

2023/2024(1)

EF234301 Web Programming

Lecture #11b

ASP.NET: State Management

Misbakhul Munir **IRFAN SUBAKTI**

司馬伊凡

Мисбакхул Мунир **Ирфан Субакти**

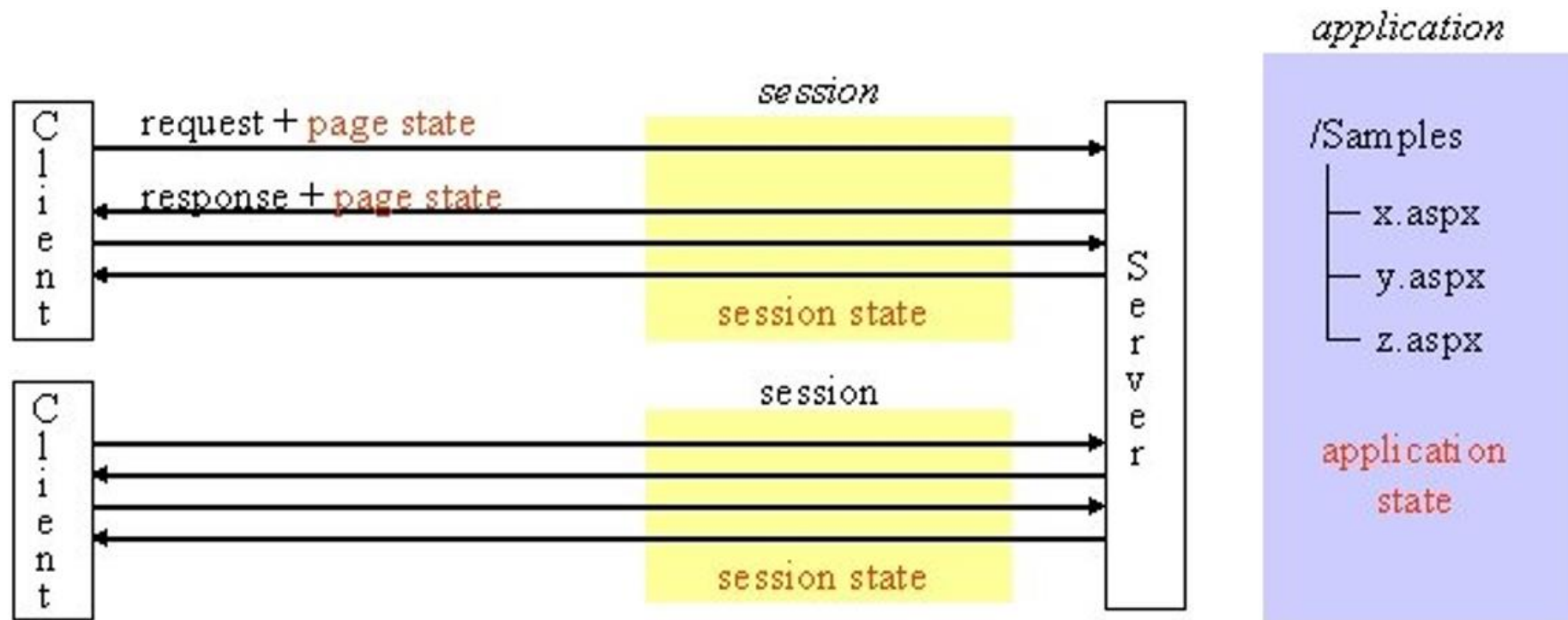
Page & control's instance: Gone when out

- Web programming traditional
 - Every time a Web page posted to the server, an instance will be created
 - It means all information about page & control on its page will be gone when we navigate out from a page to other page
 - E.g., a user filling some text on the TextBox. When she/he navigate out to other page, this text will be gone

State Management: Categories

- Page State/ViewState
 - Save a page's information, e.g., the content of TextBox, CheckBox, etc.
- Session State
 - Save a session user's information for a particular time, e.g., a cart's information from an online shopping, a user email's information, etc.
- Application State
 - Save an application's information, i.e., all files whose *.aspx extension located in the same folder as a part of virtual directory

ViewState-Session-Application State: Diagram



ViewState-Session-App. State: Differences

- ViewState
 - A web form's ViewState is available only within that web form
 - Stored in a hidden field called `_ViewState`. Because of this, ViewState will be lost, if we navigate away from the page or if the browser is closed
 - Used by *all* asp.net controls to retain their state across `postback`
- Session State
 - Available across *all* pages, but only for a given *single* session. It's like *single-user* global data.
 - Stored on the web server
 - Cleared when the user session *times out*. The default is 20 minutes. This is configurable in `web.config`
- Application State
 - Available across *all* pages and across *all* sessions. It's like *multi-user* global data
 - Stored on the web server
 - Cleared when the process hosting the application is restarted

ViewState-Session-App. State: Accessing

- **ViewState**

- **Write:** `ViewState["counter"] = counterVal;`
- **Read:** `int counterVal = (int) ViewState["counter"];`

- **Session State**

- **Write:** `Session["cart"] = shoppingCart;`
- **Read:** `DataTable shoppingCart = (DataTable) Session["cart"];`

- **Application State**

- **Write:** `Application["database"] = databaseName;`
- **Read:** `string databaseName = (string) Application["databaseName"];`

01HttpRequest.aspx

- Create a New Project: **ASP.NET Web Forms Site**

- Give a name to this new project, e.g., **MyState**

- 01HttpRequest.aspx becomes our first **Web Form** file

```
01HttpRequest.aspx 01HttpRequest.aspx
1  <%@ Page Language="C#" AutoEventWireup="true" CodeFile="01HttpRequest.aspx.cs" Inherits="_01HttpRequest" %>
2
3  <!DOCTYPE html>
4
5  <html xmlns="http://www.w3.org/1999/xhtml">
6  <head runat="server">
7      <title>HttpRequest: Get the info</title>
8  </head>
9  <body>
10     <form id="form1" runat="server">
11         <div>
12             <asp:Label ID="Label1" runat="server" Text="First name:"></asp:Label>
13             <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
14             <br />
15             <asp:Label ID="Label2" runat="server" Text="Last name:"></asp:Label>
16             <asp:TextBox ID="TextBox2" runat="server"></asp:TextBox>
17             <br />
18             <asp:CheckBox ID="CheckBox1" runat="server" Text="I'm agree" />
19             <br />
20             <asp:Button ID="btnSubmit" runat="server" OnClick="SubmitClick" Text="Submit" />
21             <br />
22             <asp:Label ID="lblExplanation" runat="server" Text="Label"></asp:Label>
23         </div>
24     </form>
25 </body>
26 </html>
```

100 % No issues found

body

First name:

Last name:

I'm agree

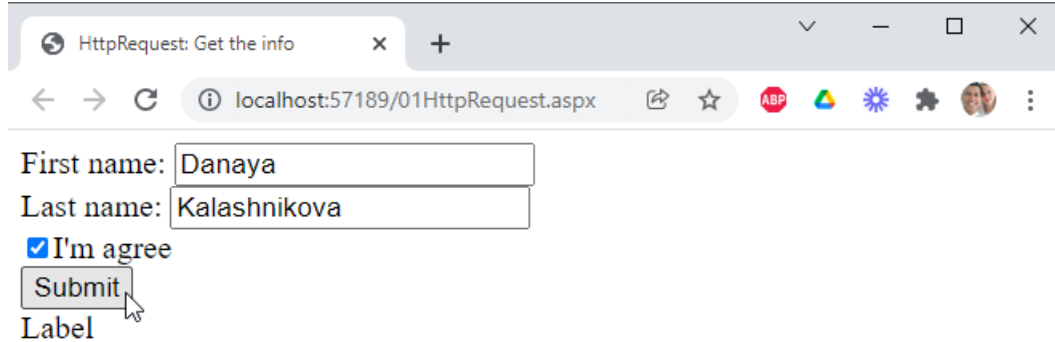
Label

01HttpRequest.aspx.cs

```
01HttpRequest.aspx.cs  X
2_01HttpRequest.aspx  01HttpRequest

1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Web;
5  using System.Web.UI;
6  using System.Web.UI.WebControls;
7
8  public partial class _01HttpRequest : System.Web.UI.Page {
9      protected void SubmitClick(object sender, EventArgs e) {
10         lblExplanation.Text = "Query string<br>";
11         foreach (string par in Request.QueryString.Keys) {
12             lblExplanation.Text += par + " = " +
13                 Request.QueryString[par] + "<br>";
14         }
15         lblExplanation.Text = "<br>Form parameters<br>";
16         foreach (string par in Request.Form.Keys) {
17             lblExplanation.Text += par + " = " +
18                 Request.Form[par] + "<br>";
19         }
20     }
21 }
```


01HttpRequest (Output)



HttpRequest: Get the info x +

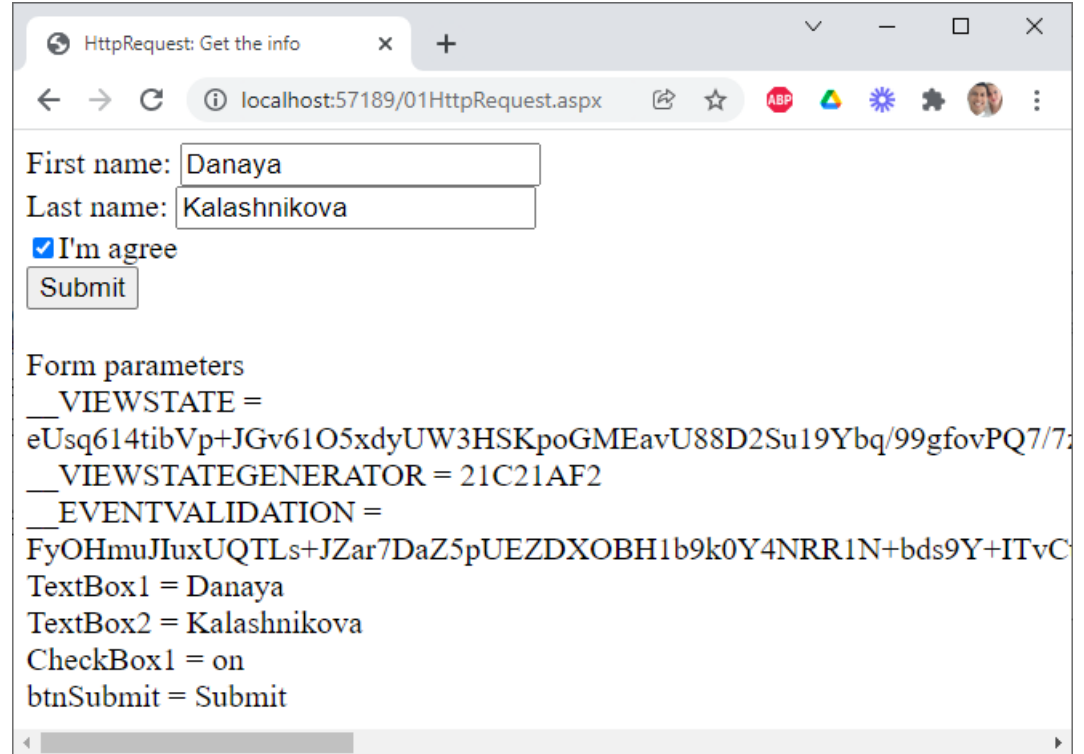
localhost:57189/01HttpRequest.aspx

First name:

Last name:

I'm agree

Label



HttpRequest: Get the info x +

localhost:57189/01HttpRequest.aspx

First name:

Last name:

I'm agree

Form parameters

```
__VIEWSTATE =  
eU5q614tibVp+JGv61O5xdyUW3HSKpoGMEavU88D2Su19Ybq/99gfovPQ7/7:  
__VIEWSTATEGENERATOR = 21C21AF2  
__EVENTVALIDATION =  
FyOHmuJluxUQTLs+JZar7DaZ5pUEZDXOBH1b9k0Y4NRR1N+bds9Y+ITvC  
TextBox1 = Danaya  
TextBox2 = Kalashnikova  
CheckBox1 = on  
btnSubmit = Submit
```

02HttpResponse.aspx

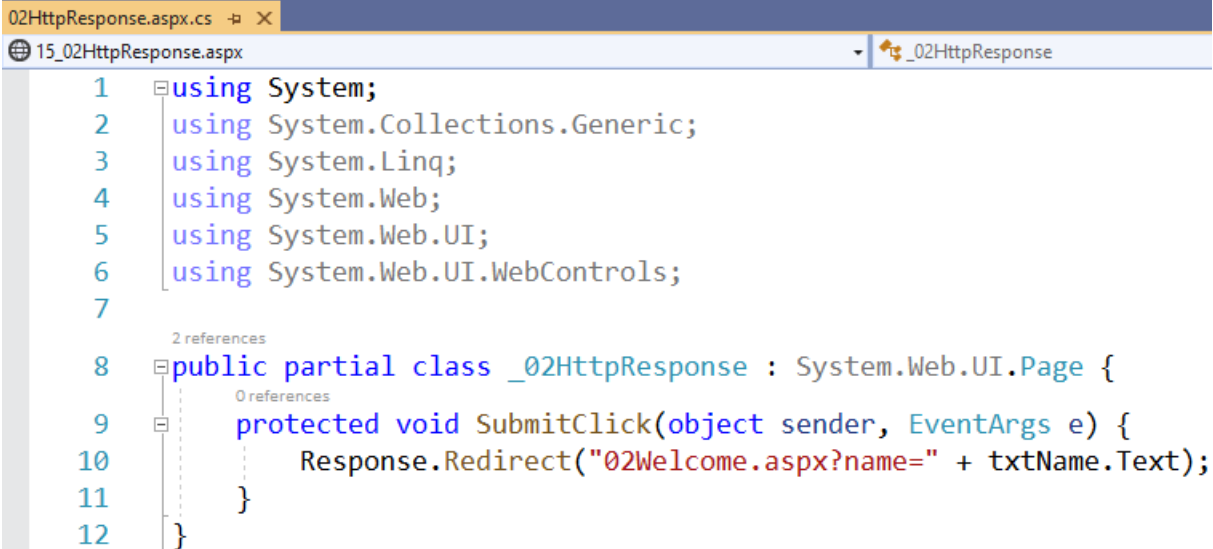
```
02HttpResponse.aspx
1 <%@ Page Language="C#" AutoEventWireup="true" CodeFile="02HttpResponse.aspx.cs" Inherits="_02HttpResponse" %>
2
3 <!DOCTYPE html>
4
5 <html xmlns="http://www.w3.org/1999/xhtml">
6 <head runat="server">
7     <title>HttpResponse: Redirecting</title>
8 </head>
9 <body>
10 <form id="form1" runat="server">
11 <div>
12     <asp:Label ID="Label1" runat="server" Text="Name:"></asp:Label>
13     <asp:TextBox ID="txtName" runat="server"></asp:TextBox>
14     <asp:Button ID="btnSubmit" runat="server" Text="Submit" OnClick="SubmitClick" />
15 </div>
16 </form>
17 </body>
18 </html>
```

100% No issues found

body

Name:

02HttpResponse.aspx.cs



```
02HttpResponse.aspx.cs - [X]
15_02HttpResponse.aspx
_02HttpResponse

1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Web;
5 using System.Web.UI;
6 using System.Web.UI.WebControls;
7
8 public partial class _02HttpResponse : System.Web.UI.Page {
9     protected void SubmitClick(object sender, EventArgs e) {
10         Response.Redirect("02Welcome.aspx?name=" + txtName.Text);
11     }
12 }
```

