

## Unit-V

### **ASP.NET – Security**

In ASP.NET, there are two closely interlinked concepts for the security of applications, one is authentication and another is Authorization. In the authentication process, some sort of identification is obtained from the users and using that identification to verify the user's identity. In authorization process is to allow an authenticated user access to resources.

#### **1. Form authentication**

Form authentication in ASP.NET is a security mechanism used to authenticate users by validating their credentials (typically a username and password) before granting access to protected resources. It is part of the ASP.NET identity and security system and works by storing user authentication information in a cookie.

How it Works:

1. **Login Page:** Users attempting to access a protected page are redirected to a login page where they enter their credentials.
2. **Authentication Process:** When the user submits the login form, the credentials are sent to the server, where they are validated against a data source (like a database).
3. **Authentication Ticket:** If the credentials are valid, ASP.NET issues an authentication ticket (usually stored in a cookie) that indicates the user is authenticated.
4. **Access Granted:** The user is then redirected back to the originally requested page, and the ticket (cookie) is used on subsequent requests to verify that the user is authenticated.

### Example of Configuration in web.config:

```
<system.web>
<authentication mode="Forms">
<forms name="demo_cookie" loginUrl="form1.aspx" defaultUrl="welcome.aspx"
timeout="1">
    <credentials passwordFormat="Clear">
    <user name="deepa" password="deepa123"/>
    <user name="priya" password="125"/>
    </credentials>
</forms>
</authentication>
<authorization>
    <deny users="?"/>
</authorization>
</system.web>
```

### Explanation:

- `<forms name="demo_cookie" loginUrl="form1.aspx" defaultUrl="welcome.aspx" timeout="1">`
- `<deny users="?"/>`
- `<credentials passwordFormat="Clear">`

- **Name:** It is the name of the cookies. by default name is .aspxauth
- **loginUrl:** It is the page where user should re-directed if no valid authentication cookie is found. The default value is login.aspx.
- **timeout:** The expiry time of cookie by value is 30.
- **?** indicates anonymous user. If someone tries to access a page without login they are said to be anonymous user.
- **PasswordFormat="Clear":** Password will be in text format.

In our project there are 3 files:

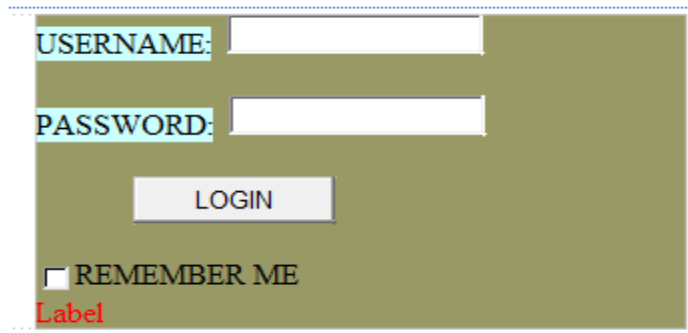
1. Form1.aspx [login page]
2. Welcome.aspx
3. Default.aspx

**If we remove authorization setting from web.config file.**

```
<authorization>
    <deny users="?"/>
</authorization>
```

Then anyone can access any page without login. But if you want to restrict someone from accessing the default or welcome page without login then we have to use authorization setting in web.config.

### **Form1.aspx:**



On button click write down following code:

```
using System.Web.Security;
namespace Authentication
{
    public partial class form1 : System.Web.UI.Page
    {
        protected void Button1_Click(object sender, EventArgs e)
        {
            if(FormsAuthentication.Authenticate(TextBox1.Text, TextBox2.Text))
            {
                FormsAuthentication.RedirectFromLoginPage(TextBox1.Text, true);
            }
            else
            {
                Label3.Text = "invalid username or password";
            }
        }
    }
}
```

- All the file related to security class are available in System.web.security class.
- FormsAuthentication is a class and Authenticate is a method which accepts 2 parameter username and password entered by user.
- Authenticate method verifies the passed username and password against the credential stored in web.config file. It returns true if username and password matches the credential stored in web.config.

```
<credentials passwordFormat="Clear">
```

```
<user name="deepa" password="deepa123"/>
  <user name="priya" password="125"/>
</credentials>
```

- FormsAuthentication.RedirectFromLoginPage(TextBox1.Text, [true](#)): On successful login to redirect on the page this method is used. we will redirect to the page that we have specified in default url.
- It also accepts 2 parameter username and true/false
- True indicates a cookie will be created for a longer period [persistent cookie]
- False indicates a cookie will be active till the browser is open [non-persistent cookie]
- Once our data is stored in the cookie we need to login again.

