**Unit 2: AWP**

1. **List and explain any 5 template to create ASP.NET application.**

Ans: ASP.NET provides several templates for creating web applications, each tailored to specific needs and use cases. Here are five common ASP.NET application templates:

### 1. \*\*Empty\*\*

- \*\*Description\*\*: This template provides the most basic starting point for an ASP.NET application. It includes minimal dependencies and configurations, allowing developers to build a custom application from scratch.

- \*\*Use Case\*\*: Ideal for advanced developers who want complete control over the application structure and prefer to add only the necessary components.

- \*\*Features\*\*:

- No predefined layouts or features.

- Requires manual setup of middleware and services.

### 2. \*\*Web Application (Model-View-Controller, MVC)\*\*

- \*\*Description\*\*: This template implements the MVC design pattern, separating the application into three interconnected components: Model, View, and Controller. It’s structured to support clean separation of concerns and facilitate testing.

- \*\*Use Case\*\*: Best suited for complex web applications requiring robust architecture and easy maintainability.

- \*\*Features\*\*:

- Pre-configured folders for models, views, and controllers.

- Built-in support for Razor syntax for dynamic views.

- Includes basic configurations for routing and dependency injection.

### 3. \*\*Web API\*\*

- \*\*Description\*\*: Designed specifically for building RESTful APIs. It includes essential configurations and libraries to handle HTTP requests and responses.

- \*\*Use Case\*\*: Ideal for creating backend services or APIs that provide data to client applications like mobile apps or single-page applications (SPAs).

- \*\*Features\*\*:

- Pre-configured controllers for handling API requests.

- Built-in support for JSON or XML serialization.

- Swagger/OpenAPI integration for API documentation.

### 4. \*\*Blazor Server App\*\*

- \*\*Description\*\*: This template is used for creating web applications with interactive user interfaces using C# instead of JavaScript. It runs on the server and communicates with the client through SignalR.

- \*\*Use Case\*\*: Suitable for developers who prefer .NET for both client and server development, particularly for internal or intranet applications.

- \*\*Features\*\*:

- Pre-configured with Razor components.

- Real-time updates via SignalR.

- Supports .NET-based UI rendering.

### 5. \*\*Single Page Application (SPA)\*\*

- \*\*Description\*\*: This template is used for building SPAs using frameworks like Angular or React, integrated with an ASP.NET backend. It combines the benefits of client-side frameworks with the power of .NET.

- \*\*Use Case\*\*: Ideal for creating modern, highly interactive web applications with a rich user experience.

- \*\*Features\*\*:

- Pre-configured project setup for Angular, React, or Vue.js.

- Integration with Webpack and Node.js for frontend build processes.

- Includes backend API support.

Each template serves a unique purpose, allowing developers to choose the most appropriate starting point for their specific project requirements.

1. **Explain the anatomy of webform.**

Ans: The anatomy of a **Web Form** in ASP.NET consists of several components and features that work together to create dynamic, interactive web applications. Here's a breakdown:

**1. ASP.NET Web Form Structure**

Web Forms use an **HTML-like structure** combined with server-side controls. A Web Form has two main parts:

* **.aspx file**: Contains the HTML and server-side controls.
* **Code-behind file** (e.g., .aspx.cs or .aspx.vb): Contains the server-side logic, written in C# or VB.NET.

**2. Key Components of a Web Form**

**a) Page Directive**

* Appears at the top of the .aspx file and defines attributes like language, inheritance, and other configurations.
* Example:

asp.net

Copy code

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

**b) Server Controls**

* Special ASP.NET controls that generate HTML and provide server-side functionality.
* Examples:
  + **Label**: <asp:Label ID="lblName" runat="server" Text="Hello!"></asp:Label>
  + **TextBox**: <asp:TextBox ID="txtInput" runat="server"></asp:TextBox>
  + **Button**: <asp:Button ID="btnSubmit" runat="server" Text="Submit" />

**c) HTML Controls**

* Standard HTML elements like <input>, <div>, etc., that can be enhanced with runat="server" to interact with the server-side logic.