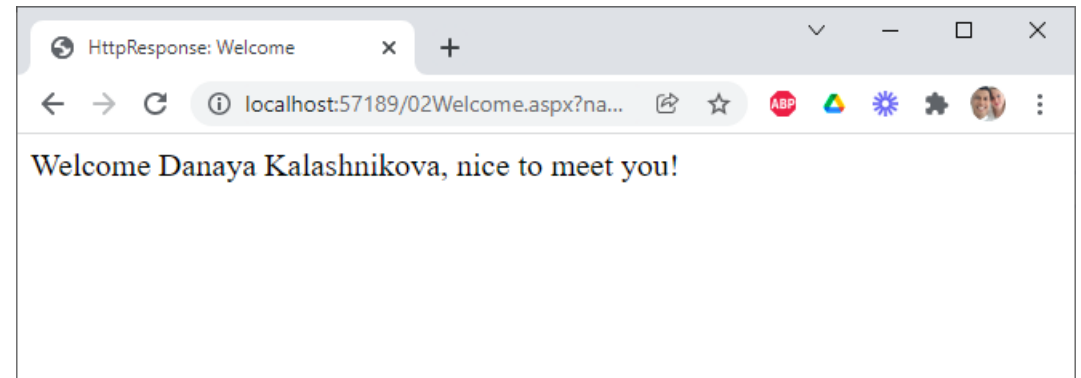
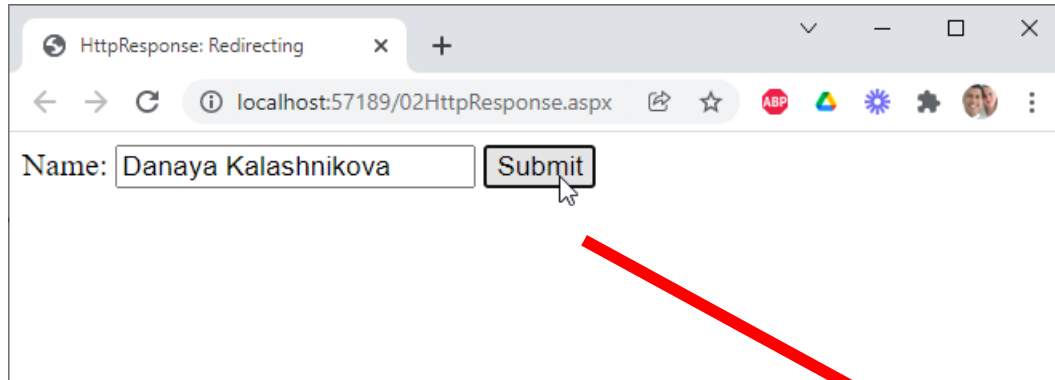
02Welcome.aspx

```
02Welcome.aspx -p x
1 <%@ Page Language="C#" AutoEventWireup="true"%>
2
3 <!DOCTYPE html>
4
5 <html xmlns="http://www.w3.org/1999/xhtml">
6 <head runat="server">
7     <title>HttpResponse: Welcome</title>
8 </head>
9 <body>
10     Welcome <%= Request.QueryString["name"] %>, nice to meet you!
11 </body>
12 </html>
```

100 % No issues found

body
Welcome , nice to meet you!

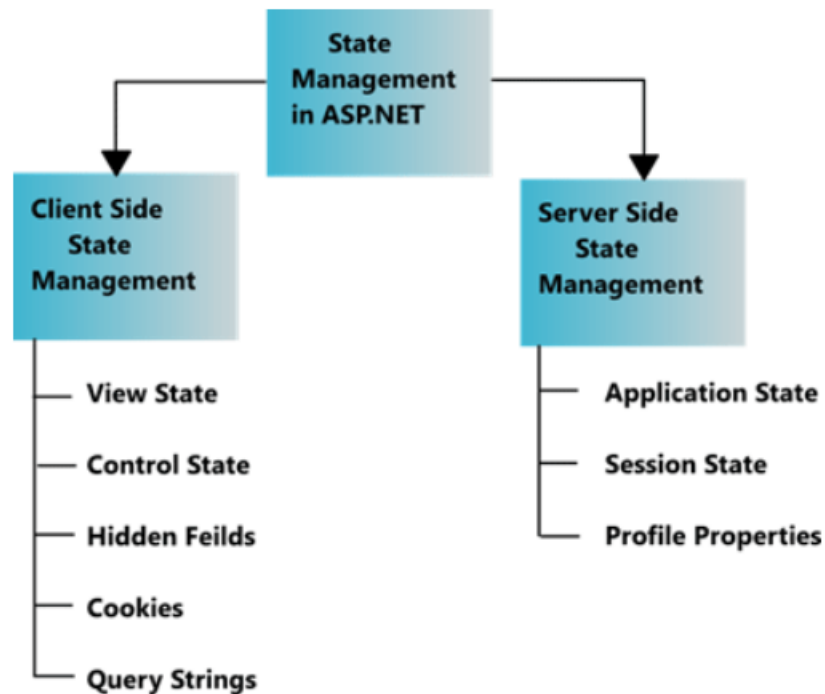
02HttpResponse (Output)



Overcoming limitation: Features

- To overcome this inherent limitation of traditional Web programming, ASP.NET includes several options that help us preserve data on both a per-page basis and an application-wide basis. These features are as follows:

- View state
- Control state
- Hidden fields
- Cookies
- Query strings
- *Application state*
- *Session state*
- *Profile Properties*



Overcoming limitation: Features (continued)

- View state, control state, hidden fields, cookies, and query strings all involve storing data on the client in various ways.
- However, *application state*, *session state*, and *profile properties* all store data in memory on the server.
- Each option has distinct advantages and disadvantages, depending on the scenario.

State Management: The option

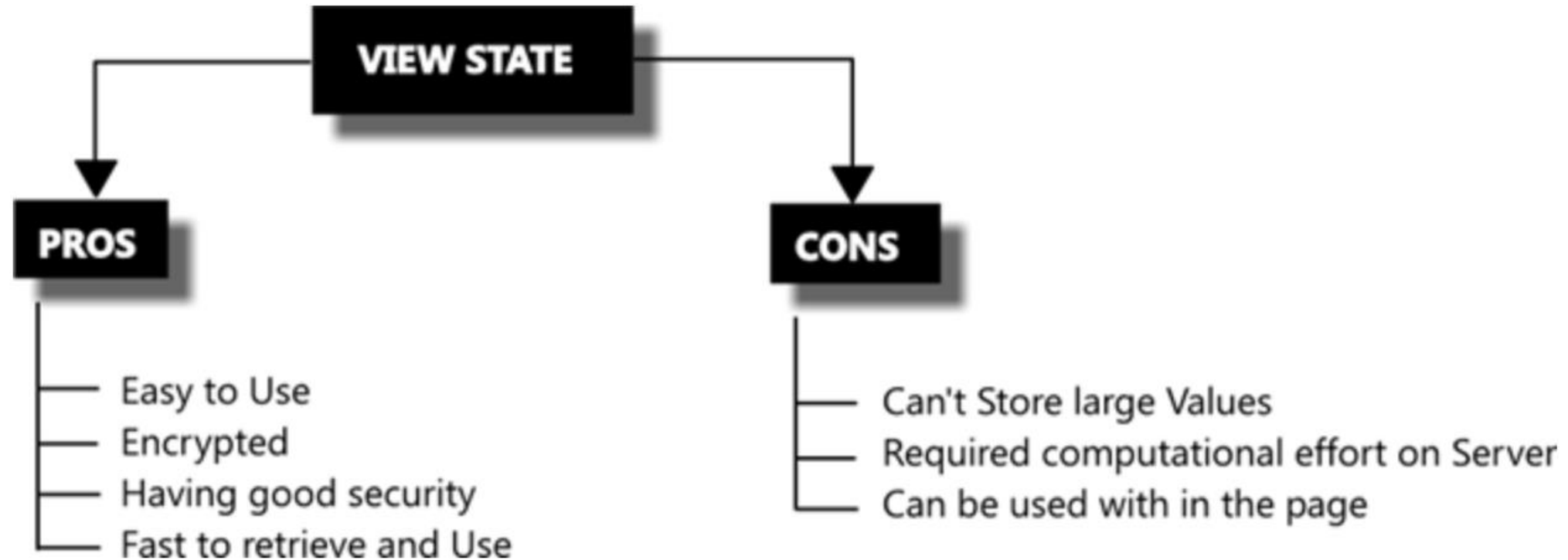
- Client-based state management
 - State management that involve storing information either in the page or on the client computer. For these options, no information is maintained on the server between round trips.
- Server-based state management
 - ASP.NET offers us a variety of ways to maintain state information on the server, rather than persisting information on the client. With server-based state management, we can decrease the amount of information sent to the client in order to preserve state, however it can use costly resources on the server. The following sections describe three server-based state management features: application state, session state, and profile properties.

Client-based state management: View State

- The `ViewState` property provides a dictionary object for retaining values between multiple requests for the same page. This is the default method that the page uses to *preserve* page and control property values between round trips.
- When the page is processed, the current state of the page and controls is hashed into a string and saved in the page as a *hidden field*, or multiple *hidden fields* if the amount of data stored in the `ViewState` property exceeds the specified value in the `MaxPageStateFieldLength` property. When the page is posted back to the server, the page parses the *view-state* string at page initialization and restores property information in the page.
- We can store values in *view state* as well.

```
ViewState["counter"] = counterVal;
```

View State: Pros & Cons (Nipun Tomar@c-sharpcorner.com)



ViewState: Disabling

- **Machine Level** – Disabling view state at machine level in `machine.config`, will disable `ViewState` of all the applications on the web server.

```
<Machine.config>  
  <system.web>  
    <pages enableViewState="false" />  
  </system.web>  
</Machine.config>
```

- **Application Level** – We can disable `ViewState` for all pages in `/web.config` file.

```
<configuration>  
  <system.web>  
    <pages enableViewState="false" />  
  </system.web>  
</configuration>
```

ViewState: Disabling (continued)

- **Page Level** - Disabling view state for a specific `aspx` file at the top.

```
<%@ Page Language="C#" .. EnableViewState="false" .. %>
```

- **Control Level** – We can disable `ViewState` for a specific control.

```
<asp:TextBox EnableViewState="false" ID="Name"  
  runat="server"></asp:TextBox>
```

03ViewState.aspx (Sachin Gargava@codeproject.com)

```
03ViewState.aspx
1 <%@ Page Language="C#" AutoEventWireup="true" CodeFile="03ViewState.aspx.cs" Inherits="ViewState" %>
2
3 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
4
5 <html xmlns="http://www.w3.org/1999/xhtml">
6 <head runat="server">
7     <title>View State</title>
8 </head>
9 <body>
10     <form id="form1" runat="server">
11         <div>
12             <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
13             <asp:Button ID="Button1" runat="server" Text="Button" OnClick="Button1_Click" /><br />
14             <br />
15             Value Saved in ViewState : &nbsp;  <asp:Label ID="Label1" runat="server"></asp:Label>
16         </div>
17     </form>
18 </body>
19 </html>
```

100% No issues found Ln: 20 Ch: 1

body

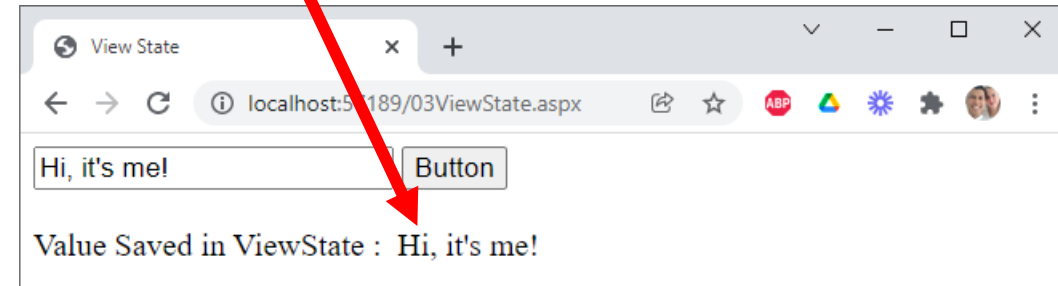
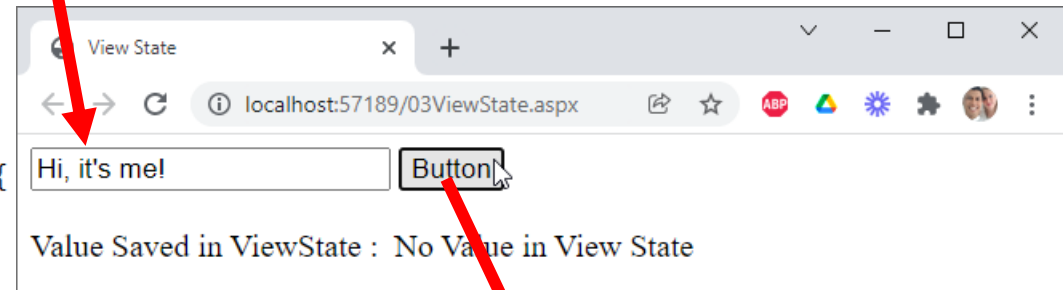
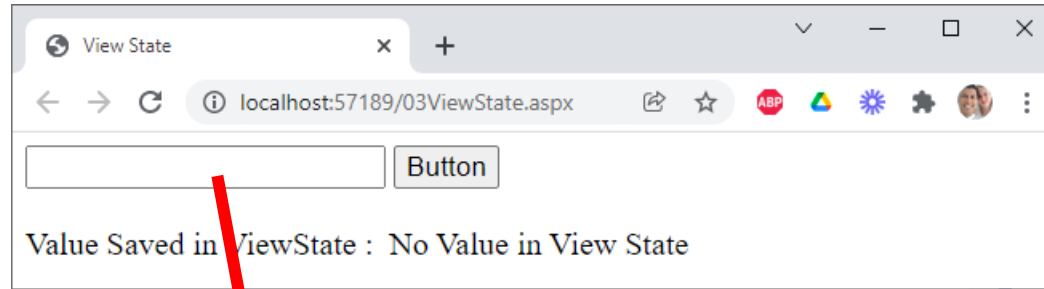
Button

Value Saved in ViewState : [Label1]