## Windows Authentication:

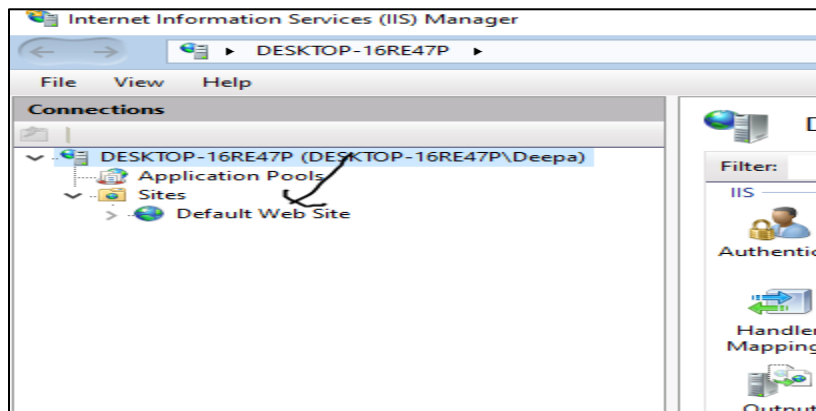
Windows Authentication in ASP.NET is a secure authentication method that integrates with the Windows operating system to authenticate users based on their Windows accounts. It leverages the built-in Windows user management system, such as Active Directory or local user accounts, to determine whether a user should have access to a web application or certain resources.

### How to Enable Windows Authentication in ASP.NET

#### 1. Add IIS MANAGER

2. Click on windows features on/off
3. Select internet information service
4. Select world wide web services
5. Click on security and select windows authentication

#### Open iis manager

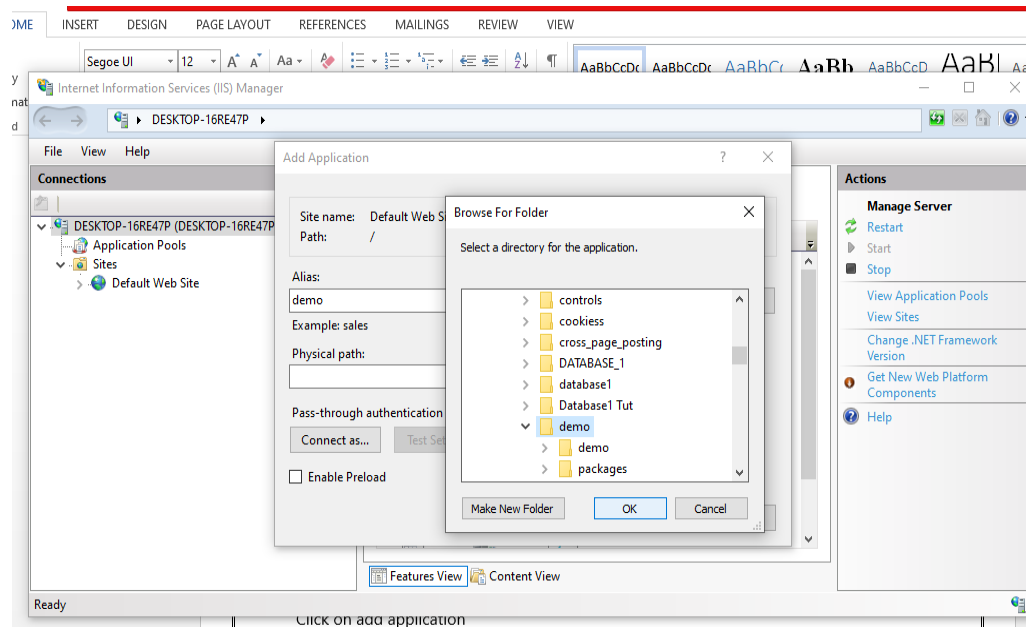


## Right click on default web site

Click on add application

->select the application on which you want to apply windows authentication

->set an alias name



Click on add

After adding double click on demo , click on authentication

➔ Disable anonymous

➔ Enable windows

## 2. Now on web.config write following code

```
<system.web>
  <authentication mode="Windows"></authentication>
  <authorization>

    <allow users="*" />
    <deny users="?" />

  </authorization>
```

## NuGet is a package manager

NuGet is a package manager for the .NET ecosystem that simplifies the process of managing and distributing libraries, tools, and dependencies in .NET projects. It

automates the process of installing, updating, and removing libraries (called packages) that a .NET application depends on.

### Key Features of NuGet:

1. **Package Management:**
  - NuGet makes it easy to add third-party libraries (or your own libraries) to a .NET project. These libraries are bundled as "NuGet packages."
2. **Package Repository:**
  - NuGet has a central repository, **NuGet.org**, which hosts thousands of publicly available packages. You can also configure private repositories for internal use.
3. **Dependency Management:**
  - When you install a package, NuGet automatically installs the dependencies (other packages) required by that package, ensuring everything works seamlessly.
4. **Version Control:**
  - NuGet allows developers to specify the version of a package they want to use, providing flexibility to stay on a stable version or upgrade as needed.
5. **Package Creation:**
  - NuGet enables developers to create their own packages. These packages can be shared with other developers, either through private repositories or by publishing them to NuGet.org.
6. **Cross-platform:**
  - NuGet supports different platforms, including .NET Framework, .NET Core, .NET 5/6/7+, and Xamarin. This helps you manage packages consistently across different types of .NET applications.

