**Asp.Net II Mid Term 2018 Solution B**

Q.1.1 ASP.NET validation controls validate the user input data to ensure that useless, unauthenticated, or contradictory data don't get stored.

ASP.NET provides the following validation controls:

* RequiredFieldValidator
* RangeValidator
* CompareValidator
* RegularExpressionValidator
* CustomValidator
* ValidationSummary

## RequiredFieldValidator Control

The RequiredFieldValidator control ensures that the required field is not empty. It is generally tied to a text box to force input into the text box.

The syntax of the control is as given:

<asp:RequiredFieldValidator ID="rfvcandidate"

 runat="server" ControlToValidate ="ddlcandidate"

 ErrorMessage="Please choose a candidate"

 InitialValue="Please choose a candidate">

</asp:RequiredFieldValidator>

## RangeValidator Control

The RangeValidator control verifies that the input value falls within a predetermined range.

The syntax of the control is as given:

<asp:RangeValidator ID="rvclass" runat="server" ControlToValidate="txtclass"

 ErrorMessage="Enter your class (6 - 12)" MaximumValue="12"

 MinimumValue="6" Type="Integer">

</asp:RangeValidator>

2 The Panel Control is a container control to host a group of similar child controls. One of the major uses I have found for a Panel Control is when you need to show and hide a group of controls. Instead of show and hide individual controls, you can simply hide and show a single Panel and all child controls.

In this article, we will demonstrate how to create and use a Panel Control in a Windows Forms application.

**Creating a Panel**

We can create a Panel Control using the Forms designer at design-time or using the Panel class in code at run-time.

**Design-time**

To create a Panel Control at design-time, you can drag and drop a Panel Control from the Toolbox to a Form in Visual Studio. After you dragging and dropping a Panel Control to the Form, the Panel looks

**Run-time**

Creating a Panel Control at run-time is merely a work of creating an instance of the Panel class, setting its properties and adding the Panel to the form controls.

The first step to create a dynamic Panel is to create an instance of the Panel class. The following code snippet creates a Panel Control object.

Panel dynamicPanel = new Panel();

In the next step, you may set the properties of a Panel Control.

The following code snippet sets the location, size and Name properties of a Panel.

dynamicPanel.Location = new System.Drawing.Point(26, 12);

dynamicPanel.Name = "Panel1";

dynamicPanel.Size = new System.Drawing.Size(228, 200);

dynamicPanel.TabIndex = 0;

Once the Panel Control is ready with its properties, the next step is to add the Panel to a form so it becomes a part of the form.

To do so, we use the "Form.Controls.Add" method that adds the Panel Control to the form's controls and displays it on the form based on the location and size of the control.

The following code snippet adds a Panel Control to the current form.

Controls.Add(dynamicPanel);

3. Difference between Security authentication and authorization

|  |  |
| --- | --- |
| **Authentication** | **Authorization** |
| Authentication confirms your identity to grant access to the system. | Authorization determines whether you are authorized to access the resources. |
| It is the process of validating user credentials to gain user access. | It is the process of verifying whether access is allowed or not. |
| It determines whether user is what he claims to be. | It determines what user can and cannot access. |
| Authentication usually requires a username and a password. | Authentication factors required for authorization may vary, depending on the security level. |
| Authentication is the first step of authorization so always comes first. | Authorization is done after successful authentication. |
| For example, students of a particular university are required to authenticate themselves before accessing the student link of the university’s official website. This is called authentication. | For example, authorization determines exactly what information the students are authorized to access on the university website after successful authentication. |

# 4. ASP.NET Dataset

DataSet provides a disconnected representation of result sets from the Data Source, and it is completely independent from the Data Source.



The Dataset can work with the data without knowing the source of the data coming from. DataSet provides much greater flexibility when dealing with related Result Sets. It gives a better advantage over DataReader , because the DataReader is working only with the connection oriented Data Sources.



A DataSet is a container for one or more DataTable objects that contain the data you retrieve from the database. We can set up Data Relations between these tables within the DataSet. The DataAdapter Object allows us to populate DataTables in a DataSet. We can use Fill method of the DataAdapter for populating data in a Dataset. The DataSet can be filled either from a data source or dynamically. From the following chapters you can learn more about Dataset and its operations.

5. A web service is a web application which is basically a class consisting of methods that could be used by other applications. It also follows a code-behind architecture such as the ASP.NET web pages, although it does not have a user interface.