03ViewState.aspx.cs (Sachin Gargava@codeproject.com)

```
03ViewState.aspx.cs
44_03ViewState.aspx
ViewState

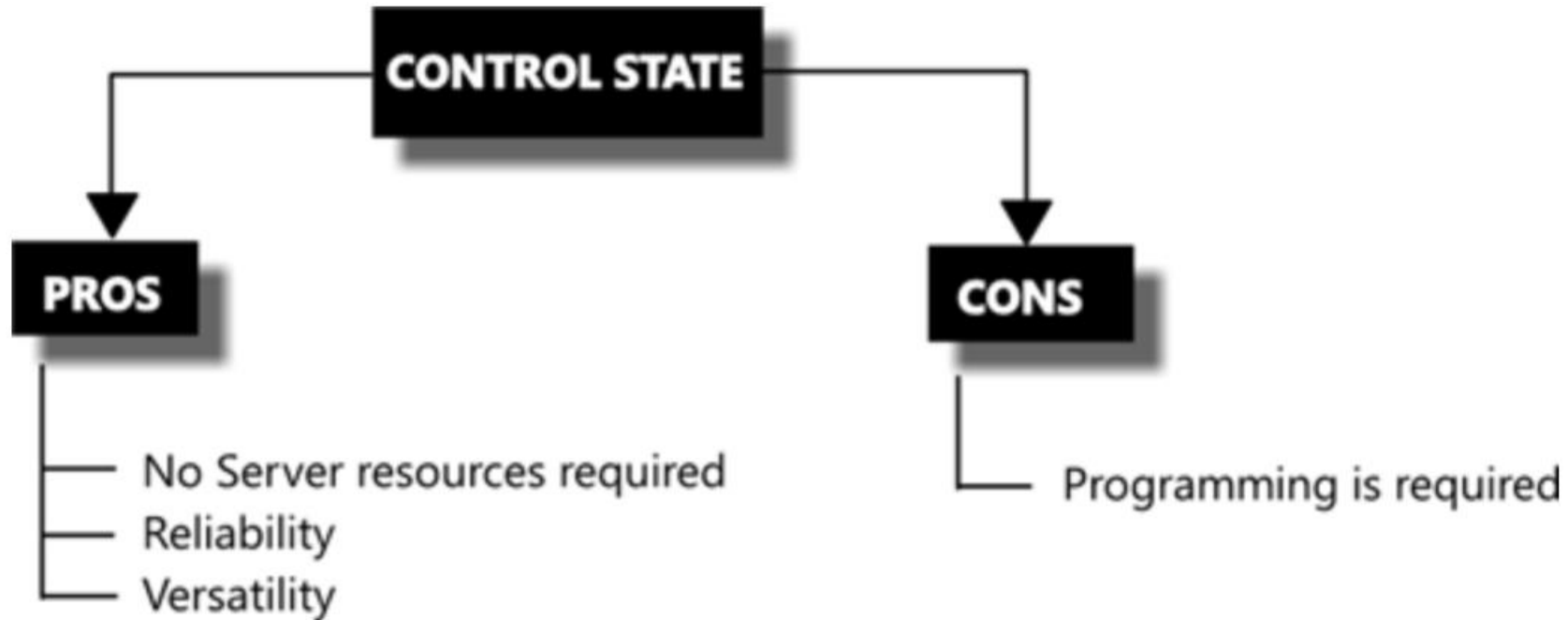
1 using System;
2 using System.Data;
3 using System.Configuration;
4 using System.Collections;
5 using System.Web;
6 using System.Web.Security;
7 using System.Web.UI;
8 using System.Web.UI.WebControls;
9 using System.Web.UI.WebControls.WebParts;
10 using System.Web.UI.HtmlControls;
11
12 public partial class ViewState : System.Web.UI.Page {
13     protected void Page_Load(object sender, EventArgs e) {
14         if (ViewState["Name"] != null) {
15             ViewState["Name"] = TextBox1.Text;
16             Label1.Text = ViewState["Name"].ToString();
17         } else {
18             ViewState["Name"] = "No Value in View State";
19             Label1.Text = ViewState["Name"].ToString();
20         }
21     }
22     protected void Button1_Click(object sender, EventArgs e) {
23
24     }
25 }
```



Control State

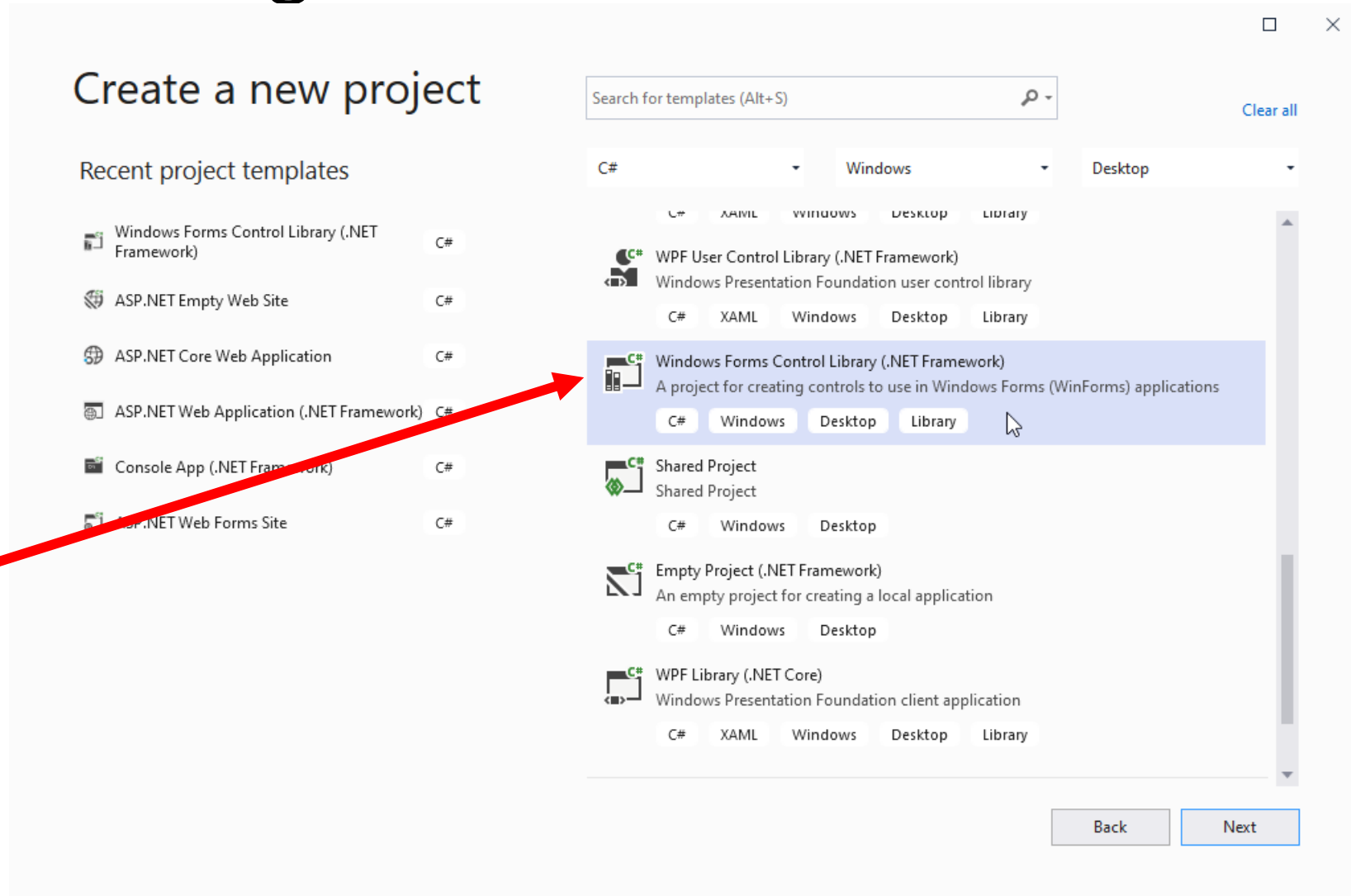
- Sometimes, we need to store control-state data in order for a control to work properly. For example, if we have written a custom control that has different tabs that show different information, in order for that control to work as expected, the control needs to know which tab is selected between round trips. The `ViewState` property can be used for this purpose, but view state can be turned off at a page level by developers, effectively breaking our control. To solve this, the ASP.NET page framework exposes a feature in ASP.NET called *control state*.
- The `ControlState` property allows us to persist property information that is specific to a control and cannot be turned off like the `ViewState` property.

Control State: Pros & Cons (Nipun Tomar@c-sharpcorner.com)



Control State: Creating Custom Control

- We are going to create our own **Custom Control**
 - Create a **New project** > **Windows Forms Control Library (.NET Framework)**



MyCustomControl: The project

- Choose a project's name for our new **Windows Forms Control Library (.NET Framework)**
 - E.g., **MyCustomControl**

Configure your new project

Windows Forms Control Library (.NET Framework) C# Windows Desktop Library

Project name
MyCustomControl

Location
D:\ITS\2021 ITS\08 Web Programming\NET\Lecture11

Solution name *i*
MyCustomControl

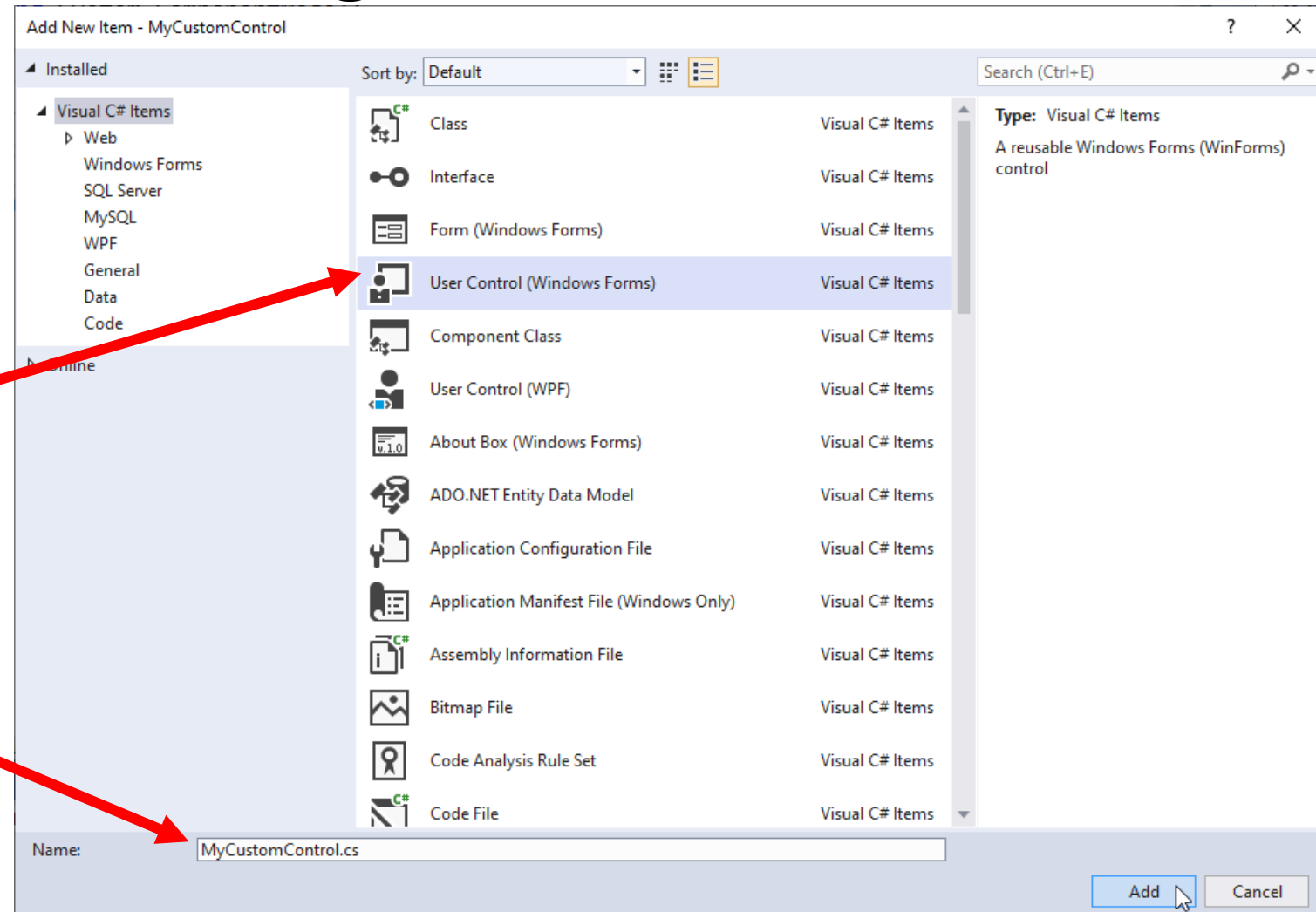
Place solution and project in the same directory

Framework
.NET Framework 4.7.2

Back Create

MyCustomControl: Adding User Control

- Right click on the **MyCustomControl** on Solution Explorer > **Add** > **New Item...** > **User Control (Windows Forms)**
- Choose a name
 - E.g., **MyCustomControl.cs**



MyCustomControl.cs (Rahul Rajat Singh@codeproject.com)



```
1 using System;
2 using System.Collections.Generic;
3 using System.ComponentModel;
4 using System.Text;
5 using System.Web;
6 using System.Web.UI;
7 using System.Web.UI.WebControls;
8
9 namespace MyCustomControl {
10     [DefaultProperty("Text")]
11     [ToolboxData("<{0}:WebCustomControl runat=server></{0}:WebCustomControl>")]
12     public class WebCustomControl : WebControl {
13         [Bindable(true)]
14         [Category("Appearance")]
15         [DefaultValue("")]
16         [Localizable(true)]
17         public string Text {
18             get {
19                 String s = (String)ViewState["Text"];
20                 return ((s == null) ? String.Empty : s);
21             }
22             set {
23                 ViewState["Text"] = value;
24             }
25         }
26     }
27 }
```

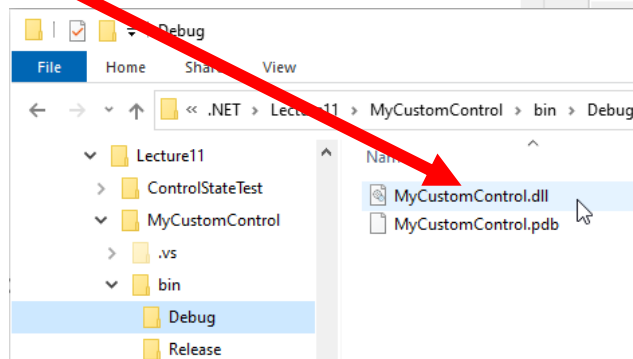
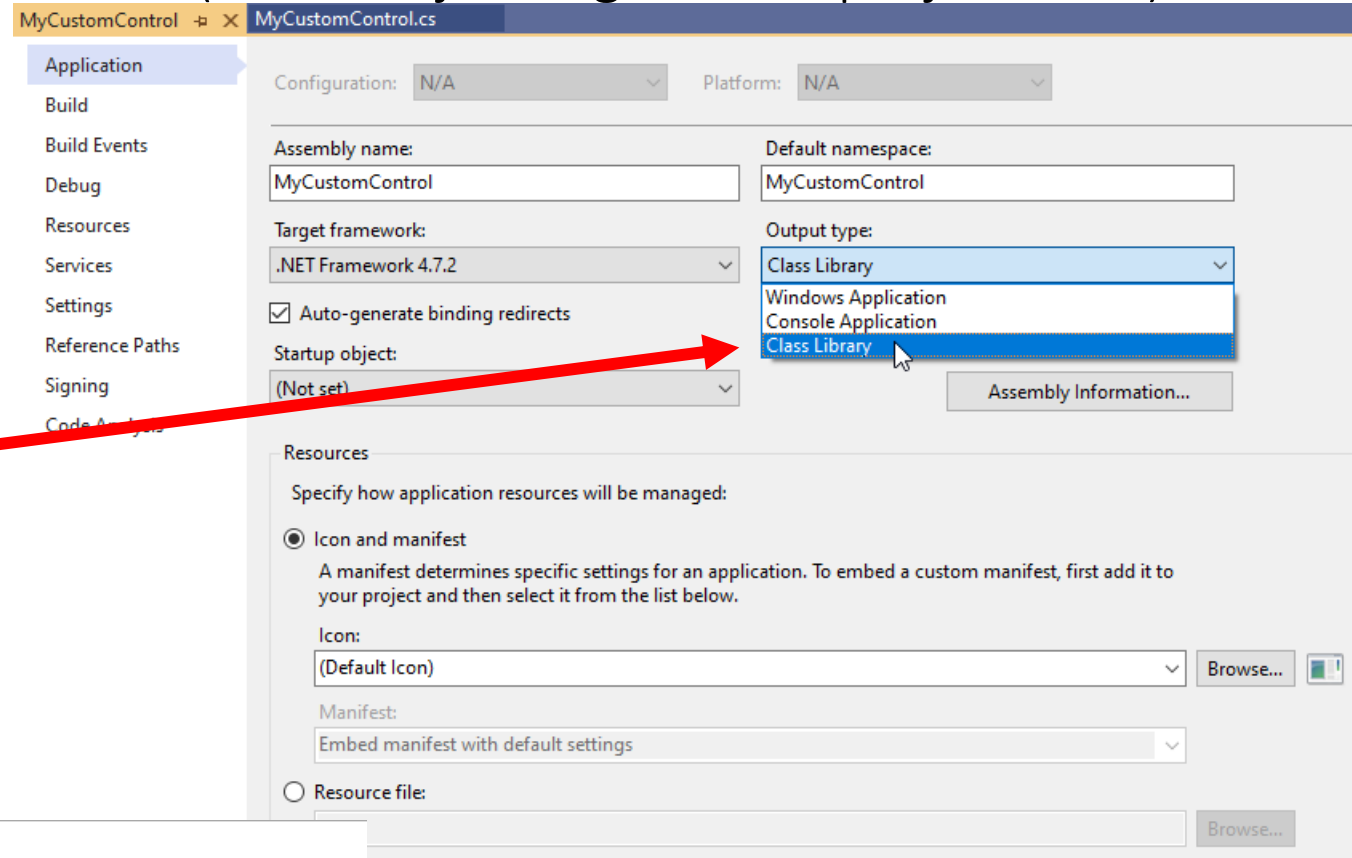
The image shows a screenshot of Visual Studio. The main editor window displays the code for MyCustomControl.cs. The code defines a namespace MyCustomControl containing a class WebCustomControl that inherits from WebControl. The class has a public string property Text with a getter and setter. The getter returns the value from ViewState, and the setter sets the value to ViewState. The class is decorated with several attributes: [DefaultProperty("Text")], [ToolboxData("<{0}:WebCustomControl runat=server></{0}:WebCustomControl>")], [Bindable(true)], [Category("Appearance")], [DefaultValue("")] and [Localizable(true)]. The Solution Explorer on the right shows the project structure, with MyCustomControl.cs selected. The Properties window at the bottom right shows the file properties for MyCustomControl.cs.