**d) ViewState**

* Used to preserve the state of controls between postbacks.
* Stored as a hidden field (\_\_VIEWSTATE) in the page.

**e) Events**

* Server-side event handlers respond to user actions like button clicks.
* Example:

csharp

Copy code

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

{

lblName.Text = "You clicked Submit!";

}

**3. Lifecycle of a Web Form**

The **Page Lifecycle** defines how the Web Form is processed:

1. **Page Request**: The browser requests the page.
2. **Start**: Initialization begins.
3. **Initialization (Init)**: Controls are initialized.
4. **Load**: Page data and ViewState are loaded.
5. **Postback Event Handling**: Events triggered by user interactions are handled.
6. **Rendering**: The page is converted to HTML and sent to the client.
7. **Unload**: Cleanup operations occur.

**4. Code-Behind**

The code-behind file contains the business logic and event-handling methods. Example:

csharp

Copy code

using System;

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

{

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

{

if (!IsPostBack)

{

lblName.Text = "Welcome to ASP.NET!";

}

}

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

{

lblName.Text = "Hello, " + txtInput.Text;

}

}

**5. Configuration**

Web Forms rely on the Web.config file for configuration settings such as:

* Database connections.
* Authentication and authorization.
* Custom settings.

**6. Advantages**

* Rapid development with drag-and-drop controls.
* Rich server-side features like ViewState and event handling.
* Integration with other .NET technologies.

By combining these components, ASP.NET Web Forms provide a structured way to develop scalable, dynamic web applications.

**Explain the 5 properties of listbox and drop down list control.**

Ans: ### 5 Properties of the \*\*ListBox\*\* Control:

1. \*\*Items\*\*:

- Represents the collection of items in the ListBox.

- You can add, remove, or access items in the ListBox using this property.

- \*\*Example\*\*: `ListBox1.Items.Add("Item 1");`

2. \*\*SelectedIndex\*\*:

- Represents the index of the currently selected item.

- If no item is selected, it returns `-1`.

- \*\*Example\*\*: `int selectedIndex = ListBox1.SelectedIndex;`

3. \*\*SelectedValue\*\*:

- Represents the value of the currently selected item (if the ListBox is data-bound).

- Useful when the `Value` property is set on each list item.

- \*\*Example\*\*: `string selectedValue = ListBox1.SelectedValue;`

4. \*\*SelectionMode\*\*:

- Defines how many items can be selected in the ListBox (either `Single`, `Multiple`, or `Extended`).

- \*\*Example\*\*: `ListBox1.SelectionMode = ListSelectionMode.Multiple;`

5. \*\*EnableViewState\*\*:

- Determines whether the control’s state is preserved across postbacks.

- Set to `true` by default.

- \*\*Example\*\*: `ListBox1.EnableViewState = false;`

### 5 Properties of the \*\*DropDownList\*\* Control:

1. \*\*Items\*\*:

- Represents the collection of items in the DropDownList.

- You can add, remove, or access items using this property.

- \*\*Example\*\*: `DropDownList1.Items.Add(new ListItem("Item 1", "1"));`

2. \*\*SelectedIndex\*\*:

- Represents the index of the currently selected item in the DropDownList.

- Returns `-1` if no item is selected.

- \*\*Example\*\*: `int selectedIndex = DropDownList1.SelectedIndex;`

3. \*\*SelectedValue\*\*:

- Represents the value of the selected item (useful for data-bound DropDownLists).

- \*\*Example\*\*: `string selectedValue = DropDownList1.SelectedValue;`

4. \*\*AutoPostBack\*\*:

- Indicates whether the page will automatically post back when a selection is made.

- Set to `false` by default.

- \*\*Example\*\*: `DropDownList1.AutoPostBack = true;`

5. \*\*AppendDataBoundItems\*\*:

- Determines whether new data-bound items should be appended to the existing list of items.

- Set to `false` by default, meaning the DropDownList will overwrite existing items with the new data.

- \*\*Example\*\*: `DropDownList1.AppendDataBoundItems = true;`

Both the \*\*ListBox\*\* and \*\*DropDownList\*\* controls are useful for displaying a list of items in ASP.NET, with the main difference being that the ListBox allows multiple selections, while the DropDownList allows only single selections.

