

CS-32: Programming with ASP.NET

UNIT – 4

Master Pages and Theme Caching,
Application Pages and Data

Dharmesh Kapadiya
Dhruvita Savaliya

TOPICS :

- What is Master Page?
- Requirement Of a Master Page in an Asp.NET application
- Designing Website with Master Page, Theme and CSS
- Overview of Caching
 - Page Output Caching
 - Partial Page Caching, Absolute Cache Expiration
 - Sliding Cache Expiration
 - Data Caching

What is Master Page?

- Master Page enables you to share the same content among multiple content pages in a website.
- You can use a Master Page to create a common page layout.
- By taking advantage of Master Pages, you can make your website easier to maintain, extend, and modify.
- If you need to add a new page to your website that looks just like the other pages in your website, you simply need to apply the same Master Page to the new content page.
- If you decide to completely modify the design of your website, you do not need to change every content page.
- You can modify just a single Master Page to dramatically change the appearance of all the pages in your application.

- **Advantages of Master Pages :**
- Master pages provide functionality that developers have traditionally created by copying existing code, text, and control elements repeatedly; using framesets; using include files for common elements; using ASP.NET user controls; and so on.
- They allow you to centralize the common functionality of your pages so that you can make updates in just one place.
- They make it easy to create one set of controls and code and apply the results to a set of pages.
- For example, you can use controls on the master page to create a menu that applies to all pages.
- They give you fine-grained control over the layout of the final page by allowing you to control how the placeholder controls are rendered.
- They provide an object model that allows you to customize the master page from individual content pages.

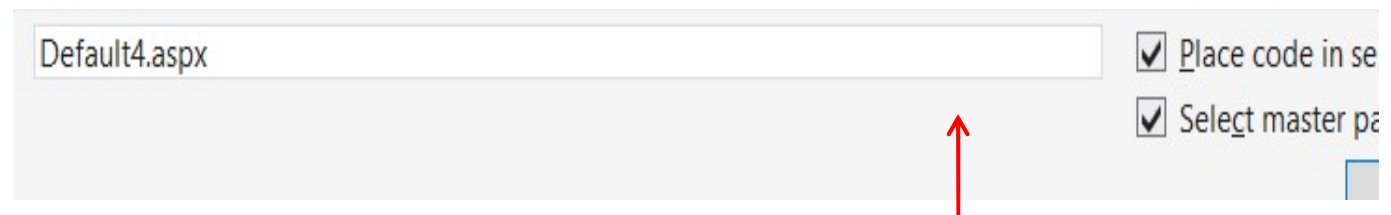
- **How to Create a Master Page :**

- Step 1 : Add master Page (Who has .master extension)

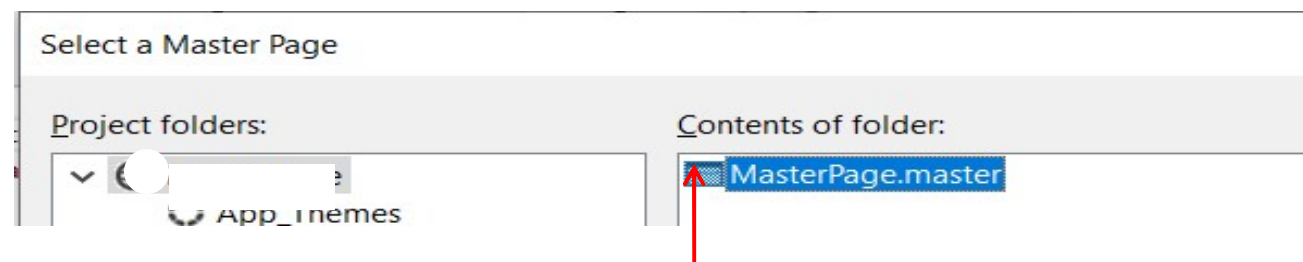
Right click on project → Add New Item → Click on Master Page → Give Appropriate Name to master file → Add.