### NuGet Package Manager in Visual Studio:

NuGet is tightly integrated into Visual Studio, offering a graphical user interface (GUI) for package management:

- **Manage NuGet Packages:** Right-click the project in Solution Explorer → Choose "Manage NuGet Packages" → Browse for the package you want.
- **NuGet Package Manager Console:** An interactive console inside Visual Studio where you can execute NuGet commands.

Installing a NuGet package in Visual Studio is a straightforward process. You can do it using the **NuGet Package Manager** UI or the **Package Manager Console**. Below are the steps for both methods.

## Using the NuGet Package Manager (GUI)

1. **Open your project** in Visual Studio.
2. **Right-click on the project** in the **Solution Explorer**.
3. Select **Manage NuGet Packages...** from the context menu.

This will open the NuGet Package Manager window.

4. **Browse for a Package:**
  - In the NuGet Package Manager window, go to the **Browse** tab.
  - Search for the package you want to install (e.g., Newtonsoft.Json).
  - Once found, click on the package to see details.
5. **Select the Package:**
  - In the package details pane, choose the appropriate **version** you want to install.
6. **Click "Install":**
  - The package and its dependencies will be downloaded and installed.
  - You might be prompted to **accept the license terms**, which you should review and accept.
7. **Check Installed Packages:**
  - Once installed, you can view all the installed packages under the **Installed** tab in the NuGet Package Manager.

## Bootstrap:

**Bootstrap** is a popular, open-source front-end framework for developing responsive, mobile-first websites and web applications. It provides a set of CSS and JavaScript-based design templates that help developers create modern, clean, and responsive designs quickly.

Key Features of Bootstrap:

1. **Responsive Grid System:**
  - Bootstrap uses a **12-column grid system** that allows you to create layouts that adjust to different screen sizes (mobile, tablet, desktop).
  - The grid is divided into rows and columns, and you can use classes like col-md-6, col-lg-4, etc., to control how elements are laid out on different screen sizes.
2. **Predefined CSS Classes:**

- Bootstrap comes with a variety of CSS classes for styling elements like typography, buttons, forms, tables, images, and more.
  - It helps developers avoid writing custom CSS by providing ready-to-use classes like .btn, .card, .navbar, .container, etc.
3. **Responsive Design:**
- Bootstrap is **mobile-first**, meaning its styles prioritize mobile devices and then scale up for larger screens.
  - It automatically adjusts layouts based on screen size, ensuring your website looks good on any device (smartphones, tablets, desktops).
4. **JavaScript Components:**
- Bootstrap includes **JavaScript plugins** powered by jQuery for adding interactivity to elements. Some popular components are:
    - **Modals:** Popup windows.
    - **Dropdowns:** Toggleable menus.
    - **Tooltips:** Text that appears when hovering over elements.
    - **Carousels:** Slideshow-like elements.
    - **Collapsibles:** Expanding and collapsing panels.

#### **Add bootstrap in web form:**

Step: write click on project->select manage NuGet packages->browse->search for bootstrap->install

#### **Add following link in head tag from content and script folder to use bootstrap :**

```
<link href="Content/bootstrap.min.css" rel="stylesheet" />
<script src="Scripts/bootstrap.bundle.js"></script>
<script src="Scripts/bootstrap.min.js"></script>
```

### **Typography:**

Bootstrap provides a comprehensive set of **typography** classes to style headings, paragraphs, lists, inline text elements, and other text-related content. These classes make it easy to ensure that text content is consistent, well-spaced, and responsive across different devices. Below is a breakdown of how typography works in Bootstrap.

#### **1. Headings**

Bootstrap offers various heading sizes (<h1> to <h6>), which scale with different font sizes and styles. Additionally, it provides utility classes for more control over headings.



## HTML Heading Tags

Bootstrap automatically applies basic styles to HTML heading tags (<h1> to <h6>).

```
<h1>Heading 1</h1>
<h2>Heading 2</h2>
<h3>Heading 3</h3>
<h4>Heading 4</h4>
<h5>Heading 5</h5>
<h6>Heading 6</h6>
```

## Display Headings

For more prominent headings, use **display** classes (.display-1 to .display-6).

```
<h1 class="display-1">Display Heading 1</h1>
<h2 class="display-2">Display Heading 2</h2>
<h3 class="display-3">Display Heading 3</h3>
<h4 class="display-4">Display Heading 4</h4>
```

## 3. Text Alignment

Bootstrap provides utility classes to align text within an element.

