Resultado

**Ejercicio 1: Programa para Calcular Comisiones**

### using System;

### using System.Windows.Forms;

### namespace ComisionVendedores

### {

### public partial class Form1 : Form

### {

### public Form1()

### {

### InitializeComponent();

### }

### private void btnCalcular\_Click(object sender, EventArgs e)

### {

### string nombre = txtNombre.Text;

### string ventasStr = txtVentas.Text;

### if (float.TryParse(ventasStr, out float ventas))

### {

### float comision = ventas \* 0.20f;

### lblResultado.Text = $"Hola {nombre}, tu comisión es de: ${comision:F2}";

### }

### else

### {

### lblResultado.Text = "Por favor, ingresa un monto de ventas válido.";

### }

### }

### }

### }

### }Ejercicio 2: Juego de Adivinar la Palabra

### using System;

### using System.Linq;

### using System.Windows.Forms;

### namespace AdivinaLaPalabra

### {

### public partial class Form1 : Form

### {

### private string palabraSecreta = "ejemplo";

### private string palabraOculta;

### private int vidas = 6;

### public Form1()

### {

### InitializeComponent();

### palabraOculta = new string('\_', palabraSecreta.Length);

### lblPalabraOculta.Text = palabraOculta;

### lblVidas.Text = $"Vidas: {vidas}";

### }

### private void btnAdivinar\_Click(object sender, EventArgs e)

### {

### char letra = txtLetra.Text[0];

### if (palabraSecreta.Contains(letra))

### {

### for (int i = 0; i < palabraSecreta.Length; i++)

### {

### if (palabraSecreta[i] == letra)

### {

### palabraOculta = palabraOculta.Remove(i, 1).Insert(i, letra.ToString());

### }

### }

### lblPalabraOculta.Text = palabraOculta;

### }

### else

### {

### vidas--;

### lblVidas.Text = $"Vidas: {vidas}";

### }

### if (vidas == 0)

### {

### MessageBox.Show("¡Has perdido! La palabra era: " + palabraSecreta);

### }

### else if (palabraOculta == palabraSecreta)

### {

### MessageBox.Show("¡Felicidades! Has adivinado la palabra.");

### }

### txtLetra.Clear();

### }

### }

### }

### Ejercicio 3: Sistema de Gestión de Informes

using System;

using System.IO;

using System.Windows.Forms;

namespace GestionInformes

{

public partial class Form1 : Form

{

private string basePath = @"C:\Kosmos";

public Form1()

{

InitializeComponent();

}

private void Form1\_Load(object sender, EventArgs e)

{

lblRuta.Text = $"Ruta: {basePath}";

if (!Directory.Exists(basePath))

{

Directory.CreateDirectory(basePath);

Directory.CreateDirectory(Path.Combine(basePath, "Marketing"));

Directory.CreateDirectory(Path.Combine(basePath, "Ventas"));

Directory.CreateDirectory(Path.Combine(basePath, "Compras"));

Directory.CreateDirectory(Path.Combine(basePath, "Metricas"));

}

lblTotalInformes.Text = $"Total de informes: {Directory.GetFiles(basePath, "\*.txt", SearchOption.AllDirectories).Length}";

}

private void btnOpcion1\_Click(object sender, EventArgs e)

{

// Lógica para leer un informe

}

private void btnOpcion2\_Click(object sender, EventArgs e)

{

// Lógica para crear un nuevo informe

}

private void btnOpcion3\_Click(object sender, EventArgs e)

{

// Lógica para crear una nueva categoría

}

private void btnOpcion4\_Click(object sender, EventArgs e)

{

// Lógica para eliminar un informe

}

private void btnOpcion5\_Click(object sender, EventArgs e)

{

// Lógica para eliminar una categoría

}

private void btnOpcion6\_Click(object sender, EventArgs e)

{

Application.Exit();

}

}

}

### 4. Programación Orientada a Objetos

public class Empleado

{

private string nombre;

private string apellido;

private decimal salario;

public Empleado(string nombre, string apellido, decimal salario)

{

this.nombre = nombre;

this.apellido = apellido;

this.salario = salario;

}

public string Nombre { get => nombre; set => nombre = value; }

public string Apellido { get => apellido; set => apellido = value; }

public decimal Salario { get => salario; set => salario = value; }

public decimal CalcularSalarioAnual()

{

return salario \* 12;

}

}

public class Gerente : Empleado

{

public string Departamento { get; set; }

public Gerente(string nombre, string apellido, decimal salario, string departamento)