- Step 2 : Add Content page (Who has .aspx extension)

Right click on project → Add New Item → Click on Web Form → Give Appropriate Name to aspx file → On the Right Side Checked a checkbox of Select Master Page → Click on Add



- ✓ Now Other Screen is open for selecting the master file → Select your Master file that creating on step - 1 → Ok



Example of master page :

File.master

```
<%@ Master Language="C#"
    AutoEventWireup="true"
    CodeFile="MasterPage.master.cs"
    Inherits="MasterPage" %>
```

```
<!DOCTYPE html>
```

```
<html
    xmlns="http://www.w3.org/199
    9/xhtml">
```

```
<head runat="server">
```

```
    <title></title>
```

```
    <asp:ContentPlaceholder
    id="head" runat="server">
```

```
    </asp:ContentPlaceholder>
```

```
</head>
```

```
<body>
```

```
<div id="dvheader">
```

```
    <h1>
```

```
        This is header content
```

```
    </h1>
```

```
</div>
```

```
<div style="min-height: 500px">
```

```
    <asp:ContentPlaceholder
```

```
    ID="ContentPlaceholder1"
```

```
    runat="server">
```

```
        </asp:ContentPlaceholder>
```

```
</div>
```

```
<div id="dvfooter">
```

```
    <h1>
```

```
        This is footer content
```

```
    </h1>
```

```
</div>
```

```
</body>
```

```
</html>
```

Default2.aspx file :

```
<%@ Page Title="" Language="C#"
    MasterPageFile="~/MasterPage.master"
    AutoEventWireup="true" CodeFile="Default2.aspx.cs"
    Inherits="Default2" %>
<asp:Content ID="Content1" ContentPlaceHolderID="head"
    Runat="Server">    </asp:Content>
<asp:Content ID="Content2"
    ContentPlaceHolderID="ContentPlaceHolder1"
    Runat="Server">
    <form runat="server">
        <asp:TextBox ID="txtName" runat="server"></asp:TextBox>
        <asp:Button ID="btnSubmit" runat="server" Text="Submit"
            OnClick="btnSubmit_Click" />
    </form>
</asp:Content>
```

- **Run-time Behavior of Master Pages**
- At run time, master pages are handled in the following sequence:
- Users request a page by typing the URL of the content page.
- When the page is fetched, the @ Page directive is read. If the directive references a master page, the master page is read as well. If this is the first time the pages have been requested, both pages are compiled.
- The master page with the updated content is merged into the control tree of the content page.
- The content of individual Content controls is merged into the corresponding ContentPlaceHolder control in the master page.
- The resulting merged page is rendered to the browser.

❖ Master Pages and Themes

- You cannot directly apply an ASP.NET theme to a master page. If you add a theme attribute to the @ Master directive, the page will raise an error when it runs.
- However, themes are applied to master pages under these circumstances:
- If a theme is defined in the content page. Master pages are resolved in the context of content pages, so the content page's theme is applied to the master page as well.

❖ Scoping Master Pages

- You can attach content pages to a master page at three levels:
- **At the page level**
- You can use a page directive in each content page to bind it to a master page, as in the following code example.
- **C#::<%@ Page Language="C#"
MasterPageFile="MySite.Master" %>**

- **At the application level**
- By making a setting in the pages element of the application's configuration file (Web.config), you can specify that all ASP.NET pages (.aspx files) in the application automatically bind to a master page.
- **<pages theme="Theme1" />**
- If you use this strategy, all ASP.NET pages in the application that have Content controls are merged with the specified master page.
- (If an ASP.NET page does not contain Content controls, the master page is not applied.)
- **At the folder level**
- This strategy is like binding at the application level, except that you make the setting in a Web.config file in one folder only.
- The masterpage bindings then apply to the ASP.NET pages in that folder.
- **Creating Master Pages :**
- You create a Master Page by creating a file that ends with the .master extension. You can locate a Master Page file any place within an application.
- Furthermore, you can add multiple Master Pages to the same application.

- Example of Theme Registration in **web.config** file :

```
<?xml version="1.0"?>
```

```
<!--
```

For more information on how to configure your ASP.NET application, please visit

<http://go.microsoft.com/fwlink/?LinkId=169433>

```
-->
```

```
<configuration>
```

```
  <system.web>
```

```
    <pages theme="Theme1"/>
```

```
    <compilation debug="true" targetFramework="4.5" />
```

```
    <httpRuntime targetFramework="4.5" />
```

```
  </system.web>
```

```
</configuration>
```

What is Theme?

