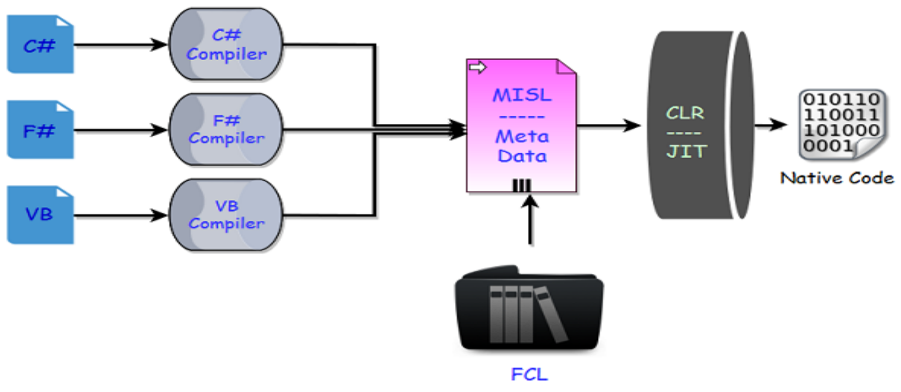
Chapter 5

ASP.Net objects and components

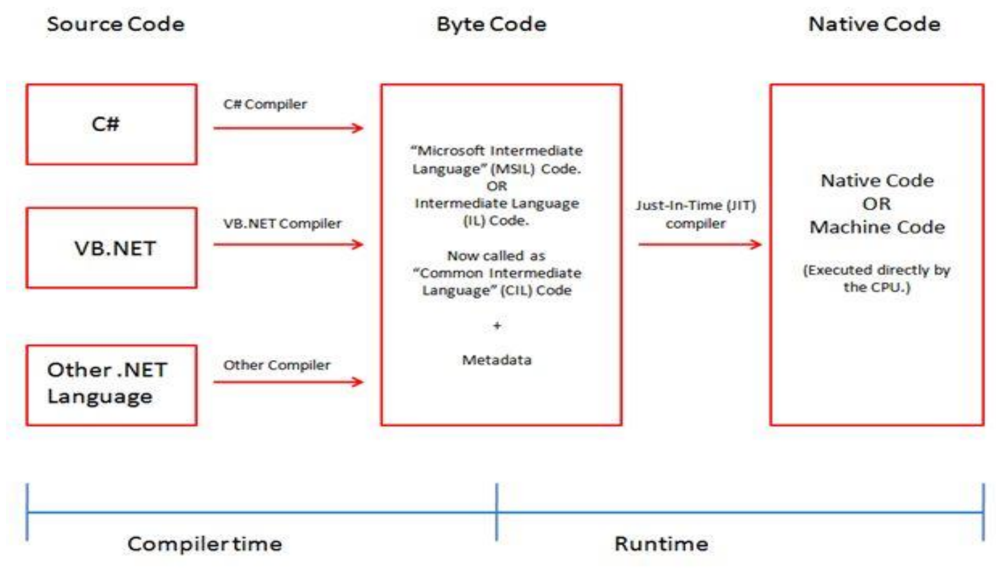
Code Execution Process:

The managed execution process includes the following steps

The Code Execution Process involves the following two stages:
1.Compiler time process.
2.Runtime process.
.



Code Execution Process:


Source Code
Byte Code
Native Code
C#
C# Compiler
VB.NET
VB.NET Compiler
Other .NET Language
Other Compiler
"Microsoft Intermediate Language" (MSIL) Code. OR Intermediate Language (IL) Code. Now called as "Common Intermediate Language" (CIL) Code + Metadata
Just-In-Time (JIT) compiler
Native Code OR Machine Code (Executed directly by the CPU.)
Compilertime
Runtime

Compiler time process

1.The .Net framework has one or more language compilers, such as Visual Basic, C#, Visual C++, JScript, or one of many third-party compilers such as an Eiffel, Perl, or COBOL compiler.

2.Anyone of the compilers translates your source code into Microsoft Intermediate Language (MSIL) code.

3.For example, if you are using the C# programming language to develop an application, when you compile the application, the C# language compiler will convert your source code into Microsoft Intermediate Language (MSIL) code.

4.In short, VB.NET, C#, and other language compilers generate MSIL code. (In other words, compiling translates your source code into MSIL and generates the required metadata.)

5.Currently "Microsoft Intermediate Language" (MSIL) code is also known as the "Intermediate Language" (IL) Code **or** "Common Intermediate Language" (CIL) Code.

SOURCE CODE -----.NET COMLIPER------> BYTE CODE (MSIL + META DATA)

Runtime process.

1.The Common Language Runtime (CLR) includes a JIT compiler for converting MSIL to native code.

2.The JIT Compiler in CLR converts the MSIL code into native machine code that is then executed by the OS.

3.During the runtime of a program, the "Just in Time" (JIT) compiler of the Common Language Runtime (CLR) uses the Metadata and converts Microsoft Intermediate Language (MSIL) into native code.

BYTE CODE (MSIL + META DATA) ----- Just-In-Time (JIT) compiler------> NATIVE CODE

When you compile a C# application or any application written in a CLS-compliant language, the application is compiled into MSIL. This MSIL is then further compiled into native CPU instructions when the application is executed for the first time by the CLR. (Actually, only the called functions are compiled the first time they are invoked.)

- You write source code in C#.
- You then compile it using the C# compiler (csc.exe) into an EXE.
- The C# compiler outputs the MSIL code and a manifest into a read-only part of the EXE that has a standard PE (Win32-portable executable) header.
- When the compiler creates the output, it also imports a function named "_CorExeMain" from the .NET runtime.
- When the application is executed, the operating system loads the PE, as well as any dependent dynamic-link libraries (DLLs), such as the one that exports the "_CorExeMain" function (mscoree.dll), just as it does with any valid PE.
- However, since the operating system obviously can't execute the MSIL code, the entry point is just a small stub that jumps to the "_CorExeMain" function in mscoree.dll.
- The "_CorExeMain" function starts the execution of the MSIL code that was placed in the PE.
- Since MSIL code cannot be executed directly, because it's not in a machine-executable format, the CLR compiles the MSIL using the Just-In-Time (JIT) compiler (or JITter) into native CPU instructions as it processes the MSIL. The JIT compiling occurs only as methods in the program are called. The compiled executable code is cached on the machine and is recompiled only if there's some change to the source code.

ASP.NET Page Life Cycle

- When a page is requested, it is loaded into the server memory, processed, and sent to the browser. Then it is unloaded from the memory.
- At each of these steps, methods and events are available, which could be overridden according to the need of the application.

The page life cycle phases are:
•Initialization
•Instantiation of the controls on the page
•Restoration and maintenance of the state
•Execution of the event handler codes
•Page rendering

- Understanding the page cycle helps in writing codes for making some specific thing happen at any stage of the page life cycle.
- It also helps in writing custom controls and initializing them at right time, populate their properties with view-state data and run control behavior code.

Page Lifecycle stages

Stage	**Description**
Page request	This stage occurs before the lifecycle begins. When a page is requested by the user, ASP.NET parses and compiles that page.
Start	In this stage, page properties such as Request and response are set. It also determines the Request type.
Initialization	In this stage, each control's UniqueID property is set. Master page is applied to the page.
Load	During this phase, if page request is postback, control properties are loaded with information.
Postback event handling	In this stage, event handler is called if page request is postback. After that, the Validate method of all validator controls is called.
Rendering	Before rendering, view state is saved for the page and all controls. During the rendering stage, the page calls the Render method for each control, providing a text writer that writes its output to the OutputStream object of the page's Response property.
Unload	At this stage the requested page has been fully rendered and is ready to terminate.at this stage all properties are unloaded and cleanup is performed.

A requested page first loaded into the server memory after that processes and sent to the bowser. At last it is unloaded from the server memory. ASP.NET provides methods and events at each stage of the page lifecycle that we can use in our application. In the following table, we are tabled events.

ASP.NET Life Cycle Events

Page Event	Typical Use
PreInit	This event is raised after the start stage is complete and before the initialization stage.
Init	This event occurs after all controls have been initialized. We can use this event to read or initialize control properties.
InitComplete	This event occurs at the end of the page's initialization stage. We can use this event to make changes to view state that we want to make sure are persisted after the next postback.
PreLoad	This event is occurs before the post back data is loaded in the controls.
Load	This event is raised for the page first time and then recursively for all child controls.

Control events	This event is used to handle specific control events such as Button control' Click event.
LoadComplete	This event occurs at the end of the event-handling stage. We can use this event for tasks that require all other controls on the page be loaded.
PreRender	This event occurs after the page object has created all controls that are required in order to render the page.
PreRenderComplete	This event occurs after each data bound control whose DataSourceID property is set calls its DataBind method.
SaveStateComplete	It is raised after view state and control state have been saved for the page and for all controls.
Render	This is not an event; instead, at this stage of processing, the Page object calls this method on each control.
Unload	This event raised for each control and then for the page.