MyCustomControl.cs (Rahul Rajat Singh@codeproject.com)

```
MyCustomControl.cs -> X
MyCustomControl -> MyCustomControl.WebCustomControl -> Text2
26 [Bindable(true)]
27 [Category("Appearance")]
28 [DefaultValue("")]
29 [Localizable(true)]
30 public string Text2 {
31     get {
32         String s = (String)ViewState["Text2"];
33         return ((s == null) ? String.Empty : s);
34     }
35     set {
36         ViewState["Text2"] = value;
37     }
38 }
39 protected override void RenderContents(HtmlTextWriter output) {
40     output.Write(Text);
41     output.Write("<br/>");
42     output.Write(Text2);
43     output.Write("<br/>");
44 }
45 }
46 }
```

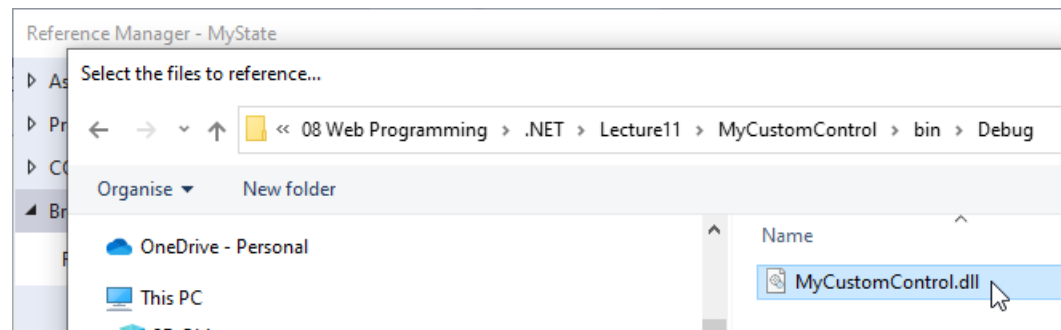
MyCustomControl.cs (Rahul Rajat Singh@codeproject.com)

- Right click on **MyCustomControl** on Solution Explorer > Properties
 - By default the output type: **Class Library**. So, let it be.
- Once we compile this project, we'll have `MyCustomControl.dll` in the folder **MyCustomControl > bin > Debug**

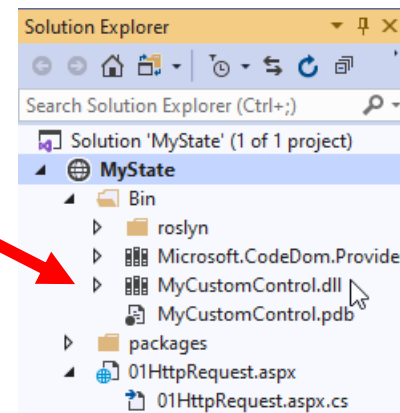


MyState: Add MyCustomControl as a reference

- Open MyState project
 - Right click on the **MyState** on Solution Explorer > **Add** > **Reference...** > **Browse...**



- Find and select MyCustomControl.dll



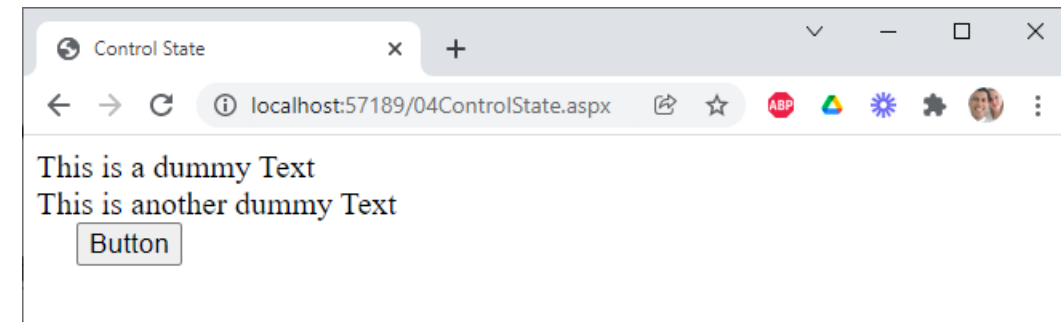
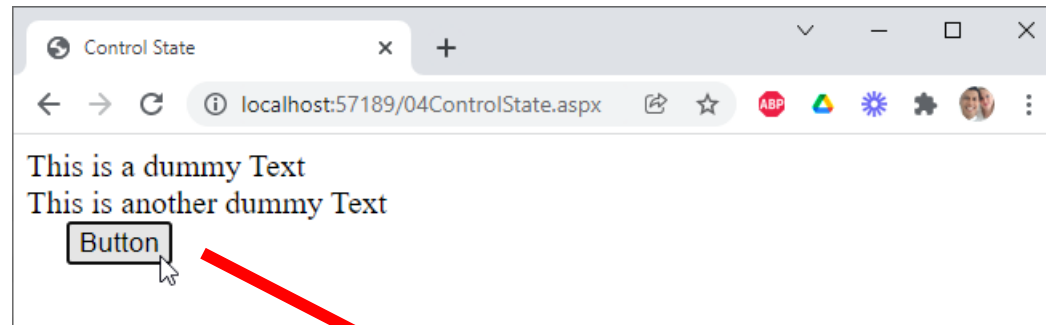
04ControlState.aspx.cs (Rahul Rajat Singh@codeproject.com)

```
04ControlState.aspx.cs
5_04ControlState.aspx
_04ControlState
Page_Load(object sender, EventArgs e)

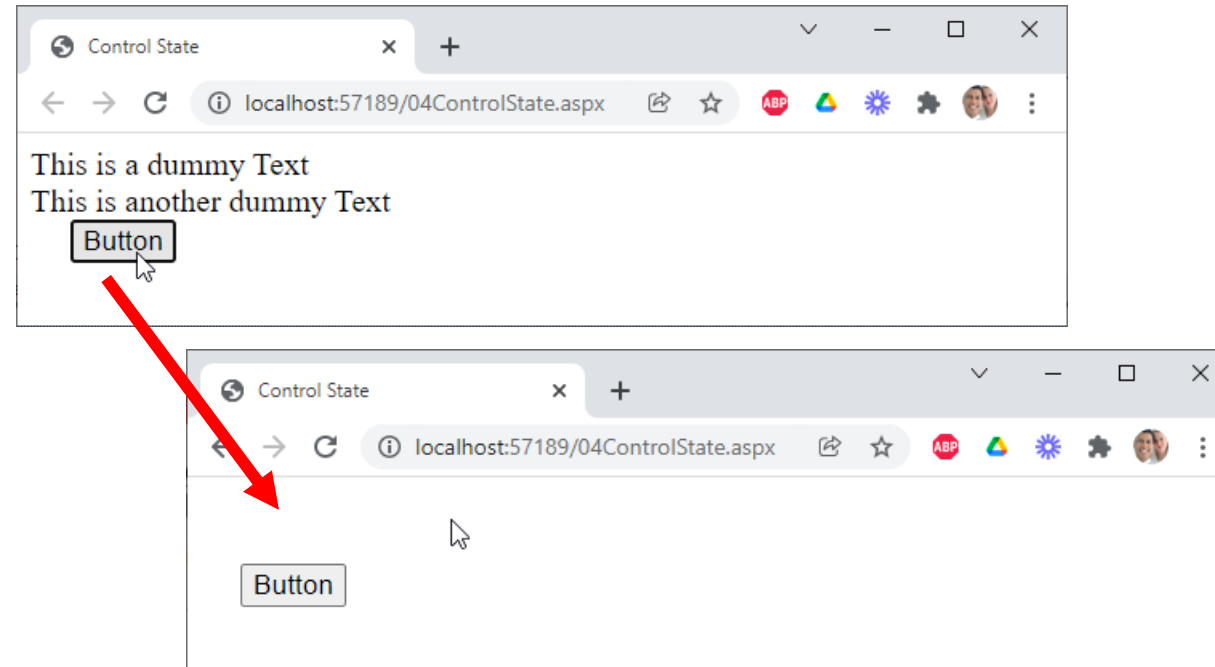
1 using System;
2 using System.Data;
3 using System.Configuration;
4 using System.Web;
5 using System.Web.Security;
6 using System.Web.UI;
7 using System.Web.UI.WebControls;
8 using System.Web.UI.WebControls.WebParts;
9 using System.Web.UI.HtmlControls;
10
11 public partial class _04ControlState : System.Web.UI.Page {
12     protected void Page_Load(object sender, EventArgs e) {
13         if (IsPostBack == false) {
14             WebCustomControl_1.Text = "This is a dummy Text";
15             WebCustomControl_1.Text2 = "This is another dummy Text";
16         }
17     }
18 }
```

04ControlState.aspx.cs (Rahul Rajat Singh@codeproject.com)

- Now for the first run, we will do the property setting and from next time onwards, we leave it up to the control to render these things.
- Now the page still has `EnableViewState` property set to `true`. So if we run the page and do `postback`, we will observe the expected behaviour. Let's look at what user will see:



04ControlState.aspx.cs (Rahul Rajat Singh@codeproject.com)



- The values are gone, the reason being our control depends on `ViewState` to function and we need to remove this dependency to `ControlState` so that even in this scenario, our control will continue to work.

MyCustomControl.cs (Rahul Rajat Singh@codeproject.com)

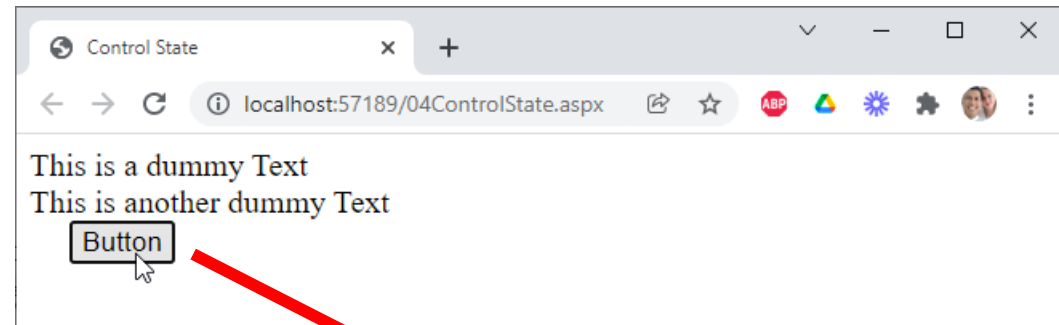
- What we need to do here is, we have to *override* the `OnInit` method and call the `RegisterRequiresControlState` method during *initialisation*. Then we have to *override* the `SaveControlState` and `LoadControlState` methods. So let us see how we do that for using `ControlState` for `Text` property of our control.

```
MyCustomControl.cs
MyCustomControl
MyCustomControl.WebCustomControl
Text2

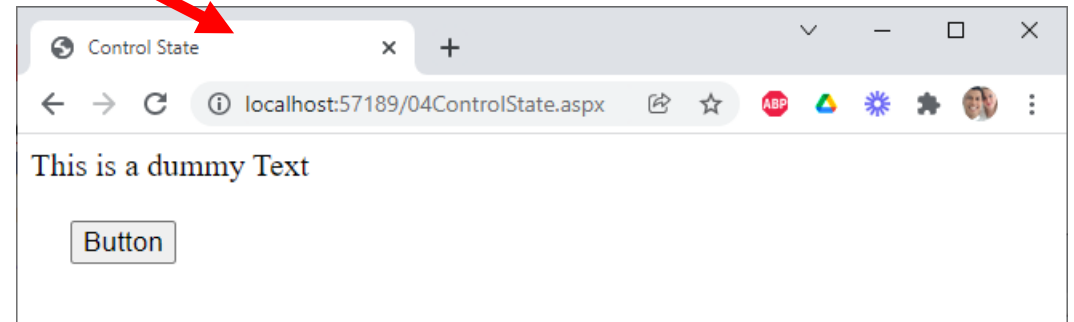
26 [Bindable(true)]
27 [Category("Appearance")]
28 [DefaultValue("")]
29 [Localizable(true)]
30 public string Text2 {
31     get {
32         String s = (String)ViewState["Text2"];
33         return ((s == null) ? String.Empty : s);
34     }
35     set {
36         ViewState["Text2"] = value;
37     }
38 }
39 protected override void RenderContents(HtmlTextWriter output) {
40     output.Write(Text);
41     output.Write("<br/>");
42     output.Write(Text2);
43     output.Write("<br/>");
44 }
45 // Overriding these 3 functions below, so that our control
46 // will continue to work even though ViewState is disabled
47 protected override void OnInit(EventArgs e) {
48     Page.RegisterRequiresControlState(this);
49     base.OnInit(e);
50 }
51 protected override void LoadControlState(object savedState) {
52     Text = (string)savedState;
53 }
54 protected override object SaveControlState() {
55     return Text;
56 }
57 }
58 }
```

04ControlState.aspx.cs (Rahul Rajat Singh@codeproject.com)

- Now we have the first property of our control using `ControlState` and the second one not using it. Now on `postback`, the output becomes:



- Look at the code to see the difference. The right way would be the way `ControlState` is being used for the first property, it should be used for all the controls inside a custom control.