- Themes are similar to Cascading Style Sheets (CSS) in that they enable you to define visual styles for your Web pages.
- Themes go further than CSS, however, in that they enable you to apply styles, graphics, and even CSS files themselves to the pages of your applications.
- You can apply ASP.NET themes at the application, page, or server control level.
- **Theme vs CSS:**
- Themes can define many properties of a control or page, not just style properties. For example, using themes, you can specify the graphics for a TreeView control, the template layout of a GridView control, and so on.
- Themes can include graphics.
- Themes do not cascade the way style sheets do. By default, any property values defined in a theme referenced by a page's Theme property override the property values declaratively set on a control.
- Only one theme can be applied to each page. You cannot apply multiple themes to a page, unlike style sheets where multiple style sheets can be applied.

- **Folder Structure for Theme:**
- In order to create your own theme, you have to follow the proper folder structure in your application.
- Create a App_Theme folder by right clicking your application, select ASP.NET Folder→Theme.
- Notice the icon its paintbrush rather than normal folder icon
- Within the App_Themes folder, you can create an additional theme folder for each and every theme that you might use in your application.
- Each theme folder must contain the elements of the theme, which can include the following:
 - A single skin file
 - CSS files
 - Images

Microsoft Visual Studio

VIEW WEBSITE BUILD DEBUG TEAM TOOLS TEST ARCHITECTURE

Google Chrome Debug

config X

```

1 <?xml version="1.0"?>
2
3 <!--
4   For more information on
5   http://go.microsoft.com

```

Bin

App_Code

App_GlobalResources

App_LocalResources

App_WebReferences

App_Data

App_Browsers

Theme

Add New Item... Ctrl+Shift+A

Existing Item... Shift+Alt+A

New Folder

Add ASP.NET Folder

New Virtual Directory...

Reference...

Service Reference...

Web Form

Web User Control

JavaScript File

Style Sheet

Web Form (with master)

Master Page

SQL Server Database

Global Application Class

Build Web Site

Publish Web Site

Scope to This

New Solution Explorer View

Add

View Class Diagram

Manage NuGet Packages...

Copy Web Site...

Start Options...

Set as StartUp Project

View in Browser (Google Chrome) Ctrl+Shift

View in Page Inspector Ctrl+K, Ct

Browse With...

Refresh Folder

Source Control

Cut Ctrl+X

Copy Ctrl+C

Paste Ctrl+V

Remove Del

- **Apply theme to Single Page:**
- You can set the theme name in Page directive like follow:
- **Using Theme:**
- `<%@ Page Language="C#" Theme="Theme_name" %>`
- **Using StyleSheetTheme**
- `<%@ Page Language="C#" StyleSheetTheme=" Theme_name " %>`
- **Apply theme to Entire Application:**
- In web.config file define the following:
- `<configuration>`
 - `<system.web>`
 - `<pages theme=" Theme_name " />`
 - `</system.web>`
- `</configuration>`
- **Removing Theme from any control and Page:**
- **Control**
- `<asp:Textbox ID="TextBox1" runat="server"`
- `BackColor="#000000" ForeColor="#ffffff" EnableTheming="false" />`
- **Page**
- `<%@ Page Language="C#" EnableTheming="False" %>`

StyleSheetTheme vs Theme:

- The StyleSheetTheme attribute works the same as the Theme attribute in that you can use it to apply a theme to a page.
- The difference is that when the attributes are set locally on the page within a particular control, the attributes are overridden by the theme if you use the Theme attribute where is StyleSheetTheme is remain in place even if they are explicitly defined in theme.
- `< %@ Page Language="C#" StyleSheetTheme="Summer" %>`
- **Example:**
- `< asp:Textbox ID="TextBox1" runat="server" BackColor="#000000" ForeColor="#ffffff" />`
- So when we use Theme these attributes are override by attributes defined in Theme.
- While if we apply theme using StyleSheetTheme these attributes are remain as it is even if they are already defined in theme file.

