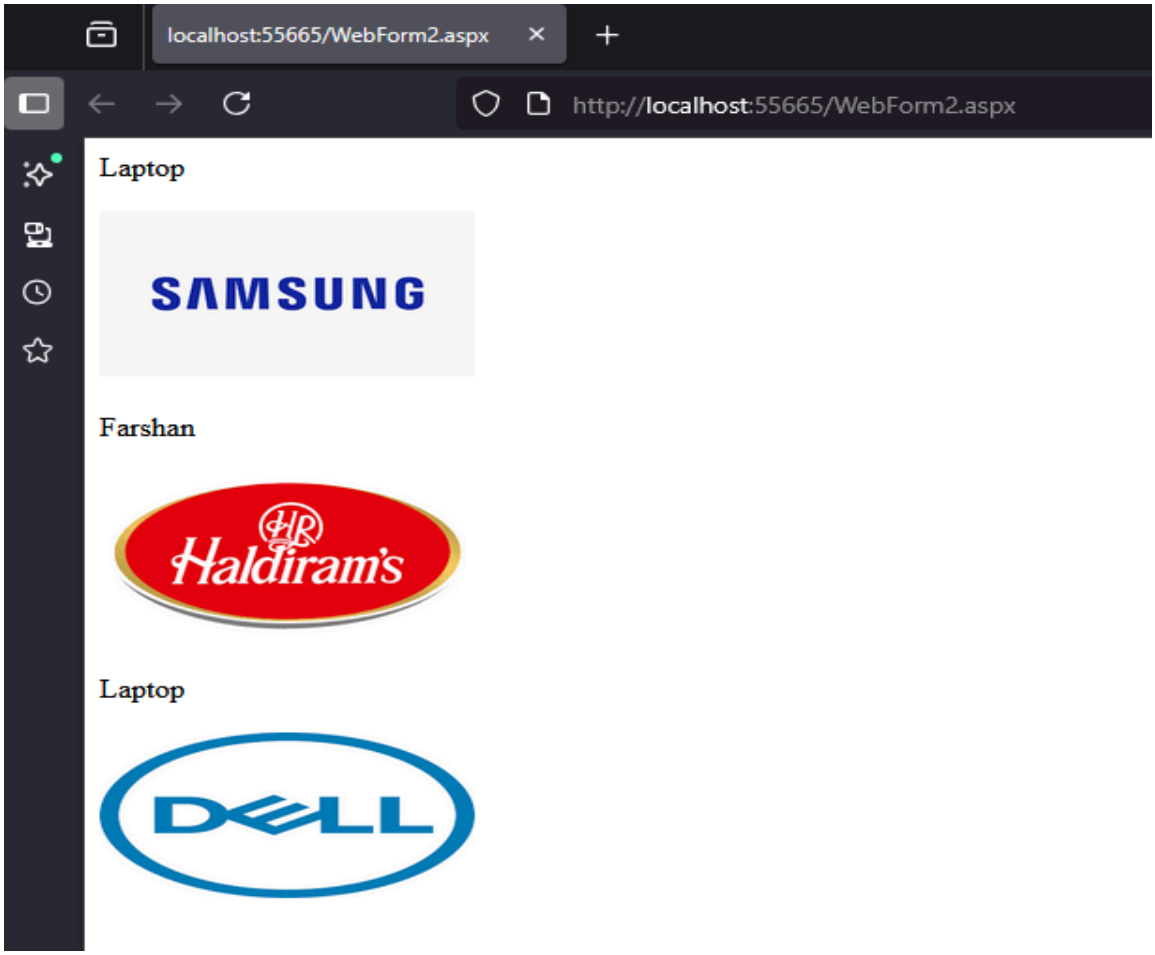
```
            <Height>
                100
            </Height>
            <NavigateUrl>https://www.harley-davidson.com/</NavigateUrl>
            <AlternateText>AD 4</AlternateText>
            <Impression>50</Impression>
            <Keyword>Milk
            </Keyword>
        </Ad>
        <Ad>
            <ImageUrl>~/image/Samsung.jpeg</ImageUrl>
            <Width>200</Width>
            <Height>
                100
            </Height>
            <NavigateUrl>https://www.samsung.com</NavigateUrl>
            <AlternateText>AD 5</AlternateText>
            <Impression>50</Impression>
            <Keyword>Laptop
            </Keyword>
        </Ad>
</Advertisements>
```

WebApplication4.aspx

```
<%@ Page Language="C#" AutoEventWireup="true"
CodeBehind="WebForm2.aspx.cs" Inherits="WebApplication4.WebForm2" %>

<!DOCTYPE html>

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

```
        <asp:AdRotator ID="AdRotator1" runat="server"
DataSourceID="XmlDataSource1" ImageUrlField="" KeywordFilter="Laptop"
NavigateUrlField="" />
        <br />
        <br />
        Farshan<br />
        <br />
        <asp:AdRotator ID="AdRotator2" runat="server"
DataSourceID="XmlDataSource1" ImageUrlField="" KeywordFilter="Farshan"
NavigateUrlField="" />
        <br />
        <br />
        Laptop<br />
        <br />
        <asp:AdRotator ID="AdRotator3" runat="server"
DataSourceID="XmlDataSource1" ImageUrlField="" KeywordFilter="Laptop"
NavigateUrlField=""/>
        <br />
        <br />
        <asp:XmlDataSource ID="XmlDataSource1" runat="server"
DataFile="~/XMLFile1.xml"></asp:XmlDataSource>
        <br />
        <br />
        <br />
        <br />
      </div>
   </form>
</body>
</html>
```

- **Output :**


localhost:55665/WebForm2.aspx
http://localhost:55665/WebForm2.aspx
Laptop
SAMSUNG
Farshan
Haldiram's
Laptop
DELL

8. Write a program to demonstrate Navigation Controls.

- **Aim :** To create a web application in Visual Studio that demonstrates the use of **Navigation Controls** such as **Menu, TreeView, and SiteMapPath**, to enable structured and user-friendly navigation between different web pages.

- **Objective :** To understand the concept of **navigation controls** in ASP.NET Web Forms.
  To learn how to create and manage hierarchical navigation using **Menu and TreeView** controls.
  To implement a **SiteMapPath** control for breadcrumb-style navigation.
  To explore how **Web.sitemap** helps in organizing and linking web pages in a site.
  To develop skills in building dynamic and consistent navigation layouts in Visual Studio.

- **Theory :** In ASP.NET Web Forms, Navigation Controls are used to help users move easily between multiple pages in a web application. They provide a structured and consistent navigation system that enhances user experience and accessibility.

  ASP.NET provides three main navigation controls:
  Menu, TreeView, and SiteMapPath — all of which can use a common data source file called Web.sitemap.

  Key Concepts:

1. **Navigation Controls Overview:**

- **Menu Control:**
  Displays a list of links in a horizontal or vertical format, often used in navigation bars.
  Example:

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

- **TreeView Control:**
  Displays hierarchical data (like folders or categories) in a collapsible tree format.
  Example:

```
<asp:TreeView ID="TreeView1" runat="server" DataSourceID="SiteMapDataSource1"></asp:TreeView>
```

- **SiteMapPath Control:**
  Also known as a breadcrumb control, it shows the current page path within the website structure.
  Example:
  <asp:SiteMapPath ID="SiteMapPath1" runat="server" />

2. Web.sitemap File:

The Web.sitemap file defines the structure of the website in XML format.
Example:

```
<siteMap>
        <siteMapNode title="Home" url="Home.aspx">
          <siteMapNode title="About Us" url="About.aspx" />
          <siteMapNode title="Services" url="Services.aspx">
            <siteMapNode title="Web Design" url="WebDesign.aspx" />
            <siteMapNode title="Development" url="Development.aspx" />
          </siteMapNode>
          <siteMapNode title="Contact" url="Contact.aspx" />
        </siteMapNode>
      </siteMap>
```

3. SiteMapDataSource Control:

This control connects the navigation controls (Menu, TreeView, SiteMapPath) to the Web.sitemap data.

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

4. Working Principle:

- The Web.sitemap file defines the navigation hierarchy.
- SiteMapDataSource connects this data to navigation controls.
- Menu and TreeView render the navigation links dynamically.
- SiteMapPath shows the user's location (breadcrumb trail) as they browse the site.

5. Advantages:

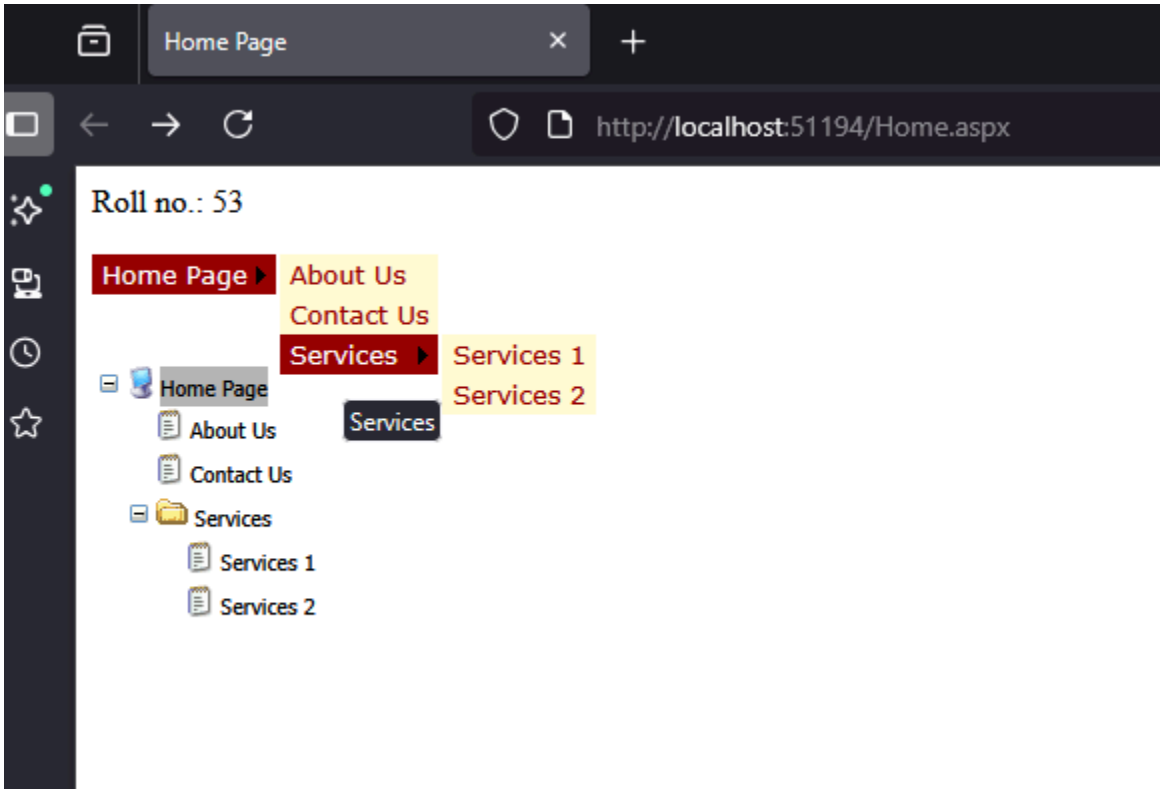
- Simplifies navigation structure creation.
- Automatically updates across all pages when the sitemap changes.
- Provides consistent and user-friendly navigation experience.

- **Code :**

  Web.sitemap