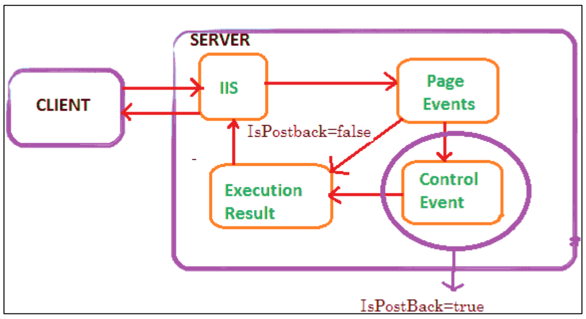
<u>PostBack:</u>

- A web page sending a request back to it self is called as postback.

  this.IsPostBack

- False if it is first request and true if it is next or postback request.

➢ Postback is actually sending all the information from client to web server, then web server process all those contents and returns back to the client. Most of the time ASP control will cause a post back (e. g. buttonclick) but some don't unless you tell them to do In certain events (Listbox Index Changed,RadioButton Checked etc..) in an ASP.NET page upon which a PostBack might be needed.



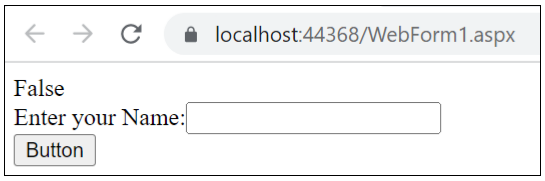
➢ IsPostBack is a property of the Asp.Net page that tells whether or not the page is on its initial load or if a user has perform a button on your web page that has caused the page to post back to itself. The value of the Page.IsPostBack property will be set to true when the page is executing after a postback, and false otherwise. We can check the value of this property based on the value and we can populate the controls on the page.

➢ Is Postback is normally used on page _load event to detect if the web page is getting generated due to postback requested by a control on the page or if the page is getting loaded for the first time.

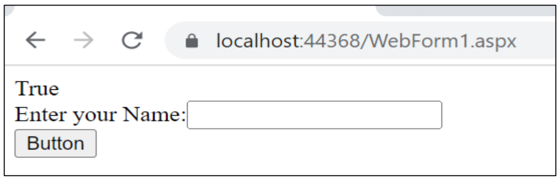
```
protected void Page_Load(object sender, EventArgs e)
{
   if (!IsPostBack)
   {
      // generate form;
   }
   else
   {
      //process submitted data;
   }
}
```

```
WebForm1.aspx
<div>
            Enter your Name:<asp:TextBox runat="server"></asp:TextBox><br />
            <asp:Button ID="Button1" runat="server" Text="Button" OnClick="Button1_Click" />
</div>
```

```
WebForm1.aspx.cs
public partial class WebForm1 : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
            Response.Write(this.IsPostBack);
        }
```



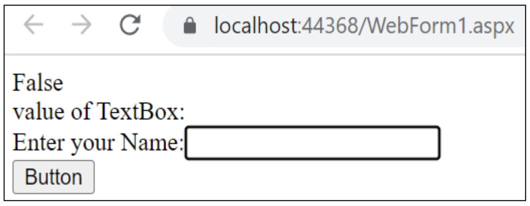
False
Enter your Name:
Button



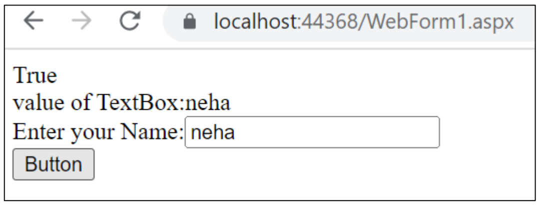
True
Enter your Name:
Button

PostBack Submission--- In a postback all the data that is associated with the form and its control will be submitted to the page on the server

```
public partial class WebForm1 : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
            Response.Write(this.IsPostBack+ "<br>");
            Response.Write("value of TextBox:"+ Textbox1.Text);
        }
```

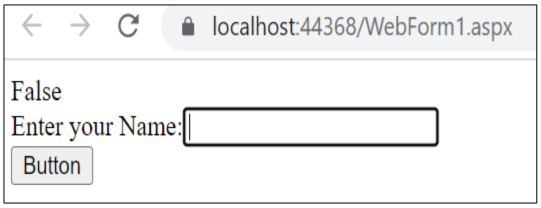


False
value of TextBox:
Enter your Name:
Button

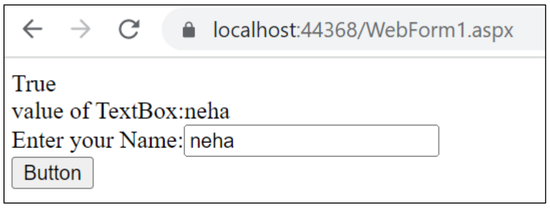


True
value of TextBox:neha
Enter your Name: neha
Button

```
WebForm1.aspx.cs

public partial class WebForm1 : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
            Response.Write(this.IsPostBack+ "<br>");
            if(this.IsPostBack==true)
                                Response.Write("value of TextBox:"+ Textbox1.Text);
        }
```



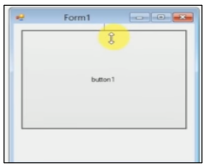
```
localhost:44368/WebForm1.aspx
False
Enter your Name:
Button
```

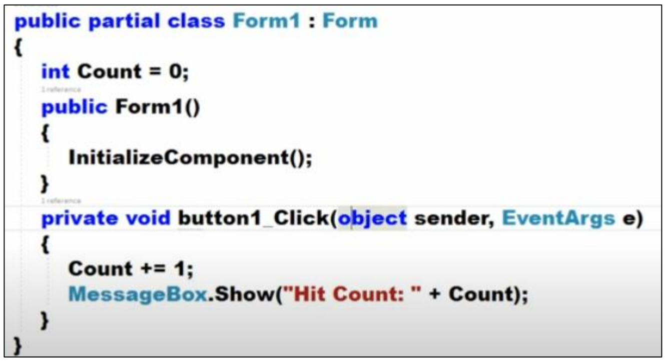


```
localhost:44368/WebForm1.aspx
True
value of TextBox:neha
Enter your Name: neha
Button
```

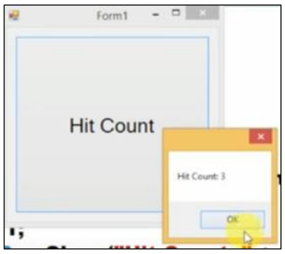
State Management:

Windows application:





```
public partial class Form1 : Form
{
    int Count = 0;
    public Form1()
    {
        InitializeComponent();
    }
    private void button1_Click(object sender, EventArgs e)
    {
        Count += 1;
        MessageBox.Show("Hit Count: " + Count);
    }
}
```



Cross Page Post back:

- ASP.NET by default, submits the form to the same page.
- Cross page posting is submitting the form to a different page.
- This is usually required when you are creating a multi page form to collect information from the user on each page.
- When moving from the source to the target page, the values of controls in the source page can be accessed in the target page.

CrossPagePostbackDemo:

Default.aspx

```
<asp:TextBox ID="txtUserName" runat="server"/>
<asp:TextBox ID="txtLocation" runat="server"/>
<asp:Button ID="btnPostback" Text="Postback" runat="server" PostBackUrl="~/Default2.aspx" />
```

PostBackUrl="~/Default2.aspx" by using this property we will submit **Default.aspx** page control values to **Default2.aspx** page.

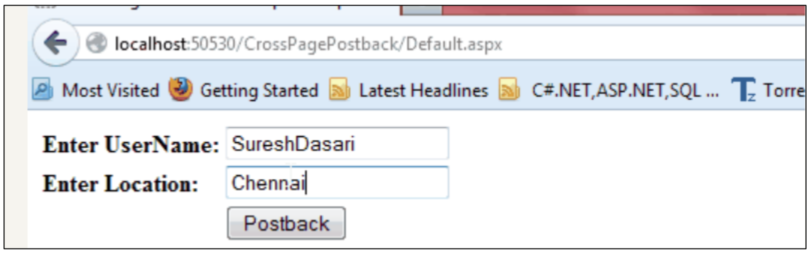
Default2.aspx

```
<label id="lblName" runat="server" /><br /><br />
<label id="lblLocation" runat="server" />
```

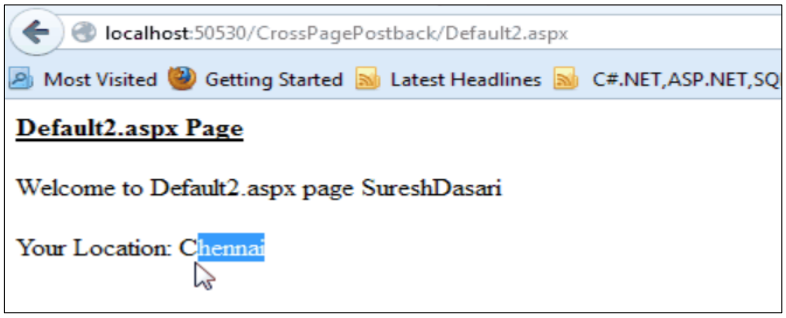
Default2.aspx.cs

```
protected void Page_Load(object sender, EventArgs e)
{
if (PreviousPage != null && PreviousPage.IsCrossPagePostBack)
{
TextBox txtName = (TextBox)PreviousPage.FindControl("txtUserName");
TextBox txtLocation = (TextBox)PreviousPage.FindControl("txtLocation");
lblName.InnerText = "Welcome to Default2.aspx page " + txtName.Text;
lblLocation.InnerText = "Your Location: " + txtLocation.Text;
}
else
{
Response.Redirect("Default.aspx");
}
}
```

