CMR
INSTITUTE OF
TECHNOLOGY

USN | | | | | | | | | | |

**Internal Assessment Test III – Jan. 2022**

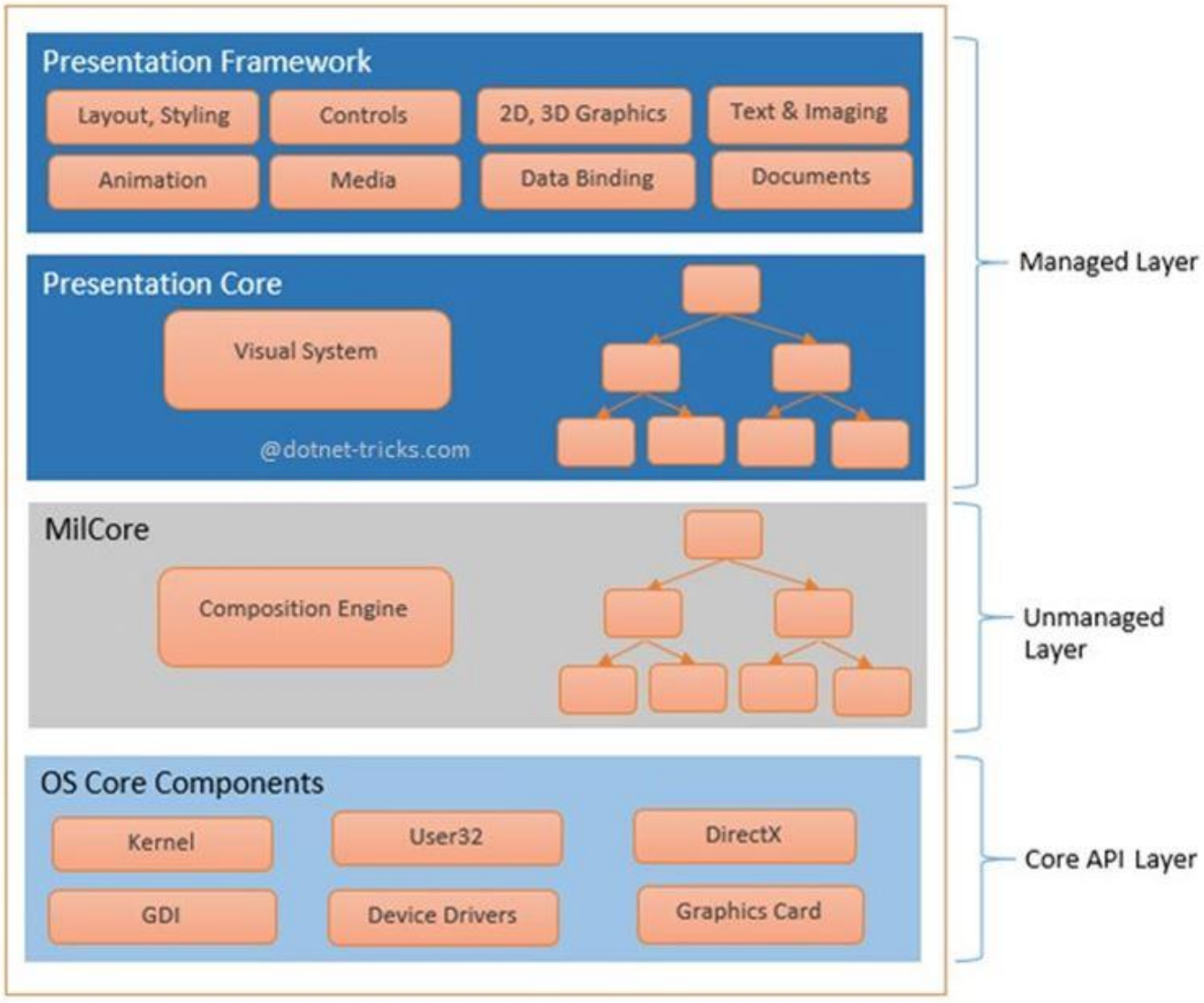
| Sub: | Programming Using C# .Net | | | | | | |
|---|---|---|---|---|---|---|---|
| Date: | 24/01//2022 | Duration: | 90 min's | Max Marks: | 50 | Sem: | V |

**Note : Answer FIVE FULL Questions, choosing ONE full question from each Module**

## PART I

1 Explain WPF architecture with neat diagram.

Windows Presentation Framework is a next generation UI framework to create applications with a rich user experience. It is part of the .NET framework 3.0 and higher. WPF architecture is a layered architecture which have Managed, Unmanaged and Core API layers




WPF Architecture

1. Managed Layer

Managed layer has two main components – Presentation Framework and Presentation Core.