```
<?xml version="1.0" encoding="utf-8" ?>
<siteMap xmlns="http://schemas.microsoft.com/AspNet/SiteMap-File-1.0" >
  <siteMapNode url="Home.aspx" title="Home Page"  description="My Home Page">
        <siteMapNode url="Aboutus.aspx" title="About Us" description="About Us"></siteMapNode>
        <siteMapNode url="Contactus.aspx" title="Contact Us" description="Contact Us"></siteMapNode>
        <siteMapNode url="Services.aspx" title="Services" description="Services">
            <siteMapNode url="Services1.aspx" title="Services 1" description="Services 1"></siteMapNode>
            <siteMapNode url="Services2.aspx" title="Services 2" description="Services 2"></siteMapNode>

        </siteMapNode>
  </siteMapNode>
</siteMap>
```

- **Output :**


Home Page
http://localhost:51194/Home.aspx
Roll no.: 53
Home Page
About Us
Contact Us
Services
Services 1
Services 2
Home Page
About Us
Contact Us
Services
Services 1
Services 2
Services

9. Write a program to demonstrate Calendar control and highlight all holidays of October 2025.

- **Aim :** To design and develop a web application in Visual Studio that demonstrates the use of the **Calendar control** and highlights all holidays of **October 2025** using server-side programming.

- **Objective :** To understand the working and properties of the **Calendar control** in ASP.NET.
  To learn how to highlight specific dates (holidays) using **DayRender event**.
  To explore customization of calendar appearance using style and color properties.
  To display or mark holidays programmatically using C# code-behind in Visual Studio.
  To develop an interactive and user-friendly date-based web application.

- **Theory :** The Calendar control in ASP.NET is a powerful server-side control used for displaying dates in a monthly calendar format. It allows users to view, select, and interact with dates easily.

  In this experiment, we use the Calendar control to display October 2025 and highlight the holiday dates by changing their background or text color dynamically.

  Key Concepts:

  1. **Calendar Control Overview:**
     The Calendar control is used to display a calendar interface in web forms.
     Example:
     <asp:Calendar ID="Calendar1" runat="server" OnDayRender="Calendar1_DayRender"></asp:Calendar>

     Common Properties:

     - VisibleDate – Sets the initial month and year displayed.
     - SelectedDate – Gets or sets the selected date.
     - DayRender – Event used to customize the rendering of each day cell.
     - BackColor, ForeColor, TitleStyle, DayHeaderStyle – Used f

2. **Highlighting Specific Dates (Holidays):**

To highlight holidays, we use the DayRender event.

The event runs for each date cell in the calendar, allowing us to compare the date and apply formatting if it matches a holiday.

Example (in C#):

```
protected void Calendar1_DayRender(object sender,
DayRenderEventArgs e)

{
        // List of holidays in October 2025
        DateTime[] holidays = {
        new DateTime(2025, 10, 2),   // Gandhi Jayanti
        new DateTime(2025, 10, 10),  // Karwa Chauth
        new DateTime(2025, 10, 20),  // Diwali
         new DateTime(2025, 10, 21)   // Lakshmi Pooja
};

foreach (DateTime holiday in holidays)
{
        if (e.Day.Date == holiday)
{
        e.Cell.BackColor = System.Drawing.Color.LightCoral;
        e.Cell.ForeColor = System.Drawing.Color.White;
        e.Cell.Font.Bold = true;
        e.Cell.ToolTip = "Holiday";
}
}
}
```

3. **Event Handling:**

The DayRender event fires automatically while rendering each date cell.

It is commonly used to highlight special days, disable past dates, or add custom content.

4. **Page Lifecycle Interaction:**

   When the page is loaded, the Calendar control renders the month view, and the DayRender event modifies specific days before final display to the user.

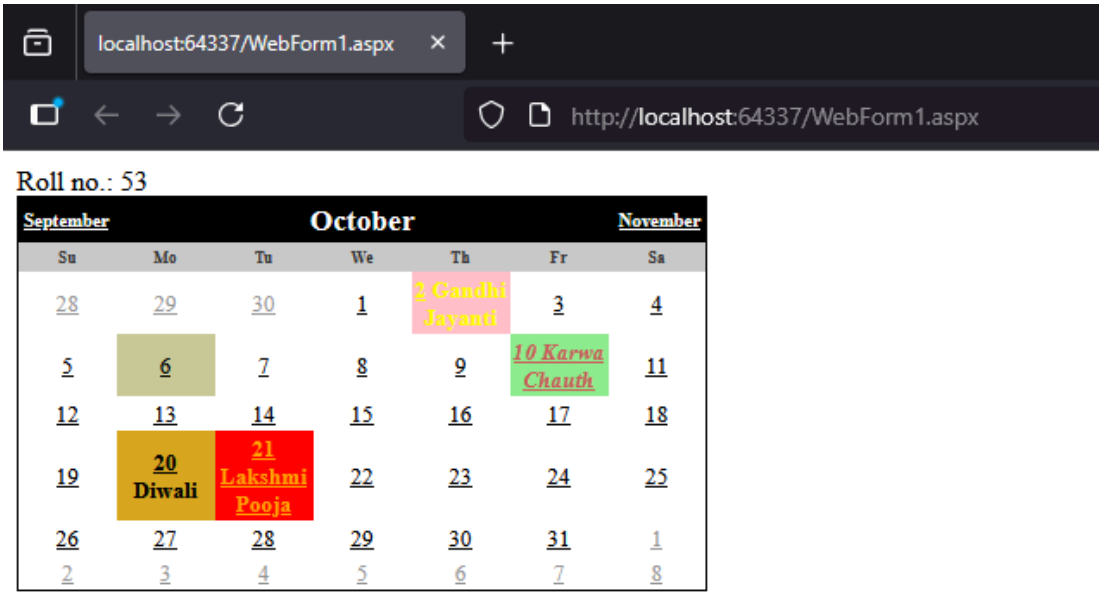
5. **Advantages:**
   Provides an intuitive and visual way to mark events or holidays.
   Can be customized dynamically without client-side scripts.
   Useful for scheduling systems, event planners, and date-based data entry.

- **Code :**

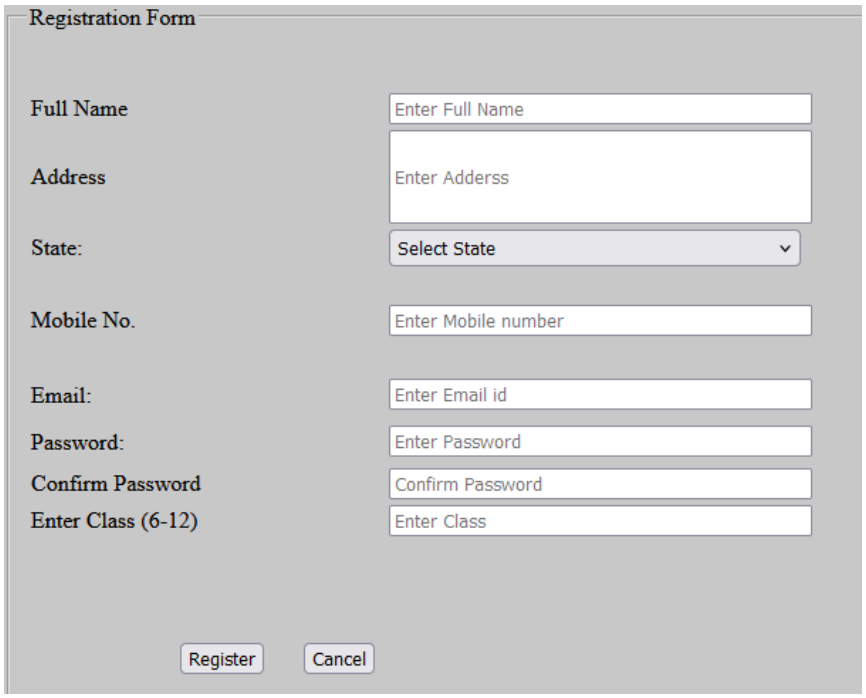
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

namespace WebApplication7
{
  public partial class WebForm1 : System.Web.UI.Page
  {
    protected void Page_Load(object sender, EventArgs e)
    {
    }
      protected void Calendar1_DayRender(object sender,
DayRenderEventArgs e)
    {
      DateTime d1=new DateTime(2025,10,2);
      if(e.Day.Date.ToString()==d1.ToString())
      {
        e.Cell.Controls.Add(new LiteralControl("\n Gandhi Jayanti"));
        e.Cell.BackColor=System.Drawing.Color.Pink;
        e.Cell.Font.Bold=true;
        e.Cell.ForeColor=System.Drawing.Color.Yellow;
      }
      DateTime d2 = new DateTime(2025, 10, 20);
      if (e.Day.Date.ToString() == d2.ToString())
```

```
            {
                e.Cell.Controls.Add(new LiteralControl("\n Diwali"));
                e.Cell.BackColor = System.Drawing.Color.Goldenrod;
                e.Cell.Font.Bold = true;
                e.Cell.ForeColor = System.Drawing.Color.Black;
            }
            DateTime d3 = new DateTime(2025, 10, 21);
            if (e.Day.Date.ToString() == d3.ToString())
            {
                e.Cell.Controls.Add(new LiteralControl("\n Lakshmi Pooja"));
                e.Cell.BackColor = System.Drawing.Color.Red;
                e.Cell.Font.Bold = true;
                e.Cell.Font.Underline = true;
                e.Cell.ForeColor = System.Drawing.Color.Orange;
            }
            DateTime d4 = new DateTime(2025, 10, 10);
            if (e.Day.Date.ToString() == d4.ToString())
            {
                e.Cell.Controls.Add(new LiteralControl("\n Karwa Chauth"));
                e.Cell.BackColor = System.Drawing.Color.LightGreen;
                e.Cell.Font.Bold = true;
                e.Cell.Font.Underline = true;
                e.Cell.Font.Italic = true;
                e.Cell.ForeColor = System.Drawing.Color.IndianRed;
            }
        }
    }
}
```

- **Output :**



10. Create a registration form having the following UI.



- **Aim :** To design and develop a **registration form** in Visual Studio using ASP.NET Web Forms that allows a user to input their personal details such as name, gender, email, password, and contact information through various web controls.

- **Objective :** To understand how to create and design a registration form using ASP.NET server controls.
  To learn how to collect user data through different input fields such as TextBox, RadioButton, DropDownList, and CheckBox.
  To apply validation controls to ensure that users enter valid and complete information.
  To demonstrate event handling and postback in ASP.NET when submitting a form.
  To provide a basic understanding of form layout design and user input handling in Visual Studio.

