Resumen Nivel 2, Lecciones 1 y 2

Introducción a Desarrollo de Software

Daniel Germán - 1128127

9 de diciembre, 2024

El segundo nivel de este programa por capacitateparaelempleo.org cubre más contenido sobre el uso del C# en un IDE, sus variables y componentes, tanto como sus aplicaciones en espacios diferentes a la consola integrada del IDE.

Proyectos para aplicación de escritorio

Estos proyectos se pueden crear dentro de las IDEs creando un espacio visual de C#. Un form es un espacio que se crea como resultado de esto y es donde correrá el programa ejecutado. Parece una función de consola en si. Se debe hacer una llamada al form tras llamar al Main. Cada form posee un .cs y un .designer.cs, con la primera siendo lo que edita el programador y la última siendo lo que lee e interpreta el dispositivo. El IDE define la estructura del programa en estos casos.

IDEs

En algunos IDEs, se encuentran paneles de exploración y propiedades, que permiten acceder al código de manera eficiente y rápida. Esto incluye todo lo que se encuentre dentro de los espacios de nombre, el archivo, y otros componentes. En un desarrollador, se puede diseñar de manera visual un form. El panel de propiedades permite acceder a los detalles de los objetos creados de manera rápida utilizando las variables dadas dicho objeto. Cada propiedad tiene una descripción.

Forms

Es el contenedor principal de la ventana que incluye controles como botones, cuadros de texto y etiquetas, permitiendo al usuario interactuar con la aplicación. Es pertinente al desarrollo front end, que se refiere a la parte de un programa con la cual un usuario interactúa de manera directa,

y el backend, que es el código que funciona detrás de lo visual para darle funcionamiento práctico y eficiente. Para este tipo de desarrollo, los controladores y propiedades de objetos son primordiales para el procesamiento lógico y procedimiento adecuado al momento de utilizar un programa de este tipo.

Controles

Los controles son los elementos gráficos que permiten a los usuarios interactuar con una aplicación. Ejemplos comunes incluyen botones (Button) para ejecutar acciones, etiquetas (Label) para mostrar texto, cajas de texto (TextBox) para entrada de datos, y listas desplegables (ComboBox) para opciones seleccionables. Los controles se agregan a los forms, ya sea visualmente usando el diseñador de VSC o mediante código. Cada control tiene propiedades (como Text, Size o Color), eventos (como Click o TextChanged) y métodos (como Show() o Hide()) que facilitan su personalización y funcionalidad. La interacción con los controles permite construir interfaces ricas y dinámicas.

Multiforms

El uso de multiforms es común para crear aplicaciones con varias ventanas o secciones independientes. Esto permite dividir la funcionalidad en forms específicos, como una ventana principal y otras secundarias para configuración, visualización de datos, o diálogos.
Para trabajar con multiforms, se puede instanciar un formulario secundario desde el principal utilizando new Form(); seguido de newForm.Show(); para mostrarlo sin bloquear la ejecución del formulario principal, o newForm.ShowDialog(); si se necesita que el formulario secundario sea modal. Además, se pueden pasar datos entre forms a través de constructores personalizados,

propiedades públicas o eventos. Esto ayuda a mantener una estructura organizada y escalable en aplicaciones con múltiples funcionalidades.

Transferencia de datos entre Forms

La transferencia de datos entre forms se realiza mediante varias técnicas: usar constructores personalizados para pasar valores al crear el form secundario; establecer propiedades públicas en el form destino para asignar datos antes de mostrarlo; emplear eventos personalizados para que el form secundario envíe información al principal; o utilizar clases estáticas como contenedores globales para datos compartidos.

Paneles

Los paneles son controles contenedores que permiten organizar y agrupar otros controles dentro de una interfaz gráfica. Se utilizan para gestionar el diseño de la aplicación, proporcionando un área delimitada donde se pueden agregar botones, etiquetas, cuadros de texto, entre otros. Los paneles son útiles para crear interfaces más ordenadas, ya que se pueden mover o modificar como un todo, ocultarlos o mostrarlos dinámicamente con la propiedad visible, y desplazarse dentro de ellos activando AutoScroll. Además, se pueden anidar paneles para lograr diseños complejos y responsivos, lo que facilita la personalización y el manejo de las interfaces.