Theme Element – Skin File:

- *A skin is a definition of styles applied to the server controls in your ASP.NET page.*
- *Skins can work in conjunction with CSS files or images.*
- To create a theme to use in your ASP.NET applications, you use just a single skin file in the theme folder.
- The skin file can have any name, but it must have a **.skin** file extension.
- The control definitions must contain the runat="server" attribute .
- There are two types of control skins,
 - Default Skin
 - Named Skin
- **How to define control definition?**

```
<asp:[controlname] runat="server" Attribute1="value"  
    Attribute2="value"  
.....> </asp:[controlname]>
```

- **Default Skin**
 - A default skin is automatically applied to all controls of the same type when a theme is applied to a page.
 - A control skin is a default skin.
 - Default skin does not have a SkinID.
- **Named Skin**
 - A named skin is a control skin with a SkinID property set. Named skins do not automatically apply to controls by type.

- **Code of Skin File :**

- `<asp:TextBox runat="server" BackColor="Red"></asp:TextBox>`

← Default skin

- `<asp:TextBox SkinId="txt_skin" runat="server" BackColor="Yellow"></asp:TextBox>`

← Named skin

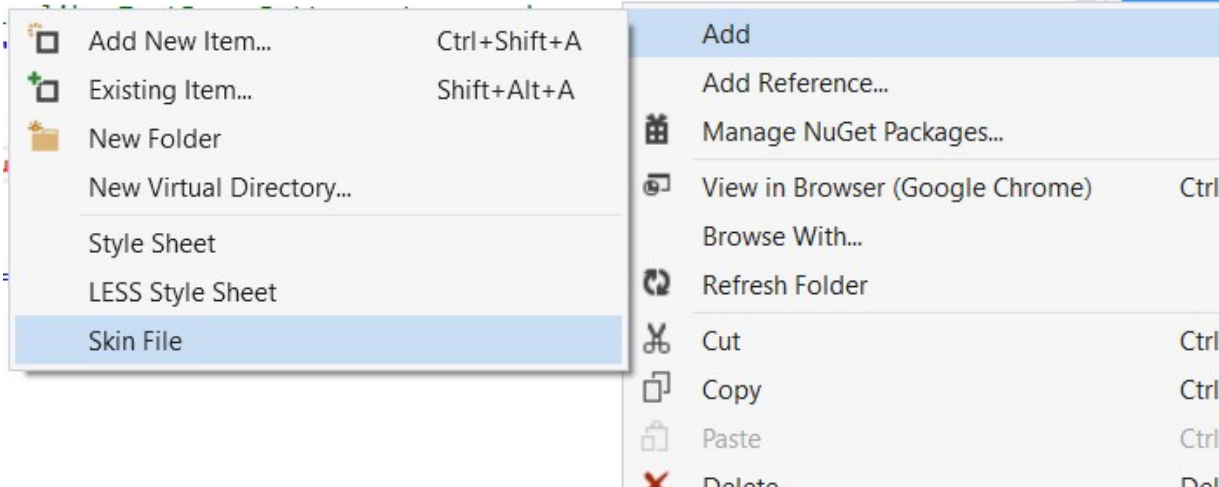
- **Code of .aspx file :**

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

`

Output :

- `<asp:TextBox ID="pwd" runat="server" SkinID="txt_skin"></asp:TextBox>`



← How to add
← skin file

- **Theme Element – CSS File**
- ASP.NET pages are routinely made up of Server controls, raw HTML, tags or even raw or plain text.
- The Server Control definition(style) are defined in .skin file
- So for rest of the things we have to make CSS file inside the theme because when user change the theme everything including the plain text also setup according to the theme.

```
body
{
font-size: x-small; font-family: Verdana; color: red;
}
a:link { color: Blue;
text-decoration: none;
}
a:visited
{
color: Blue;
text-decoration: none;
}
a:hover { color: Red;
text-decoration: underline overline;
}
```

- **Theme Element - Image**

- The coolest reason why Theme not CSS is that themes enables you to incorporate actual images into the style definition.
- Many controls use images to create a better visual appearance.
- First step in this is that you create a Image folder in your theme and put required image into it.
- You can directly incorporate the image from the folder using .skin file.