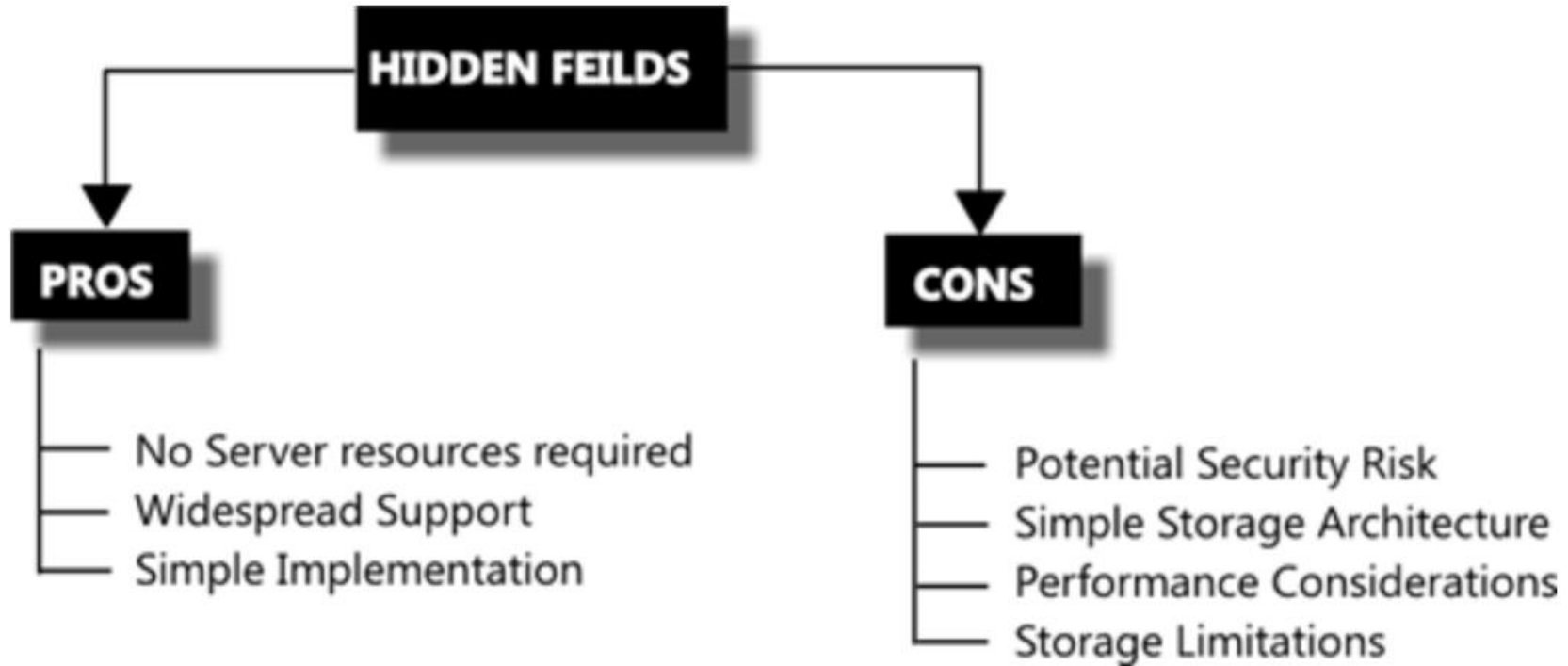
Hidden Fields

- ASP.NET allows us to store information in a `HiddenField` control, which renders as a standard HTML hidden field. A *hidden field* does not render visibly in the browser, but we can set its properties just as we can with a standard control. When a page is submitted to the server, the content of a hidden field is sent in the HTTP form collection along with the values of other controls. A hidden field acts as a repository for any page-specific information that we want to store directly in the page.
- A `HiddenField` control stores a single variable in its `Value` property and must be explicitly added to the page. The following example shows a `HiddenField` control with an initial value.

```
<asp:hiddenfield id="ExampleHiddenField"  
    value="Example Value"  
    runat="server"/>
```

- In order for hidden-field values to be available during page processing, we must submit the page using an HTTP POST command. If we use hidden fields and a page is processed in response to a link or an HTTP GET command, the hidden fields will not be available.

Hidden Fields: Pros & Cons (Nipun Tomar@c-sharpcorner.com)

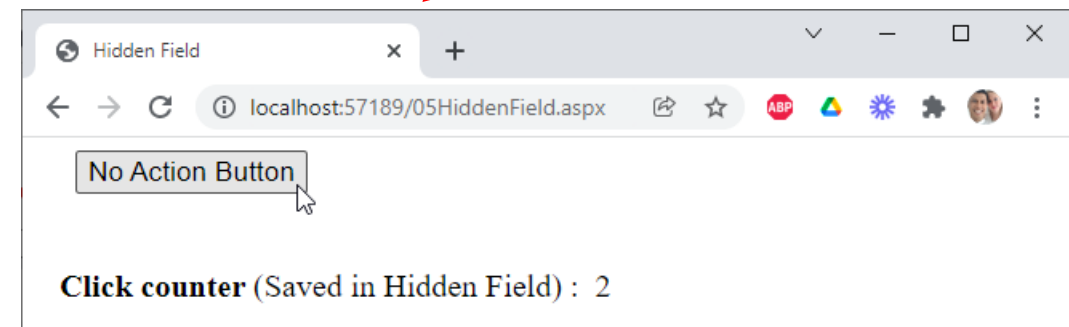
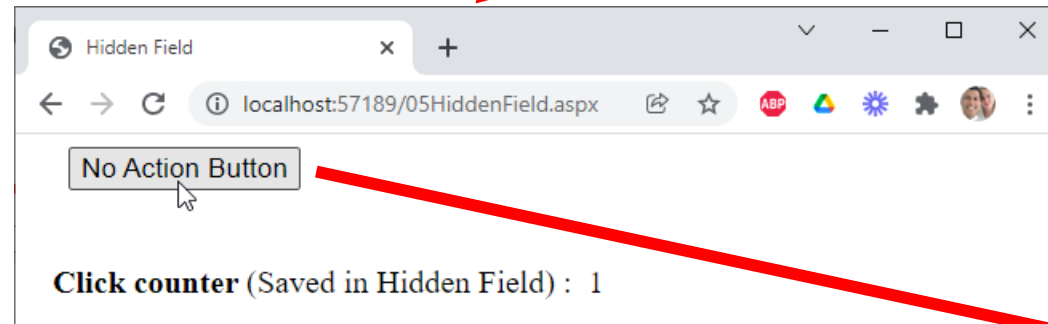
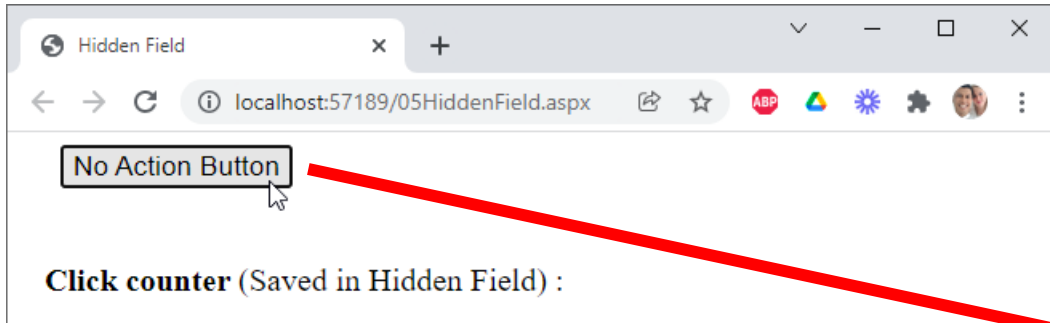


05HiddenField.aspx.cs (Sachin Gargava@codeproject.com)

```
05HiddenField.aspx.cs
14_05HiddenField.aspx
HiddenFile
Profile

1 using System;
2 using System.Data;
3 using System.Configuration;
4 using System.Collections;
5 using System.Web;
6 using System.Web.Security;
7 using System.Web.UI;
8 using System.Web.UI.WebControls;
9 using System.Web.UI.WebControls.WebParts;
10 using System.Web.UI.HtmlControls;
11
12 public partial class HiddenFile : System.Web.UI.Page {
13     protected void Page_Load(object sender, EventArgs e) {
14         if (IsPostBack) {
15             if (HiddenField1.Value != null) {
16                 int val = Convert.ToInt32(HiddenField1.Value) + 1;
17                 HiddenField1.Value = val.ToString();
18                 Label1.Text = val.ToString();
19             } else {
20             }
21         }
22     }
23
24     protected void Button1_Click(object sender, EventArgs e) {
25
26     }
27 }
```

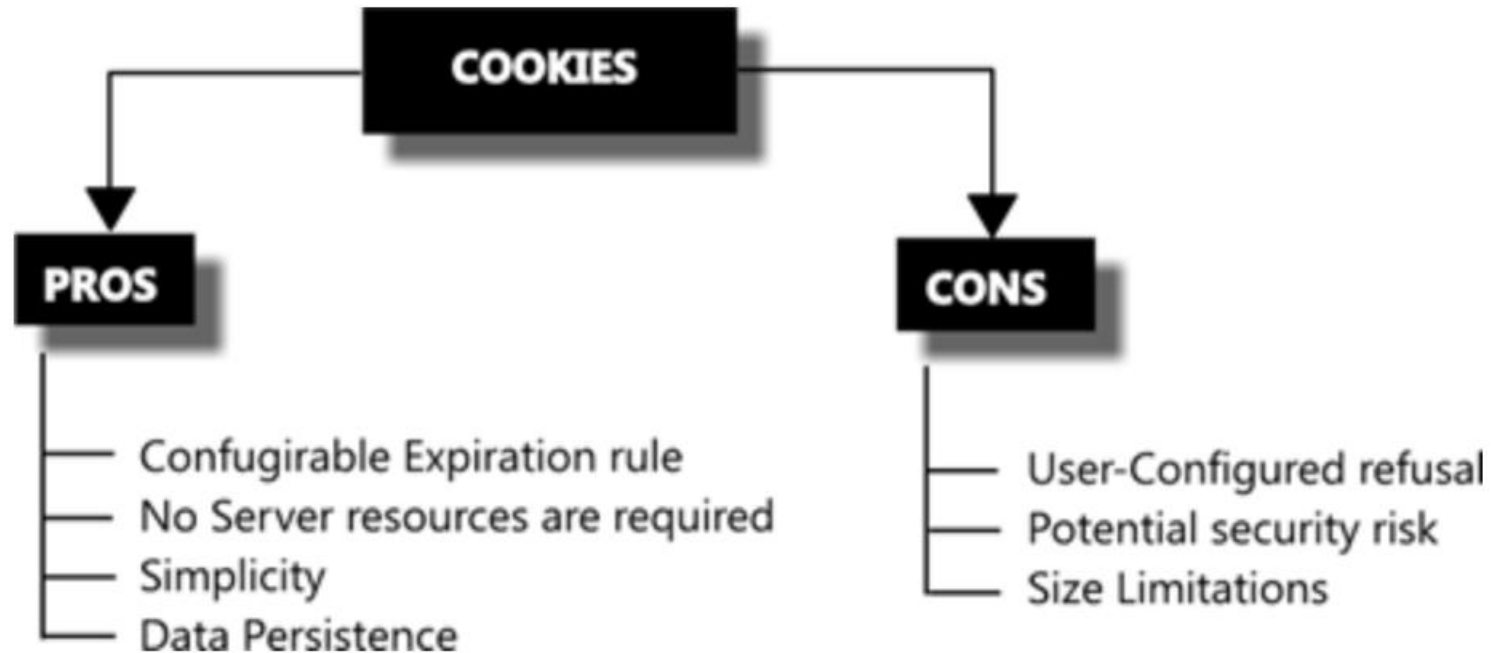
05HiddenField (Output)



Cookies

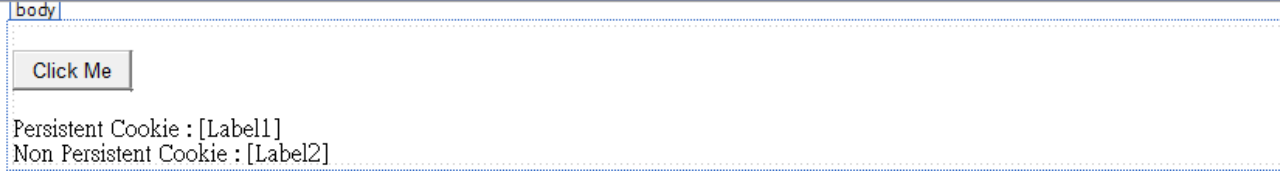
- A cookie is a small amount of data that is stored either in a text file on the client file system or in-memory in the client browser session. It contains site-specific information that the server sends to the client along with page output. Cookies can be temporary (with specific expiration times and dates) or persistent.
- We can use cookies to store information about a particular client, session, or application. The cookies are saved on the client device, and when the browser requests a page, the client sends the information in the cookie along with the request information. The server can read the cookie and extract its value. A typical use is to store a token (perhaps encrypted) indicating that the user has already been authenticated in our application.

Cookies: Pros & Cons (Nipun Tomar@c-sharpcorner.com)



06Cookies.aspx (Sachin Gargava@codeproject.com)

```
06Cookies.aspx
1 <%@ Page Language="C#" AutoEventWireup="true" CodeFile="06Cookies.aspx.cs"
2   Inherits="ViewState" %>
3
4 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
5   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
6 <html xmlns="http://www.w3.org/1999/xhtml">
7 <head runat="server">
8   <title>Cookies</title>
9 </head>
10 <body>
11   <form id="form1" runat="server">
12     <div>
13       <br />
14       <asp:Button ID="Button1" runat="server"
15         OnClick="Button1_Click" Text="Click Me" />
16       <br />
17       <br />
18       Persistent Cookie :
19       <asp:Label ID="Label1" runat="server"></asp:Label>
20       <br />
21       Non Persistent Cookie :
22       <asp:Label ID="Label2" runat="server"></asp:Label>
23     </div>
24   </form>
25 </body>
26 </html>
```



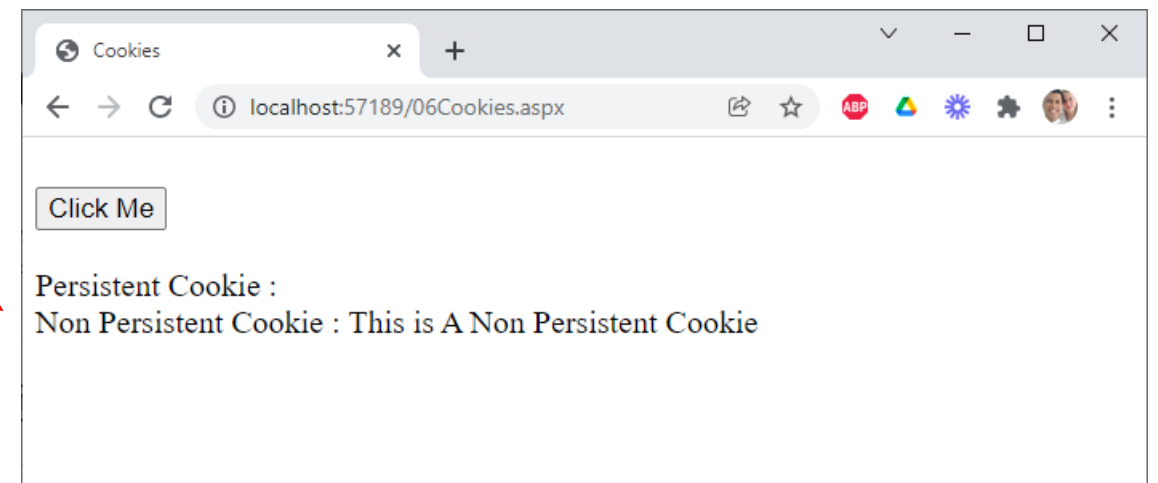
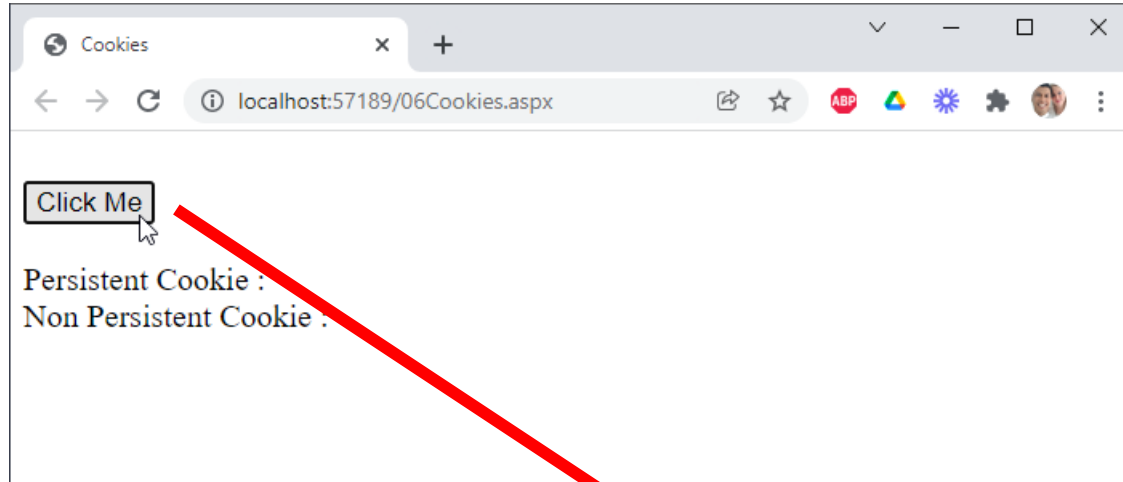
06Cookies.aspx.cs (Sachin Gargava@codeproject.com)

```
06Cookies.aspx.cs X
17_06Cookies.aspx ViewState Profile
1 using System;
2 using System.Data;
3 using System.Configuration;
4 using System.Collections;
5 using System.Web;
6 using System.Web.Security;
7 using System.Web.UI;
8 using System.Web.UI.WebControls;
9 using System.Web.UI.WebControls.WebParts;
10 using System.Web.UI.HtmlControls;
11 public partial class ViewState : System.Web.UI.Page {
12     protected void Page_Load(object sender, EventArgs e) {
13         if (!IsPostBack) {
14             // Approach 1. Creating a non persistent cookies
15             // Response.Cookies["nameWithNPCookies"].Value = "This is A Non Persistent Cookie";
16
17             // Approach 2. Creating a non persistent cookies
18             HttpCookie aCookieValNonPer = new HttpCookie("NonPersistent");
19             aCookieValNonPer.Value = "This is A Non Persistent Cookie";
20             Response.Cookies.Add(aCookieValNonPer);
21
22             // Approach 1. Creating a persistent cookies
23             //Response.Cookies["nameWithPCookies"].Value = "This is A Persistent Cookie";
24             //Response.Cookies["nameWithPCookies"].Expires = DateTime.Now.AddSeconds(10);
25
26             // Approach 2. Creating a persistent cookies
27             HttpCookie aCookieValPer = new HttpCookie("Persistent");
28             aCookieValPer.Value = "This is A Persistent Cookie";
29             //Response.Cookies["nameWithPCookies"].Expires = DateTime.Now.AddSeconds(5); //Check
30             aCookieValPer.Expires = DateTime.Now.AddSeconds(10);
31             Response.Cookies.Add(aCookieValPer);
32         }
33     }
}
```