- **Theory :** A Registration Form is one of the fundamental elements in web applications that allows users to provide their personal information to register for a service or website. In ASP.NET Web Forms, such forms are created using server-side controls which simplify form creation, data validation, and submission.

1. ASP.NET Controls Used:

Label Control: Displays static text or field names.

Example:

```
<asp:Label ID="lblName" runat="server" Text="Full Name:"></asp:Label>
```

TextBox Control: Allows users to enter data such as name, email, or password.

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

RadioButtonList: Enables users to choose a gender or similar options.

```
<asp:RadioButtonList ID="rblGender" runat="server">
  <asp:ListItem>Male</asp:ListItem>
  <asp:ListItem>Female</asp:ListItem>
</asp:RadioButtonList>
```

DropDownList: Provides a list of selectable options like country or state.

```
<asp:DropDownList ID="ddlCountry" runat="server">
  <asp:ListItem>India</asp:ListItem>
  <asp:ListItem>USA</asp:ListItem>
  <asp:ListItem>UK</asp:ListItem>
</asp:DropDownList>
```

CheckBox: Used for agreement or preferences (e.g., accepting terms).

```
<asp:CheckBox ID="chkAgree" runat="server" Text="I agree to the terms and conditions" />
```

Button: Triggers an event (e.g., form submission).

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

Label (Output): Displays the result or confirmation message.

```
<asp:Label ID="lblResult" runat="server"></asp:Label>
```

2. Validation Controls:

ASP.NET provides built-in Validation Controls to ensure data integrity before submission:

- **RequiredFieldValidator** – Ensures a field is not empty.
- **RegularExpressionValidator** – Checks pattern (like email format).
- **CompareValidator** – Confirms passwords match.
- **RangeValidator** – Restricts numeric input range.

Example:

```
<asp:RequiredFieldValidator ID="rfvName" runat="server"
        ControlToValidate="txtName" ErrorMessage="Name is required!"
ForeColor="Red"></asp:RequiredFieldValidator>
```

3. Event Handling (C# Code-Behind):

Event-driven programming in ASP.NET allows server-side code to execute when a user performs an action like clicking a button.

Example:

```
protected void btnSubmit_Click(object sender, EventArgs e)
{
    lblResult.Text = "Thank you " + txtName.Text + ", your registration is
successful!";
}
```

4. Postback Mechanism:

When a form is submitted in ASP.NET, the page data is sent back to the same page on the server.
This is known as a postback, allowing the server to process the data and return results dynamically.

5. Advantages:

- Easy to build and validate forms using pre-defined controls.
- Automatic state management for server controls.
- Secure and structured data collection.
- User-friendly and professional interface.

- **Code :**
  Web.config

```
<?xml version="1.0" encoding="utf-8"?>
<!--For more information on how to configure your ASP.NET application, please visit
  https://go.microsoft.com/fwlink/?LinkId=169433-->
<configuration>
  <system.web>
    <compilation debug="true" targetFramework="4.7.2" />
    <httpRuntime targetFramework="4.7.2" />
  </system.web>
  <system.codedom>
    <compilers>
      <compiler language="c#;cs;csharp" extension=".cs"
type="Microsoft.CodeDom.Providers.DotNetCompilerPlatform.CSharpCodeProvider, Microsoft.CodeDom.Providers.DotNetCompilerPlatform, Version=2.0.1.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35" warningLevel="4"
compilerOptions="/langversion:default /nowarn:1659;1699;1701" />
      <compiler language="vb;vbs;visualbasic;vbscript" extension=".vb"
type="Microsoft.CodeDom.Providers.DotNetCompilerPlatform.VBCodeProvider, Microsoft.CodeDom.Providers.DotNetCompilerPlatform, Version=2.0.1.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35" warningLevel="4"
compilerOptions="/langversion:default /nowarn:41008
/define:_MYTYPE=\&amp;quot;Web\&amp;quot; /optionInfer+" />
    </compilers>
  </system.codedom>
        <appSettings>
                <add key="ValidationSettings:UnobtrusiveValidationMode"
value="None" />
        </appSettings>
</configuration>
```

WebForm2.aspx

```
<%@ Page Language="C#" AutoEventWireup="true"
CodeBehind="WebForm2.aspx.cs" Inherits="WebApplication7.WebForm2" %>

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title></title>
    <style type="text/css">
        .auto-style1 {
            width: 185px;
        }
        .auto-style2 {
            margin-left: 280px;
        }
        .auto-style3 {
            width: 302px;
        }
        .auto-style4 {}
        .auto-style5 {}
        .auto-style6 {
            width: 185px;
            height: 33px;
        }
        .auto-style7 {
            width: 302px;
            height: 33px;
        }
        .auto-style8 {
            height: 33px;
        }
        .auto-style9 {}
        .auto-style10 {}
        .auto-style11 {}
        .auto-style12 {
            width: 185px;
            height: 40px;
```

```
        }
        .auto-style13 {
            width: 302px;
            height: 40px;
        }
        .auto-style14 {
            height: 40px;
        }
        .auto-style15 {
            width: 185px;
            height: 61px;
        }
        .auto-style16 {
            width: 302px;
            height: 61px;
        }
        .auto-style17 {
            height: 61px;
        }
        .auto-style18 {}
        .auto-style19 {}
        .auto-style20 {}
    </style>
    <script type="text/javascript">
        function MyValidation(src, args) {
            var input =
document.getElementById('<%=TextBox4.ClientID%>').value;

            args.IsValid = input.length > 8;
        }
    </script>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <table style="width: 100%; height: 283px;">
                <tr>
                    <td class="auto-style1">
                        <asp:Label ID="Label1" runat="server" Text="Full
Name"></asp:Label>
```

```
                </td>
                <td class="auto-style3">
                    <asp:TextBox ID="TextBox1" runat="server"
CssClass="auto-style5" Height="20px"  Width="285px"></asp:TextBox>
                </td>
                <td>
                    <asp:RequiredFieldValidator ID="RequiredFieldValidator1"
runat="server" ControlToValidate="TextBox1"
ErrorMessage="RequiredFieldValidator" ForeColor="Red"
ValidationGroup="Submit"></asp:RequiredFieldValidator>
                    <br />
                    <br />
                </td>
            </tr>
            <tr>
                <asp:Label ID="Label2" runat="server" Text="Label"></asp:Label>
                <td class="auto-style1">
                    <asp:Label ID="Label4" runat="server"
Text="Address"></asp:Label>
                </td>
                <td class="auto-style3">
                    <asp:TextBox ID="TextBox2" runat="server"
CssClass="auto-style4" Height="79px" Width="284px"></asp:TextBox>
                </td>
                <td>
                    <asp:RequiredFieldValidator ID="RequiredFieldValidator2"
runat="server" ControlToValidate="TextBox2" ErrorMessage="Address filled not
be empty" ForeColor="Red"
ValidationGroup="Submit"></asp:RequiredFieldValidator>
                    <br />
                    <br />
                </td>
            </tr>
            <tr>
                <td class="auto-style1">
                    <asp:Label ID="Label5" runat="server"
Text="State"></asp:Label>
                </td>
                <td class="auto-style3">
```

```
                <asp:DropDownList ID="DropDownList1" runat="server"
CssClass="auto-style18" Height="19px" Width="240px">
                    <asp:ListItem>Select state</asp:ListItem>
                    <asp:ListItem>Maharashtra</asp:ListItem>
                    <asp:ListItem>Goa</asp:ListItem>
                    <asp:ListItem>Gujarat</asp:ListItem>
                    <asp:ListItem>Karnataka</asp:ListItem>
                    <asp:ListItem Value="Rajasthan">Rajasthan</asp:ListItem>
                </asp:DropDownList>
            </td>
            <td>
                <asp:RequiredFieldValidator ID="RequiredFieldValidator3"
runat="server" ControlToValidate="DropDownList1" Display="Dynamic"
ErrorMessage="Select state" ForeColor="Red" InitialValue="Select State"
ValidationGroup="Submit"></asp:RequiredFieldValidator>
                <br />
                <br />
            </td>
        </tr>
        <tr>
            <td class="auto-style1">
                <asp:Label ID="Label6" runat="server" Text="Mobile No.
:"></asp:Label>
            </td>
            <td class="auto-style3">
                <asp:TextBox ID="TextBox5" MaxLength="10" runat="server"
CssClass="auto-style11" Width="278px"></asp:TextBox>
            </td>
            <td>
                <asp:RequiredFieldValidator ID="RequiredFieldValidator4"
runat="server" ControlToValidate="TextBox5" Display="Dynamic"
                    ErrorMessage="Enter mobile no." ForeColor="Red"
                    ></asp:RequiredFieldValidator>
                <br />
                <asp:RegularExpressionValidator
ID="RegularExpressionValidator1" runat="server"
ControlToValidate="TextBox5" Display="Dynamic" ErrorMessage="Invalid
Mobile number" ForeColor="Red" ValidationExpression="\d{10}"
ValidationGroup="Submit"></asp:RegularExpressionValidator>
                <br />
```

```
                </td>
            </tr>
            <tr>
                <td class="auto-style1">
                    <asp:Label ID="Label7" runat="server" Text="Email
:"></asp:Label>
                </td>
                <td class="auto-style3">
                    <asp:TextBox ID="TextBox3" runat="server"
CssClass="auto-style9" Width="277px"></asp:TextBox>
                </td>
                <td>
                    <asp:RequiredFieldValidator ID="RequiredFieldValidator5"
runat="server" ControlToValidate="TextBox3"
ErrorMessage="RequiredFieldValidator" ForeColor="Red"
ValidationGroup="Submit"></asp:RequiredFieldValidator>
                    <br />
                    <asp:RegularExpressionValidator
ID="RegularExpressionValidator2" runat="server"
ControlToValidate="TextBox3" ErrorMessage="Invalid email address"
ForeColor="Red"
ValidationExpression="\w+([-+.']\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*"
ValidationGroup="Submit"></asp:RegularExpressionValidator>
                    <br />
                </td>
            </tr>
            <tr>
                <td class="auto-style15">
                    <asp:Label ID="Label9" runat="server"
Text="Pasword"></asp:Label>
                </td>
                <td class="auto-style16">
                    <asp:TextBox ID="TextBox4" runat="server"
CssClass="auto-style10" Width="278px"></asp:TextBox>
                </td>
                <td class="auto-style17">
                    <asp:RequiredFieldValidator ID="RequiredFieldValidator6"
runat="server" ControlToValidate="TextBox4" Enabled="False"
ErrorMessage="Enter password" ForeColor="Red"
ValidationGroup="Submit"></asp:RequiredFieldValidator>
```

```
                    <br />
                    <asp:CustomValidator ID="CustomValidator1" runat="server"
ClientValidationFunction="MyValidation" ControlToValidate="TextBox4"
Display="Dynamic" ErrorMessage="Enter moe than 8 password"
ForeColor="Red"></asp:CustomValidator>
                    <br />
                </td>
            </tr>
            <tr>
                <td class="auto-style6">
                    <asp:Label ID="Label8" runat="server" Text="Confirm
password"></asp:Label>
                </td>
                <td class="auto-style7">
                    <asp:TextBox ID="TextBox6" runat="server"
CssClass="auto-style19" Width="272px"></asp:TextBox>
                </td>
                <td class="auto-style8">
                    <br />
                    <asp:CompareValidator ID="CompareValidator1" runat="server"
ControlToCompare="TextBox4" ControlToValidate="TextBox6"
ErrorMessage="CompareValidator" ForeColor="Red"
ValidationGroup="Submit"></asp:CompareValidator>
                    <br />
                </td>
            </tr>
            <tr>
                <td class="auto-style12">
                    <asp:Label ID="Label10" runat="server" Text="Enter
Class(6-12)"></asp:Label>
                </td>
                <td class="auto-style13">
                    <asp:TextBox ID="TextBox7" runat="server"
CssClass="auto-style20" Width="266px"></asp:TextBox>
                </td>
                <td class="auto-style14">
                    <br />
                    <asp:RangeValidator ID="RangeValidator4" runat="server"
ControlToValidate="TextBox7" ErrorMessage="Enter range between 6-12"
```