: base(nombre, apellido, salario)

{

Departamento = departamento;

}

public void IncrementarSalario(decimal porcentaje)

{

Salario += Salario \* porcentaje / 100;

}

}

// Uso de las clases

var empleado = new Empleado("Juan", "Perez", 3000m);

Console.WriteLine($"Salario anual: {empleado.CalcularSalarioAnual()}");

var gerente = new Gerente("Ana", "Lopez", 5000m, "Marketing");

gerente.IncrementarSalario(10);

Console.WriteLine($"Nuevo salario del gerente: {gerente.Salario}");

### 5. Lista de Tareas en una Web

### Configurar el Modelo

clase TaskItem para representar las tareas.

using System.ComponentModel.DataAnnotations;

namespace TaskListApp.Models

{

public class TaskItem

{

public int Id { get; set; }

[Required]

public string Title { get; set; }

public string Description { get; set; }

public DateTime CreatedDate { get; set; } = DateTime.Now;

public bool IsCompleted { get; set; }

}

}

### Configurar la Base de Datos con Entity Framework Core

**paquetes de Entity Framework Core necesarios:**

dotnet add package Microsoft.EntityFrameworkCore

dotnet add package Microsoft.EntityFrameworkCore.SqlServer

dotnet add package Microsoft.EntityFrameworkCore.Tools

**Contexto de la base de datos ApplicationDbContext.**

using Microsoft.EntityFrameworkCore;

namespace TaskListApp.Data

{

public class ApplicationDbContext : DbContext

{

public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)

: base(options)

{

}

public DbSet<TaskListApp.Models.TaskItem> TaskItems { get; set; }

}

}

1. Configura la cadena de conexión en appsettings.json.

json

"ConnectionStrings": {

"DefaultConnection": "Server=(localdb)\\mssqllocaldb;Database=TaskListApp;Trusted\_Connection=True;MultipleActiveResultSets=true"

}

Configuracion del servicio de la base de datos en Startup.cs o Program.cs.

public void ConfigureServices(IServiceCollection services)

{

services.AddControllersWithViews();

services.AddDbContext<ApplicationDbContext>(options =>

options.UseSqlServer(Configuration.GetConnectionString("DefaultConnection")));

}

Crea y aplica la migración.

sql

dotnet ef migrations add InitialCreate

dotnet ef database update

### Controlador y las Vistas

1. Crea el controlador TaskItemsController.

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using TaskListApp.Data;

using TaskListApp.Models;

using System.Linq;

using System.Threading.Tasks;

namespace TaskListApp.Controllers

{

public class TaskItemsController : Controller

{

private readonly ApplicationDbContext \_context;

public TaskItemsController(ApplicationDbContext context)

{

\_context = context;

}

// GET: TaskItems

public async Task<IActionResult> Index(string searchString)

{

var tasks = from t in \_context.TaskItems

select t;

if (!String.IsNullOrEmpty(searchString))

{

tasks = tasks.Where(s => s.Title.Contains(searchString));

}

return View(await tasks.ToListAsync());

}

// GET: TaskItems/Create

public IActionResult Create()

{

return View();

}

// POST: TaskItems/Create

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Create([Bind("Id,Title,Description,CreatedDate,IsCompleted")] TaskItem taskItem)

{

if (ModelState.IsValid)

{

\_context.Add(taskItem);

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

return View(taskItem);

}

// GET: TaskItems/Edit/5

public async Task<IActionResult> Edit(int? id)

{

if (id == null)

{

return NotFound();

}

var taskItem = await \_context.TaskItems.FindAsync(id);

if (taskItem == null)

{

return NotFound();

}

return View(taskItem);

}

// POST: TaskItems/Edit/5

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Edit(int id, [Bind("Id,Title,Description,CreatedDate,IsCompleted")] TaskItem taskItem)

{

if (id != taskItem.Id)

{

return NotFound();

}

if (ModelState.IsValid)

{

try

{

\_context.Update(taskItem);

await \_context.SaveChangesAsync();

}

catch (DbUpdateConcurrencyException)

{

if (!TaskItemExists(taskItem.Id))

{

return NotFound();

}

else

{

throw;

}

}

return RedirectToAction(nameof(Index));

}

return View(taskItem);

}

// GET: TaskItems/Delete/5

public async Task<IActionResult> Delete(int? id)

{

if (id == null)

{

return NotFound();

}

var taskItem = await \_context.TaskItems

.FirstOrDefaultAsync(m => m.Id == id);

if (taskItem == null)