1. **write a short note on adRotator control.**

Ans: The \*\*AdRotator\*\* control in ASP.NET is used to display advertisements or rotating images on a webpage. It allows you to cycle through a set of images (or advertisements) based on a specified rotation logic, either randomly or sequentially. This is commonly used for displaying banner ads or promotional graphics.

### Key Features:

1. \*\*Data Binding\*\*: The AdRotator control requires an XML file to define the advertisement images and their associated URLs. This XML file contains the information about each advertisement, including its image file, a hyperlink to navigate when the ad is clicked, and the ad's weight (to control the frequency of display).

2. \*\*Key Properties\*\*:

- `AdvertisementFile`: Specifies the path to the XML file that contains the ad details.

- `ImageUrl`: Defines the image URL of the current advertisement being displayed.

- `NavigateUrl`: Defines the URL that the user is redirected to when they click the advertisement.

3. \*\*Ad Rotation\*\*: The control can rotate ads in a random or sequential order, based on the settings in the XML file, which can include the ad's weight for controlling frequency.

### Example of Usage:

```xml

<asp:AdRotator ID="AdRotator1" runat="server" AdvertisementFile="ads.xml" />

```

### Example XML (ads.xml):

```xml

<ads>

<ad>

<imageurl>images/ad1.jpg</imageurl>

<navigateurl>http://www.example1.com</navigateurl>

<alttext>Ad 1</alttext>

<weight>3</weight>

</ad>

<ad>

<imageurl>images/ad2.jpg</imageurl>

<navigateurl>http://www.example2.com</navigateurl>

<alttext>Ad 2</alttext>

<weight>2</weight>

</ad>

</ads>

```

The AdRotator control is useful for easily managing and displaying rotating advertisements in ASP.NET applications without requiring custom code for each ad display.

1. **what is postback? Explain isPostback property with the example.**

**Ans:** ### What is Postback?

In ASP.NET, a \*\*postback\*\* refers to the process of submitting a form on a webpage to the server for processing, causing the page to reload or refresh. When a user interacts with a control (e.g., clicking a button or selecting a value from a dropdown), the page is sent back to the server, where the server processes the request, re-executes the page lifecycle, and generates a new HTML page to be sent to the client (browser).

### \*\*isPostBack\*\* Property

The `IsPostBack` property is a built-in property in ASP.NET that helps you determine whether the page is being loaded as a result of a postback (i.e., a request after form submission) or a first-time page load. This property is useful to avoid re-initializing data that doesn't need to be reset when the page is refreshed.

- \*\*True\*\*: If `IsPostBack` is `true`, it means that the page is being loaded in response to a postback (i.e., a user action).

- \*\*False\*\*: If `IsPostBack` is `false`, it means that the page is being loaded for the first time.

### Why Use `IsPostBack`?

`IsPostBack` is typically used to prevent redundant code execution, such as re-binding data to controls or resetting values during every postback. For example, you can initialize controls (e.g., dropdown lists, grids) only during the first load and not on subsequent postbacks, which improves performance.

### Example of Using `IsPostBack`

Here is a simple example that demonstrates how to use the `IsPostBack` property:

#### ASPX Page (Default.aspx):

```html

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

<asp:Label ID="lblMessage" runat="server" />

```

#### Code-Behind (Default.aspx.cs):

```csharp

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

{

// Check if the page is being loaded for the first time (not postback)

if (!IsPostBack)

{

// Initialize controls only during the first load

lblMessage.Text = "Welcome to the page! Please click Submit.";

}

}

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

{

// Perform some action when the button is clicked

lblMessage.Text = "Thank you for submitting!";

}

```

### Explanation:

- \*\*Page\_Load\*\*: During the first page load (`IsPostBack == false`), we initialize the `lblMessage` label with a welcome message.

- \*\*Button Click\*\*: When the button is clicked, the page makes a postback. The message in the label changes to "Thank you for submitting!" on the subsequent request, but the page's controls (like the label) will not be re-initialized unless explicitly coded.

