Universidad Estatal de Sonora
La Fuerza del Saber Estimulará mi Espíritu

MANUAL DE PRÁCTICAS DE LABORATORIO

Nombre de la asignatura

Laboratorio

Programa Académico:	**Ingenieria en Software I.S**
Plan de Estudios	**2021**
Fecha de elaboración	**27/06/2025**
Versión del Documento	**1**

Universidad Estatal de Sonora
La Fuerza del Saber Estimulará mi Espíritu

Dra. Martha Patricia Patiño Fierro
Rectora

Mtra. Ana Lisette Valenzuela Molina
Encargada del Despacho de la Secretaría General Académica

Mtro. José Antonio Romero Montaño
Secretario General Administrativo

Lic. Jorge Omar Herrera Gutiérrez
Encargado de Despacho de Secretario General de Planeación

Tabla de contenido

INTRODUCCIÓN

Como parte de las herramientas esenciales para la formación académica de los estudiantes de la Universidad Estatal de Sonora, se definen manuales de práctica de laboratorio como elemento en el cual se define la estructura normativa de cada práctica y/o laboratorio, además de representar una guía para la aplicación práctica del conocimiento y el desarrollo de las competencias clave en su área de estudio. Su diseño se encuentra alineado con el modelo educativo institucional, el cual privilegia el aprendizaje basado en competencias, el aprendizaje activo y la conexión con escenarios reales.

Con el propósito de fortalecer la autonomía de los estudiantes, su pensamiento crítico y sus habilidades para la resolución de problemas, las prácticas de laboratorio integran estrategias didácticas como el aprendizaje basado en proyectos, el trabajo colaborativo, la experimentación guiada y el uso de tecnologías educativas. De esta manera, se promueve un proceso de enseñanza-aprendizaje dinámico, en el que los estudiantes no solo adquieren conocimientos teóricos, sino que también desarrollan habilidades prácticas y reflexivas para su desempeño profesional.

Señalar en este apartado brevemente los siguientes elementos según corresponda:

- Propósito del manual
- Justificación de su uso en el programa académico
- Competencias a desarrollar
    - **Competencias blandas:** Habilidades transversales que se refuerzan en las prácticas, como la comunicación, el trabajo en equipo, el uso de tecnologías, etc.
    - **Competencias disciplinares:** Conocimientos específicos del área del laboratorio, incluyendo fundamentos teóricos y habilidades técnicas.
    - **Competencias profesionales:** Aplicación de los conocimientos adquiridos en escenarios reales o simulados, en concordancia con el perfil de egreso del programa.

IDENTIFICACIÓN

Nombre de la Asignatura		Programación Aplicada	
Clave	061CP035	Créditos	6
Asignaturas Antecedentes	061CP040	Plan de Estudios	2021

Área de Competencia	Competencia del curso
Desarrollar software y servicios de soporte técnico y redes, con la finalidad de solucionar problemas y agilizar procesos en la toma de decisiones en empresas públicas y privadas, bajo estándares de calidad nacional e internacional, a través del análisis de problemas, comunicación, liderazgo e innovación.	Desarrollar aplicaciones de software, con la interacción de base de datos a través de un entorno de programación visual, bajo estándares de calidad, para agilizar los procesos y la toma de decisiones en las organizaciones, con un enfoque de análisis de problemas y trabajo en equipo.

Carga Horaria de la asignatura

Horas Supervisadas			Horas Independientes	Total de Horas
Aula	Laboratorio	Plataforma		
3	1	1	2	7

Consignación del Documento

Unidad Académica	Unidad Académica Hermosillo
Fecha de elaboración	27/06/2025
Responsables del diseño	**Julian Flores Figueroa** julian.flores@ues.mx https://orcid.org/0000-0002-4155-8153 **Jalil Gerardo Espinoza Zepeda** jalil.espinoza@ues.mx https://orcid.org/0009-0007-3064-077X
Validación Recepción	Coordinación de Procesos Educativos

MATRIZ DE CORRESPONDENCIA

Señalar la relación de cada práctica con las competencias del perfil de egreso

PRÁCTICA	PERFIL DE EGRESO
Práctica No. 1: Creación de una Aplicación Básica en Windows Forms con Visual Studio	Desarrollar software con la finalidad de agilizar los procesos y la toma de decisiones en empresas públicas y privadas, bajo estándares de calidad nacional e internacional con enfoque de liderazgo.
Práctica No. 2: Desarrollo de un Sistema de Gestión de Contactos con Arquitectura por Capas en Windows Forms	Aplicar soluciones e innovaciones tecnológicas con la finalidad de automatizar procesos, atendiendo los principios de organización y gestión de información.
Práctica No. 3: Diseño de Formularios Interactivos con Controles Avanzados y Validación de Datos en Windows Forms	Crear bases de datos para una gestión eficiente de la información, garantizando la integridad y seguridad de los datos, atendiendo los requerimientos de la organización.
Práctica No. 4: Desarrollo de un Sistema Empresarial Básico con Arquitectura en Capas en Windows Forms	Desarrollar software con la finalidad de agilizar los procesos y la toma de decisiones en empresas públicas y privadas, bajo estándares de calidad nacional e internacional con enfoque de liderazgo.
Práctica No. 5: Diseño de Interfaces para un Sistema Escolar en Windows Forms: Organización Visual, Personalización y Control de Componentes	Desarrollar software con la finalidad de agilizar los procesos y la toma de decisiones en empresas públicas y privadas, bajo estándares de calidad nacional e internacional con enfoque de liderazgo.
Práctica No. 6: Diseño Modular de Aplicaciones en C# utilizando Arquitectura en Tres Capas con Validación de Entradas y Gestión de Imágenes	Aplicar soluciones e innovaciones tecnológicas con la finalidad de automatizar los procesos, atendiendo los principios de la organización y gestión efectiva de la información, poniendo en práctica sus habilidades de trabajo en equipo y planeación.
Práctica No. 7: Validación Avanzada de Entradas y Uso de Depuración con Unicode en Aplicaciones Windows Forms	Desarrollar soporte y asistencia técnica para la prevención y corrección de problemas en los sistemas de cómputo, atendiendo los requerimientos y políticas de la organización, garantizando la optimización y el uso responsable de los recursos.
Práctica No. 8: Gestión de Formularios con	Desarrollar software con la finalidad de

Colecciones de Controles y Validación Secuencial en Windows Forms	agilizar los procesos y la toma de decisiones en empresas públicas y privadas, bajo estándares de calidad nacional e internacional con enfoque de liderazgo.
Práctica No. 9: Conversión de Imágenes a Arreglos de Bytes en Aplicaciones Windows Forms con C#	Crear bases de datos para una gestión eficiente de la información, garantizando la integridad y seguridad de los datos, atendiendo los requerimientos de la organización con un sentido de liderazgo e innovación.
Práctica No. 10: Instalación, Administración y Conexión de SQL Server desde Visual Studio con LINQ to DB	Crear bases de datos para una gestión eficiente de la información, garantizando la integridad y seguridad de los datos, atendiendo los requerimientos de la organización con un sentido de liderazgo e innovación
Práctica No. 11: Almacenamiento de Imágenes en Bases de Datos con C# y LINQ to DB mediante Arreglos de Bytes	Crear bases de datos para una gestión eficiente de la información, garantizando la integridad y seguridad de los datos, atendiendo los requerimientos de la organización con un sentido de liderazgo e innovación
Práctica No. 12: Visualización de Registros e Imágenes en un DataGridView desde una Base de Datos con C# y LINQ to DB	Crear bases de datos para una gestión eficiente de la información, garantizando la integridad y seguridad de los datos, atendiendo los requerimientos de la organización con un sentido de liderazgo e innovación
Práctica No. 13: Operaciones CRUD: Insertar y Editar Estudiantes con Imágenes	Crear bases de datos para una gestión eficiente de la información, garantizando la integridad y seguridad de los datos, atendiendo los requerimientos de la organización con un sentido de liderazgo e innovación.
Práctica No. 14: Implementación del Método Eliminar con Acceso a Base de Datos	Crear bases de datos para una gestión eficiente de la información, garantizando la integridad y seguridad de los datos, atendiendo los requerimientos de la organización con un sentido de liderazgo e innovación.

RELACIÓN DE PRÁCTICAS DE LABORATORIO POR ELEMENTO DE COMPETENCIA

Elemento de Competencia al que pertenece la práctica	**Indicar EC: I** Identificar el entorno de desarrollo, herramientas de programación y diseño de visual estudio para la Creación de un sistema computacional bajo estándares de calidad, en las organizaciones a través del trabajo en equipo.

PRÁCTICA	NOMBRE	COMPETENCIA
Práctica No. 1	Creación de una Aplicación Básica en Windows Forms con Visual Studio	Construir una aplicación básica en Windows Forms para generar un mensaje personalizado de saludo, utilizando Visual Studio 2022 y el lenguaje de programación C#, en un entorno de desarrollo gráfico, aplicando buenas prácticas de programación y fomentando el trabajo colaborativo.
Práctica No. 2	Desarrollo de un Sistema de Gestión de Contactos con Arquitectura por Capas en Windows Forms	Diseñar un sistema modular para la gestión de contactos con formularios interactivos, con la finalidad de integrar la validación de datos y la arquitectura por capas, utilizando Visual Studio 2022 y C# en un entorno de desarrollo de aplicaciones de escritorio, promoviendo el pensamiento lógico y el trabajo colaborativo.
Práctica No. 3	Diseño de Formularios Interactivos con Controles Avanzados y Validación de Datos en Windows Forms	Diseñar formularios interactivos con controles visuales en Windows Forms para capturar y validar datos personales, utilizando Visual Studio y el lenguaje C# bajo el modelo de programación orientada a eventos, en el desarrollo de aplicaciones de escritorio, fomentando la creatividad y la atención al detalle.
Práctica No. 4	Desarrollo de un Sistema Empresarial Básico con Arquitectura en Capas en Windows Forms	Desarrollar un sistema básico de gestión empresarial con separación por capas para manejar clientes, proveedores e inventarios, mediante el uso de Visual Studio 2022 y el lenguaje C#, en un entorno de programación orientada a objetos, fortaleciendo la capacidad de análisis lógico y la organización del código.

Elemento de Competencia al que pertenece la práctica	**Indicar EC: II**
	Programar formularios que permitan la recuperación de información de una base de datos bajo estándares de calidad, para el apoyo en la toma de decisiones dentro de las organizaciones, mediante el análisis de problemas.

PRÁCTICA	NOMBRE	COMPETENCIA
Práctica No. 5	Diseño de Interfaces para un Sistema Escolar en Windows Forms: Organización Visual, Personalización y Control de Componentes	Diseñar interfaces gráficas para un sistema escolar utilizando controles visuales personalizados, con la finalidad de organizar y mejorar la experiencia del usuario, mediante formularios desarrollados en Visual Studio y el entorno Windows Forms, promoviendo la atención al detalle y la creatividad en el diseño de aplicaciones.
Práctica No. 6	Diseño Modular de Aplicaciones en C# utilizando Arquitectura en Tres Capas con Validación de Entradas y Gestión de Imágenes	Implementa una aplicación con arquitectura en tres capas, con validación de entradas y carga de imágenes, para fortalecer la separación de responsabilidades y la reutilización del código, utilizando Visual Studio con C# en un entorno de Windows Forms, promoviendo el pensamiento analítico y el trabajo colaborativo.
Práctica No. 7	Validación Avanzada de Entradas y Uso de Depuración con Unicode en Aplicaciones Windows Forms	Aplica técnicas de validación de entradas en campos de texto utilizando códigos Unicode y eventos del teclado, con la finalidad de garantizar la integridad de los datos capturados, bajo condiciones controladas de depuración en tiempo de ejecución, dentro de un entorno de desarrollo en Windows Forms con C#, fortaleciendo el pensamiento crítico y la atención al detalle.
Práctica No. 8	Gestión de Formularios con Colecciones de Controles y Validación Secuencial en Windows Forms	Implementa la validación secuencial de campos en formularios utilizando colecciones de controles TextBox y Label, con la finalidad de garantizar la integridad de los datos ingresados, bajo condiciones de retroalimentación visual en tiempo de ejecución, en un entorno de desarrollo con C# y Windows Forms, fortaleciendo la organización del trabajo y la responsabilidad en la codificación.
Práctica No. 9	Conversión de Imágenes a Arreglos de Bytes en Aplicaciones Windows Forms con C#	Convierte imágenes a arreglos de bytes para su almacenamiento o procesamiento digital, mediante el uso de métodos personalizados y control PictureBox, en un entorno de desarrollo con C# y Windows Forms, fortaleciendo la precisión técnica y la capacidad de análisis.

Elemento de Competencia al que pertenece la práctica	**Indicar EC: III** Desarrollar aplicaciones para la automatización de tareas recurrentes dentro de las organizaciones bajo el paradigma cliente-servidor utilizando un lenguaje de programación visual bajo estándares de calidad mediante el análisis de problemas y trabajo en equipo.

PRÁCTICA	NOMBRE	COMPETENCIA
Práctica No. 10	Instalación, Administración y Conexión de SQL Server desde Visual Studio con LINQ to DB	Configura e integra una base de datos SQL Server en una aplicación de escritorio, con la finalidad de almacenar y gestionar información de manera estructurada, utilizando LINQ to DB como herramienta de acceso a datos, en un entorno de desarrollo con Visual Studio y C#, fortaleciendo la autonomía técnica y la capacidad de resolución de problemas.
Práctica No. 11	Almacenamiento de Imágenes en Bases de Datos con C# y LINQ to DB mediante Arreglos de Bytes	Inserta imágenes en una base de datos relacional utilizando arreglos de bytes como medio de almacenamiento, con la finalidad de vincular contenido gráfico al registro de entidades, mediante el uso de C#, Windows Forms y LINQ to DB, fortaleciendo la atención al detalle y la capacidad de estructurar soluciones técnicas integradas.
Práctica No. 12	Visualización de Registros e Imágenes en un DataGridView desde una Base de Datos con C# y LINQ to DB	Despliega datos e imágenes almacenados en una base de datos en un control DataGridView, con la finalidad de representar visualmente la información de forma estructurada y funcional, mediante el uso de C#, LINQ to DB y Windows Forms, fortaleciendo la atención al detalle y la organización en el diseño de interfaces visuales.
Práctica No. 13	Operaciones CRUD: Insertar y Editar Estudiantes con Imágenes	Implementa un módulo de registro y edición de estudiantes con manejo de imágenes, para actualizar y mantener la información visual y textual en una base de datos, utilizando eventos del DataGridView y controles de formulario en C#, dentro del entorno de desarrollo Visual Studio con Windows Forms, demostrando atención al detalle y pensamiento lógico en la solución de problemas.
Práctica No. 14	Implementación del Método Eliminar con Acceso a Base de Datos	Implementa un método para eliminar registros de estudiantes de la base de datos, con el fin de mantener actualizada y depurada la información del sistema, previa validación de existencia y confirmación del usuario, utilizando el lenguaje C# en un entorno Windows Forms con acceso a datos mediante LINQ, demostrando responsabilidad y toma de decisiones conscientes en la gestión de datos.

UES
Universidad Estatal de Sonora
La Fuerza del Saber Estimulará mi Espíritu

PRÁCTICAS

NOMBRE DE LA PRÁCTICA No. 1	**Creación de una Aplicación Básica en Windows Forms con Visual Studio**
COMPETENCIA DE LA PRÁCTICA	Construir una aplicación básica en Windows Forms para generar un mensaje personalizado de saludo, utilizando Visual Studio 2022 y el lenguaje de programación C#, en un entorno de desarrollo gráfico, aplicando buenas prácticas de programación y fomentando el trabajo colaborativo.

FUNDAMENTO TÉORICO

Windows Forms es una tecnología de .NET para crear aplicaciones de escritorio con interfaz gráfica en Windows, usando C#. Visual Studio 2022 facilita su desarrollo mediante herramientas visuales y de depuración. La práctica integra principios de programación orientada a objetos (POO), manejo de eventos con delegados, y diseño centrado en el usuario, además de introducir el ciclo de compilación y ejecución de aplicaciones.

MATERIALES, EQUIPAMIENTO Y/O REACTIVOS

Equipo de cómputo

- Computadora personal o de laboratorio con sistema operativo Windows 10 o superior
- Procesador mínimo: Intel Core i3 o equivalente
- Memoria RAM mínima: 8 GB
- Espacio disponible en disco: al menos 10 GB

Software

- Visual Studio 2022 (versión Community, Professional o Enterprise) con los siguientes componentes:
  - Desarrollo de escritorio con .NET
  - Windows Forms .NET Framework o .NET 6/7, según elección del estudiante
  - .NET SDK correspondiente a la versión seleccionada (recomendado: .NET 6 o .NET 7)

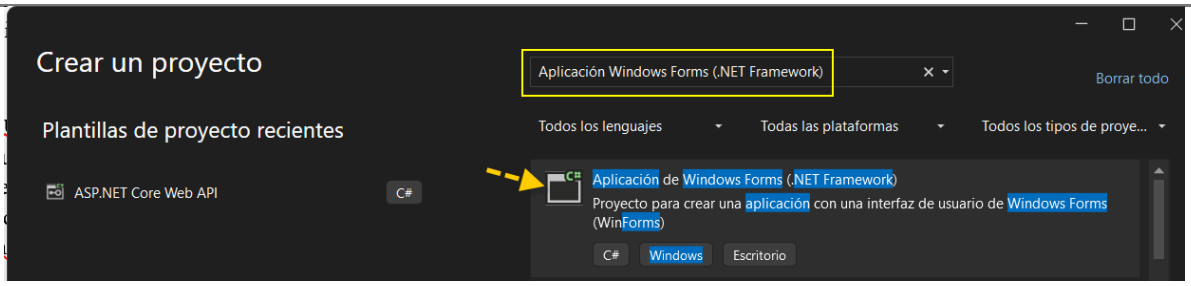
Recursos digitales

- Acceso a Internet para descarga de herramientas (en caso de ser necesario)
- Carpeta asignada o unidad compartida para guardar el proyecto generado
- Cuenta de Microsoft (opcional, para sincronización de Visual Studio)

PROCEDIMIENTO O METODOLOGÍA

Crear un nuevo proyecto

1. Abre Visual Studio 2022.
2. Haz clic en **"Crear un nuevo proyecto"**.
3. En el cuadro de búsqueda, escribe **"Windows Forms App"**.
4. Selecciona:
   - **"Aplicación Windows Forms (.NET Framework)"**, si necesitas usar versiones anteriores de .NET Framework.
   - **"Aplicación Windows Forms (.NET)"**, si prefieres usar **.NET 5/6/7.**
5. Haz clic en **"Siguiente"**.

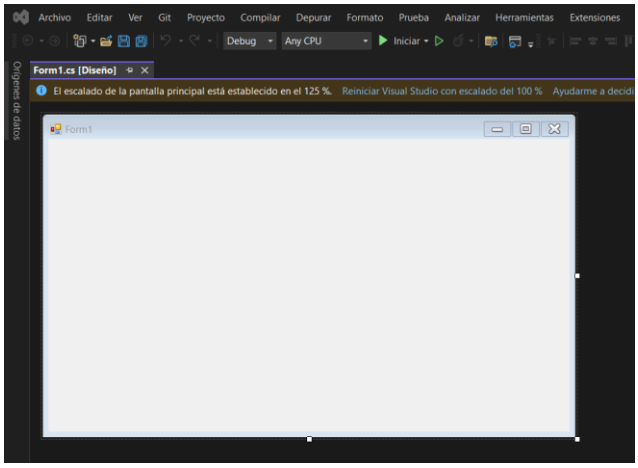


Configuración del proyecto

1. En la ventana de configuración:
   - **Nombre del proyecto**: Escribe un nombre significativo, como **MiAppWindowsForm.**
   - **Ubicación**: Selecciona la carpeta donde guardarás el proyecto.
   - **Nombre de la solución**: Puedes dejarlo igual al nombre del proyecto.
   - **Marco de trabajo (Framework)**: Elige la versión deseada (recomendado: .NET 6 o superior para aplicaciones modernas).
2. Haz clic en **"Crear"**.

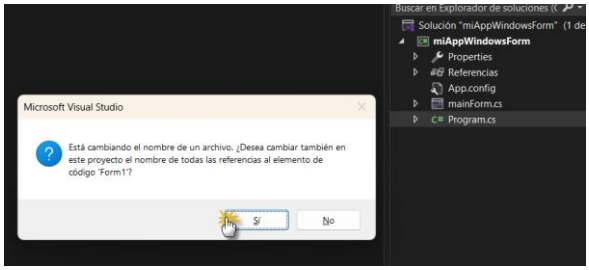
Personalizar el formulario principal

Una vez creado el proyecto, Visual Studio abrirá el editor de diseño de **Windows Forms.**



Cambiar el nombre del formulario

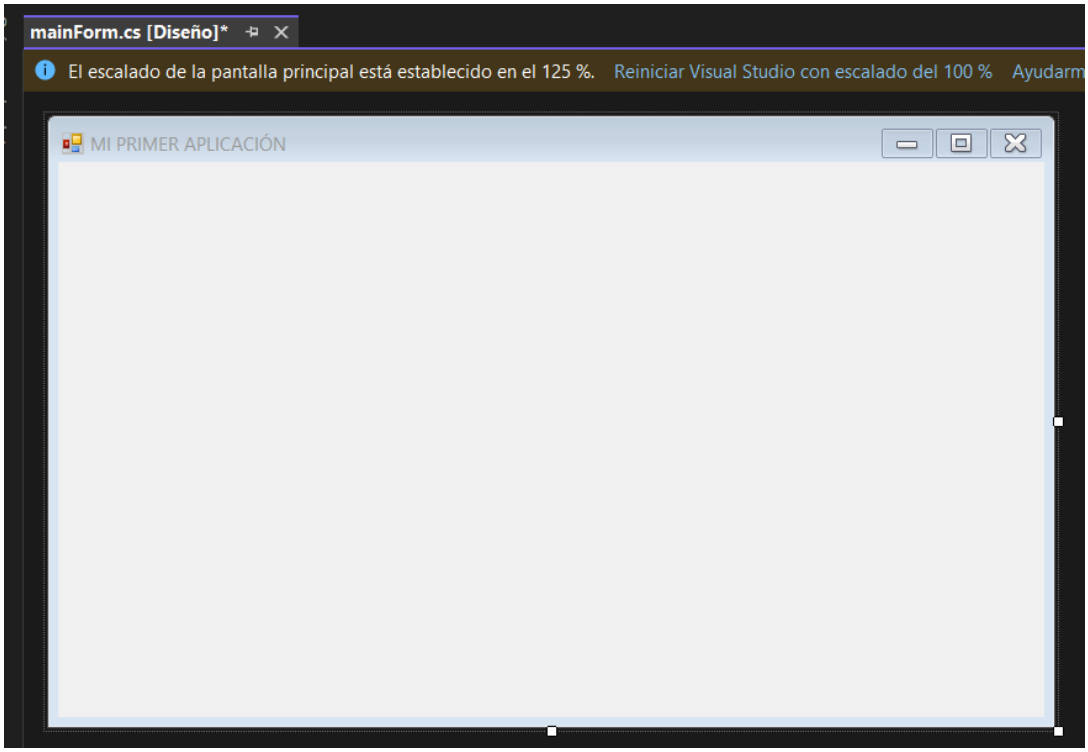
1. Haz clic derecho en el archivo **Form1.cs** desde el Explorador de soluciones.
2. Selecciona **"Renombrar"** y escribe un nombre nuevo, como **MainForm**.
3. **Acepta** los cambios y Visual Studio actualizará automáticamente las referencias internas.



Cambiar propiedades del formulario

1. Haz clic en el **formulario** principal en el área de diseño.

2. En la ventana **Propiedades**:
    - Cambia la propiedad **Text** a un título como **"Mi Primera Aplicación".**
    - Ajusta **StartPosition** a **CenterScreen** para que el formulario aparezca centrado al ejecutarse.

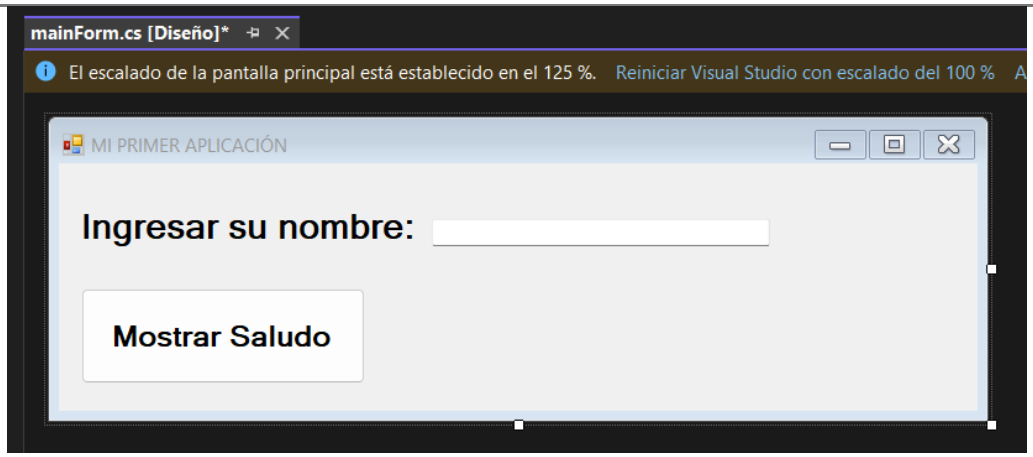


Añadir controles al formulario

Usa la **Caja de herramientas** (Toolbox) para arrastrar y soltar controles visuales al formulario.

Controles básicos

1. **Label (Etiqueta)**:
    - Arrastra un control **Label** al formulario.
    - Cambia su propiedad **Text** a algo como **"Ingrese su nombre:".**
2. **TextBox (Caja de texto)**:
    - Arrastra un control **TextBox** debajo del Label.
    - Cambia su propiedad **Name** a **txtNombre**.
3. **Button (Botón)**:
    - Arrastra un control **Button** al formulario.
    - Cambia su propiedad **Text** a **"Mostrar saludo".**
    - Cambia su propiedad **Name** a **btnSaludo**.



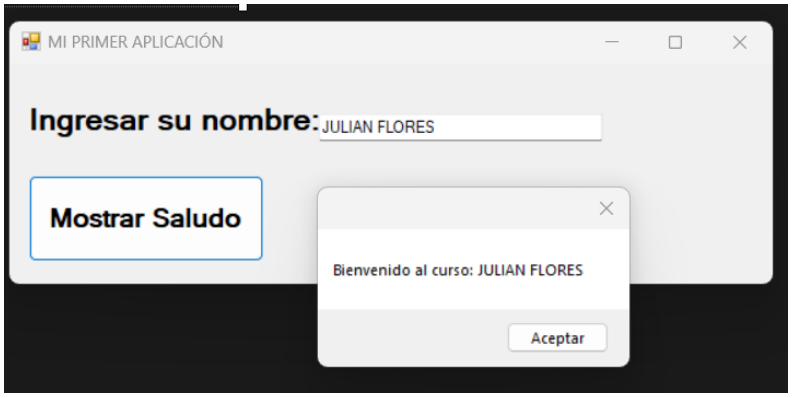
Programar eventos de los controles

1. Haz doble clic en el botón **btnSaludo**. Visual Studio abrirá el archivo de código asociado al formulario y creará un **evento Click.**
2. Dentro del **evento**, agrega el siguiente código:
3. **Guarda** los cambios.

```
private void btnSaludo_Click(object sender, EventArgs e)
{
   //Declaramos la variable y le asignaremos el valor contenido en el control
   string nombre = txtNombre.Text;

   //Enviamos un saludo
   MessageBox.Show($"Bienvenido al curso: {nombre} ");

}
```

Compilar y ejecutar la aplicación

1. Haz clic en el botón **Iniciar** (icono de ▶) o **presiona F5.**
2. Visual Studio compilará el proyecto y ejecutará tu aplicación.
3. Prueba ingresar un nombre en la caja de texto y presiona el botón. Debería aparecer un cuadro de diálogo con el saludo personalizado.



RESULTADOS ESPERADOS

Al concluir esta práctica, el estudiante deberá ser capaz de:

- Crear un proyecto funcional en Visual Studio 2022 utilizando la plantilla de Windows Forms.
- Configurar correctamente los parámetros del proyecto: nombre, ubicación, versión del framework y nombre de la solución.
- Diseñar una interfaz gráfica básica que incluya controles esenciales como Label, TextBox y Button.
- Asignar nombres lógicos y propiedades adecuadas a los controles del formulario.
- Programar eventos básicos, en particular el evento Click de un botón, para capturar datos introducidos por el usuario y mostrar un mensaje personalizado.
- Ejecutar y depurar la aplicación para validar su funcionalidad.
- Entregar un reporte con capturas de pantalla del proceso y del resultado final de la aplicación.

ANÁLISIS DE RESULTADOS

Responde de forma clara y reflexiva las siguientes preguntas con base en la práctica realizada:

1. ¿Qué pasos consideraste esenciales para crear correctamente el proyecto en Visual Studio?
2. ¿Cuál fue la utilidad de cada uno de los controles agregados al formulario (Label, TextBox, Button)?
3. ¿Qué sucedió cuando ejecutaste la aplicación y escribiste tu nombre? ¿El resultado fue el esperado?
4. ¿Qué problemas o errores encontraste durante la práctica y cómo los resolviste?
5. ¿Qué aprendiste sobre la programación de eventos en C# a través del botón btnSaludo?
6. ¿Cómo relacionas esta práctica con el desarrollo de aplicaciones reales en tu área profesional?
7. ¿Qué habilidades técnicas y blandas pusiste en práctica durante esta actividad?

CONCLUSIONES Y REFLEXIONES

Al finalizar esta práctica, el estudiante podrá concluir que:

- La creación de interfaces gráficas con Windows Forms representa una base sólida para el desarrollo de aplicaciones de escritorio en el ecosistema .NET.
- Visual Studio es una herramienta poderosa e intuitiva que facilita tanto el diseño visual como la programación de eventos en C#.
- Comprender la estructura del formulario, el uso adecuado de los controles y la lógica detrás de los eventos es esencial para construir aplicaciones funcionales.
- La interacción usuario-aplicación se logra mediante la captura de datos (por ejemplo, desde un TextBox) y la respuesta programada del sistema (como mostrar un mensaje en pantalla).
- Esta práctica refuerza el pensamiento lógico, la atención al detalle y la capacidad de resolución de problemas, habilidades fundamentales en el desarrollo de software.

ACTIVIDADES COMPLEMENTARIAS

Realiza las siguientes actividades adicionales para afianzar los conocimientos adquiridos durante la práctica:

1. **Modificar el diseño del formulario:**
   - Cambia los colores de fondo y fuente del formulario y de los controles.
   - Agrega un ícono personalizado a la ventana de la aplicación.
   - Ajusta el tamaño del formulario y organiza los controles usando un GroupBox.
2. **Agregar nuevos controles y funcionalidades:**
   - Inserta un segundo TextBox para capturar el apellido del usuario.
   - Modifica el botón para que el saludo incluya el nombre completo.
   - Añade un segundo botón llamado **"Limpiar"** que borre el contenido de ambos TextBox.
3. **Manejar validaciones básicas:**
   - Agrega una validación para evitar que el mensaje se muestre si los campos están vacíos.
   - Muestra un MessageBox de advertencia si no se captura ningún dato.
4. **Investigación:**
   - Investiga qué es y cómo funciona el **event handler** en C#.
   - Investiga cómo crear y utilizar **clases personalizadas** en aplicaciones Windows Forms.
5. **Documentación y entrega:**
   - Documenta las nuevas funciones implementadas con capturas de pantalla.
   - Elabora una breve reflexión sobre cómo estas mejoras pueden aplicarse a un sistema real de captura de datos.

EVALUACIÓN Y EVIDENCIAS DE APRENDIZAJE	
Criterios de evaluación	**Configuración correcta del proyecto en Visual Studio.** • Diseño funcional del formulario con controles básicos. • Programación correcta del evento Click. • Ejecución funcional de la aplicación sin errores. • Análisis reflexivo de resultados. • Implementación de al menos una mejora. • Entrega de reporte con evidencia del trabajo.
Rúbricas o listas de cotejo para valorar desempeño	Lista de cotejo con siete criterios clave que evalúan la creación del proyecto, el diseño de la interfaz, la funcionalidad del código, el análisis reflexivo y la presentación del reporte. Cada ítem se evalúa con las opciones: “Cumple / No cumple” y espacio para observaciones.
Formatos de reporte de prácticas	**Formato libre estructurado que incluya:** 1. Datos del alumno. 2. Nombre de la práctica. 3. Objetivo. 4. Capturas de pantalla del proyecto. 5. Código implementado. 6. Respuestas al análisis. 7. Conclusiones y reflexión.

NOMBRE DE LA PRÁCTICA No. 2	**Desarrollo de un Sistema de Gestión de Contactos con Arquitectura por Capas en Windows Forms**
COMPETENCIA DE LA PRÁCTICA	Diseñar un sistema modular para la gestión de contactos con formularios interactivos, con la finalidad de integrar la validación de datos y la arquitectura por capas, utilizando Visual Studio 2022 y C# en un entorno de desarrollo de aplicaciones de escritorio, promoviendo el pensamiento lógico y el trabajo colaborativo.

FUNDAMENTO TÉORICO

Esta práctica aplica el modelo de arquitectura por capas, que separa la interfaz de usuario (UI), la lógica de negocio y el acceso a datos para lograr un diseño modular, mantenible y escalable. Se utilizan formularios Windows Forms para crear interfaces gráficas, mientras que las clases y métodos en C# implementan principios de programación orientada a objetos, como encapsulación y reutilización del código. La capa de presentación gestiona la interacción visual con el usuario, la lógica de negocio valida los datos y la capa de acceso a datos permite insertar registros en la base. Este enfoque refuerza el desarrollo estructurado de aplicaciones y mejora la experiencia del usuario mediante validaciones y retroalimentación visual.

MATERIALES, EQUIPAMIENTO Y/O REACTIVOS

Equipo de cómputo

- Computadora personal o de laboratorio con sistema operativo Windows 10 o superior
- Procesador mínimo: Intel Core i3 o equivalente
- Memoria RAM mínima: 8 GB
- Espacio disponible en disco: al menos 10 GB

Software

- Visual Studio 2022 (versión Community, Professional o Enterprise) con los siguientes componentes:
  - Desarrollo de escritorio con .NET
  - Windows Forms .NET Framework o .NET 6/7, según elección del estudiante
  - .NET SDK correspondiente a la versión seleccionada (recomendado: .NET 6 o .NET 7)

Recursos digitales

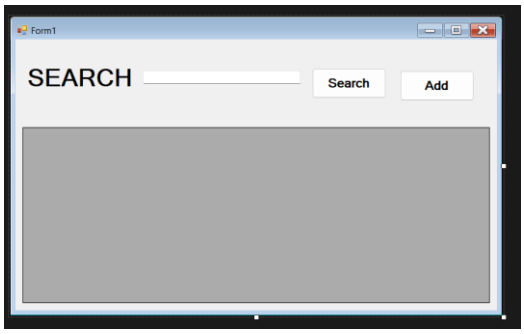
- Acceso a Internet para descarga de herramientas (en caso de ser necesario)
- Carpeta asignada o unidad compartida para guardar el proyecto generado
- Cuenta de Microsoft (opcional, para sincronización de Visual Studio)

PROCEDIMIENTO O METODOLOGÍA

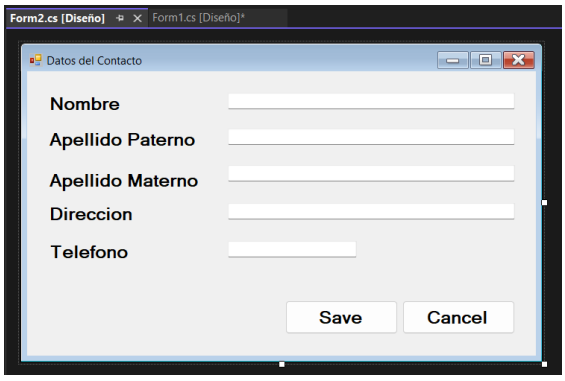
Objetivo: Desarrollar un sistema modular y funcional con formularios interactivos que permita gestionar contactos, integrando una arquitectura de diseño por capas y validación de datos.

Creación de Formularios

1. **Formulario Contacts:**
   - Crea un **formulario** principal denominado **Contacts**. Este formulario debe incluir Grid donde se visualizarán los contactos existentes, y botones para las acciones principales (**Agregar**, **Editar**, **Eliminar**).



2. **Formulario ContactDetails:**
   - Crea un formulario secundario denominado ContactDetails. Este formulario debe contener campos para capturar la información de un contacto, como:
     - Nombre
     - Apellido Paterno
     - Apellido Materno
     - Teléfono
     - Correo Electrónico
   - Agrega un botón identificado como Save.
   - Agrega un botón identificado como Cancel.



Modularidad del Código

3. **Creación de Funciones:**

- Implementa una función llamada **OpenContactDetails** que permita abrir el formulario **ContactDetails** desde el formulario principal.
- Utiliza comentarios y regiones para organizar el código relacionado con cada funcionalidad

```
#region Funciones para Formularios

public void OpenContactDetails()

{

    // Lógica para abrir el formulario de detalles

}

#endregion
```

Interactividad

4. **Evento de Interacción en el Botón "Agregar":**
   - Programa el botón **Agregar** en el formulario **Contacts** para que al hacer clic:
     - Abra una **ventana flotante** (**ContactDetails**) utilizando la función **OpenContactDetails**.

Organización Modular - Diseño por Capas

5. **Capa Model:**
   - **Crea una clase** denominada **Contact** que represente los **datos** de un **contacto**. Define las propiedades autoimplementadas necesarias, como:

```
public class Contact
{
    //Propiedades auto-implementadas
}
```

Capa BusinessLogic:

- Implementa una clase **BusinessLogic** con métodos para validar los datos de entrada. **Por ejemplo:**

```
public class BusinessLogic
{
    public bool ValidateContact(string firstName, string lastName, string middleName)
```

```
    {
        return !string.IsNullOrEmpty(firstName) &&
               !string.IsNullOrEmpty(lastName) &&
               !string.IsNullOrEmpty(middleName);
    }
}
```

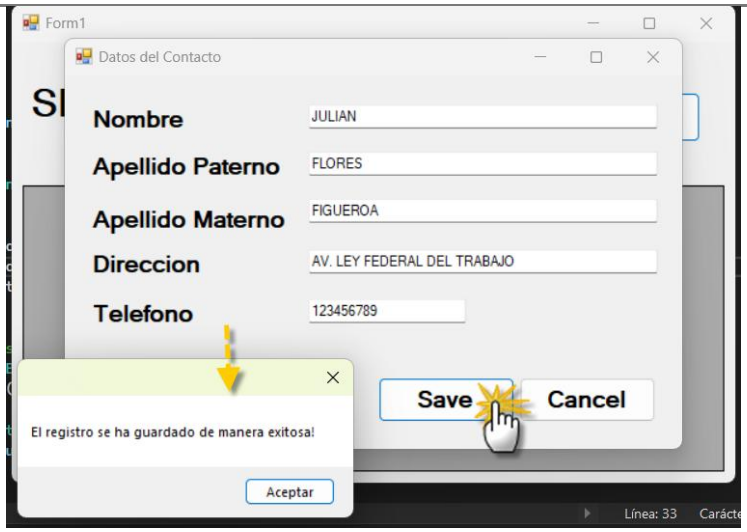
Capa DataAccess:

- Implementa una clase **DataAccess** que permita interactuar con la base de datos. Crea un método para **insertar contactos:**

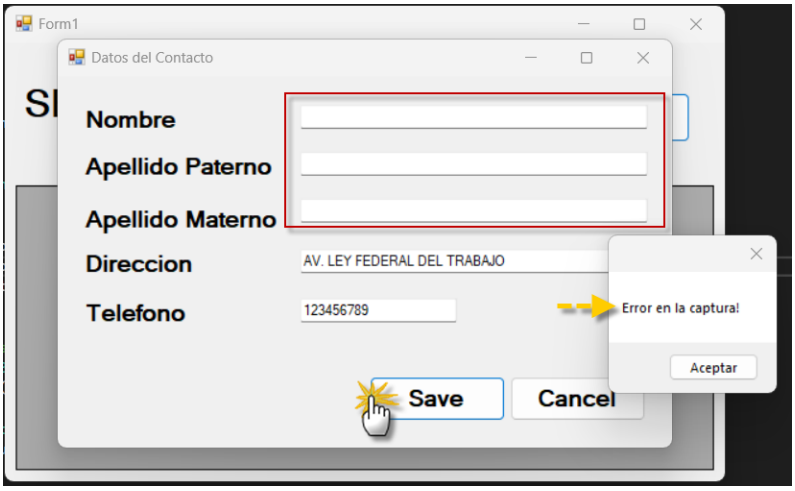
```
public class DataAccess
{
    public void InsertContact(Contact contact)
    {
        string query = "INSERT INTO Contacts (FirstName, LastName, MiddleName, Phone, Email) VALUES (@FirstName, @LastName, @MiddleName, @Phone, @Email)";
        // Lógica para ejecutar el query con los parámetros adecuados.
    }
}
```

Integración con la Interfaz de Usuario

8. **Programar el Botón Save:**
    - En el formulario **ContactDetails**, programa el botón **Save** para que realice las siguientes acciones:
        1. **Validación:**
            - Llama a la función de validación de la capa **BusinessLogic**.
            - Si los datos **son válidos:**
                - Llama a la función de la capa **DataAccess** para guardar los datos en la base de datos.
                - Muestra un mensaje de confirmación (por ejemplo, "Contacto guardado correctamente").



