| **Name of Student:** Ajay Karthikesan | | | |
| --- | --- | --- | --- |
| **Roll Number:** 57 | | **Assignment Number:** 8 | |
| **Aim of Assignment:**  State Management | | | |
| **DOP:** | | **DOS:** 3.6.23 | |
| **CO Mapped:**  CO3 | **PO Mapped:**  PO3, PO5, PSO1, PSO2 | **Faculty Signature:** | **Marks:** |

## 

## Practical No. 8

**Aim:** State Management

**Theory:**

State Overview:

As we all know, browsers are generally stateless.

Now the question arises here, what does stateless actually mean?

Stateless means, whenever we visit a website, our browser

communicates with the respective server depending on our

requested functionality or the request. The browser

communicates with the respective server using the HTTP or

HTTPs protocol.

But after that response, what&#39;s next or what will happen when

we visit that website again after closing our web browser?

In this case HTTP/HTTPs doesn&#39;t remember what website or

URL we visited, or in other words we can say it doesn&#39;t hold the

state of a previous website that we visited before closing our

browser, that is called stateless. Now I guess you have at least

an idea of what state and stateless actually means. So our

browsers are stateless.

State Outline:

As I said in the beginning, HTTP is a stateless protocol. It just

cleans up or we can say removes all the resources/references

that were serving a specific request in the past. These resources

can be:

• Objects

• Allocated Memory

• Sessions ID&#39;s

• Some URL info and so on.

State Management Types:

In ASP.NET there are the following 2 State Management

methodologies:

State Management Techniques:

State Management techniques are based on client side and

server side. Their functionality differs depending on the change

in state, so here is the hierarchy:

Client-Side State Management:

Whenever we use Client-Side State Management, the state

related information will directly get stored on the client-side.

That specific information will travel back and communicate

with every request generated by the user then afterwards

provides responses after server-side communication.

This architecture is something like the following,

Hidden Field:

A hidden field is used for storing small amounts of data on the

client side. In most simple words it&#39;s just a container of some

objects but their result is not rendered on our web browser. It is

invisible in the browser.

It stores a value for the single variable and it is the preferable

way when a variable&#39;s value is changed frequently but we don&#39;t

need to keep track of that every time in our application or web

program.

Cookies:

A set of Cookies is a small text file that is stored in the user&#39;s

hard drive using the client&#39;s browser. Cookies are just used for

the sake of the user&#39;s identity matching as it only stores

information such as session’s ids, some frequent navigation or

post-back request objects.

Whenever we get connected to the internet for accessing a

specific service, the cookie file is accessed from our hard drive

via our browser for identifying the user. The cookie access

depends upon the life cycle or expiration of that specific cookie

file.

Query Strings:

Query strings are used for some specific purpose. These in a

general case are used for holding some value from a different

page and move these values to the different page. The

information stored in it can be easily navigated to one page to

another or to the same page as well.

View State:

In general, we can say it is used for storing user data in

ASP.NET, sometimes in ASP.NET applications the user wants

to maintain or store their data temporarily after a post-back. In

this case VIEW STATE is the most used and preferred way of

doing that.

This property is enabled by default but we can make changes

depending on our functionality, what we need to do is just

change the EnableViewState value to either TRUE for enabling

it or FALSE for the opposite operation.

Server-Side State Management:

Server-Side State Management is different from Client-Side

State Management but the operations and working are somewhat

the same in functionality. In Server-Side State Management all

the information is stored in the user memory. Due to this

functionality, there is more secure domains at the server side in

comparison toClient-Side State Management.

The structure is something like the following,

It is another way which ASP.NET provides to store the user&#39;s

specific information or the state of the application on the server

machine. It completely makes use of server resources (the

server&#39;s memory) to store information.

This management technique basically makes use of the

following,

a. Session State

b. Application State

Session State:

Session is one of the most common way which is being used by

developers to maintain the state of the application. The Session

basically stores the values as a dictionary collection in key/value

pairs. It completely utilizes server resources to store the data. It

is a secure way of storing data, since the data will never be

passed to the client.

For each and every user, a separate Session is created, and each

and every Session has its Unique ID. This ID is being stored in

the client&#39;s machine using cookies. If there are multiple users

who are accessing a web application, then for each user a

separate Session is created. If a single user logs in and logs out

the Session is killed, then if the same user again logs into the

application, then a new Session ID is being created for the same

user.

The Session has a default timeout value (20 minutes). We can

also set the timeout value for a session in the web.config file.

There are various ways in which we can store a session and they

are as follows:

1. OFF

2. InProc

3. State Server

4. SQL Server

Application State:

If the information that we want to be accessed or stored globally

throughout the application, even if multiple users access the site

or application at the same time, then we can use an Application

Object for such purposes.

It stores information as a Dictionary Collection in key - value

pairs. This value is accessible across the pages of the application

/ website.