The `IsPostBack` property ensures that initialization code only runs when necessary, saving time and resources, especially when dealing with data binding or other expensive operations.

1. **List and describe the various file type in an Asp.net framework.**

Ans: Here are 10 important and commonly used file types in ASP.NET :

### 1. \*\*.aspx (Web Form)\*\*

- Defines the UI structure of a web page using HTML, CSS, and ASP.NET server controls. It serves as the main file for web pages in ASP.NET Web Forms applications.

### 2. \*\*.aspx.cs / .aspx.vb (Code-Behind File)\*\*

- Contains the server-side code for handling events and application logic for the corresponding `.aspx` page. It separates the UI from the application logic.

### 3. \*\*.ascx (User Control)\*\*

- Defines reusable UI components that can be used across multiple pages. It helps to modularize the application and avoid redundant code.

### 4. \*\*.ascx.cs / .ascx.vb (User Control Code-Behind)\*\*

- Contains the server-side logic for handling events and application logic specific to the user control defined in `.ascx` files.

### 5. \*\*.config (Configuration Files)\*\*

- Stores configuration settings for the application, such as database connections, authentication settings, and application-specific configurations.

### 6. \*\*.cshtml / .vbhtml (Razor View Files)\*\*

- Used in ASP.NET MVC applications to render dynamic content. Razor syntax allows embedding server-side code (C# or VB.NET) within HTML to generate dynamic views.

### 7. \*\*.cs / .vb (Class Files)\*\*

- Contains business logic, classes, methods, and models in C# or VB.NET, providing the functionality needed for the application’s operations and data handling.

### 8. \*\*.ashx (Generic Handler)\*\*

- Defines HTTP handlers that handle custom HTTP requests, such as returning dynamic content like images or processing AJAX requests, without requiring a full-page lifecycle.

### 9. \*\*.dll (Dynamic Link Library)\*\*

- Compiled assemblies that contain reusable code (e.g., helper methods, business logic, components) used by the application. These libraries can be referenced by multiple projects or components.

### 10. \*\*.js (JavaScript File)\*\*

- Contains client-side JavaScript code that adds interactivity to web pages. It is used for tasks like form validation, handling events, and making AJAX calls to the server.

These file types are essential for building and maintaining a well-structured ASP.NET web application, ensuring separation of concerns and ease of management.

1. **what is the purpose of validation control ? List and explain the validation control available in Asp.net. vo required field wala bhi karlena.practical wala.**

Ans: ### Purpose of Validation Controls in ASP.NET

Validation controls in ASP.NET are used to ensure that user input meets specific criteria before it is processed or saved to a database. These controls help to enforce data integrity and improve the user experience by providing immediate feedback when the input is invalid. Validation controls help in performing client-side and server-side validation, ensuring that data entered by users is accurate, consistent, and within the required format.

### Types of Validation Controls in ASP.NET

1. \*\*RequiredFieldValidator\*\*

- \*\*Purpose\*\*: Ensures that the user does not leave a required input field empty.

- \*\*Attributes\*\*:

- `ControlToValidate`: The ID of the control to validate.

- `ErrorMessage`: The message to display if validation fails.

- \*\*Example\*\*: Validates if a text box is filled with some data.

2. \*\*RangeValidator\*\*

- \*\*Purpose\*\*: Checks whether the user input is within a specified range of values (e.g., numbers, dates).

- \*\*Attributes\*\*:

- `ControlToValidate`: The ID of the control to validate.

- `MinimumValue` and `MaximumValue`: Defines the acceptable range for the input.

- `Type`: Specifies the type of data (e.g., `Integer`, `Date`).

- `ErrorMessage`: Message to display if input is out of range.

- \*\*Example\*\*: Validates if the entered age is between 18 and 100.

3. \*\*CompareValidator\*\*

- \*\*Purpose\*\*: Compares the values of two controls (e.g., checking if two password fields match).

- \*\*Attributes\*\*:

- `ControlToValidate`: The ID of the control to validate.

- `ControlToCompare`: The ID of the control to compare with.

- `Operator`: Defines the type of comparison (e.g., `Equal`, `NotEqual`).