- **Left-aligned text** (default):  

```
<p class="text-start">Left-aligned text.</p>
```
- **Center-aligned text:**  

```
<p class="text-center">Center-aligned text.</p>
```
- **Right-aligned text:**  

```
<p class="text-end">Right-aligned text.</p>
```

## 4. Text Colors

You can apply various text colors using predefined Bootstrap classes. These include primary, secondary, success, danger, warning, info, light, dark, and muted colors.

```
<p class="text-primary">This text is primary color.</p>
<p class="text-secondary">This text is secondary color.</p>
<p class="text-success">This text is success color.</p>
<p class="text-danger">This text is danger color.</p>
```

```
<p class="text-warning">This text is warning color.</p>
<p class="text-info">This text is info color.</p>
<p class="text-light bg-dark">This text is light color on a dark background.</p>
<p class="text-dark">This text is dark color.</p>
<p class="text-muted">This text is muted (faded) color.</p>
```

## 5. Text Transformation

Bootstrap allows you to change text casing (capitalization) with these utility classes:

- **Uppercase:**  

```
<p class="text-uppercase">This text is uppercase.</p>
```
- **Lowercase:**  

```
<p class="text-lowercase">THIS TEXT IS LOWERCASE.</p>
```
- **Capitalize** (first letter of each word):  

```
<p class="text-capitalize">this text is capitalized.</p>
```

## 6. Text Decoration

Control text decoration with utility classes like `text-decoration-none`, `text-decoration-underline`, and `text-decoration-line-through`.

- **No underline:**  

```
<a href="#" class="text-decoration-none">This link has no underline.</a>
```
- **Underline text:**  

```
<p class="text-decoration-underline">This text is underlined.</p>
```
- **Strike through text:**  

```
<p class="text-decoration-line-through">This text has a line through it.</p>
```

## 7. Lists

Bootstrap styles HTML lists (`<ul>`, `<ol>`, and `<dl>`) and offers additional utility classes.

### Unordered List

```
<ul>
  <li>Item 1</li>
  <li>Item 2</li>
  <li>Item 3</li>
</ul>
```

### **Ordered List**

```
<ol>
  <li>First item</li>
  <li>Second item</li>
  <li>Third item</li>
</ol>
```

### **Inline List**

.list-inline and .list-inline-item to make list items display inline (horizontally).

```
<ul class="list-inline">
  <li class="list-inline-item">Item 1</li>
  <li class="list-inline-item">Item 2</li>
  <li class="list-inline-item">Item 3</li>
</ul>
```

## **8. Responsive Text Alignment and Display**

You can control text alignment and display properties based on screen size using responsive utility classes:

- **Responsive text alignment:**

```
<p class="text-md-start">Left-aligned on medium and up</p>
<p class="text-md-center">Center-aligned on medium and up</p>
<p class="text-md-end">Right-aligned on medium and up</p>
```

### **Bootstrap Buttons:**

Bootstrap provides a wide variety of buttons with different styles, sizes, and behaviors. These buttons are built using predefined classes that are easy to apply and customize. Below is a comprehensive guide to Bootstrap buttons.

## 1. Basic Button Styles

Buttons in Bootstrap are created using the `<button>` or `<a>` elements with the `.btn` class. You can apply different styles by adding contextual classes like `.btn-primary`, `.btn-secondary`, etc.

### Basic Button Example

```
<button type="button" class="btn btn-primary">Primary Button</button>
<button type="button" class="btn btn-secondary">Secondary Button</button>
<button type="button" class="btn btn-success">Success Button</button>
<button type="button" class="btn btn-danger">Danger Button</button>
<button type="button" class="btn btn-warning">Warning Button</button>
<button type="button" class="btn btn-info">Info Button</button>
<button type="button" class="btn btn-light">Light Button</button>
<button type="button" class="btn btn-dark">Dark Button</button>
```

### Link Button Example

You can also use `<a>` elements styled like buttons.

```
<a href="#" class="btn btn-primary">Link Button</a>
```

## 2. Button Color Variants

Bootstrap offers various color classes to style buttons. These are the most commonly used color variants:

- **Primary** (`btn-primary`): Used for main actions.
- **Secondary** (`btn-secondary`): Used for alternative actions.
- **Success** (`btn-success`): Indicates successful or positive actions.
- **Danger** (`btn-danger`): Represents dangerous or negative actions.
- **Warning** (`btn-warning`): Alerts the user or draws attention.
- **Info** (`btn-info`): Used for informational purposes.
- **Light** (`btn-light`): For lighter-colored buttons.
- **Dark** (`btn-dark`): For darker buttons.

### 3. Outline Buttons

Bootstrap also provides **outline buttons**, which have a transparent background with a colored border. These are useful when you want a lighter, more subtle button.

- Example:

```
<button type="button" class="btn btn-outline-primary">Primary  
Outline</button>  
<button type="button" class="btn btn-outline-secondary">Secondary  
Outline</button>  
<button type="button" class="btn btn-outline-success">Success  
Outline</button>  
<button type="button" class="btn btn-outline-danger">Danger  
Outline</button>  
<button type="button" class="btn btn-outline-warning">Warning  
Outline</button>  
<button type="button" class="btn btn-outline-info">Info Outline</button>  
<button type="button" class="btn btn-outline-light">Light Outline</button>  
<button type="button" class="btn btn-outline-dark">Dark Outline</button>
```

### 4. Button Sizes

Bootstrap buttons can be resized using the `.btn-lg`, `.btn-sm`, and `.btn-block` classes.