06Cookies.aspx.CS (Sachin Gargava@codeproject.com)

```
34     protected void Button1_Click(object sender, EventArgs e) {
35         Label2.Text = ""; Label1.Text = "";
36         //if (Request.Cookies["nameWithNPCookies"] != null)
37         //    Label2.Text = Server.HtmlEncode(Request.Cookies["nameWithNPCookies"].Value);
38         //if (Request.Cookies["nameWithPCookies"] != null)
39         //    Label1.Text = Server.HtmlEncode(Request.Cookies["nameWithPCookies"].Value);
40         //if (Request.Cookies["NonPersistent"] != null)
41         //{
42         //    //HttpCookie aCookie = Request.Cookies["NonPersistent"];
43         //    //Label1.Text = Server.HtmlEncode(aCookie.Value);
44         //    Label2.Text = Request.Cookies["NonPersistent"].Value;
45         //}
46
47         if (Request.Cookies["NonPersistent"] != null) {
48             Label2.Text = Request.Cookies["NonPersistent"].Value;
49         }
50
51         if (Request.Cookies["Persistent"] != null) {
52             Label1.Text = Request.Cookies["Persistent"].Value;
53         }
54
55         //if (Request.Cookies["aCookieValNonPer"] != null)
56         //    Label2.Text = Request.Cookies.Get(aCookieValNonPer).ToString();
57         //    //Label2.Text = Server.HtmlEncode(Request.Cookies["aCookieValNonPer"].Value);
58         //if (Request.Cookies["aCookieValPer"] != null)
59         //    Label2.Text = Request.Cookies.Get(aCookieValPer).ToString();
60         //    //Label1.Text = Server.HtmlEncode(Request.Cookies["aCookieValPer"].Value);
61     }
62 }
```

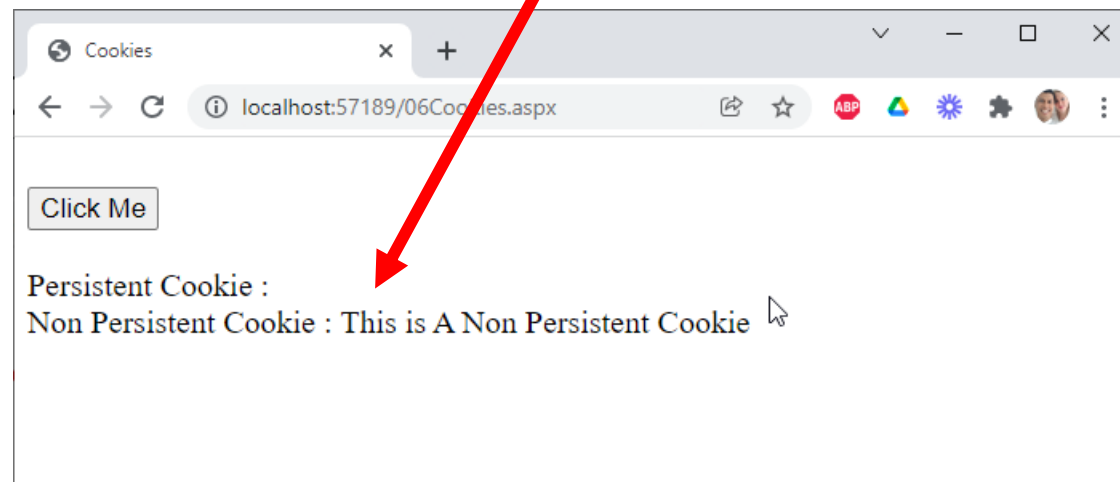
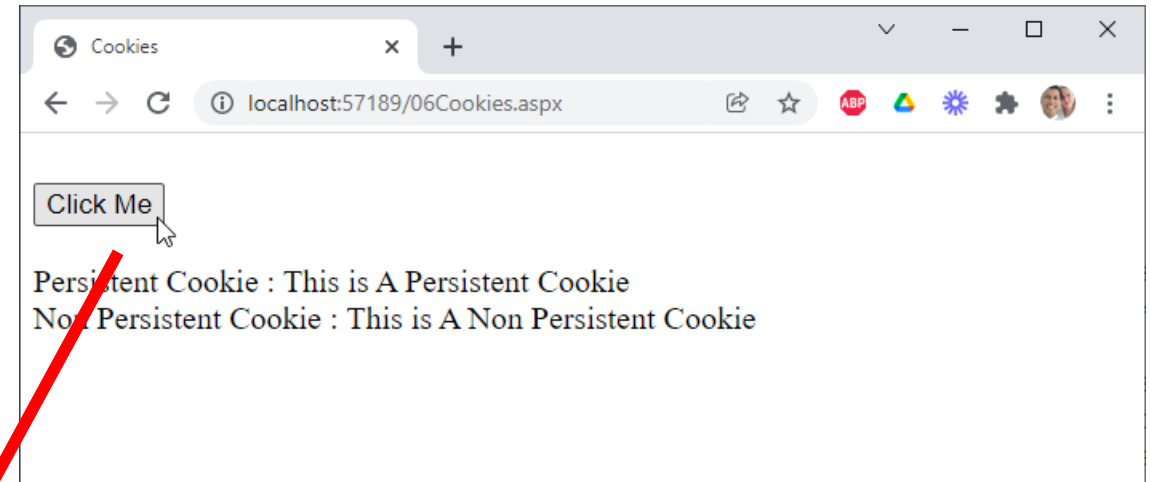
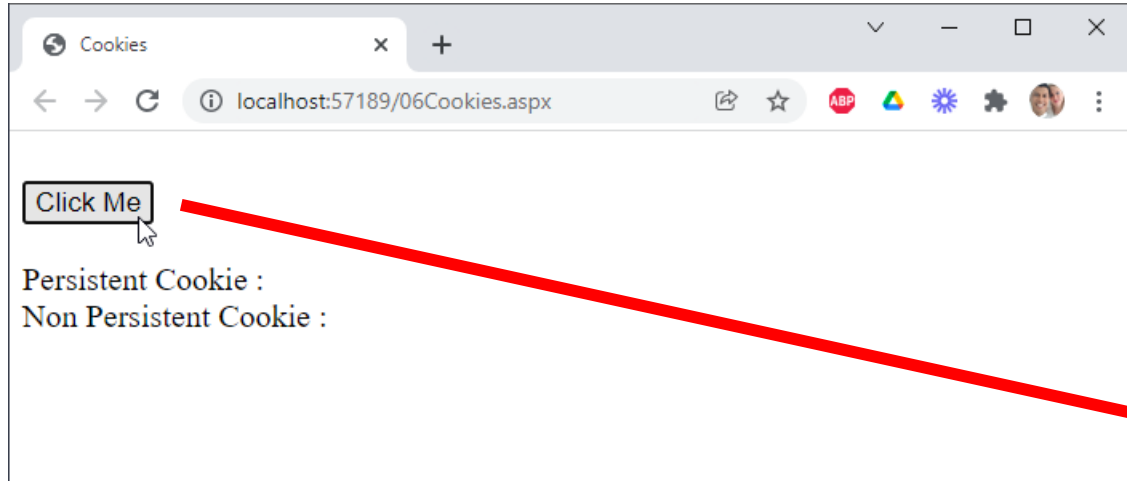
06Cookies (Output)



06Cookies.aspx.cs (Sachin Gargava@codeproject.com)

```
06Cookies.aspx.cs X
20_06Cookies.aspx ViewState Profile
1 using System;
2 using System.Data;
3 using System.Configuration;
4 using System.Collections;
5 using System.Web;
6 using System.Web.Security;
7 using System.Web.UI;
8 using System.Web.UI.WebControls;
9 using System.Web.UI.WebControls.WebParts;
10 using System.Web.UI.HtmlControls;
11 public partial class ViewState : System.Web.UI.Page {
12     protected void Page_Load(object sender, EventArgs e) {
13         if (!IsPostBack) {
14             // Approach 1. Creating a non persistent cookies
15             // Response.Cookies["nameWithNPCookies"].Value = "This is A Non Persistent Cookie";
16
17             // Approach 2. Creating a non persistent cookies
18             HttpCookie aCookieValNonPer = new HttpCookie("NonPersistent");
19             aCookieValNonPer.Value = "This is A Non Persistent Cookie";
20             Response.Cookies.Add(aCookieValNonPer);
21
22             // Approach 1. Creating a persistent cookies
23             //Response.Cookies["nameWithPCookies"].Value = "This is A Persistent Cookie";
24             //Response.Cookies["nameWithPCookies"].Expires = DateTime.Now.AddSeconds(10);
25
26             // Approach 2. Creating a persistent cookies
27             HttpCookie aCookieValPer = new HttpCookie("Persistent");
28             aCookieValPer.Value = "This is A Persistent Cookie";
29             Response.Cookies["nameWithPCookies"].Expires = DateTime.Now.AddSeconds(5); //Check
30             aCookieValPer.Expires = DateTime.Now.AddSeconds(10);
31             Response.Cookies.Add(aCookieValPer);
32         }
33     }
}
```

06Cookies (Output)



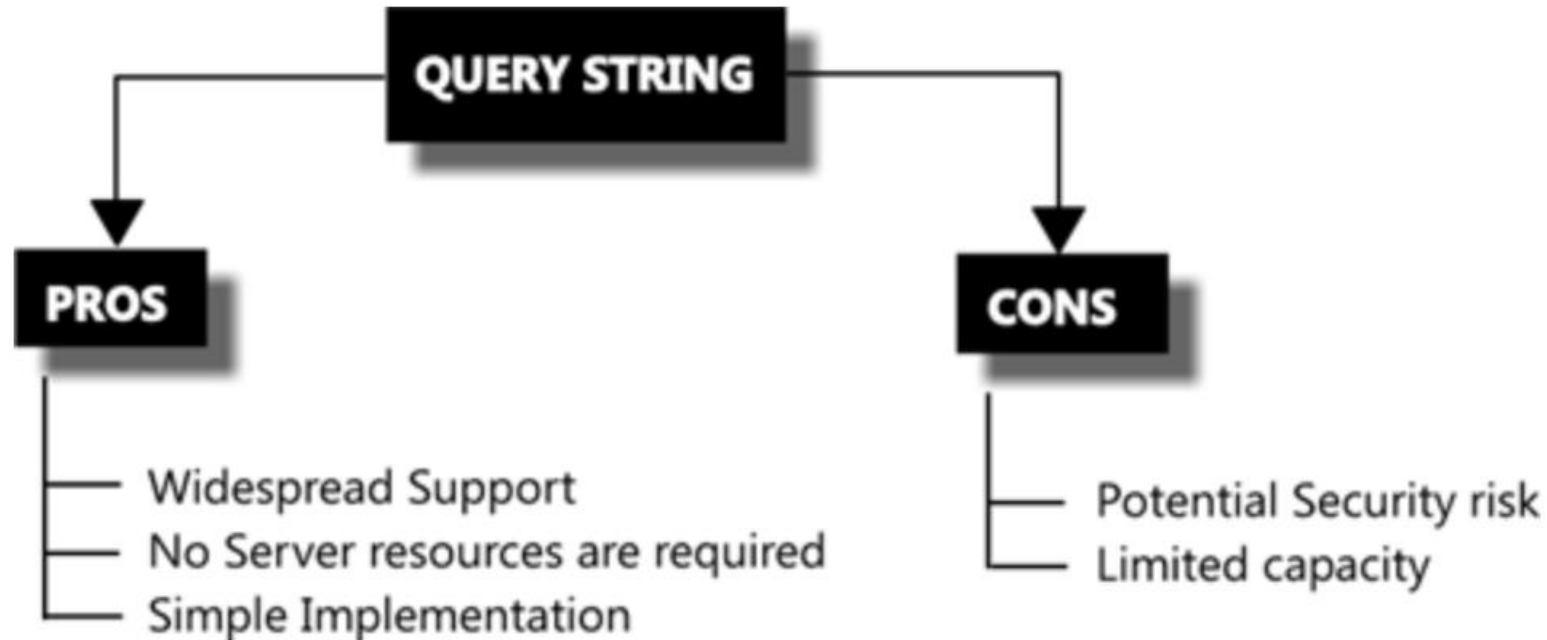
Query Strings

- A query string is information that is appended to the end of a page URL. A typical query string might look like the following example:

```
https://www.contoso.com/listwidgets.aspx?category=basic&price=100
```

- In the URL path above, the query string starts with a question mark (?) and includes two attribute/value pairs, one called "category" and the other called "price."
- Query strings provide a simple but limited way to maintain state information. For example, they are an easy way to pass information from one page to another, such as passing a product number from one page to another page where it will be processed. However, some browsers and client devices impose a 2083-character limit on the length of the URL.
- In order for query string values to be available during page processing, we must submit the page using an HTTP GET command. That is, we cannot take advantage of a query string if a page is processed in response to an HTTP POST command.