- Si los datos **no son válidos**:
  - Muestra un mensaje de error indicando qué campos faltan o son incorrectos.



RESULTADOS ESPERADOS

Al finalizar la práctica, el estudiante será capaz de:

- Construir una aplicación funcional de escritorio para la gestión de contactos utilizando Windows Forms y C#.
- Diseñar una interfaz gráfica modular compuesta por formularios principales y secundarios (Contacts y ContactDetails).
- Implementar una estructura de código organizada por capas: presentación (UI), lógica de negocio (BusinessLogic) y acceso a datos (DataAccess).
- Crear y utilizar clases personalizadas para representar modelos de datos (Contact).
- Validar la entrada de datos antes de realizar operaciones de guardado.
- Ejecutar funciones para abrir formularios secundarios y mostrar retroalimentación al usuario.

- Aplicar buenas prácticas de codificación, incluyendo el uso de regiones, comentarios y separación de responsabilidades.
- Entregar un proyecto documentado con evidencias funcionales y estructurales del sistema desarrollado

ANÁLISIS DE RESULTADOS

Responde con base en la experiencia obtenida durante la práctica:

1. ¿Cómo se relacionan los formularios "Contacts" y "ContactDetails" en el funcionamiento general del sistema?
2. ¿Qué ventajas observaste al implementar la arquitectura por capas en el desarrollo del sistema?
3. ¿Fue clara la separación entre la lógica de presentación, la lógica de negocio y el acceso a datos? Justifica tu respuesta.
4. ¿Qué dificultades encontraste al validar los datos ingresados por el usuario? ¿Cómo las solucionaste?
5. ¿Qué sucede si los campos obligatorios están vacíos y se intenta guardar el contacto?
6. ¿Qué técnicas usaste para mantener el código organizado y legible?
7. ¿Cómo podría escalarse esta aplicación para incluir otras funcionalidades (por ejemplo, búsqueda o filtrado)?
8. ¿Qué habilidades blandas consideras que fortaleciste durante esta práctica?

CONCLUSIONES Y REFLEXIONES

Esta práctica permitió al estudiante aplicar la arquitectura por capas para desarrollar una aplicación modular de escritorio, separando la interfaz, la lógica de negocio y el acceso a datos. Se reforzaron conceptos clave como el diseño de formularios interactivos, la validación de datos y la organización del código en funciones reutilizables. Además, se fortalecieron habilidades técnicas y blandas esenciales para el desarrollo profesional, como el pensamiento lógico y el trabajo en equipo.

ACTIVIDADES COMPLEMENTARIAS

1. **Agregar funcionalidad de edición y eliminación de contactos**
   - Implementa los botones “Editar” y “Eliminar” en el formulario **Contacts**.
   - Permite modificar o eliminar contactos previamente registrados mediante la reutilización del formulario **ContactDetails**.
2. **Incluir validación avanzada de datos**
   - Agrega reglas como formato válido de correo electrónico y longitud mínima del número telefónico.
   - Muestra mensajes descriptivos al usuario en caso de error.
3. **Persistencia de datos simulada o con base de datos real**
   - Integra una base de datos local (SQL Server Express o SQLite) o una estructura de almacenamiento en memoria.
   - Asegúrate de conectar correctamente la capa de acceso a datos.

EVALUACIÓN Y EVIDENCIAS DE APRENDIZAJE

Criterios de evaluación	• Diseño correcto de formularios. • Aplicación de arquitectura por capas. • Validación funcional de datos. • Modularidad y organización del código. • Pruebas y ejecución exitosa. • Análisis y reflexión final.
Rúbricas o listas de cotejo para valorar desempeño	Lista de cotejo con siete criterios clave que evalúan la creación del proyecto, el diseño de la interfaz, la funcionalidad del código, el análisis reflexivo y la presentación del reporte. Cada ítem se evalúa con las opciones: “Cumple / No cumple” y espacio para observaciones.
Formatos de reporte de prácticas	**Formato libre estructurado que incluya:** 1. Datos del alumno. 2. Nombre de la práctica. 3. Objetivo. 4. Capturas de pantalla del proyecto. 5. Código implementado. 6. Respuestas al análisis. 7. Conclusiones y reflexión.

NOMBRE DE LA PRÁCTICA No. 3	**Diseño de Formularios Interactivos con Controles Avanzados y Validación de Datos en Windows Forms**
COMPETENCIA DE LA PRÁCTICA	Diseñar formularios interactivos con controles visuales en Windows Forms para capturar y validar datos personales, utilizando Visual Studio y el lenguaje C# bajo el modelo de programación orientada a eventos, en el desarrollo de aplicaciones de escritorio, fomentando la creatividad y la atención al detalle.

FUNDAMENTO TÉORICO

Esta práctica se basa en el uso de Windows Forms para crear interfaces gráficas en aplicaciones de escritorio, aplicando los principios de la programación orientada a eventos. Se integran controles visuales como etiquetas, botones, cajas de texto y cuadros de imagen para facilitar la interacción usuario-aplicación. Además, se implementan técnicas de validación de datos y visualización dinámica mediante el uso de listas y el control DataGridView, fortaleciendo la lógica de programación, la experiencia del usuario y la estructuración del código en C#.

MATERIALES, EQUIPAMIENTO Y/O REACTIVOS

Equipo de cómputo

- Computadora personal o de laboratorio con sistema operativo Windows 10 o superior
- Procesador mínimo: Intel Core i3 o equivalente
- Memoria RAM mínima: 8 GB
- Espacio disponible en disco: al menos 10 GB

Software

- Visual Studio 2022 (versión Community, Professional o Enterprise) con los siguientes componentes:
  - Desarrollo de escritorio con .NET
  - Windows Forms .NET Framework o .NET 6/7, según elección del estudiante
  - .NET SDK correspondiente a la versión seleccionada (recomendado: .NET 6 o .NET 7)

Recursos digitales

- Acceso a Internet para descarga de herramientas (en caso de ser necesario)
- Carpeta asignada o unidad compartida para guardar el proyecto generado
- Cuenta de Microsoft (opcional, para sincronización de Visual Studio)

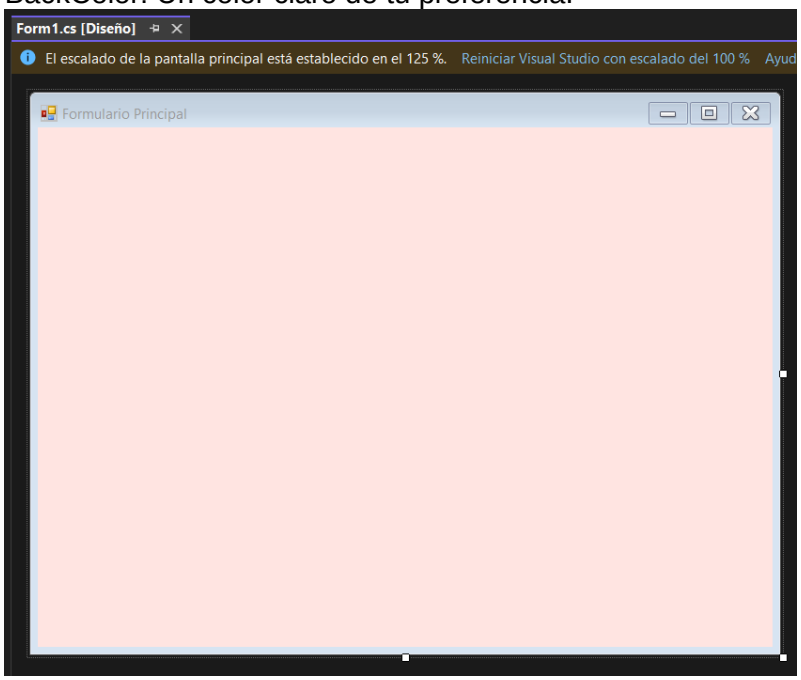
PROCEDIMIENTO O METODOLOGÍA

Objetivo:

Que los alumnos desarrollen habilidades prácticas en el diseño y creación de formularios utilizando Visual Studio y el lenguaje de programación C#. Se busca que implementen controles básicos como etiquetas, botones, cajas de texto, groupbox, picturebox, y otros elementos de diseño.

Instrucciones Generales

1. **Abrir Visual Studio:** Inicia un nuevo proyecto de tipo **Windows Forms App (.NET Framework)** en Visual Studio.
   - Nombra el proyecto: Practica_Formularios.
2. **Configurar el Formulario Principal:**
   - Cambia las propiedades del formulario principal:
     - Text: "Formulario Principal".
     - Size: 800 x 600.
     - BackColor: Un color claro de tu preferencia.



Requisitos de Diseño del Formulario:
Incluye los siguientes controles y funcionalidades:

Controles y Funcionalidades a Implementar

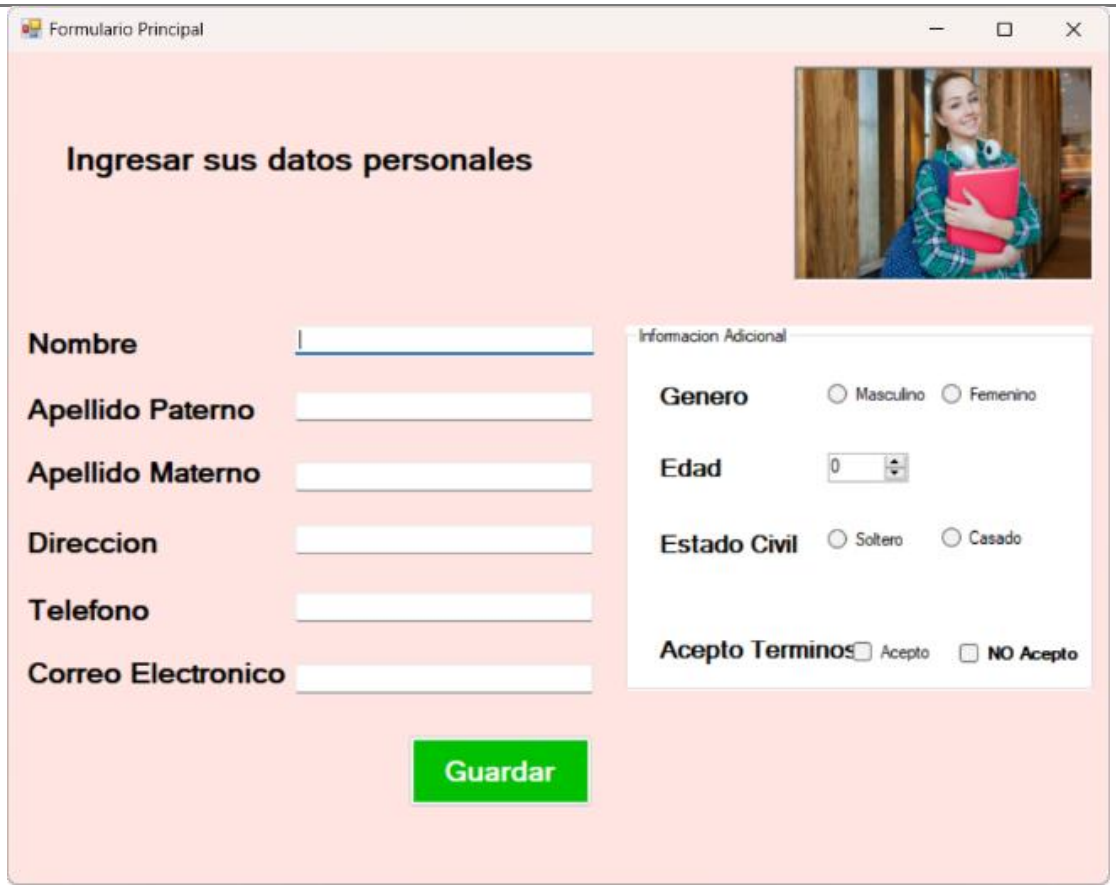
1. **Etiquetas (Label):**
   - Crea una etiqueta con el texto: "Ingrese sus datos personales".
   - Cambia las propiedades:
     - Font: Arial, Negrita, 14.
     - ForeColor: Azul oscuro.

2. **Cajas de Texto (TextBox):**
   - Agrega dos cajas de texto para capturar:
     - Nombre completo.
     - Correo electrónico.
   - Configura los PlaceholderText o utiliza etiquetas para describir cada campo.
3. **Botones (Button):**
   - Agrega un botón con el texto: "Guardar Datos".
     - Cambia las propiedades:
       - BackColor: Verde claro.
       - ForeColor: Blanco.
   - Al hacer clic en el botón, se debe mostrar un **MessageBox** que diga: "Datos guardados correctamente".

4. **GroupBox:**
   - Inserta un **GroupBox** con el título "Información Adicional".
     - Dentro del **GroupBox**, agrega controles como:
       - Una etiqueta con el texto: "Género:".
       - Dos controles de tipo **RadioButton** para seleccionar entre "Masculino" y "Femenino".
       - Una etiqueta con el texto: "Edad:" y un control **NumericUpDown** para capturar la edad del usuario.

5. **PictureBox:**
   - Inserta un control **PictureBox** para mostrar una imagen representativa.
     - Configura las propiedades:
       - SizeMode: **StretchImage**.
       - Agrega una imagen desde el explorador de archivos.
       - Coloca un borde alrededor de la imagen usando la propiedad BorderStyle.

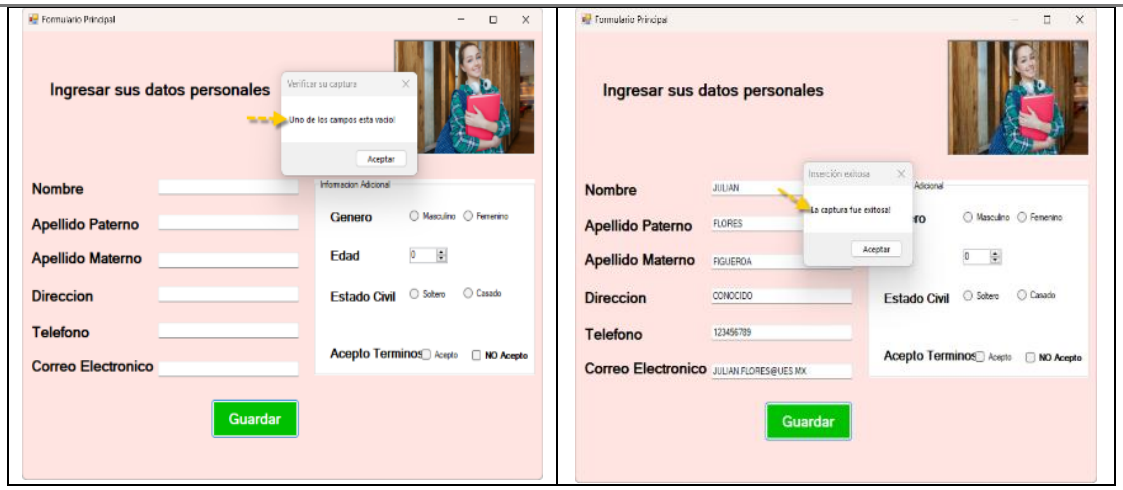
6. **Otros Controles Opcionales:**
   - **ComboBox:** Para seleccionar una opción, por ejemplo: "Estado Civil" con opciones como "Soltero", "Casado", "Otro".
   - **CheckBox:** Para agregar casillas como "Acepto términos y condiciones".



Funcionalidades Adicionales

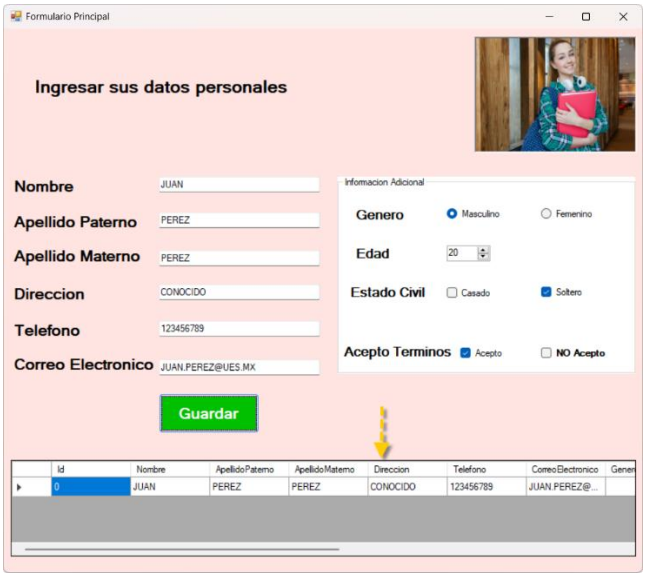
1. **Validación de Campos:**
   - Implementa validación para que no se permitan campos vacíos. Si algún campo está vacío, muestra un mensaje como:
     "Por favor, complete todos los campos.".
2. **Interactividad:**
   - Cambia el color del botón al pasar el mouse sobre él usando eventos como MouseEnter y MouseLeave.

CASO: SE VALIDAN LOS CAMPOS VACIOS	CASO: SE INGRESA EL REGISTRO



Control: DataGridView

Agregar un control DataGridView, el cual deberás configurar para que reciba los valores que has capturado



Evaluación de la Práctica

1. Diseño y distribución de los controles en el formulario.
2. Correcta implementación de las funcionalidades solicitadas.
3. Creatividad en el uso de colores y estilos del formulario.
4. Validación de los campos y mensajes de error amigables para el usuario.
5. Documentación del código (comentarios en las secciones clave).

Para guardar la información de las personas en un DataGridView, necesitas crear una lista de objetos que representen los datos y asignarla como fuente de datos. Además, en tu código actual, el

DataSource está configurado incorrectamente, ya que intentas asignar un solo objeto (nuevasPersonas) al DataGridView, en lugar de una colección.
Aquí te dejo una versión corregida del método btnGuardar_Click y los ajustes necesarios:
Ajustes en el Código

1. **Declarar una Lista de Personas**: Define una lista para almacenar las instancias de Personas en tu clase Form1.
2. **Actualizar el DataGridView**: Asigna la lista como fuente de datos al DataGridView después de agregar un nuevo objeto.
3. **Mantener la Referencia del DataSource**: Utiliza un mecanismo para que el DataGridView se actualice automáticamente al modificar la lista.

Código

```
namespace practicaFormularios
{
    public partial class Form1 : Form
    {
        // Lista para almacenar las personas
        private List<Personas> listaPersonas = new List<Personas>();

        public Form1()
        {
            InitializeComponent();
            // Configura el DataGridView para usar la lista como fuente de datos
            dgvPersonas.DataSource = new BindingList<Personas>(listaPersonas);
        }

        private void btnGuardar_Click(object sender, EventArgs e)
        {
            // Crear una nueva instancia de Personas
            Personas nuevasPersonas = new Personas
            {
                Nombre = txtNombre.Text,
                ApellidoPaterno = txtApellidoPaterno.Text,
                ApellidoMaterno = txtApellidoMaterno.Text,
                Direccion = txtDireccion.Text,
                Telefono = txtTelefono.Text,
                CorreoElectronico = txtCorreoElectronico.Text,
                GeneroMasculino = radioButtonMasculino.Checked,
                GeneroFemenino = radioButtonFemenino.Checked,
                Edad = (int)numericUpDownEdad.Value,
                EstadoCivilCasado = checkBoxCasado.Checked,
                EstadoCivilSoltero = checkBoxSoltero.Checked,
```

```
                TerminoAceptado = checkBoxAcepto.Checked,
                TerminoRechazado = checkBoxNoAcepto.Checked
            };

            // Validar la captura de datos
            BusinesLogic businesLogic = new BusinesLogic();
            if (businesLogic.ValidarCaptura(nuevasPersonas))
            {
                // Agregar el objeto a la lista
                listaPersonas.Add(nuevasPersonas);

                // Actualizar el DataGridView
                dgvPersonas.DataSource = null;
                dgvPersonas.DataSource = listaPersonas;

                // Notificar al usuario
                MessageBox.Show("La captura fue exitosa!", "Inserción exitosa");
            }
            else
            {
                // Notificar error de validación
                MessageBox.Show("Uno de los campos está vacío!", "Verificar su captura");
            }
        }
    }
}
```

RESULTADOS ESPERADOS

Al finalizar esta práctica, el estudiante será capaz de:

- Crear un formulario funcional utilizando controles visuales básicos y avanzados en Windows Forms.
- Configurar propiedades estéticas y funcionales de controles como Label, TextBox, Button, GroupBox, RadioButton, CheckBox, ComboBox, PictureBox y DataGridView.
- Implementar validación de campos para garantizar la integridad de los datos capturados.
- Gestionar la interacción con el usuario mediante eventos como Click, MouseEnter y MouseLeave.
- Almacenar y visualizar los datos capturados en un control DataGridView utilizando listas de objetos.
- Documentar el código mediante comentarios y aplicar principios de organización mediante regiones de código.
- Demostrar creatividad en la distribución, diseño visual e interactividad del formulario.

ANÁLISIS DE RESULTADOS

Responde de manera reflexiva y fundamentada las siguientes preguntas con base en la experiencia obtenida durante la práctica:

1. ¿Qué controles utilizaste para capturar y visualizar los datos? ¿Cuál fue su función específica?
2. ¿Cómo implementaste la validación de campos vacíos y qué efecto tuvo en la experiencia del usuario?
3. ¿Qué técnicas aplicaste para organizar visualmente los controles en el formulario?
4. ¿Cómo configuraste el control DataGridView para mostrar los datos capturados? ¿Tuvo el comportamiento esperado?
5. ¿Qué dificultades encontraste al trabajar con eventos como Click o MouseEnter, y cómo las resolviste?
6. ¿Qué importancia tiene la documentación del código y el uso de regiones en el mantenimiento del proyecto?
7. ¿De qué manera esta práctica te ayudó a desarrollar habilidades técnicas y de organización del trabajo?

CONCLUSIONES Y REFLEXIONES

Esta práctica permitió al estudiante aplicar conocimientos fundamentales en el diseño de interfaces gráficas utilizando Windows Forms y C#. A través de la integración de controles visuales, validación de datos y almacenamiento en listas, se fortalecieron habilidades técnicas relacionadas con la programación orientada a eventos, así como competencias transversales como la organización, la precisión en la codificación y la mejora de la experiencia del usuario.

ACTIVIDADES COMPLEMENTARIAS

1. **Incorporar funcionalidad de edición y eliminación de registros**
   - Modifica la aplicación para que los datos del DataGridView puedan seleccionarse y editarse mediante un nuevo formulario o controles habilitados. Agrega un botón para eliminar registros seleccionados.
2. **Exportar datos a un archivo externo**
   - Implementa una funcionalidad que permita guardar el contenido del DataGridView en un archivo de texto (.txt) o en formato CSV, utilizando clases de entrada/salida de archivos (StreamWriter o File.WriteAllText).
3. **Aplicar estilos personalizados al formulario**
   - Mejora la presentación de la interfaz aplicando fuentes, colores, iconos y alineaciones personalizadas. Considera adaptar el diseño para mejorar la legibilidad y accesibilidad de los controles.

EVALUACIÓN Y EVIDENCIAS DE APRENDIZAJE

Criterios de evaluación	• Diseño correcto y funcional del formulario principal y controles asociados. • Aplicación adecuada de propiedades visuales (colores, fuentes, tamaños, disposición).

	• Programación efectiva de eventos básicos (Click, MouseEnter, MouseLeave). • Implementación de validación de campos obligatorios. • Integración funcional del control DataGridView con los datos capturados. • Documentación del código con comentarios y organización por regiones. • Entrega de evidencia visual (capturas de pantalla) y análisis reflexivo en el reporte.
Rúbricas o listas de cotejo para valorar desempeño	Lista de cotejo con siete criterios clave que evalúan la creación del proyecto, el diseño de la interfaz, la funcionalidad del código, el análisis reflexivo y la presentación del reporte. Cada ítem se evalúa con las opciones: “Cumple / No cumple” y espacio para observaciones.
Formatos de reporte de prácticas	**Formato libre estructurado que incluya:** 1. Datos del alumno. 2. Nombre de la práctica. 3. Objetivo. 4. Capturas de pantalla del proyecto. 5. Código implementado. 6. Respuestas al análisis. 7. Conclusiones y Reflexión.

NOMBRE DE LA PRÁCTICA No. 4	**Desarrollo de un Sistema Empresarial Básico con Arquitectura en Capas en Windows Forms**
COMPETENCIA DE LA PRÁCTICA	Desarrollar un sistema básico de gestión empresarial con separación por capas para manejar clientes, proveedores e inventarios, mediante el uso de Visual Studio 2022 y el lenguaje C#, en un entorno de programación orientada a objetos, fortaleciendo la capacidad de análisis lógico y la organización del código.

FUNDAMENTO TÉORICO
Esta práctica se basa en la aplicación del modelo de arquitectura en capas, que permite separar responsabilidades en una aplicación: la capa de presentación (formularios), la capa de lógica del negocio (validaciones y reglas) y la capa de acceso a datos (interacción con almacenamiento). Se utilizan principios de la programación orientada a objetos como encapsulamiento, reutilización y organización modular del código, promoviendo el desarrollo estructurado y mantenible de sistemas empresariales en C# con Visual Studio.

MATERIALES, EQUIPAMIENTO Y/O REACTIVOS
Equipo de cómputo • Computadora personal o de laboratorio con sistema operativo Windows 10 o superior • Procesador mínimo: Intel Core i3 o equivalente • Memoria RAM mínima: 8 GB • Espacio disponible en disco: al menos 10 GB

Software

- Visual Studio 2022 (versión Community, Professional o Enterprise) con los siguientes componentes:
  - Desarrollo de escritorio con .NET
  - Windows Forms .NET Framework o .NET 6/7, según elección del estudiante
  - .NET SDK correspondiente a la versión seleccionada (recomendado: .NET 6 o .NET 7)

Recursos digitales

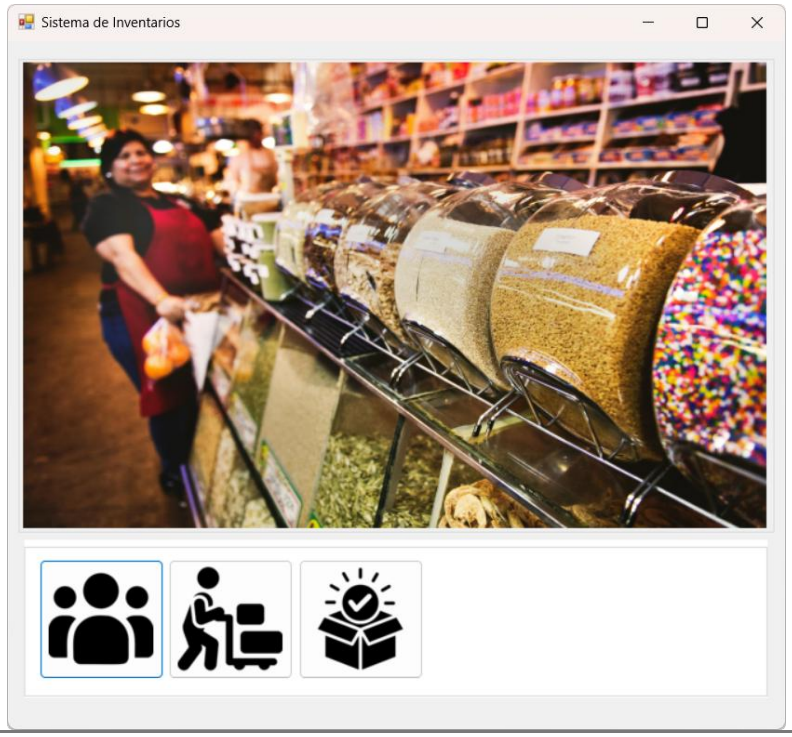
- Acceso a Internet para descarga de herramientas (en caso de ser necesario)
- Carpeta asignada o unidad compartida para guardar el proyecto generado
- Cuenta de Microsoft (opcional, para sincronización de Visual Studio)

PROCEDIMIENTO O METODOLOGÍA

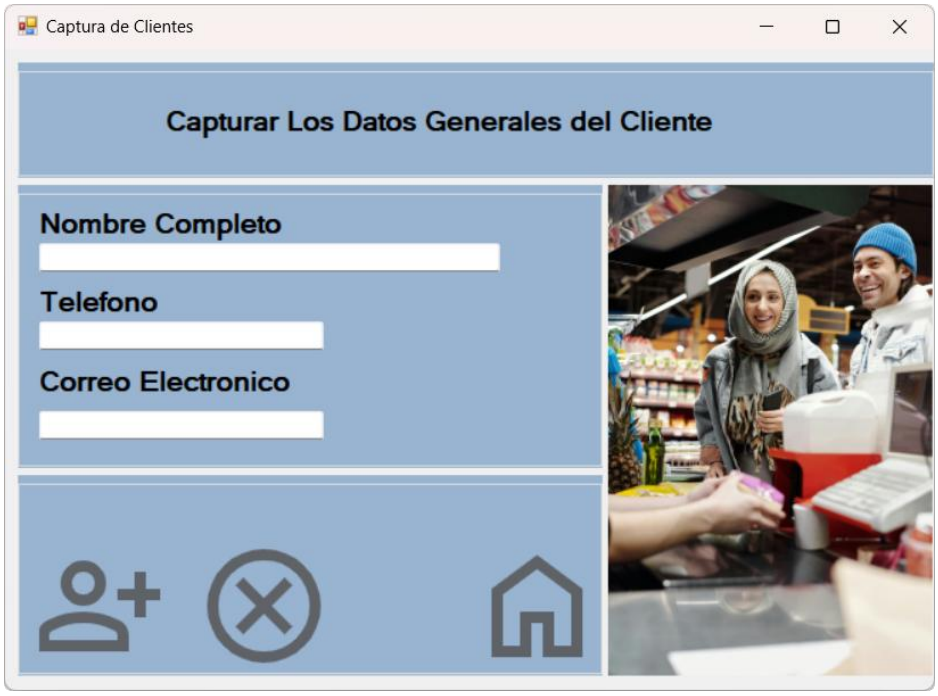
El objetivo es desarrollar un sistema básico de gestión empresarial que maneje **clientes**, **proveedores** e **inventarios** utilizando una arquitectura en capas: **capa de presentación (formularios)**, **capa de lógica del negocio (BL)** y **capa de acceso a datos (DAL)**.

Instrucciones

1. Crear un proyecto en Visual Studio 2022, con las siguientes características:
   - **Proyecto principal (WinForms)**: Actuará como la capa de presentación, en donde generaras el acceso a los tres Formularios: *Clientes, Proveedores e Inventarios*
     - **Formulario Principal**

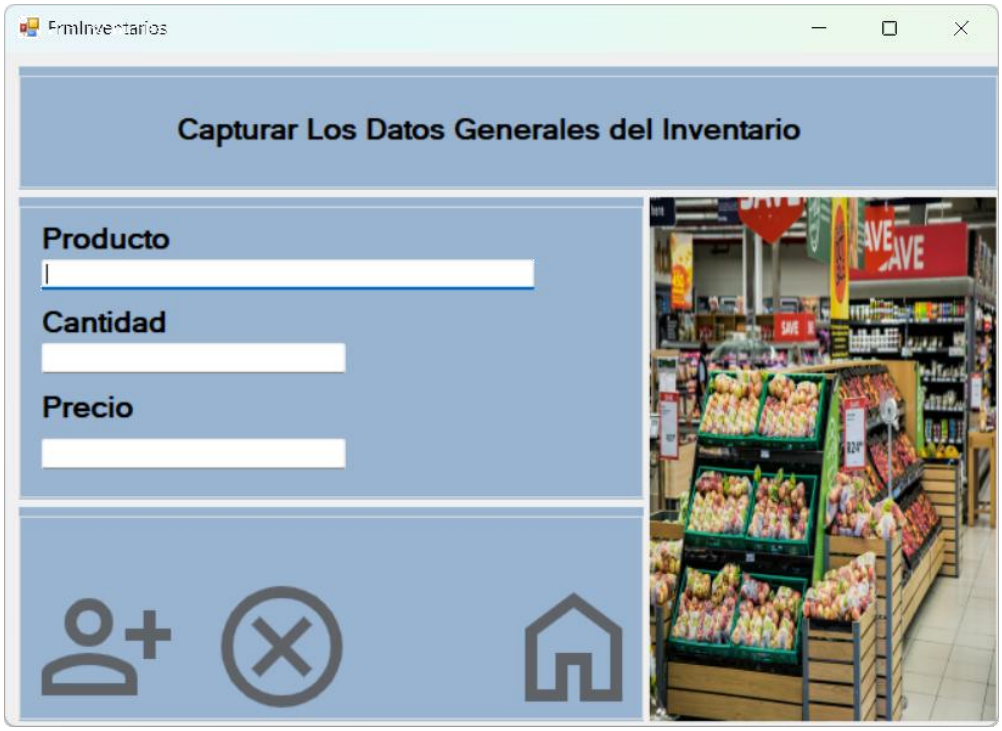


- **Formulario Clientes: FrmClientes**



- **Formulario Proveedores: FrmProveedores**

- **Formulario Inventarios: FrmInventarios**



  - **Librería de clases (BL)**: Implementará la lógica del negocio.
  - **Librería de clases (DC)**: Implementará el acceso a datos.
    - **InsercionRegistroCliente**
    - **InsercionRegistroProducto**
    - **InsercionRegistroProveedor**

2. **Diseño del Modelo de Clases:** Crear tres clases en la capa de modelo con las siguientes propiedades:
   - **Clase Clientes**:
   - Con las propiedades autoimplementadas
     - Id, NombreCompletoCliente, TelefonoCliente, CorreoCliente
   - **Proveedores:**
   - Con las propiedades autoimplementadas
     - Id, NombreCompletoProveedor, TelefonoProveedor, CorreoProveedor
   - **Inventarios:**
   - Con las propiedades autoimplementadas
     - Id, Producto, Cantidad, Precio

Validaciones, las cuales deberás realizar en la capa de la lógica del negocio (BusinesLogic)

- **Clientes**
  - El nombre es obligatorio
  - El email es obligatorio
- **Proveedores**

- El nombre es obligatorio
- **Inventarios**
  - El producto es obligatorio
  - La cantidad no puede ser negativa
  - El precio debe ser mayor a 0

RESULTADOS ESPERADOS

Al finalizar esta práctica, el estudiante será capaz de:

- Crear un proyecto en Visual Studio 2022 que implemente una arquitectura en capas (presentación, lógica del negocio y acceso a datos).
- Diseñar formularios funcionales para la gestión de clientes, proveedores e inventarios.
- Desarrollar clases de modelo con propiedades autoimplementadas para representar entidades del sistema.
- Aplicar validaciones de datos desde la capa de lógica del negocio para asegurar la integridad de la información.
- Organizar el código de forma estructurada, facilitando su mantenimiento y reutilización.
- Demostrar comprensión del flujo de datos entre formularios, lógica de negocio y persistencia.

ANÁLISIS DE RESULTADOS

Responde de forma clara y reflexiva las siguientes preguntas:

1. ¿Cómo se relacionan las capas de presentación, lógica del negocio y acceso a datos en tu proyecto?
2. ¿Qué beneficios encontraste al organizar tu código bajo una arquitectura en capas?
3. ¿Tuviste dificultades al implementar las validaciones en la capa de lógica del negocio? ¿Cómo las solucionaste?
4. ¿De qué manera aseguraste la integridad de los datos al capturar información en cada formulario?
5. ¿Cómo estructuraste las clases de modelo y qué criterios seguiste para definir sus propiedades?
6. ¿Qué aspectos mejorarías en tu diseño para una futura ampliación del sistema?
7. ¿Qué habilidades técnicas y organizativas fortaleciste durante esta práctica?

CONCLUSIONES Y REFLEXIONES

Esta práctica permitió al estudiante comprender y aplicar la estructura de una aplicación basada en arquitectura en capas, favoreciendo la separación de responsabilidades y la organización del código. Además, se fortalecieron habilidades técnicas como el diseño de formularios, la validación de datos y la construcción de clases, así como competencias blandas como la planificación, el pensamiento lógico y la claridad en la codificación.

ACTIVIDADES COMPLEMENTARIAS

1. **Agregar funcionalidad de edición y eliminación de registros**
   - Implementa botones en los formularios de Clientes, Proveedores e Inventarios que permitan

seleccionar, modificar o eliminar registros existentes. Asegúrate de mantener la coherencia entre las capas.

2. **Exportar los datos a un archivo externo**
   - Crea una función que permita guardar la información capturada en un archivo de texto o en formato CSV, utilizando clases de manejo de archivos. Esto reforzará el uso de la capa de acceso a datos.

3. **Implementar una capa adicional de servicios**
   - Agrega una capa intermedia entre la lógica del negocio y la presentación (por ejemplo, una capa de servicios o controladores) para mejorar la escalabilidad del sistema y practicar la abstracción funcional.

EVALUACIÓN Y EVIDENCIAS DE APRENDIZAJE	
Criterios de evaluación	• Creación del proyecto con arquitectura en capas correctamente estructurada. • Implementación funcional de los formularios de Clientes, Proveedores e Inventarios. • Diseño adecuado de las clases de modelo con propiedades autoimplementadas. • Aplicación efectiva de validaciones en la capa de lógica del negocio. • Integración adecuada entre las capas: presentación, lógica de negocio y acceso a datos. • Correcta captura y organización de datos desde la interfaz hasta su almacenamiento. • Documentación clara del código con comentarios y buena organización de archivos. • Presentación del reporte con evidencias, análisis reflexivo y funcionamiento del sistema.
Rúbricas o listas de cotejo para valorar desempeño	Lista de cotejo con siete criterios clave que evalúan la creación del proyecto, el diseño de la interfaz, la funcionalidad del código, el análisis reflexivo y la presentación del reporte. Cada ítem se evalúa con las opciones: “Cumple / No cumple” y espacio para observaciones.
Formatos de reporte de prácticas	**Formato libre estructurado que incluya:** 1. Datos del alumno. 2. Nombre de la práctica. 3. Objetivo. 4. Capturas de pantalla del proyecto. 5. Código implementado. 6. Respuestas al análisis. 7. Conclusiones y Reflexión

ELEMENTO DE COMPETENCIA II

NOMBRE DE LA PRÁCTICA No.5	**Diseño de Interfaces para un Sistema Escolar en Windows Forms: Organización Visual, Personalización y Control de Componentes**
COMPETENCIA DE LA PRÁCTICA	Diseñar interfaces gráficas para un sistema escolar utilizando controles visuales personalizados, con la finalidad de organizar y mejorar la experiencia del usuario, mediante formularios desarrollados en Visual Studio y el entorno Windows Forms, promoviendo la atención al detalle y la creatividad en el diseño de aplicaciones.

FUNDAMENTO TÉORICO

Esta práctica se fundamenta en el uso de Windows Forms como herramienta de desarrollo de interfaces gráficas en aplicaciones de escritorio, aplicando principios de programación orientada a eventos. Se abordan conceptos técnicos como la personalización de controles, el uso de propiedades para ajustar comportamiento y apariencia, y la configuración del diseño visual mediante anclaje, orden de tabulación y agrupación lógica. Estos elementos permiten crear formularios funcionales, adaptables y estéticamente coherentes, mejorando la usabilidad del sistema.

MATERIALES, EQUIPAMIENTO Y/O REACTIVOS

Equipo de cómputo

- Computadora personal o de laboratorio con sistema operativo Windows 10 o superior
- Procesador mínimo: Intel Core i3 o equivalente
- Memoria RAM mínima: 8 GB
- Espacio disponible en disco: al menos 10 GB

Software

- Visual Studio 2022 (versión Community, Professional o Enterprise) con los siguientes componentes:
  - Desarrollo de escritorio con .NET
  - Windows Forms .NET Framework o .NET 6/7, según elección del estudiante
  - .NET SDK correspondiente a la versión seleccionada (recomendado: .NET 6 o .NET 7)

Recursos digitales

- Acceso a Internet para descarga de herramientas (en caso de ser necesario)
- Carpeta asignada o unidad compartida para guardar el proyecto generado

Cuenta de Microsoft (opcional, para sincronización de Visual Studio)

PROCEDIMIENTO O METODOLOGÍA

DESARROLLO DE SISTEMA ESCOLAR

Propiedades del Formulario

Las propiedades son configuraciones que modifican la apariencia y el comportamiento de un formulario. Se acceden desde la **ventana de Propiedades** en Visual Studio.