OUTPUT of POSTBACK:



Enter UserName: SureshDasari
Enter Location: Chennai



Default2.aspx Page

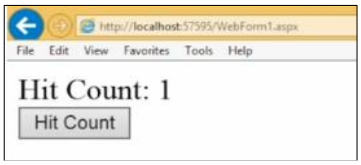
Welcome to Default2.aspx page SureshDasari

Your Location: Chennai

Web Application

```
WebForm1.aspx
        <div>
        <asp:Button ID="Button1" runat="server" Text="Hit Count" OnClick="Button1.Click"/>
        </div>
```

```
WebForm1.aspx.cs

namespace WebApplication5
{
    public partial class WebForm1 : System.Web.UI.Page
    {
        int Count = 0;
        protected void Page_Load(object sender, EventArgs e)
        {
            Count += 1;
            Response.Write("Hit Count" + Count);
        }
    }
}
```



Hit Count: 1

State management
To preserve state of a control, web page, object/data, and user in the application explicitly because all ASP.NET web applications are stateless, i.e., by default, for each page posted to the server, the state of controls is lost. Nowadays all web apps demand a high level of state management from control to application level.

```
-Web applications are stateless.
-State management is a process of maintaining the state of values between
multiple requests of the page(s).

-To main the state of values ASP.Net provides us different options where
those values can be maintianed either on the client machine or on the
server machine.
```

Types of state management
There are two types of state management techniques: client side and server side.

Client side
1.Hidden Field
2.View State
3.Cookies
4.Control State
5.Query Strings

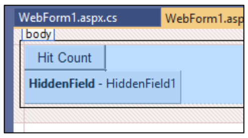
Server side
1.Session
2.Application

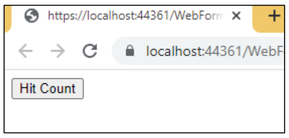
Client side methods

1. Hidden field

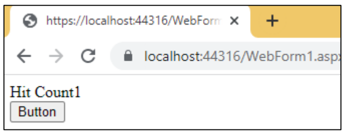
Hidden field is a control provided by ASP.NET which is used to store small amounts of data on the client. It store one value for the variable and it is a preferable way when a variable's value is changed frequently. Hidden field control is not rendered to the client (browser) and it is invisible on the browser. A hidden field travels with every request like a standard control's value.

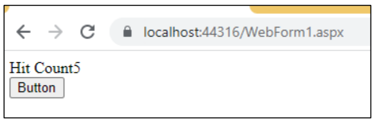
```
WebForm1.aspx
<body>
    <form id="form1" runat="server">
        <div>
        <asp:Button ID="Button1" runat="server" Text="Hit Count" OnClick="Button1.Click"/>

        <asp:HiddenField ID="HiddenField1" runat="server" />
        </div>
    </form>
</body>
```



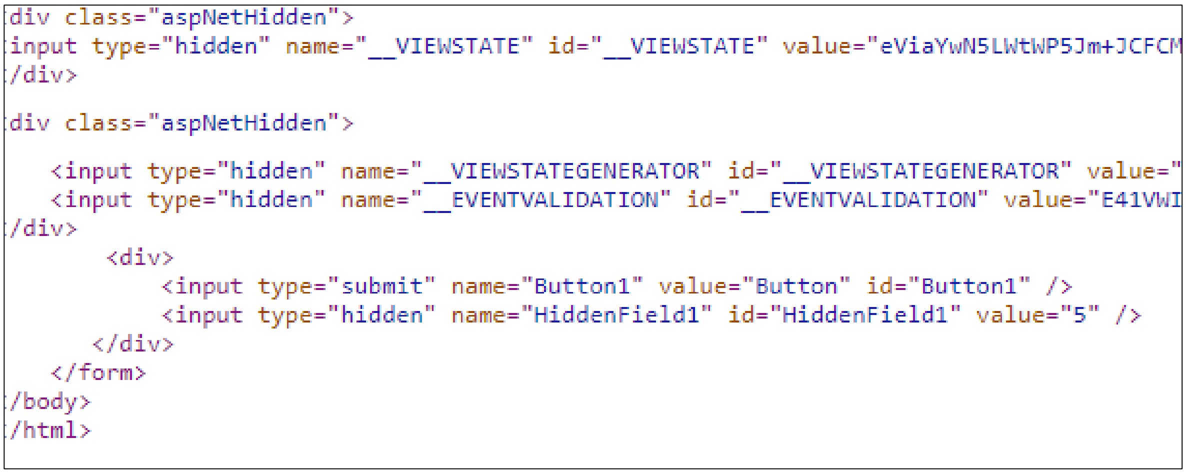


```
WebForm1.aspx.cs

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

        }

        protected void Button1_Click(object sender, EventArgs e)
        {
            Count = int.Parse(HiddenField1.Value);//accessing the value of hidden field,value is
                                                         string use coversion

            Count += 1;
            Response.Write("Hit Count" + Count);
            HiddenField1.Value = Count.ToString();
        }
    }
```


```
Hit Count1
```


```
Hit Count5
```

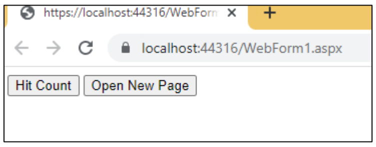
Hidden Field is visible ---It is render on a browser

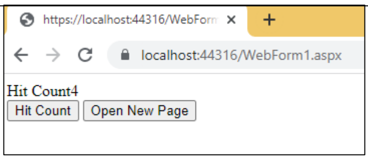

```
div class="aspNetHidden">
input type="hidden" name="__VIEWSTATE" id="__VIEWSTATE" value="eViaYwN5LWtWP5Jm+JCFCM
/div>

div class="aspNetHidden">

   <input type="hidden" name="__VIEWSTATEGENERATOR" id="__VIEWSTATEGENERATOR" value="
   <input type="hidden" name="__EVENTVALIDATION" id="__EVENTVALIDATION" value="E41VWI
/div>
       <div>
           <input type="submit" name="Button1" value="Button" id="Button1" />
           <input type="hidden" name="HiddenField1" id="HiddenField1" value="5" />
      </div>
   </form>
/body>
/html>
```

Sending to another form:

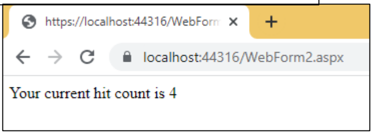
```
WebForm2.aspx
<form id="form1" runat="server">
        <div>
            <asp:Button ID="Button1" runat="server" Text="Hit Count" OnClick="Button1_Click" />
            <asp:Button ID="Button2" runat="server" Text="Open New Page" OnClick="Button2_Click"
PostBackUrl="~/WebForm2.aspx" />
            <asp:HiddenField ID="HiddenField1" runat="server" value="0"/>
       </div>
    </form>
```

```
WebForm2.aspx.cs
public partial class WebForm2 : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
            string Value = Request.Form["HiddenField1"];
            Response.Write("Your current hit count is " + Value);
        }
    }
```


https://localhost:44316/WebForm
localhost:44316/WebForm1.aspx
Hit Count
Open New Page


https://localhost:44316/WebForm
localhost:44316/WebForm1.aspx
Hit Count
Open New Page

Hit Count4


https://localhost:44316/WebForm
localhost:44316/WebForm2.aspx

Your current hit count is 4

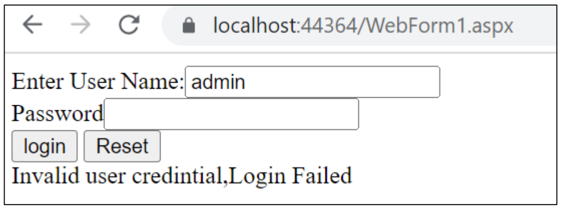
WebForm1.aspx

```
<body>
    <form id="form1" runat="server">
        <div>
            Enter User Name:<asp:TextBox ID="TextBox1" runat="server"></asp:TextBox><br />
            Password<asp:TextBox ID="TextBox2" runat="server"
TextMode="Password"></asp:TextBox><br />

            <asp:Button ID="Button1" runat="server" Text="login" OnClick="Button1_Click" />
            <asp:Button ID="Button2" runat="server" Text="Reset" />
        </div>
        <asp:Label ID="Label1" runat="server" Text="Label"></asp:Label>
    </form>
</body>
```