#### ***Large Buttons***

```
<button type="button" class="btn btn-primary btn-lg">Large Button</button>
```

#### ***Small Buttons***

```
<button type="button" class="btn btn-primary btn-sm">Small Button</button>
```

#### ***Block-Level Buttons***

Use the `.btn-block` class to make a button take the full width of its parent container.

```
<button type="button" class="btn btn-primary btn-block">Block Button</button>
```

## **Bootstrap Forms:**

Form controls automatically receive some global styling with Bootstrap:

All textual `<input>`, `<textarea>`, and `<select>` elements with class `.form-control` have a width of 100%.

## Bootstrap Form Layouts

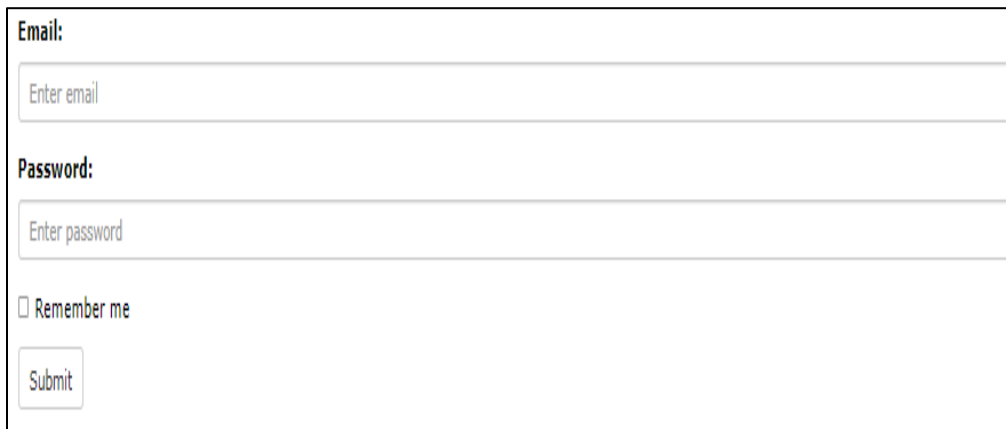
Bootstrap provides three types of form layouts:

- Vertical form (this is default)
- Horizontal form
- Inline form

### Standard rules for all three form layouts:

- Wrap labels and form controls in `<div class="form-group">` (needed for optimum spacing)
- Add class `.form-control` to all textual `<input>`, `<textarea>`, and `<select>` elements

## Bootstrap Vertical Form (default)



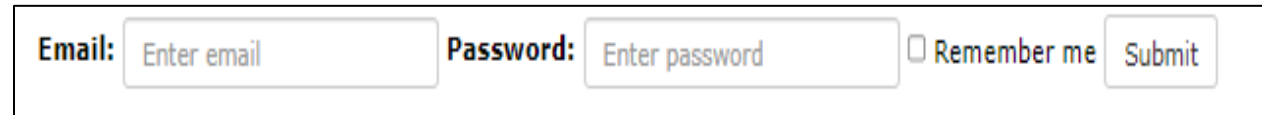
The following example creates a vertical form with two input fields, one checkbox, and a submit button:

### Example

```
<form action="/action_page.php">
  <div class="form-group">
    <label for="email">Email address:</label>
    <input type="email" class="form-control" id="email">
  </div>
  <div class="form-group">
    <label for="pwd">Password:</label>
    <input type="password" class="form-control" id="pwd">
  </div>
```

```
<div class="checkbox">
  <label><input type="checkbox"> Remember me</label>
</div>
<button type="submit" class="btn btn-default">Submit</button>
</form>
```

## Bootstrap Inline Form



In an inline form, all of the elements are inline, left-aligned, and the labels are alongside.

**Note: This only applies to forms within viewports that are at least 768px wide!**

Additional rule for an inline form:

- Add class `.form-inline` to the `<form>` element

The following example creates an inline form with two input fields, one checkbox, and one submit button:

### Example

```
<form class="form-inline" action="/action_page.php">
  <div class="form-group">
    <label for="email">Email address:</label>
    <input type="email" class="form-control" id="email">
  </div>
  <div class="form-group">
    <label for="pwd">Password:</label>
    <input type="password" class="form-control" id="pwd">
  </div>
  <div class="checkbox">
    <label><input type="checkbox"> Remember me</label>
  </div>
  <button type="submit" class="btn btn-default">Submit</button>
</form>
```

## Bootstrap Horizontal Form



A horizontal form means that the labels are aligned next to the input field (horizontal) on large and medium screens. On small screens (767px and below), it will transform to a vertical form (labels are placed on top of each input).