Propiedades Importantes

- **Text:** Define el título que aparece en la barra superior del formulario.
  - **Ejemplo:** Cambiar **"Form1"** a **"**Sistema de Estudiantes**".**
- **BackColor:** Cambia el color de fondo del formulario.
  - **Ejemplo:** Establecer un color **"**Web**"** para una apariencia personalizada.
- **StartPosition:** Ubica el formulario en el centro de la pantalla al iniciar.
  - Valor recomendado: **CenterScreen**.
- **Size:** Ajusta el ancho y alto del formulario.
  - **Ejemplo: Width =** 900**, Height =** 700**.**
- **FormBorderStyle:** Controla el estilo del borde del formulario (fijo, redimensionable, sin bordes, etc.).

Analogía: Las propiedades son como los ajustes de un automóvil: puedes cambiar el color, el tamaño de los espejos o cómo se abre una puerta, según tus necesidades.

Controles en Formularios de .NET Framework

Un control es un componente visual que se agrega a un formulario para interactuar con el usuario. Algunos controles comunes incluyen botones, cuadros de texto y contenedores como el **GroupBox**.

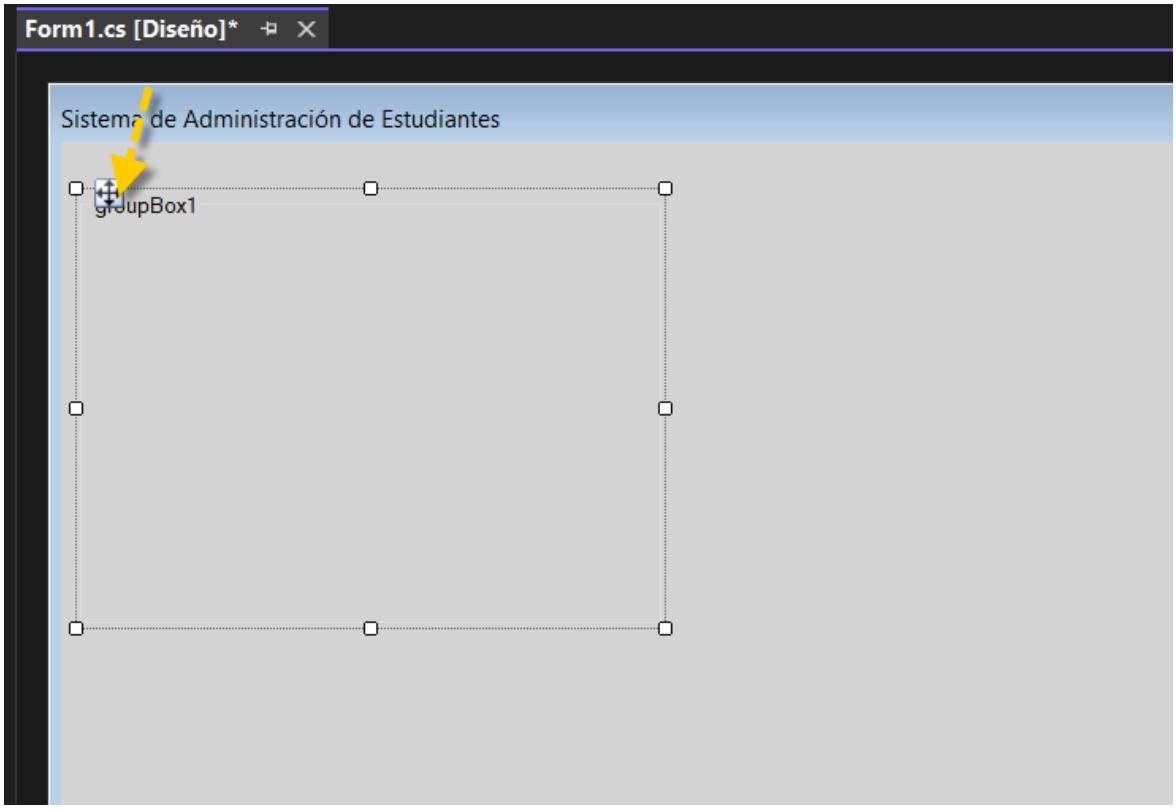
El Control GroupBox

El **GroupBox** es un contenedor que agrupa controles relacionados, lo que mejora la organización y claridad de la interfaz de usuario. En este Apartado del curso, el control **GroupBox** se usa para:

- Agrupar elementos en el formulario.
- Personalizar su apariencia mediante propiedades como **Text** y **BackColor**.

Agregar un GroupBox al Formulario

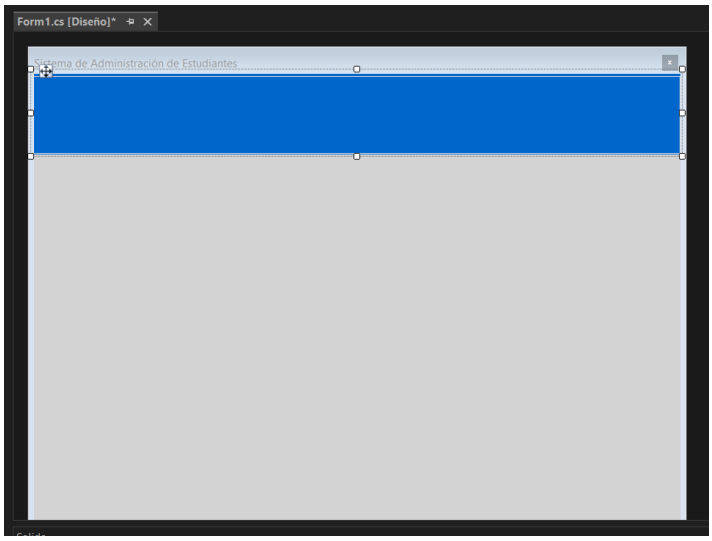
1. **Abrir el Cuadro de Herramientas**: El cuadro de herramientas contiene todos los controles disponibles. Puedes buscar **"GroupBox"** para localizarlo rápidamente.
2. **Arrastrar y Soltar**: Selecciona el control **GroupBox**, arrástralo y suéltalo en el formulario.
3. **Posicionarlo y Redimensionarlo**: Ajusta su tamaño y ubicación para que se adapte al diseño deseado.



Configurar Propiedades Del GroupBox

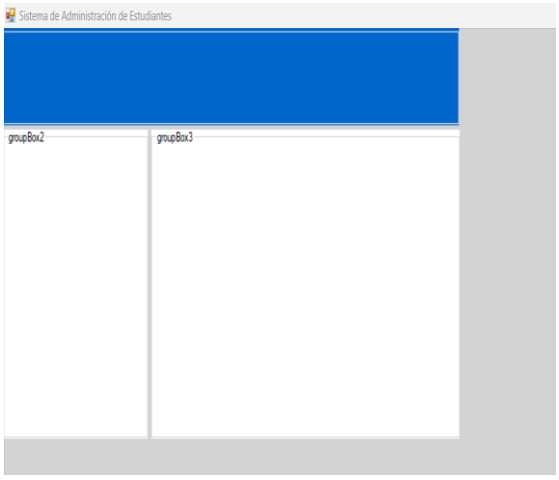
- Text: Cambia o elimina el texto predeterminado que aparece en la parte superior del **GroupBox**.
    - **Ejemplo:** Si no deseas mostrar texto, deja el valor en blanco.

- BackColor: Cambia el color de fondo para mejorar la estética.
    - **Ejemplo:** Selecciona un color desde la propiedad **BackColor** en el panel de propiedades.
- Anchor: Permite que el control se ajuste automáticamente al tamaño del formulario.
    - Configuración inicial: El control suele anclarse a la parte superior e izquierda.
    - **Configuración personalizada:** Anclarlo a los lados derecho e inferior permite que se ajuste al redimensionar el formulario.



Agregar Múltiples GroupBox

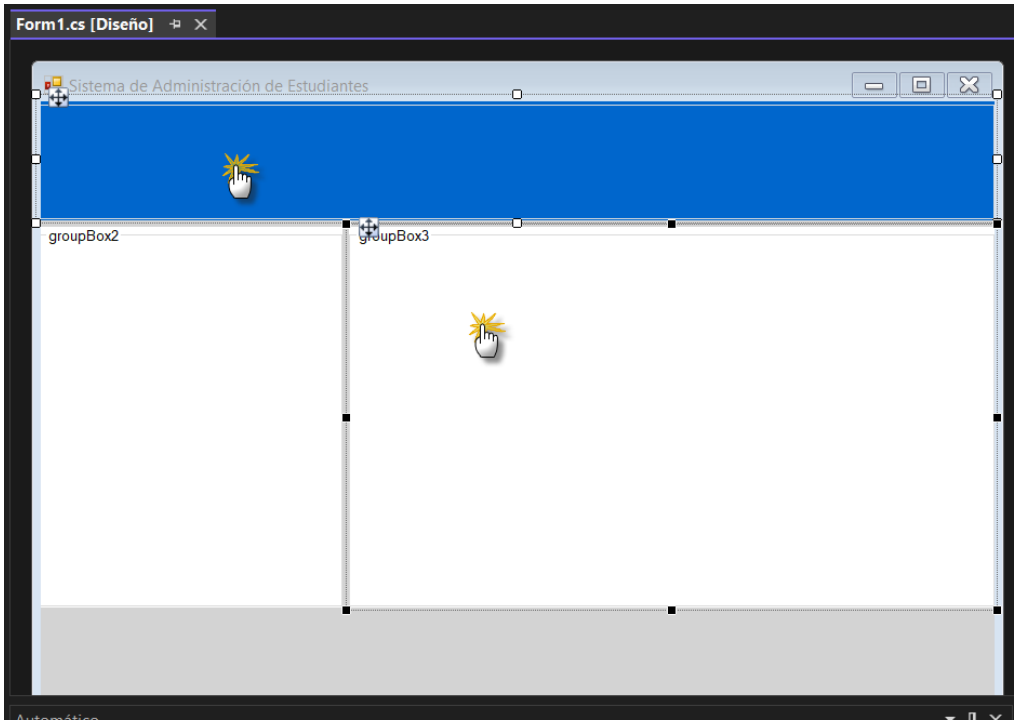
1. Crea otros dos controles de tipo **GroupBox** y posiciona uno seguido del otro debajo del que colocamos en la parte superior.
2. Ajusta sus dimensiones para que ocupen espacios específicos en el formulario.
3. Configura colores distintivos para diferenciar visualmente cada grupo.



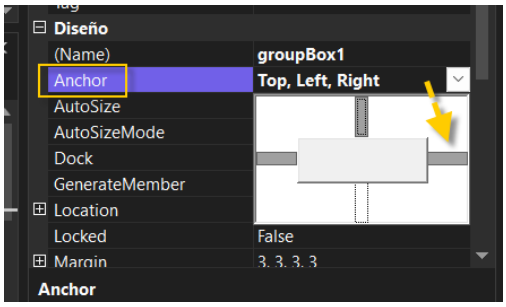
Ajustar Controles al Formulario Principal

Usar la propiedad **Anchor** para que los controles se adapten al formulario padre:

1. Selecciona el **GroupBox**.

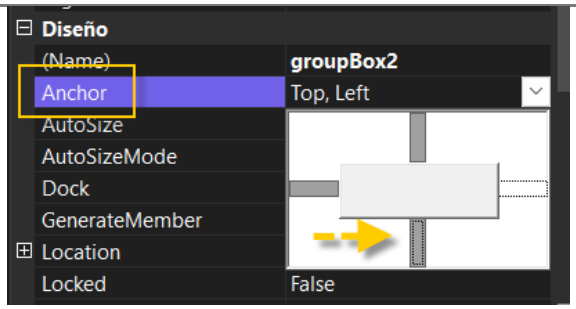


2. Configura Anchor para que incluya las posiciones deseadas (superior, inferior, izquierda, derecha).

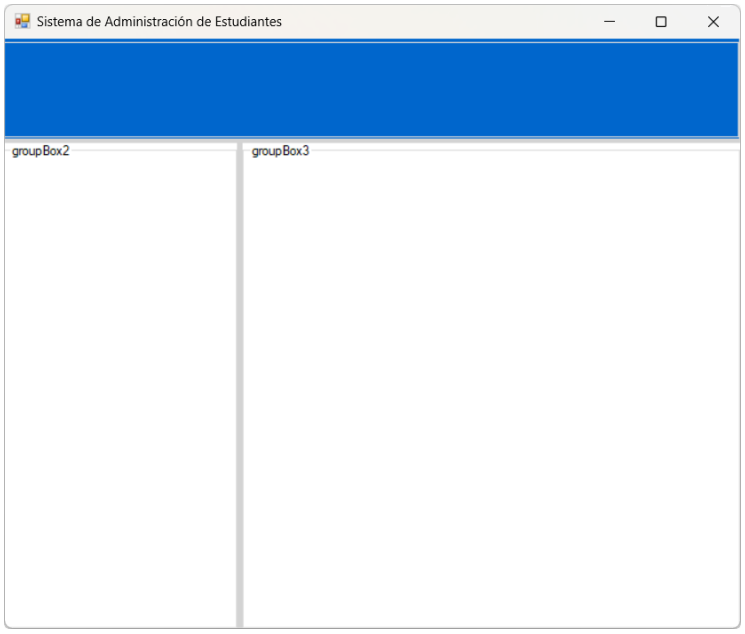


3. Verifica el comportamiento ejecutando la aplicación y maximizando el formulario.

4. También es importante hacer el ajuste para que el control se adapte de manera vertical



5. Verifica el comportamiento ejecutando la aplicación y maximizando el formulario.



Agregar y Configurar Etiquetas (Labels)

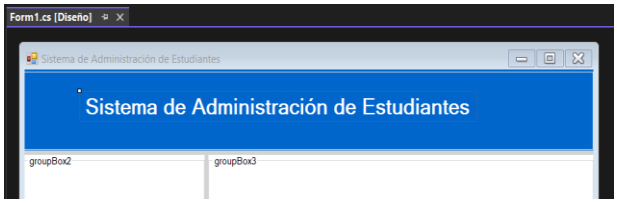
Paso 1: Agregar un Label

1. Selecciona el control **Label** del cuadro de herramientas y arrástralo dentro del **GroupBox**.
2. Ajusta su posición y tamaño.

Paso 2: Personalizar la Etiqueta

Configura las propiedades en el panel:

- **Text**: Define el texto que se mostrará, como **"Sistema de Estudiantes".**
- **Font**: Haz clic en el botón de fuente para cambiar el estilo (**negrita**, **cursiva**) y el tamaño (por ejemplo, **16**).
- **ForeColor**: Cambia el color del texto (por ejemplo, blanco).



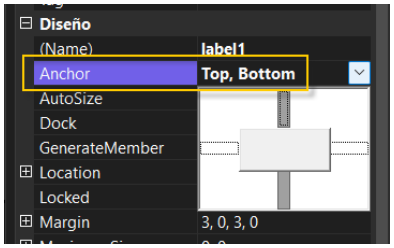
Ejemplo práctico: Si estás diseñando un formulario para una universidad, podrías mostrar encabezados como **"Lista de Estudiantes"** o **"Información del Estudiante"**.

Personalización y Adaptación de Controles

La Propiedad Anchor

La propiedad **Anchor** permite que un control se adapte automáticamente al tamaño de su contenedor principal. Esto es útil para garantizar que los controles se vean bien en pantallas de diferentes tamaños.

1. Selecciona el **Label** y configura su propiedad **Anchor** en el panel:
    - Por ejemplo, selecciona **Top** y **Bottom** para que el texto se mantenga centrado verticalmente.

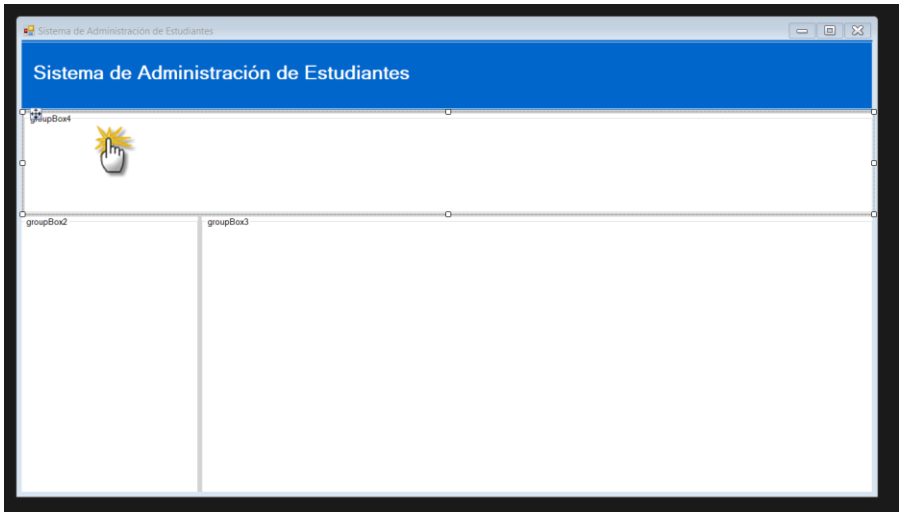


Caso de uso: Imagina que el usuario redimensiona la ventana de la aplicación. Configurar correctamente la propiedad **Anchor** asegura que los controles mantengan su diseño ordenado.

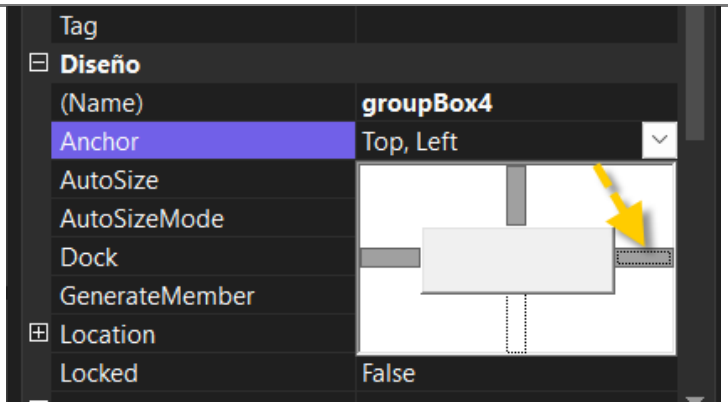
Añadir Nuevos Controles y Agrupaciones

Paso 1: Añadir Controles Secundarios

- Agrega otro **GroupBox** al formulario. Este contendrá controles como **Labels** adicionales o campos de entrada.
- Ajusta el tamaño del formulario si es necesario para acomodar los nuevos elementos.



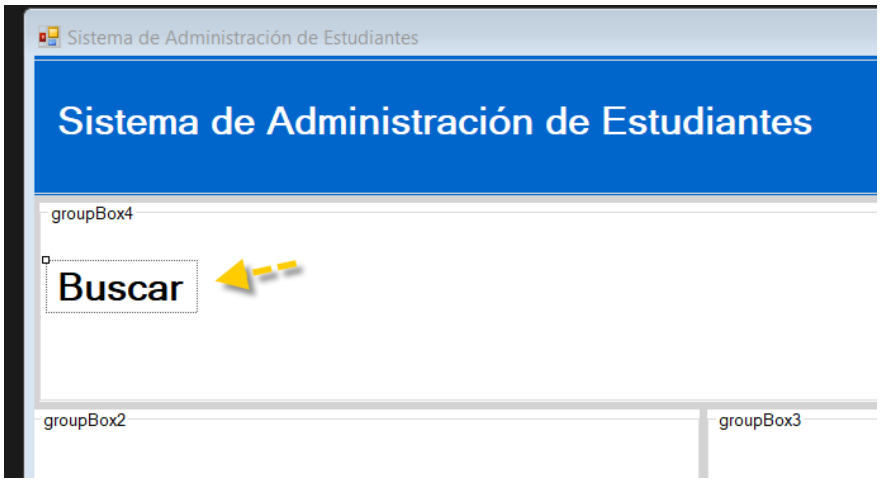
- Ajusta la propiedad **anchor**



Paso 2: Copiar y Pegar Controles

Puedes duplicar controles existentes:

- Copia y pega un **Label**.
- Cambia su propiedad **Text** a algo nuevo, como **"Buscar"**



Añadir Nuevos Controles y Agrupaciones

Paso 1: Añadir Controles Secundarios

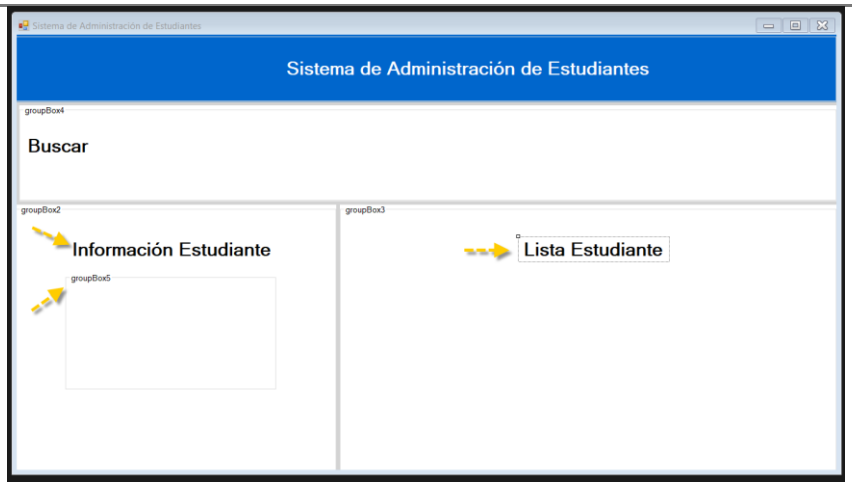
- Agrega otro **GroupBox** al formulario. Este contendrá controles como **Labels** adicionales o campos de entrada.

1. Ajusta el tamaño del formulario si es necesario para acomodar los nuevos elementos.

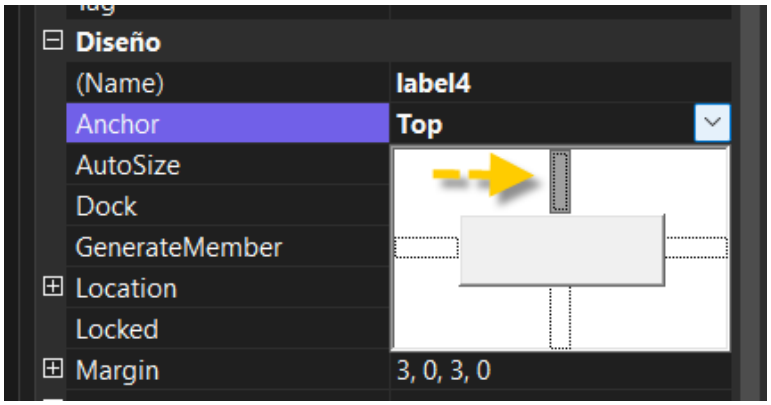
Paso 2: Copiar y Pegar Controles

Puedes duplicar controles existentes:

- Copia y pega un **Label**.
- Cambia su propiedad **Text** a algo nuevo, como **"Información" y "Lista de Estudiantes"**.

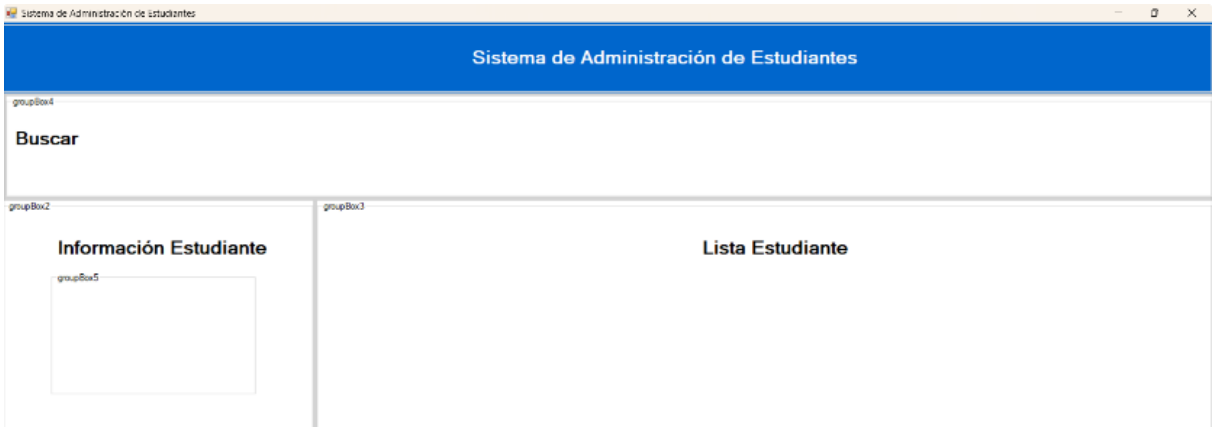


- Ajusta la propiedad **anchor,** de las etiquetas que has colocado en el formulario, solamente adaptándolas a la parte superior.



Pruebas y Ejecución

1. Guarda tu proyecto y ejecuta la aplicación (**Ctrl + F5**).
2. Interactúa con los controles para asegurarte de que se visualizan correctamente y se adaptan al tamaño del formulario.



Errores comunes:

- Los controles no se centran correctamente.
- Los colores o fuentes no son consistentes.
- La interfaz no responde al redimensionar la ventana.

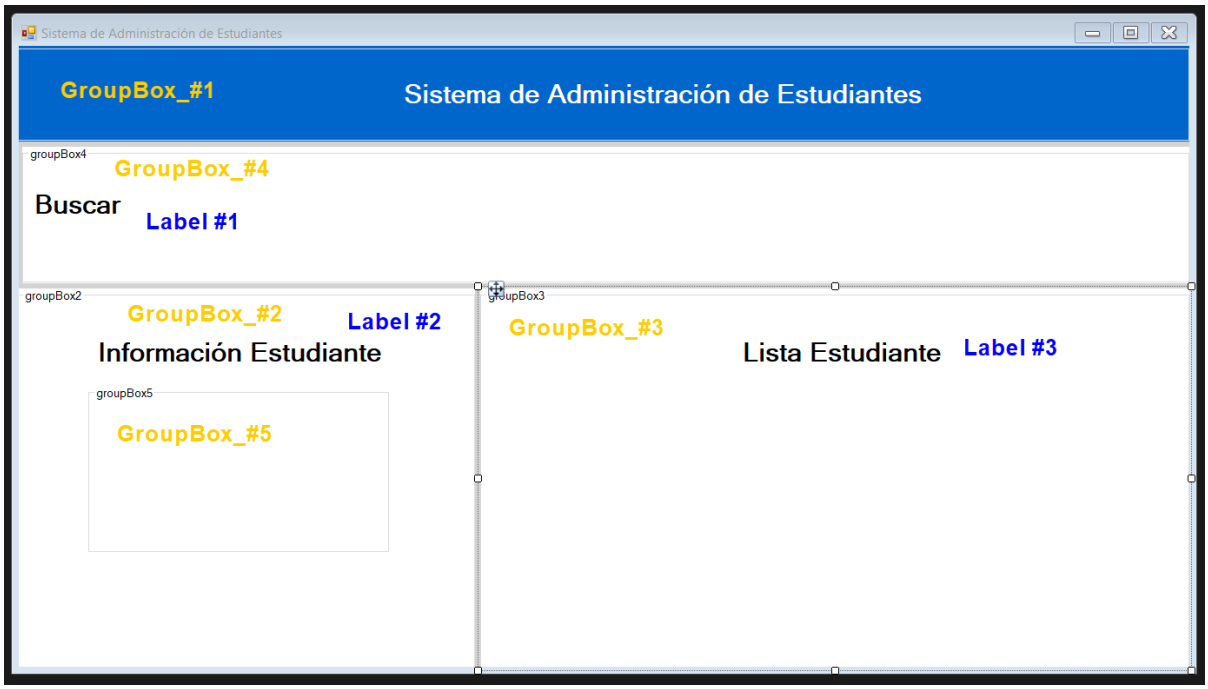
Solución: Ajusta las propiedades de cada control y vuelve a probar.

Ejemplo Completo de Interfaz

Tu interfaz debe incluir:

- 5 controles de tipo **GroupBox** principal para agrupar controles relacionados.
- Varios **Labels** configurados con texto claro, fuentes legibles y colores consistentes.

3. Controles bien posicionados que se adapten automáticamente al tamaño del formulario.

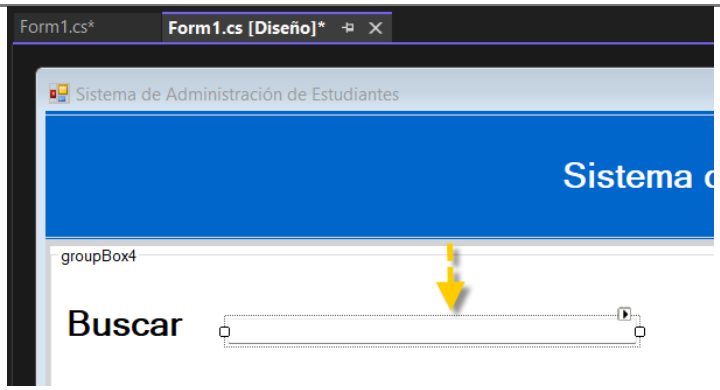


Crear y Agregar un Campo de Texto

1. **Abrir el Formulario**
   - En tu proyecto de .NET, abre el formulario principal (por ejemplo, Form1).
2. **Agregar un Campo de Texto**

- Desde la **Caja de herramientas** (**Toolbox**), arrastra un control **TextBox** al formulario.
- Suelta el control en la ubicación deseada dentro del formulario o dentro de un **GroupBox** si estás organizando los elementos.

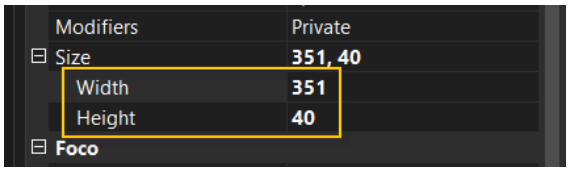
3. **Configuración Inicial del Campo de Texto**
   - Selecciona el control y accede a la **Ventana de propiedades**. Aquí puedes personalizar su apariencia y comportamiento.



Personalización del Campo de Texto

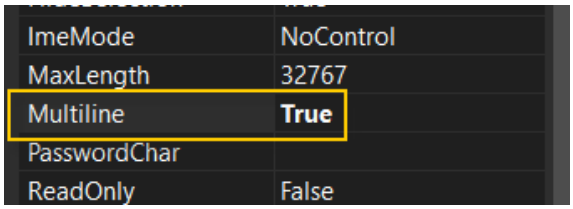
a. Cambiar el Tamaño

- Ajusta el tamaño del campo de texto directamente en el formulario arrastrando las esquinas.
- Alternativamente, modifica las propiedades de tamaño (**Width** y **Height**) en la ventana de propiedades.



b. Activar Modo Multilínea

- Habilita la propiedad Multiline para permitir la entrada de varias líneas de texto.
- En la **Ventana de propiedades**, busca **Multiline** y selecciona **True**.
- Esto es útil para notas, descripciones largas, o datos que requieren más espacio.



c. Cambiar el Nombre del Control

- En la propiedad Name, cambia el nombre predeterminado (**textBox1**) a uno significativo, como **textBoxBuscar**, para identificarlo fácilmente en el código.
- Ejemplo: Usa **textBoxNombre** para un campo que captura nombres.

Agregar Etiquetas para los Campos de Texto

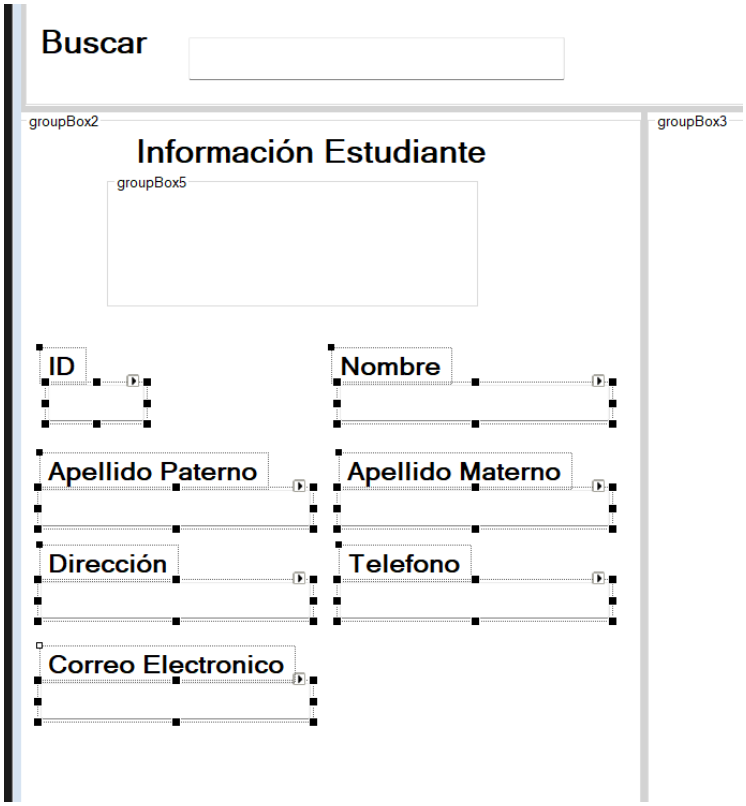
Las etiquetas (**Label**) son textos descriptivos que ayudan al usuario a entender la finalidad de cada campo.

1. Arrastra un control **Label** desde la Caja de herramientas.
2. Coloca la etiqueta encima o al lado del campo de texto.
3. Cambia su propiedad **Text** para indicar la función del campo (por ejemplo, **"Nombre", "Apellido", "Correo Electrónico"**).
4. Cambia la propiedad Name, para colocar de acuerdo al valor que recibe el indicativo relacionado, ejemplo: **Nombre,** tendrá en su propiedad Name **txtNombre**

Crear Varios Campos

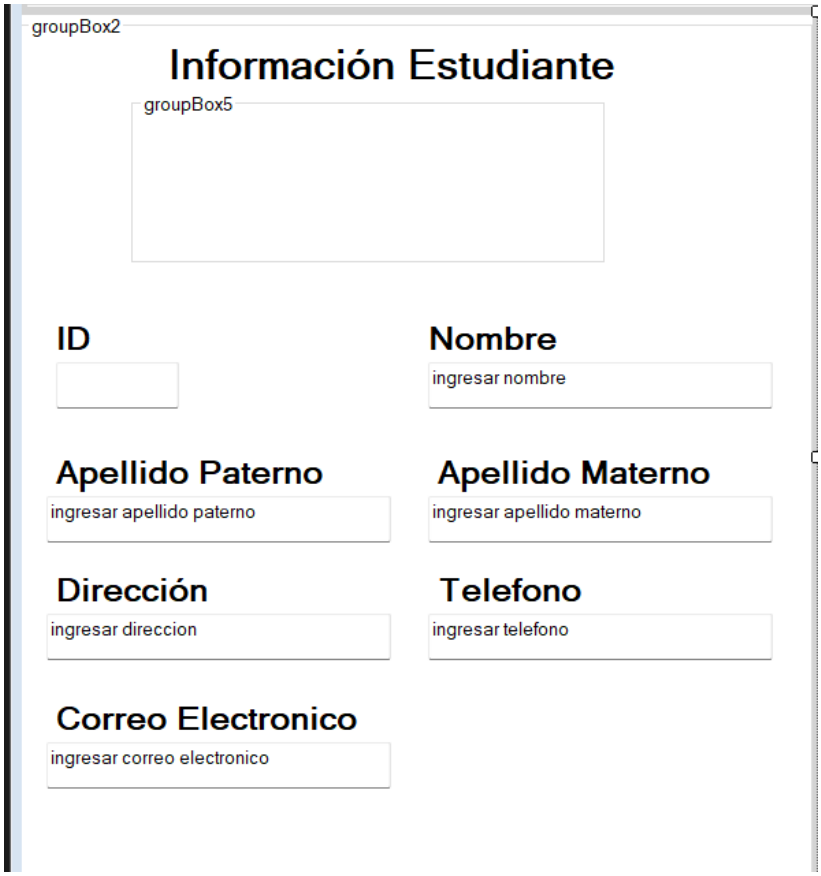
Imagina que estás creando un formulario de registro. Aquí tienes una lista de campos básicos que podrías configurar:

1. **Campo de Texto para Nombre**
   - **Etiqueta**: Nombre
   - TextBox: Cambiar Name a textBoxNombre.
2. **Campo de Texto para Apellido**
   - **Etiqueta**: Apellido
   - TextBox: Cambiar Name a textBoxApellido.
3. **Campo de Texto para Número de Identidad**
   - **Etiqueta**: Número de Identidad
   - TextBox: Cambiar Name a textBoxNumeroIdentidad.
4. **Campo de Texto para Correo Electrónico**
   - **Etiqueta**: Email
   - TextBox: Cambiar Name a textBoxEmail.

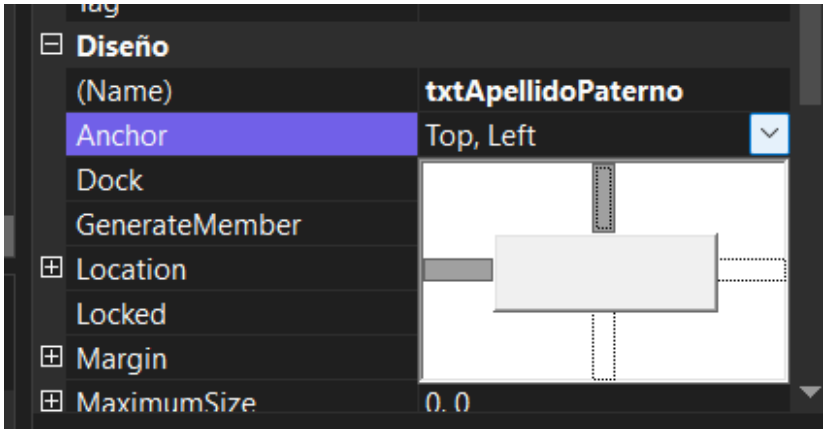


Propiedades Adicionales Útiles

- **Fuente (**Font**)**: Cambia el estilo y tamaño del texto mostrado en el campo. Ejemplo: Usa una fuente más grande para mejorar la legibilidad.
- **Texto Inicial (**Text**)**: Define un texto predeterminado.
- Ejemplo: "Ingrese su nombre aquí".



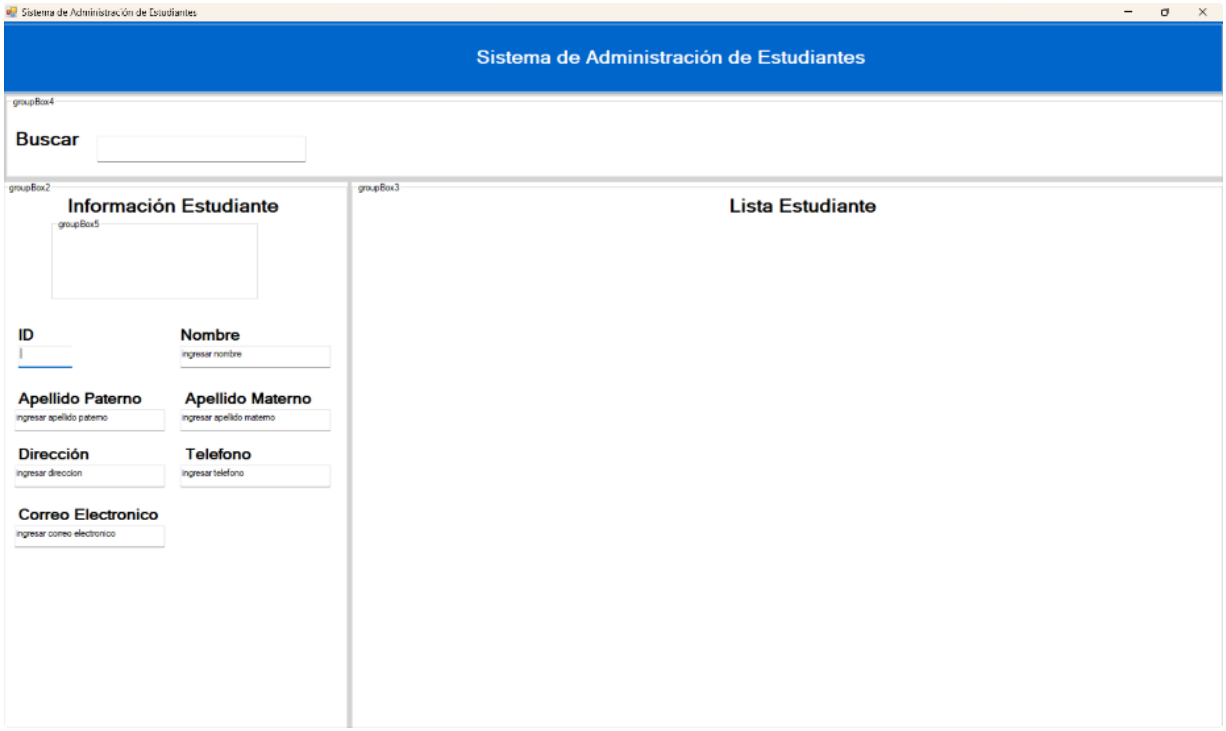
- **Anclaje (**Anchor**)**: Permite que el campo de texto se ajuste al tamaño de la ventana cuando el usuario redimensiona la aplicación.