- **Example - .skin file**

```
<asp:TreeView runat="server" BorderColor="#FFFFFF"
BackColor="#FFFFFF" ForeColor="#585880"
Font-Names="Verdana"
LeafNodeStyle-ImageUrl="image\2.jpeg"
RootNodeStyle-ImageUrl="image\1.jpeg"
ParentNodeStyle-ImageUrl="image\3.jpeg"
NodeIndent="30">
</asp:TreeView>
```

Overview of Caching :

- **What is Caching?**
- Caching technique allows to store/cache page output or application data on the client.
- The cached information is used to serve subsequent requests that avoid the overhead of recreating the same information.
- This enhances performance when same information is requested many times by the user.
- **Advantages of Caching**
 - It increases performance of the application by serving user with cached output.
 - It decreases server round trips for fetching data from database by persisting data in the memory.
 - It greatly reduces overhead from server resources.
- **Disadvantages :**
- It will delay while accessing Website as the data needs to be written before accessing.
- It would be an overhead for the Client to carry out this activity again and again.
- It would lead to an increase in maintenance.
- as the cached data needs to get updated regularly.
- Again there would also be scalability issues.

- **Reason to USE Caching :**
- Every time the page request some information, it has to go through cycle consisting of Client, Server and Database.
- Again in the response it has to go through another cycle consisting of database, server and client.
- This might increase the time, which in turn affect the performance, that's why to improve performance caching is used.
- When multiple users want to access the same information again and again at that time caching is considered as one of the important aspect.
- Here, data would be cached in memory and thus avoiding the database access with every pages request, which increases the performance of the applications.

- ASP.NET provides the following different types of caching:
- **Output Caching** : Output cache stores a copy of the finally rendered HTML pages or part of pages sent to the client. When the next client requests for this page, instead of regenerating the page, a cached copy of the page is sent, thus saving time.
- **Data Caching** : Data caching means caching data from a data source.
As long as the cache is not expired, a request for the data will be fulfilled from the cache.
When the cache is expired, fresh data is obtained by the data source and the cache is refilled.
- **Object Caching** : Object caching is caching the objects on a page, such as data-bound controls. The cached data is stored in server memory.
- **Class Caching** : Web pages or web services are compiled into a page class in the assembly, when run for the first time. Then the assembly is cached in the server. Next time when a request is made for the page or service, the cached assembly is referred to. When the source code is changed, the CLR recompiles the assembly.
- **Configuration Caching** : Application wide configuration information is stored in a configuration file.
Configuration caching stores the configuration information in the server memory.
- **We will learn following Types of Caching :**
 1. Page Output Caching
 2. Partial Page Caching
 3. Data Caching

Page Output Caching :

- The page output caching is one of the simplest technique of caching in ASP.NET.
- In this technique complete HTML output of a rendered ASP.NET page is stored in cache and all the subsequent requests of the page are sent from the cache itself.
- The cache engine checks the cache for the existing page requests. This engine is also known as **cache server**, as it acts as a server that stores web pages and other internet content of request locally.
- If the engine finds a request for the requested page in the cache, the cached HTML output is sent to the client.
- If no request is found, then the engine starts rendering the page again and saves a copy of the HTML output in cache.
- **Syntax for OutputCache directive:**
`<%@ OutputCache Duration="15" VaryByParam="None" %>`

Attribute	Values	Description
Duration	Number	Number of seconds the page or control is cached.
VaryByParam	None * Param- name	Semicolon delimited list of string specifies query string values in a GET request or variable in a POST request.
DiskCacheable	true/false	Specifies that output could be written to a disk based cache.
NoStore	true/false	Specifies that the "no store" cache control header is sent or not.
CacheProfile	String name	Name of a cache profile as to be stored in web.config.
VaryByHeader	* Header names	Semicolon delimited list of strings specifies headers that might be submitted by a client.
VaryByCustom	Browser Custom string	Tells ASP.NET to vary the output cache by browser name and version or by a custom string.
Location	Any Client Downstream Server None	Any: page may be cached anywhere. Client: cached content remains at browser. Downstream: cached content stored in downstream and server both. Server: cached content saved only on server. None: disables caching.