Actividad

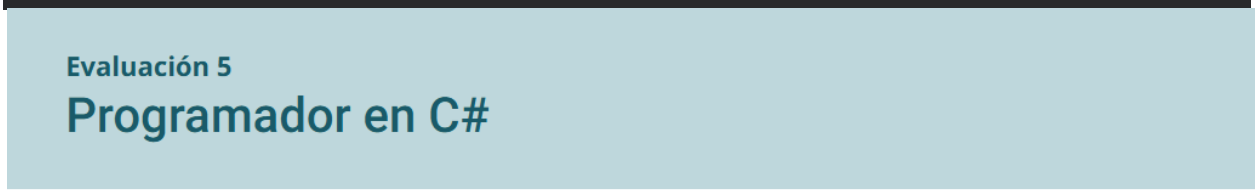
```
using System;
using System.Windows.Forms;

namespace LoginApp
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void btnRegistro_Click(object sender, EventArgs e)
        {
            RegistroForm registro = new RegistroForm();
            registro.Show();
        }

        private void btnLogin_Click(object sender, EventArgs e)
        {
            LoginForm login = new LoginForm();
            login.Show();
        }
    }
}
```

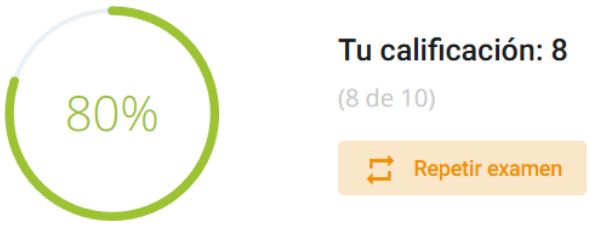
```
using System;
using System.Windows.Forms;

namespace LoginApp
{
    public partial class RegistroForm : Form
    {
        public RegistroForm()
        {
            InitializeComponent();
```

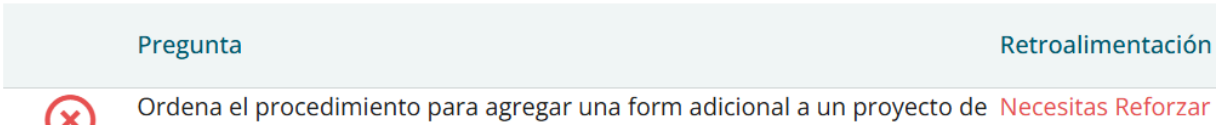
```
        }

        private void btnGuardar_Click(object sender, EventArgs e)
        {
            string nombre = txtNombre.Text;
            string apellido = txtApellido.Text;
            string email = txtEmail.Text;
            string sexo = cmbSexo.SelectedItem.ToString();
            string usuario = txtUsuario.Text;
            string contraseña = txtContraseña.Text;

            MessageBox.Show("Registro completado con éxito.", "Éxito",
MessageBoxButtons.OK, MessageBoxIcon.Information);
            this.Close();
        }
    }
}
```

```
using System;
using System.Windows.Forms;

namespace LoginApp
{
    public partial class LoginForm : Form
    {
        public LoginForm()
        {
            InitializeComponent();
        }

        private void btnIngresar_Click(object sender, EventArgs e)
        {
            string usuario = txtUsuario.Text;
            string contraseña = txtContraseña.Text;

            if (usuario == "admin" && contraseña == "1234")
            {
```

```
                MessageBox.Show("Ingreso exitoso", "Éxito",
MessageBoxButtons.OK, MessageBoxIcon.Information);
                this.Close();
            }
            else
            {
                MessageBox.Show("Usuario o contraseña incorrectos",
"Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
        }
    }
}
```







Sistema Gestor de Base de Datos

Un Sistema Gestor de Base de Datos (SGBD) es un conjunto de programas que permite crear, administrar y manipular bases de datos. Su principal función es facilitar el almacenamiento, la recuperación y la manipulación de datos de manera eficiente, garantizando la integridad,

seguridad y consistencia de la información. Un SGBD actúa como un intermediario entre las aplicaciones de los usuarios y las bases de datos.

Procesos almacenados CRUD

Los procesos almacenados son bloques de código que se almacenan en un sistema gestor de base de datos y se ejecutan de manera centralizada en el servidor de la base de datos. Se utilizan para realizar diversas operaciones de manera eficiente y segura. Son útiles para realizar operaciones comunes, como las CRUD (Crear, Leer, Actualizar, Eliminar) de manera más eficiente y segura. Al utilizar procesos almacenados, se mejora el rendimiento, ya que la lógica de negocio se ejecuta en el servidor y no en el cliente. Además, permiten una mayor seguridad, controlando el acceso a los datos mediante permisos específicos, y facilitan el mantenimiento, pues centralizan las operaciones en el servidor, lo que reduce la duplicación de código en las aplicaciones cliente.