1. Presentation Framework provides the required functionalities that we need to build the WPF applications such as controls, data bindings, styling, shapes, media, documents, annotations, animation and more. PresentationFamework.dll is responsible for this purpose.
2. Presentation Core acts as a managed wrapper around MILCore and provides public interface for MIL. Presentation Core is the home for WPF Visual System and provides classes for creating application visual tree. The Visual System creates visual tree which contains applications Visual Elements and rendering instructions. PresentationCore.dll is responsible for this purpose.

2. Unmanaged Layer

This layer is also called milcore or Media Integration Library Core. MilCore is written in unmanaged code in order to enable tight integration with DirectX. DirectX engine is underlying technology used in WPF to display all graphics, allowing for efficient hardware and software rendering. MIL has Composition System that receives rendering instructions from Visual System and translates into data that can be understood by DirectX to render user interface.

3. Core API Layer

This layer has OS core components like Kernel, User32, GDI, Device Drivers, Graphic cards etc. These components are used by the application to access low level APIs. User32 manages memory and process separation.

**OR**

2 Explain in detail about the following.

1. Event driven GUI 2. MDI windows

GUIs are *event driven* (i.e., they generate *events* when the program's user interacts with the GUI). Typical interactions include moving the mouse, clicking the mouse, clicking a button, typing in a text box, selecting an item from a menu and closing a window. Event handlers are methods that process events and perform tasks. For example, consider a form that changes color when a button is clicked. When clicked, the button generates an event and passes it to the event handler, and the event-handler code changes the form's color.

Each control that can generate events has an associated delegate that defines the signature for that control's event handlers. Recall from Chapter 10 that delegates are objects that contain pointers to methods. Event delegates are *multicast* (class `MulticastDelegate`)—they contain lists of method pointers. Each method must have the same *signature* (i.e., the same list of parameters). In the event-handling model, delegates act as intermediaries between objects that generate events and methods that handle those events (Fig. 12.5).


Fig. 12.5 Event-handling model using delegates.

The Multiple-Document Interface (MDI) is a specification that defines a user interface for applications that enable the user to work with more than one document at the same time under one parent form (window).

Visualize the working style of an application in which you are allowed to open multiple forms in one parent container window, and all the open forms will get listed under the Windows menu. Whereas having an individual window for each instance of the same application is termed as single document interface (SDI); applications such as Notepad, Microsoft Paint, Calculator, and so on, are SDI applications. SDI applications get opened only in their own windows and can become difficult to manage, unlike when you have multiple documents or forms open inside one MDI interface.

Hence, MDI applications follow a parent form and child form relationship model. MDI applications allow you to open, organize, and work with multiple documents at the same time by opening them under the context of the MDI parent form; therefore, once opened, they can't be dragged out of it like an individual form.

The parent (MDI) form organizes and arranges all the child forms or documents that are currently open. You might have seen such options in many Windows applications under a Windows menu, such as Cascade, Tile Vertical, and so on.

**PART II**

3 List and Explain Control class properties and methods.

| Control Properties and Methods | Description |
|---|---|
| *Common Properties* | |
| `BackColor` | Background color of the control. |
| `BackgroundImage` | Background image of the control. |
| `Enabled` | Specifies whether the control is enabled (i.e., if the user can interact with it). A disabled control will still be displayed, but "grayed-out"—portions of the control will become gray. |
| `Focused` | Specifies whether the control has focus. |
| `Font` | Font used to display control's `Text`. |
| `ForeColor` | Foreground color of the control. This is usually the color used to display the control's `Text` property. |
| `TabIndex` | Tab order of the control. When the *Tab* key is pressed, the focus is moved to controls in increasing tab order. This order can be set by the programmer if the `TabStop` property is true. |
| `TabStop` | If `true` (the default value), user can use the *Tab* key to select the control. |
| `Text` | Text associated with the control. The location and appearance varies with the type of control. |
| `TextAlign` | The alignment of the text on the control. One of three horizontal positions (left, center or right) and one of three vertical positions (top, middle or bottom). |
| `Visible` | Specifies whether the control is visible. |