Additional rules for a horizontal form:

- Add class `.form-horizontal` to the `<form>` element
- Add class `.control-label` to all `<label>` elements

**Tip:** Use Bootstrap's predefined grid classes to align labels and groups of form controls in a horizontal layout.

The following example creates a horizontal form with two input fields, one checkbox, and one submit button.

### Example

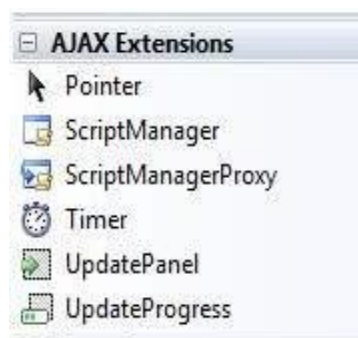
```
<form class="form-horizontal" action="/action_page.php">
  <div class="form-group">
    <label class="control-label col-sm-2" for="email">Email:</label>
    <div class="col-sm-10">
      <input type="email" class="form-
control" id="email" placeholder="Enter email">
    </div>
  </div>
  <div class="form-group">
    <label class="control-label col-sm-2" for="pwd">Password:</label>
    <div class="col-sm-10">
      <input type="password" class="form-
control" id="pwd" placeholder="Enter password">
    </div>
  </div>
  <div class="form-group">
```



```
<div class="col-sm-offset-2 col-sm-10">
  <div class="checkbox">
    <label><input type="checkbox"> Remember me</label>
  </div>
</div>
<div class="form-group">
  <div class="col-sm-offset-2 col-sm-10">
    <button type="submit" class="btn btn-default">Submit</button>
  </div>
</div>
</form>
```

## AJAX :

- With AJAX, web applications can retrieve data from the server asynchronously, in the background, without reloading the entire browser page. The use of AJAX has led to an increase in interactive animation on web pages.
- AJAX stands for Asynchronous JavaScript and XML. This is a cross platform technology which speeds up response time. The AJAX server controls add script to the page which is executed and processed by the browser.
- However like other ASP.NET server controls, these AJAX server controls also can have methods and event handlers associated with them, which are processed on the server side.
- The control toolbox in the Visual Studio IDE contains a group of controls called the 'AJAX Extensions'



## The ScriptManager Control

The ScriptManager control is the most important control and must be present on the page for other controls to work.

**It has the basic syntax:**

```
<asp:ScriptManager ID="ScriptManager1" runat="server">
</asp:ScriptManager>
```

If you create an 'Ajax Enabled site' or add an 'AJAX Web Form' from the 'Add Item' dialog box, the web form automatically contains the script manager control. The ScriptManager control takes care of the client-side script for all the server side controls.

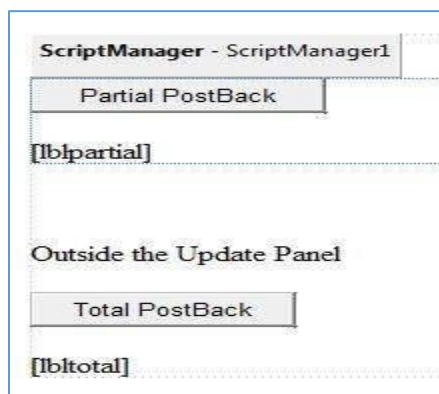
## The UpdatePanel Control

- The UpdatePanel control is a container control and derives from the Control class. It acts as a container for the child controls within it and does not have its own interface. When a control inside it triggers a post back, the UpdatePanel intervenes to initiate the post asynchronously and update just that portion of the page.
- For example, if a button control is inside the update panel and it is clicked, only the controls within the update panel will be affected, the controls on the other parts of the page will not be affected. This is called the partial post back or the asynchronous post back.

## Example

Add an AJAX web form in your application. It contains the script manager control by default. Insert an update panel. Place a button control along with a label control within the update panel. Place another set of button and label outside the panel.

**The design view looks as follows:**



```
<form id="form1" runat="server">
  <div>
    <asp:ScriptManager ID="ScriptManager1" runat="server" />
```

```

</div>

<asp:UpdatePanel ID="UpdatePanel1" runat="server">
  <ContentTemplate>
    <asp:Button ID="btnpartial" runat="server" onclick="btnpartial_Click"
Text="Partial PostBack"/>
    <br />
    <br />
    <asp:Label ID="lblpartial" runat="server"> </asp:Label>
  </ContentTemplate>
</asp:UpdatePanel>

<p> </p>
<p>Outside the Update Panel</p>
<p>
  <asp:Button ID="btntotal" runat="server" onclick="btntotal_Click" Text="Total
PostBack" />
</p>
<asp:Label ID="lbltotal" runat="server"> </asp:Label>
</form>