Default.aspx

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Default.aspx.cs"
    Inherits="_Default" %>
```

```
<%@ OutputCache Duration="20" VaryByParam="none" %>
```

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

```
<div>
```

Server Response :

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

```
<br />
```

Client Response :

Server Response : 09-03-2025 5.59.01 PM

```
<script>
```

Client Response : Sun Mar 09 2025 17:59:01 GMT+0530 (India)

```
    document.write(Date());
```

```
</script>
```

Server Response : 09-03-2025 5.59.01 PM

```
</div>
```

Client Response : Sun Mar 09 2025 17:59:13 GMT+0530 (India)

```
</form>
```

Default.aspx.cs

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

```
{
```

```
    Label1.Text = DateTime.Now.ToString();
```

```
}
```

You can notice the difference in time of server & client side

Data Caching :

- The main aspect of data caching is caching the data source controls. We have already discussed that the data source controls represent data in a data source, like a database or an XML file. These controls derive from the abstract class `DataSourceControl` and have the following inherited properties for implementing caching:
- **CacheDuration** - It sets the number of seconds for which the data source will cache data.
- **CacheExpirationPolicy** - It defines the cache behavior when the data in cache has expired.
- **CacheKeyDependency** - It identifies a key for the controls that auto-expires the content of its cache when removed.
- **EnableCaching** - It specifies whether or not to cache the data.

- For creating a data cache, we have to use Cache Keyword. It is available in the **System.Web.Caching namespace**.
- Basic **Syntax** :
`Cache["VariableName"]="Value";`
- There are three main things needed to perform data caching :
- Inserting Value in Cache
`Cache.Insert(Key, Value)`
- Retrieve the Cached Value
`Cache("Key")` or `Cache["key"]`
- Deleting the Cached Value
`Cache.Remove("Key");`

- Simple Data Caching Using Insert Remove Method :

- **File.aspx :**

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

```
  <div>
```

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

```
    <asp:Button ID="btn_ins" runat="server" Text="Create Data  
Cache" OnClick="btn_ins_Click" />
```

```
    <asp:Button ID="btn_rmv" runat="server" Text="Remove Data  
Cache" OnClick="btn_rmv_Click" />
```

```
  </div>
```

```
</form>
```

File.aspx.cs

```
protected void Page_Load(object sender, EventArgs e)
{
    if(Cache["user"]==null)
    {
        Label1.Text = "Cache Was Removed";
    }
    else
    {
        Label1.Text = Cache["user"].ToString();
    }
}

protected void btn_ins_Click(object sender, EventArgs e)
{
    Cache.Insert("user","TYBCA");    //Cache["user"] = "TYBCA";
}

protected void btn_rmv_Click(object sender, EventArgs e)
{
    Cache.Remove("user");
}
```

- **File.aspx of DataCaching :**

```

<form id="form1" runat="server">
  <div>
    <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox><br />
    <asp:TextBox ID="TextBox2" runat="server"></asp:TextBox><br />
    <asp:Button ID="Button1" runat="server" Text="Insert Record" OnClick="Button1_Click"
    /><br />
    <asp:GridView ID="GridView1" runat="server" AutoGenerateColumns="False"
    DataKeyNames="Id" DataSourceID="SqlDataSource1">
      <Columns>
        <asp:BoundField DataField="Id" HeaderText="Id" ReadOnly="True"
        SortExpression="Id" />
        <asp:BoundField DataField="name" HeaderText="name" SortExpression="name" />
        <asp:BoundField DataField="city" HeaderText="city" SortExpression="city" />
      </Columns>
    </asp:GridView>
    <asp:SqlDataSource ID="SqlDataSource1" runat="server" ConnectionString="<%%$
    ConnectionStrings:ConnectionString %>" EnableCaching="True" CacheDuration="25"
    SelectCommand="SELECT * FROM [employee]"></asp:SqlDataSource>
  </div>
</form>

```

- **File.aspx.cs : for insert dynamic record**

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

```
{
```

```
    SqlDataAdapter adpt = new SqlDataAdapter("insert into employee  
values('" + TextBox1.Text + "','" + TextBox2.Text + "') ",con);
```