There are 3 events of the Application which are as follows

• Application\_Start

• Application\_Error

• Application\_End

Example

Just for an example, I am setting the Page title in the

Application Start event of the Global.asax file.

the Key and &quot;Welcome to State Management Application is the

value.

**Code:**

File: a/WebForm1.apsx

﻿<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="WebForm1.aspx.cs" Inherits="Practical8.WebForm1" %>

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

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

<head runat="server">

<title></title>

</head>

<body>

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

<div>

<asp:Label ID="Label1" runat="server" Text="Name"></asp:Label>

<asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>

<br />

<asp:Label ID="Label2" runat="server" Text="Age:"></asp:Label>

<asp:TextBox ID="TextBox2" runat="server"></asp:TextBox>

<br />

<br />

<asp:Label ID="Label3" runat="server" Text="Branch:"></asp:Label>

<asp:TextBox ID="TextBox3" runat="server"></asp:TextBox>

<br />

<br />

<asp:Button ID="Button2" runat="server" OnClick="Button2\_Click" Text="Button" />

<asp:Button ID="Button1" runat="server" PostBackUrl="~/WebForm2.aspx"

Text="Submit" />

<br />

<br />

<asp:HiddenField ID="HiddenField1" runat="server" />

<asp:HiddenField ID="HiddenField2" runat="server" />

<asp:HiddenField ID="HiddenField3" runat="server" />

<br />

</div>

</form>

</body>

</html>

File: a/WebForm1.apsx.cs

﻿using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.UI;

using System.Web.UI.WebControls;

namespace Practical8

{

public partial class WebForm1 : System.Web.UI.Page

{

protected void Page\_Load(object sender, EventArgs e)

{

}

protected void Button2\_Click(object sender, EventArgs e)

{

HiddenField1.Value = TextBox1.Text;

HiddenField2.Value = TextBox2.Text;

HiddenField3.Value = TextBox3.Text;

Response.Write(HiddenField1.Value);

Response.Write(HiddenField2.Value);

Response.Write(HiddenField3.Value);

}

}

}

File: a/WebForm2.apsx.cs

﻿<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="WebForm2.aspx.cs" Inherits="Practical8.WebForm2" %>

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

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

<head runat="server">

<title></title>

</head>

<body>

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

<div>

<asp:Label ID="Label1" runat="server" Text="Label"></asp:Label>

<asp:Label ID="Label2" runat="server" Text="Label"></asp:Label>

<asp:Label ID="Label3" runat="server" Text="Label"></asp:Label>

</div>

</form>

</body>

</html>

File: a/WebForm2.apsx.cs

﻿using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.UI;

using System.Web.UI.WebControls;

namespace Practical8

{

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

{

protected void Page\_Load(object sender, EventArgs e)

{

String value1 = Request.Form["HiddenField1"];

String value2 = Request.Form["HiddenField2"];

String value3 = Request.Form["HiddenField3"];

Label1.Text = value1;

Label2.Text = value2;

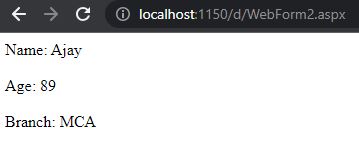
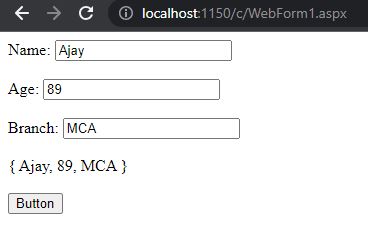
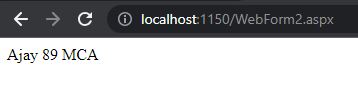
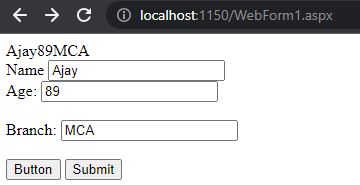
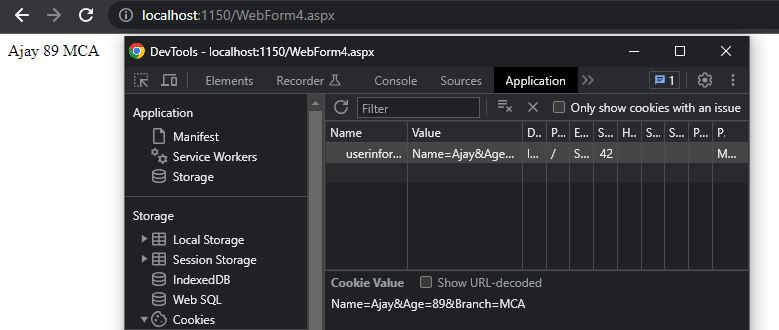
Label3.Text = value3;

}

}

}

**Output:**

e

**Conclusion:**

I learnt how to do state management in ASP.NET.