Ejecución y Pruebas

Ejecutar la Aplicación

- Presiona F5 para iniciar tu aplicación.
- Interactúa con los campos de texto para verificar su funcionamiento.



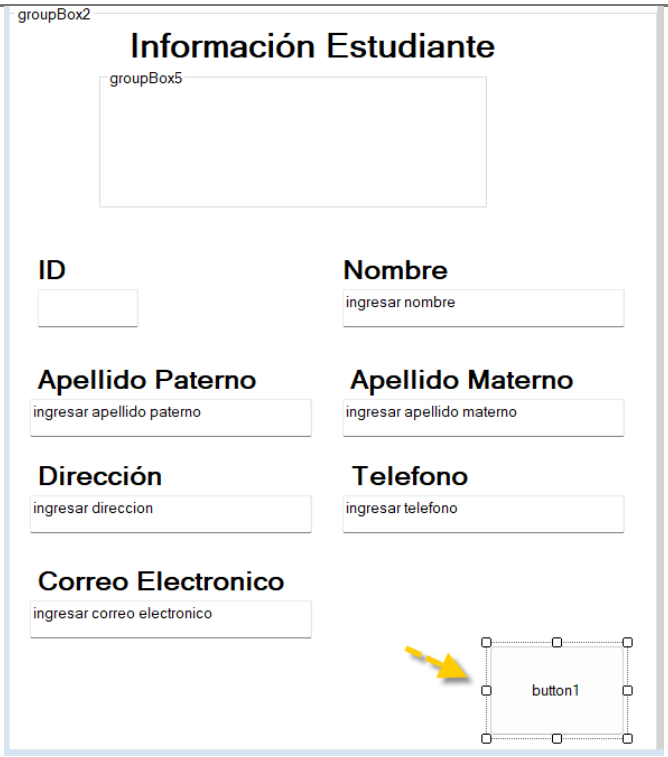
Controles de tipo Button

En este caso, trabajaremos con botones, uno de los controles más comunes.

Ejemplo Práctico: Imagina que estás desarrollando una aplicación para gestionar inventarios. Cada formulario puede tener botones como "Agregar", "Eliminar" o "Guardar", diseñados para ejecutar funciones específicas.

Agregar un Botón a un Formulario

1. **Abrir el cuadro de herramientas:** Busca el control de tipo botón.
2. **Arrastrar y soltar:** Coloca el botón en el formulario dentro de un contenedor como un GroupBox.
3. **Redimensionar y ajustar:** Selecciona el botón y modifica su tamaño para que se ajuste al diseño.



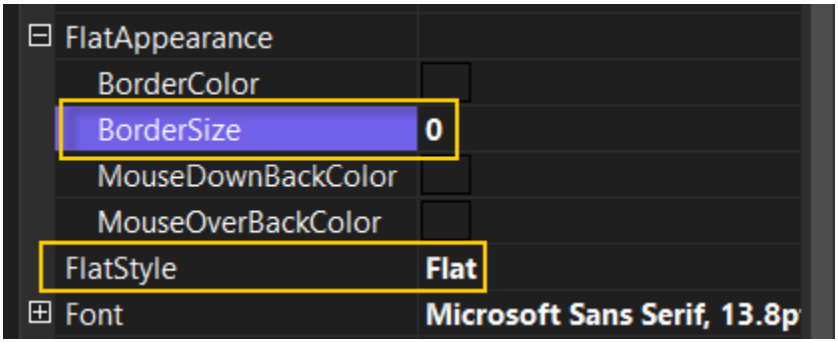
Analogía: Piensa en el botón como un interruptor en un panel de control. Puedes cambiar su tamaño y ubicación para que sea accesible y práctico.

Configuración de Propiedades del Botón

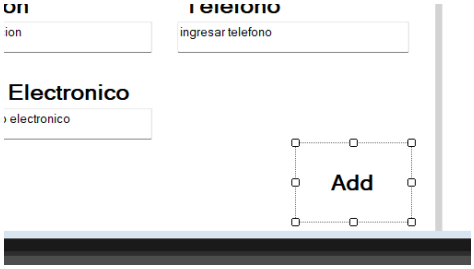
Cada control en un formulario tiene propiedades que puedes modificar para personalizar su comportamiento y apariencia.

Propiedades Básicas:

- **Texto (**Text**):** Define lo que se muestra en el botón. Por ejemplo, cambia el texto a **"Agregar"** para indicar su función.
- **Name:** Ajusta la propiedad de acuerdo a la acción, ejemplo: btnGuardar, btnAgregar
- **Apariencia (**FlatStyle**):** Cambia el estilo del botón a Flat para un diseño más moderno.
- **Bordes (**BorderSize**):** Ajusta esta propiedad a 0 si prefieres un botón sin bordes.



Obtendremos el siguiente resultado

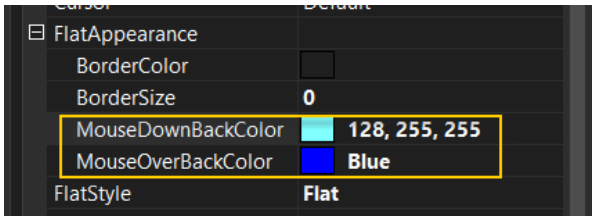


Ejemplo Práctico: Modifica el texto a "Cancelar" y ajusta el estilo para que el botón tenga un diseño plano y minimalista.

Personalización de Colores

Un aspecto importante del diseño de botones es cómo cambian de color en diferentes estados:

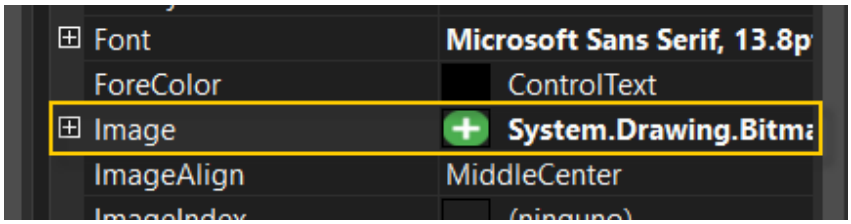
- **Color predeterminado:** Se muestra cuando el botón no está interactuando.
- **Color al hacer clic:** Cambia el color para indicar que el botón fue presionado.
- **Color al pasar el mouse:** Define un color diferente para cuando el cursor se encuentra sobre el botón.



Caso Real: En una aplicación bancaria, el botón **"**Confirmar**"** puede ser verde por defecto, cambiar a azul cuando el usuario pase el mouse y volverse gris al ser presionado.

Agregar Imágenes e Íconos a un Botón

1. **Eliminar texto:** Si el botón mostrará solo un ícono, elimina el texto configurando la propiedad Text como vacía.
2. **Seleccionar una imagen:** Usa la propiedad Image para cargar un archivo desde tu computadora.
3. **Ajustar dimensiones:** Asegúrate de que el ícono tenga un tamaño apropiado para el botón (ej., **28x28 píxeles**).



Página de Recursos:

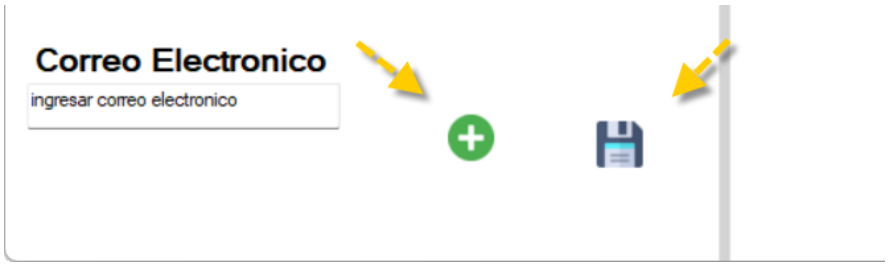
- Visita sitios como Flaticon para descargar íconos gratuitos y profesionales: https://www.flaticon.com/
- Visita la siguiente página para descargar iconos: https://fonts.google.com/icons
- Vista el sitio para redimensionar en pixeles: https://www.resizepixel.com/es

Obtendremos el siguiente resultado

Ejemplo: Descarga un ícono de **"Agregar"** en formato **PNG**, impórtalo a tu proyecto y aplícalo a un botón para representar gráficamente su función.

Propiedades Avanzadas

- **Nombre del control:** Usa nombres descriptivos como **btnAgregar** o **btnCancelar** para identificar cada botón en el código.
- **Cursor:** Cambia el cursor a una mano (Hand) para mejorar la experiencia de usuario al pasar sobre el botón.



Ejemplo Práctico: Un botón llamado **btnGuardar** puede tener un ícono de disquete, cambiar su cursor a una mano y mostrar el texto "**Guardar**".

Organización de Botones en el Formulario

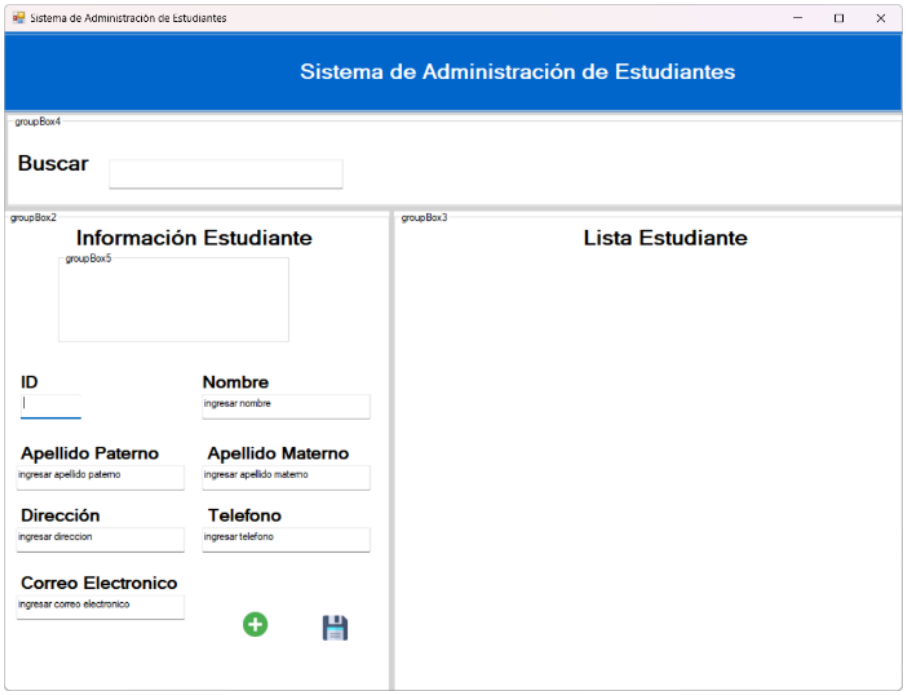
1. **Copiar y pegar:** Si necesitas múltiples botones, duplica uno existente.
2. **Alinear:** Asegúrate de que los botones estén alineados correctamente para mantener un diseño limpio.

   Tip: Utiliza guías de alineación o propiedades como Anchor y Dock para mantener consistencia en el diseño.

Guardar y Ejecutar

Después de personalizar los botones, guarda el proyecto y ejecuta la aplicación para probar los cambios:

- Asegúrate de que los colores cambien correctamente al interactuar.
- Verifica que los íconos se muestren en alta calidad.
- Confirma que los nombres y eventos asociados funcionen según lo esperado.



Personalización de controles

Cambiar íconos de botones

1. Selecciona un botón.
2. En la ventana de **Propiedades**, localiza la propiedad Image.
3. Haz clic en el botón con los tres puntos (...) y selecciona un ícono previamente descargado.
    - **Recomendación:** Usa íconos de dimensiones **28x28** para un mejor ajuste.
    - **Logramos un total de tres iconos, Agregar, Guardar y Eliminar**

Uso del control PictureBox

El **PictureBox** permite mostrar imágenes en la aplicación.

Agregar un PictureBox

1. Arrastra un control **PictureBox** al formulario.
2. Ajusta su tamaño y posición.
3. En la propiedad Image, selecciona una imagen desde tu PC.
4. Cambia la propiedad SizeMode para ajustar cómo se muestra la imagen:
    - **Normal:** Muestra la imagen en su tamaño original.
    - **StretchImage:** Adapta la imagen al tamaño del PictureBox.
    - **AutoSize:** Ajusta el tamaño del PictureBox al tamaño de la imagen.

    Caso de uso:
    - Usa PictureBox para mostrar fotos de perfil de usuarios o logotipos del sistema.

    Ejemplo práctico
    - Carga un logotipo en el PictureBox.
    - Cambia la propiedad BorderStyle a FixedSingle para agregar un borde.
    - Configura la propiedad Cursor como Hand para indicar que es interactivo.

Página de Recursos:

- Visita la siguiente página para obtener imágenes: https://www.pexels.com/es-es/

Logrando un resultado como el siguiente

Configuración del TabIndex

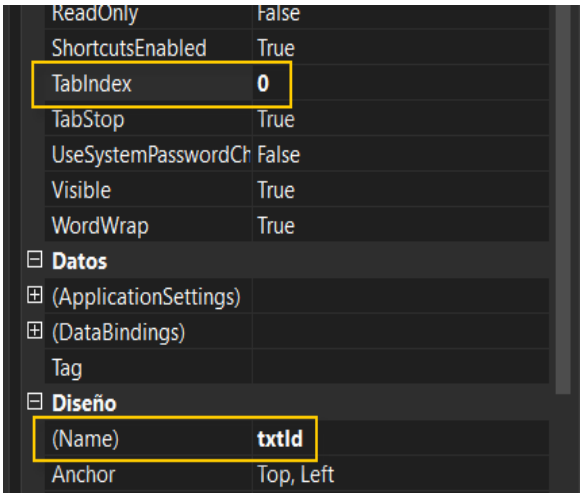
El **TabIndex** define el orden en el que los controles reciben el foco al usar la tecla Tab.

Configuración del TabIndex

1. Selecciona un control (por ejemplo, un cuadro de texto).
2. En la propiedad TabIndex, establece un valor numérico que indique su posición en el orden de tabulación.
    - Ejemplo: El primer campo de texto tiene TabIndex = 0, el segundo TabIndex = 1, y así sucesivamente.

    Ejemplo práctico

    - Para un formulario con campos como Nombre, Apellido, y Email:
        - Establece **TabIndex = 0** para ID.
        - Establece **TabIndex = 1** para Nombre.
        - Establece **TabIndex = 2** para Apellido Paterno.
        - Establece **TabIndex = 3** para Apellido Materno.
        - Establece **TabIndex = 4** para Direccion.
        - Establece **TabIndex = 5** para Telefono.
        - Establece **TabIndex = 6** para Correo.



Ejecución y pruebas

Verifica tu diseño

1. Haz clic en **Ejecutar (F5)** para compilar y ejecutar la aplicación.
2. Prueba las interacciones:
    - Asegúrate de que los botones y campos funcionen correctamente.
    - Verifica que las imágenes se muestren como esperas.

    Ajustes finales

    - Si los controles no se visualizan correctamente, ajusta sus propiedades (posición, tamaño, etc.).
    - Realiza pruebas de navegación con la tecla Tab para asegurarte de que el orden es el correcto.

RESULTADOS ESPERADOS

Al finalizar esta práctica, el estudiante será capaz de:

- Configurar propiedades esenciales de formularios y controles para mejorar la presentación e interacción en una aplicación de escritorio.
- Diseñar una interfaz organizada mediante el uso adecuado de contenedores como GroupBox y controles como Label, TextBox, Button y PictureBox.
- Aplicar personalización visual coherente (colores, fuentes, íconos) para mejorar la experiencia del usuario.
- Utilizar propiedades como Anchor y TabIndex para lograr una interfaz adaptable y accesible.
- Probar, ajustar y validar el diseño visual y funcional de formularios en un sistema escolar simulado.

ANÁLISIS DE RESULTADOS

- ¿Las propiedades configuradas en los formularios permitieron una correcta visualización y adaptación de los controles al redimensionar la ventana?
- ¿Qué controles fueron más útiles para organizar visualmente los datos dentro del formulario escolar?
- ¿Se respetó el orden de tabulación y la coherencia visual en la distribución de los elementos?
- ¿Cómo influyó el uso de colores, fuentes e íconos en la claridad y estética de la interfaz diseñada?
- ¿Qué dificultades se presentaron al personalizar los controles y cómo se resolvieron?
- ¿Qué mejoras podrías implementar en una siguiente versión del formulario escolar?

CONCLUSIONES Y REFLEXIONES

La práctica permitió fortalecer habilidades en el diseño visual de formularios, enfocándose en la organización, personalización y adaptación de controles en un sistema escolar. Se comprendió la importancia de aplicar propiedades correctamente para mejorar la experiencia del usuario y garantizar interfaces funcionales, estéticas y coherentes. Además, se promovió la atención al detalle y la lógica en la disposición de elementos visuales.

ACTIVIDADES COMPLEMENTARIAS

1. **Diseñar una pantalla de inicio personalizada** que incluya logotipo institucional, botones de navegación y un mensaje de bienvenida utilizando controles como Label, PictureBox y Button.
2. **Crear un formulario de búsqueda de estudiantes** que integre un TextBox para ingresar criterios, un Button para ejecutar la búsqueda y un DataGridView para mostrar resultados simulados.
3. **Elaborar una interfaz adaptativa para tabletas o pantallas pequeñas**, configurando adecuadamente las propiedades Anchor y Dock para todos los controles utilizados.

EVALUACIÓN Y EVIDENCIAS DE APRENDIZAJE

Criterios de evaluación	• Configura adecuadamente las propiedades principales del formulario (Text, Size, BackColor, StartPosition, FormBorderStyle).

	• Utiliza correctamente controles visuales básicos (Label, TextBox, Button, PictureBox, GroupBox). • Organiza visualmente los controles dentro del formulario de forma coherente y funcional. • Aplica una personalización estética coherente (tipografías, colores, íconos). • Implementa correctamente la propiedad Anchor para lograr formularios adaptables. • Establece un orden lógico de tabulación utilizando la propiedad TabIndex. • Añade funcionalidades básicas como cambio de color o comportamiento de controles al interactuar. • Verifica visual y funcionalmente el diseño al momento de ejecutar y redimensionar la aplicación. • Documenta correctamente los nombres de controles y su uso en el diseño. • Presenta evidencia clara del formulario y sus funcionalidades mediante capturas o demostración.
Rúbricas o listas de cotejo para valorar desempeño	Lista de cotejo con siete criterios clave que evalúan la creación del proyecto, el diseño de la interfaz, la funcionalidad del código, el análisis reflexivo y la presentación del reporte. Cada ítem se evalúa con las opciones: “Cumple / No cumple” y espacio para observaciones.
Formatos de reporte de prácticas	**Formato libre estructurado que incluya:** 1. Datos del alumno. 2. Nombre de la práctica. 3. Objetivo. 4. Capturas de pantalla del proyecto. 5. Código implementado. 6. Respuestas al análisis. 7. Conclusiones y Reflexión.

NOMBRE DE LA PRÁCTICA No.6	**Diseño Modular de Aplicaciones en C# utilizando Arquitectura en Tres Capas con Validación de Entradas y Gestión de Imágenes**
COMPETENCIA DE LA PRÁCTICA	Implementa una aplicación con arquitectura en tres capas, con validación de entradas y carga de imágenes, para fortalecer la separación de responsabilidades y la reutilización del código, utilizando Visual Studio con C# en un entorno de Windows Forms, promoviendo el pensamiento analítico y el trabajo colaborativo.

FUNDAMENTO TÉORICO

La práctica se basa en los principios de la arquitectura de software en capas, que promueve la separación de responsabilidades para mejorar la mantenibilidad, escalabilidad y reutilización del código. Se aplican conceptos de programación orientada a objetos, como encapsulamiento y herencia, junto con técnicas de validación de entrada y manejo de eventos en interfaces gráficas, dentro del entorno de desarrollo Visual Studio con C#.

MATERIALES, EQUIPAMIENTO Y/O REACTIVOS

Equipo de cómputo

- Computadora personal o de laboratorio con sistema operativo Windows 10 o superior
- Procesador mínimo: Intel Core i3 o equivalente
- Memoria RAM mínima: 8 GB
- Espacio disponible en disco: al menos 10 GB

Software

- Visual Studio 2022 (versión Community, Professional o Enterprise) con los siguientes componentes:
  - Desarrollo de escritorio con .NET
  - Windows Forms .NET Framework o .NET 6/7, según elección del estudiante
  - .NET SDK correspondiente a la versión seleccionada (recomendado: .NET 6 o .NET 7)

Recursos digitales

- Acceso a Internet para descarga de herramientas (en caso de ser necesario)
- Carpeta asignada o unidad compartida para guardar el proyecto generado

Cuenta de Microsoft (opcional, para sincronización de Visual Studio)

PROCEDIMIENTO O METODOLOGÍA

Arquitectura en Tres Capas

En esta sección del curso aprenderemos cómo estructurar un sistema informático utilizando la arquitectura en tres capas. Este modelo se utiliza ampliamente en el desarrollo de software porque facilita la organización del código, mejora la escalabilidad, y hace que los sistemas sean más fáciles de mantener. Usaremos un ejemplo práctico en Visual Studio para crear un proyecto con esta

arquitectura. Aunque el texto original tiene un enfoque introductorio, aquí ofreceremos un enfoque más estructurado y detallado.

¿Qué es la Arquitectura en Tres Capas?
La arquitectura en tres capas divide un sistema en tres partes principales, cada una con funciones bien definidas:

1. **Capa de Presentación**: Es la interfaz gráfica del usuario (GUI). Aquí interactúan los usuarios con el sistema.
2. **Capa de Negocio o Lógica**: Contiene las reglas del negocio y gestiona las solicitudes de la capa de presentación.
3. **Capa de Datos**: Gestiona el acceso a las bases de datos, ya sea para almacenar o recuperar información.

Este enfoque modular ofrece múltiples ventajas, como la separación de responsabilidades, lo que facilita la depuración, el mantenimiento y la posibilidad de cambiar tecnologías sin afectar todo el sistema.

Descripción de Cada Capa

1. Capa de Presentación (Interfaz Gráfica del Usuario)

- **Función**: Es la capa visible para el usuario. Recoge sus datos y muestra resultados.
- **Ejemplo**:
  - Un formulario con campos de texto y botones en una aplicación de escritorio.
  - Validaciones simples, como asegurarse de que el usuario introduzca un correo electrónico válido.
- **Analogía**: Piensa en una tienda física; la capa de presentación es el escaparate donde los clientes ven y eligen los productos.

2. Capa de Negocio (Lógica del Negocio)

- **Función**: Se encarga de procesar las solicitudes de la capa de presentación. Aquí residen las reglas del negocio y la lógica del sistema.
- **Tareas principales**:
  - Validar datos (e.g., comprobar que un usuario existe en la base de datos).
  - Procesar cálculos o transformaciones (e.g., calcular impuestos).
  - Comunicarse con la capa de datos para obtener o almacenar información.

- **Ejemplo**:
  - Un módulo que valida si un usuario tiene saldo suficiente antes de realizar una transferencia bancaria.
- **Analogía**: Es el "almacén" detrás del escaparate de la tienda, donde se preparan los pedidos antes de entregarlos.

3. Capa de Datos

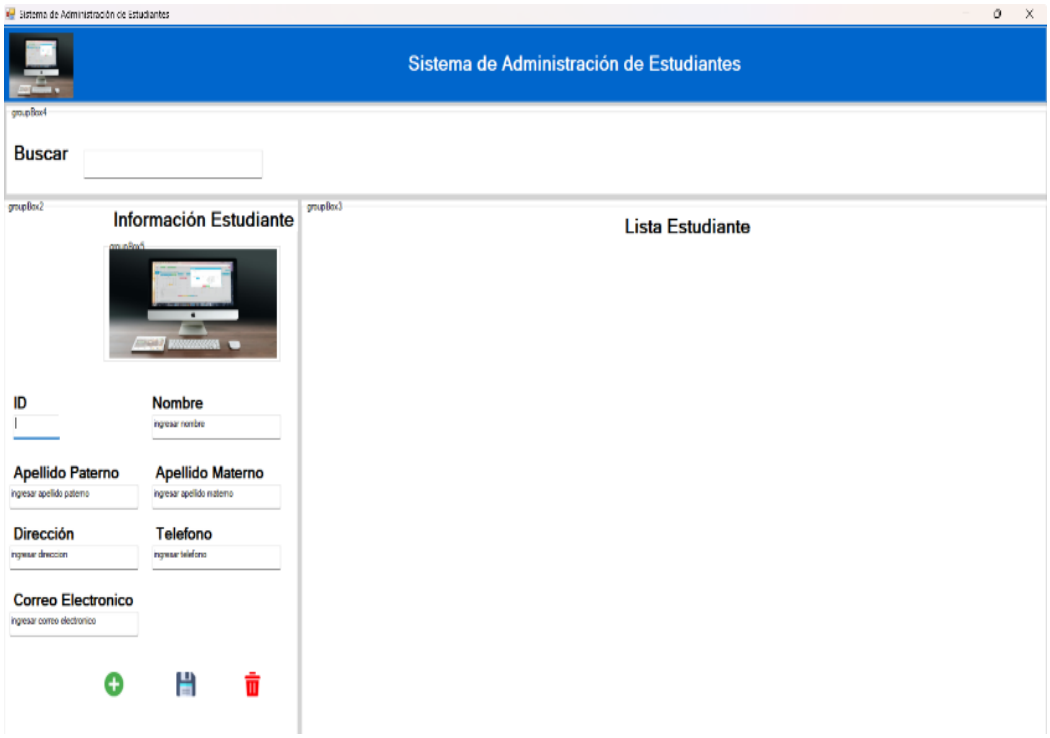
- **Función**: Gestiona la conexión a bases de datos, así como la recuperación y almacenamiento de información.
- **Ejemplo**:
  - Consultar una tabla SQL para obtener los datos de un cliente.

- Almacenar un nuevo pedido en una base de datos relacional como MySQL o SQL Server.
- **Tecnologías comunes**:
  - MySQL, SQL Server, PostgreSQL.
- **Analogía**: Es el "almacén central" de productos, donde todo está registrado y se guarda para referencia futura.

Pasos para Implementar la Arquitectura en Tres Capas

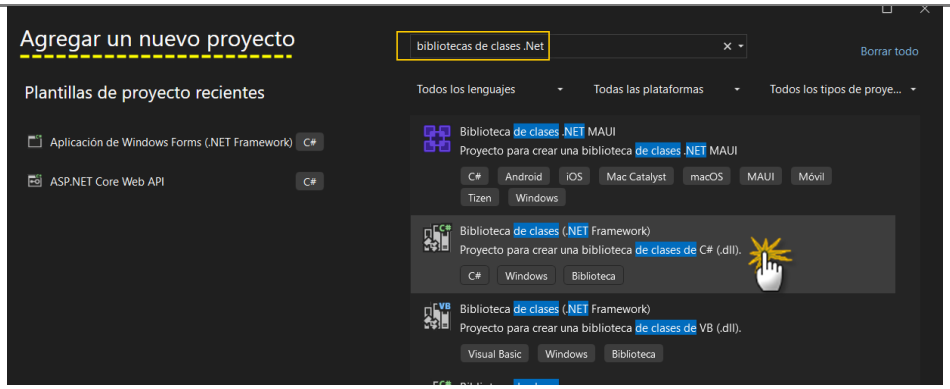
1. Crear la Capa de Presentación

1. Abre **Visual Studio** y crea un proyecto llamado **"Presentación".**
2. Diseña una interfaz gráfica utilizando herramientas como:
   - **Forms**: Botones, cuadros de texto, etiquetas.
   - Valida entradas simples como formatos de correo electrónico.



2. Crear la Capa de Negocio

1. En el explorador de soluciones, agrega un **nuevo proyecto** llamado **"Lógica".**
2. Selecciona una **Biblioteca de Clases (.NET Framework)**.

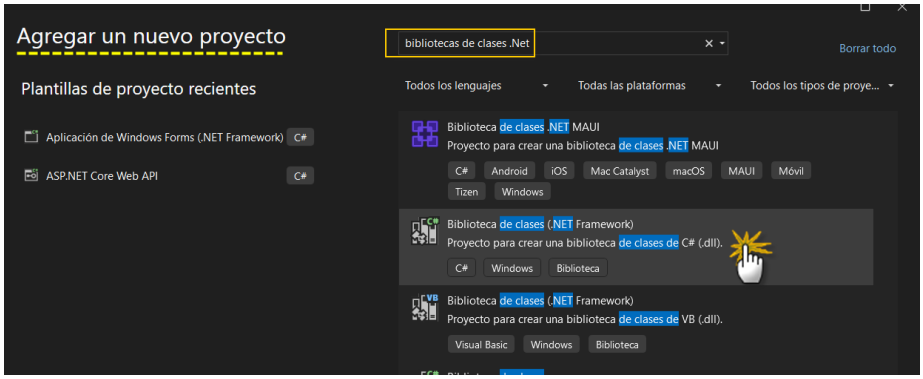


3. Define métodos que procesen solicitudes de la capa de presentación, como

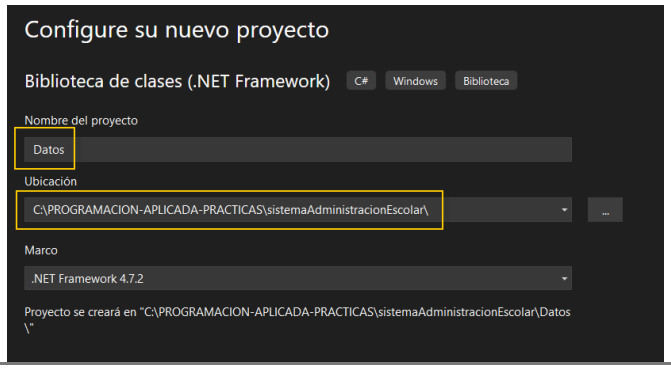
```
namespace sistemaAdministracionEscolar
{
    internal class BusinesLogic
    {
        //Creamos funciones que permitan Validar
    }
}
```

Crear la Capa de Datos

1. En el explorador de soluciones, agrega otro proyecto llamado **"Datos".**
2. Selecciona una Biblioteca de Clases (.NET Framework).



3. Establece el identificador y selecciona el directorio, en donde se encuentra alojado tu proyecto



4. Define clases que manejen las conexiones con las bases de datos, **por ejemplo:**

```
public class ConexionBaseDatos
{
    public string CadenaConexion { get; set; }

    public ConexionBaseDatos()
    {
        // Define la cadena de conexión
        CadenaConexion = "Server=myServer;Database=myDB;User Id=myUser;Password=myPassword;";
    }

    public DataTable EjecutarConsulta(string consulta)
    {
        // Ejecuta consultas SQL y devuelve los resultados
        using (SqlConnection conexion = new SqlConnection(CadenaConexion))
        {
            SqlDataAdapter adaptador = new SqlDataAdapter(consulta, conexion);
            DataTable resultados = new DataTable();
            adaptador.Fill(resultados);
            return resultados;
        }
    }
}
```

Integrar las Capas

- **Conexión entre capas**:
  - La capa de presentación llama a los métodos de la capa de negocio.
  - La capa de negocio utiliza los métodos de la capa de datos para acceder a la base de datos.
- **Ejemplo completo**:
  - La interfaz de usuario recoge el nombre de usuario y contraseña.
  - La capa lógica valida estos datos.
  - La capa de datos consulta la base de datos para verificar la existencia del usuario.

Ventajas de la Arquitectura en Tres Capas

1. **Mantenibilidad**: Puedes actualizar una capa sin afectar las otras.
2. **Escalabilidad**: Es fácil añadir funcionalidades sin romper el sistema.
3. **Flexibilidad**: Permite cambiar de base de datos o interfaz gráfica sin reescribir toda la lógica.

Ejemplo Analógico:

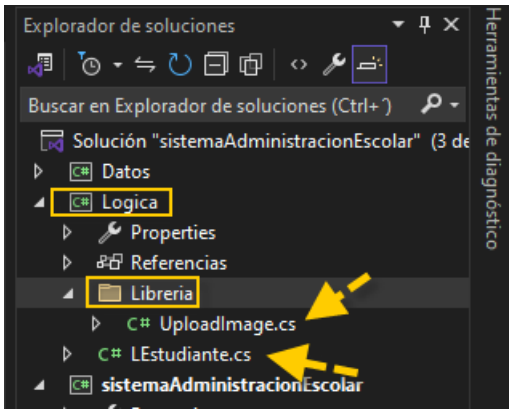
Piensa en una pizzería:

- El cliente (usuario) habla con el mesero (capa de presentación) para hacer un pedido.
- El mesero transmite la orden al cocinero (capa de lógica), quien sigue las recetas y prepara la pizza.
- El cocinero consulta el inventario (capa de datos) para obtener los ingredientes necesarios.

Estructura del Procedimiento para Crear Clase la cual permita alojar imágenes

Crear un procedimiento que permita al usuario seleccionar una imagen desde su computadora y mostrarla en un control llamado **PictureBox** dentro de la interfaz gráfica.

1. **Crear una Clase para Manejo de Imágenes:**
   - Renombramos a la clase, la cual indicaremos como **“LEstudiante.cs”**
   - La clase que indicaremos como **“UploadImage.cs”** se aloja en el proyecto **"Lógica"** dentro de una carpeta llamada **Librería**.
   - Nos posicionaremos en la clase, con la finalidad de poder definir métodos reutilizables para procesar imágenes.



2. **Método para Seleccionar y Cargar Imágenes:**
   - Se utiliza el componente **OpenFileDialog**, que abre una ventana para explorar archivos.
   - Se filtran las extensiones permitidas (e.g., .jpg, .png).

   Código Simplificado:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms; //Nos aseguramos de importar la librería

namespace Logica.Libreria
{
    //Establecemos el ámbito de lectura como publico
    public class UploadImage
    {
        //Creamos un objeto, con ámbito de lectura privado
        private OpenFileDialog fileDialog = new OpenFileDialog();

        //Crearemos un metodo público, el cual contiene un parámetro de tipo PictureBox
        public void CargarImagen(PictureBox pictureBox)
        {
            //Haciendo uso del parámetro, invocamos el metodo
            pictureBox.WaitOnLoad = true;
            //Haremos uso del objeto y le indicamos la propiedad filtro
            fileDialog.Filter = "Imagenes|*.jpg;*.gif;*.png;*.bmp";
            //Utilizando el objeto, invocamos el metodo ShowDialog
            fileDialog.ShowDialog();
            //Creamos una condición
            if (fileDialog.FileName != string.Empty)
            {
                //Utilizamos el objeto, para invocar la propiedad
                pictureBox.ImageLocation = fileDialog.FileName;
            }


        }

    }
}
```

3. **Invocar el Método desde la Interfaz:**
    - Al hacer clic en un botón o control, el método se ejecuta, cargando la imagen seleccionada.

      Errores Comunes y Soluciones:

      - **Conflicto de Nombres:** Si dos clases o namespaces tienen el mismo nombre, el compilador genera un error. La solución es usar nombres únicos o especificar el namespace al llamar la clase.
      - **Referencias Faltantes:** Si una capa necesita acceder a otra, se debe agregar una referencia explícita en el proyecto.

4. **Organización y Herencia entre Clases**

   El texto destaca el uso de Herencia para reutilizar código. Por ejemplo, si se crea una clase base con métodos genéricos, otras clases pueden heredar de ella y extender su funcionalidad.

   Ejemplo de Herencia Aplicada:

   - Nos dirigiremos hacia la clase, que se indica como **LEstudiante.cs**, con la finalidad de establecer la herencia con la clase **UploadImage.cs**

```
using Logica.Libreria; //Se incorporó la ruta del directorio y clase
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Logica
{
    //Indicamos la Herencia al compilador
    public class LEstudiante: UploadImage
    {
    }
}
```

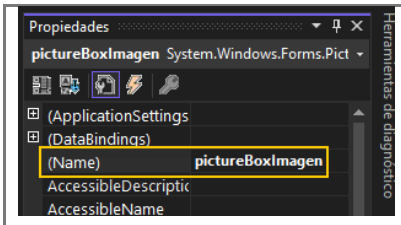
5. **Integración y Comunicación entre Capas**
    - Cada capa interactúa siguiendo principios de separación de responsabilidades:
        - La **capa de presentación** llama métodos de la lógica.
        - La **lógica** se comunica con la capa de datos cuando necesita información.

          Ejemplo de Flujo Completo:

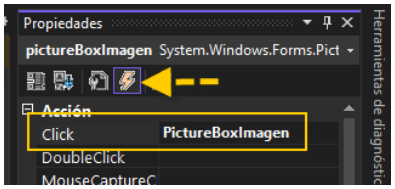
          1. El usuario selecciona una imagen en la interfaz gráfica.
          2. La capa de presentación llama al método CargarImagen de la capa lógica.
          3. La lógica procesa la solicitud y, si es necesario, interactúa con la capa de datos.

Propiedades y Eventos

Lo siguiente que haremos, será establecer la propiedad Name, del control PictureBox



Lo siguiente que haremos, será ingresar al evento **Clic**, y seleccionaremos el control **Picturebox**



- Pulsaremos un clic y podremos notar que ingresaremos al código

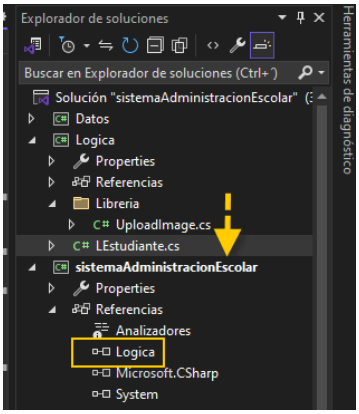
```
private void PictureBoxImagen(object sender,
EventArgs e)
{

}
```

Instanciar

Lo siguiente que haremos, será crear un objeto de la clase **"LEstudiante.cs"**

```
using Logica; //Nos aseguramos que se importe la referencia
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace sistemaAdministracionEscolar
{
    public partial class Form1 : Form
    {
        //Creamos un objeto de la clase Estudiante
        private LEstudiante lEstudiante=new LEstudiante();

        public Form1()
        {
            InitializeComponent();
        }


        private void PictureBoxImagen(object sender, EventArgs e)
        {

        }

    }
}
```

Podremos observar que se ha cargado la referencia



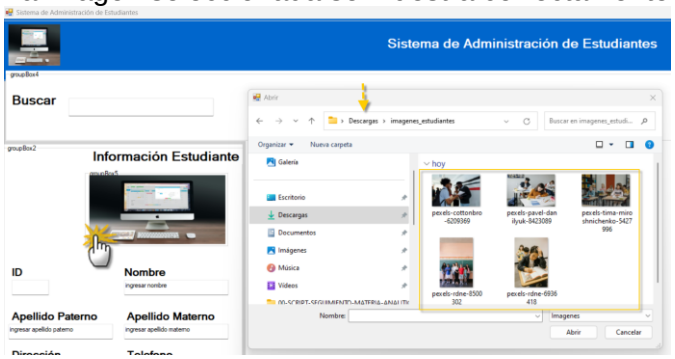
Utilizando el objeto, invocamos al metodo que creamos en la clase **"UploadImage.cs"**

```
private void PictureBoxImagen(object sender, EventArgs e)
{
    //Haciendo uso del objeto, invocamos al metodo y le pasamos el argumento
    //El argumento es el nombre del control pictureBox
    lEstudiante.CargarImagen(pictureBoxImagen);
}
```

6. Pruebas y Ejecución

Al ejecutar la aplicación, se verifica si:

- La imagen seleccionada se muestra correctamente en el PictureBox.



- No hay conflictos de nombres ni errores en las referencias.

Pruebas Básicas:

1. Probar con diferentes formatos de imagen.
2. Seleccionar un archivo no válido (e.g., un documento) para comprobar el manejo de errores.

3. Verificar la ejecución fluida entre capas.

Mejores Prácticas y Consejos Finales

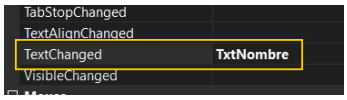
1. **Organización del Proyecto:**
   - Nombrar clases y carpetas de manera descriptiva para evitar conflictos.
   - Mantener cada capa independiente y evitar mezclar responsabilidades.
2. **Gestión de Errores:**
   - Implementar validaciones para manejar escenarios como archivos inexistentes o incompatibles.
3. **Extensibilidad:**
   - Diseñar métodos genéricos y reutilizables que puedan adaptarse a futuros requerimientos.

Eventos en Campos de Texto

Haremos uso de **eventos** para capturar y procesar la información ingresada en los campos de texto.

Evento TextChanged

- Se ejecuta cada vez que cambia el contenido de un campo de texto.
- Permite realizar validaciones en tiempo real.
- **Ejemplo:** Mostrar un mensaje cuando el usuario deja vacío un campo obligatorio.
  - Para el desarrollo del ejemplo, deberás ajustar la propiedad **Name** del control **Label y** también la propiedad del control **TextBox**
  - Utilizaremos el evento **TextChanged**, del control TextBox




```
vate void TxtNombre(object sender, EventArgs e)

    //Establecemos una condición
    if (txtNombre.Text=="")
    {
        //cambiamos el color de la etiqueta
        LblNombre.ForeColor = Color.Red;
    }
    else
    {
        LblNombre.ForeColor = Color.Green;
    }
```

Explicación:

- txtNombre.Text == "": Se verifica si el campo está vacío.
- lblNombre.ForeColor = Color.Red; Si está vacío, la etiqueta se vuelve roja.
- lblNombre.ForeColor = Color.Green; Si hay datos, la etiqueta se vuelve verde.

Creación de la Clase de Validación

Para el desarrollo de la aplicación crearemos una clase dedicada a manejar validaciones en los campos de texto. Para ello, debemos crear una nueva clase llamada **TextBoxEvent** dentro de una carpeta específica del proyecto.

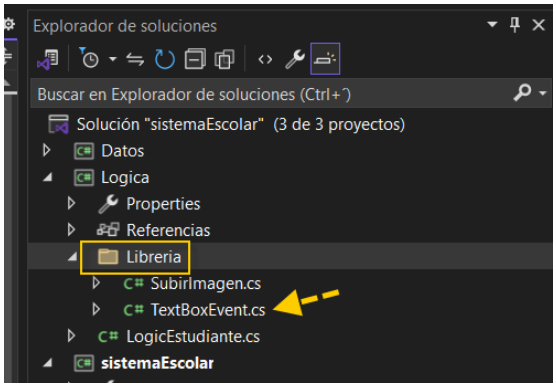
¿Por qué crear una clase separada para validaciones?

Separar la lógica de validación en una clase independiente ayuda a:

1. **Reutilización del código**: Se puede utilizar la misma validación en varios campos sin duplicar código.
2. **Mantenibilidad**: Facilita la modificación o mejora de las reglas de validación sin afectar otras partes del código.
3. **Organización del código**: Separa la lógica de validación de la interfaz gráfica.

Ejemplo de Código para Crear la Clase TextBoxEvent

- Nos dirigiremos hacia el directorio que se identifica como "**Libreria**", directorio el cual se encuentra alojado dentro de la carpeta que se identifica como "**Logica**"



- La clase **TextBoxEvent** es una estructura pública, diseñada para restringir la entrada de caracteres en un control de texto dentro de una aplicación de Windows Forms, garantizando que solo se acepten **letras**, **espacios** y la **tecla de retroceso.**
  - Su método **SoloLetras** recibe un objeto de tipo **KeyPressEventArgs**, el cual proporciona información sobre la tecla presionada. A través de una condición lógica, la función evalúa si el carácter ingresado es una letra mediante **char.IsLetter (eventArgs.KeyChar),** permitiendo su entrada; en caso contrario, verifica si la tecla presionada es retroceso **('\b')** o espacio **(' '),** las cuales también están permitidas.
  - Si la tecla ingresada no cumple con estos criterios, se activa **eventArgs.Handled = true**, lo que impide que el carácter no válido sea procesado por el control.
  - Este enfoque es fundamental para validar entradas en tiempo real y asegurar que los datos ingresados cumplan con las restricciones deseadas en la interfaz de usuario.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
```

```
using System.Windows.Forms; //Nos aseguramos que se establezca la importación

namespace Logica.Libreria
{
    //Definiremos la clase como pública
    public class TextBoxEvent
    {
        //Creamos una funcion sin retorno
        public void SoloLetras(KeyPressEventArgs eventArgs) {

            //Establecemos una Estructura condicional
            if (!char.IsLetter(eventArgs.KeyChar) &&
                eventArgs.KeyChar != '\b' &&
                eventArgs.KeyChar != ' ' )
            {
                //En caso de devolver True
                eventArgs.Handled = true;
            }

        }
    }
}
```

Explicación del código

- **char.IsLetter (keyPressEventArgs.KeyChar)**: Comprueba si el carácter presionado (KeyChar) es una **letra** (mayúscula o minúscula).
- **! char.IsLetter(...)**: Si el carácter **no** es una letra, la condición se evalúa como **true**.
- **keyPressEventArgs.KeyChar!= '\b'**: Se asegura de que la tecla presionada **no** sea la tecla **Backspace** (\b), permitiendo así que el usuario pueda borrar texto.
- **keyPressEventArgs.KeyChar!= ' '**: Permite la captura del **espacio en blanco**.
- **keyPressEventArgs.Handled = true;**: Indica que el evento ya ha sido manejado y **bloquea la entrada del carácter en el campo de texto**.