{

return NotFound();

}

return View(taskItem);

}

// POST: TaskItems/Delete/5

[HttpPost, ActionName("Delete")]

[ValidateAntiForgeryToken]

public async Task<IActionResult> DeleteConfirmed(int id)

{

var taskItem = await \_context.TaskItems.FindAsync(id);

\_context.TaskItems.Remove(taskItem);

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

private bool TaskItemExists(int id)

{

return \_context.TaskItems.Any(e => e.Id == id);

}

}

}

1. Crear las vistas correspondientes para Index, Create, Edit y Delete en la carpeta Views/TaskItems.

### Estilos CSS

Crea un archivo CSS llamado site.css en wwwroot/css y utiliza los colores de Konecta (gamas de azules).

css

body {

font-family: Arial, sans-serif;

background-color: #f4f8fb;

color: #333;

}

header {

background-color: #003366;

color: white;

padding: 1rem;

text-align: center;

}

a {

color: #0066cc;

text-decoration: none;

}

a:hover {

text-decoration: underline;

}

table {

width: 100%;

border-collapse: collapse;

margin-top: 1rem;

}

table, th, td {

border: 1px solid #ddd;

}

th, td {

padding: 0.5rem;

text-align: left;

}

th {

background-color: #003366;

color: white;

}

button {

background-color: #0066cc;

color: white;

border: none;

padding: 0.5rem 1rem;

cursor: pointer;

}

button:hover {

background-color: #005bb5;

}

input[type="text"], input[type="date"] {

width: 100%;

padding: 0.5rem;

margin: 0.5rem 0;

box-sizing: border-box;

}

### CRUD en las Vistas

Ejemplo de la vista Index.cshtml:

html

@model IEnumerable<TaskListApp.Models.TaskItem>

@{

ViewData["Title"] = "Task List";

}

<h1>Task List</h1>

<form method="get" asp-controller="TaskItems" asp-action="Index">

<input type="text" name="searchString" placeholder="Search tasks..." value="@ViewData["searchString"]" />

<button type="submit">Search</button>

</form>

<p>

<a asp-controller="TaskItems" asp-action="Create">Create New Task</a>

</p>

<table>

<thead>

<tr>

<th>

@Html.DisplayNameFor(model => model.Title)

</th>

<th>

@Html.DisplayNameFor(model => model.Description)

</th>

<th>

@Html.DisplayNameFor(model => model.CreatedDate)

</th>

<th>

@Html.DisplayNameFor(model => model.IsCompleted)

</th>

<th></th>

</tr>

</thead>

<tbody>

@foreach (var item in Model)

{

<tr>

<td>

@Html.DisplayFor(modelItem => item.Title)

</td>

<td>

@Html.DisplayFor(modelItem => item.Description)

</td>

<td>

@Html.DisplayFor(modelItem => item.CreatedDate)

</td>

<td>

@Html.DisplayFor(modelItem => item.IsCompleted)

</td>

<td>

<a asp-controller="TaskItems" asp-action="Edit" asp-route-id="@item.Id">Edit</a> |

<a asp-controller="

### 6. Aplicación de Clima

Imports System.Net.Http

Imports Newtonsoft.Json

Public Class Form1

Private Async Sub btnObtenerClima\_Click(sender As Object, e As EventArgs) Handles btnObtenerClima.Click

Dim ciudad As String = txtCiudad.Text

Dim apiKey As String = "TU\_API\_KEY"

Dim url As String = $"http://api.openweathermap.org/data/2.5/weather?q={ciudad}&appid={apiKey}&units=metric"

Using client As New HttpClient()

Try

Dim response As HttpResponseMessage = Await client.GetAsync(url)

response.EnsureSuccessStatusCode()

Dim responseBody As String = Await response.Content.ReadAsStringAsync()

Dim clima As Clima = JsonConvert.DeserializeObject(Of Clima)(responseBody)

lblTemperatura.Text = $"Temperatura: {clima.main.temp}°C"

lblDescripcion.Text = $"Descripción: {clima.weather(0).description}"

lblSensacion.Text = $"Sensación térmica: {clima.main.feels\_like}°C"

Catch ex As Exception

MessageBox.Show("Error al obtener los datos del clima.")

End Try

End Using

End Sub

End Class

Public Class Clima

Public Property main As Main

Public Property weather As List(Of Weather)

End Class

Public Class Main

Public Property temp As Double

Public Property feels\_like As Double

End Class

Public Class Weather

Public Property description As String

End Class