| Control Properties and Methods | Description |
|---|---|
| *Common Methods* | |
| `Focus` | Transfers the focus to the control. |
| `Hide` | Hides the control (equivalent to setting `Visible` to `false`). |
| `Show` | Shows the control (equivalent to setting `Visible` to `true`). |

Fig. 12.11 `Control` class properties and methods. (Part 2 of 2.)

Visual Studio .NET allows the programmer to *anchor* and *dock* controls, which helps to specify the layout of controls inside a container (such as a form). Anchoring allows controls to stay a fixed distance from the sides of the container, even when the control is resized. Docking allows controls to extend themselves along the sides of their containers.

A user may want a control to appear in a certain position (top, bottom, left or right) in a form even when that form is resized. The user can specify this by *anchoring* the control to a side (top, bottom, left or right). The control then maintains a fixed distance from the side to its parent container. In most cases, the parent container is a form; however, other controls can act as a parent container.

When parent containers are resized, all controls move. Unanchored controls move relative to their original position on the form, while anchored controls move so that they will be the same distance from each side that they are anchored to. For example, in Fig. 12.12, the topmost button is anchored to the top and left sides of the parent form. When the form is resized, the anchored button moves so that it remains a constant distance from the top and left sides of the form (its parent). The unanchored button changes position as the form is resized.

**OR**

4 Explain in detail about the following.
1. XAML elements 2. Markup extension classes in XAML

XAML is a new descriptive programming language developed by Microsoft to write user interfaces for next-generation managed applications. XAML is the language to build user interfaces for Windows and Mobile applications that use Windows Presentation Foundation (WPF), UWP, and Xamarin Forms.

The purpose of XAML is simple, to create user interfaces using a markup language that looks like XML. Most of the time, you will be using a designer to create your XAML but you're free to directly manipulate XAML by hand.

XAML uses the XML format for elements and attributes. Each element in XAML represents an object which is an instance of a type. The scope of a type (class, enumeration etc.) is defined as a namespace that physically resides in an assembly (DLL) of the .NET Framework library.
Similar to XML, a XAML element syntax always starts with an open angle bracket (<) and ends with a close angle bracket (>). Each element tag also has a start tag and an end tag. For example, a Button object is represented by the <Button> object element. The following code snippet represents a Button object element.

<Button></Button>

Alternatively, you can use a self-closing format to close the bracket.

<Button />

An object element in XAML represents a type. A type can be a control, a class or other objects

defined in the framework library.
The Root Elements

Each XAML document must have a root element. The root element usually works as a cont
defines the namespaces and basic properties of the element. Three most common root eleme
<Windows />, <Page />, and <UserControl >. The <ResourceDirectory /> and <Applicatio
other two root elements that can be used in a XAML file.

The Window element represents a Window container. The following code snippet shows a
element with its Height, Width, Title and x:Name attributes. The x:Name attribute of an ele
represents the ID of an element used to access the element in the code-behind. The code sni
sets xmlns and xmlns:x attributes that represent the namespaces used in the code. The x:Cla
attribute represents the code-behind class name.