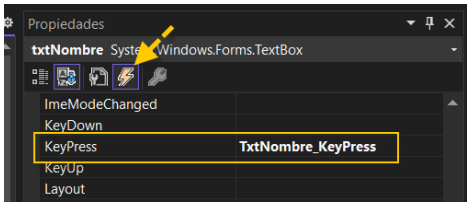
El código indica que solo permite letras y espacios en blanco, si se presiona cualquier otra tecla (numeros, signos de puntuación, caracteres especiales), la condición será True, lo que indica que se bloqueara la entrada del carácter

Asociación del Método de Validación con los Campos de Texto

Después de crear la clase **TextBoxEvent**, el programador debe asociar el método **SoloLetras** al evento **KeyPress** de los campos de texto.

Ejemplo de Asociación en un Formulario

- Nos dirigiremos hacia el evento **KeyPress** del control TextBox, el cual identificamos como "**TxtNombre.text**"



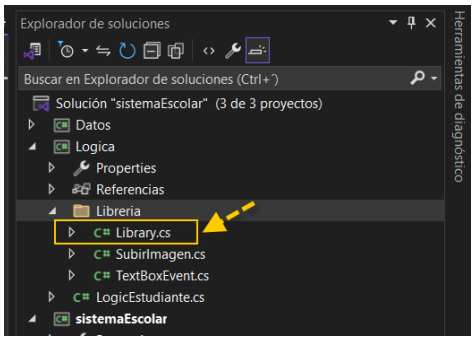
- La línea de código **TextBoxEvent boxEvent = new TextBoxEvent ();** realiza la instanciación de un objeto de la clase **TextBoxEvent**, lo que significa que se está creando una nueva instancia de esta clase en memoria para su posterior uso. En términos técnicos, esta operación invoca el constructor predeterminado de **TextBoxEvent**, asignando la referencia del nuevo objeto a la variable **boxEvent**, permitiendo así acceder a los métodos y propiedades definidos dentro de dicha clase

```
using Logica; //Nos aseguramos de establecer la referencia
using Logica.Libreria; //Nos aseguramos de establecer la referencia
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace sistemaEscolar
{

    public partial class Form1 : Form
    {
        private LogicEstudiante objLogicEstudiante = new LogicEstudiante();
        //Instanciamos
        TextBoxEvent boxEvent = new TextBoxEvent();

         private void TxtNombre_KeyPress(object sender, KeyPressEventArgs e)
         {
             //Utilizamos el objeto e invocamos el metodo sin retorno
             //y le pasamos el argumento de tipo KeyPressEventArgs
             textBoxEvent.SoloLetras(e);
         }
```

Manejo de Herencia y Múltiples Objetos

Como no podemos usar la **herencia múltiple** en C#. Para resolver esto, se crea una clase intermedia llamada **Librería** para manejar múltiples objetos.

Creación de la Clase **Library**



- El código define una clase pública llamada **Library,** dentro de esta clase, se crean dos instancias de otras clases, **TextBoxEvent** y **SubirImagen**, lo que indica que la clase **Library** utilizará funcionalidades de estas clases. **TextBoxEvent** y **SubirImagen** están definidas en otro lugar

del proyecto. La clase Library no contiene métodos ni propiedades adicionales, por lo que actualmente solo sirve para instanciar estos objetos.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Logica.Libreria
{
    //Establecemos el ámbito de lectura como publico
    public class Library
    {
        //Instanciamos desde la clase TextBoxEvent
        public TextBoxEvent textBoxEvent = new TextBoxEvent();
        //Instanciamos desde la clase SubirImagen
        public SubirImagen subirImagen = new SubirImagen();
    }
}
```

Uso de Libreria en la Clase **LogicEstudiante**

- La clase llamada **LogicEstudiante** que hereda de otra clase llamada **Library**. Esto significa que LogicEstudiante tendrá acceso a los métodos y propiedades públicos y protegidos de Library, lo que permite reutilizar su funcionalidad sin necesidad de reescribirla.

```
using Logica.Libreria; //Establecemos la referencia
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Logica
{
    //Herencia desde la clase Library
    public class LogicEstudiante: Library
    {

    }
}
```

Ahora, en la clase **Form1.cs,** se puede acceder a la validación de esta manera:

- Se define dos métodos manejadores de eventos para una aplicación con interfaz gráfica. El primer método, **PictureBox1**, se ejecuta cuando ocurre un evento en un **PictureBox** y llama al método **cargarImagen** del objeto **subirImagen** dentro de **objLogicEstudiante**, pasando como argumento el control **pictureBox1** para cargar una imagen.
- El segundo método, **TxtNombre_KeyPress**, se activa cuando el usuario presiona una tecla en un **TextBox**, llamando al método **SoloLetras** del objeto **textBoxEvent** en **objLogicEstudiante**, con el argumento **(e)** de tipo **KeyPressEventArgs**, para restringir la entrada a solo letras.

```
private void PictureBox1(object sender, EventArgs e)
{
    //Utilizar el objeto
```

```
        objLogicEstudiante.subirImagen.cargarImagen(pictureBox1);
    }


    private void TxtNombre_KeyPress(object sender, KeyPressEventArgs e)
    {
        //Utilizamos el objeto e invocamos el metodo sin retorno
        //y le pasamos el argumento de tipo KeyPressEventArgs
        objLogicEstudiante.textBoxEvent.SoloLetras(e);
    }
```

Prueba y Ejecución

Después de implementar el código, se ejecuta la aplicación para probar el campo de texto:

- **Se pueden ingresar letras**
- **Se pueden generar espacios entre palabras**
- **Se puede eliminar caracteres con Backspace**
- **No se pueden ingresar números o caracteres especiales**
- **No se permite presionar Enter**

RESULTADOS ESPERADOS

1. El estudiante estructura correctamente una solución basada en la arquitectura en tres capas, identificando y separando las responsabilidades de cada módulo (presentación, lógica y datos).
2. Implementa métodos de validación de entrada en campos de texto utilizando eventos y clases reutilizables.
3. Integra la funcionalidad de carga de imágenes en la interfaz gráfica mediante clases especializadas.
4. Establece comunicación eficiente entre capas utilizando referencias y objetos, respetando la modularidad del sistema.
5. Aplica principios de reutilización de código y buenas prácticas de programación orientada a objetos.
6. Demuestra pensamiento analítico y autonomía al resolver problemas durante el desarrollo de la aplicación.

ANÁLISIS DE RESULTADOS

- ¿Cómo favorece la arquitectura en tres capas la organización y mantenibilidad del código en una aplicación?
- ¿Qué ventajas observaste al reutilizar métodos mediante herencia o instanciación de clases?
- ¿La implementación de la carga de imágenes funcionó como se esperaba? ¿Qué ajustes realizaste para lograrlo?
- ¿Qué dificultades encontraste al validar los datos de entrada en los campos de texto? ¿Cómo las resolviste?
- ¿Qué capa consideras más crítica en términos de seguridad o integridad del sistema? ¿Por qué?
- ¿Qué mejoras aplicarías al proyecto para hacerlo más robusto o escalable?
- ¿De qué manera el trabajo en capas te permitió detectar y corregir errores más fácilmente durante el desarrollo?

CONCLUSIONES Y REFLEXIONES

La implementación de la arquitectura en tres capas permitió al estudiante comprender la importancia de separar las responsabilidades del sistema, facilitando la organización del código y su mantenimiento. La reutilización de clases y la validación de entradas reforzaron el uso de principios de programación orientada a objetos. Además, se evidenció que una estructura modular mejora la escalabilidad, el control de errores y la adaptabilidad del software ante futuros cambios. Esta práctica también promovió el pensamiento lógico y la capacidad de análisis durante el desarrollo de soluciones funcionales.

ACTIVIDADES COMPLEMENTARIAS

1. Integración de un Módulo de Registro:
   - Desarrollar un formulario adicional que permita registrar nuevos usuarios, almacenando la información en la base de datos mediante la capa de lógica y datos, con validaciones aplicadas a cada campo.
2. Extensión de la Funcionalidad de Imagen:
   - Implementar una función para eliminar o reemplazar la imagen cargada en el PictureBox, utilizando controles adicionales (botón de limpieza o reemplazo) y aplicando principios de

encapsulamiento.

3. Análisis Comparativo de Arquitecturas:
   - Investigar y elaborar un resumen comparativo entre la arquitectura en tres capas y otras arquitecturas comunes (por ejemplo, MVC o cliente-servidor), identificando ventajas, desventajas y escenarios de aplicación.

EVALUACIÓN Y EVIDENCIAS DE APRENDIZAJE	
Criterios de evaluación	• Estructura correctamente una solución basada en la arquitectura en tres capas, diferenciando claramente las capas de presentación, lógica y datos. • Implementa validaciones de entrada utilizando clases específicas y eventos del entorno Windows Forms. • Desarrolla la funcionalidad para cargar imágenes desde el equipo del usuario y mostrarlas en la interfaz gráfica. • Aplica principios de programación orientada a objetos, como reutilización de código y herencia, en el diseño de las clases. • Establece una comunicación funcional y coherente entre las capas del sistema. • Demuestra pensamiento analítico al resolver problemas de integración, validación y funcionamiento del sistema. • Organiza adecuadamente los componentes del proyecto, manteniendo una estructura modular y comprensible. • Interpreta y responde con fundamento técnico las preguntas de análisis planteadas en la práctica.
Rúbricas o listas de cotejo para valorar desempeño	Lista de cotejo con siete criterios clave que evalúan la creación del proyecto, el diseño de la interfaz, la funcionalidad del código, el análisis reflexivo y la presentación del reporte. Cada ítem se evalúa con las opciones: “Cumple / No cumple” y espacio para observaciones.
Formatos de reporte de prácticas	Lista de cotejo con siete criterios clave que evalúan la creación del proyecto, el diseño de la interfaz, la funcionalidad del código, el análisis reflexivo y la presentación del reporte. Cada ítem se evalúa con las opciones: “Cumple / No cumple” y espacio para observaciones.

NOMBRE DE LA PRÁCTICA	**Validación Avanzada de Entradas y Uso de Depuración con Unicode en Aplicaciones Windows Forms**
COMPETENCIA DE LA PRÁCTICA	Aplica técnicas de validación de entradas en campos de texto utilizando códigos Unicode y eventos del teclado, con la finalidad de garantizar la integridad de los datos capturados, bajo condiciones controladas de depuración en tiempo de ejecución, dentro de un entorno de desarrollo en Windows Forms con C#, fortaleciendo el pensamiento crítico y la atención al detalle.

FUNDAMENTO TÉORICO

La práctica se fundamenta en la programación orientada a eventos y en el uso del estándar Unicode para identificar y filtrar caracteres en tiempo real. Se aplican técnicas de depuración mediante breakpoints para analizar el comportamiento del programa, así como principios de validación de datos en interfaces gráficas desarrolladas con C# en Windows Forms.

MATERIALES, EQUIPAMIENTO Y/O REACTIVOS

Equipo de cómputo

- Computadora personal o de laboratorio con sistema operativo Windows 10 o superior
- Procesador mínimo: Intel Core i3 o equivalente
- Memoria RAM mínima: 8 GB
- Espacio disponible en disco: al menos 10 GB

Software

- Visual Studio 2022 (versión Community, Professional o Enterprise) con los siguientes componentes:
  - Desarrollo de escritorio con .NET
  - Windows Forms .NET Framework o .NET 6/7, según elección del estudiante
  - .NET SDK correspondiente a la versión seleccionada (recomendado: .NET 6 o .NET 7)

Recursos digitales

- Acceso a Internet para descarga de herramientas (en caso de ser necesario)
- Carpeta asignada o unidad compartida para guardar el proyecto generado

Cuenta de Microsoft (opcional, para sincronización de Visual Studio)

PROCEDIMIENTO O METODOLOGÍA

Uso de depuración y Unicode para identificar caracteres

Uno de los puntos clave en el video es el uso de **puntos de interrupción (breakpoints)** para depurar la aplicación y observar cómo se comporta la validación en tiempo de ejecución.

¿Qué es la depuración?

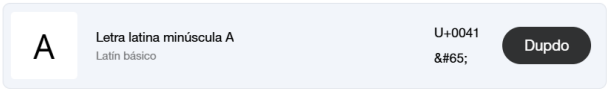
La **depuración** (debugging) es el proceso de ejecución controlada de un programa para identificar errores o verificar su correcto funcionamiento. En este caso, se utiliza para ver qué valores se capturan cuando se ingresa un carácter en un campo de texto.

El desarrollador usa un sitio web para obtener el **código Unicode** de los caracteres. El Unicode es un estándar que asigna un número único a cada carácter, lo que permite su identificación precisa.

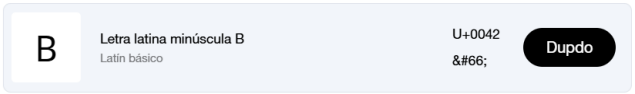
Ejemplo de obtención de Unicode:

Cuando se presiona la tecla **"A"**, se obtiene su código Unicode:

- Letra **"A"** en mayuscula → **Código Unicode: 65**



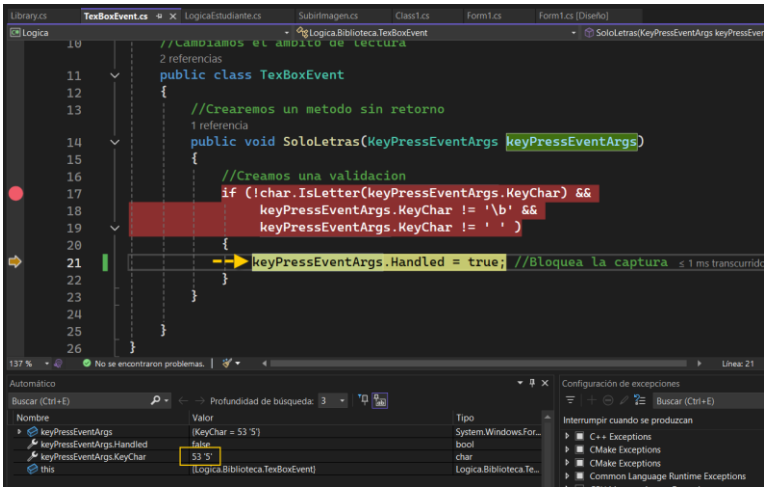
- Letra **"B"** en mayuscula → **Código Unicode: 66**



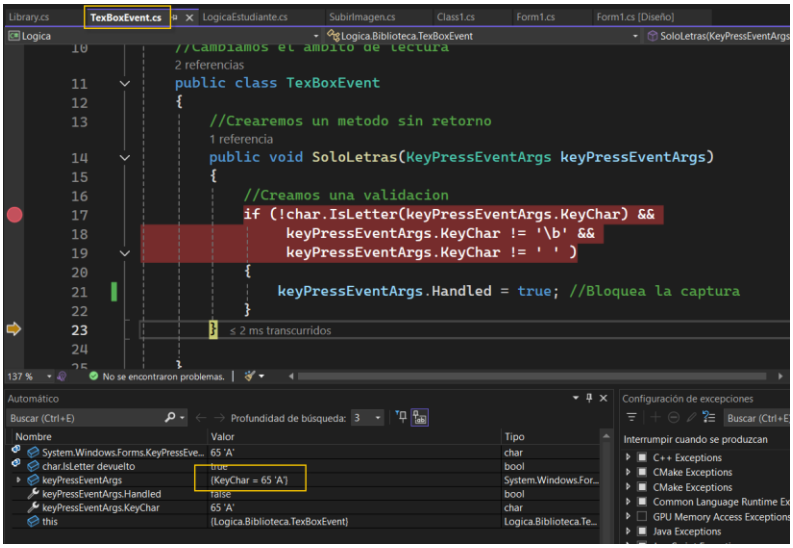
Sitio que nos permite visualizar el Código Unicode: https://symbl.cc/es/#control-character

Con esta información, el programa puede verificar si el valor ingresado corresponde a una letra o a un número.

Ejemplo, ingresando un número, en donde podremos apreciar que nos devuelve un **True**, eso significa que no permitirá el ingreso de ese valor numérico



Ejemplo, ingresando una letra, en donde podremos apreciar que nos devuelve un **False**, eso significa que permitirá el ingreso de ese valor de tipo carácter



Validación de valores numéricos

En el caso del campo **Número de Identidad**, la validación debe permitir solo números y bloquear letras o caracteres especiales.

El proceso es similar al anterior:

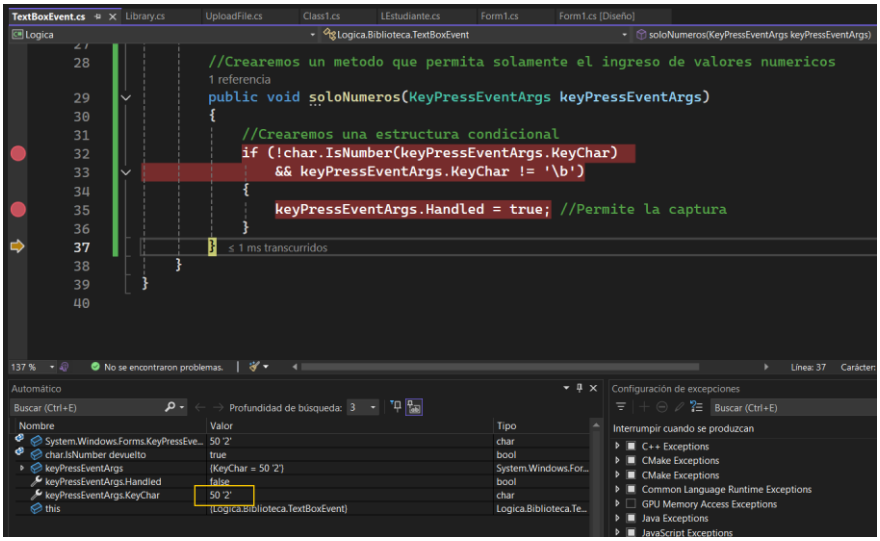
1. Capturar el carácter ingresado.
2. Verificar si es un **dígito numérico**.
3. Permitir la entrada solo si es un número.

Crearemos un metodo público, sin retorno, el cual identificaremos como **“soloNumeros”**

- El método **soloNumeros** en restringe la entrada de caracteres en un campo de texto para permitir solo valores numéricos. Recibe como parámetro un objeto **KeyPressEventArgs,** que captura la tecla presionada. Luego, mediante una estructura condicional, verifica si el carácter ingresado es un número usando **char.IsNumber (keyPressEventArgs.KeyChar)** o si es la tecla **Backspace (\b),** permitiéndolos pasar. Si no cumple con estas condiciones, se establece **keyPressEventArgs.Handled = true,** lo que evita que el carácter aparezca en el campo de texto.

```
//Crearemos un metodo que permita solamente el ingreso de valores numéricos
public void soloNumeros(KeyPressEventArgs keyPressEventArgs)
{
    //Crearemos una estructura condicional
    if (!char.IsNumber(keyPressEventArgs.KeyChar)
        && keyPressEventArgs.KeyChar != '\b')
    {
        keyPressEventArgs.Handled = true; //No Permite la captura
    }
}
```

Ejecutamos y podremos observar que solamente se permiten valores de tipo numérico



Validación de correos electrónicos

El campo de **correo electrónico** es una excepción, ya que permite:

- Letras (A-Z, a-z)
- Números (0-9)
- Caracteres especiales como @, ., _, -

Por ello, la validación es más flexible y se maneja de manera distinta.

Ejemplo de código para validar correos electrónicos:

- El código implementa una estructura condicional en C# dentro de un evento de teclado para validar la entrada de caracteres en un campo de texto. La condición **if** verifica si el carácter ingresado (**e.KeyChar**) no es una letra ni un dígito **(!char.IsLetterOrDigit(e.KeyChar))** y tampoco pertenece a un conjunto de caracteres permitidos, que incluyen la tecla de retroceso **('\b'),** el símbolo arroba **('@'),** el punto **('.')** y el guion bajo **('_').** Si el carácter no cumple con estas condiciones, la propiedad **e.Handled** se establece en **true**, lo que evita que el carácter sea procesado y mostrado en el campo de entrada, asegurando así un filtro para limitar la entrada a caracteres alfanuméricos y algunos símbolos específicos, como los utilizados en direcciones de correo electrónico.

```
//Crearemos una estructura condicional
if (!char.IsLetterOrDigit(e.KeyChar) &&
    e.KeyChar != '\b' &&
    e.KeyChar != '@' &&
    e.KeyChar != '.' &&
    e.KeyChar != '_'  ) {

    e.Handled = true; //Evitamos la captura
}
```

RESULTADOS ESPERADOS

- El estudiante identifica y utiliza correctamente los códigos Unicode para reconocer caracteres ingresados por el usuario.
- Implementa métodos de validación que restringen la entrada de letras, números o caracteres especiales según el tipo de campo.
- Utiliza técnicas de depuración con breakpoints para observar el flujo de ejecución y analizar el comportamiento del código en tiempo real.
- Aplica estructuras condicionales para permitir o bloquear caracteres en campos como nombre, número de identidad o correo electrónico.
- Desarrolla soluciones precisas y funcionales que mejoran la calidad de los datos capturados en la interfaz de usuario.

ANÁLISIS DE RESULTADOS

- ¿Qué observaste al utilizar breakpoints durante la ejecución del programa?
- ¿Qué utilidad tuviste al consultar los códigos Unicode para validar caracteres ingresados por el usuario?
- ¿Cómo aseguraste que un campo de texto solo permitiera letras, números o caracteres especiales según el caso?
- ¿Qué errores encontraste durante la validación de entradas y cómo los resolviste?
- ¿Cuál fue el beneficio de separar la lógica de validación en métodos reutilizables?
- ¿Qué dificultades técnicas experimentaste al manejar eventos del tipo KeyPress en los controles TextBox?
- ¿De qué manera el uso de depuración te ayudó a comprender mejor el flujo del programa?

CONCLUSIONES Y REFLEXIONES

La práctica permitió comprender la importancia de validar correctamente la entrada de datos para evitar errores en la interfaz de usuario. El uso de Unicode y eventos KeyPress facilitó la identificación de caracteres válidos, mientras que la depuración ayudó a visualizar el comportamiento del programa en tiempo real. Además, separar la lógica de validación en métodos específicos mejoró la claridad, reutilización y mantenimiento del código.

ACTIVIDADES COMPLEMENTARIAS

1. Ampliación de validaciones personalizadas:
   - Crear métodos adicionales que validen campos con formatos específicos, como números telefónicos, fechas o contraseñas con requisitos mínimos (letras, números y símbolos).
2. Simulación de errores mediante depuración:
   - Generar intencionalmente errores comunes en validaciones y utilizar breakpoints para analizarlos y corregirlos paso a paso.
3. Comparación de métodos de validación:
   - Investigar y documentar las diferencias entre el uso de KeyPress, expresiones regulares (Regex) y validaciones posteriores a la entrada, evaluando ventajas y desventajas de cada enfoque.

EVALUACIÓN Y EVIDENCIAS DE APRENDIZAJE	
Criterios de evaluación	• Identifica y utiliza correctamente códigos Unicode para validar caracteres ingresados. • Implementa métodos de validación que restringen adecuadamente el ingreso de caracteres según el tipo de campo (letras, números, símbolos). • Emplea técnicas de depuración con breakpoints para analizar el comportamiento del programa durante la ejecución. • Desarrolla estructuras condicionales funcionales para controlar las entradas en controles TextBox. • Separa la lógica de validación en métodos reutilizables, aplicando buenas prácticas de programación. • Comprueba, mediante pruebas, que los campos restringen correctamente los caracteres no deseados. • Refleja comprensión del flujo del programa mediante respuestas técnicas fundamentadas en el análisis de resultados. • Organiza su código de forma clara, con comentarios y nombres descriptivos para clases, métodos y variables.
Rúbricas o listas de cotejo para valorar desempeño	Lista de cotejo con siete criterios clave que evalúan la creación del proyecto, el diseño de la interfaz, la funcionalidad del código, el análisis reflexivo y la presentación del reporte. Cada ítem se evalúa con las opciones: “Cumple / No cumple” y espacio para observaciones.
Formatos de reporte de prácticas	**Formato libre estructurado que incluya:** 1. Datos del alumno. 2. Nombre de la práctica. 3. Objetivo. 4. Capturas de pantalla del proyecto. 5. Código implementado. 6. Respuestas al análisis. 7. Conclusiones y Reflexión.

NOMBRE DE LA PRÁCTICA No.8	Gestión de Formularios con Colecciones de Controles y Validación Secuencial en Windows Forms
COMPETENCIA DE LA PRÁCTICA	Implementa la validación secuencial de campos en formularios utilizando colecciones de controles TextBox y Label, con la finalidad de garantizar la integridad de los datos ingresados, bajo condiciones de retroalimentación visual en tiempo de ejecución, en un entorno de desarrollo con C# y Windows Forms, fortaleciendo la organización del trabajo y la responsabilidad en la codificación.

FUNDAMENTO TÉORICO

La práctica se fundamenta en la programación orientada a objetos y en la gestión eficiente de interfaces gráficas mediante el uso de colecciones de controles. Se aplican estructuras condicionales para validar secuencialmente campos de entrada y se implementa retroalimentación visual como mecanismo de control de errores, promoviendo una arquitectura modular, reutilizable y mantenible en aplicaciones Windows Forms.

MATERIALES, EQUIPAMIENTO Y/O REACTIVOS

Equipo de cómputo

- Computadora personal o de laboratorio con sistema operativo Windows 10 o superior
- Procesador mínimo: Intel Core i3 o equivalente
- Memoria RAM mínima: 8 GB
- Espacio disponible en disco: al menos 10 GB

Software

- Visual Studio 2022 (versión Community, Professional o Enterprise) con los siguientes componentes:
  - Desarrollo de escritorio con .NET
  - Windows Forms .NET Framework o .NET 6/7, según elección del estudiante
  - .NET SDK correspondiente a la versión seleccionada (recomendado: .NET 6 o .NET 7)

Recursos digitales

- Acceso a Internet para descarga de herramientas (en caso de ser necesario)
- Carpeta asignada o unidad compartida para guardar el proyecto generado

Cuenta de Microsoft (opcional, para sincronización de Visual Studio)

PROCEDIMIENTO O METODOLOGÍA

Gestión de la información con colecciones de datos

¿Qué es una colección de datos?

Es una estructura que permite almacenar y manipular varios elementos de manera organizada. En este caso, se usa una **colección de TextBox** para gestionar los campos de entrada.

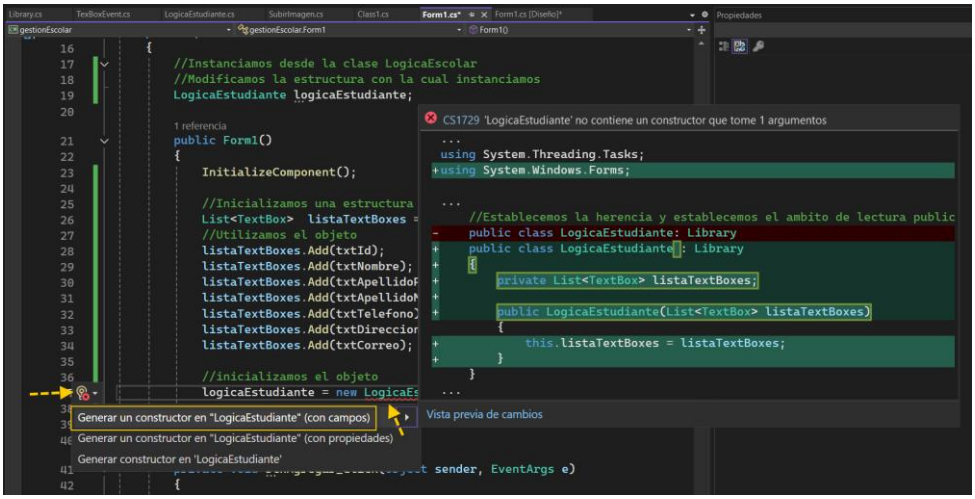
Ejemplo de código para agregar elementos a una colección:

- En el código podremos observar, que dentro del constructor **Form1(),** se inicializa la interfaz gráfica llamando a **InitializeComponent().** Luego, se crea una lista de tipo **List<TextBox>,** denominada **listaTextBoxes**, para almacenar referencias a varios controles **TextBox** presentes en el formulario, los cuales representan diferentes atributos de un estudiante como **Id**, **Nombre**, **ApellidoPaterno**, **ApellidoMaterno**, **Telefono**, **Direccion** y **Correo**. Posteriormente, se instancia un objeto de la clase **LogicaEstudiante**, pasándole como argumento la lista de TextBox, lo que sugiere que esta clase manejará la lógica relacionada con la gestión de los datos ingresados en los cuadros de texto.

NOTA: se resalta en color rojo, el objeto, y esto se debe a que todavía no tenemos un constructor, el cual agregue los argumentos

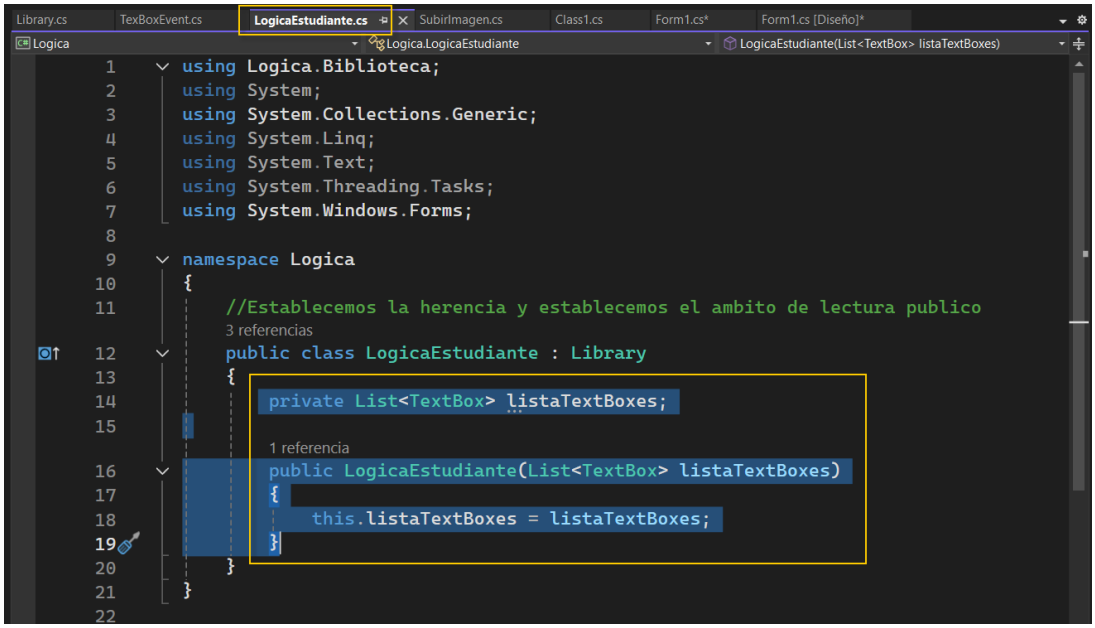
```
//Instanciamos desde la clase LogicaEscolar
//Modificamos la estructura con la cual instanciamos
LogicaEstudiante logicaEstudiante;

public Form1()
{
    InitializeComponent();

    //Inicializamos una estructura de datos
    List<TextBox>  listaTextBoxes = new List<TextBox>(); ;
    //Utilizamos el objeto
    listaTextBoxes.Add(txtId);
    listaTextBoxes.Add(txtNombre);
    listaTextBoxes.Add(txtApellidoPaterno);
    listaTextBoxes.Add(txtApellidoMaterno);
    listaTextBoxes.Add(txtTelefono);
    listaTextBoxes.Add(txtDireccion);
    listaTextBoxes.Add(txtCorreo);

    //inicializamos el objeto
    logicaEstudiante = new LogicaEstudiante(listaTextBoxes);

}
```

Crear el Contructor utilizando la ayuda del IDE



Nos dirigiremos hacia la clase que se identifica como **“LogicaEstudiante.cs”,** en donde podremos observar que se aplicaron cambios en la clase, los cuales son los siguientes:

- La clase **LogicaEstudiante** hereda de **Library** y está diseñada para gestionar una lista de objetos **TextBox**, lo que indica que forma parte de una interfaz gráfica de usuario.
- Dentro de la clase, se declara un campo privado **listaTextBoxes** de tipo **List<TextBox>,** el cual almacena una colección de cajas de texto.
- El **constructor** de la clase recibe como parámetro una lista de **TextBox** y la asigna al campo correspondiente, permitiendo así que la clase manipule estos elementos según sea necesario.
- Esta estructura facilita la agrupación y gestión de múltiples cajas de texto en una aplicación, proporcionando una forma organizada de trabajar con entradas de usuario dentro de una interfaz gráfica.

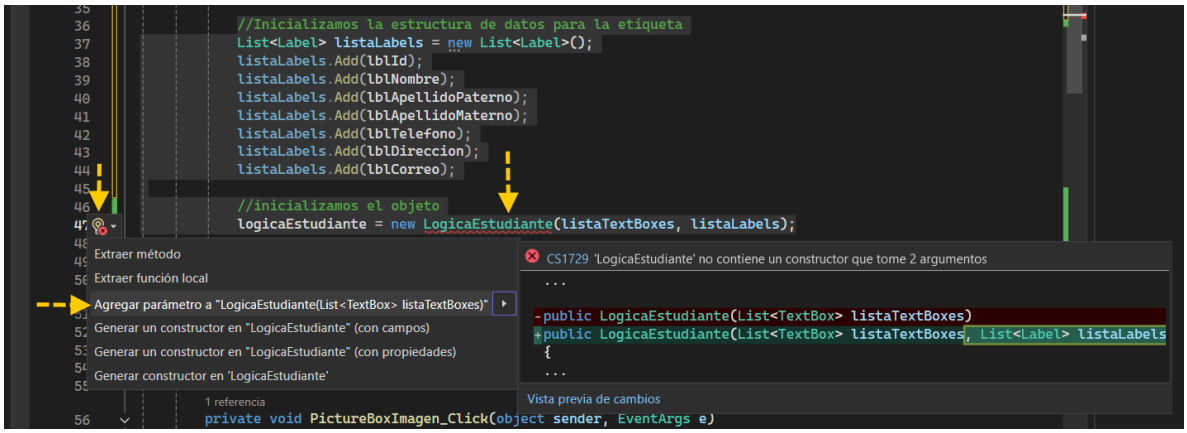


Nos dirigiremos nuevamente a la clase que se identifica como **“Form1.cs”**

- En este fragmento de código en C#, se inicializa una lista de etiquetas (**labels**) para asociarlas con un conjunto de cuadros de texto utilizados en la interfaz gráfica de usuario. Se crea una lista de objetos de tipo Label, denominada **listaLabels**, y a esta se le agregan diversas etiquetas como **lblId**, **lblNombre**, **lblApellidoPaterno**, **lblApellidoMaterno**, **lblTelefono**, **lblDireccion** y **lblCorreo**, cada una representando un campo de información correspondiente a un estudiante.
- Posteriormente, se instancia un objeto de la clase **LogicaEstudiante**, al que se le pasan dos listas como parámetros: **listaTextBoxes**, que agrupa los cuadros de texto donde se ingresarán los datos del **estudiante**, y **listaLabels**, que contiene las etiquetas descriptivas de cada campo. Este diseño permite estructurar la información de manera organizada y facilitar la interacción del usuario con los datos en la interfaz gráfica.