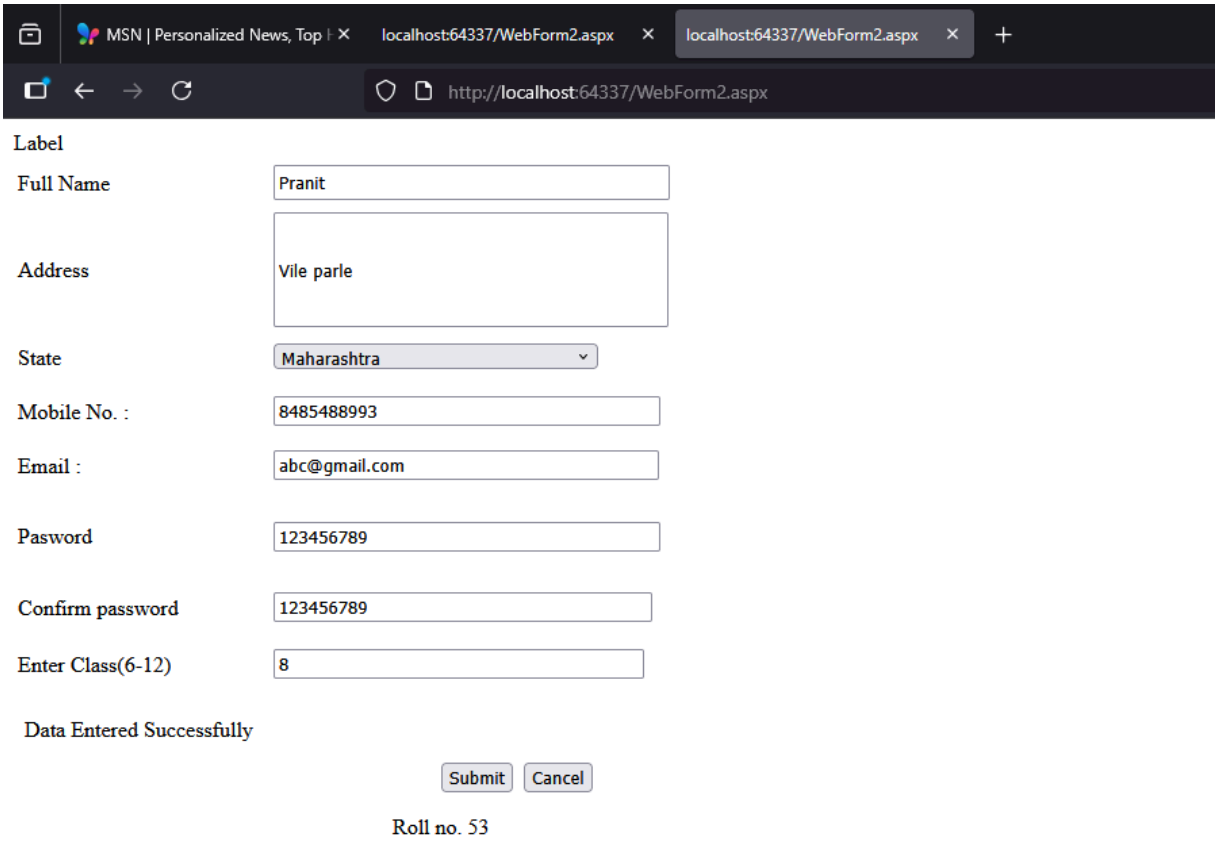
```
ForeColor="Red" Type="Integer" ValidationGroup="Submit"
MaximumValue="12" MinimumValue="6"></asp:RangeValidator>
                    <br />
                </td>
            </tr>
            <asp:Panel ID="Panel1" runat="server">
            </asp:Panel>
        </table>
        <asp:ValidationSummary ID="ValidationSummary1" runat="server" />
        <p>
 
            <asp:Label ID="Label11" runat="server"></asp:Label>
           
           
           
    
        </p>
    </div>
    <p class="auto-style2">
        
        <asp:Button ID="Button1" runat="server" Text="Submit"
ValidationGroup="Submit" OnClick="Button1_Click" />
 <asp:Button ID="Button2" runat="server" Text="Cancel"
OnClick="Button2_Click" />
    </p>
    <p class="auto-style2">
        Roll no. 53</p>
  </form>
</body>
</html>
```

WebForm2.aspx.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

namespace WebApplication7
```

```
{
   public partial class WebForm2 : System.Web.UI.Page
   {


      protected void Button1_Click(object sender, EventArgs e)
      {
         Label11.Text = "Data Entered Successfully";
      }

      protected void Button2_Click(object sender, EventArgs e)
      {
         TextBox1.Text = string.Empty;
         TextBox2.Text = string.Empty;
         TextBox5.Text = string.Empty;
         TextBox3.Text = string.Empty;
         TextBox4.Text = string.Empty;
         TextBox6.Text = string.Empty;
         TextBox7.Text = string.Empty;
         Label11.Visible = false;
      }


   }
}
```

- **Output :**


MSN | Personalized News, Top
localhost:64337/WebForm2.aspx
localhost:64337/WebForm2.aspx
http://localhost:64337/WebForm2.aspx
Label
Full Name
Pranit
Address
Vile parle
State
Maharashtra
Mobile No. :
8485488993
Email :
abc@gmail.com
Pasword
123456789
Confirm password
123456789
Enter Class(6-12)
8
Data Entered Successfully
Submit
Cancel
Roll no. 53

11. Write a program to demonstrate Master Page.

- **Aim :** To design and implement a **Master Page** in **ASP.NET** Web Forms using Visual Studio that defines a common layout for multiple web pages in a web application.

- **Objective :** To understand the concept and purpose of a Master Page in ASP.NET.
  To create a consistent layout (header, footer, and navigation) across multiple content pages.
  To demonstrate how ContentPlaceHolder controls work with Content Pages.
  To learn how to link content pages to the master page.
  To enhance the maintainability and uniformity of a web application using Visual Studio.

- **Theory :** A Master Page in ASP.NET Web Forms provides a template-based approach for creating a consistent look and feel across all the pages of a web application. It defines a common structure (such as header, footer, and menu), while the individual content of each page is defined separately using content pages.

1. Concept of Master Page:

A Master Page acts as a layout container that holds elements shared by multiple web pages.
It uses ContentPlaceHolder controls to define regions where child (content) pages can insert their unique content.

For example, you might have a common website header and navigation bar across all pages, while the main content changes based on the page.

2. Creating a Master Page:

In Visual Studio:

1. Go to **File → New → File → Web Form Master Page** and name it Site.master.
2. Add a **ContentPlaceHolder** control where you want page-specific content to appear.

Example (Site.master):

```
<!DOCTYPE html>
<html>
<head runat="server">
        <title>My Website</title>
</head>
<body>
<div style="background-color: #2196F3; color: white; padding: 10px;">
  <h1>My Website Header</h1>
 </div>

<div style="background-color: #f0f0f0; padding: 10px;">
 <a href="Home.aspx">Home</a> |
 <a href="About.aspx">About</a> |
<a href="Contact.aspx">Contact</a>
</div>

<asp:ContentPlaceHolder ID="MainContent" runat="server" />

<div style="background-color: #2196F3; color: white; text-align:center;
padding: 10px;">
<p>© 2025 My Website. All rights reserved.</p>
</div>
</body>
</html>
```

3. Creating a Content Page:

A content page is linked to a master page and defines content that replaces the placeholder in the master.

Example (Home.aspx):

```
<%@ Page Title="Home" Language="C#" MasterPageFile="~/Site.master" AutoEventWireup="true" %>
<asp:Content ID="Content1" ContentPlaceHolderID="MainContent" runat="server">
    <h2>Welcome to My Website</h2>
<p>This is the home page content that appears inside the master page layout.</p>
</asp:Content>
```

When this page runs, ASP.NET merges the master page layout with the content page's content to form a single complete webpage.

4. Advantages of Using Master Pages:

- Consistency: Ensures a uniform look across all pages.
- Easy Maintenance: Changes in the master page automatically apply to all content pages.
- Reusability: Common design elements (like navigation or headers) can be reused.
- Separation of Design and Content: Designers and developers can work independently.

5. Working Principle:

When a content page is requested:

1. ASP.NET combines the master page layout and the content page content.
2. The final output is a single merged HTML page that is sent to the browser.

- **Code :**

Site1.Master

```
<%@ Master Language="C#" AutoEventWireup="true" CodeBehind="Site1.master.cs" Inherits="WebApplication8.Site1" %>

<!DOCTYPE html>

<html>
<head runat="server">
    <title></title>
    <asp:ContentPlaceHolder ID="head" runat="server">
    </asp:ContentPlaceHolder>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <div style="background-color:#007BFF"; color:white; padding:10px;">
                <h1>My Website Header</h1>
                                        <a href="Home.aspx" style="color.white"; margin-right:10px;">Home</a>
                                    <a href="AboutUs.aspx" style="color.white"; margin-right:10px;">About Us</a>
            </div>
            <div style="padding:20px;">
                <!----Content from child pages will be displayed here-->
                <asp:ContentPlaceHolder ID="ContentPlaceHolder1" runat="server">
                </asp:ContentPlaceHolder>
            </div>
           <div style="background-color:#f1f1f1; text-align:center; padding:10px;">
    <p>All rights reserved.</p>
</div>