```
    DataSet ds = new DataSet();
```

```
    adpt.Fill(ds);
```

```
}
```

- You can check Refreshed data only after 25 seconds.
- Because till 25 seconds you can able to watch data from cache, after 25 seconds cache will be destroyed and new created.

Partial Page Caching / Fragment Caching :

- Caching of a complete requested web page is not required always.
- We may need implement caching on specific part of a web page.
- The remaining content of the web page would be refreshed each time the page is requested, This is called Partial Page Caching or fragment caching.
- **Step to apply partial page caching :**
- Identify the parts of web page that requires more processing than the other parts, while rendering the page.
- Create **user controls** and decide the caching policies for them.
- Caching policies would check the validity of the cached data on each user request.
- Add the user controls in the web application.
- Now caching of the defined user controls would only be done.
- While other controls would be refreshed for each request.

Creating a User Control :

- In ASP.NET, a user control is a reusable, self-contained component that encapsulates UI elements and functionality, similar to a mini-page, allowing developers to create and reuse common UI patterns across multiple web pages.
- User controls enable developers to create reusable UI elements and logic, promoting code reusability and simplifying the development of complex web applications.
- User controls are created using the **.ascx** file extension, similar to ASP.NET pages (.aspx).
- They contain both markup (HTML and server controls) and code-behind logic (similar to ASP.NET pages).
- User controls cannot run independently; they must be included in an ASP.NET page.
- **Create the User Control:** Create a new file with the .ascx extension in your project.
- **Add Controls and Code:** Add the necessary HTML elements, server controls, and code-behind logic to the user control.
- **Include in a Page:** Use the @Register directive to register the user control in the ASP.NET page where you want to use it.
- **Use the Control:** Use the registered tag name to include the user control in the page markup.

user_control_partial_page.ascx file :

```
<%@ Control Language="C#" AutoEventWireup="true"  
    CodeFile="user_control_partial_page.ascx.cs"  
    Inherits="partial_page_user_control_partial_page" %>  
<%@ OutputCache Duration="20" VaryByParam="None" %>  
<p>  
    Server Time <asp:Label ID="Label1" runat="server"  
        Text="Label"></asp:Label></p>  
<p>  
    Client Time</p>  
    <script>  
        document.write(Date());  
    </script>
```

user_control_partial_page.ascx.cs file :

```
protected void Page_Load(object sender, EventArgs e)  
{  
    Label1.Text = DateTime.Now.ToString();  
}
```

- Add Web_form.aspx.
- Now Drag & Drop UserControlfile.ascx in web_form.aspx in design section :
- `<%@ Register
Src="~/partial_page/user_control_partial_page.ascx"
TagPrefix="uc1" TagName="user_control_partial_page" %>`
- `<uc1:user_control_partial_page
ID="user_control_partial_page1" runat="server" />`
- Here in this file also add server & client side timing to check the cache will be added or not.

Default.aspx

- <%@ Page Language="C#" AutoEventWireup="true" CodeFile="Default.aspx.cs" Inherits="partial_page_Default" %>
- <%@ **Register** Src="~/partial_page/user_control_partial_page.ascx" TagPrefix="uc1" TagName="user_control_partial_page" %>

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