WebForm1.aspx.cs

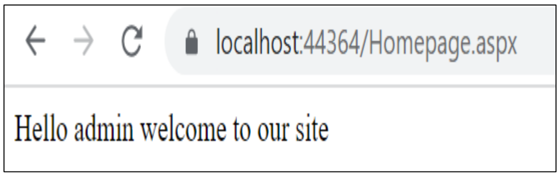
```
namespace WebApplication7
{
    public partial class WebForm1 : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
            if (!IsPostBack)
                TextBox1.Focus();
        }

        protected void Button1_Click(object sender, EventArgs e)
        {
            if (TextBox1.Text == "admin" && TextBox2.Text == "admin")
                Response.Redirect("Homepage.aspx");
            else
                Label1.Text = "Invalid user credintial,Login Failed";
        }
    }
}
```

```
Homepage.aspx.cs

protected void Page_Load(object sender, EventArgs e)
        {
                        Response.Write("Hello admin welcome to our site");
        }
```



localhost:44364/WebForm1.aspx

Enter User Name: admin
Password
login Reset
Invalid user credintial,Login Failed



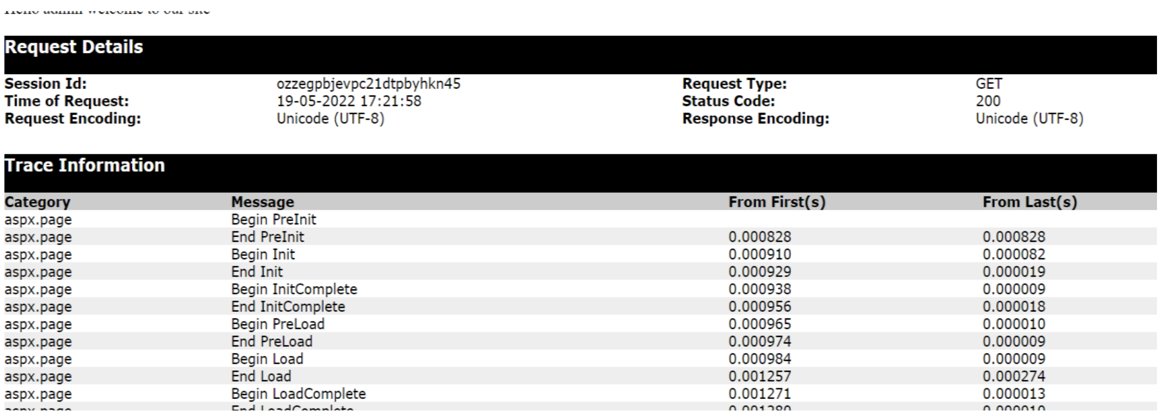
localhost:44364/Homepage.aspx

Hello admin welcome to our site

WebForm1--→Web Browser-→Homepage

TextBox1-→We can not access in homepage

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



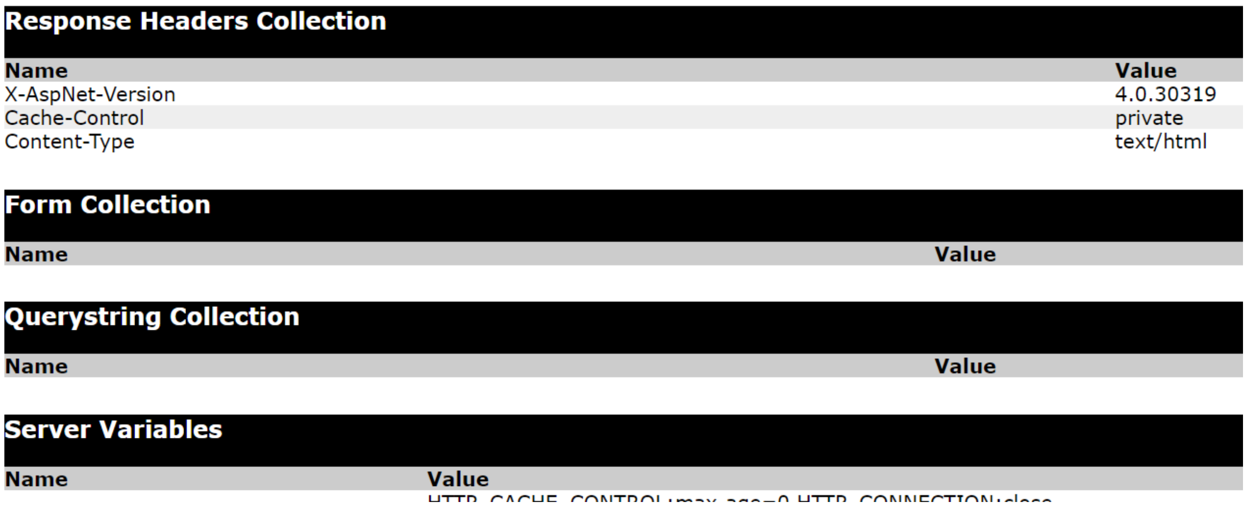
Hello admin welcome to our site

Request Details

Session Id:	ozzegpbjevpc21dtpbyhkn45	**Request Type:**	GET
Time of Request:	19-05-2022 17:21:58	**Status Code:**	200
Request Encoding:	Unicode (UTF-8)	**Response Encoding:**	Unicode (UTF-8)

Trace Information

Category	Message	From First(s)	From Last(s)
aspx.page	Begin PreInit		
aspx.page	End PreInit	0.000828	0.000828
aspx.page	Begin Init	0.000910	0.000082
aspx.page	End Init	0.000929	0.000019
aspx.page	Begin InitComplete	0.000938	0.000009
aspx.page	End InitComplete	0.000956	0.000018
aspx.page	Begin PreLoad	0.000965	0.000010
aspx.page	End PreLoad	0.000974	0.000009
aspx.page	Begin Load	0.000984	0.000009
aspx.page	End Load	0.001257	0.000274
aspx.page	Begin LoadComplete	0.001271	0.000013
aspx.page	End LoadComplete	0.001280	0.000010



Response Headers Collection

Name	Value
X-AspNet-Version	4.0.30319
Cache-Control	private
Content-Type	text/html

Form Collection

Name	Value

Querystring Collection

Name	Value

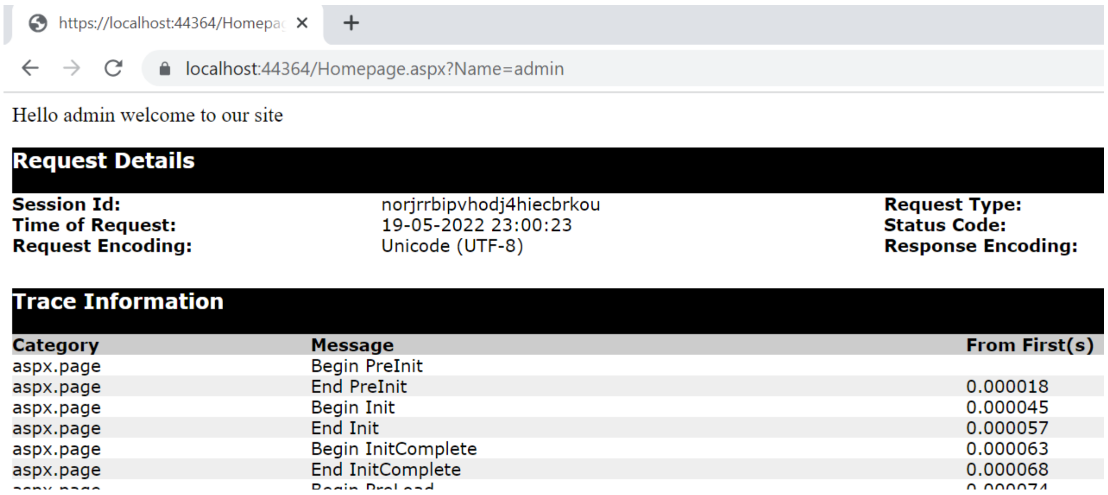
Server Variables

Name	Value
	HTTP_CACHE_CONTROL:max-age=0 HTTP_CONNECTION:close

Using Querystring

```
WebForm1.aspx.cs
protected void Button1_Click(object sender, EventArgs e)
        {
            if (TextBox1.Text == "admin" && TextBox2.Text == "admin")
                Response.Redirect("Homepage.aspx?Name="+ TextBox1.Text);
            else
                Label1.Text = "Invalid user credintial,Login Failed";
        }
```

```
Homepage.aspx.cs

protected void Page_Load(object sender, EventArgs e)
        {
                       Response.Write("Hello admin welcome to our site");
        }
```



https://localhost:44364/Homepa

localhost:44364/Homepage.aspx?Name=admin

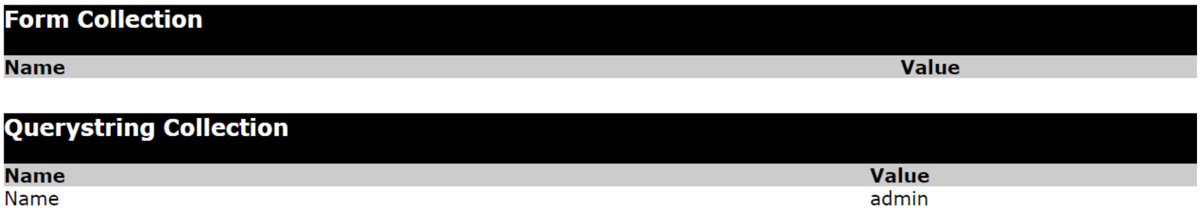
Hello admin welcome to our site

Request Details

Session Id:	norjrrbipvhodj4hiecbrkou	**Request Type:**	
Time of Request:	19-05-2022 23:00:23	**Status Code:**	
Request Encoding:	Unicode (UTF-8)	**Response Encoding:**	