```
{
    //Instanciamos desde la clase LogicaEscolar
    //Modificamos la estructura con la cual instanciamos
    LogicaEstudiante logicaEstudiante;

    1 referencia
    public Form1()
    {
        InitializeComponent();

        //Inicializamos una estructura de datos
        List<TextBox>  listaTextBoxes = new List<TextBox>(); ;
        //Utilizamos el objeto
        listaTextBoxes.Add(txtId);
        listaTextBoxes.Add(txtNombre);
        listaTextBoxes.Add(txtApellidoPaterno);
        listaTextBoxes.Add(txtApellidoMaterno);
        listaTextBoxes.Add(txtTelefono);
        listaTextBoxes.Add(txtDireccion);
        listaTextBoxes.Add(txtCorreo);

        //Inicializamos la estructura de datos para la etiqueta
        List<Label> listaLabels = new List<Label>();
        listaLabels.Add(lblId);
        listaLabels.Add(lblNombre);
        listaLabels.Add(lblApellidoPaterno);
        listaLabels.Add(lblApellidoMaterno);
        listaLabels.Add(lblTelefono);
        listaLabels.Add(lblDireccion);
        listaLabels.Add(lblCorreo);

        //inicializamos el objeto
        logicaEstudiante = new LogicaEstudiante(listaTextBoxes, listaLabels);

    }
```

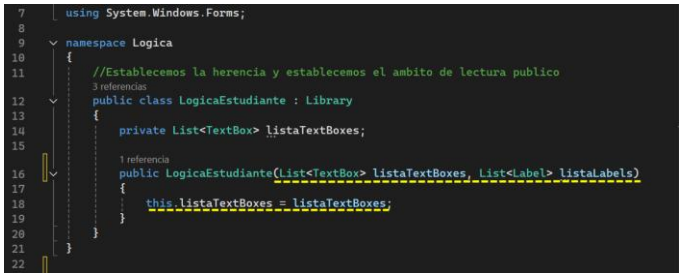
Modificar el Contructor utilizando la ayuda del IDE, utilizando la ayuda, podremos observar que esta nos permite agregar parámetros al constructor



Nos dirigiremos hacia la clase, que se indica como **“LogicaEstudiante.cs”** y podremos observar lo siguiente:

- El constructor **LogicaEstudiante** en C# está diseñado para inicializar una instancia de la clase, recibiendo dos **listas** como parámetros: una de **TextBox** y otra de **Label**, que

representan elementos de la interfaz gráfica de usuario. Sin embargo, dentro del cuerpo del constructor, solo se asigna la lista de TextBox a la variable de instancia **this.listaTextBoxes**, mientras que la lista de Label no se almacena ni se utiliza, lo que podría indicar un posible error en la implementación o una omisión en la asignación.



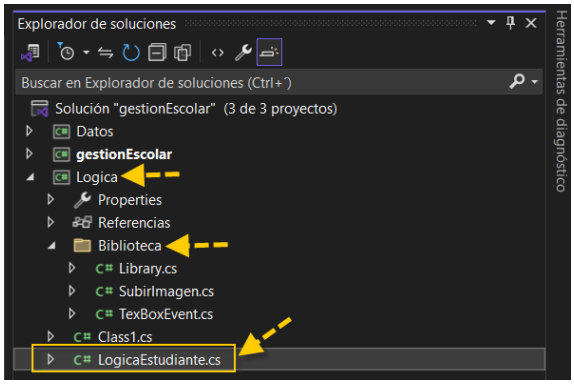
Declarar Atributo y asignar el valor del parámetro

- En el fragmento de código presentado en el lenguaje de programación C#, se declara un atributo privado denominado **listaLabel**, que es una lista de objetos de tipo **Label**. Posteriormente, en el constructor de la clase **LogicaEstudiante**, se recibe como parámetros dos listas: una de **TextBox** y otra de **Label**. Dentro del constructor, los valores de estos parámetros se asignan a los atributos correspondientes de la clase, permitiendo así almacenar referencias a los controles visuales que serán utilizados en la lógica del programa.

```
//Establecemos la herencia y establecemos el ámbito de lectura publico
public class LogicaEstudiante : Library
{
    private List<TextBox> listaTextBoxes;
    //Declaramos un atributo privado
    private List<Label> listaLabel;

    public LogicaEstudiante(List<TextBox> listaTextBoxes, List<Label> listaLabels)
    {
        this.listaTextBoxes = listaTextBoxes;
        //Asignamos el valor del parámetro hacia el atributo
        this.listaLabel = listaLabels;
    }
}
```

Validación de Datos en Formularios: Control de Entrada y Retroalimentación Visual

- Nos dirigiremos hacia la clase que se indica como **“LogicaEstudiante.cs”**



- El método público, que se muestra en el siguiente fragmento de código y el cual se identifica como **registrarDatosFormulario ()** se encarga de validar la información ingresada en un formulario mediante el uso de una lista de cajas de texto (**listaTextBoxes**) y etiquetas (**listaLabel**). En su implementación, verifica si la primera caja de texto está vacía utilizando el método **Equals ("").** Si la condición se cumple, se asigna un mensaje de error en la etiqueta correspondiente, indicando que el valor es requerido, y se cambia el color del texto a rojo para resaltar la advertencia. Adicionalmente, se establece el enfoque en la caja de texto vacía para facilitar la corrección por parte del usuario. En caso contrario, el código deja abierta la posibilidad de continuar con la validación de los siguientes campos del formulario, lo que sugiere la necesidad de ampliar la lógica para garantizar la integridad de los datos ingresados.

```
//Crearemos un metodo publico
public void registrarDatosFormulario()
{
    if (listaTextBoxes[0].Text.Equals(""))
    {
        listaLabel[0].Text = "El valor es requerido";
        listaLabel[0].ForeColor = Color.Red;
        //Movemos el enfoque a la caja de texto
        listaTextBoxes[0].Focus();
    }
    else
    {
        //Continuamos evaluando la segunda caja de texto
    }
}
```

Validación Secuencial de Campos en un Formulario de Entrada: Segunda caja de texto

- En este fragmento de código, se implementa una validación secuencial para un conjunto de cajas de texto dentro de una aplicación en C#. Si la primera condición previa no se cumple, se evalúa el contenido de la segunda caja de texto dentro de la lista **listaTextBoxes**. Si esta caja de texto está vacía, se notifica al usuario mediante la asignación de un mensaje en el segundo elemento de la **lista** listaLabel, indicando que el campo "**Nombre**" es obligatorio y cambiando el color del texto a rojo para resaltar el error. Además, se redirige el foco a la caja de texto vacía para facilitar la corrección del usuario. En caso de que la segunda caja de texto contenga un valor, el programa continúa con la evaluación del siguiente campo de entrada.

```
public void registrarDatosFormulario()
{
    if (listaTextBoxes[0].Text.Equals(""))
    {
        listaLabel[0].Text = "El valor ID es requerido";
        listaLabel[0].ForeColor = Color.Red;
        //Movemos el enfoque a la caja de texto
        listaTextBoxes[0].Focus();
    }
    else
    {
        //Continuamos evaluando la segunda caja de texto
        if (listaTextBoxes[1].Text.Equals(""))
        {
            //Enviamos el mensaje al usuario
            listaLabel[1].Text = "El valor del Nombre es requerido";
```

```
                listaLabel[1].ForeColor = Color.Red;
                //Enviamos el foco a la caja de texto
                listaTextBoxes[1].Focus();
            }
            else
            {
                //Continuamos con la evaluación de la tercer caja de texto
            }
        }
    }
```

Validación Secuencial de Campos en un Formulario de Entrada: Tercer Caja de Texto

- El fragmento de código en C# evalúa si la tercera caja de texto de una lista, representada por **listaTextBoxes[2],** se encuentra vacía, y en caso afirmativo, notifica al usuario con un mensaje en un **Label** correspondiente **(listaLabel[2]),** estableciendo el texto **"El valor Apellido Paterno es requerido"** en color rojo para enfatizar la advertencia; además, dirige el foco a la caja de texto vacía para facilitar su corrección inmediata; si el campo no está vacío, el código continúa con la validación de la siguiente caja de texto, lo que sugiere una estructura secuencial de verificación de datos en un formulario.

```
public void registrarDatosFormulario()
{
    if (listaTextBoxes[0].Text.Equals(""))
    {
        listaLabel[0].Text = "El valor ID es requerido";
        listaLabel[0].ForeColor = Color.Red;
        //Movemos el enfoque a la caja de texto
        listaTextBoxes[0].Focus();
    }
    else
    {
        //Continuamos evaluando la segunda caja de texto
        if (listaTextBoxes[1].Text.Equals(""))
        {
            //Enviamos el mensaje al usuario
            listaLabel[1].Text = "El valor del Nombre es requerido";
            listaLabel[1].ForeColor = Color.Red;
            //Enviamos el foco a la caja de texto
            listaTextBoxes[1].Focus();
        }
        else
        {
            //Continuamos con la evaluación de la tercer caja de texto
            if (listaTextBoxes[2].Text.Equals(""))
            {
                //Enviamos mensaje al usuario
                listaLabel[2].Text = "El valor Apellido Paterno es requerido";
                listaLabel[2].ForeColor = Color.Red;
                //Enviamos el foco a la caja de texto vacía
                listaTextBoxes[2].Focus();
            }
            else
            {
                //Continuamos con la evaluación a la cuarta caja de texto
            }
        }
    }
}
```

Validación Secuencial de Campos en un Formulario de Entrada: Cuarta Caja de Texto

- El fragmento de código en C# realiza una validación secuencial de campos de entrada en una interfaz gráfica, asegurando que los usuarios completen correctamente los datos requeridos. En este caso, dentro de una estructura condicional else, se evalúa si la cuarta caja de texto dentro de la lista **listaTextBoxes** está vacía mediante el método **Equals("").** Si

la condición se cumple, se notifica al usuario asignando un mensaje de advertencia al cuarto elemento de **listaLabel**, indicando que el campo **"Apellido Materno"** es obligatorio, y se cambia el color del texto a rojo **(Color.Red)** para destacar la advertencia visualmente. Además, se utiliza el **método Focus()** para trasladar el cursor a la caja de texto vacía, facilitando al usuario la corrección del error. En caso contrario, el flujo del programa continúa con la evaluación de la siguiente caja de texto sin interrupciones.

```
public void registrarDatosFormulario()
{
    if (listaTextBoxes[0].Text.Equals(""))
    {
        listaLabel[0].Text = "El valor ID es requerido";
        listaLabel[0].ForeColor = Color.Red;
        //Movemos el enfoque a la caja de texto
        listaTextBoxes[0].Focus();
    }
    else
    {
        //Continuamos evaluando la segunda caja de texto
        if (listaTextBoxes[1].Text.Equals(""))
        {
            //Enviamos el mensaje al usuario
            listaLabel[1].Text = "El valor del Nombre es requerido";
            listaLabel[1].ForeColor = Color.Red;
            //Enviamos el foco a la caja de texto
            listaTextBoxes[1].Focus();
        }
        else
        {
            //Continuamos con la evaluación de la tercer caja de texto
            if (listaTextBoxes[2].Text.Equals(""))
            {
                //Enviamos mensaje al usuario
                listaLabel[2].Text = "El valor Apellido Paterno es requerido";
                listaLabel[2].ForeColor = Color.Red;
                //Enviamos el foco a la caja de texto vacía
                listaTextBoxes[2].Focus();
            }
            else
            {
                //Continuamos con la evaluación a la cuarta caja de texto
                if (listaTextBoxes[3].Text.Equals(""))
                {
                    //Notificaremos al usuario
                    listaLabel[3].Text = "El valor Apellido Materno es requerido";
                    listaLabel[3].ForeColor = Color.Red;
                    //Enviamos el foco a la caja de texto vacía
                    listaTextBoxes[3].Focus();
                }
                else
                {
                    //Continuamos evaluando a la quinta caja de texto
                }
            }
        }
    }
}
```

Validación Secuencial de Campos en un Formulario de Entrada: Quinta Caja de Texto

El fragmento de código presentado forma parte de una estructura de control condicional utilizada en un entorno de programación orientado a objetos, específicamente en C#. Su función principal es

validar que el quinto campo de entrada en una lista de cajas de texto no esté vacío antes de continuar con la evaluación de los siguientes campos. Si el valor de la quinta caja de texto es una cadena vacía, el programa notifica al usuario mediante la modificación de un mensaje de error en un label asociado, cambiando su color a rojo para resaltar la advertencia. Además, se redirige automáticamente el foco a la caja de texto vacía para facilitar la corrección por parte del usuario. En caso de que el campo contenga un valor, la estructura de control indica que el proceso continuará con la validación del siguiente campo de entrada. Este enfoque es útil en la validación de formularios, asegurando que los datos esenciales sean ingresados antes de proceder con el flujo normal del programa.

```
public void registrarDatosFormulario()
{
    if (listaTextBoxes[0].Text.Equals(""))
    {
        listaLabel[0].Text = "El valor ID es requerido";
        listaLabel[0].ForeColor = Color.Red;
        //Movemos el enfoque a la caja de texto
        listaTextBoxes[0].Focus();
    }
    else
    {
        //Continuamos evaluando la segunda caja de texto
        if (listaTextBoxes[1].Text.Equals(""))
        {
            //Enviamos el mensaje al usuario
            listaLabel[1].Text = "El valor del Nombre es requerido";
            listaLabel[1].ForeColor = Color.Red;
            //Enviamos el foco a la caja de texto
            listaTextBoxes[1].Focus();
        }
        else
        {
            //Continuamos con la evaluación de la tercer caja de texto
            if (listaTextBoxes[2].Text.Equals(""))
            {
                //Enviamos mensaje al usuario
                listaLabel[2].Text = "El valor Apellido Paterno es requerido";
                listaLabel[2].ForeColor = Color.Red;
                //Enviamos el foco a la caja de texto vacía
                listaTextBoxes[2].Focus();
            }
            else
            {
                //Continuamos con la evaluación a la cuarta caja de texto
                if (listaTextBoxes[3].Text.Equals(""))
                {
                    //Notificaremos al usuario
                    listaLabel[3].Text = "El valor Apellido Materno es requerido";
                    listaLabel[3].ForeColor = Color.Red;
                    //Enviamos el foco a la caja de texto vacía
                    listaTextBoxes[3].Focus();
                }
                else
                {
                    //Continuamos evaluando a la quinta caja de texto
                    if (listaTextBoxes[4].Text.Equals(""))
                    {
                        //Notificaremos al usuario
                        listaLabel[4].Text = "El valor telefono es requerido";
                        listaLabel[4].ForeColor = Color.Red;
                        //Enviamos el foco a la caja de texto vacía
                        listaTextBoxes[4].Focus();
                    }
                    else
                    {
                        //Continuamos evaluando a la sexta caja de texto
                    }
                }
            }
        }
    }
}
```

Validación Secuencial de Campos en un Formulario de Entrada: Sexta Caja de Texto

El fragmento de código en C# forma parte de una secuencia de validación en la que se verifica si el usuario ha ingresado información en una serie de cajas de texto (**TextBox**). En este caso, se evalúa

específicamente la sexta caja de texto contenida en la lista **listaTextBoxes**. Si el contenido de esta caja de texto está vacío **("")**, se genera un mensaje de advertencia en el sexto elemento de **listaLabel**, indicando al usuario que el campo de dirección es obligatorio, y se cambia el color del texto de la etiqueta a rojo para resaltar la alerta visualmente. Además, se establece el foco en la caja de texto correspondiente para facilitar la corrección del error. Si la caja de texto contiene un valor, la validación continúa con la siguiente caja de texto, lo que sugiere que esta estructura forma parte de un proceso secuencial de verificación de datos ingresados por el usuario en un formulario.

```
public void registrarDatosFormulario()
{
    if (listaTextBoxes[0].Text.Equals(""))
    {
        listaLabel[0].Text = "El valor ID es requerido";
        listaLabel[0].ForeColor = Color.Red;
        //Movemos el enfoque a la caja de texto
        listaTextBoxes[0].Focus();
    }
    else
    {
        //Continuamos evaluando la segunda caja de texto
        if (listaTextBoxes[1].Text.Equals(""))
        {
            //Enviamos el mensaje al usuario
            listaLabel[1].Text = "El valor del Nombre es requerido";
            listaLabel[1].ForeColor = Color.Red;
            //Enviamos el foco a la caja de texto
            listaTextBoxes[1].Focus();
        }
        else
        {
            //Continuamos con la evaluación de la tercer caja de texto
            if (listaTextBoxes[2].Text.Equals(""))
            {
                //Enviamos mensaje al usuario
                listaLabel[2].Text = "El valor Apellido Paterno es requerido";
                listaLabel[2].ForeColor = Color.Red;
                //Enviamos el foco a la caja de texto vacía
                listaTextBoxes[2].Focus();
            }
            else
            {
                //Continuamos con la evaluación a la cuarta caja de texto
                if (listaTextBoxes[3].Text.Equals(""))
                {
                    //Notificaremos al usuario
                    listaLabel[3].Text = "El valor Apellido Materno es requerido";
                    listaLabel[3].ForeColor = Color.Red;
                    //Enviamos el foco a la caja de texto vacía
                    listaTextBoxes[3].Focus();
                }
                else
                {
                    //Continuamos evaluando a la quinta caja de texto
                    if (listaTextBoxes[4].Text.Equals(""))
                    {
                        //Notificaremos al usuario
                        listaLabel[4].Text = "El valor telefono es requerido";
                        listaLabel[4].ForeColor = Color.Red;
                        //Enviamos el foco a la caja de texto vacía
                        listaTextBoxes[4].Focus();
                    }
                    else
                    {
                        //Continuamos evaluando a la sexta caja de texto
                        if (listaTextBoxes[5].Text.Equals(""))
                        {
                            //Notificaremos al usuario
                            listaLabel[5].Text = "El valor dirección es requerido";
                            listaLabel[5].ForeColor = Color.Red;
                            //Enviamos el foco
                            listaTextBoxes[5].Focus();
```

```
                    }
                    else
                    {
                        //Continuamos evaluando a la séptima caja de texto
                    }
                }
            }
        }
    }
}
```

Validación Secuencial de Campos en un Formulario de Entrada: Séptima Caja de Texto

En este fragmento de código, se evalúa si el séptimo cuadro de texto en una lista de controles de entrada **(listaTextBoxes[6])** está vacío, lo que indica que el usuario no ha ingresado un valor requerido, en este caso, un número de teléfono. Si la condición se cumple, se genera un mensaje de advertencia en la etiqueta correspondiente **(listaLabel[6]),** estableciendo su texto en **"El valor del teléfono es requerido"** y cambiando su color a rojo para resaltar la alerta. Posteriormente, se emplea el método **Focus()** para dirigir automáticamente el cursor al cuadro de texto vacío, facilitando al usuario la corrección del error sin necesidad de desplazarse manualmente. Esta implementación mejora la experiencia del usuario al proporcionar retroalimentación inmediata y guiarlo en el llenado correcto de los datos requeridos.

```
//Crearemos un metodo publico
public void registrarDatosFormulario()
{
    if (listaTextBoxes[0].Text.Equals(""))
    {
        listaLabel[0].Text = "El valor ID es requerido";
        listaLabel[0].ForeColor = Color.Red;
        //Movemos el enfoque a la caja de texto
        listaTextBoxes[0].Focus();
    }
    else
    {
        //Continuamos evaluando la segunda caja de texto
        if (listaTextBoxes[1].Text.Equals(""))
        {
            //Enviamos el mensaje al usuario
            listaLabel[1].Text = "El valor del Nombre es requerido";
            listaLabel[1].ForeColor = Color.Red;
            //Enviamos el foco a la caja de texto
            listaTextBoxes[1].Focus();
        }
        else
        {
            //Continuamos con la evaluación de la tercer caja de texto
            if (listaTextBoxes[2].Text.Equals(""))
            {
                //Enviamos mensaje al usuario
                listaLabel[2].Text = "El valor Apellido Paterno es requerido";
                listaLabel[2].ForeColor = Color.Red;
                //Enviamos el foco a la caja de texto vacia
                listaTextBoxes[2].Focus();
            }
            else
            {
                //Continuamos con la evaluación a la cuarta caja de texto
                if (listaTextBoxes[3].Text.Equals(""))
                {
                    //Notificaremos al usuario
                    listaLabel[3].Text = "El valor Apellido Materno es requerido";
                    listaLabel[3].ForeColor = Color.Red;
                    //Enviamos el foco a la caja de texto vacía
                    listaTextBoxes[3].Focus();
                }
                else
                {
                    //Continuamos evaluando a la quinta caja de texto
```

```
                    if (listaTextBoxes[4].Text.Equals(""))
                    {
                        //Notificaremos al usuario
                        listaLabel[4].Text = "El valor telefono es requerido";
                        listaLabel[4].ForeColor = Color.Red;
                        //Enviamos el foco a la caja de texto vacia
                        listaTextBoxes[4].Focus();
                    }
                    else
                    {
                        //Continuamos evaluando a la sexta caja de texto
                        if (listaTextBoxes[5].Text.Equals(""))
                        {
                            //Notificaremos al usuario
                            listaLabel[5].Text = "El valor dirección es requerido";
                            listaLabel[5].ForeColor = Color.Red;
                            //Enviamos el foco
                            listaTextBoxes[5].Focus();
                        }
                        else
                        {
                            //Continuamos evaluando a la séptima caja de texto
                            if (listaTextBoxes[6].Text.Equals(""))
                            {
                                //Notificaremos al usuario
                                listaLabel[6].Text = "El valor del telefono es requerido";
                                listaLabel[6].ForeColor = Color.Red;
                                //Enviamos el foco, a la caja de texto vacía
                                listaTextBoxes[6].Focus();
                            }
                        }
                    }
                }
            }
        }
    }
}
```

Manejo de Eventos y Modularización en Windows Forms

Dentro de la clase Form1.cs, se define un método asociado al evento de clic del botón btnAgregar en una aplicación de Windows Forms. Al ejecutarse este evento, se invoca el método **registrarDatosFormulario ()** a través del objeto **logicaEstudiante**, lo que sugiere que este método se encarga de procesar y registrar los datos ingresados en el formulario por el usuario. Este enfoque promueve la separación de responsabilidades, ya que la lógica de negocio se encapsula dentro de la clase logicaEstudiante, evitando que la interfaz de usuario gestione directamente los datos y permitiendo una estructura más modular y mantenible en la aplicación.

```
private void btnAgregar_Click(object sender, EventArgs e)
 {
     //Utilizando el objeto, invocamos el metodo
     logicaEstudiante.registrarDatosFormulario();
 }
```

Ejecutamos y podremos observar que al momento en el que el usuario no coloque un valor en una de las cajas de texto, este se indica en color rojo y se muestra una leyenda, la cual le informa que el valor del campo es requerido

RESULTADOS ESPERADOS

- El estudiante agrupa y gestiona controles TextBox y Label mediante colecciones para facilitar la validación de datos.
- Implementa una validación secuencial que recorre los campos del formulario, identificando valores vacíos y generando retroalimentación visual.
- Desarrolla métodos que centralizan la lógica de validación, mejorando la organización y reutilización del código.
- Aplica principios de encapsulamiento y separación de responsabilidades al distribuir la lógica en clases específicas.
- Demuestra capacidad para guiar al usuario en el llenado correcto del formulario mediante mensajes claros y resaltado visual.

ANÁLISIS DE RESULTADOS

- ¿Qué ventajas encontraste al utilizar listas de controles para validar los datos del formulario?
- ¿Cómo influye la validación secuencial en la experiencia del usuario al completar un formulario?
- ¿La retroalimentación visual facilitó la detección de errores? ¿Por qué?
- ¿Qué beneficios identificas al separar la lógica de validación de la interfaz gráfica?
- ¿Qué mejoras podrías aplicar al método de validación para hacerlo más flexible o reutilizable?
- ¿Cómo contribuye esta técnica al mantenimiento y escalabilidad de la aplicación?
- ¿Qué dificultades enfrentaste al trabajar con colecciones de objetos y cómo las resolviste?

CONCLUSIONES Y REFLEXIONES

La práctica permitió comprender la utilidad de las colecciones para gestionar múltiples controles en formularios, facilitando una validación más estructurada y eficiente. La retroalimentación visual contribuyó a mejorar la experiencia del usuario y la precisión de los datos. Asimismo, separar la lógica

de validación de la interfaz promovió una mejor organización del código y mayor facilidad para su mantenimiento y reutilización.

ACTIVIDADES COMPLEMENTARIAS
1. Validación dinámica mediante bucles: • Modificar el método de validación para recorrer automáticamente todas las cajas de texto mediante un ciclo for, evitando estructuras condicionales anidadas. 2. Gestión de formularios con pestañas (TabControl): • Diseñar un formulario con varias secciones organizadas por pestañas y aplicar validación secuencial en cada una de ellas utilizando colecciones separadas por grupo. 3. Retroalimentación positiva con validación progresiva: • Implementar indicadores visuales (por ejemplo, cambio de color a verde en etiquetas) cuando el usuario completa correctamente cada campo, fomentando la interacción intuitiva.

EVALUACIÓN Y EVIDENCIAS DE APRENDIZAJE	
Criterios de evaluación	• Declara e inicializa correctamente colecciones de controles TextBox y Label en el formulario. • Asocia las colecciones a una clase lógica externa mediante un constructor bien estructurado. • Implementa validaciones secuenciales que detectan campos vacíos y generan retroalimentación visual efectiva. • Utiliza estructuras condicionales apropiadas para verificar el contenido de los campos de entrada. • Emplea correctamente el método Focus() para guiar al usuario en la corrección de errores. • Mantiene una separación clara entre la lógica de validación y la interfaz gráfica. • Aplica principios de reutilización y organización de código en el desarrollo de la funcionalidad. • Refleja comprensión del flujo de validación y sus efectos en la experiencia del usuario.
Rúbricas o listas de cotejo para valorar desempeño	Lista de cotejo con siete criterios clave que evalúan la creación del proyecto, el diseño de la interfaz, la funcionalidad del código, el análisis reflexivo y la presentación del reporte. Cada ítem se evalúa con las opciones: “Cumple / No cumple” y espacio para observaciones.
Formatos de reporte de prácticas	**Formato libre estructurado que incluya:** 1. Datos del alumno. 2. Nombre de la práctica. 3. Objetivo. 4. Capturas de pantalla del proyecto. 5. Código implementado. 6. Respuestas al análisis. 7. Conclusiones y Reflexión.

NOMBRE DE LA PRÁCTICA No.9	**Conversión de Imágenes a Arreglos de Bytes en Aplicaciones Windows Forms con C#**
COMPETENCIA DE LA PRÁCTICA	Convierte imágenes a arreglos de bytes para su almacenamiento o procesamiento digital, mediante el uso de métodos personalizados y control PictureBox, en un entorno de desarrollo con C# y Windows Forms, fortaleciendo la precisión técnica y la capacidad de análisis.

FUNDAMENTO TÉORICO

La práctica se fundamenta en la serialización de imágenes para su manipulación en memoria, aplicando principios de conversión de tipos en C# y uso de clases como ImageConverter. Se aborda el manejo de objetos gráficos en tiempo de ejecución y la transformación de datos visuales a estructuras binarias (byte[]), esenciales para su almacenamiento, transmisión o procesamiento en aplicaciones modernas.

MATERIALES, EQUIPAMIENTO Y/O REACTIVOS

Equipo de cómputo

- Computadora personal o de laboratorio con sistema operativo Windows 10 o superior
- Procesador mínimo: Intel Core i3 o equivalente
- Memoria RAM mínima: 8 GB
- Espacio disponible en disco: al menos 10 GB

Software

- Visual Studio 2022 (versión Community, Professional o Enterprise) con los siguientes componentes:
  - Desarrollo de escritorio con .NET
  - Windows Forms .NET Framework o .NET 6/7, según elección del estudiante
  - .NET SDK correspondiente a la versión seleccionada (recomendado: .NET 6 o .NET 7)

Recursos digitales

- Acceso a Internet para descarga de herramientas (en caso de ser necesario)
- Carpeta asignada o unidad compartida para guardar el proyecto generado

Cuenta de Microsoft (opcional, para sincronización de Visual Studio)

PROCEDIMIENTO O METODOLOGÍA

Convertir una Imagen a un Array de Bytes (byte [ ])

En el desarrollo de aplicaciones, es común la necesidad de convertir imágenes en **arrays de bytes** [1]**(byte [ ]).** Este procedimiento es útil en múltiples escenarios, como:

- **Almacenamiento en bases de datos**: Muchas bases de datos almacenan imágenes en formato binario (**BLOB**).

[1] Un arreglo de bytes es una estructura de datos que consiste en una secuencia contigua de elementos, donde cada elemento es un byte (generalmente, una unidad de 8 bits). Esto significa que cada posición en el arreglo puede almacenar un valor numérico entre 0 y 255 (en representación sin signo) o entre -128 y 127 (en representación con signo, dependiendo del lenguaje de programación).

- **Transmisión de imágenes a través de la red**: En aplicaciones web o móviles, enviar imágenes en formato **byte []** permite reducir problemas de compatibilidad.
- **Manipulación en memoria**: Convertir imágenes a bytes facilita la aplicación de algoritmos de compresión, cifrado, y edición.

A continuación, analizaremos el código que realiza esta conversión, explicando cada línea de manera detallada para asegurar una comprensión profunda.

Código Completo
Nos dirigiremos hacia la clase **UploadImage.cs,** en donde crearemos el **método** que implementa la conversión de la imagen:

```
//Crearemos un metodo público, para convertir la imagen
public byte[] ImageToByte(Image img)
{
    var converter = new ImageConverter();
    return (byte[])converter.ConvertTo(img, typeof(byte[]));
}
```

1. Definición del Método

public byte[] ImageToByte(Image img)

Explicación:

- **public:** El modificador de acceso **public** permite que el método sea accesible desde cualquier parte del código donde se instancie la clase.
- **byte [ ]:** Es el tipo de retorno del método, indicando que devuelve un arreglo de bytes.
- **ImageToByte (Image img):** Define el nombre del método **ImageToByte** y especifica que recibe un parámetro **img** de tipo Image. La imagen será el objeto a convertir en un array de bytes.

2. Creación de un Convertidor de Imágenes

var converter = new ImageConverter();

Explicación:

- **var converter:** Se declara una variable utilizando inferencia de tipos (**var**), que almacenará un objeto del tipo **ImageConverter**.
- **new ImageConverter ( ):** Se crea una nueva instancia de **ImageConverter**[2], una clase del espacio de nombres **System.Drawing** que permite convertir imágenes a otros formatos.

[2] La clase ImageConverter es parte del espacio de nombres System.Drawing y hereda de TypeConverter. Su función principal es proporcionar métodos para convertir objetos de tipo Image a otras representaciones (por ejemplo, un arreglo de bytes) y viceversa. Esto facilita la serialización, deserialización y manipulación de imágenes en aplicaciones desarrolladas con .NET, especialmente en entornos como Windows Forms.

3. Conversión de la Imagen a un Array de Bytes

```
return (byte[])converter.ConvertTo(img, typeof(byte[]));
```

Explicación:

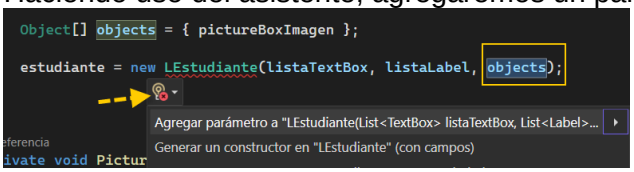
- **converter.ConvertTo(img, typeof(byte[])):**
  - El método **ConvertTo ( )** de la clase **ImageConverter** transforma el **objeto img** en un array de bytes **(byte [ ]).**
  - **typeof (byte [ ]):** Indica que la conversión debe realizarse a un array de bytes.
- **(byte[]):**
  - Se realiza un **casting explícito** para asegurarse de que el objeto devuelto por **ConvertTo ()** sea del tipo **byte [].**

Invocación del Metodo

Nos dirigiremos hacia el archivo que se indica como **Form1.cs**, y lo que haremos, será **crear un arreglo** de tipo objects

```
//Arreglo de objetos
Object[] objects = { pictureBoxImagen };

estudiante = new LEstudiante(listaTextBox, listaLabel, objects);
```

Haciendo uso del asistente, agregaremos un parámetro al constructor



Explicación:

- Se está declarando un arreglo de tipo **Object []** que contiene un solo elemento: **pictureBoxImagen**.
- **Object** es la clase base de todos los tipos, lo que significa que **objects** puede almacenar cualquier tipo de dato, incluyendo controles gráficos como **PictureBox**.

Lo siguiente que haremos, será dirigirnos hacia la clase que se identifica como **LEstudiante.cs** y establecemos el siguiente atributo: **PictureBox pictureBox;**

- Inicializamos el objeto, en el interior del metodo constructor

```
public class LEstudiante : Library
{
    //Atributos
    private List<TextBox> listaTextBox;
    private List<Label> listaLabel;

    //Crearemos un objeto
    PictureBox pictureBox;

    public LEstudiante(List<TextBox> listaTextBox, List<Label> listaLabel,
```

```
        object[] objects)
        {
            //Asignacion del parámetro hacia el atributo
            this.listaTextBox = listaTextBox;
            this.listaLabel = listaLabel;

            //Inicializamos el objeto y le asignamos el parámetro
            pictureBox = (PictureBox)objects[0];


        }
```

Explicación:

- **objects [0]**: Se accede al primer elemento del arreglo **objects**.
- **(PictureBox)**: Se realiza un **casting** (conversión de tipo) para indicar que el objeto en la posición **0** del arreglo es de tipo *PictureBox*.
- **pictureBox = (PictureBox) objects [0];**
  - Asigna ese objeto convertido a la variable **pictureBox**.

Conversión de una Imagen a un Arreglo de Bytes en C#

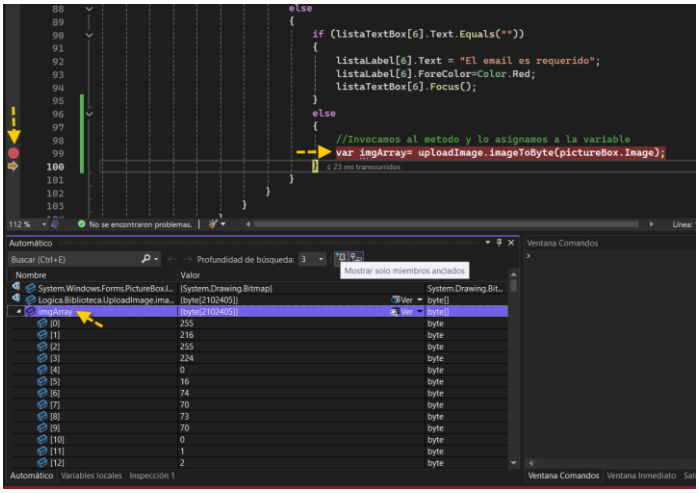
Se invoca el método **imageToByte** de la clase u objeto **uploadImage**, pasándole como argumento la imagen contenida en **pictureBox.Image.**