```
<div>
```

```
Server Time : <asp:Label ID="Label1" runat="server"
```

```
Text="Label"></asp:Label>
```

```
Client Time&nbsp;
```

```
<script>
```

```
document.write(Date());
```

```
</script>
```

```
<br />
```

```
<strong>User Control Data</strong>
```

```
<uc1:user_control_partial_page ID="user_control_partial_page1"
```

```
runat="server" />
```

```
</div>
```

```
</form>
```

Default.aspx.cs file

```
protected void Page_Load(object sender, EventArgs e)
{
    Label1.Text =
    DateTime.Now.ToString();
}
```

Cache Expiration Policy :

- ASP.NET cache provides a great feature through which we can increase the performance of the website by reducing the server round trips to the database.
- We can also cache any serializable data into cache memory.
- There are many ways to cache data; one of the simplest way is to use an insert method for storing data in caching.
- After data is stored in Cache we need to periodically validate cache and also set the expiration time for cache as data may be updated on the database.
- So for that we need to define an expiration time after this defined time the cache will expire and then it will again get the new fresh data from the database into the cache. This method is called as Time Base Expiration.
- There are two ways of expiration policy used
 1. Absolute Cache Expiration
 2. Sliding Cache Expiration

Absolute Cache Expiration :

- A **DateTime** object that specifies when the data should be removed from the cache. When you specify "**AbsoluteExpiration**", the cache item will expire at that time, irrespective of whether the cached item is accessed or not.
- **For example**, the following line will cache dataset, ds, for 10 seconds, irrespective of whether the dataset, is accessed or not within those 10 seconds, after it is cached. Since we are using absolute expiration, we specified **Cache.NoSlidingExpiration** for "**SlidingExpiration**" parameter of the Insert() method.
- `Cache.Insert("ProductsData",ds, null, DateTime.Now.AddSeconds(10), System.Web.Caching.Cache.NoSlidingExpiration);`

Sliding Cache Expiration :

- A **TimeSpan** object that identifies how long the data should remain in the cache after the data was last accessed.
- **For example**, the following line will cache dataset, ds, for 10 seconds. If the dataset is accessed from the cache within the specified 10 seconds, then, from that point, the dataset will remain in cache for the next 10 seconds. Since we are using sliding expiration, we specified
- **Cache.NoAbsoluteExpiration** for "**AbsoluteExpiration**" parameter of the Insert() method.
- `Cache.Insert("ProductsData", ds, null, System.Web.Caching.Cache.NoAbsoluteExpiration, TimeSpan.FromSeconds(10));`
- What happens if you specify both, **AbsoluteExpiration** and **SlidingExpiration** when caching application data?
You get a run time exception stating 'absoluteExpiration must be DateTime.MaxValue or slidingExpiration must be TimeSpan.Zero'

Let's Learn What is Dependencies & Expiration & Priority using **Cache.Insert** method :

Cache.Insert(Key, Value, Dependencies, AbsoluteExpiration, SlidingExpiration, Priority)

- **key** [String](#) The cache key used to reference the object.
- **value** [Object](#) The object to be inserted in the cache.
- **dependencies** [CacheDependency](#) The file or cache key dependencies for the item. When any dependency changes, the object becomes invalid and is removed from the cache. If there are no dependencies, this parameter contains null.
- **absoluteExpiration** [DateTime](#) The time at which the inserted object expires and is removed from the cache. To avoid possible issues with local time such as changes from standard time to daylight saving time, use [UtcNow](#) rather than [Now](#) for this parameter value. If you are using absolute expiration, the slidingExpiration parameter must be [NoSlidingExpiration](#).

- **slidingExpiration** [TimeSpan](#) The interval between the time the inserted object was last accessed and the time at which that object expires. If this value is the equivalent of 20 minutes, the object will expire and be removed from the cache 20 minutes after it was last accessed. If you are using sliding expiration, the absoluteExpiration parameter must be [NoAbsoluteExpiration](#).
- **priority** [CacheItemPriority](#) The cost of the object relative to other items stored in the cache, as expressed by the [CacheItemPriority](#) enumeration. This value is used by the cache when it evicts objects; objects with a lower cost are removed from the cache before objects with a higher cost.
- **CacheItemPriority enum values:**
 - CacheItemPriority.Low
 - CacheItemPriority.BelowNormal
 - CacheItemPriority.Normal
 - CacheItemPriority.Default
 - CacheItemPriority.AboveNormal
 - CacheItemPriority.High
 - CacheItemPriority.NotRemovable

Example :

```
con.Open();
    DataSet ds = new DataSet();
    if (Cache["employee"] == null)
    {
        ds = new DataSet();
        SqlDataAdapter da;
        da = new SqlDataAdapter("select * from employee", con);
        da.Fill(ds, "employee");
        Cache.Insert("employee", ds, new
System.Web.Caching.CacheDependency(Server.MapPath("~/con")),
DateTime.Now.AddSeconds(10), TimeSpan.Zero);
    }
    else
    {
        ds = (DataSet)Cache["employee"];
    }
    GridView2.DataSource = ds;
    GridView2.DataBind();
    con.Close();
```

Question for Exam :

1. What is Master Page ?
2. What is Theme ?
3. What is CSS ?
4. Difference between CSS v/s Theme
5. What is Caching?
6. What is OutputCache Directive?
7. What is Register Directive?
8. What is Control Directive?
9. Explain theme elements(css,skin,image).
10. What is skin file?
11. Explain Caching Types in detail.
12. What is output caching ?
13. What is Data caching ?
14. What is Partial Page caching ?
15. Difference between absoluteExpiration v/s slidingExpiration.