To understand the concept let us create a web service to provide stock price information. The clients can query about the name and price of a stock based on the stock symbol. To keep this example simple, the values are hardcoded in a two-dimensional array. This web service has three methods:

* A default HelloWorld method
* A GetName Method
* A GetPrice Method

Take the following steps to create the web service:

**Step (1)** : Select File -> New -> Web Site in Visual Studio, and then select ASP.NET Web Service.

**Step (2)** : A web service file called Service.asmx and its code behind file, Service.cs is created in the App\_Code directory of the project.

**Step (3)** : Change the names of the files to StockService.asmx and StockService.cs.

**Step (4)** : The .asmx file has simply a WebService directive on it:

<%@ WebService Language="C#" CodeBehind="~/App\_Code/StockService.cs" Class="StockService" %>

## Q.2 Master Pages

Master pages allow you to create a consistent look and behavior for all the pages (or group of pages) in your web application.

A master page provides a template for other pages, with shared layout and functionality. The master page defines placeholders for the content, which can be overridden by content pages. The output result is a combination of the master page and the content page.

The content pages contain the content you want to display.

When users request the content page, ASP.NET merges the pages to produce output that combines the layout of the master page with the content of the content page.

## Master Page Example

<%@ Master %>

<html>
<body>
<h1>Standard Header From Masterpage</h1>
<asp:ContentPlaceHolder id="CPH1" runat="server">
</asp:ContentPlaceHolder>
</body>
</html>

The master page above is a normal HTML page designed as a template for other pages.

The **@ Master** directive defines it as a master page.

The master page contains a placeholder tag **<asp:ContentPlaceHolder>** for individual content.

The **id="CPH1"** attribute identifies the placeholder, allowing many placeholders in the same master page.

This master page was saved with the name **"master1.master".**

|  |  |
| --- | --- |
|  | **Note:** The master page can also contain code, allowing dynamic content. |

## Content Page Example

<%@ Page MasterPageFile="master1.master" %>

<asp:Content ContentPlaceHolderId="CPH1" runat="server">
  <h2>Individual Content</h2>
  <p>Paragraph 1</p>
  <p>Paragraph 2</p>
</asp:Content>

The content page above is one of the individual content pages of the web.

The **@ Page** directive defines it as a standard content page.

The content page contains a content tag **<asp:Content>** with a reference to the master page (ContentPlaceHolderId="CPH1").

This content page was saved with the name **"mypage1.aspx"**.

When the user requests this page, ASP.NET merges the content page with the master page.

## Content Page With Controls

<%@ Page MasterPageFile="master1.master" %>

<asp:Content ContentPlaceHolderId="CPH1" runat="server">
  <h2>W3Schools</h2>
  <form runat="server">
    <asp:TextBox id="textbox1" runat="server" />
    <asp:Button id="button1" runat="server" text="Button" />
  </form>
</asp:Content>

2.2 One of the important goals of the **Asp.net** language is code minimization. Data controls play an important role in this purpose. Data controls used to display the records in the form of reports.

**Some of the Data controls are:**

1. Repeater data control
2. Detailsview data control
3. GridView data control
4. DataList data control
5. FormView data control

**GridView data control:**

This control displays the data in tabular form.

It not only support the editing and deleting of data, it also support sorting and paging of data.

**Example:**



**DetailsView data control:**

This control displays single record of data at a time.

It supports updating, insertion or deletion of the record.

It can be used in conjunction with the GridView control to provide a master-detail view of your data.

**Example:**



**FormView data control:**

This control displays a single record of data at a time like DetailsView control and supports the editing of record.

This control requires the use of template to define the rendering of each item.

Developer can completely customize the appearance of the record.

**Example:**



**DataListView control:**

This control displays data as items in a list.

Presentation of the data can be customized via templates.

Inline editing and deleting of data is supported by this control.

**Example:**



**Repeater Data control:**

This control is used to display the repeated list of the items that are bound to the control.

It provides complete flexibility in regards to the HTML presentation.

A Repeater has five templates to format it:

* HeaderTemplate
* AlternatingItemTemplate
* Itemtemplate
* SeoperatorTemplate
* FooterTemplate

**Example**



3.1 Before proceeding ahead we need to understand four important vocabularies which you will see in this article again and again: - authentication, authorization, principal and identity. Let’s first start with authentication and authorization. If you search in www.google.com for the dictionary meaning of authentication and authorization, you will land up with something below:-

**Authentication: -** prove genuineness

**Authorization: -** process of granting approval or permission on resources.

The same dictionary meaning applies to ASP.NET as well. In ASP.NET authentication means to identify the user or in other words its nothing but to validate that he exists in your database and he is the proper user.
Authorization means does he have access to a particular resource on the IIS website. A resource can be an ASP.NET web page, media files (MP4, GIF, JPEG etc), compressed file (ZIP, RAR) etc.
So the first process which happens is authentication and then authorization. Below is a simple graphical representation of authentication and authorization. So when the user enters ‘userid’ and ‘password’ he is first authenticated and identified by the user name.
Now when the user starts accessing resources like pages, ASPDOTNETauthentication, videos etc, he is checked whether he has the necessary access for the resources. The process of identifying the rights for resources is termed as ‘Authorization’.