Trace Information

Category	Message	From First(s)
aspx.page	Begin PreInit	
aspx.page	End PreInit	0.000018
aspx.page	Begin Init	0.000045
aspx.page	End Init	0.000057
aspx.page	Begin InitComplete	0.000063
aspx.page	End InitComplete	0.000068



Form Collection

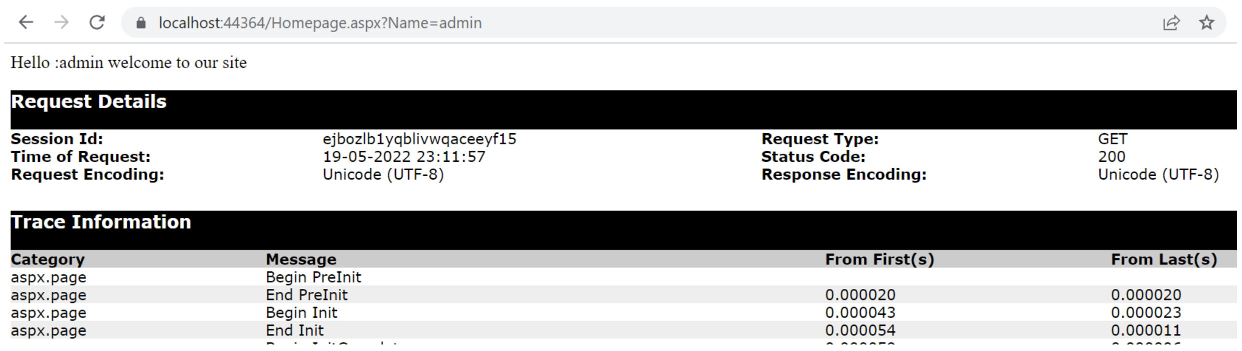
Name	Value

Querystring Collection

Name	Value
Name	admin

```
WebForm1.aspx.cs
protected void Button1_Click(object sender, EventArgs e)
        {
            if (TextBox1.Text == "admin" && TextBox2.Text == "admin")
                Response.Redirect("Homepage.aspx?Name="+ TextBox1.Text);
            else
                Label1.Text = "Invalid user credintial,Login Failed";
        }
```

```
Homepage.aspx.cs

namespace WebApplication7
{
    public partial class Homepage : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
            string Name = Request.QueryString["Name"];
            Response.Write("Hello :"+ Name+ "     welcome to our site");
        }
    }
}
```



localhost:44364/Homepage.aspx?Name=admin

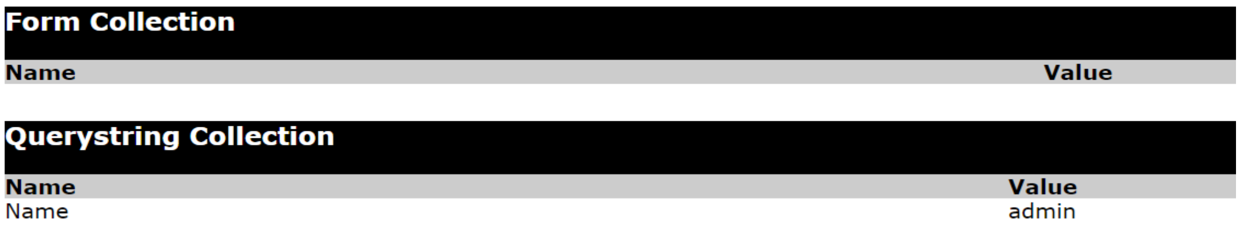
Hello :admin welcome to our site

Request Details

Session Id:	ejbozlb1yqblivwqaceeyf15	**Request Type:**	GET
Time of Request:	19-05-2022 23:11:57	**Status Code:**	200
Request Encoding:	Unicode (UTF-8)	**Response Encoding:**	Unicode (UTF-8)

Trace Information

Category	Message	From First(s)	From Last(s)
aspx.page	Begin PreInit		
aspx.page	End PreInit	0.000020	0.000020
aspx.page	Begin Init	0.000043	0.000023
aspx.page	End Init	0.000054	0.000011



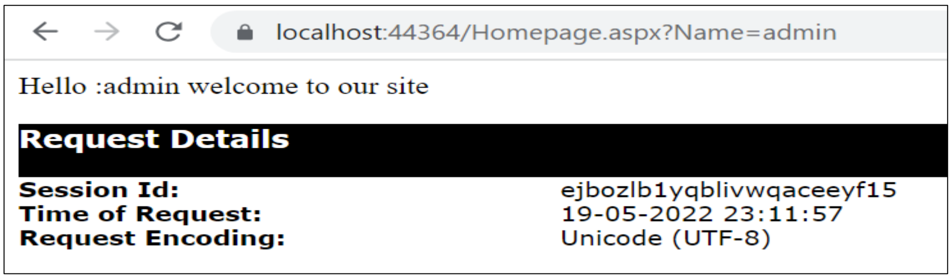
Form Collection

Name	Value

Querystring Collection

Name	Value
Name	admin

OUTPUT



localhost:44364/Homepage.aspx?Name=admin

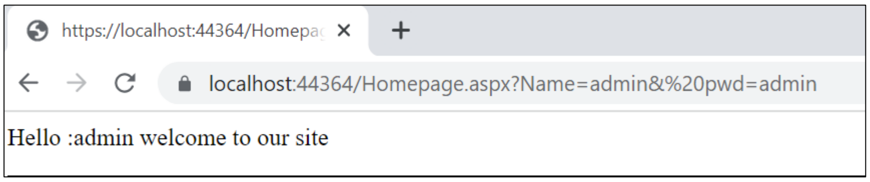
Hello :admin welcome to our site

Request Details

```
Session Id:                 ejbozlb1yqblivwqaceeyf15
Time of Request:            19-05-2022 23:11:57
Request Encoding:           Unicode (UTF-8)
```

Displaying More than 1 value

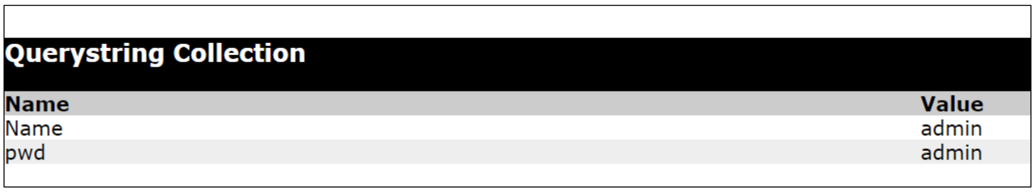
```
        protected void Button1_Click(object sender, EventArgs e)
        {
            if (TextBox1.Text == "admin" && TextBox2.Text == "admin")
                Response.Redirect("Homepage.aspx?Name="+ TextBox1.Text + "& pwd="+TextBox2.Text);
            else
                Label1.Text = "Invalid user credintial,Login Failed";
        }
```



https://localhost:44364/Homepa

localhost:44364/Homepage.aspx?Name=admin&%20pwd=admin

Hello :admin welcome to our site



Querystring Collection

Name	Value
Name	admin
pwd	admin

View State and control state:

ViewState[string key] = value (object)

object value = ViewState[string key]

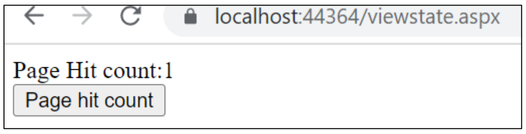
Viewstate.aspx

```
<form id="form1" runat="server">
        <div>
            <asp:Button ID="Button1" runat="server" Text="Page hit count" OnClick="Button1_Click" />
        </div>
    </form>
```

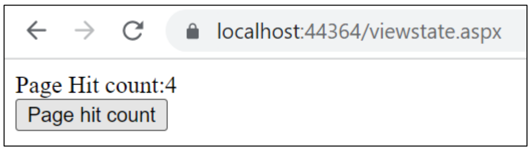
Viewstate.aspx.cs

```
protected void Button1_Click(object sender, EventArgs e)
        {
            int Count = 0;
            if (ViewState["Counter"] == null)
                Count += 1;
            else
                Count = ((int)ViewState["Counter"]) + 1;//return type is object so explicitly
conversion
            ViewState["Counter"] = Count;
            Response.Write("Page Hit count:" + Count);



        }
```

View state stores the value on browser instead of server.


localhost:44364/viewstate.aspx

Page Hit count:1


localhost:44364/viewstate.aspx

Page Hit count:4

Advantages:
1. Values are not stored on server.
2. Values are stored in an encrypted format.

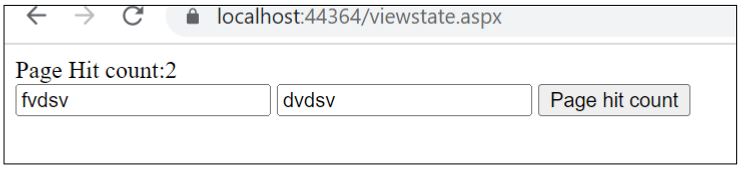
Dis-Advantages:
1. When we have huge volumes of data stored in ViewState all this data will be submitted to the server for every post back and sent back to the browser.
2. Even if viewstate values are encrypted it is still possible to crack the values by de-crypting

-Use ViewState when we want to store any smaller volumes of data.