- `ErrorMessage`: Message to display if values do not match.

- \*\*Example\*\*: Validates if two password fields contain the same value.

4. \*\*RegularExpressionValidator\*\*

- \*\*Purpose\*\*: Validates input using a regular expression to ensure it follows a specific pattern (e.g., email format, phone number format).

- \*\*Attributes\*\*:

- `ControlToValidate`: The ID of the control to validate.

- `ValidationExpression`: The regular expression pattern to validate the input.

- `ErrorMessage`: Message to display if the input does not match the pattern.

- \*\*Example\*\*: Validates if the entered email is in a correct format (e.g., `user@example.com`).

5. \*\*CustomValidator\*\*

- \*\*Purpose\*\*: Allows custom validation logic, where you can define a custom function to validate user input.

- \*\*Attributes\*\*:

- `ControlToValidate`: The ID of the control to validate.

- `ValidationFunction`: A JavaScript function for client-side validation or a server-side function.

- `ErrorMessage`: Message to display if the input fails the custom validation.

- \*\*Example\*\*: Validates based on complex business rules or custom logic.

6. \*\*ValidationSummary\*\*

- \*\*Purpose\*\*: Displays a summary of all validation errors in a single location on the page.

- \*\*Attributes\*\*:

- `ShowMessageBox`: If `true`, displays the validation summary in a message box.

- `ShowSummary`: If `true`, displays the validation summary as a list.

- `HeaderText`: The header text to display before the validation summary.

- \*\*Example\*\*: Displays all validation error messages in a single area instead of showing messages next to individual controls.

7. \*\*LowerBoundValidator\*\* \*(Custom implementation)\*

- \*\*Purpose\*\*: This control checks whether a numeric input is greater than or equal to a specified lower bound.

- \*\*Attributes\*\*:

- `ControlToValidate`: The ID of the control to validate.

- `MinimumValue`: The minimum value required.

- `ErrorMessage`: The message to display if the validation fails.

- \*\*Example\*\*: Ensures that an entered number is at least 18.

8. \*\*FileUploadValidator\*\* \*(Custom implementation)\*

- \*\*Purpose\*\*: Ensures that a file uploaded meets specific criteria, such as file size or type.

- \*\*Attributes\*\*:

- `ControlToValidate`: The ID of the file upload control.

- `FileType`: Defines valid file types (e.g., `jpg`, `pdf`).

- `MaxFileSize`: The maximum allowed file size.

- `ErrorMessage`: The message to display if the validation fails.

- \*\*Example\*\*: Validates that the uploaded file is a PDF file and not larger than 5 MB.

### Summary:

Validation controls in ASP.NET play a crucial role in ensuring data quality and consistency by checking user input against specified criteria. The most commonly used validation controls include the \*\*RequiredFieldValidator\*\*, \*\*RangeValidator\*\*, \*\*CompareValidator\*\*, \*\*RegularExpressionValidator\*\*, and \*\*CustomValidator\*\*, each of which serves a specific purpose in different validation scenarios. Using these controls helps in building more reliable and secure web applications by preventing invalid data from being submitted.

1. **what is view state ? Give advantage and disadvantage of view state.**

Ans: ### What is ViewState?

**ViewState** is a mechanism in ASP.NET used to preserve the state of web form controls between postbacks. Since web applications are stateless by nature, ViewState allows ASP.NET to maintain control values across multiple requests for the same page.

* **How it works**:  
  ViewState stores the data in a hidden field (\_\_VIEWSTATE) on the client-side in the form of a Base64-encoded string. This data is sent to the server with every request and returned to the client with the response.
* **Example**:

html

Copy code

<input type="hidden" name="\_\_VIEWSTATE" value="dDw1Mz..." />

**Advantages of ViewState**

1. **Simple to Use**
   * Automatically enabled for all controls.
   * No additional code is required to maintain state.
2. **Server Independence**
   * ViewState stores data on the client-side, reducing the server's memory and processing load.
3. **Control Persistence**
   * Ensures that form values (e.g., text in text boxes, selected items in dropdowns) persist between postbacks.
4. **No Server Configuration Required**
   * Works out of the box without requiring additional infrastructure or setup.
5. **Enhanced User Experience**
   * Allows for dynamic updates without resetting form inputs, improving user experience.