To put it in simple words to identify “he is shiv” is authentication and to identify that “Shiv is admin” is authorization.

## Detecting authentication and authorization: - The principal and identity objects

At any moment of time if you want to know who the user is and what kind of authentication type he using you can use the identity object. If you want to know what kind of roles it’s associated with then we need to use the principal object. In other words to get authentication details we need to the identity object and to know about authorization details of that identity we need the principal object.



For instance below is a simple sample code which shows how to use identity and principal object to display name and check roles.

Response.Write(User.Identity.Name +"<br>");

Response.Write(User.Identity.AuthenticationType + "<br>");

Response.Write(User.Identity.IsAuthenticated + "<br>");

Response.Write(User.IsInRole("Administrators") + "<br>");

## Types of authentication and authorization in ASP.NET

There are three ways of doing authentication and authorization in ASP.NET:-
• **Windows authentication: -** In this methodology ASP.NET web pages will use local windows users and groups to authenticate and authorize resources.

• **Forms Authentication: -** This is a cookie based authentication where username and password are stored on client machines as cookie files or they are sent through URL for every request. Form-based authentication presents the user with an HTML-based Web page that prompts the user for credentials.

**• Passport authentication :-** Passport authentication is based on the passport website provided
by the Microsoft .So when user logins with credentials it will be reached to the passport website ( i.e. hotmail,devhood,windows live etc) where authentication will happen. If Authentication is successful it will return a token to your website.

• **Anonymous access: -** If you do not want any kind of authentication then you will go for Anonymous access.

<authentication mode="Windows"/>

We also need to ensure that all users are denied except authorized users. The below code snippet inside the authorization tag that all users are denied. ‘?’ indicates any

Hide   Copy Code

unknown user.

<authorization>

<deny users="?"/>

</authorization>

**Step 4:- Setup authorization**
We also need to specify the authorization part. We need to insert the below snippet in the ‘web.config’ file stating that only ‘Administrator’ users will have access to

Hide   Copy Code

‘Admin.aspx’ pages.

<location path="Admin.aspx">

<system.web>

<authorization>

<allow roles="questpon-srize2\Administrator"/>

<deny users="\*"/>

</authorization>

</system.web>

</location>

3.2 **ADO.NET**

ADO.NET is a set of classes (a framework) to interact with data sources such as databases and XML files. ADO is the acronym for ActiveX Data Objects. It allows us to connect to underlying data or databases. It has classes and methods to retrieve and manipulate data.

The following are a few of the .NET applications that use ADO.NET to connect to a database, execute commands and retrieve data from the database.

* ASP.NET Web Applications
* Console Applications
* Windows Applications.

**Various Connection Architectures**

There are the following two types of connection architectures:

1. **Connected architecture:** the application remains connected with the database throughout the processing.
2. **Disconnected architecture:** the application automatically connects/disconnects during the processing. The application uses temporary data on the application side called a DataSet.

**Understanding ADO.NET and its class library

**

In this diagram, we can see that there are various types of applications (Web Application, Console Application, Windows Application and so on) that use ADO.NET to connect to databases (SQL Server, Oracle, OleDb, ODBC, XML files and so on).

**Important Classes in ADO.NET**
We can also observe various classes in the preceding diagram. They are:

1. Connection Class
2. Command Class
3. DataReader Class
4. DataAdaptor Class
5. DataSet.Class

**1. Connection Class**In ADO.NET, we use these connection classes to connect to the database. These connection classes also manage transactions and connection pooling.

**2. Command Class**
The Command class provides methods for storing and executing SQL statements and Stored Procedures. The following are the various commands that are executed by the Command Class.

* **ExecuteReader:** Returns data to the client as rows. This would typically be an SQL select statement or a Stored Procedure that contains one or more select statements. This method returns a DataReader object that can be used to fill a DataTable object or used directly for printing reports and so forth.
* **ExecuteNonQuery:** Executes a command that changes the data in the database, such as an update, delete, or insert statement, or a Stored Procedure that contains one or more of these statements. This method returns an integer that is the number of rows affected by the query.
* **ExecuteScalar:** This method only returns a single value. This kind of query returns a count of rows or a calculated value.
* **ExecuteXMLReader:** (SqlClient classes only) Obtains data from an SQL Server 2000 database using an XML stream. Returns an XML Reader object.