-Do not use viewstate when your data volume is more and also dont store any sensitive data like passwords because they can be de-crypted.

-All out asp.net server controls by default maintains viewstate and we can call it as control state.

```
<form id="form1" runat="server">
        <div>
            <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
            <asp:TextBox ID="TextBox2" runat="server"></asp:TextBox>
            <asp:Button ID="Button1" runat="server" Text="Page hit count" OnClick="Button1_Click" />
        </div>

    </form>
```


localhost:44364/viewstate.aspx

Page Hit count:2

fvdsv | dvdsv | Page hit count

```
<div>
    <asp:TextBox ID="TextBox1" runat="server" EnableViewState="false"></asp:TextBox>
    <asp:TextBox ID="TextBox2" runat="server" EnableViewState="false"></asp:TextBox>
    <asp:Button ID="Button1" runat="server" Text="Page hit count" OnClick="Button1_Click" />
</div>
```

<u>Cookies:</u>

- ASP.NET Cookie is a small bit of text that is used to store user-specific information. This information can be read by the web application whenever user visits the site.
- When a user requests for a web page, web server sends not just a page, but also a cookie containing the date and time. This cookie stores in a folder on the user's hard disk.
- When the user requests for the web page again, browser looks on the hard drive for the cookie associated with the web page. Browser stores separate cookie for each different sites user visited.
- Cookies can be broadly classified into 2 types:

<u>Persistent cookies:</u>

1. Remain on the clients computer,even after the browser is closed.
2. You can configure how long the cookies remain using the expires property of the HTTPCookie object

<u>Non Persistent cookies:</u>

1. If you don't set the Expires property,then the cookie is called as a Non Persistent cookie.
2. Non persistent cookies only remain in memory until the browser is closed

WebForm1.aspx

```
<body>
    <form id="form1" runat="server">
        <div>
       This is form 1: <br />
    Enter your name:<asp:TextBox ID="TextBox1" runat="server"></asp:TextBox><br />
    Enter your email:<asp:TextBox ID="TextBox2" runat="server"></asp:TextBox><br />
    <asp:Button ID="Button1" runat="server" Text="Go to webform 2"
OnClick="Button1_Click" />
        </div>
    </form>
</body>
```

WebForm1.aspx

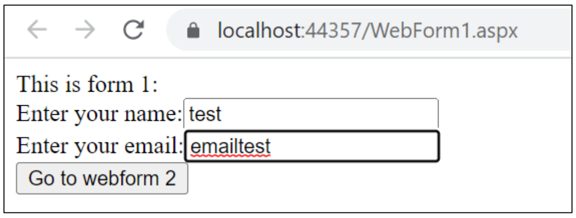
```
protected void Button1_Click(object sender, EventArgs e)
        {
            HttpCookie cookie = new HttpCookie("userinfo");//created a cookie object
given a name
            cookie["username"] = TextBox1.Text;
            cookie["emailid"] = TextBox2.Text;
            //write the cokie object to the client machine
            Response.Cookies.Add(cookie);
            Response.Redirect("WebForm2.aspx");
```

WebForm2.aspx

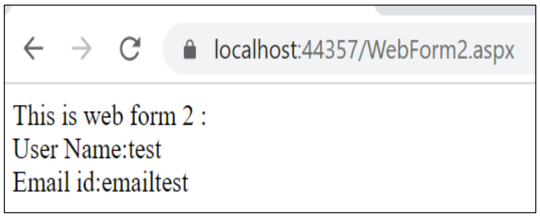
```
<form id="form1" runat="server">
        <div>
   This is web form 2 :<br />
            User Name:<asp:Label ID="Label1" runat="server" Text="Label"></asp:Label><br />
            Email id:<asp:Label ID="Label2" runat="server" Text="Label"></asp:Label>
        </div>
    </form>
```

WebForm2.aspx.cs

```
protected void Page_Load(object sender, EventArgs e)
        {
            HttpCookie cookie =Request.Cookies["userinfo"];
            if(cookie !=null)
            {
            Label1.Text = cookie["username"];
            Label2.Text = cookie["emailid"];
            }
        }
```


localhost:44357/WebForm1.aspx

This is form 1:
Enter your name: test
Enter your email: emailtest
Go to webform 2

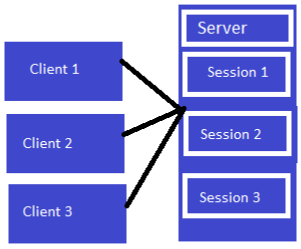

localhost:44357/WebForm2.aspx

This is web form 2 :
User Name:test
Email id:emailtest

```
Non-Persistent Cookies

protected void Button1_Click(object sender, EventArgs e)
        {
            HttpCookie cookie = new HttpCookie("userinfo");//created a cookie
                                                            object given a name
            cookie["username"] = TextBox1.Text;
            cookie["emailid"] = TextBox2.Text;
            cookie.Expires = DateTime.Now.AddDays(30);
            //write the cokie object to the client machine
            Response.Cookies.Add(cookie);
            Response.Redirect("WebForm2.aspx");

        }
```

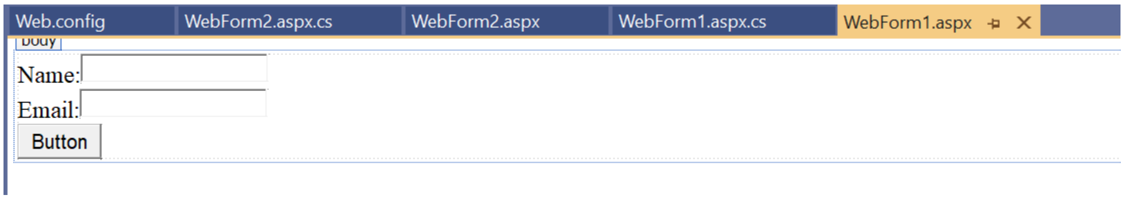
Server side—Session

- HTTP protocol is a stateless protocol; in other words, when a client sends a request to the server, an instance of the page is created and the page is converted to HTML format and then the server provides the response and then the instance of the page and the value of the control are destroyed.
- So if we have a requirement to store the values of the controls and pass them into another web form then a State Management Technique is used.
- Session is a State Management Technique.
- A Session can store the value on the Server. It can support any type of object to be stored along with our own custom objects.
- A session is one of the best techniques for State Management because it stores the data as client-based, in other words, the data is stored for every user separately and the data is secured also because it is on the server.



Session state variables are stored on the web server by default, and are kept for the life time of a session

Example:



Page1.aspx.cs

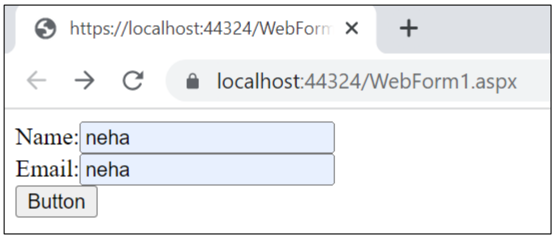
```
protected void Button1_Click(object sender, EventArgs e)
       {
           Session["name"] = TextBox1.Text;
           Session["email"] = TextBox2.Text;
           Response.Redirect("~/Webform2.aspx");
       }
```

```
Page2.aspx

<form id="form1" runat="server">
        <div>
            This is web form 2:<br />
            Name:<asp:Label ID="Label1" runat="server" Text="Label"></asp:Label><br />
            Email:<asp:Label ID="Label2" runat="server" Text="Label"></asp:Label>
        </div>
    </form>
```

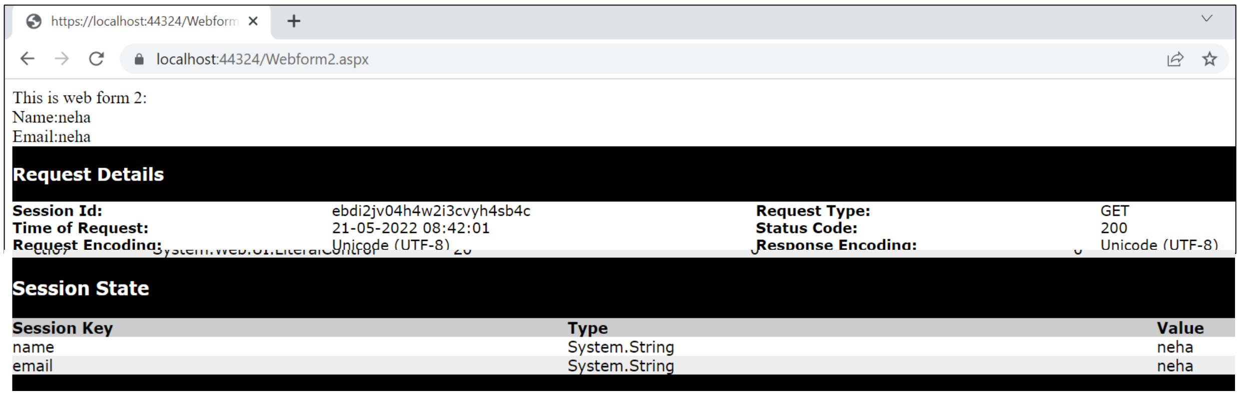
```
Page2.aspx.cs

protected void Page_Load(object sender, EventArgs e)
        {
            Label1.Text = Session["name"].ToString();
            Label2.Text = Session["email"].ToString();
        }
```

```
Web.Config
<system.web>
  <sessionState mode="InProc" timeout="20"></sessionState>
    <compilation debug="true" targetFramework="4.7.2" />
    <httpRuntime targetFramework="4.7.2" />
  </system.web>
```

OUTPUT:


https://localhost:44324/WebForm
localhost:44324/WebForm1.aspx
Button

Name: neha
Email: neha


https://localhost:44324/Webform
localhost:44324/Webform2.aspx

This is web form 2:
Name:neha
Email:neha

Request Details

Session Id:	ebdi2jv04h4w2i3cvyh4sb4c	**Request Type:**	GET
Time of Request:	21-05-2022 08:42:01	**Status Code:**	200
Request Encoding:	Unicode (UTF-8)	**Response Encoding:**	Unicode (UTF-8)

Session State

Session Key	Type	Value
name	System.String	neha
email	System.String	neha

The default state mode is InProc

Session-State Modes:

ASP.NET session state supports several different storage options for session data.