**Disadvantages of ViewState**

1. **Increased Page Size**
   * ViewState data is included in every page request and response, which can increase page size significantly, leading to slower performance for large forms.
2. **Security Risks**
   * Although ViewState can be encrypted, by default, it is Base64-encoded, which is not secure. Sensitive data should never be stored in ViewState.
3. **Performance Overhead**
   * Encoding and decoding ViewState data adds processing time on both the client and server.
4. **Limited Scope**
   * ViewState only works for a single page. Data does not persist across different pages in the application.
5. **Complex Debugging**
   * The Base64-encoded data can make debugging state issues challenging.

**Best Practices**

* **Use ViewState selectively**: Disable ViewState (EnableViewState="false") for controls that do not require state persistence.
* **Optimize ViewState size**: Avoid storing large amounts of data.
* **Secure ViewState**: Use encryption by setting the ViewStateEncryptionMode property to Always.

ViewState is a powerful tool but should be used judiciously to balance functionality and performance.

1. **explain the need of user control? How it is created and used?**

Ans: ### \*\*Need for User Controls in ASP.NET\*\*

User controls in ASP.NET are essential for creating modular and reusable components, which are vital in building scalable and maintainable web applications. They allow developers to bundle UI elements and their corresponding functionality into a single control that can be reused across multiple pages. This leads to several benefits:

- \*\*Code Reusability\*\*: Once created, user controls can be reused in multiple pages, reducing redundancy.

- \*\*Modular Development\*\*: Complex pages can be broken into smaller, manageable units, making the application easier to maintain and scale.

- \*\*Consistency\*\*: User controls ensure consistent design and behavior across the application by reusing the same component.

- \*\*Maintainability\*\*: When changes are made to a user control, they automatically reflect across all pages that use it, reducing maintenance effort.

### \*\*Creating and Using User Controls\*\*

1. \*\*Creating a User Control\*\*:

- \*\*Add a New User Control\*\*: In Visual Studio, add a new file with the `.ascx` extension (e.g., `MyControl.ascx`).

- \*\*Design the UI\*\*: Inside the `.ascx` file, define the UI using standard HTML or ASP.NET controls.

- \*\*Write the Logic\*\*: Add the logic in the code-behind file (`MyControl.ascx.cs`), which handles events and any server-side processing.

Example (`MyControl.ascx`):

```html

<%@ Control Language="C#" AutoEventWireup="true" CodeBehind="MyControl.ascx.cs" Inherits="WebApp.MyControl" %>

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

<asp:Button ID="Button1" runat="server" Text="Click Me" OnClick="Button1\_Click" />

```

2. \*\*Using the User Control\*\*:

- \*\*Register the Control\*\*: Use the `@Register` directive in the `.aspx` page to register the user control.

```html

<%@ Register Src="~/MyControl.ascx" TagPrefix="uc" TagName="MyControl" %>

```

- \*\*Add the Control to the Page\*\*: After registering, use the user control on the page by adding its tag.

```html

<uc:MyControl ID="MyUserControl" runat="server" />

```

3. \*\*Interacting with the Control\*\*:

You can access the user control programmatically in the `.aspx` page’s code-behind file using the control’s `ID` and call its methods or properties if needed.

Example:

```csharp

MyControl control = (MyControl)FindControl("MyUserControl");

control.UpdateText("New Text");

```

### \*\*Advantages of User Controls\*\*

- \*\*Code Reusability\*\*: Once a user control is created, it can be reused on multiple pages, reducing redundant code and design.

- \*\*Modular Design\*\*: Complex pages are easier to manage by breaking them into smaller, self-contained components.

- \*\*Consistency\*\*: Ensures that the UI components behave and look the same across different pages.

- \*\*Centralized Maintenance\*\*: Any changes made to the user control automatically reflect on all pages using that control, simplifying maintenance.

- \*\*Separation of Concerns\*\*: It allows for better separation between logic (code-behind) and presentation (UI), which makes the code more maintainable and testable.