**3. DataReader Class**The DataReader is used to retrieve data. It is used in conjunction with the Command class to execute an SQL Select statement and then access the returned rows.

**4. DataAdapter Class**
The DataAdapter is used to connect DataSets to databases. The DataAdapter is most useful when using data-bound controls in Windows Forms, but it can also be used to provide an easy way to manage the connection between your application and the underlying database tables, views and Stored Procedures.

**5. DataSet Class**

The DataSet is the heart of ADO.NET. The DataSet is essentially a collection of DataTable objects. In turn each object contains a collection of DataColumn and DataRow objects. The DataSet also contains a Relations collection that can be used to define relations among Data Table Objects.

**How to Connect to a Database using ADO.NET**
Now let us learn how to connect to a database using ADO.NET. To create a connection, you must be familiar with connection strings. A connection string is required as a parameter to SQLConnection. A ConnectionString is a string variable (not case sensitive).

This contains key and value pairs, like provider, server, database, userid and word as in the following:

Server="nameof the server or IP Address of the server"

Database="name of the database"

userid="user name who has permission to work with database"

word="the word of userid"

**Example**

**SQL Authentication**String constr="server=.;database=institute;user id=rakesh;word=abc@123";

Or:

String constr="data source=.;initial catalog=institute;uid=rakesh;pwd=abc@213";

**Windows Authentication**

String constr="server=.;database=institute;trusted\_connection=true"

Or:

String constr="server=.;initial catalog=institute;integrated security=true"

**How to retrieve and display data from a database**
Procedure:

1. Create a SqlConnection object using a connection string.
2. Handle exceptions.
3. Open the connection.
4. Create a SQLCommand. To represent a SQLCommand like (select \* from studentdetails) and attach the existing connection to it. Specify the type of SQLCommand (text/storedprocedure).
5. Execute the command (use executereader).
6. Get the Result (use SqlDataReader). This is a forwardonly/readonly dataobject.
7. Close the connection
8. Process the result
9. Display the Result

The following is code for connecting to a SQL Database:



You must use the System.Data.SqlClient namespace to connect to a SQL Database. In the preceding code we are using the SqlConnection class, SqlCommand class and SqlDataReader class because our application is talking to SQL Server. SQL Server only understands SQL.

**Code for connecting to Oracle Database**

If you want to connect to an Oracle database, all you need to do is to change the connection class name from SqlConnection to **OracleConnection** Command class name from SqlCommand to OracleCommand and **SqlDataReader** to **OracleDataReader** and also in the beginning use the namespace **System.Data.OralceClient.**

## 4.1 ASP.NET Page Life Cycle

When a page is requested, it is loaded into the server memory, processed, and sent to the browser. Then it is unloaded from the memory. At each of these steps, methods and events are available, which could be overridden according to the need of the application. In other words, you can write your own code to override the default code.

The Page class creates a hierarchical tree of all the controls on the page. All the components on the page, except the directives, are part of this control tree. You can see the control tree by adding trace= "true" to the page directive. We will cover page directives and tracing under 'directives' and 'event handling'.

The page life cycle phases are:

* Initialization
* Instantiation of the controls on the page
* Restoration and maintenance of the state
* Execution of the event handler codes
* Page rendering

Understanding the page cycle helps in writing codes for making some specific thing happen at any stage of the page life cycle. It also helps in writing custom controls and initializing them at right time, populate their properties with view-state data and run control behavior code.

Following are the different stages of an ASP.NET page:

* **Page request** - When ASP.NET gets a page request, it decides whether to parse and compile the page, or there would be a cached version of the page; accordingly the response is sent.
* **Starting of page life cycle** - At this stage, the Request and Response objects are set. If the request is an old request or post back, the IsPostBack property of the page is set to true. The UICulture property of the page is also set.
* **Page initialization** - At this stage, the controls on the page are assigned unique ID by setting the UniqueID property and the themes are applied. For a new request, postback data is loaded and the control properties are restored to the view-state values.
* **Page load** - At this stage, control properties are set using the view state and control state values.
* **Validation** - Validate method of the validation control is called and on its successful execution, the IsValid property of the page is set to true.
* **Postback event handling** - If the request is a postback (old request), the related event handler is invoked.
* **Page rendering** - At this stage, view state for the page and all controls are saved. The page calls the Render method for each control and the output of rendering is written to the OutputStream class of the Response property of page.
* **Unload** - The rendered page is sent to the client and page properties, such as Response and Request, are unloaded and all cleanup done.