```

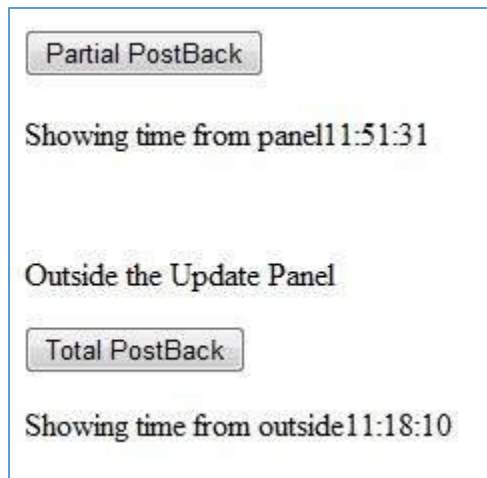
**Both the button controls have same code for the event handler:**

```

string time = DateTime.Now.ToLongTimeString();
lblpartial.Text = "Showing time from panel" + time;
lbltotal.Text = "Showing time from outside" + time;

```

Observe that when the page is executed, if the total post back button is clicked, it updates time in both the labels but if the partial post back button is clicked, it only updates the label within the update panel.



A page can contain multiple update panels with each panel containing other controls like a grid and displaying different part of data.

When a total post back occurs, the update panel content is updated by default. This default mode could be changed by changing the UpdateMode property of the control. Let us look at other properties of the update panel.

## The UpdateProgress Control

The UpdateProgress control provides a sort of feedback on the browser while one or more update panel controls are being updated. For example, while a user logs in or waits for server response while performing some database oriented job.

It provides a visual acknowledgement like "Loading page...", indicating the work is in progress.

The syntax for the UpdateProgress control is:

```
<asp:UpdateProgress ID="UpdateProgress1" runat="server"
DynamicLayout="true" AssociatedUpdatePanelID="UpdatePanel1" >

    <ProgressTemplate>
        Loading...
    </ProgressTemplate>

</asp:UpdateProgress>
```

The above snippet shows a simple message within the ProgressTemplate tag. However, it could be an image or other relevant controls. The UpdateProgress control displays for every asynchronous postback unless it is assigned to a single update panel using the AssociatedUpdatePanelID property.

## The Timer Control

The timer control is used to initiate the post back automatically. This could be done in two ways:

(1) Setting the Triggers property of the UpdatePanel control:

```
<Triggers>  
  <asp:AsyncPostBackTrigger ControlID="btnpanel2" EventName="Click" />  
</Triggers>
```

(2) Placing a timer control directly inside the UpdatePanel to act as a child control trigger. A single timer can be the trigger for multiple UpdatePanels.

```
<asp:UpdatePanel ID="UpdatePanel1" runat="server"  
UpdateMode="Always">  
  
  <ContentTemplate>  
    <asp:Timer ID="Timer1" runat="server" Interval="1000">  
      </asp:Timer>  
  
    <asp:Label ID="Label1" runat="server" Height="101px"  
style="width:304px" >  
      </asp:Label>  
    </ContentTemplate>  
  
</asp:UpdatePanel>
```

### Timer:

```
Label1.Text=DateTime.Now.ToString("hh:mm:ss");//48
```

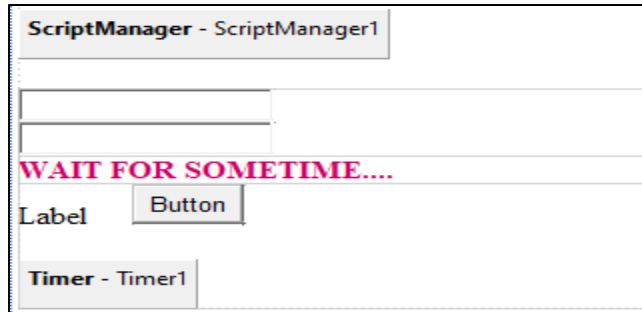
## **UPDATEPROGRESS:**

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

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

    </div>
    <p>
      <asp:Label ID="Label3" runat="server" Text="Label"> </asp:Label>
      <asp:ScriptManager ID="ScriptManager1" runat="server">
        </asp:ScriptManager>
      </p>
      <asp:UpdatePanel ID="UpdatePanel1" runat="server">

        <ContentTemplate>
          <asp:TextBox ID="TextBox1" runat="server"> </asp:TextBox>
          <br />
          <asp:TextBox ID="TextBox2" runat="server"> </asp:TextBox>
          <br />
          <asp:UpdateProgress ID="UpdateProgress1" runat="server">
            <ProgressTemplate>
              <asp:Label ID="Label4" runat="server" Text="WAIT FOR SOMETIME...." Font-
Bold="True" ForeColor="#CC0066"> </asp:Label>
            </ProgressTemplate>
          </asp:UpdateProgress>
          <asp:Label ID="Label2" runat="server" Text="Label"> </asp:Label>
        <asp:Button ID="Button1" runat="server" CssClass="auto-style1" OnClick="Button1_Click1"
Text="Button" />
        <br />
        <br />
        <asp:Timer ID="Timer1" runat="server" Interval="1000"
OnTick="Timer1_Tick"> </asp:Timer>
      </ContentTemplate>
    </asp:UpdatePanel>
  </form>
</body>
</html>
```