```
//Invocamos al metodo
Var uploadImage.imageToByte(pictureBox.Image);
```

Este código en **C#** está convirtiendo una imagen en un arreglo de bytes.

- **uploadImage.imageToByte (pictureBox.Image):** Se está llamando a un método llamado **imageToByte** del objeto **uploadImage**, el cual recibe una imagen **(pictureBox.Image)** como parámetro y la convierte en un arreglo de bytes.
- **var imgArray =...:** Se almacena el resultado en la variable **imgArray**, que contendrá la representación en bytes de la imagen.

Ejecutamos y colocamos un punto de ruptura para poder visualizar el resultado



RESULTADOS ESPERADOS

- El estudiante comprende la utilidad de convertir imágenes a arreglos de bytes (byte[]) para su uso en almacenamiento o transmisión.
- Implementa un método personalizado en C# que convierte una imagen en un arreglo de bytes utilizando la clase ImageConverter.
- Asocia correctamente un control PictureBox al proceso de conversión mediante un constructor con paso de parámetros.
- Almacena el resultado de la conversión en una variable y verifica su contenido mediante herramientas de depuración.
- Aplica correctamente técnicas de conversión de tipos (casting) y manipulación de objetos en tiempo de ejecución.

ANÁLISIS DE RESULTADOS

- ¿Qué ventajas identificaste al convertir una imagen en un arreglo de bytes en lugar de trabajar directamente con archivos?
- ¿Cómo se comportó la aplicación al ejecutar el método ImageToByte durante la depuración?
- ¿Qué tipo de errores podrían ocurrir si no se realiza el casting adecuado al extraer el PictureBox del arreglo de objetos?
- ¿De qué forma esta técnica puede aplicarse a situaciones reales como el almacenamiento de fotos en bases de datos?
- ¿Qué mejoras podrías implementar en el método de conversión para hacerlo más robusto o reutilizable?
- ¿Cómo facilita esta práctica la comprensión del manejo de memoria y tipos de datos en C#?

CONCLUSIONES Y REFLEXIONES

La práctica permitió comprender el proceso de conversión de imágenes a arreglos de bytes como un recurso esencial para el almacenamiento y manipulación digital. Se reforzó el uso del control PictureBox, la creación de métodos personalizados y la aplicación de casting de tipos. Además, se destacó la importancia de estructurar adecuadamente los datos visuales para integrarlos en soluciones informáticas eficientes y reutilizables.

ACTIVIDADES COMPLEMENTARIAS

1. Guardar la imagen convertida en una base de datos local:
   - Implementar un módulo que almacene el arreglo de bytes generado en un campo tipo varbinary de una base de datos SQL Server.
2. Reconversión del arreglo de bytes a imagen:
   - Crear un método que reciba un byte[] y reconstruya la imagen original para mostrarla nuevamente en un PictureBox.
3. Simulación de transmisión de imágenes:
   - Desarrollar una función que emule el envío del arreglo de bytes por red (por ejemplo, mediante escritura en archivo temporal o stream), y visualizar la imagen en otro formulario o sección.

EVALUACIÓN Y EVIDENCIAS DE APRENDIZAJE	
Criterios de evaluación	• Declara e implementa correctamente un método que convierte una imagen en un arreglo de bytes utilizando ImageConverter. • Asocia adecuadamente un control PictureBox mediante paso de parámetros en el constructor de la clase lógica. • Realiza el casting correcto del objeto recibido como PictureBox desde un arreglo de tipo object[]. • Invoca el método de conversión con argumentos válidos y almacena el resultado en una variable del tipo adecuado (byte[]). • Verifica y analiza el resultado de la conversión mediante el uso de herramientas de depuración. • Comprende el propósito técnico del uso de imágenes en formato binario para su manipulación o almacenamiento. • Mantiene la organización y claridad en la estructura del código, aplicando buenas prácticas de desarrollo. • Demuestra capacidad de análisis al explicar el flujo de conversión y su aplicación en contextos reales.
Rúbricas o listas de cotejo para valorar desempeño	Lista de cotejo con siete criterios clave que evalúan la creación del proyecto, el diseño de la interfaz, la funcionalidad del código, el análisis reflexivo y la presentación del reporte. Cada ítem se evalúa con las opciones: “Cumple / No cumple” y espacio para observaciones.
Formatos de reporte de prácticas	**Formato libre estructurado que incluya:** 1. Datos del alumno. 2. Nombre de la práctica. 3. Objetivo. 4. Capturas de pantalla del proyecto. 5. Código implementado. 6. Respuestas al análisis. 7. Conclusiones y Reflexión.

ELEMENTO DE COMPETENCIA III

NOMBRE DE LA PRÁCTICA No.10	**Instalación, Administración y Conexión de SQL Server desde Visual Studio con LINQ to DB**
COMPETENCIA DE LA PRÁCTICA	Configura e integra una base de datos SQL Server en una aplicación de escritorio, con la finalidad de almacenar y gestionar información de manera estructurada, utilizando LINQ to DB como herramienta de acceso a datos, en un entorno de desarrollo con Visual Studio y C#, fortaleciendo la autonomía técnica y la capacidad de resolución de problemas.

FUNDAMENTO TÉORICO

La práctica se basa en los fundamentos de los sistemas de gestión de bases de datos relacionales (RDBMS) y en la arquitectura cliente-servidor. Se aplican principios de persistencia de datos, configuración de conexiones mediante cadenas de conexión y uso de ORM (Object-Relational Mapping) a través de LINQ to DB, permitiendo integrar y administrar datos en aplicaciones .NET de forma eficiente y estructurada.

MATERIALES, EQUIPAMIENTO Y/O REACTIVOS

Equipo de cómputo

- Computadora personal o de laboratorio con sistema operativo Windows 10 o superior
- Procesador mínimo: Intel Core i3 o equivalente
- Memoria RAM mínima: 8 GB
- Espacio disponible en disco: al menos 10 GB

Software

- Visual Studio 2022 (versión Community, Professional o Enterprise) con los siguientes componentes:
  - Desarrollo de escritorio con .NET
  - Windows Forms .NET Framework o .NET 6/7, según elección del estudiante
  - .NET SDK correspondiente a la versión seleccionada (recomendado: .NET 6 o .NET 7)

Recursos digitales

- Acceso a Internet para descarga de herramientas (en caso de ser necesario)
- Carpeta asignada o unidad compartida para guardar el proyecto generado
- Cuenta de Microsoft (opcional, para sincronización de Visual Studio)

PROCEDIMIENTO O METODOLOGÍA

Instalación y Configuración de SQL Server y su Administración en Visual Studio

En este apartado del curso, exploraremos el proceso de instalación, configuración y conexión de **SQL Server** para desarrollar aplicaciones de escritorio que interactúan con bases de datos. También aprenderemos cómo administrar bases de datos utilizando **SQL Server Management Studio**

(SSMS) y cómo conectar un servidor de bases de datos desde **Visual Studio** para hacer más eficiente nuestro flujo de trabajo.

Este material está diseñado para ayudar a principiantes y programadores intermedios a comprender los fundamentos de la gestión de bases de datos SQL Server dentro de un entorno de desarrollo integrado.

1. Instalación de SQL Server y SQL Server Management Studio (SSMS)

¿Qué es SQL Server y por qué necesitamos SSMS?

SQL Server es un sistema de gestión de bases de datos relacional (**RDBMS**) desarrollado por Microsoft, ampliamente utilizado en entornos empresariales para almacenar, gestionar y manipular datos.

SQL Server Management Studio (SSMS) es la herramienta que permite administrar instancias de SQL Server de manera visual y simplificada. Desde SSMS, podemos:

- Crear y administrar bases de datos.
- Ejecutar consultas SQL.
- Configurar permisos y autenticaciones.
- Realizar respaldos y restauraciones.

Descarga e Instalación de SQL Server

Para instalar SQL Server en tu computadora, sigue estos pasos:

1. **Descarga SQL Server**
   - Visita el sitio oficial de Microsoft: https://www.microsoft.com/en-us/sql-server/sql-server-downloads
   - Descarga la versión de **SQL Server Express** (gratuita) o la versión completa según tus necesidades.
2. **Ejecuta el instalador**
   - Elige la opción **"Instalación básica"** si es tu primera vez.
   - Acepta los términos de licencia.
   - Espera a que el instalador configure el entorno.
3. **Configura la autenticación**
   - Puedes elegir entre **Autenticación de Windows** (recomendada para entornos locales) o **Autenticación de SQL Server** (para conexiones con usuario y contraseña).

Descarga e Instalación de SQL Server Management Studio (SSMS)

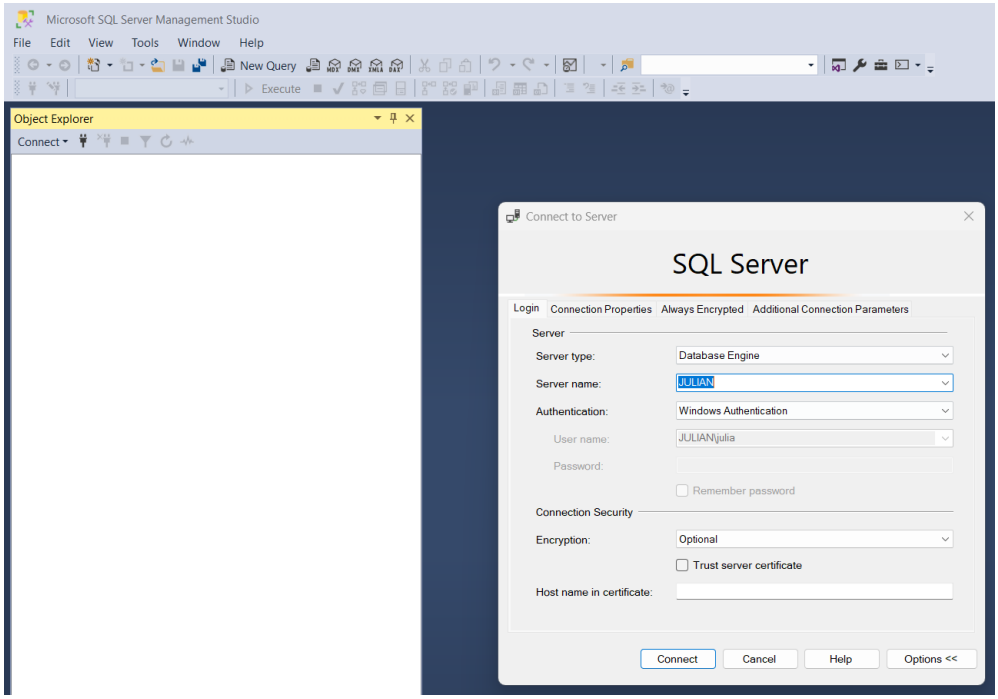
Para administrar las bases de datos más fácilmente, necesitamos instalar **SSMS**:

1. **Descarga SSMS**
   - Visita el sitio oficial: https://aka.ms/ssmsfullsetup
   - Descarga el instalador y ejecútalo.
2. **Instala SSMS**
   - Sigue las instrucciones en pantalla.
   - Una vez instalado, inicia la aplicación.

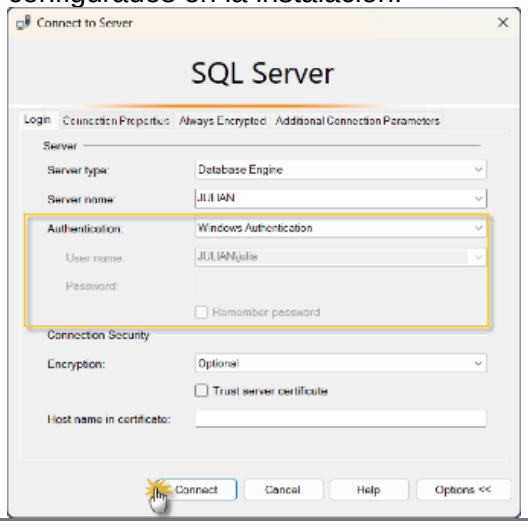
2. Conexión a SQL Server desde SSMS

Después de instalar SQL Server y SSMS, el siguiente paso es conectarnos a la base de datos:

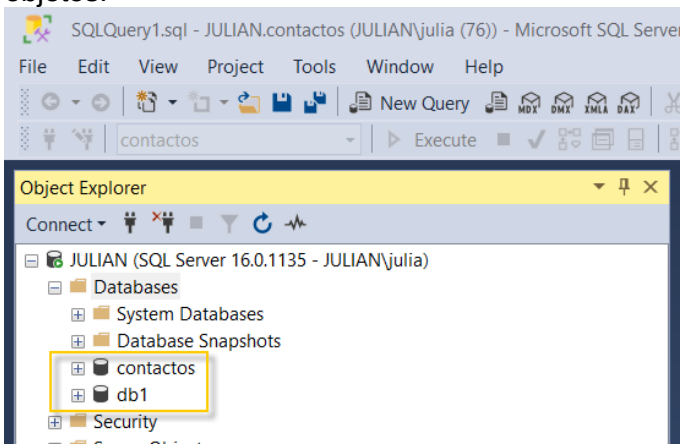
1. **Abre SSMS**
   - Se abrirá una ventana de inicio de sesión.



2. **Configura los datos de conexión**
   - **Servidor:** Si instalaste **SQL Server** en tu computadora, usa **localhost** o el nombre de tu equipo.
   - **Tipo de autenticación:**
     - **Autenticación de Windows:** Usa tus credenciales de usuario de Windows.
     - **Autenticación de SQL Server:** Introduce el usuario y contraseña configurados en la instalación.



3. **Conéctate al servidor**
    - Haz clic en **"Conectar"** y verás una lista de bases de datos en el explorador de objetos.



3. Administración de Bases de Datos en SSMS

Ahora que estamos conectados, podemos comenzar a trabajar con bases de datos.

Creación de una nueva base de datos

1. En SSMS, expande el explorador de objetos.
2. Haz clic derecho en Bases de Datos → Nueva Base de Datos.
3. Asigna un nombre a la base de datos y haz clic en "Aceptar".

Ejecutar una consulta SQL básica

Después de crear una base de datos, podemos ejecutar consultas SQL.

Ejemplo: Crear una tabla en la base de datos:

```
CREATE TABLE Empleados (
    ID INT PRIMARY KEY IDENTITY,
    Nombre VARCHAR(100),
    Edad INT,
    Departamento VARCHAR(50)
);
```

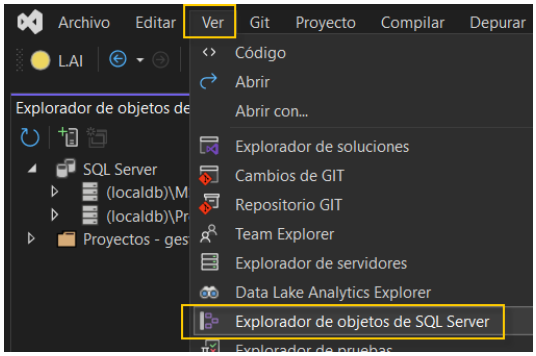
Para Ejecutar esta consulta:

1. Abre un nuevo **Editor de consultas** en SSMS.
2. Asegúrate de seleccionar la base de datos recién creada.
3. Copia y pega la consulta y presiona **Ejecutar**.
    4. Conexión de SQL Server desde Visual Studio

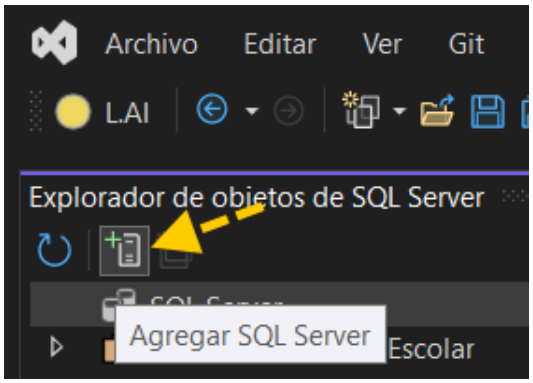
Podemos interactuar con nuestras bases de datos desde **Visual Studio**, lo cual nos permite integrarlas directamente en nuestras aplicaciones.

Pasos para conectar SQL Server desde Visual Studio

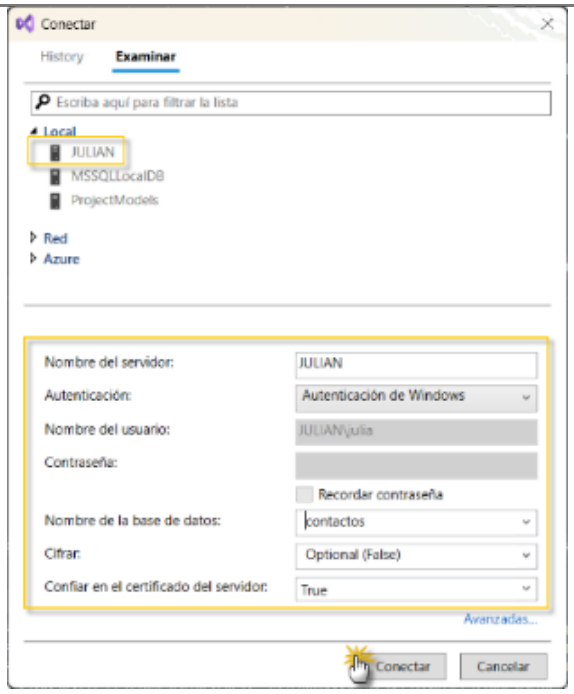
1. **Abre Visual Studio** y crea o abre un proyecto.
2. **Habilita el Explorador de Objetos de SQL Server**
   - Ve a **Ver** → **Explorador de Servidores** o **Explorador de Objetos de SQL Server**.



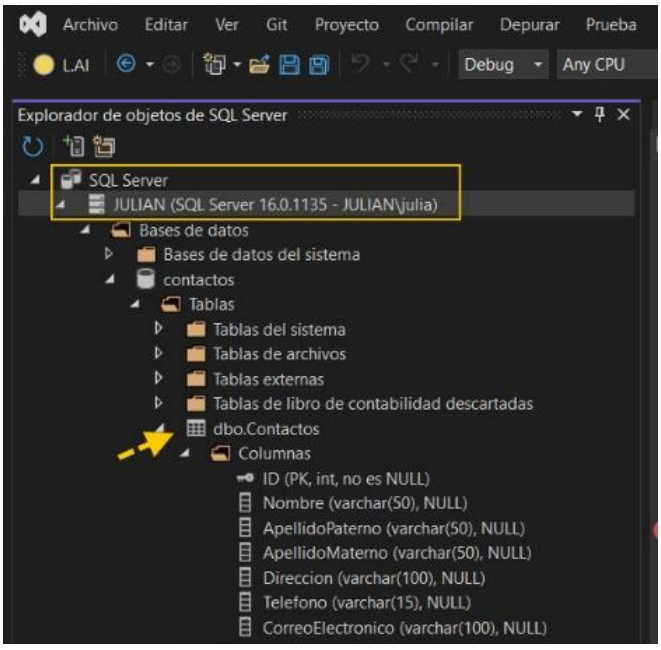
3. **Agregar una nueva conexión**
   - Haz clic en **Conectar a Base de Datos**.



   - Selecciona **Microsoft SQL Server** como origen de datos.
   - Introduce los datos de conexión:
     - **Servidor:** localhost
     - **Autenticación:** Windows o SQL Server según la configuración.
     - **Base de datos:** Elige la base de datos creada.
4. **Haz clic en Conectar** y verás la base de datos en el Explorador de Servidores.



La imagen muestra la interfaz de **Visual Studio** con el **Explorador de Objetos de SQL Server** abierto. Aquí se observa la conexión a un servidor de bases de datos SQL Server y la estructura de una base de datos específica.



1. **Conexión al servidor SQL Server**

- Se muestra el servidor **"JULIAN (SQL Server 16.0.1135 - JULIAN\Julia)"**, lo que indica que el usuario ha establecido correctamente la conexión a una instancia de SQL Server en su máquina local.

2. **Bases de datos en el servidor**
   - Dentro del servidor, se visualiza una base de datos llamada **"contactos"**, la cual está expandida para mostrar su contenido.
3. **Estructura de la base de datos "contactos"**
   - Se observa la carpeta **"Tablas"**, que contiene diferentes tipos de tablas:
     - **Tablas del sistema** (propias del sistema de SQL Server).
     - **Tablas de archivos, externas y contabilidad descartadas** (no relevantes en este contexto).
     - Una tabla de usuario llamada **"dbo.Contactos"**.
4. **Detalles de la tabla "dbo.Contactos"**
   - Se listan las columnas de la tabla, mostrando los nombres y los tipos de datos de cada campo:
     - **ID (PK, int, no es NULL):** Clave primaria, tipo entero, no permite valores nulos.
     - **Nombre (varchar(50), NULL):** Cadena de texto de hasta 50 caracteres, permite valores nulos.
     - **ApellidoPaterno (varchar(50), NULL)**
     - **ApellidoMaterno (varchar(50), NULL)**
     - **Direccion (varchar(100), NULL)**
     - **Telefono (varchar(15), NULL)**
     - **CorreoElectronico (varchar(100), NULL)**

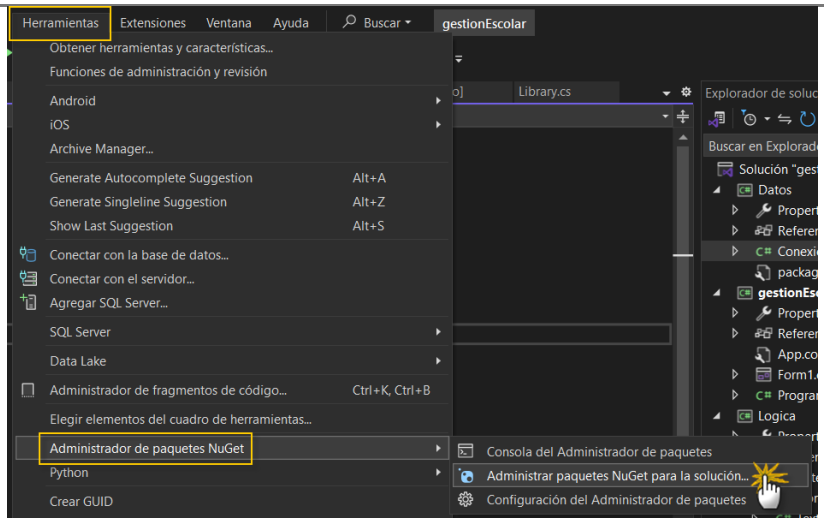
Creando la base de datos en los servidores de SQL Server y MySQL

Microsoft SQL Server es un sistema de gestión de bases de datos (DBMS) ampliamente utilizado en entornos empresariales. Para interactuar con SQL Server desde una aplicación .NET, se requiere configurar correctamente la cadena de conexión y utilizar las herramientas adecuadas para gestionar la comunicación.

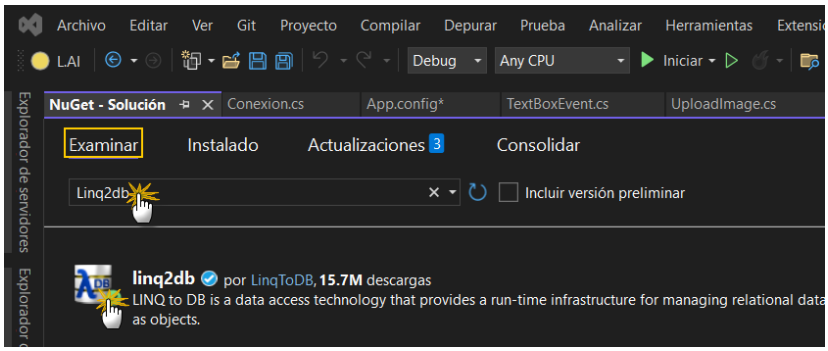
Instalación de la Librería para la Conexión a SQL Server

Pasos para instalar la librería:

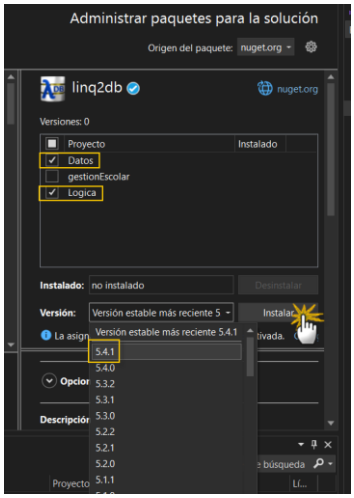
1. **Abrir Visual Studio** y cargar el proyecto.
2. **Acceder al Administrador de Paquetes NuGet:**
   - Ir a **Herramientas** → **Administrador de Paquetes NuGet** → **Administrar paquetes para la solución**.



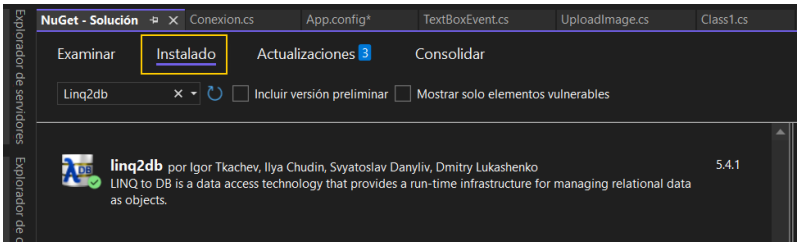
- En la pestaña **Examinar**, y buscar **linq2db**



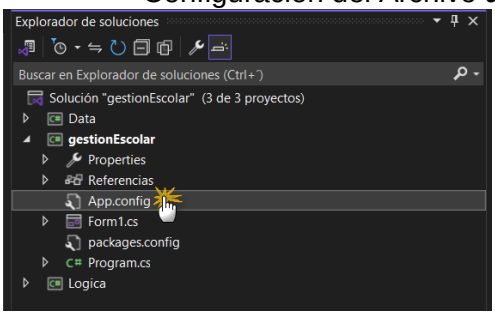
- Seleccionar la última versión disponible, en este caso es 5.41
- Instalar el paquete en los proyectos adecuados (Data y Logic).



Podremos visualizar que se ha instalado de manera correcta



Configuración del Archivo **app.config**



Una aplicación **.NET** almacena sus cadenas de conexión en un archivo de configuración.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <startup>
    <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.7.2" />
  </startup>

  <connectionStrings>
    <add name="miConexionSQL"
         connectionString="Data Source=JULIAN; Database=contactos; Integrated
Security=True;"
         providerName="System.Data.SqlClient" />
  </connectionStrings>


</configuration>
```

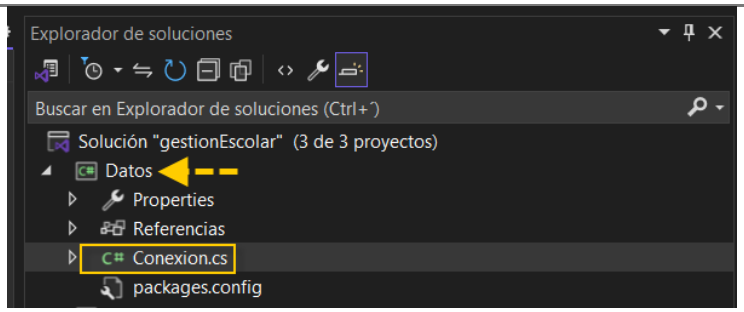
Creación de una Clase de Conexión en .NET Core

Concepto de Clase de Conexión

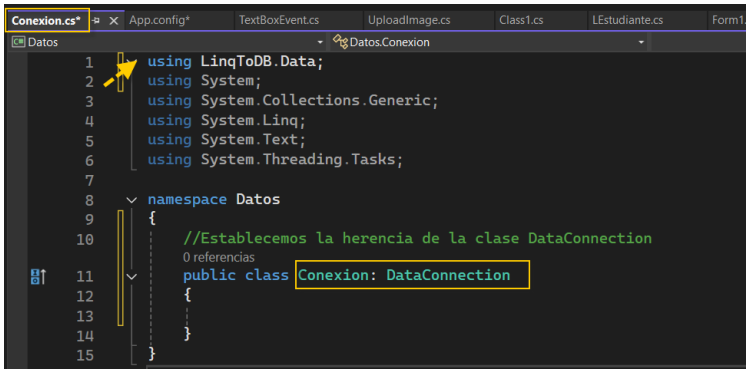
Una **clase de conexión** se encarga de gestionar la conexión entre la aplicación y la base de datos. En .NET Core.

Pasos para Crear una Clase de Conexión

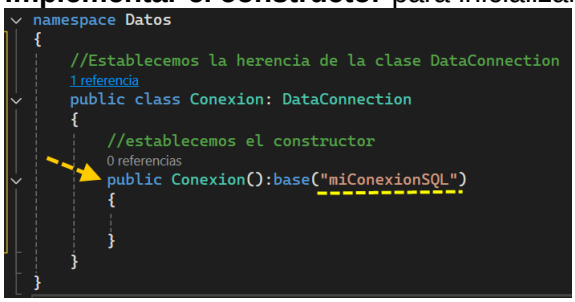
1. **Crear una nueva clase** en el proyecto.



2. **Heredar de una clase base DataConnection**.



3. **Implementar el constructor** para inicializar la conexión.

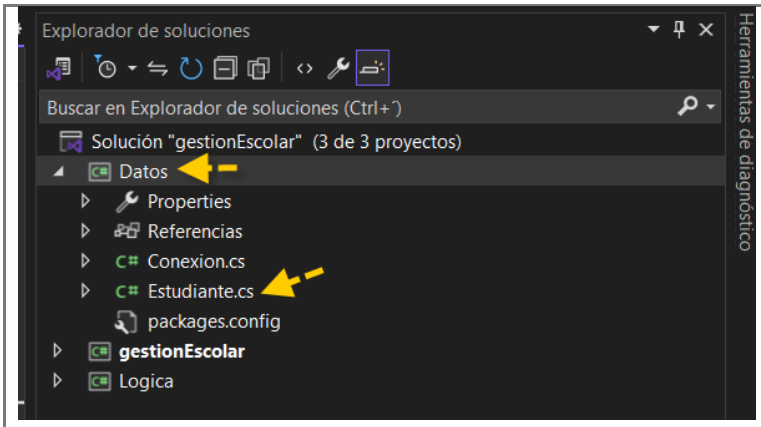


Explicación:

1. **Herencia:** Conexion extiende la clase **DataConnection**, lo que significa que hereda su funcionalidad.
2. **Constructor:** Se define un constructor para Conexion, el cual llama al constructor de la clase base (**DataConnection**) utilizando la palabra clave **base**.
3. **Parámetro "miConexionSQL":** Se pasa el nombre de la conexión "**miConexionSQL**" al constructor de **DataConnection**, lo que indica que esta clase **Conexion** está configurando una conexión a una base de datos específica.

Creación de la Clase de Modelo: Estudiante

El primer paso es definir una clase que represente la **tabla** en la base de datos.




```
using LinqToDB.Mapping;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Datos
{
    public class Estudiante
    {
        public string ClaveUsuario { get; set; }
        public string Nombre { get; set; }
        public string ApellidoPaterno { get; set; }
        public string ApellidoMaterno { get; set; }
        public string Telefono { get; set; }
        public string Direccion { get; set; }
        public string CorreoElectronico { get; set; }
    }
}
```

Explicación

En el presente código estamos definiendo una clase llamada **Estudiante** con un modificador de acceso **Public**, lo que significa que solo puede ser utilizada en todo el ámbito del proyecto. La clase contiene propiedades autoimplementadas, las cuales permiten almacenar información relacionada con un estudiante, tales como:

- **Id:** Identificador único del estudiante.
- **Nombre:** Nombre del estudiante.
- **ApellidoPaterno** y **ApellidoMaterno:** Apellidos del estudiante.
- **Direccion:** Dirección del estudiante.
- **Telefono:** Número de contacto.
- **CorreoElectronico:** Dirección de correo electrónico.
- **ClaveUsuario:** Contraseña o clave de acceso del estudiante.

Al usar propiedades autoimplementadas **({get; set;}),** C# genera automáticamente los métodos de acceso **(get)** y modificación **(set)** para cada una, facilitando la gestión de los datos sin necesidad de escribir código adicional.

Creación de la Clase de Conexion

Definición de la clase **Conexión**

```
public class Conexion : DataConnection
```

- **Conexion** es una clase que hereda de **DataConnection**, lo que indica que está configurada para manejar la conexión a una base de datos.
- **DataConnection** proviene de la biblioteca **LINQ** to **DB**, que facilita el acceso a bases de datos mediante consultas **LINQ**.

Constructor de la clase **Conexion**

```
public Conexion() : base("miConexionSQL") { }
```

- Este constructor llama al constructor de la clase base **DataConnection**, pasándole como argumento **"miConexionSQL".**
- **"miConexionSQL"** es el nombre de la cadena de conexión definida en la configuración del proyecto (por ejemplo, en **appsettings.json** o en un archivo de configuración).

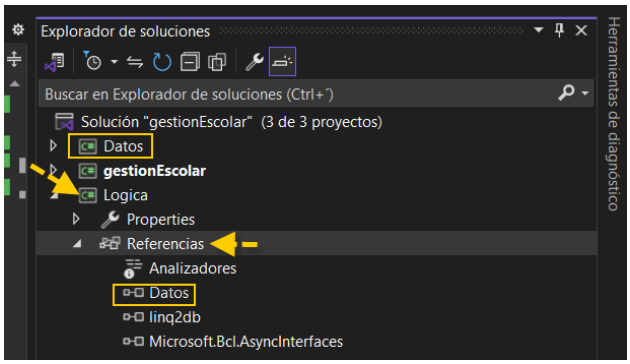
Propiedad **Estudiantes**

```
public ITable<Estudiante> Estudiantes { get; set; }
```

- **ITable<Estudiante>** representa una tabla de la base de datos, donde cada fila es un objeto del tipo Estudiante.
- **ITable<>** es una interfaz de **LINQ** to **DB** que permite realizar consultas **LINQ** sobre la tabla **Estudiante**.
- **get; set;** define una propiedad pública para acceder y modificar los datos de la tabla Estudiantes.

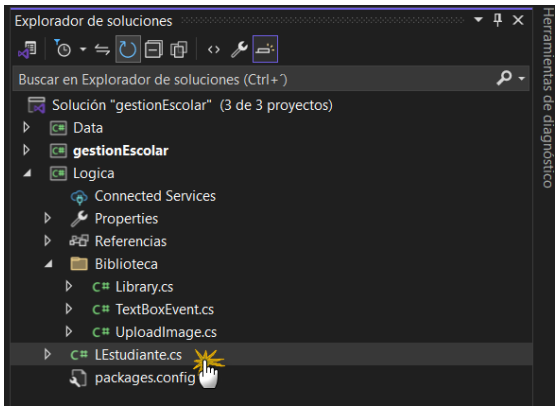
ESTABLECER LA REFERENCIA

Establecemos la referencia de la **capa de Datos**, en las referencias de **la capa Logica**

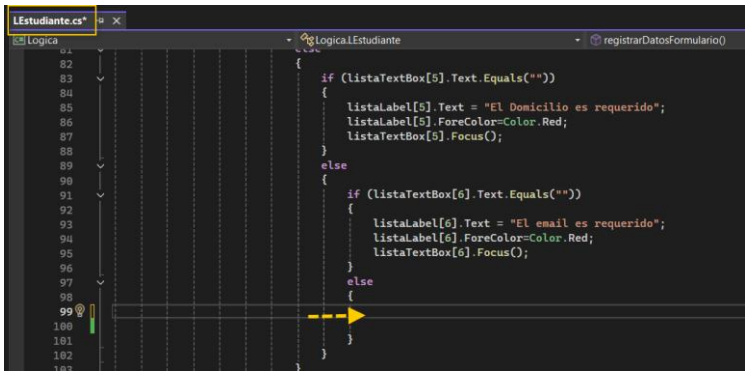


Clase LEstudiante.cs

Nos dirigiremos hacia la clase que se identifica como **"LEstudiante.cs"**



Nos posicionaremos en la última evaluación del metodo **registrarDatosFormulario ()**



Explicación pasó a paso para crear la inserción del registro

Instanciación del objeto **Conexion**

```
Conexion conexion = new Conexion();
```

- Se crea una nueva instancia de la clase **Conexion**, lo que indica que esta clase maneja la conexión a una base de datos.

Llamada al método **Insert**

```
conexion.Insert( new Estudiante { });
```

- Se invoca el método **Insert** de la instancia conexion, lo que indica que esta función está diseñada para insertar un objeto de tipo **Estudiante** en la base de datos.

Creación e inicialización del objeto **Estudiante**

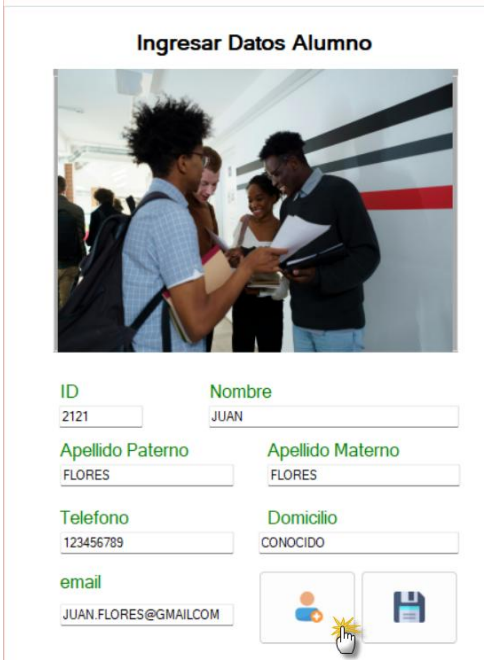
```
ClaveUsuario= listaTextBox[0].Text,
Nombre= listaTextBox[1].Text,
ApellidoPaterno= listaTextBox[2].Text,
```

```
ApellidoMaterno= listaTextBox[3].Text,
Telefono= listaTextBox[4].Text,
Direccion= listaTextBox[5].Text,
CorreoElectronico= listaTextBox[6].Text
```

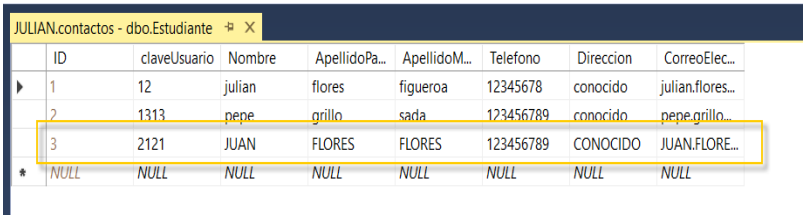
- Se crea un nuevo objeto de la clase **Estudiante** y se inicializan sus propiedades con los valores extraídos de una lista de **TextBox** llamada **listaTextBox**.
- Se asume que **listaTextBox** es un arreglo o lista de elementos de entrada en una interfaz gráfica (en Windows Forms).
- Cada campo de texto contiene información correspondiente a los atributos del estudiante, como su clave de **usuario, nombre, apellidos, teléfono, dirección y correo electrónico.**

Captura de Valores

Nos dirigiremos hacia el formulario, en donde capturamos los valores que solicita el diseño del formulario, una vez completado el formulario, pulsaremos un clic sobre el botón **"Agregar"**



Nos dirigiremos hacia la Tabla de la base de datos y podremos observar que los datos



RESULTADOS ESPERADOS

- El estudiante instala correctamente SQL Server y SQL Server Management Studio (SSMS) en su equipo.
- Configura una conexión funcional entre SQL Server y Visual Studio mediante autenticación adecuada.
- Crea y administra una base de datos desde SSMS, incluyendo la definición de tablas y tipos de datos.
- Implementa una clase de conexión en C# utilizando LINQ to DB para interactuar con la base de datos.
- Define una clase de modelo que representa la estructura de una tabla en la base de datos.
- Inserta registros en la base de datos desde un formulario Windows Forms, validando previamente los campos.
- Verifica el almacenamiento correcto de la información mediante consulta directa en SSMS.

ANÁLISIS DE RESULTADOS

- ¿Qué dificultades encontraste durante la instalación de SQL Server y cómo las resolviste?
- ¿Qué ventajas ofrece el uso de SQL Server Management Studio para la administración de bases de datos?
- ¿Cómo verificaste que la conexión entre Visual Studio y SQL Server se estableció correctamente?
- ¿Fue clara la relación entre la clase de modelo en C# y la estructura de la tabla en la base de datos?
- ¿Qué errores o advertencias encontraste al insertar datos desde el formulario y cómo los abordaste?
- ¿En qué medida LINQ to DB facilitó el acceso y manipulación de datos en comparación con otras técnicas?
- ¿Qué aspectos del proceso podrías optimizar para futuras implementaciones de bases de datos en tus aplicaciones?

CONCLUSIONES Y REFLEXIONES

La práctica permitió comprender el proceso completo de instalación, configuración y conexión de SQL Server en entornos .NET. Se fortalecieron competencias técnicas al integrar la base de datos con la interfaz de usuario mediante LINQ to DB, facilitando el acceso estructurado a la información. Además, se resaltó la importancia de una correcta configuración y validación para garantizar la integridad y funcionalidad de las aplicaciones orientadas a datos

ACTIVIDADES COMPLEMENTARIAS

1. Consulta de registros desde el formulario:
   - Implementar una funcionalidad que permita recuperar y mostrar en la interfaz los datos almacenados en la base de datos utilizando consultas LINQ.
2. Actualización y eliminación de registros:
   - Crear métodos que permitan modificar y eliminar registros existentes en la tabla, asegurando la correcta conexión y validación previa.

3. Configuración de múltiples entornos de conexión: • Agregar y probar diferentes cadenas de conexión en el archivo de configuración (app.config) para trabajar con distintas bases de datos o instancias de servidor.

EVALUACIÓN Y EVIDENCIAS DE APRENDIZAJE	
Criterios de evaluación	• Instala correctamente SQL Server y SQL Server Management Studio (SSMS) en su equipo. • Configura una conexión funcional entre Visual Studio y SQL Server utilizando autenticación adecuada. • Crea bases de datos y tablas desde SSMS aplicando correctamente los tipos de datos y claves primarias. • Define una clase de conexión en C# utilizando LINQ to DB con base en la cadena de configuración. • Declara una clase de modelo coherente con la estructura de la base de datos. • Implementa la inserción de datos desde un formulario Windows Forms hacia la base de datos. • Verifica el almacenamiento correcto de los registros mediante consultas directas en SSMS. • Aplica buenas prácticas en la organización del código, separación de capas y validación de campos.
Rúbricas o listas de cotejo para valorar desempeño	Lista de cotejo con siete criterios clave que evalúan la creación del proyecto, el diseño de la interfaz, la funcionalidad del código, el análisis reflexivo y la presentación del reporte. Cada ítem se evalúa con las opciones: “Cumple / No cumple” y espacio para observaciones.
Formatos de reporte de prácticas	**Formato libre estructurado que incluya:** 1. Datos del alumno. 2. Nombre de la práctica. 3. Objetivo. 4. Capturas de pantalla del proyecto. 5. Código implementado. 6. Respuestas al análisis. 7. Conclusiones y Reflexión.

NOMBRE DE LA PRÁCTICA No.11	**Almacenamiento de Imágenes en Bases de Datos con C# y LINQ to DB mediante Arreglos de Bytes**
COMPETENCIA DE LA PRÁCTICA	Inserta imágenes en una base de datos relacional utilizando arreglos de bytes como medio de almacenamiento, con la finalidad de vincular contenido gráfico al registro de entidades, mediante el uso de C#, Windows Forms y LINQ to DB, fortaleciendo la atención al detalle y la capacidad de estructurar soluciones técnicas integradas.

FUNDAMENTO TÉORICO

La práctica se basa en la serialización de imágenes mediante arreglos de bytes (byte[]), permitiendo su almacenamiento en bases de datos como objetos binarios (VARBINARY). Se aplican conceptos de representación digital de datos, manejo de memoria y encapsulamiento en programación orientada a objetos, integrando el uso de C#, LINQ to DB y SQL Server para lograr una solución completa y funcional.

MATERIALES, EQUIPAMIENTO Y/O REACTIVOS

Equipo de cómputo

- Computadora personal o de laboratorio con sistema operativo Windows 10 o superior
- Procesador mínimo: Intel Core i3 o equivalente
- Memoria RAM mínima: 8 GB
- Espacio disponible en disco: al menos 10 GB

Software

- Visual Studio 2022 (versión Community, Professional o Enterprise) con los siguientes componentes:
  - Desarrollo de escritorio con .NET
  - Windows Forms .NET Framework o .NET 6/7, según elección del estudiante
  - .NET SDK correspondiente a la versión seleccionada (recomendado: .NET 6 o .NET 7)

Recursos digitales

- Acceso a Internet para descarga de herramientas (en caso de ser necesario)
- Carpeta asignada o unidad compartida para guardar el proyecto generado

Cuenta de Microsoft (opcional, para sincronización de Visual Studio)

PROCEDIMIENTO O METODOLOGÍA

Inserción de Imagen en la Clase "Estudiante" en C#

En esta sección, vamos a trabajar con la capa de **"Datos"** de nuestra aplicación. Específicamente, vamos a modificar la clase **Estudiante.cs** para incluir una nueva propiedad que permitirá almacenar imágenes. La propiedad que añadiremos será:

```
public byte[] Image { get; set; }
```

En C#, una imagen no puede almacenarse directamente en una base de datos o en memoria como una imagen visible, sino que se debe convertir en una estructura de datos que pueda ser procesada fácilmente por la aplicación. Para esto, usamos un **array de bytes (byte[])**, que es una secuencia de valores numéricos que representan los datos de la imagen en formato binario.

Analogía para Comprender el Uso de byte [] para Imágenes

Imagina que una imagen es como un **rompecabezas**. A simple vista, ves la imagen completa, pero para almacenarla en la computadora, se debe descomponer en piezas más pequeñas que pueden ser organizadas de manera lógica. En términos de programación, estas piezas son **bytes**, y un conjunto de **bytes** organizados en un arreglo **(byte [])** representa la imagen en su forma digital.

¿Por qué Usar byte[] en Lugar de un Archivo de Imagen Directamente?

1. **Compatibilidad con Bases de Datos**: Los motores de bases de datos, como SQL Server, permiten almacenar datos binarios en columnas de tipo **VARBINARY**, lo que hace que **byte[]** sea una opción natural para manejar imágenes en C#.
2. **Facilidad de Transporte**: Al manejar imágenes como **byte[],** podemos enviarlas fácilmente a través de servicios web o almacenarlas en memoria sin depender de archivos físicos.
3. **Flexibilidad en la Manipulación**: Permite convertir imágenes a otros formatos o modificarlas sin necesidad de acceder a archivos externos.