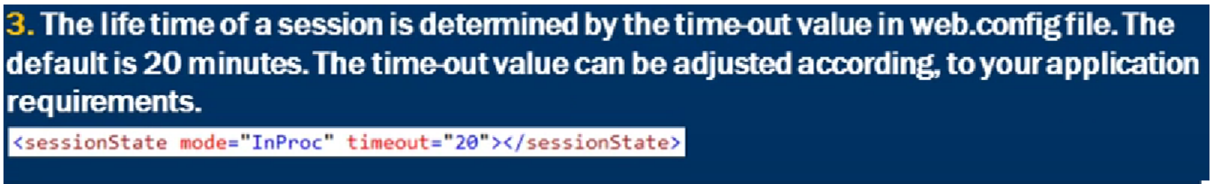
•InProc mode, which stores session state in memory on the Web server. This is the default.

•StateServer mode, which stores session state in a separate process called the ASP.NET state service. This ensures that session state is preserved if the Web application is restarted and also makes session state available to multiple Web servers in a Web form.

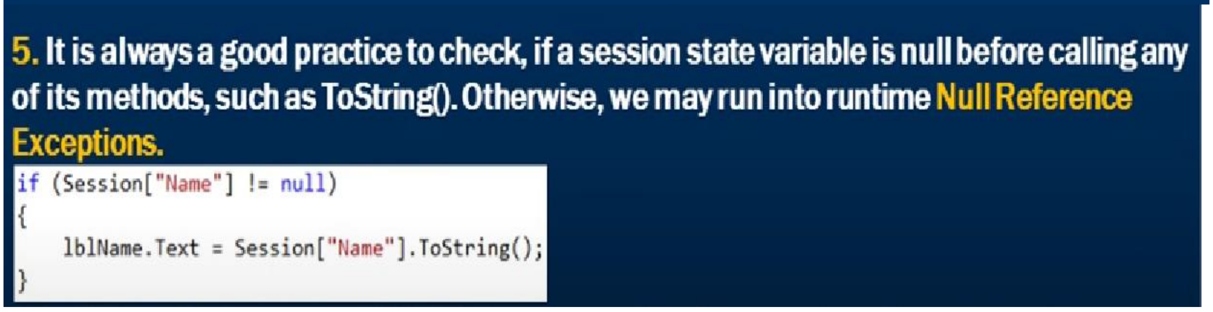
•SQLServer mode stores session state in a SQL Server database. This ensures that session state is preserved if the Web application is restarted and also makes session state available to multiple Web servers in a Web form.

•Custom mode, which enables you to specify a custom storage provider.

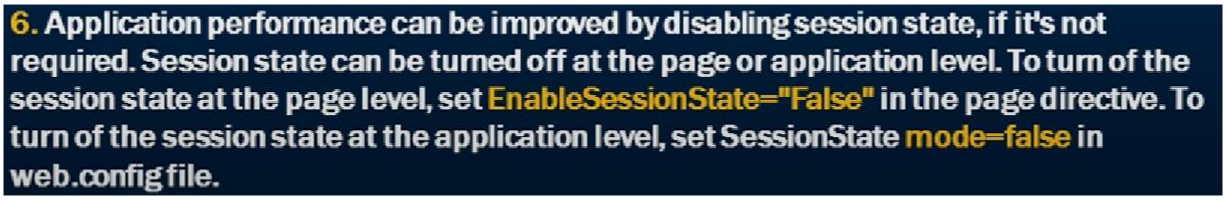
•Off mode, which disables session state.


3. The life time of a session is determined by the time-out value in web.config file. The default is 20 minutes. The time-out value can be adjusted according, to your application requirements.

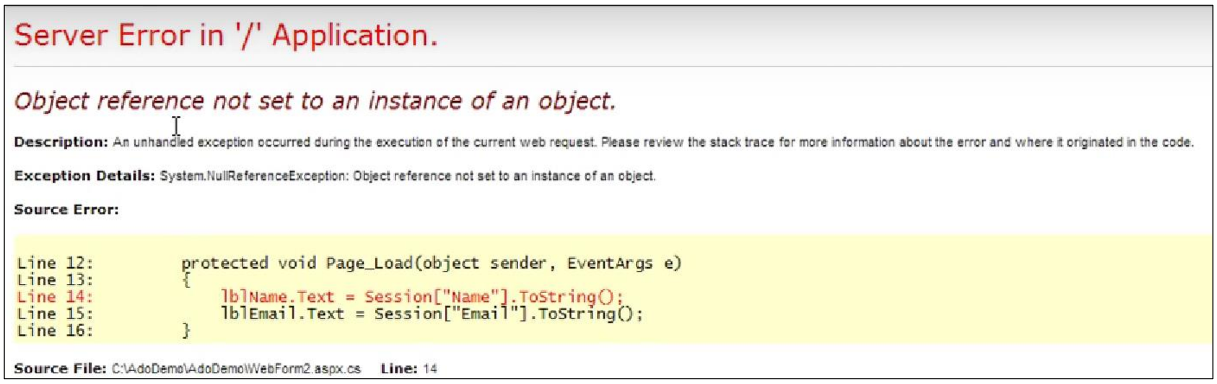
```
<sessionState mode="InProc" timeout="20"></sessionState>
```


5. It is always a good practice to check, if a session state variable is null before calling any of its methods, such as ToString(). Otherwise, we may run into runtime Null Reference Exceptions.

```
if (Session["Name"] != null)
{
    lblName.Text = Session["Name"].ToString();
}
```


6. Application performance can be improved by disabling session state, if it's not required. Session state can be turned off at the page or application level. To turn of the session state at the page level, set EnableSessionState="False" in the page directive. To turn of the session state at the application level, set SessionState mode=false in web.config file.

Try to access after 20 mins


Server Error in '/' Application.

Object reference not set to an instance of an object.

Description: An unhandled exception occurred during the execution of the current web request. Please review the stack trace for more information about the error and where it originated in the code.

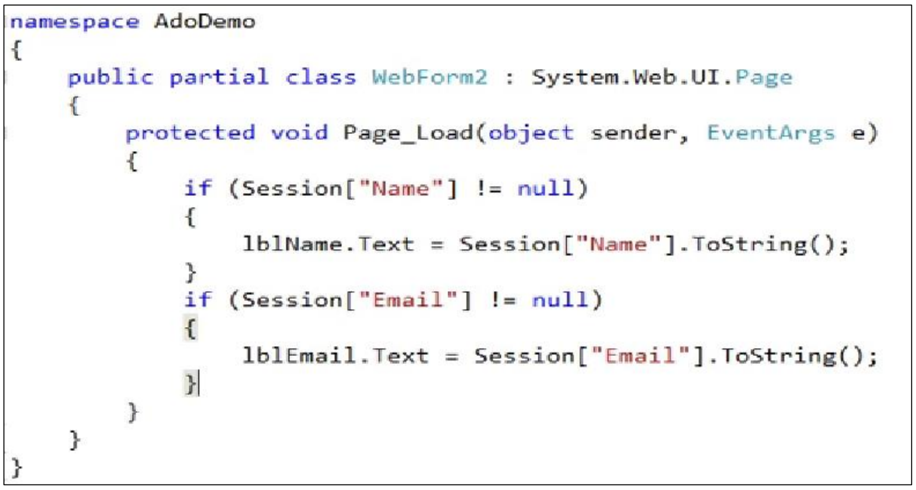
Exception Details: System.NullReferenceException: Object reference not set to an instance of an object.