Query Strings: Pros & Cons (Nipun Tomar@c-sharpcorner.com)

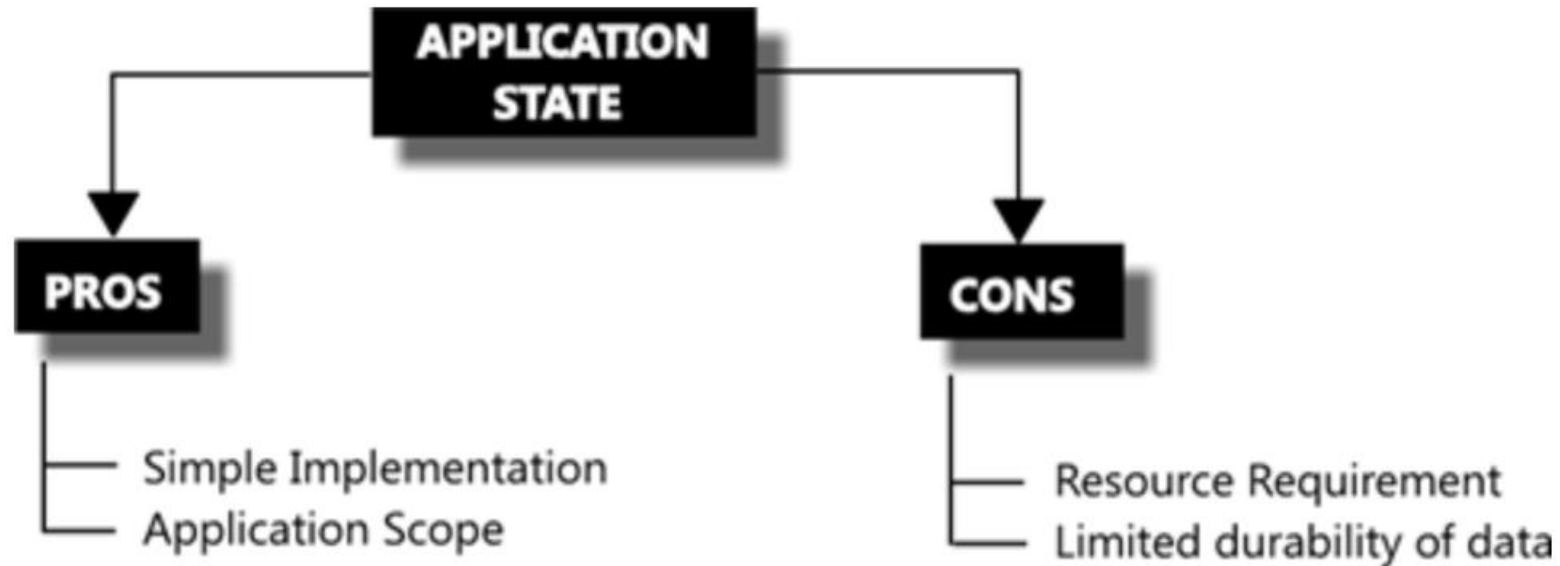


Server-based state mng't: Application State

- ASP.NET allows us to save values using application state — which is an instance of the `HttpApplicationState` class — for each active Web application. Application state is a global storage mechanism that is accessible from all pages in the Web application. Thus, application state is useful for storing information that needs to be maintained between server round trips and between requests for pages.
- Application state is stored in a key/value dictionary that is created during each request to a specific URL. We can add our application-specific information to this structure to store it between page requests.
- Once we add our application-specific information to application state, the server manages it.
- The following example shows how to assign a value in application state.

```
Application["WelcomeMessage"] = "Welcome to the Contoso site.";
```

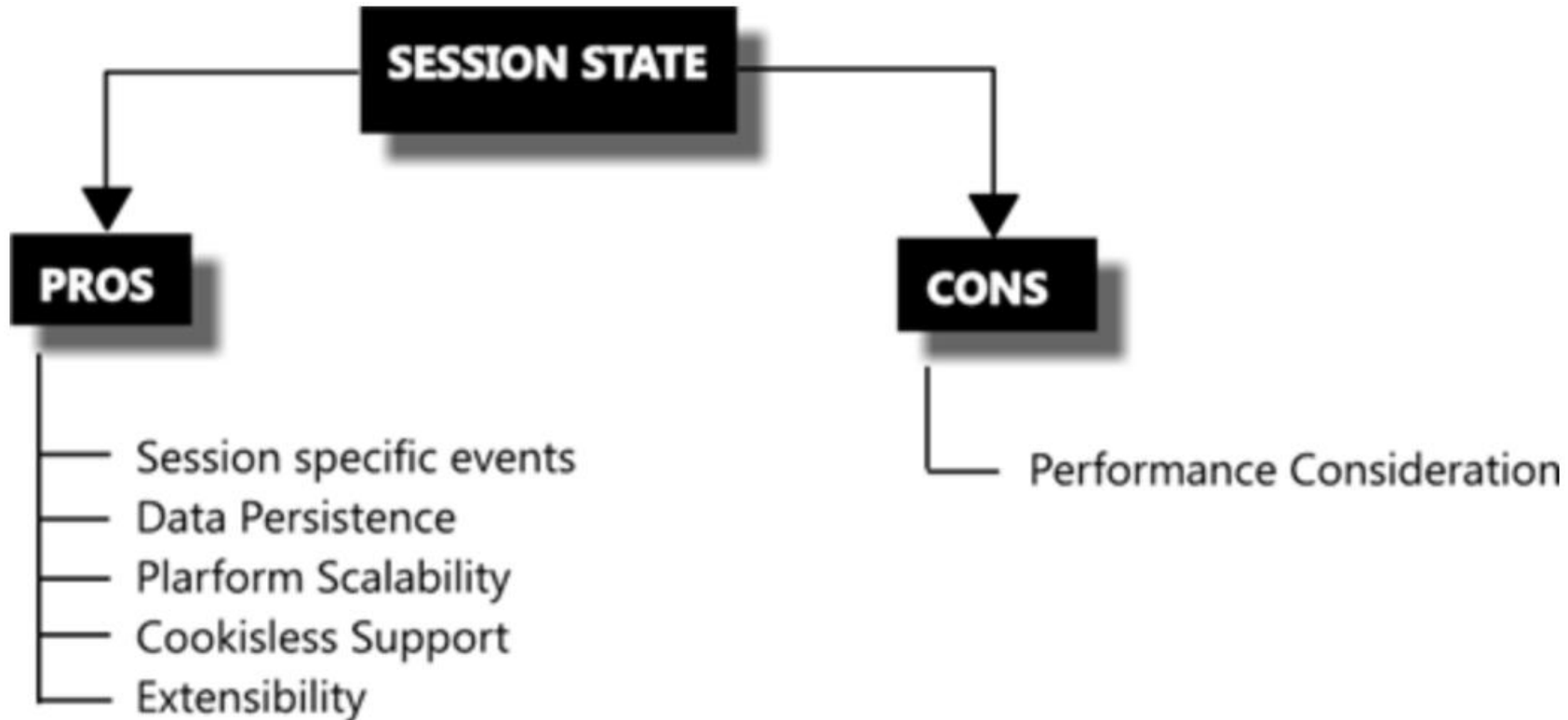
Application State: Pros & Cons (Nipun Tomar@c-sharpcorner.com)



Session State

- ASP.NET allows us to save values by using session state — which is an instance of the `HttpSessionState` class — for each active Web-application session.
- *Session state* is similar to *application state*, except that it is scoped to the current browser session. If different users are using our application, each user session will have a different *session state*. In addition, if a user leaves our application and then returns later, the second user session will have a different session state from the first.
- *Session state* is structured as a key/value dictionary for storing session-specific information that needs to be maintained between server round trips and between requests for pages.
- We can use *session state* to accomplish the following tasks:
 - Uniquely identify browser or client-device requests and map them to an individual session instance on the server.
 - Store session-specific data on the server for use across multiple browser or client-device requests within the same session.
 - Raise appropriate session management *events*. In addition, we can write application code leveraging these events.
- Once we add our application-specific information to session state, the server manages this object. Depending on which options we specify, session information can be stored in cookies, on an out-of-process server, or on a computer running Microsoft SQL Server.

Session State: Pros & Cons (Nipun Tomar@c-sharpcorner.com)



07Session.aspx (Sachin Gargava@codeproject.com)

```
07Session.aspx
1 <%@ Page Language="C#" AutoEventWireup="true"
2   CodeFile="07Session.aspx.cs" Inherits="MySession" %>
3
4 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
5   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
6
7 <html xmlns="http://www.w3.org/1999/xhtml" >
8 <head runat="server">
9   <title>Session</title>
10 </head>
11 <body>
12 <form id="form1" runat="server">
13 <div>
14   <asp:Button ID="Button1" runat="server" Text="Button" OnClick="Button1_Click" />
15   <br />
16   <br />
17   No of postback from starting to application &nbsp;(saved in Application) :
18   <asp:Label ID="Label1" runat="server" ></asp:Label><br />
19   No of postback (saved in Session) :<asp:Label ID="Label2" runat="server" ></asp:Label><br />
20   Total no of click on "Redirect to the same page" button (saved in QueryString) :
21   <asp:Label ID="Label3" runat="server" ></asp:Label><br />
22   <br />
23   <br />
24   <asp:Button ID="Button2" runat="server" Text="Redirect to the same page"
25     OnClick="Button2_Click" /></div>
26 </form>
27 </body>
28 </html>
```

100% No issues found

Button

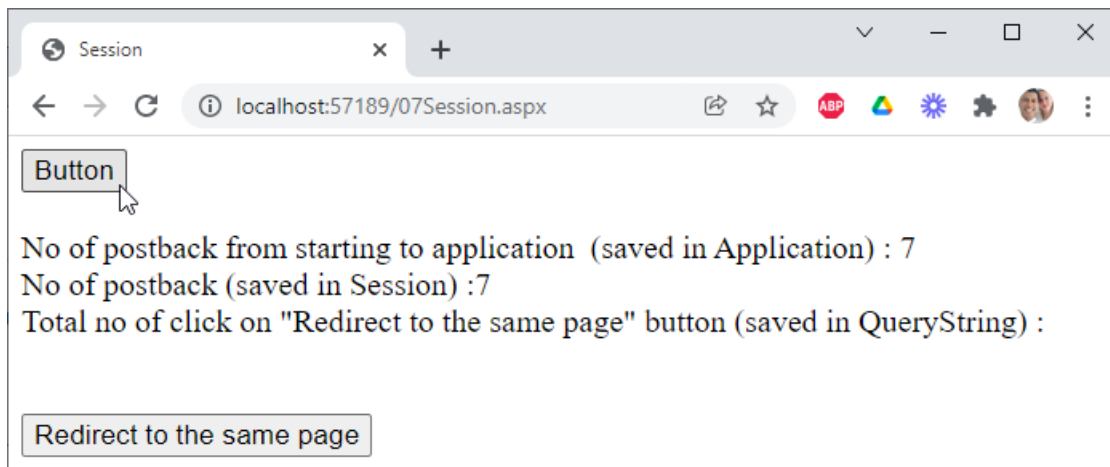
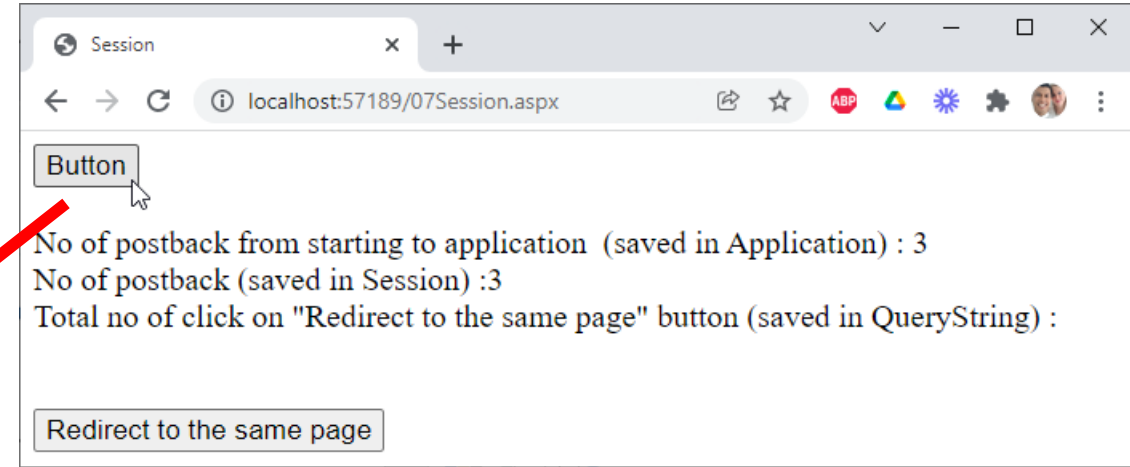
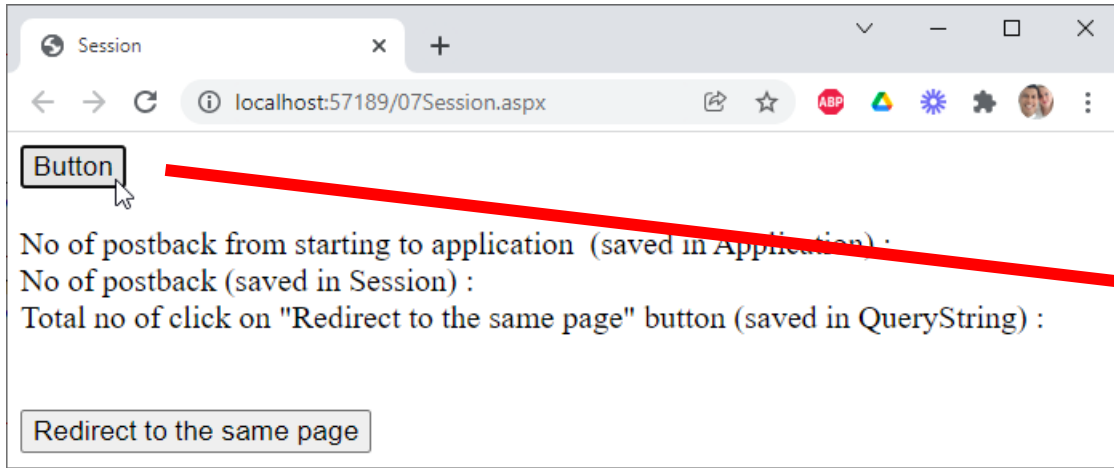
No of postback from starting to application (saved in Application) : [Label1]
No of postback (saved in Session) : [Label2]
Total no of click on "Redirect to the same page" button (saved in QueryString) : [Label3]