```
protected void Button1_Click1(object sender, EventArgs e)
{
    System.Threading.Thread.Sleep(2000);
    Label2.Text = TextBox2.Text + TextBox1.Text;
}
```

## We can also add image or gif in progress bar

```
<asp:UpdateProgress ID="UpdateProgress1" runat="server">
    <ProgressTemplate>

asp:Image ID="Image1" runat="server" Height="55px"
ImageUrl="~/Youtube_loading_symbol_1_(wobbly).gif" Width="193px" />

    </ProgressTemplate>
</asp:UpdateProgress>
```

## Tracing in asp.net:

ASP.NET tracing enables you to view diagnostic information about a single request for an ASP.NET page. ASP.NET tracing enables you to follow a page's execution path, display diagnostic information at run time, and debug your application. ASP.NET tracing can be integrated with system-level tracing to provide multiple levels of tracing output in distributed and multi-tier applications.

There are two ways:

1. Page Level Tracing
2. Application Level Tracing

### Page Level Tracing

We can control whether tracing is enabled or disabled for individual pages. If tracing is enabled, when the page is requested, ASP.NET appends to the page a series of tables containing

execution details about the page request. Tracing is disabled by default in an ASP.NET application.

To enable Page Level Tracing follow the steps:

i) Include Trace="true" in <%@ Page Title="" Language="C#"...%> directive, for example:

```
1. <%@ Page Title="" Language="C#" MasterPageFile="~/MasterPage.master" AutoEventWireup="true" CodeFile="Default.aspx.cs" Inherits="chat_Default" Trace="true"%>
```

Look at the above code, I'll be using Trace=true at the end.

ii) Now press F5 to run the application, you will see the immediate trace record on an existing page that you have set Trace and TraceMode.

You can see the traced record in the Trace Viewer as well; you will learn this in 'Application Level Tracing'.

### **Application Level Tracing**

Instead of enabling tracing for individual pages, you can enable it for your entire application. In that case, every page in your application displays trace information. Application tracing is useful when you are developing an application because you can easily enable it and disable it without editing individual pages. When your application is complete, you can turn off tracing for all pages at once.

When you enable tracing for an application, ASP.NET collects trace information for each request to the application, up to the maximum number of requests you specify. The default number of requests is 10. You can view trace information with the trace viewer.

By default, when the trace viewer reaches its request limit, the application stops storing trace requests. However, you can configure application-level tracing to always store the most recent tracing data, discarding the oldest data when the maximum number of requests is reached.

To enable Application Level Tracing follow the steps:

i) Delete your Page Level Tracking for better result.

ii) Open the Web.config file and add the following information to it; if there is not a Web.config file available then add a new one in the root.

```
1. <system.web>
2.   <trace enabled="true" pageOutput="true" requestLimit="40" localOnly="false"/>
3. </system.web>
4. </configuration>
```

iii) The above code has many attributes used, find the detailed information about them below.



Enabled: Set it true to enable tracing for the application; otherwise, false. The default is false. You can override this setting for individual pages by setting the Trace attribute in the @ Page directive of a page to true or false.

PageOutput: Set it true to display trace both in pages and in the trace viewer (trace.axd); otherwise, false. The default is false.

RequestLimit: The number of trace requests to store on the server. The default is 10.

LocalOnly: Set it true to make the trace viewer (trace.axd) available only on the host Web server; otherwise, false. The default is true.

Request Details			
Session Id:	ziunghd1moc24jg3dnu0c5p5	Request Type:	GET
Time of Request:	6/4/2011 1:00:40 AM	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	0.0187102690425738	0.018710
aspx.page	Begin Init	0.0187967325456168	0.000086
aspx.page	End Init	0.0188343071535628	0.000038
aspx.page	Begin InitComplete	0.0188460404883861	0.000012
aspx.page	End InitComplete	0.0188584722359965	0.000012
aspx.page	Begin PreLoad	0.0188688087452456	0.000010
aspx.page	End PreLoad	0.0188798436672817	0.000011
aspx.page	Begin Load	0.0188900404939734	0.000010
aspx.page	End Load	0.0190406881321509	0.000151
aspx.page	Begin LoadComplete	0.0190546563878929	0.000014
aspx.page	End LoadComplete	0.0190672278180607	0.000013
aspx.page	Begin PreRender	0.0190778436924246	0.000011
aspx.page	End PreRender	0.0190994944888247	0.000022
aspx.page	Begin PreRenderComplete	0.0191124151125386	0.000013
aspx.page	End PreRenderComplete	0.0191239389363732	0.000012
aspx.page	Begin SaveState	0.0195118373983286	0.000388
aspx.page	End SaveState	0.0556956261200795	0.036184
aspx.page	Begin SaveStateComplete	0.0557392769192733	0.000044
aspx.page	End SaveStateComplete	0.148471802980546	0.092733
aspx.page	Begin Render	0.148538082354042	0.000066
aspx.page	End Render	0.149570406294655	0.001032

\*\*\*\*