LINQ

LINQ (Language Integrated Query) es una característica de los lenguajes de programación .NET que permite realizar consultas de datos de manera sencilla y eficiente, integrando la consulta directamente en el código. Con LINQ, los desarrolladores pueden trabajar con diferentes fuentes de datos como colecciones en memoria, bases de datos, XML y servicios web, utilizando una sintaxis declarativa similar a SQL pero dentro del lenguaje de programación. Esto facilita tareas como el filtrado, ordenamiento, agrupamiento y agregación de datos, mejorando la legibilidad y mantenimiento del código, además de reducir el riesgo de errores comunes en las consultas tradicionales.

Instalador

Se puede utilizar Visual Studio Installer, una herramienta integrada en Visual Studio que permite generar instaladores de manera sencilla. Primero, se desarrolla la aplicación en Visual Studio, y luego se crea un proyecto de instalación donde se incluyen los archivos necesarios, como el ejecutable y sus dependencias. Se pueden personalizar opciones como la ubicación de instalación, accesos directos y archivos adicionales. Al compilar el proyecto de instalación, Visual Studio genera un archivo .exe que se puede distribuir a los usuarios para instalar la aplicación en sus sistemas.

Actividad

```
using System;
using System.Data.SqlClient;
using System.Linq;

public class Database
{
    private string connectionString =
"Server=your_server;Database=RegistroUsuarios;Integrated Security=True;";

    public void InsertarUsuario(string nombre, string usuario, string
contraseña, string correo)
    {
        using (SqlConnection conn = new SqlConnection(connectionString))
        {
            conn.Open();
            SqlCommand cmd = new SqlCommand("sp_InsertarUsuario", conn);
            cmd.CommandType = System.Data.CommandType.StoredProcedure;
            cmd.Parameters.AddWithValue("@Nombre", nombre);
            cmd.Parameters.AddWithValue("@Usuario", usuario);
            cmd.Parameters.AddWithValue("@Contraseña", contraseña);
            cmd.Parameters.AddWithValue("@Correo", correo);
            cmd.ExecuteNonQuery();
        }
```

```
    }

    public IQueryable<Usuario> ObtenerUsuarios()
    {
        using (SqlConnection conn = new SqlConnection(connectionString))
        {
            conn.Open();
            SqlCommand cmd = new SqlCommand("sp_ObtenerUsuarios", conn);
            cmd.CommandType = System.Data.CommandType.StoredProcedure;
            SqlDataReader reader = cmd.ExecuteReader();

            var usuarios = new List<Usuario>();
            while (reader.Read())
            {
                usuarios.Add(new Usuario
                {
                    Id = (int)reader["Id"],
                    Nombre = (string)reader["Nombre"],
                    Usuario = (string)reader["Usuario"],
                    Correo = (string)reader["Correo"]
                });
            }

            return usuarios.AsQueryable();
        }
    }
}

private void btnRegistrar_Click(object sender, EventArgs e)
{
    string nombre = txtNombre.Text;
    string usuario = txtUsuario.Text;
    string contraseña = txtContraseña.Text;
    string correo = txtCorreo.Text;

    Database db = new Database();
    db.InsertarUsuario(nombre, usuario, contraseña, correo);
    MessageBox.Show("Usuario registrado correctamente");
}
```

```
private void btnLogin_Click(object sender, EventArgs e)
{
    string usuario = txtUsuario.Text;
    string contraseña = txtContraseña.Text;

    using (SqlConnection conn = new SqlConnection(connectionString))
    {
        conn.Open();
        SqlCommand cmd = new SqlCommand("SELECT * FROM Usuarios WHERE
Usuario = @Usuario AND Contraseña = @Contraseña", conn);
        cmd.Parameters.AddWithValue("@Usuario", usuario);
        cmd.Parameters.AddWithValue("@Contraseña", contraseña);

        SqlDataReader reader = cmd.ExecuteReader();
        if (reader.HasRows)
        {
            MessageBox.Show("Login exitoso");
            UsuariosCRUD usuariosCRUD = new UsuariosCRUD();
            usuariosCRUD.Show();
        }
        else
        {
            MessageBox.Show("Usuario o contraseña incorrectos");
        }
    }
}

public partial class UsuariosCRUD : Form
{
    public UsuariosCRUD()
    {
        InitializeComponent();
    }

    private void UsuariosCRUD_Load(object sender, EventArgs e)
    {
        Database db = new Database();
        var usuarios = db.ObtenerUsuarios();
        dataGridView1.DataSource = usuarios.ToList();
    }
```

```
    private void btnGuardar_Click(object sender, EventArgs e)
    {
    }

    private void btnEliminar_Click(object sender, EventArgs e)
    {
    }
}
```

Examen