### \*\*Conclusion\*\*

User controls in ASP.NET are a powerful tool for creating reusable, modular, and maintainable components that improve consistency and simplify the management of web applications. By creating self-contained units of functionality, they reduce duplication, improve maintainability, and enhance scalability.

1. **what is an event ? How is an event handler added. (with application )**

Ans: ### \*\*What is an Event?\*\*

An \*\*event\*\* in ASP.NET (and other object-oriented programming languages) is an action or occurrence that the program can respond to, usually triggered by user actions or system-generated processes. Events are used to notify other components that something has happened, such as a button being clicked, a page being loaded, or data being submitted.

In ASP.NET, events are commonly used in the context of server controls (e.g., buttons, textboxes, dropdowns) to trigger server-side logic in response to user interactions on a webpage.

\*\*Example of common events:\*\*

- `Button\_Click`: Triggered when a user clicks a button.

- `TextChanged`: Triggered when the content of a text box changes.

- `Page\_Load`: Triggered when the page is loaded.

### \*\*How is an Event Handler Added?\*\*

An \*\*event handler\*\* is a method that defines the actions to be performed when a specific event occurs. Event handlers are added by associating an event with a method in the code-behind file of the page.

There are two main ways to attach an event handler:

1. \*\*Declarative Syntax\*\* (in the `.aspx` markup):

In the markup, you associate the control's event with a method defined in the code-behind.

Example:

```html

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

```

In the above example:

- The `OnClick` attribute specifies that the `btnSubmit\_Click` method in the code-behind should be called when the button is clicked.

2. \*\*Programmatic Syntax\*\* (in the code-behind):

You can also attach event handlers programmatically in the `Page\_Load` or other lifecycle events.

Example (in the `.aspx.cs` file):

```csharp

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

{

btnSubmit.Click += new EventHandler(btnSubmit\_Click);

}

```

Here, the `Click` event of the `btnSubmit` button is manually linked to the `btnSubmit\_Click` method.

### \*\*How an Event Works in ASP.NET\*\*

1. \*\*Control triggers an event\*\*: An event is raised when a user interacts with a control (e.g., clicking a button, selecting an item in a dropdown).

2. \*\*Event handler is invoked\*\*: The associated event handler method is invoked to handle the event and perform necessary actions, such as updating a database or changing a UI element.

3. \*\*Page lifecycle continues\*\*: After the event handler executes, the page continues through the lifecycle (e.g., rendering the updated page).

### \*\*Example of Event and Event Handler\*\*

1. \*\*Markup\*\* (`.aspx`):

```html

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

```

2. \*\*Code-Behind\*\* (`.aspx.cs`):

```csharp

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

{

// Handle button click event

lblMessage.Text = "Button clicked!";

}

```

In this example:

- When the user clicks the `btnSubmit` button, the `btnSubmit\_Click` method is invoked, and the text of the `lblMessage` label is updated.

### \*\*Applications of Events in ASP.NET\*\*

1. \*\*User Interactions\*\*: Events are used to capture user actions like button clicks, text changes, and selections in controls such as buttons, textboxes, and dropdowns.

2. \*\*Page Lifecycle\*\*: ASP.NET pages have their own events, such as `Page\_Load`, `Page\_PreRender`, `Page\_Init`, which can be handled in the code-behind to manage page behavior during the page's lifecycle.

3. \*\*Data Binding\*\*: Events can trigger data binding operations in controls such as `GridView` and `ListView`.

4. \*\*State Management\*\*: Events like `ItemCommand` in controls like `GridView` can be used to handle user actions such as updating or deleting data.

5. \*\*Custom Controls\*\*: Events are also essential in creating custom user controls or server-side logic that interacts with a web page.

### \*\*Summary\*\*

An \*\*event\*\* is an action or occurrence that can trigger a specific response in an ASP.NET web page. Event handlers are methods that respond to these events. You can add event handlers either declaratively (through markup) or programmatically (in the code-behind). Events are crucial in handling user interactions and defining the behavior of controls on a web page.

Answer Jayda he but programing he toh honjayega