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

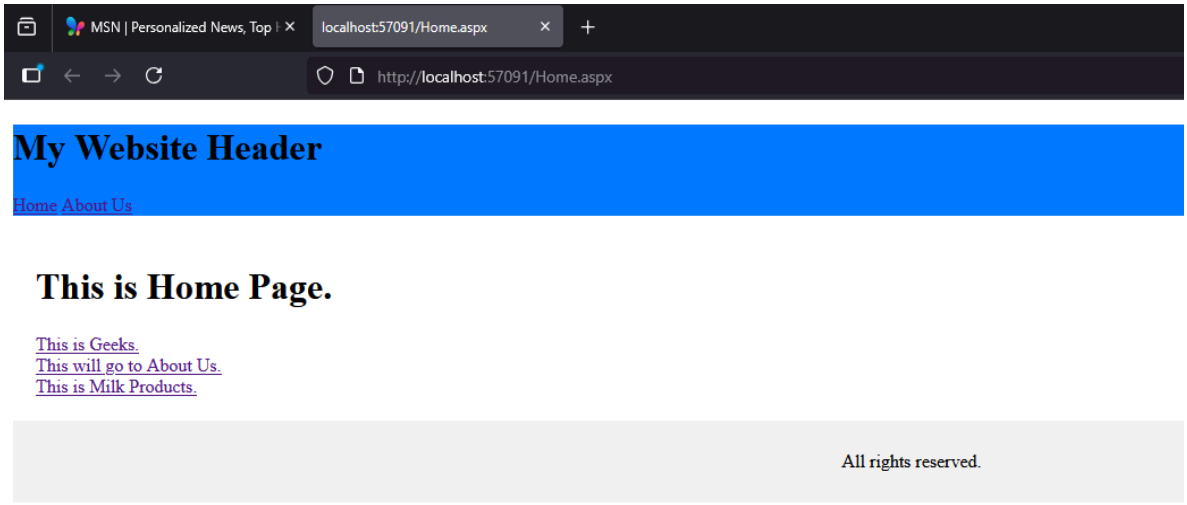
Home.aspx

```
<%@ Page Title="" Language="C#" MasterPageFile="~/Site1.Master" AutoEventWireup="true" CodeBehind="Home.aspx.cs" Inherits="WebApplication8.Home" %>
<asp:Content ID="Content1" ContentPlaceHolderID="head" runat="server">
</asp:Content>
<asp:Content ID="Content2" ContentPlaceHolderID="ContentPlaceHolder1" runat="server">
    <h1 class="center-text">This is Home Page.</h1>
    <a href="https://www.geeksforgeeks.org/">This is Geeks.</a><br />
    <a href="AboutUs.aspx">This will go to About Us.</a><br />
    <a href="https://amul.com/">This is Milk Products.</a>
</asp:Content>
```

AboutUs.aspx

```
<%@ Page Title="" Language="C#" MasterPageFile="~/Site1.Master" AutoEventWireup="true" CodeBehind="AboutUs.aspx.cs" Inherits="WebApplication8.AboutUs" %>
<asp:Content ID="Content1" ContentPlaceHolderID="head" runat="server">
</asp:Content>
<asp:Content ID="Content2" ContentPlaceHolderID="ContentPlaceHolder1" runat="server">
    <h1 class="center-text">This is about us.</h1>
</asp:Content>
```

- **Output :**


MSN | Personalized News, Top
localhost:57091/Home.aspx
http://localhost:57091/Home.aspx
My Website Header
Home About Us
This is Home Page.
This is Geeks.
This will go to About Us.
This is Milk Products.
All rights reserved.

Module - 2

12. Write a program to demonstrate the ViewState.

- **Aim :** To design and implement a web application using ASP.NET Web Forms that demonstrates the concept of ViewState — how data can be preserved between postbacks on the same page.

- **Objective :** To understand the concept of state management in ASP.NET.
  To learn how ViewState preserves data of controls during page postbacks.
  To implement ViewState to store and retrieve values across multiple postback events.
  To differentiate between client-side and server-side state management techniques.
  To demonstrate how ViewState helps maintain the state of form data in ASP.NET Web Forms.

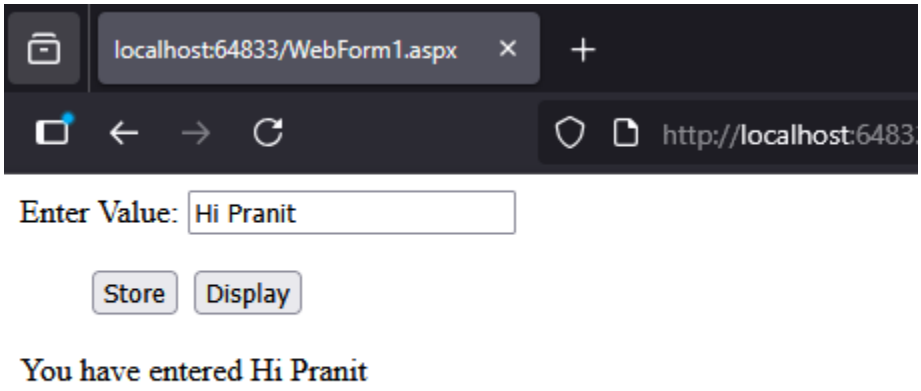
- **Theory:**
  ViewState is a client-side state management technique in ASP.NET used to preserve data of controls between postbacks.
  It stores information in a hidden field called __VIEWSTATE within the same page.
  Data saved in ViewState remains available as long as the user stays on that page.
  It is simple to use, secure, and automatically handled by ASP.NET, but increases page size.

- **Code :**

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

namespace WebApplication12
{
   public partial class WebForm1 : System.Web.UI.Page
   {
```

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

        }

        protected void Button1_Click(object sender, EventArgs e)
        {
            ViewState["msg"] = TextBox1.Text;
        }

        protected void Button2_Click(object sender, EventArgs e)
        {
            Label1.Text += "You have entered " + ViewState["msg"].ToString();
        }
    }
}
```

- **Output :**



\\

13. Write a program to demonstrate the Hidden Object.

- **Aim :** To demonstrate the use of the **HiddenField object** in ASP.NET for storing and retrieving information between postbacks without displaying it on the web page.

- **Objective :**
  To understand the concept of HiddenField in ASP.NET.
  To learn how to store temporary data on the client side using a hidden control.
  To demonstrate how the HiddenField value can be accessed and modified during postbacks.
  To understand the role of hidden fields in maintaining state in web applications.

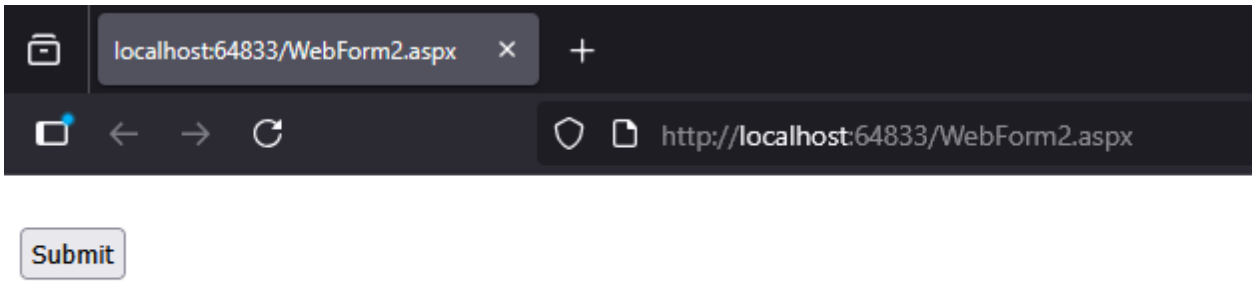
- **Theory:**
  The **HiddenField** control in ASP.NET is used to store small amounts of information that are not visible to the user but are sent to the server during postbacks.
  It helps maintain data without using **ViewState**, **Session**, or **Cookies**.
  The data is stored as an invisible HTML input element (<input type="hidden" />) on the webpage.
  Hidden fields are useful for preserving non-sensitive data like user IDs, counter values, or temporary states across postbacks.

- **Code :**
  Use 2 .aspx files add 1 HiddenField and 1 Button in one file.

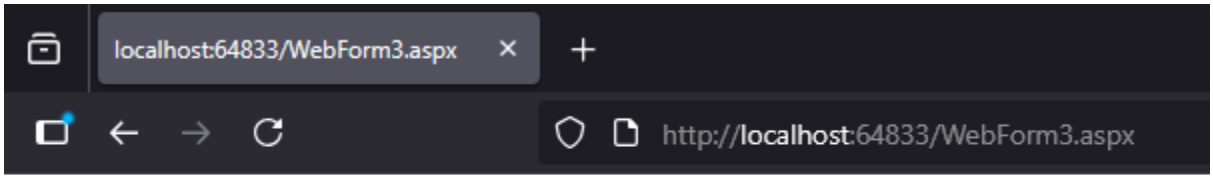
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

namespace WebApplication12
{
   public partial class WebForm3 : System.Web.UI.Page
   {
      protected void Page_Load(object sender, EventArgs e)
      {
```

```
            Response.Write(Request.Form["HiddenField1"].ToString());
        }

        protected void Button1_Click(object sender, EventArgs e)
        {

        }
    }
}
```

- **Output :**



Submit



Pranit

14. Write a program to demonstrate the Query String.

- **Aim :** To demonstrate how to use a Query String in ASP.NET to pass data from one web page to another through the URL.

- **Objective :**
  To understand the concept of Query String as a client-side state management technique.
  To learn how to send and retrieve values between pages using the URL.
  To demonstrate how to access query string values using the **Request.QueryString** object.
  To identify the advantages and limitations of using query strings for data transfer.

- **Theory:**
  A **Query String** is a part of a URL used to pass small amounts of information between web pages in key–value pair format (e.g., Page2.aspx?name=John).
  Data is appended to the URL using a **?** symbol followed by key–value pairs separated by **&**.
  On the target page, values can be retrieved using Request.QueryString["key"].
  Query strings are simple to implement and require no server resources but are visible in the browser's address bar and unsuitable for sensitive data.

- **Code :**

  3_Query_String.aspx

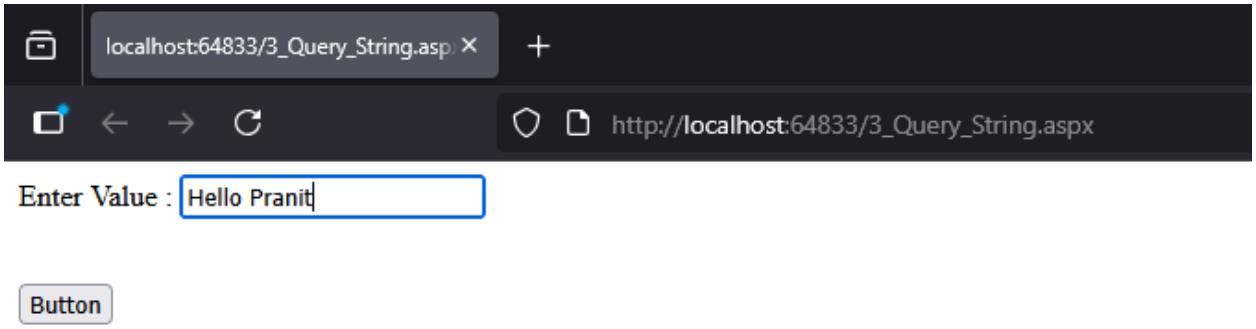
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

namespace WebApplication12
{
   public partial class _3_Query_String : System.Web.UI.Page
   {
```

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

        }

        protected void Button1_Click(object sender, EventArgs e)
        {
            Response.Redirect("QueryString2aspx.aspx?msg=" + TextBox1.Text);
        }
    }
}
```

QueryString2aspx.aspx

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

namespace WebApplication12
{
    public partial class QueryString2aspx : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
            if (Request.QueryString["msg"]!=null)
            {
                Response.Write(Request.QueryString["msg"].ToString());
            }
            else
            {
                Response.Redirect("3_Query_String.aspx");
            }
        }
    }
}
```

- **Output :**


localhost:64833/3_Query_String.asp
http://localhost:64833/3_Query_String.aspx
Enter Value : Hello Pranit
Button

15. Create a web application to get following information from a user. Name, Age, height, Email, Gender. Validate the input by using proper validation control. If the gender is male then navigate to Male.aspx web page, and if the gender is female then navigate to Female.aspx. Then depending upon the gender and height of the individual suggest the ideal weight.
[Note : use cookies to pass information between pages]

The height to weight ratio is as follows

Height (CM)	**150**	**160**	**170**	**180**	**190**
Ideal weight for Male	**60**	**65**	**70**	**75**	**80**
Ideal weight for Female	**55**	**60**	**65**	**70**	**75**

- **Aim :** To create a web application in ASP.NET that collects user details such as name, age, height, email, and gender, validates the inputs, and navigates to a respective page (Male.aspx or Female.aspx). The program should use cookies to pass user information between pages and display the ideal weight based on gender and height.