Source Error:

```
Line 12:         protected void Page_Load(object sender, EventArgs e)
Line 13:         {
Line 14:             lblName.Text = Session["Name"].ToString();
Line 15:             lblEmail.Text = Session["Email"].ToString();
Line 16:         }
```

Source File: C:\AdoDemo\AdoDemo\WebForm2.aspx.cs   **Line:** 14

Check whether session variables are null or not

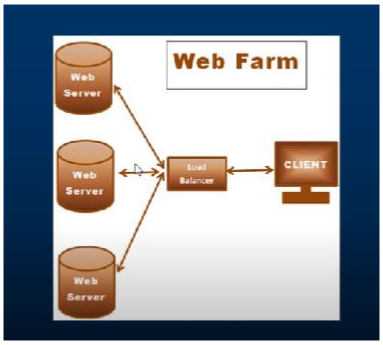


```
namespace AdoDemo
{
    public partial class WebForm2 : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
            if (Session["Name"] != null)
            {
                lblName.Text = Session["Name"].ToString();
            }
            if (Session["Email"] != null)
            {
                lblEmail.Text = Session["Email"].ToString();
            }
        }
    }
}
```

Application State:

- Application State is a state management technique.
- Application State is stored in the memory of the the server and is faster than storing and retrieving information in a database.
- Session sate is specific for a single user session, but Application State is for all users and sessions. Application State does not have a default expiration period.
- When we close the worker process the application object will be lost.
- Technically the data is shared amongst users by a HTTPApplcationState class and the data can be stored here in a key/value pair. It can also be accessed using the application property of the HTTPContext class.

- Application State variables are available across all pages and across all sessions.
- Application state variables are stored on the web server.
- Application state variables are cleared only when the process hosting the application is restarted that is when the application ends.
- Application Variables are not shared across a web farm.

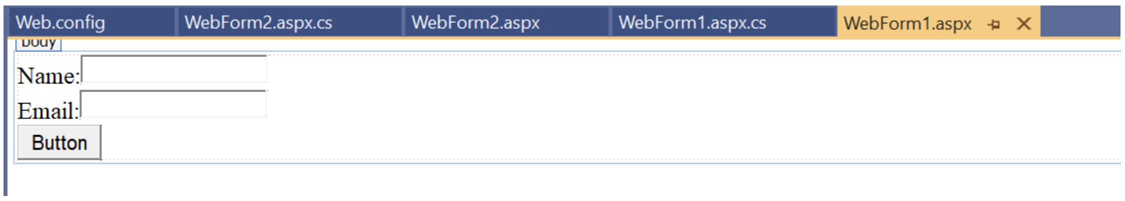


- Application state variables are not thread safe. Lock and unlock method of the application class must be used to protect against race condition, deadlocks.

```
Application.Lock();
Application[“GlobalVariable”]=(int)Application[“GlobalVariable”]+1;
Application.Unlock();
```

Use Application state Variable only, When the variables need to have global access and when you need them for entire time, during the life time of an application.

Example:

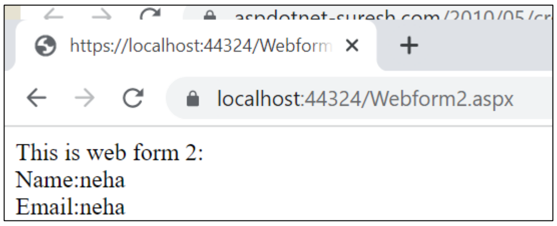


Page1.aspx.cs

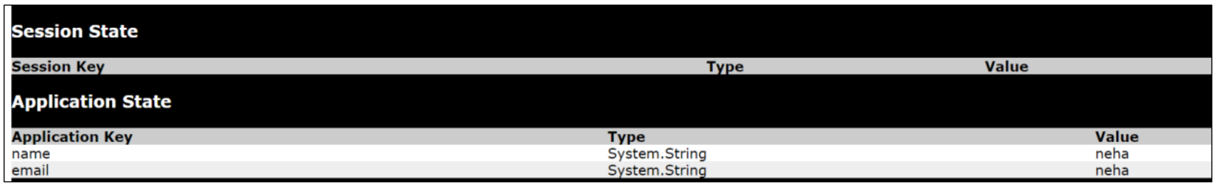
```
protected void Button1_Click(object sender, EventArgs e)
        {
            Application["name"] = TextBox1.Text;
            Application["email"] = TextBox2.Text;
            Response.Redirect("~/Webform2.aspx");
        }
```

```
Page2.aspx

<form id="form1" runat="server">
        <div>
            This is web form 2:<br />
            Name:<asp:Label ID="Label1" runat="server" Text="Label"></asp:Label><br />
            Email:<asp:Label ID="Label2" runat="server" Text="Label"></asp:Label>
        </div>
    </form>
```

```
Page2.aspx.cs

protected void Page_Load(object sender, EventArgs e)
        {
            Label1.Text = Application["name"].ToString();
            Label2.Text = Application["email"].ToString();
        }
```


https://localhost:44324/Webform
localhost:44324/Webform2.aspx

This is web form 2:
Name:neha
Email:neha

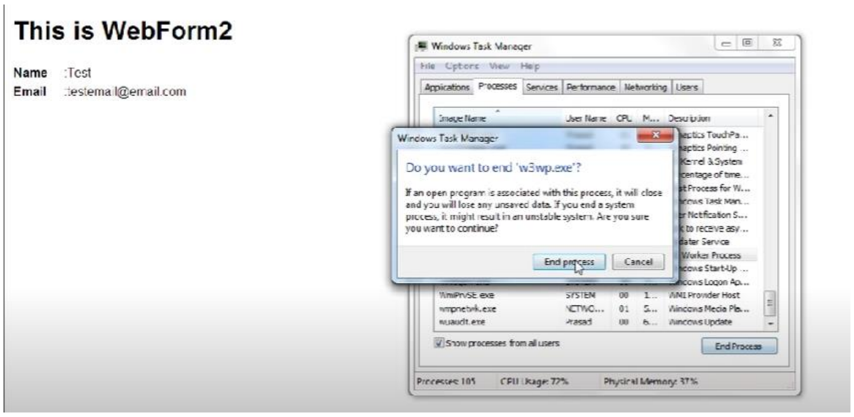


Session State

Session Key	Type	Value

Application State

Application Key	Type	Value
name	System.String	neha
email	System.String	neha



This is WebForm2

Name :Test
Email :testemail@email.com

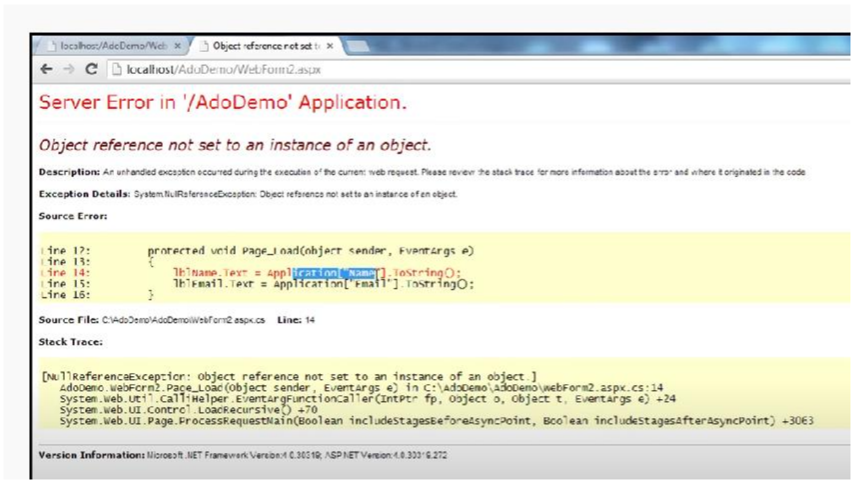
Windows Task Manager
File Options View Help
Applications Processes Services Performance Networking Users
Image Name User Name CPU M... Description
Show processes from all users
End Process
Processes: 105 CPU Usage: 72% Physical Memory: 37%

Windows Task Manager

Do you want to end 'w3wp.exe'?

If an open program is associated with this process, it will close and you will lose any unsaved data. If you end a system process, it might result in an unstable system. Are you sure you want to continue?

End process Cancel



Server Error in '/AdoDemo' Application.

Object reference not set to an instance of an object.

Description: An unhandled exception occurred during the execution of the current web request. Please review the stack trace for more information about the error and where it originated in the code

Exception Details: System.NullReferenceException: Object reference not set to an instance of an object.

Source Error:

```
Line 12:         protected void Page_Load(object sender, EventArgs e)
Line 13:         {
Line 14:             lblName.Text = Application["Name"].ToString();
Line 15:             lblEmail.Text = Application["Email"].ToString();
Line 16:         }
```

Source File: C:\AdoDemo\AdoDemo\WebForm2.aspx.cs **Line:** 14

Stack Trace:

```
[NullReferenceException: Object reference not set to an instance of an object.]
   AdoDemo.WebForm2.Page_Load(Object sender, EventArgs e) in C:\AdoDemo\AdoDemo\WebForm2.aspx.cs:14
   System.Web.Util.CalliHelper.EventArgFunctionCaller(IntPtr fp, Object o, Object t, EventArgs e) +24
   System.Web.UI.Control.LoadRecursive() +70
   System.Web.UI.Page.ProcessRequestMain(Boolean includeStagesBeforeAsyncPoint, Boolean includeStagesAfterAsyncPoint) +3063
```

Version Information: Microsoft .NET Framework Version:4.0.30319; ASP.NET Version:4.0.30319.272

Check whether Application variables are null or not

```
protected void Page_Load(object sender, EventArgs e)
        {
            if (Application["name"] != null)
            {
                Label1.Text = Application["name"].ToString();
            }
            if (Application["email"] != null)
            {
                Label2.Text = Application["email"].ToString();
            }
        }
```