## PART III

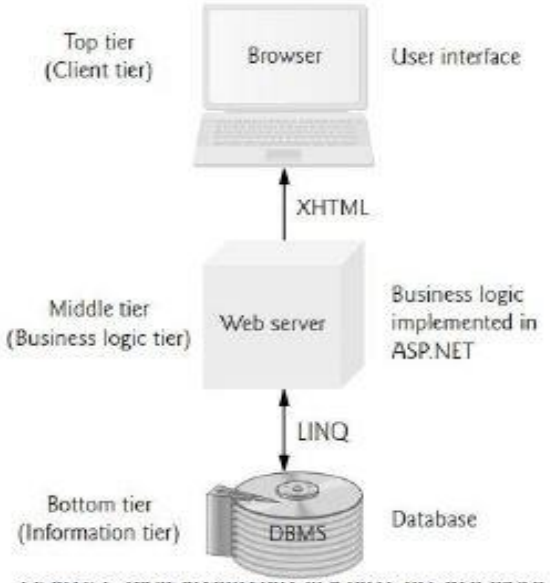
5 Explain in detail multitier application architecture

### Multi-tier Applications Architecture:

Web-based applications are multitier applications and also referred as n-tier applications.

- Multitier applications divide functionality into separate tiers (that is, logical groupings of functionality).
- Tiers can be located on the same computer, the tiers of web-based applications commonly reside on separate computers for security and scalability.

There are 3 tiers. They are:




**application**

**i. Information Tier:**

- The information tier also called the bottom tier.
- It maintains the application's data.
- This tier typically stores data in a relational database management system.

Example: A retail store might have a database for storing product information, such as descriptions, prices and quantities in stock.
The same database also might contain customer information, such as user names, billing addresses and credit card numbers.

- This tier can contain multiple databases, which together comprise the data needed for an application.

**ii. Business Logic:**

- The middle tier implements business logic, *controller logic* and *presentation logic* to control interactions between the application's clients and its data.
- The middle tier acts as an intermediary between data in the information tier and the application's clients.
- The middle-tier **controller logic** processes client requests (such as requests to view a product catalog) and retrieves data from the database.
- The middle-tier **presentation logic** then processes data from the information tier and presents the content to the client.
- Web applications typically present data to clients as web pages.
- Business logic in the middle tier enforces business rules and ensures that data is reliable before the server application updates the database or presents the data to users.
- Business rules dictate how clients can and cannot access application data, and how applications process data.

Example: A business rule in the middle tier of a retail store's web-based application might ensure that all product quantities remain positive.

A client request to set a negative quantity in the bottom tier's product information database would be rejected by the middle tier's business logic.

**iii. Client Tier:**

- The client tier, or top tier, is the application's user interface, which gathers input and displays output.
- Users interact directly with the application through the user interface (typically viewed in a web browser), keyboard and mouse.
- In response to user actions (Example: clicking a hyperlink), the client tier interacts with the middle tier to make requests and to retrieve data from the information tier.
- The client tier then displays to the user the data retrieved from the middle tier.
- The client tier never directly interacts with the information tier.

**OR**

6 Explain different validation controls with suitable example supported by asp .net

### RequiredFieldValidator Control:

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

Syntax:

```
<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.
It has **three** specific properties:

| Properties | Description |
|---|---|
| Type | It defines the type of the data. The available values are: Currency, Date, Double, Integer, and String. |
| MinimumValue | It specifies the minimum value of the range. |
| MaximumValue | It specifies the maximum value of the range. |

Syntax:

```
<asp:RangeValidator ID="rvclass" runat="server" ControlToValidate="txtclass"
        ErrorMessage="Enter your class (6 - 12)" MaximumValue="12"
        MinimumValue="6" Type="Integer">
</asp:RangeValidator>
```

### CompareValidator Control:

The CompareValidator control compares a value in one control with a fixed value or a value in another con

It has the following specific properties:

| Properties | Description |
|---|---|
| Type | It specifies the data type. |
| ControlToCompare | It specifies the value of the input control to compare with. |
| ValueToCompare | It specifies the constant value to compare with. |
| Operator | It specifies the comparison operator, the available values are: Equal, NotEqual, GreaterThan, GreaterThanEqual, LessThan, LessThanEqual, and DataTypeCheck. |

**Validation control** or **Validator**, which determines whether the data in another web control is in the proper format.

- **Validator**s provide a mechanism for validating user input on the client.
- When the page is sent to the client, the validator is *converted into JavaScript* that performs the validation in the client web browser.
- JavaScript is a scripting language that executed on the client. Unfortunately, some client browsers might not support scripting or the user might disable it.

For this reason, you should *always perform validation* on the **server**. ***ASP.NET validation controls*** can function on the **client**, on the **server** or both.

An important aspect of creating ASP.NET Web pages for user input is to be able to check that the information users enter is valid. ASP.NET provides a set of validation controls that provide an easy-to-use but powerful way to check for errors and, if necessary, display messages to the user.

There are six types of validation controls in ASP.NET that listed below:

**The below table describes the controls and their work:**

| Validation Control | Description |
|---|---|
| **RequiredFieldValidation** | Makes an input control a required field |
| **CompareValidator** | Compares the value of one input control to the value of another input control or to a fixed value |
| **RangeValidator** | Checks that the user enters a value that falls between two values |
| **RegularExpressionValidator** | Ensures that the value of an input control matches a specified pattern |
| **CustomValidator** | Allows you to write a method to handle the validation of the value entered |
| **ValidationSummary** | Displays a report of all validation errors occurred in a Web page |

All these validation control classes are inherited from the **BaseValidator** class hence they inherit its properties and methods that are ControlToValidate, Display, EnableClientScript, Enabled, Text, isValid, and validate() method.

### ValidationSummary:

The ValidationSummary control does not perform any validation but shows a summary of all errors in the page.

The summary displays the values of the ErrorMessage property of all validation controls that failed validation.

The following **two mutually inclusive properties** list out the error message:

- ShowSummary : shows the error messages in specified format.
- ShowMessageBox : shows the error messages in a separate window.

## PART IV

Explain in detail about the following.

1. Cookies 2. Session management 3.Master pages

7.

.

**Cookies:** Cookies provide you with a tool for personalizing web pages. A **cookie** is a piece of data stored by web browsers in a small text file on the user's computer. A cookie maintains information about the client during and between browser sessions.

**Session tracking** using the .NET class **HttpSessionState:**
If the user clicks the link for book recommendations, the information stored in the user's unique **HttpSessionState** object is read and used to form the list of recommendations. That can be done using **Session property.**

**Session Property:**

Every Web Form includes a user-specific **HttpSessionState** object, which is accessible through property **Session** of class Page. We use this property to manipulate the current user's **HttpSessionState** object.

When a page is first requested, a unique HttpSessionState object is created by ASP.NET and assigned to the Page's **Session** property.

The session object is created from the HttpSessionState class, which defines a collection of session state items.

The **HttpSessionState class** has the following **properties**:

| Properties | Description |
|---|---|
| SessionID | The unique session identifier. |
| Item(name) | The value of the session state item with the specified name. This is the default property of the HttpSessionState class. |
| Count | The number of items in the session state collection. |
| TimeOut | Gets and sets the amount of time, in minutes, allowed between requests before the session-state provider terminates the session. |

The **HttpSessionState** class has the following **methods**:

| Methods | Description |
|---|---|
| Add(name, value) | Adds an item to the session state collection. |
| Clear | Removes all the items from session state collection. |
| Remove(name) | Removes the specified item from the session state collection. |
| RemoveAll | Removes all keys and values from the session-state collection. |
| RemoveAt | Deletes an item at a specified index from the session-state collection. |

ASP.NET master pages allow you to create a consistent layout for the pages in your application. A single master page defines the look and feel and standard behavior that you want for all of the pages (or a group of pages) in your application. You can then create individual content pages that contain the content you want to display. When users request the content pages, they merge with the master page to produce output that combines the layout of the master page with the content from the content page.

A master page is an ASP.NET file with the extension .master (for example, MySite.master) with a predefined layout that can include static text, HTML elements, and server controls. The master page is identified by a special @ Master directive that replaces the @ Page directive that is used for ordinary .aspx pages. The directive looks like the following.

VB Copy

```
<%@ Master Language="VB" %>
```

The @ Master directive can contain most of the same directives that a @ Control directive can contain. For example, the following master-page directive includes the name of a code-behind file, and assigns a class name to the master page.

VB Copy

```
<%@ Master Language="VB" CodeFile="MasterPage.master.vb" Inherits="MasterPage" %>
```

In addition to the @ Master directive, the master page also contains all of the top-level HTML elements for a page, such as html, head, and form. For example, on a master page you might use an HTML table for the layout, an img element for your company logo, static text for the copyright notice, and server controls to create standard navigation for your site. You can use any HTML and any ASP.NET elements as part of your master page.

**OR**

8 Explain in detail about the following.

1. Script Manager Control

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.

2. Update Panel Control

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.

**PARTV**

9 Explain in detail about ajax and its extension controls.

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 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.

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.

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 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>
```