- **Objective :**
  To learn how to collect and validate user input using ASP.NET validation controls.
  To understand how to use cookies for transferring data between web pages.
  To demonstrate page navigation using conditions (male or female).
  To compute and display the ideal weight of the user based on their gender and height.

- **Theory:**
  1. **Cookies** are small pieces of data stored on the client's browser that help maintain information between multiple page requests.
     In ASP.NET, cookies are created using:
     `Response.Cookies["Name"].Value = txtName.Text;`

and retrieved using:
string name = Request.Cookies["Name"].Value;

2. **Validation Controls** such as RequiredFieldValidator, RangeValidator, and RegularExpressionValidator ensure that the user enters valid information before submission.

3. **Conditional Navigation** can be achieved by checking the gender selected by the user and redirecting accordingly using:

```
if (rblGender.SelectedValue == "Male")
   Response.Redirect("Male.aspx");
else
  Response.Redirect("Female.aspx");
```

4. On the redirected page, the program retrieves height and gender from cookies and displays the corresponding **ideal weight** based on a predefined table.

5. This approach demonstrates **client-side state management** using cookies and dynamic **server-side navigation** based on user input.

- **Code :**
  4_Cookies.aspx

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

namespace WebApplication12
{
  public partial class _4_Cookies : System.Web.UI.Page
  {
    protected void Page_Load(object sender, EventArgs e)
    {
    }
```

```
        protected void Button1_Click(object sender, EventArgs e)
        {
            HttpCookie cookie = new HttpCookie("data");
            cookie.Values["name"]=TextBox1.Text;
            cookie.Values["age"] = TextBox2.Text;
            cookie.Values["height"] = TextBox3.Text;
            cookie.Values["email"] = TextBox4.Text;
            string gender = "";
            if (RadioButton2.Checked) { gender = "Male"; }
            if (RadioButton1.Checked) { gender = "Female"; }

            cookie.Values["gender"]=gender;

            Response.Cookies.Add(cookie);
            if (gender == "Male")
            {
                Response.Redirect("~/male.aspx");
            }
            else
            {

                Response.Redirect("~/female.aspx");
            }

        }

        protected void Unnamed1_CheckedChanged(object sender, EventArgs e)
        {

        }
    }
}
```

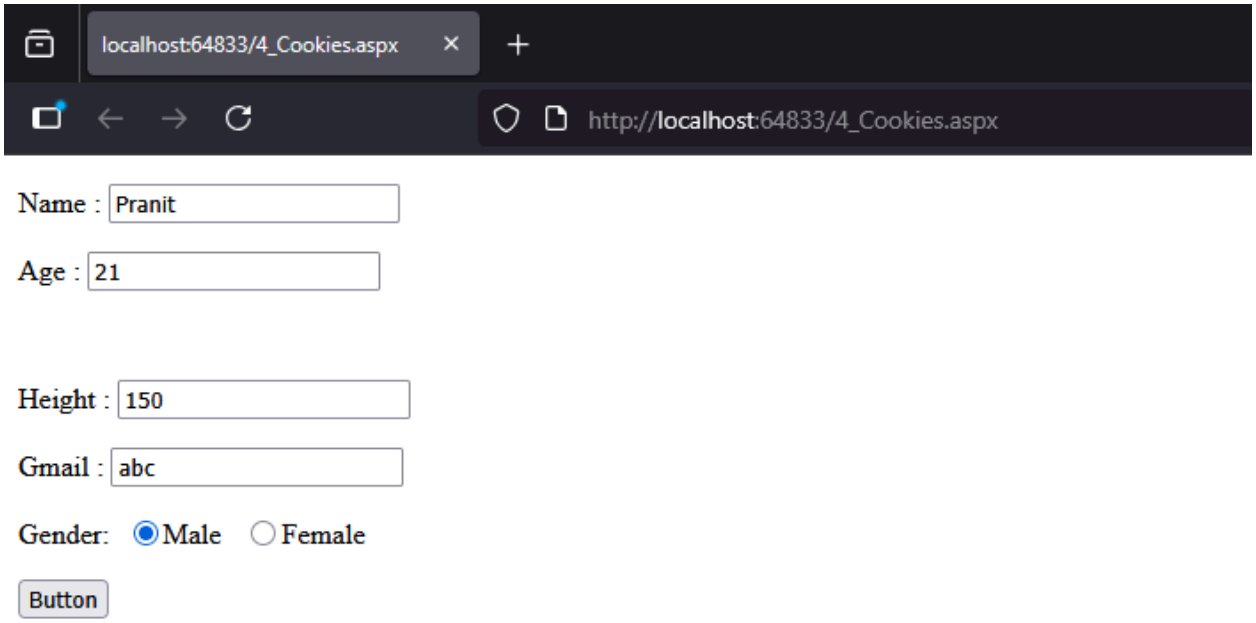
male.aspx

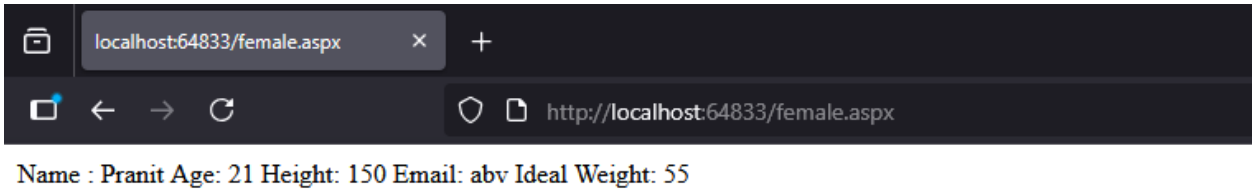
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

namespace WebApplication12
{
   public partial class male : System.Web.UI.Page
   {
      protected void Page_Load(object sender, EventArgs e)
      {
         int weight = 0;
         string name = Request.Cookies["data"].Values["name"];
         string age = Request.Cookies["data"].Values["age"];
         string height = Request.Cookies["data"].Values["height"];
         string email = Request.Cookies["data"].Values["email"];
         if (int.Parse(height) == 150) { weight = 60; }
         if (int.Parse(height) == 160) { weight = 65; }
         if (int.Parse(height) == 170) { weight = 70; }
         if (int.Parse(height) == 180) { weight = 75; }
         if (int.Parse(height) == 190) { weight = 80; }
         Response.Write("Name : " + name + "\nAge: " + age + "\nHeight: " +
height + "\nEmail: " + email+"\nIdeal Weight: "+weight);
      }
   }
}
```

female.aspx

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

namespace WebApplication12
{
   public partial class female : System.Web.UI.Page
   {
      protected void Page_Load(object sender, EventArgs e)
      {
         int weight = 0;
         string name = Request.Cookies["data"].Values["name"];
         string age = Request.Cookies["data"].Values["age"];
         string height = Request.Cookies["data"].Values["height"];
         string email = Request.Cookies["data"].Values["email"];
         if (int.Parse(height) == 150) { weight = 55; }
         if (int.Parse(height) == 160) { weight = 60; }
         if (int.Parse(height) == 170) { weight = 65; }
         if (int.Parse(height) == 180) { weight = 70; }
         if (int.Parse(height) == 190) { weight = 75; }
         Response.Write("Name : " + name + "\nAge: " + age + "\nHeight: " +
height + "\nEmail: " + email + "\nIdeal Weight: " + weight);


      }

      protected void TextBox2_TextChanged(object sender, EventArgs e)
      {

      }
   }
}
```

- **Output :**


localhost:64833/4_Cookies.aspx
http://localhost:64833/4_Cookies.aspx
Name : Pranit
Age : 21
Height : 150
Gmail : abc
Gender: Male Female
Button


localhost:64833/female.aspx
http://localhost:64833/female.aspx
Name : Pranit Age: 21 Height: 150 Email: abv Ideal Weight: 55

16. Write a program to demonstrate session state.

- **Aim :** To demonstrate the use of Session State in ASP.NET to store and retrieve user-specific information across multiple web pages during a user session.

- **Objective :**
  To understand the concept of Session State in ASP.NET.
  To learn how to store and access data using the Session object.
  To maintain user information across different web pages within the same session.
  To demonstrate how session data is automatically cleared when the session ends.

- **Theory :**
  Session State is a server-side state management technique in ASP.NET used to store user-specific data that needs to persist across multiple requests or pages.

  Each user is assigned a unique Session ID, which is used by the server to track data for that user.

  Data can be stored in a session using:
  `Session["UserName"] = txtName.Text;`
  and retrieved using:
  `string name = Session["UserName"].ToString();`

  Session variables are stored on the server and are available until the user closes the browser, logs out, or the session times out.

  Session state is useful for maintaining user login information, shopping cart data, or any data that must persist during a user's visit to a website.

- **Code :**
  WebForm1.aspx

  ```
  using System;
  using System.Collections.Generic;
  using System.Linq;
  using System.Web;
  using System.Web.UI;
  using System.Web.UI.WebControls;
  ```

```
namespace WebApplication13
{
   public partial class WebForm1 : System.Web.UI.Page
   {
      protected void Page_Load(object sender, EventArgs e)
      {

      }

      protected void Button1_Click(object sender, EventArgs e)
      {
         Session["Fname"] = TextBox1.Text;
         Session["Lname"] = TextBox2.Text;