Redirect to the same page

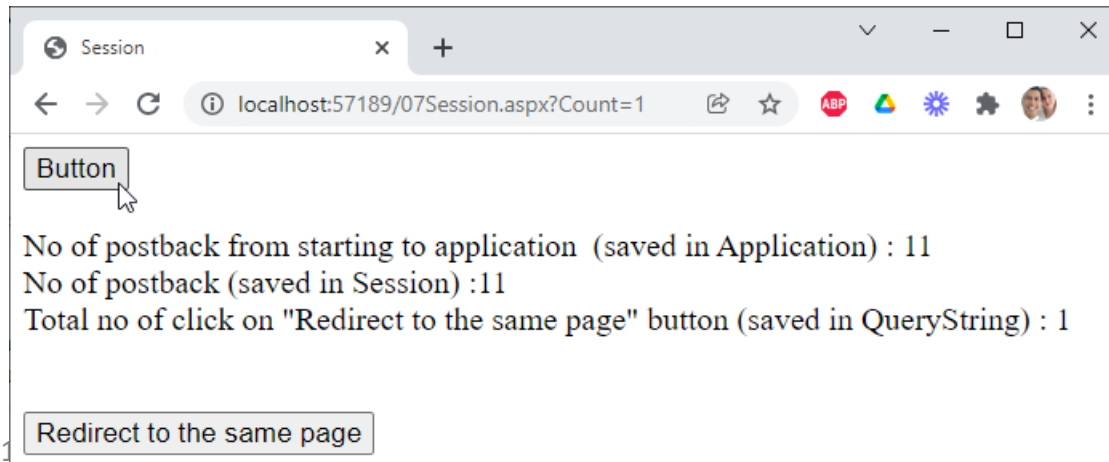
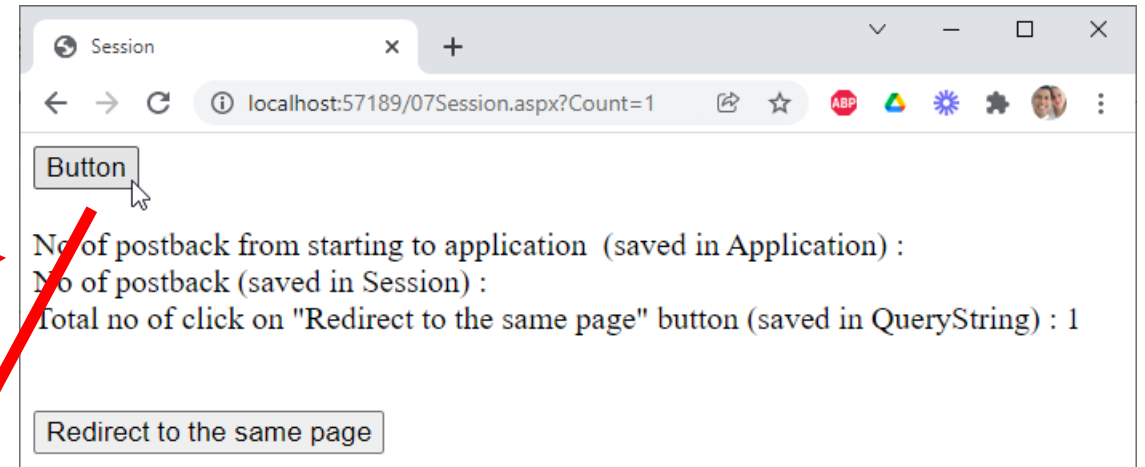
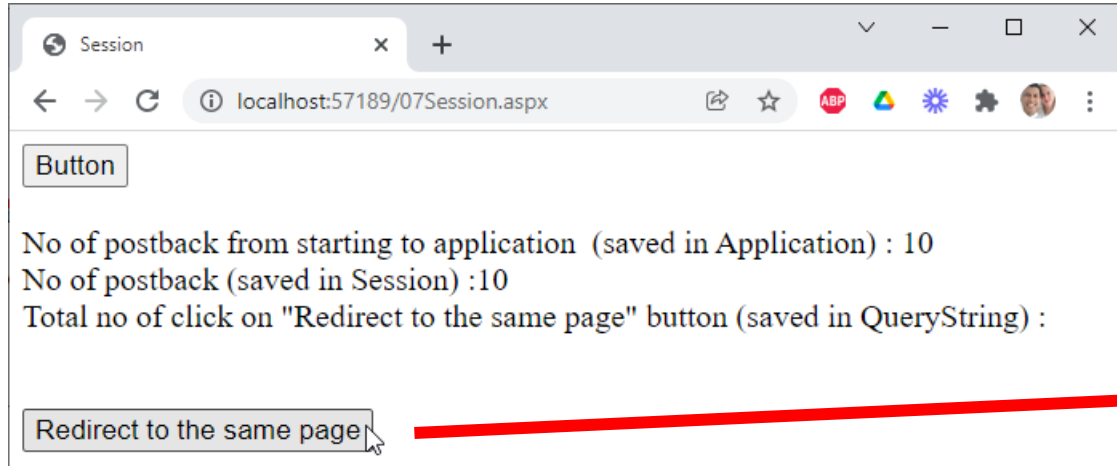
07Session.aspx.cs (Sachin Gargava@codeproject.com)

```
07Session.aspx.cs X
23_07Session.aspx MySession
1 using System;
2 using System.Data;
3 using System.Configuration;
4 using System.Collections;
5 using System.Web;
6 using System.Web.Security;
7 using System.Web.UI;
8 using System.Web.UI.WebControls;
9 using System.Web.UI.WebControls.WebParts;
10 using System.Web.UI.HtmlControls;
11 public partial class MySession : System.Web.UI.Page {
12     protected void Page_Load(object sender, EventArgs e) {
13         if (Request.QueryString["Count"] != null) { // Let's retrieve, increase and store again
14             Label3.Text = Request.QueryString["Count"];
15         }
16     }
17     protected void Button2_Click(object sender, EventArgs e) {
18         int noOfRedirection = 0;
19         if (Request.QueryString["Count"] != null) { // Let's retrieve, increase and store again
20             noOfRedirection = Convert.ToInt32(Request.QueryString["Count"]) + 1;
21         } else { // First postback, let's store the info
22             noOfRedirection = 1;
23         }
24         Response.Redirect("07Session.aspx?Count=" + noOfRedirection);
25     }
26     protected void Button1_Click(object sender, EventArgs e) {
27         Application.Lock();
28         Application["Count"] = Convert.ToInt32(Application["Count"]) + 1;
29         Application.UnLock();
30         Label1.Text = Application["Count"].ToString();
31         //Store in Session State -----
32         Session["Count"] = Convert.ToInt32(Session["Count"]) + 1;
33         Label2.Text = Session["Count"].ToString();
34     }
35 }
```

07Session (Output)



07Session (Output)



07Session (Output)

Session

localhost:57189/07Session.aspx?Count=1

Button

No of postback from starting to application (saved in Application) : 11
No of postback (saved in Session) : 11
Total no of click on "Redirect to the same page" button (saved in QueryString) : 1

Redirect to the same page

Session

localhost:57189/07Session.aspx?Count=1

Button

No of postback from starting to application (saved in Application) : 12
No of postback (saved in Session) : 12
Total no of click on "Redirect to the same page" button (saved in QueryString) : 1

Redirect to the same page

Session

localhost:57189/07Session.aspx?Count=1

Button

No of postback from starting to application (saved in Application) : 12
No of postback (saved in Session) : 12
Total no of click on "Redirect to the same page" button (saved in QueryString) : 1

Redirect to the same page

Session

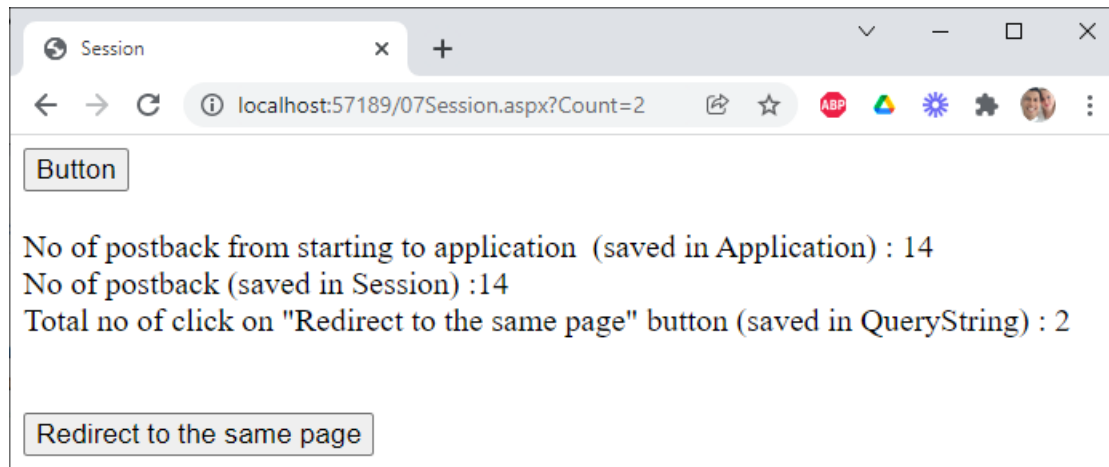
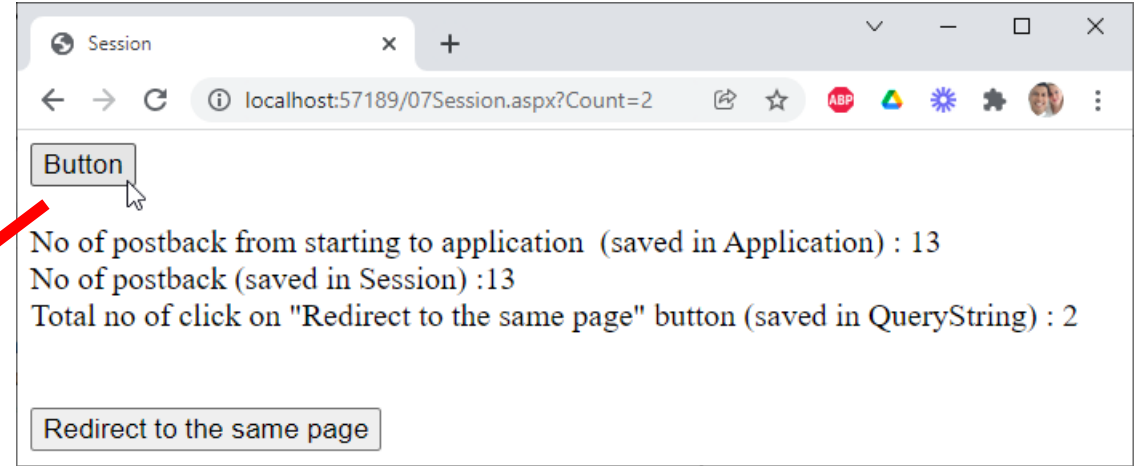
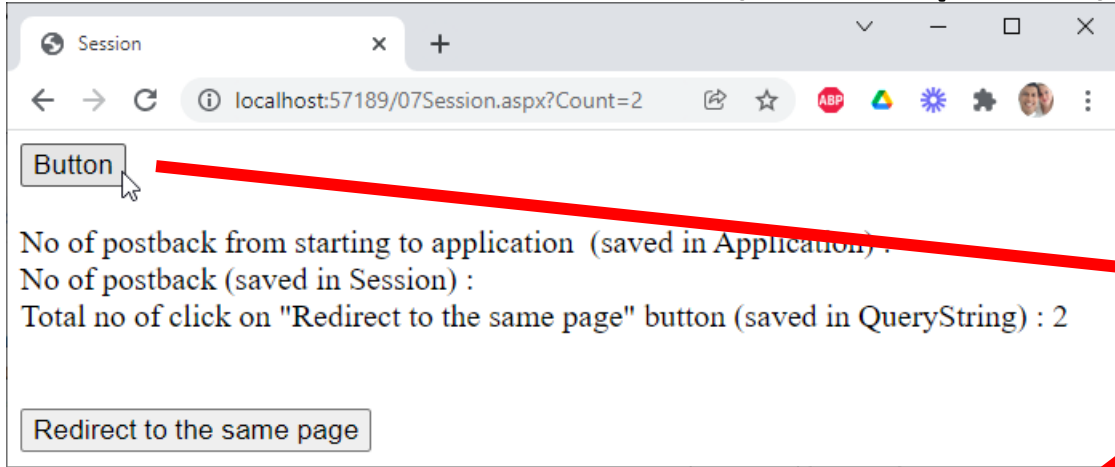
localhost:57189/07Session.aspx?Count=2

Button

No of postback from starting to application (saved in Application) :
No of postback (saved in Session) :
Total no of click on "Redirect to the same page" button (saved in QueryString) : 2

Redirect to the same page

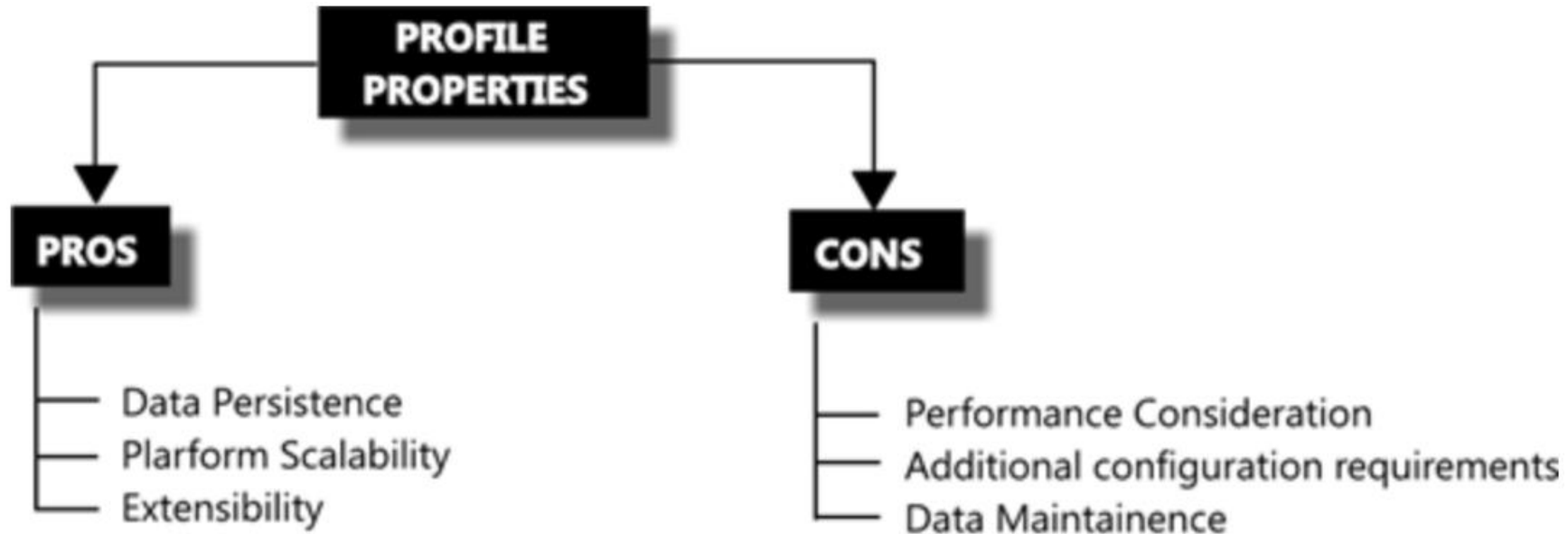
07Session (Output)



Profile Properties

- ASP.NET provides a feature called profile properties, which allows us to store user-specific data. This feature is similar to *session state*, except that the profile data is not lost when a user's session expires. The profile-properties feature uses an ASP.NET profile, which is stored in a persistent format and associated with an individual user. The ASP.NET profile allows us to easily manage user information without requiring us to create and maintain our own database. In addition, the profile makes the user information available using a *strongly typed* API that we can access from anywhere in our application. We can store objects of any type in the profile. The ASP.NET profile feature provides a generic storage system that allows us to define and maintain almost any kind of data while still making the data available in a type-safe manner.
- To use profile properties, we must configure a profile provider. ASP.NET includes a `SqlProfileProvider` class that allows us to store profile data in a SQL database, but we can also create our own profile provider class that stores profile data in a custom format and to a custom storage mechanism such as an XML file, or even to a web service.
- Because data that is placed in profile properties is not stored in application memory, it is preserved through *Internet Information Services* (IIS) restarts and worker-process restarts without losing data. Additionally, profile properties can be persisted across multiple processes such as in a Web farm or a Web garden.

Profile Properties: Pros & Cons (Nipun Tomar@c-sharpcorner.com)



Profile Properties: Example (Nipun Tomar@c-sharpcorner.com)

Sample

```
01. <profile>
02.   <properties>
03.     <add item="item name" />
04.   </properties>
05. </profile>
```

Code Example

```
01. <authentication mode="Windows" />
02. <profile>
03.   <properties>
04.     <add name="FirstName"/>
05.     <add name="LastName"/>
06.     <add name="Age"/>
07.     <add name="City"/>
08.   </properties>
09. </profile>
```