**OR**

10 Write the code for implementing postal management system web application

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Data.OleDb;
using System.Data;

public partial class View_letter : System.Web.UI.Page
{
    String connStr = @"Provider=Microsoft.ACE.OLEDB.12.0;Data Source=C:\Users\Neha Agrawal\Documents\db_PSM.accdb;Persist Security Info=False;";
    protected void Page_Load(object sender, EventArgs e)
    {
        load_Postman_details();
    }
    public void load_Postman_details()
    {
        if (!IsPostBack)
        {
            using (OleDbConnection con = new OleDbConnection(connStr))
            {
                con.Open();

                String query = "select IdArea,PostmanName from tbl_PostmanDetails";
                OleDbCommand cmd = new OleDbCommand(query, con);

                DropDownList1.DataSource = cmd.ExecuteReader();

                DropDownList1.DataTextField = "PostmanName";
                DropDownList1.DataValueField = "IdArea";
                DropDownList1.DataBind();
                con.Close();
            }
        }
        DropDownList1.Items.Insert(0, new ListItem("--Select Customer--", "0"));
    }
    protected void DropDownList1_SelectedIndexChanged(object sender, EventArgs e)
    {
        OleDbConnection con = new OleDbConnection(connStr);
        try
        {
            con.Open();
            string query = "select IdLetter, LetterAddress from tbl_AreaLetters where IdArea=" + DropDownList1.SelectedValue;
            OleDbCommand cmd = new OleDbCommand(query, con);

            OleDbDataAdapter da = new OleDbDataAdapter(cmd);


            DataSet ds = new DataSet();

            da.Fill(ds);
            GridView1.DataSource = ds;
            GridView1.DataBind();
            con.Close();
        }
        catch (Exception ex)
        {
            Response.Write(ex.Message);
        }

    }
    protected void Button1_Click(object sender, EventArgs e)
    {

    }
```

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

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title></title>
    <asp:ContentPlaceHolder id="head" runat="server">
    </asp:ContentPlaceHolder>
    <style type="text/css">
        .auto-style1
        {
            width: 119px;
            height: 23px;
        }
        .auto-style2
        {
            width: 689px;
            height: 23px;
        }
        .auto-style3
        {
            width: 119px;
            height: 443px;
        }
        .auto-style4
        {
            width: 689px;
            height: 443px;
        }
        .auto-style5
        {
            width: 119px;
            height: 75px;
        }
        .auto-style6
        {
            width: 689px;
            height: 75px;
        }
    </style>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <table style="width: 91%; height: 265px;">
            <tr>
                <td class="auto-style5">
                    <asp:Image ID="Image1" runat="server" ImageUrl="~/Image/postal logo.JPG" Width="137px" />
                </td>
                <td class="auto-style6">
                    <asp:Image ID="Image2" runat="server" Height="116px" ImageUrl="~/Image/postal title.JPG" Width="658px" />
                </td>
            </tr>
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Data.OleDb;

public partial class PostMan_details : System.Web.UI.Page
{
    String connStr = @"Provider=Microsoft.ACE.OLEDB.12.0;Data Source=C:\Users\Neha Agrawal\Documents\db_PSM.accdb;Persist Security Info=False;";



    protected void Page_Load(object sender, EventArgs e)
    {

        load_areaname();
    }



    public void load_areaname()
    {
        if (!IsPostBack)
        {
            using (OleDbConnection con = new OleDbConnection(connStr))
            {
                    con.Open();

                    String query="select IdArea,AreaName from tbl_AreaDetails";
                    OleDbCommand cmd = new OleDbCommand(query, con);

                    DropDownList1.DataSource=cmd.ExecuteReader();

                    DropDownList1.DataTextField = "AreaName";
                    DropDownList1.DataValueField = "IdArea";
                    DropDownList1.DataBind();
                    con.Close();
                }
            }
```