         Response.Redirect("Session_2.aspx");
      }
   }
}
```

Session_2.aspx

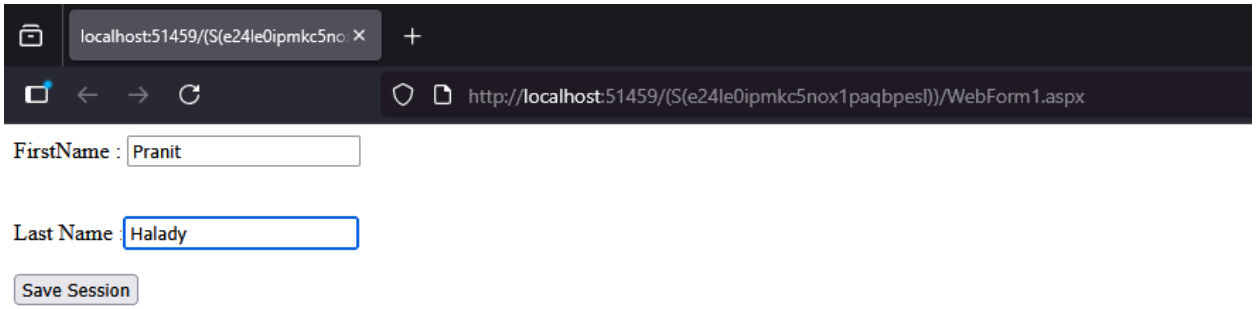
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

namespace WebApplication13
{
   public partial class Session_2 : System.Web.UI.Page
   {
      protected void Page_Load(object sender, EventArgs e)
      {
         if (Session["Fname"]!=null&& Session["Lname"]!=null)
         {
            string fname = Session["Fname"].ToString();
            string lname=Session["Lname"].ToString();
            Response.Write("First name= " + fname + "\nLast name= " + lname);
         }
```

```
            else
            {
                Response.Redirect("WebForm1.aspx");
            }

        }
    }
}
```

Web.config

```
<system.web>
 <sessionState mode="InProc" cookieless="UseUri" timeout="1"></sessionState>
  <compilation debug="true" targetFramework="4.7.2" />
  <httpRuntime targetFramework="4.7.2" />
</system.web>
```

- **Output :**



17. Write a program to count the number of live users in your web application.

- **Aim :** To develop an ASP.NET web application that counts and displays the number of active (live) users currently using the website by using Application State and Session State objects.

- **Objective :**
  To understand how to track the number of users currently active in a web application.
  To learn how to use Application State and Session State for maintaining user count.
  To increment the user count when a new session starts and decrement it when a session ends.
  To display the total number of live users dynamically on the webpage.

- **Theory :**
  Live user tracking can be implemented using the Application object in ASP.NET, which is shared across all sessions.
  The counter is incremented in the Session_Start event and decremented in Session_End in the Global.asax file.
  Application state ensures that the counter value is accessible to all users and persists for the application's lifetime.
  This technique helps monitor website traffic and manage resources effectively.

- **Code :**
  Global.asax

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Security;
using System.Web.SessionState;

namespace WebApplication13
{
   public class Global : System.Web.HttpApplication
   {
      protected void Application_Start(object sender, EventArgs e)
```

```
        {
            Application["sessioncount"] = 0;
        }

        protected void Session_Start(object sender, EventArgs e)
        {
            Application.Lock();
            int count = (int)Application["sessioncount"];
            Application["sessioncount"]=count+1;
            Application.UnLock();
        }

        protected void Application_BeginRequest(object sender, EventArgs e)
        {
        }

        protected void Application_AuthenticateRequest(object sender, EventArgs e)
        {

        }

        protected void Application_Error(object sender, EventArgs e)
        {
        }

        protected void Session_End(object sender, EventArgs e)
        {
            Application.Lock();
            int count = (int)Application["sessioncount"];
            Application["sessioncount"] = count - 1;
            Application.UnLock();

        }

        protected void Application_End(object sender, EventArgs e)
        {

        }
    }
}
```

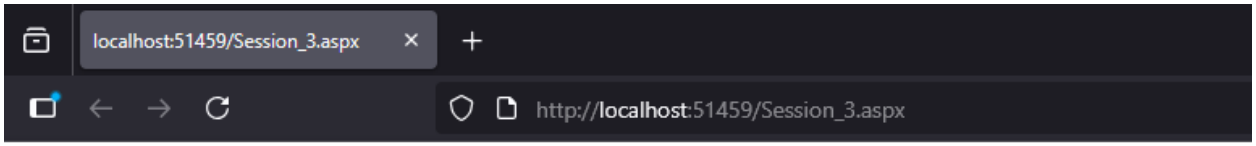
Session_3.aspx

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

namespace WebApplication13
{
   public partial class Session_3 : System.Web.UI.Page
   {
      protected void Page_Load(object sender, EventArgs e)
      {
         Response.Write("Total Visitors= " +
Application["sessioncount"].ToString());

      }
   }
}
```

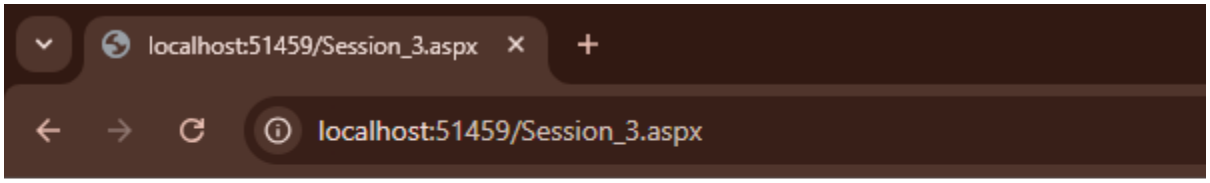
Web.config :

```
 <system.web>
  <sessionState mode="InProc" timeout="1"></sessionState>
   <compilation debug="true" targetFramework="4.7.2" />
   <httpRuntime targetFramework="4.7.2" />
 </system.web>
```

- **Output :**



Total Visitors= 1



Total Visitors= 2

18. Write a web application to display digital clock using ajax.

- **Aim :** To create a web application in ASP.NET that displays a real-time digital clock on the webpage using AJAX for automatic updates without refreshing the page.

- **Objective :**
  To understand how AJAX allows partial page updates in ASP.NET.
  To implement a digital clock that updates every second dynamically.
  To reduce server load by avoiding full page postbacks.
  To enhance user experience with real-time data display.

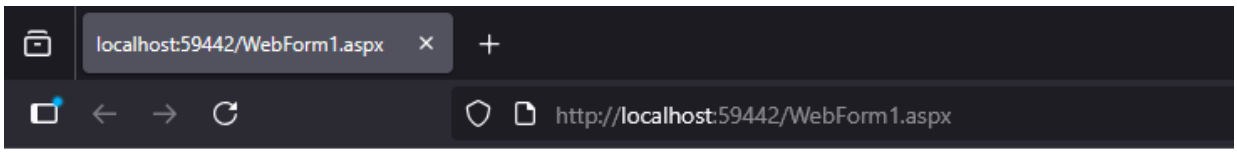
- **Theory :**
  AJAX (Asynchronous JavaScript and XML) enables the webpage to communicate with the server asynchronously without full page reloads.
  In ASP.NET, the UpdatePanel control allows partial page updates for dynamic content like a digital clock.
  A timer control can trigger the UpdatePanel every second to refresh the clock display.
  This approach improves performance, reduces flickering, and provides a smooth real-time experience.

- **Code :**
  WebForm1.aspx

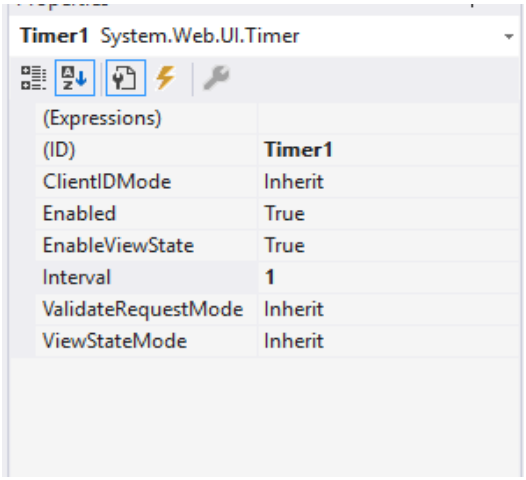
```
<%@ Page Language="C#" AutoEventWireup="true"
CodeBehind="WebForm1.aspx.cs" Inherits="WebApplication14.WebForm1" %>

<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
   <title></title>
</head>
<body>
   <form id="form1" runat="server">
      <div>
         <asp:ScriptManager ID="ScriptManager1" runat="server">
         </asp:ScriptManager>
         <asp:UpdatePanel ID="UpdatePanel1" runat="server">
```

```
            <ContentTemplate>
                <asp:Timer ID="Timer1" runat="server" OnTick="Timer1_Tick"
Interval="1"></asp:Timer>
                <asp:Label ID="Label1" runat="server" Text="Label"
BackColor="Yellow"></asp:Label>
            </ContentTemplate>
            <Triggers>
                <asp:AsyncPostBackTrigger ControlID="Timer1" EventName="tick"
/>
            </Triggers>
        </asp:UpdatePanel>
        <br />
        <br />

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

- **Output :**



10/14/2025 2:40:44 PM



19. Write a web application to create an image slider using ajax.

- **Aim :** To develop a web application that displays an image slider using AJAX to dynamically load images without reloading the entire page.

- **Objective :**
  To understand the concept of AJAX (Asynchronous JavaScript and XML).
  To implement a dynamic image slider that fetches images from the server asynchronously.
  To demonstrate client-server communication using AJAX.
  To enhance user experience with smooth, real-time content updates.

- **Theory :**
  AJAX (Asynchronous JavaScript and XML) is a technique used to send and receive data from the server without reloading the entire webpage.
  It allows for dynamic content updates, improving the speed and interactivity of web applications.
  In an image slider, AJAX is used to fetch image data asynchronously from the server (e.g., from a database or file).
  The images are then displayed one by one on the client side using JavaScript, providing a smooth sliding effect.
  This approach enhances user experience and reduces server load by avoiding full page refreshes.

- **Code :**
  WebForm1.aspx

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="WebForm1.aspx.cs" Inherits="WebApplication1.WebForm1" %>

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">

<head runat="server">

    <title></title>

</head>

<body>

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

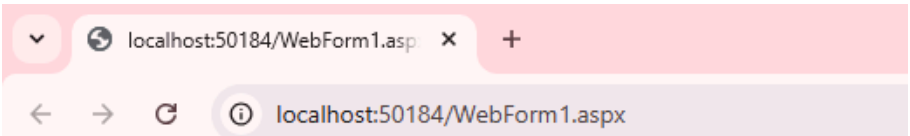
```
        <div>
            <asp:ScriptManager ID="ScriptManager1" runat="server">
            </asp:ScriptManager>
            <br />
            <asp:UpdatePanel ID="UpdatePanel1" runat="server">
                <ContentTemplate>
                    <asp:Image ID="image1" runat="server" Height="300"/>
                                <asp:Timer ID="Timer1" runat="server" Interval="3000"
OnTick="Timer1_Tick" />
                </ContentTemplate>
                <Triggers>
                    <asp:AsyncPostBackTrigger ControlID="Timer1" EventName="tick"
/>
                </Triggers>
            </asp:UpdatePanel>
            <br />
            <br />
            <br />
            <br />
            <br />
        </div>
    </form>
</body>
</html>
```

WebForm1.aspx.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

namespace WebApplication1
{
   public partial class WebForm1 : System.Web.UI.Page
   {
      protected void Page_Load(object sender, EventArgs e)
      {
      }
      static int i = 0;
      protected void Timer1_Tick(object sender, EventArgs e)
      {
         if (i == 0)
         {
            image1.ImageUrl = "~/Images/image1.jpg";
            i = 1;
         }
         else if (i == 1)
         {
```

```
            image1.ImageUrl = "~/Images/image2.jpg";
            i = 0;
        }
        else
        {
            image1.ImageUrl = "~/Images/image3.jpg";
        }
    }
  }
}
```

- **Output :**



20. Write a program to select state and on the basis of state fill city information using ajax.

- **Aim :** To develop a web application that allows the user to select a state and dynamically load the corresponding city names using AJAX without refreshing the page.