**ASP.NET Page Life Cycle Events**

At each stage of the page life cycle, the page raises some events, which could be coded. An event handler is basically a function or subroutine, bound to the event, using declarative attributes such as Onclick or handle.

Following are the page life cycle events:

* **PreInit** - PreInit is the first event in page life cycle. It checks the IsPostBack property and determines whether the page is a postback. It sets the themes and master pages, creates dynamic controls, and gets and sets profile property values. This event can be handled by overloading the OnPreInit method or creating a Page\_PreInit handler.
* **Init** - Init event initializes the control property and the control tree is built. This event can be handled by overloading the OnInit method or creating a Page\_Init handler.
* **InitComplete** - InitComplete event allows tracking of view state. All the controls turn on view-state tracking.
* **LoadViewState** - LoadViewState event allows loading view state information into the controls.
* **LoadPostData** - During this phase, the contents of all the input fields are defined with the <form> tag are processed.
* **PreLoad** - PreLoad occurs before the post back data is loaded in the controls. This event can be handled by overloading the OnPreLoad method or creating a Page\_PreLoad handler.
* **Load** - The Load event is raised for the page first and then recursively for all child controls. The controls in the control tree are created. This event can be handled by overloading the OnLoad method or creating a Page\_Load handler.
* **LoadComplete** - The loading process is completed, control event handlers are run, and page validation takes place. This event can be handled by overloading the OnLoadComplete method or creating a Page\_LoadComplete handler
* **PreRender** - The PreRender event occurs just before the output is rendered. By handling this event, pages and controls can perform any updates before the output is rendered.
* **PreRenderComplete** - As the PreRender event is recursively fired for all child controls, this event ensures the completion of the pre-rendering phase.
* **SaveStateComplete** - State of control on the page is saved. Personalization, control state and view state information is saved. The HTML markup is generated. This stage can be handled by overriding the Render method or creating a Page\_Render handler.
* **UnLoad** - The UnLoad phase is the last phase of the page life cycle. It raises the UnLoad event for all controls recursively and lastly for the page itself. Final cleanup is done and all resources and references, such as database connections, are freed. This event can be handled by modifying the OnUnLoad method or creating a Page\_UnLoad handler.

4.2 **DataGrid :**

The Windows Forms DataGrid control provides a user interface to ADO.NET datasets, displays ADO.NET tabular data in a scrollable grid, and allows for updates to the data source. In cases where the DataGrid is bound to a data source with a single table containing no relationships, the data appears in simple rows and columns, as in a spreadsheet. The DataGrid control is one of the most useful and flexible controls in Windows Forms. As soon as the DataGrid control is set to a valid data source, the control is automatically populated, by creating columns and rows based on the structure of the data. The DataGrid control can be used to display either a single table or the hierarchical relationships between a set of tables.

private void fnSaveUpdate()

{

 try

 {

 //put the modified DataSet into a new DataSet(myChangedDataset)

 DataSet myChangedDataset= this.dataSet11.GetChanges();

 if (myChangedDataset != null)

 {

 //get how many rows changed

 int modifiedRows = this.oleDbDataAdapter1.Update(myChangedDataset);

 MessageBox.Show("Database has been updated successfully: " +

 modifiedRows + " Modified row(s) ", "Success");

 this.dataSet11.AcceptChanges(); //accept the all changes

 fnRefresh();

 }

 else

 }

 MessageBox.Show("Nothing to save", "No changes");

 }//if-else

 }

 catch(Exception ex)

 {

 //if something somehow went wrong

 MessageBox.Show("An error occurred updating the database: " +

 ex.Message, "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);

 this.dataSet11.RejectChanges(); //cancel the changes

 }//try-catch

 fnEnableDisableAllButtons(true);

}

### Reading the data from an XML file

Sometimes you have to read the data from an XML file and display it in a DataGrid. In order to do this, I pass the XML file to the method; if the file exists, I clear the DataSet and read the XML data into the DataSet using the specified file. The following code snippet shows this:

Hide   Copy Code

private void fnDataReadingFromXMLFile(string filename)

{

 //check if the file exists

 if ( File.Exists(filename))

 {

 //clear the DataSet contents

 this.dataSet11.Clear();

 MessageBox.Show("Data reading from "+filename+" -file");

 this.dataSet11.ReadXml(filename);

 } //if

 else {

 MessageBox.Show(filename + " does NOT exist; Please click" +

 " first the button 'CopyToXML' ", "File Error",

 MessageBoxButtons.OK, MessageBoxIcon.Error );

 }//else

}