- **Objective :**
  To understand the concept of asynchronous data transfer using AJAX.
  To implement client–server communication in ASP.NET Web Forms.
  To dynamically update webpage content without reloading the entire page.
  To enhance user experience through real-time data loading.

- **Theory :**
  AJAX (Asynchronous JavaScript and XML) allows web pages to fetch data from the server asynchronously, meaning parts of a webpage can be updated without reloading the entire page.
  Client-side scripting (JavaScript/jQuery) is used to send and handle requests from the browser to the server.
  Server-side methods (in ASP.NET C#) process these AJAX requests and return data, usually in JSON format.
  The DOM (Document Object Model) is updated dynamically with the received data to reflect new content on the page.
  AJAX improves performance and interactivity, providing a smoother and faster user experience.

- **Code :**
  WebForm2.aspx

  ```
  <%@ Page Language="C#" AutoEventWireup="true" CodeBehind="WebForm2.aspx.cs" Inherits="WebApplication1.WebForm2" %>

  <!DOCTYPE html>

  <html xmlns="http://www.w3.org/1999/xhtml">

  <head runat="server">

      <title></title>

  </head>

  <body>
  ```

```
<form id="form1" runat="server">
    <div>
        <asp:ScriptManager ID="ScriptManager1" runat="server">
        </asp:ScriptManager>
        <br />
        <asp:UpdatePanel ID="UpdatePanel1" runat="server">
            <ContentTemplate>
                Select State:
                <asp:DropDownList ID="DropDownList1" runat="server"
                    AutoPostBack="true"

OnSelectedIndexChanged="DropDownList1_SelectedIndexChanged">
                    <asp:ListItem>Select State</asp:ListItem>
                    <asp:ListItem Value="Maharashtra">Maharashtra</asp:ListItem>
                    <asp:ListItem Value="Goa">Goa</asp:ListItem>
                    <asp:ListItem Value="Gujarat">Gujarat</asp:ListItem>

                </asp:DropDownList>
                <br />
                <br />
                Select City:
                <asp:DropDownList ID="DropDownList2" runat="server">
                </asp:DropDownList>
            </ContentTemplate>
        </asp:UpdatePanel>
```

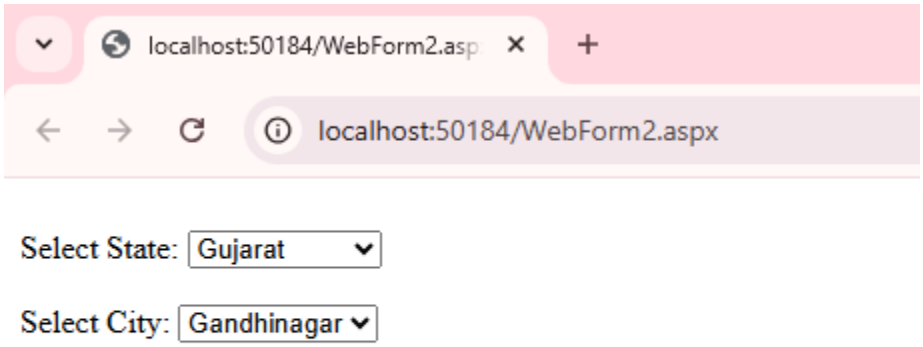
```
        </div>
    </form>
</body>
</html>
```

WebForm2.aspx.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

namespace WebApplication1
{
    public partial class WebForm2 : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
        }
            protected void DropDownList1_SelectedIndexChanged(object sender, EventArgs e)
        {
            DropDownList2.Items.Clear();
            if(DropDownList1.SelectedItem.Text=="Maharashtra")
```

```
        {
            DropDownList2.Items.Add("Mumbai");
            DropDownList2.Items.Add("Pune");
            DropDownList2.Items.Add("Nagpur");
            DropDownList2.Items.Add("Nashik");
        }
        if (DropDownList1.SelectedItem.Text == "Goa")
        {
            DropDownList2.Items.Add("Panaji");
            DropDownList2.Items.Add("Margao");
            DropDownList2.Items.Add("Vasco da Gama");
            DropDownList2.Items.Add("Bardez");
        }
        if (DropDownList1.SelectedItem.Text == "Gujarat")
        {
            DropDownList2.Items.Add("Gandhinagar");
            DropDownList2.Items.Add("Ahmedabad");
            DropDownList2.Items.Add("Surat");
            DropDownList2.Items.Add("Rajkot");
        }
    }
  }
}
```

- **Output :**


localhost:50184/WebForm2.asp
localhost:50184/WebForm2.aspx
Select State: Gujarat
Select City: Gandhinagar

21. Write a web application to demonstrate page output caching.

- **Aim :** To create a web application that demonstrates Page Output Caching in ASP.NET, showing how caching improves performance by storing and reusing the output of a web page.

- **Objective :**
  To understand the concept and purpose of page output caching in web applications.
  To implement output caching in an ASP.NET web page.
  To observe how caching reduces server processing time for repeated requests.
  To learn how to set cache duration and control cache behavior.

- **Theory :**
  Definition: Page Output Caching stores the HTML output of a web page so that subsequent requests can be served from the cache instead of reprocessing the page.

  Purpose: It reduces server load and response time by reusing previously generated page output for a specified duration.

  Directive Used: In ASP.NET, caching is enabled using the @OutputCache directive in the .aspx page header.

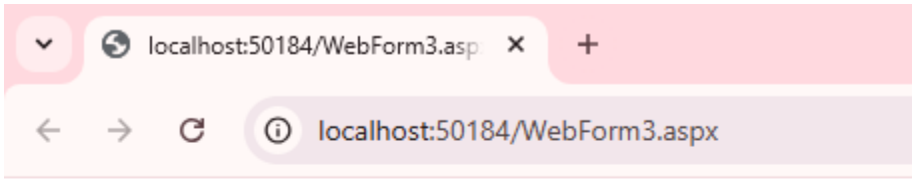
  Duration Attribute: The Duration property specifies the number of seconds the page output is cached.

  VaryByParam: This attribute determines whether the cache should vary based on query string or form parameters (e.g., VaryByParam="None" or VaryByParam="id").

- **Code :**
  Webform3.aspx

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="WebForm3.aspx.cs" Inherits="WebApplication1.WebForm3" %>

<%@ OutputCache Duration="10" VaryByParam="None" %>

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">

<head runat="server">

    <title></title>

</head>

<body>

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

        <div>

            <h1>Output Caching</h1>

            At the tone,the time will be:

            <%= DateTime.Now.ToString() %>

        </div>

    </form>

</body>

</html>
```

- **Output :**



Output Caching

At the tone,the time will be: 11/3/2025 4:21:05 PM

22. Write a web application to demonstrate data caching.

- **Aim :** To develop a web application that demonstrates Data Caching in ASP.NET, showing how data can be stored temporarily to improve application performance and reduce database access.

- **Objective :**
  To understand the concept of data caching in ASP.NET applications.
  To learn how to store and retrieve data from the cache.
  To minimize database access using cached data.
  To study cache expiration and dependency features.

- **Theory :**
  Definition: Data caching is the technique of storing frequently used data in memory so that future requests can access it faster without repeatedly fetching it from the database.

  Purpose: It improves application performance and reduces load on the database server by avoiding redundant data retrieval.

  Cache Object: ASP.NET provides a Cache object (in System.Web.Caching) to store data like datasets, datatables, or objects.

  Expiration: Cached data can be set to expire after a specified time using absolute or sliding expiration.

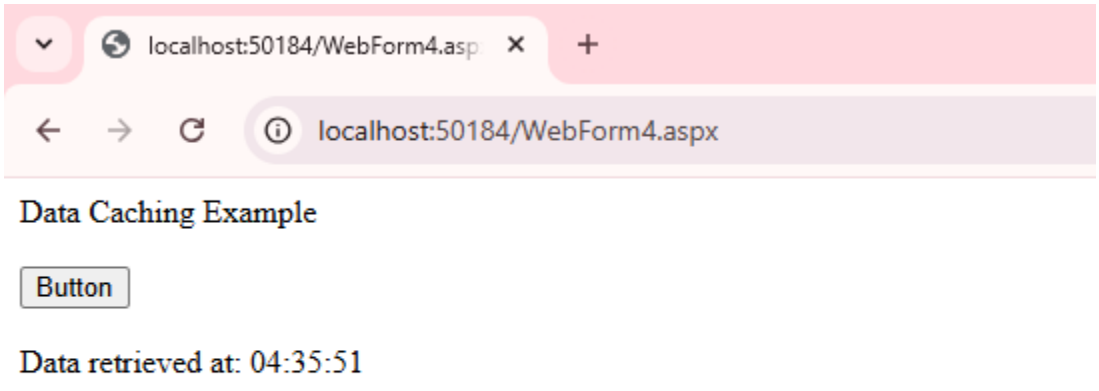
  Retrieval: Before accessing data, the application checks whether it exists in the cache — if not, it loads fresh data and stores it in the cache again.

- **Code :**
  WebForm4.aspx.cs

  ```
  using System;

  using System.Collections.Generic;

  using System.Linq;

  using System.Web;

  using System.Web.UI;

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

```
namespace WebApplication1
{
    public partial class WebForm4 : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
        }
        protected void Button1_Click(object sender, EventArgs e)
        {
            String cachedData = (string)Cache["CachedData"];
            if(cachedData == null )
            {
                cachedData = GetDataFromDatabase();
Cache.Insert("CachedData", cachedData, null, DateTime.Now.AddSeconds(10),TimeSpan.Zero);
            }
            Label1.Text = cachedData;
        }
        private string GetDataFromDatabase()
        {
            return "Data retrieved at: " + DateTime.Now.ToString("hh:mm:ss");
        }
    }
}
```

- **Output :**


localhost:50184/WebForm4.asp
localhost:50184/WebForm4.aspx
Data Caching Example
Button
Data retrieved at: 04:35:51