Código Completo con la Nueva Propiedad

A continuación, se muestra cómo quedaría la clase Estudiante.cs después de agregar la nueva propiedad:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Datos
{
    //Cambiaremos el ámbito de la lectura
    public class Estudiante
    {
        public string ClaveUsuario { get; set; }
        public string Nombre { get; set; }
        public string ApellidoPaterno { get; set; }
        public string ApellidoMaterno { get; set; }
        public string Telefono { get; set; }
        public string Direccion { get; set; }
```

```
        public string CorreoElectronico { get; set; }
        public byte[] Image { get; set; } //Propiedad para almacenar byte[]
    }
}
```

Asignar Valor al Atributo

Lo siguiente que haremos es dirigirnos a la clase **LogicaEstudiante.cs**, donde se encuentra la lógica relacionada con los estudiantes dentro de nuestra aplicación. Dentro de esta clase, trabajaremos con el objeto **Estudiante**, que representa a un estudiante en nuestro sistema. Ahora, necesitamos asignar un valor al atributo **Image** de este objeto. Para hacerlo, utilizamos la variable **imgArray**, que contiene la imagen en un formato adecuado para ser almacenado en el objeto **Estudiante**.

Analogía para una mejor comprensión

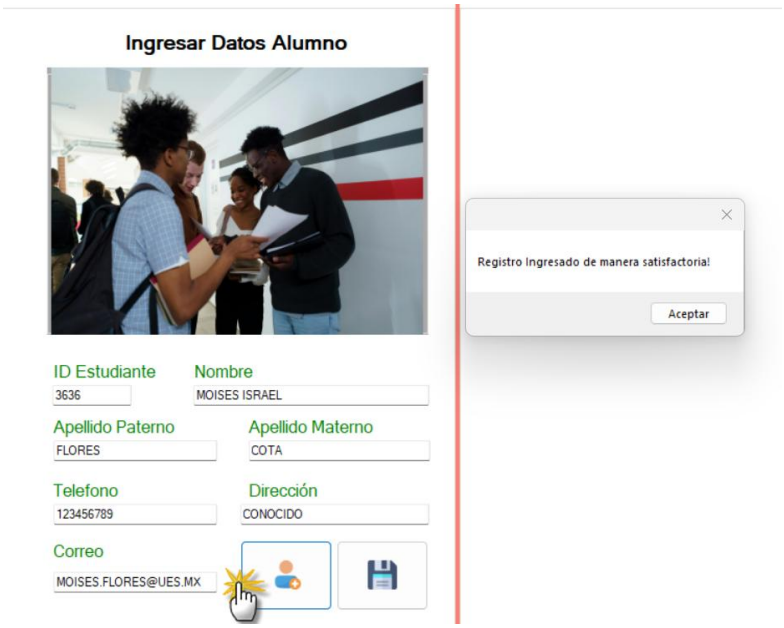
Imagina que el objeto Estudiante es una carpeta de un estudiante en formato digital. Dentro de esta carpeta, hay distintos archivos que contienen información como su nombre, matrícula, calificaciones y, en este caso, su fotografía. El atributo **Image** es como el espacio dentro de la carpeta donde se guarda la fotografía del estudiante. La variable **imgArray** es el archivo de imagen en sí, listo para ser colocado en su respectivo lugar dentro de la carpeta.

Codigo completo cuando hacemos la asignación:

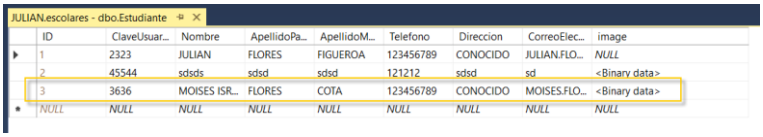
```
//Invocar el metodo publico
var imgArray = subirImagen.ImgToByte(pictureBox.Image);

//Instanciar desde la clase Conexion
Conexion conexion = new Conexion();
conexion.Insert(new Estudiante
{
    ClaveUsuario= listaTextBoxes[0].Text,
    Nombre= listaTextBoxes[1].Text,
    ApellidoPaterno= listaTextBoxes[2].Text,
    ApellidoMaterno= listaTextBoxes[3].Text,
    Telefono= listaTextBoxes[4].Text,
    Direccion= listaTextBoxes[5].Text,
    CorreoElectronico= listaTextBoxes[6].Text,
    Image= imgArray //Asignamos el arreglo de bytes[] a la propiedad

});
```

Es como si estuviéramos tomando el archivo de imagen y pegándolo dentro de la carpeta del estudiante, asegurándonos de que su foto quede guardada correctamente en el sistema.

Capturamos los siguientes datos en el Formulario



Logrando el siguiente resultado



JULIAN.escolares - dbo.Estudiante

ID	ClaveUsuar...	Nombre	ApellidoPa...	ApellidoM...	Telefono	Direccion	CorreoElec...	image
1	2323	JULIAN	FLORES	FIGUEROA	123456789	CONOCIDO	JULIAN.FLO...	NULL
2	45544	sdsds	sdsd	sdsd	121212	sdsd	sd	<Binary data>
3	3636	MOISES ISR...	FLORES	COTA	123456789	CONOCIDO	MOISES.FLO...	<Binary data>
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

RESULTADOS ESPERADOS

- El estudiante modifica correctamente la clase de modelo para incluir una propiedad de tipo byte[] destinada al almacenamiento de imágenes.
- Convierte una imagen a formato binario utilizando métodos personalizados en C#.
- Asocia la imagen convertida al objeto del modelo antes de su inserción en la base de datos.
- Inserta de forma exitosa un registro con imagen en una tabla SQL Server mediante LINQ to DB.
- Comprende la representación de datos visuales como secuencias binarias y su integración en estructuras de datos.
- Utiliza analogías funcionales para interpretar la lógica de conversión y almacenamiento de imágenes en un contexto práctico.

ANÁLISIS DE RESULTADOS

- ¿Cómo influyó la conversión de imagen a byte[] en la forma de almacenar datos visuales en la base de datos?
- ¿Qué dificultades encontraste al asociar la imagen convertida al objeto del modelo Estudiante?
- ¿Por qué es importante realizar la conversión a formato binario antes de insertar una imagen en SQL Server?
- ¿Cómo comprobaste que la imagen fue almacenada correctamente en la base de datos?

- ¿Qué beneficios ofrece el uso de LINQ to DB en la inserción de datos con contenido multimedia?
- ¿Qué mejoras implementarías en el proceso de conversión o inserción para hacerlo más robusto o reutilizable?
- ¿De qué manera esta práctica fortalece tu comprensión sobre el tratamiento de datos no textuales en sistemas de información?

CONCLUSIONES Y REFLEXIONES

La práctica permitió comprender cómo se representa y gestiona una imagen en formato binario dentro de una aplicación. La conversión a byte[] facilitó su integración en bases de datos SQL Server, demostrando la utilidad de encapsular datos visuales junto con otros atributos del modelo. Además, se reforzó el uso de LINQ to DB como herramienta eficiente para manipular datos complejos, consolidando habilidades en el manejo de estructuras no textuales dentro de entornos de desarrollo profesional.

ACTIVIDADES COMPLEMENTARIAS

1. Visualización de la imagen almacenada:
   - Implementar una funcionalidad que recupere el arreglo de bytes desde la base de datos y reconstruya la imagen para mostrarla en un PictureBox.
2. Validación de formato de imagen antes de la conversión:
   - Agregar un control de validación que permita únicamente la carga de archivos con extensiones válidas (.jpg, .png, .bmp) antes de convertirlos a byte[].
3. Inserción de múltiples imágenes por registro:
   - Ampliar la clase modelo y la interfaz gráfica para permitir el almacenamiento de más de una imagen por estudiante, utilizando listas de byte[] o creando una tabla relacionada.

EVALUACIÓN Y EVIDENCIAS DE APRENDIZAJE

Criterios de evaluación	• Modifica correctamente la clase de modelo Estudiante para incluir la propiedad de tipo byte[] destinada a la imagen. • Implementa un método funcional que convierte imágenes a arreglos de bytes utilizando ImageConverter en C#. • Asocia adecuadamente el arreglo byte[] generado a la propiedad Image del objeto Estudiante. • Inserta correctamente el objeto con imagen en la base de datos mediante LINQ to DB. • Valida que los datos, incluyendo la imagen, sean capturados correctamente desde el formulario. • Utiliza estructuras de control y conversión de tipos sin generar errores de ejecución. • Demuestra comprensión del manejo de datos binarios aplicados a contextos prácticos de almacenamiento. • Aplica buenas prácticas en la organización del código, claridad en la asignación de propiedades y uso de analogías funcionales para explicar su solución.
Rúbricas o listas de cotejo	Lista de cotejo con siete criterios clave que evalúan la creación del

para valorar desempeño	proyecto, el diseño de la interfaz, la funcionalidad del código, el análisis reflexivo y la presentación del reporte. Cada ítem se evalúa con las opciones: “Cumple / No cumple” y espacio para observaciones.
Formatos de reporte de prácticas	**Formato libre estructurado que incluya:** 1. Datos del alumno. 2. Nombre de la práctica. 3. Objetivo. 4. Capturas de pantalla del proyecto. 5. Código implementado. 6. Respuestas al análisis. 7. Conclusiones y Reflexión.

NOMBRE DE LA PRÁCTICA No.12	**Visualización de Registros e Imágenes en un DataGridView desde una Base de Datos con C# y LINQ to DB**
COMPETENCIA DE LA PRÁCTICA	Despliega datos e imágenes almacenados en una base de datos en un control DataGridView, con la finalidad de representar visualmente la información de forma estructurada y funcional, mediante el uso de C#, LINQ to DB y Windows Forms, fortaleciendo la atención al detalle y la organización en el diseño de interfaces visuales.

FUNDAMENTO TÉORICO

La práctica se fundamenta en la programación orientada a objetos y en la manipulación de datos visuales dentro de interfaces gráficas. Se aplican conceptos de vinculación entre capas de presentación y datos, conversión de formatos binarios a objetos visuales (byte[] a Image) y uso del control DataGridView para la representación tabular dinámica de información, integrando tecnologías como C#, LINQ to DB y SQL Server.

MATERIALES, EQUIPAMIENTO Y/O REACTIVOS

Equipo de cómputo

- Computadora personal o de laboratorio con sistema operativo Windows 10 o superior
- Procesador mínimo: Intel Core i3 o equivalente
- Memoria RAM mínima: 8 GB
- Espacio disponible en disco: al menos 10 GB

Software

- Visual Studio 2022 (versión Community, Professional o Enterprise) con los siguientes componentes:
  - Desarrollo de escritorio con .NET
  - Windows Forms .NET Framework o .NET 6/7, según elección del estudiante
  - .NET SDK correspondiente a la versión seleccionada (recomendado: .NET 6 o .NET 7)

Recursos digitales

- Acceso a Internet para descarga de herramientas (en caso de ser necesario)
- Carpeta asignada o unidad compartida para guardar el proyecto generado

Cuenta de Microsoft (opcional, para sincronización de Visual Studio)

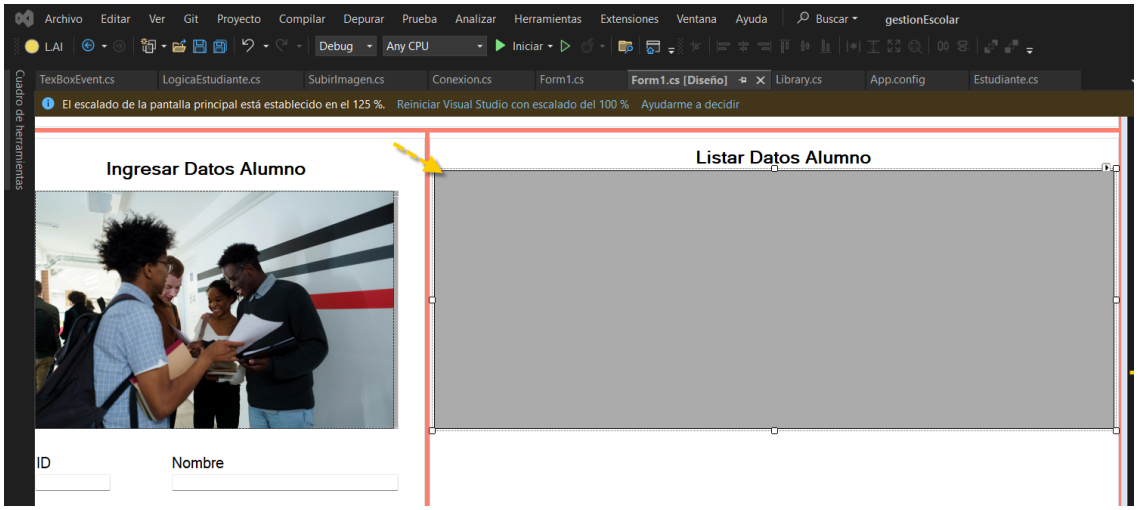
PROCEDIMIENTO O METODOLOGÍA

Agregar un DataGridView al Formulario

Analogía: Piensa en el **DataGridView** como una pizarra blanca donde escribiremos los datos que queremos mostrar. Antes de escribir, debemos asegurarnos de que está bien colocada en la pared.

1. **Abre tu formulario** en el diseñador de Windows Forms.
2. **Arrastra un control DataGridView** desde la caja de herramientas (Toolbox) hasta el formulario.
3. **Ajusta el tamaño** y posición según lo necesites.

Logramos el siguiente resultado;

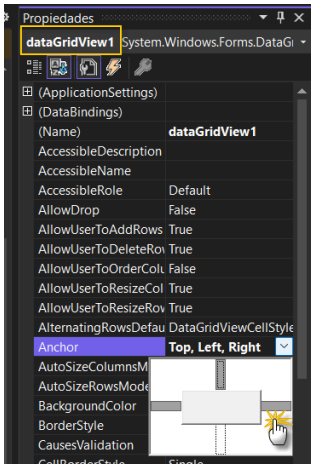


Configurar la Propiedad **Anchor**

Analogía: Imagina que el DataGridView es una mesa dentro de una habitación. Si la habitación cambia de tamaño (porque la ventana se maximiza o minimiza), queremos que la mesa también se ajuste.

Para lograr esto:

1. **Selecciona el DataGridView** en el diseñador.
2. **Busca la propiedad `Anchor`` en la ventana de propiedades.**
3. **Establece los valores adecuados:**
    - Si quieres que el **DataGridView** se expanda cuando el formulario cambie de tamaño, ajusta Anchor en **Top**, **Left**, **Right**, **Bottom**.
    - Si solo deseas que crezca horizontalmente, usa **Top**, **Left**, **Right**.



Arreglo de Objetos: Clase [Form1.cs]

Análisis del Código

```
// Arreglo de objetos
Object[] objects = { pictureBoxImagen, dataGridView1 };
```

Object [ ] objects:

- Nos posicionaremos en el arreglo de objetos llamado **objects**.
- Como Object es la clase base de todos los tipos en C#, podemos almacenar cualquier objeto en este arreglo.

{PictureBoxImagen, dataGridView1}:

- Se inicializa el arreglo con dos objetos:
  - **pictureBoxImagen:** Un PictureBox (que muestra imágenes).
  - **dataGridView1:** Un DataGridView (que muestra datos en una tabla).

Aunque estos objetos pertenecen a clases diferentes (PictureBox y DataGridView), ambos son hijos de la clase **Object**, por lo que pueden coexistir en el mismo arreglo.

Asignación del DataGridView en la Clase LogicaEstudiante.cs

Dentro del archivo de la clase **LogicaEstudiante.cs**, declaramos el atributo privado que almacenará la referencia a un control **DataGridView:**

```
DataGridView gridView;
```

Inicialización del Atributo en el Constructor

Para asegurarnos de que el **DataGridView** sea correctamente referenciado en nuestra clase, lo asignamos dentro del constructor de **LogicaEstudiante**. En este constructor, también recibimos listas de **TextBox** y **Label**, así como un arreglo de objetos que contiene otros controles de la interfaz gráfica.

```
public LogicaEstudiante(List<TextBox> listaTextBoxes, List<Label> listaLabels,
object[] objects)
{
    this.listaTextBoxes = listaTextBoxes;
    //Asignamos el valor del parámetro hacia el atributo
    this.listaLabel = listaLabels;
    pictureBox= (PictureBox)objects[0];
    gridView = (DataGridView)objects[1];
}
```

Metodo para convertir byte a Imagen

```
//Metodo para convertir byte a Imagen
 private Image ByteArrayToImage(byte[] byteArray)
 {
     using (MemoryStream ms = new MemoryStream(byteArray))
     {
         return Image.FromStream(ms);
     }
 }
```

Explicación del código del método

1. **Entrada:** El método recibe como parámetro un **arreglo de bytes (byteArray).** Este arreglo representa la información de una imagen codificada en formato binario.
2. **Creación de MemoryStream:**
   - Se crea un objeto **MemoryStream** (**ms**) utilizando el **byteArray** como origen de datos.
   - **MemoryStream** actúa como un flujo en memoria que simula un archivo con los datos de la imagen.
3. **Conversión a Image:**
   - Se usa el método **Image.FromStream** (**ms**), que toma el **MemoryStream** y lo convierte en una imagen (**Image**).
4. **Retorno de la imagen:** Se devuelve el objeto Image creado a partir del flujo.

Método para Cargar Datos en un DataGridView

El siguiente método público tiene como objetivo obtener una lista de estudiantes desde la base de datos y mostrarla en un control **DataGridView**:

```
//Metodo público, el cual permite desplegar los datos en el control DataGridView
public void CargarListaEstudiantes()
{
    //Instanciamos el objeto
    Conexion conexion = new Conexion();

    //Declaramos la variable que almacena una lista
    var listaEstudiantes = conexion.GetTable<Estudiante>()
        .Select(e=> new
        {
            e.ClaveUsuario,
            e.Nombre,
            e.ApellidoPaterno,
            e.ApellidoMaterno,
            e.Telefono,
            e.Direccion,
            e.CorreoElectronico,
            e.Image


        }) .ToList();

    //Haremos el llamado al grid
    gridView.DataSource = listaEstudiantes;

}
```

Explicación Del Código:

1. **Método CargarListaEstudiantes**
   - Se define como **public** para que pueda ser llamado desde otras clases.

- No recibe parámetros y no retorna ningún valor **(void).**

2. **Instancia de la Conexión a la Base de Datos**
   - Se crea un objeto Conexion que maneja la comunicación con la base de datos.
3. **Consulta a la Base de Datos**
   - Se obtiene la lista de estudiantes mediante **GetTable<Estudiante> ().**
   - Se usa **Select** para proyectar solo los campos necesarios, evitando cargar información innecesaria.
   - Se convierte la imagen almacenada en formato **byte []** a una imagen compatible con el control mediante **ByteArrayToImage (e.Image),** en caso de que no sea **null**.
4. **Asignación de Datos al DataGridView**
   - Se asigna la lista procesada como **DataSource** del **gridView**, permitiendo que los datos se desplieguen automáticamente en la interfaz gráfica.

Logramos el siguiente resultado

RESULTADOS ESPERADOS

- El estudiante agrega correctamente un control DataGridView al formulario y configura su propiedad Anchor.
- Declara y asigna un DataGridView desde un arreglo de objetos hacia un atributo en la clase lógica.
- Implementa un método funcional para convertir imágenes en byte[] a objetos Image.
- Desarrolla un método que carga dinámicamente los registros desde la base de datos al DataGridView.
- Visualiza correctamente los datos y las imágenes almacenadas en la base de datos en la interfaz gráfica.
- Comprende la relación entre la estructura de datos, el modelo de clases y la representación visual en el formulario.

ANÁLISIS DE RESULTADOS

- ¿Cómo aseguraste que el DataGridView se ajustara correctamente al redimensionar el formulario?
- ¿Qué ventajas ofrece utilizar un arreglo de objetos para compartir controles entre formularios y clases?
- ¿Qué dificultades encontraste al convertir el arreglo byte[] en una imagen visible en el formulario?
- ¿La información visualizada en el DataGridView coincide con la almacenada en la base de datos? ¿Cómo lo verificaste?
- ¿Cómo podrías reutilizar el método CargarListaEstudiantes() para mostrar datos de otras tablas?
- ¿Qué importancia tiene separar la lógica de presentación de la lógica de datos en este tipo de aplicaciones?

CONCLUSIONES Y REFLEXIONES

La práctica permitió integrar de manera funcional los controles visuales con los datos almacenados, facilitando la presentación estructurada de información mediante el DataGridView. Se reforzó el uso de arreglos de objetos para gestionar componentes y la importancia de convertir datos binarios a formatos visuales. Asimismo, se valoró la utilidad de mantener una arquitectura organizada para lograr una interfaz dinámica y coherente con el modelo de datos.

ACTIVIDADES COMPLEMENTARIAS

1. Agregar botón de recarga dinámica:
   - Implementar un botón en el formulario que actualice los datos del DataGridView cada vez que se inserta un nuevo registro en la base de datos.
2. Visualización detallada de un registro:
   - Programar un evento que, al hacer doble clic sobre una fila del DataGridView, muestre todos los datos e imagen del estudiante seleccionado en controles individuales del formulario.
3. Filtrado de registros por nombre o apellido:
   - Agregar un campo de búsqueda con funcionalidad para filtrar los datos mostrados en el DataGridView según el criterio ingresado (por ejemplo, nombre o apellido paterno).

EVALUACIÓN Y EVIDENCIAS DE APRENDIZAJE

Criterios de evaluación	• Agrega correctamente el control DataGridView al formulario y configura adecuadamente su propiedad Anchor. • Declara e inicializa correctamente los controles PictureBox y DataGridView a través de un arreglo de objetos. • Implementa correctamente el método de conversión de byte[] a Image utilizando MemoryStream. • Desarrolla un método funcional que carga registros desde la base de datos al DataGridView mediante LINQ to DB. • Visualiza adecuadamente los datos y las imágenes en el DataGridView. • Mantiene la separación de responsabilidades entre la capa lógica y la interfaz gráfica.

	• Utiliza estructuras de control y conversión de tipos de forma segura y sin errores de ejecución. • Presenta el código con buena organización, claridad en la estructura y nomenclatura coherente.
Rúbricas o listas de cotejo para valorar desempeño	Lista de cotejo con siete criterios clave que evalúan la creación del proyecto, el diseño de la interfaz, la funcionalidad del código, el análisis reflexivo y la presentación del reporte. Cada ítem se evalúa con las opciones: “Cumple / No cumple” y espacio para observaciones.
Formatos de reporte de prácticas	**Formato libre estructurado que incluya:** 1. Datos del alumno. 2. Nombre de la práctica. 3. Objetivo. 4. Capturas de pantalla del proyecto. 5. Código implementado. 6. Respuestas al análisis. 7. Conclusiones y Reflexión.

NOMBRE DE LA PRÁCTICA No.13	**Operaciones CRUD: Insertar y Editar Estudiantes con Imágenes**
COMPETENCIA DE LA PRÁCTICA	Implementa un módulo de registro y edición de estudiantes con manejo de imágenes, para actualizar y mantener la información visual y textual en una base de datos, utilizando eventos del DataGridView y controles de formulario en C#, dentro del entorno de desarrollo Visual Studio con Windows Forms, demostrando atención al detalle y pensamiento lógico en la solución de problemas.

FUNDAMENTO TÉORICO

Esta práctica se basa en los principios de programación orientada a objetos, manejo de eventos en interfaces gráficas (Windows Forms) y persistencia de datos mediante acceso a bases de datos. Se aplican técnicas para la conversión de imágenes a arreglos de bytes (byte[]), y su recuperación, visualización y edición, garantizando integridad y consistencia de la información almacenada. Además, se promueve el diseño modular y reutilizable de código mediante la separación por capas.

MATERIALES, EQUIPAMIENTO Y/O REACTIVOS

Equipo de cómputo

- Computadora personal o de laboratorio con sistema operativo Windows 10 o superior
- Procesador mínimo: Intel Core i3 o equivalente
- Memoria RAM mínima: 8 GB
- Espacio disponible en disco: al menos 10 GB

Software

- Visual Studio 2022 (versión Community, Professional o Enterprise) con los siguientes componentes:
  - Desarrollo de escritorio con .NET
  - Windows Forms .NET Framework o .NET 6/7, según elección del estudiante
  - .NET SDK correspondiente a la versión seleccionada (recomendado: .NET 6 o .NET 7)

Recursos digitales

- Acceso a Internet para descarga de herramientas (en caso de ser necesario)
- Carpeta asignada o unidad compartida para guardar el proyecto generado

Cuenta de Microsoft (opcional, para sincronización de Visual Studio)

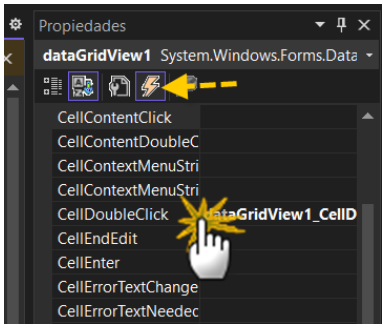
PROCEDIMIENTO O METODOLOGÍA

Desarrollo del módulo: Registro y edición de estudiantes

Vamos a continuar con el desarrollo de nuestro sistema. Hasta este punto ya tenemos funcionalidad para registrar estudiantes y visualizarlos en un **DataGridView**. Ahora, avanzaremos con una nueva funcionalidad: **obtener** y **mostrar** la información detallada del estudiante seleccionado desde el **DataGridView**, con el propósito de editarla posteriormente.

Detectar la selección del estudiante

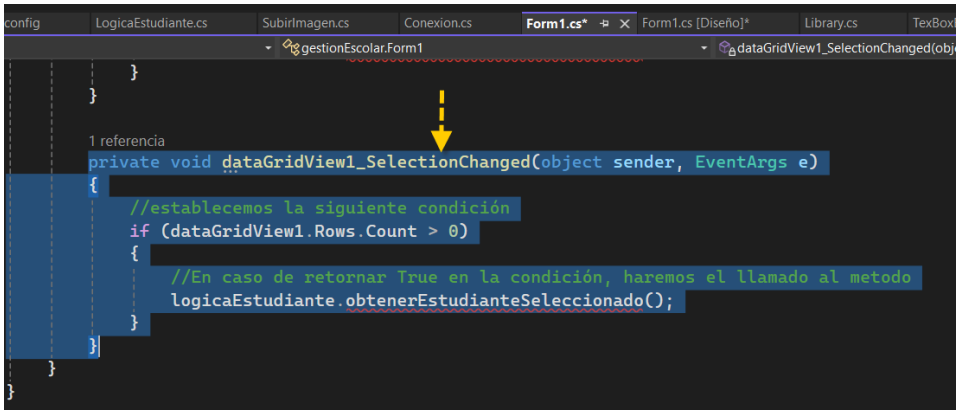
1. En el diseñador de Windows Forms (**Form1.cs** [**Design**]), seleccionamos el **DataGridView**.
2. En la ventana de propiedades, ubicamos la sección de eventos (**ícono de rayo**).
3. Asignamos el evento **CellDoubleClick** o **SelectionChanged** según la lógica que deseemos (clic o navegación con teclas).



4. Este método se ejecuta cuando el usuario hace doble clic en una celda del DataGridView. Primero, verifica si hay al menos una fila en el DataGridView (**dataGridView1.Rows.Count > 0**). Si la condición se cumple, llama al método **obtenerEstudianteSeleccionado ()** del objeto **logicaEstudiante**, el cual obtiene o procesa la información del estudiante seleccionado en la tabla.

```
private void dataGridView1_CellDoubleClick(object sender, DataGridViewCellEventArgs e)
{
    //establecemos la siguiente condición
    if (dataGridView1.Rows.Count>0)
    {
        //En caso de retornar True en la condición, haremos el llamado al metodo
        logicaEstudiante.obtenerEstudianteSeleccionado();
    }
}
```

5. Copiamos y pegamos el mismo código en el evento **SelectionChanged**

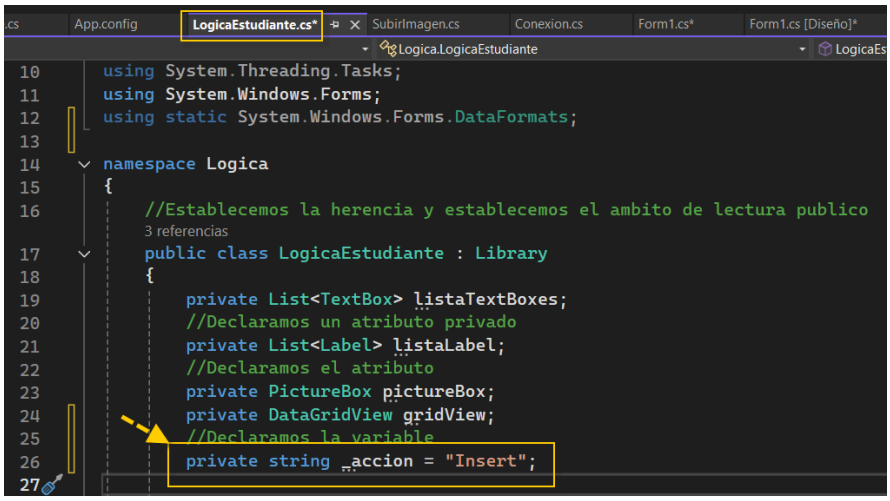


NOTA: se genera una alerta, ya que todavía no hemos creado el metodo

Crear el método ObtenerEstudianteSeleccionado ( ) en la clase Estudiante

Dentro de la clase Estudiante, creamos un **método público sin retorno**, que leerá los datos de la fila seleccionada y los asignará a los controles visuales del formulario:

- Lo primero que deberemos hacer, es realizar la de una variable privada llamada **_accion**, de tipo cadena de texto (**string**), que se inicializa con el valor **"Insert".**



- **Creación del metodo**

```
//Creamos una variable global
public int _idEstudiante = 0;

//Crearemos el metodo público, sin retorno
public void obtenerEstudianteSeleccionado()
{
    //Haremos uso de la variable global y le asignamos el siguiente texto
    _accion = "update";
    _idEstudiante = Convert.ToInt32(gridView.CurrentRow.Cells[0].Value);

    //También vamos a obtener el resto de las posiciones y las asignaremos
    //a las cajas de texto
    listaTextBoxes[0].Text = Convert.ToString(gridView.CurrentRow.Cells[0].Value);
    listaTextBoxes[1].Text = Convert.ToString(gridView.CurrentRow.Cells[1].Value);
    listaTextBoxes[2].Text = Convert.ToString(gridView.CurrentRow.Cells[2].Value);
    listaTextBoxes[3].Text = Convert.ToString(gridView.CurrentRow.Cells[3].Value);
    listaTextBoxes[4].Text = Convert.ToString(gridView.CurrentRow.Cells[4].Value);
    listaTextBoxes[5].Text = Convert.ToString(gridView.CurrentRow.Cells[5].Value);
    listaTextBoxes[6].Text = Convert.ToString(gridView.CurrentRow.Cells[6].Value);

    //Establecemos un Try-Catch
    try
    {
        //Declaramos la variable que recibe el array
        byte[] imagenBytes = (byte[]) gridView.CurrentRow.Cells[7].Value;
        pictureBox.Image = ArrayToImage(imagenBytes);
    }
    catch (Exception ex) {
        MessageBox.Show ("No econtre la imagen!");
    }

}
```

Explicación del Código

Este método llamado **obtenerEstudianteSeleccionado** se encarga de **recuperar la información del estudiante seleccionado** en un gridView (una tabla en pantalla) y mostrarla en varios controles

de la interfaz gráfica (**textboxes** y **picturebox**).

Paso a paso:

1. **Variable global:**
   Se tiene una variable global **_idEstudiante** para guardar el **ID** del estudiante seleccionado.
2. **Acción de actualización:**
   Se indica que la acción actual es **"update"** (actualizar).
3. **Asignación de valores:**
   Se extraen los valores de las celdas de la fila actual seleccionada (**gridView.CurrentRow.Cells**) y se asignan a una lista de cajas de texto (**listaTextBoxes**), del índice **0** al **6**.
4. **Carga de imagen:**
   En un bloque **try-catch**, se intenta recuperar una imagen (almacenada como arreglo de bytes en la celda 7) y mostrarla en un pictureBox.
   Si no se encuentra o ocurre un error, se muestra un mensaje.

Imagen como Array

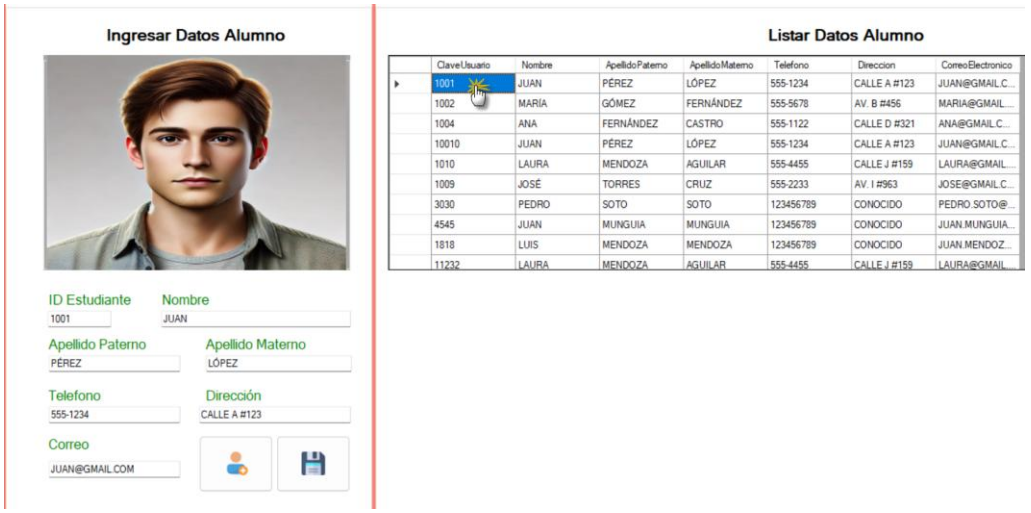
A continuación, nos dirigiremos al método que hemos identificado como **ListarRegistrosTabla ().** Este método es responsable de desplegar los valores en el control DataGridView. En esta etapa, lo que haremos será recuperar la imagen almacenada como un arreglo de bytes (array), para poder visualizarla correctamente en el DataGridView. La modificación del método quedará estructurada de la siguiente manera:

- El código **gridView.Columns [7].Visible = false;** se utiliza para ocultar una columna específica dentro de un control GridView. En este caso, accede a la octava columna del GridView (recordando que la indexación comienza en cero) y establece su propiedad Visible en false. Esto significa que dicha columna no será mostrada al usuario en la interfaz gráfica, aunque los datos que contiene seguirán estando presentes internamente.

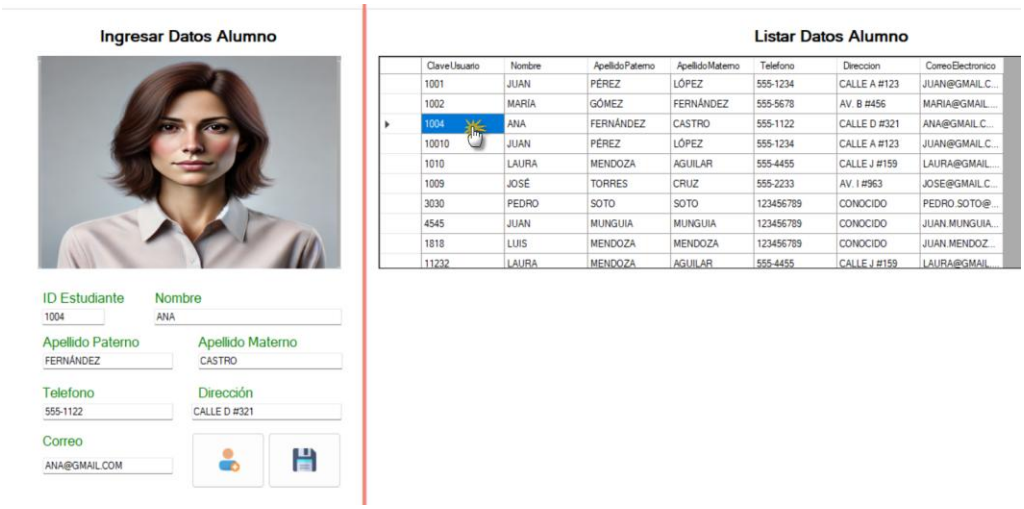
```
public void ListarRegistrosTabla()

        {
            //Instanciamos
            Conexion conexion = new Conexion();

            //Declaramos la variable que recibe el listado
            var listadoUsuarios = conexion.GetTable<Estudiante>()
               .Select(e => new
               {
                   e.ClaveUsuario,
                   e.Nombre,
                   e.ApellidoPaterno,
                   e.ApellidoMaterno,
                   e.Telefono,
                   e.Direccion,
                   e.CorreoElectronico,
                   //imagen= ArrayToImage(e.Image)
                   imagen = e.Image //Mostramos la imagen como array de datos
               }) .ToList();

            gridView.DataSource = listadoUsuarios;
            //Ocultamos la visibilidad de la columna que carga la imagen
            gridView.Columns[7].Visible = false;
        }
```

Logrando el siguiente resultado

- Pulsando el primer registro



- Pulsando el tercer registro



Edición de Registros

La edición de registros en un sistema informático es un proceso crítico que permite mantener actualizada la información de las entidades almacenadas, como en este caso, los estudiantes.

- Establecer la llave Primaria: **Estudiante.cs**

```
public class Estudiante
{
    [PrimaryKey]
    public string ClaveUsuario { get; set; }
```

  - **[PrimaryKey]:** Esta anotación marca ClaveUsuario como la clave primaria, es decir, el identificador único de cada estudiante en la base de datos.

Lógica de Negocio: LogicaEstudiante.cs

- Desarrollo del Método **guardar()**

```
public void guardar()
{
    // Convertir imagen a arreglo de bytes
    var imgConvertidaArray = subirImagen.ImgToByte(pictureBox.Image);

    // Crear instancia de conexión a la base de datos
    Conexion conexion = new Conexion();

    // Verificar tipo de operación
    switch (_accion)
    {
        case "insert":
            // Insertar un nuevo estudiante
            break;

        case "update":
            // Actualizar un estudiante existente
            break;
    }
}
```

Inserción de un Nuevo Estudiante

- Cuando "**_accion**" tiene el valor "**insert**", se ejecuta la lógica de alta:

```
conexion.Insert(new Estudiante
{
    ClaveUsuario = listaTextBoxes[0].Text,
    Nombre = listaTextBoxes[1].Text,
    ApellidoPaterno = listaTextBoxes[2].Text,
    ApellidoMaterno = listaTextBoxes[3].Text,
    Telefono = listaTextBoxes[4].Text,
    Direccion = listaTextBoxes[5].Text,
    CorreoElectronico = listaTextBoxes[6].Text,
    Image = imgConvertidaArray
});

MessageBox.Show("Registro Ingresado de manera correcta", _accion);
```

  - Se crea un nuevo objeto Estudiante tomando los valores de la interfaz gráfica (textboxes).
  - Se llama al método Insert de la clase Conexion para agregar el registro.
  - Se notifica al usuario con un mensaje emergente.

Actualización de un Estudiante Existente

- Cuando "**_accion**" es "**update**", el proceso es actualizar:

```
var estudianteExistente = conexion.GetTable<Estudiante>()
    .FirstOrDefault(e => e.ClaveUsuario == _idEstudiante.ToString());

if (estudianteExistente != null)
{
    Estudiante estudianteEditado = new Estudiante
    {
        ClaveUsuario = listaTextBoxes[0].Text,
```

```
                Nombre = listaTextBoxes[1].Text,
                ApellidoPaterno = listaTextBoxes[2].Text,
                ApellidoMaterno = listaTextBoxes[3].Text,
                Telefono = listaTextBoxes[4].Text,
                Direccion = listaTextBoxes[5].Text,
                CorreoElectronico = listaTextBoxes[6].Text,
                Image = imgConvertidaArray
            };

            conexion.Update(estudianteEditado);

            MessageBox.Show("Actualización exitosa del registro");
        }
```

- Se busca el estudiante mediante la clave primaria (ClaveUsuario).
- Se verifica que exista (estudianteExistente != null).
- Se crea una nueva instancia de Estudiante con los datos actualizados.
- Se utiliza el método Update para guardar los cambios en la base de datos.

Creación de un Método para Limpiar Campos en un Formulario

Cuando se desarrolla una aplicación con formularios, especialmente en lenguajes como C# usando Windows Forms, es común necesitar una funcionalidad que "limpie" los campos del formulario después de realizar acciones como registrar, editar, o cancelar una operación.
Este proceso, conocido como limpieza de campos, garantiza que el formulario quede listo para recibir nueva información, mejorando la experiencia del usuario y evitando errores de datos residuales.
A continuación, explicaremos paso a paso cómo crear un método reutilizable para limpiar tanto cajas de texto (TextBox) como una imagen cargada en un PictureBox, utilizando buenas prácticas de programación.

Método LimpiarCampos

```
// Creación del método que permite Limpiar
public void LimpiarCampos(List<TextBox> listaTextBoxes, PictureBox pictureBox)
{
    // Inicializamos la variable
    _accion = "insert";

    // Creamos un bucle
    foreach (var textbox in listaTextBoxes)
    {
        textbox.Clear();
    }

    if (pictureBox != null)
    {
        pictureBox.Image = null;
    }

        //Limpiamos el control Data Grid View
        gridView.DataSource = null;
        gridView.Rows.Clear();
        gridView.Refresh();
}
```

- **public:** El método es público, es decir, puede ser invocado desde otras clases o formularios donde se tenga una referencia al objeto que contiene este método.
- **void:** No retorna ningún valor. Su objetivo es realizar una acción: limpiar.
- **List<TextBox> listaTextBoxes:** Recibe como parámetro una lista de todos los controles TextBox que deben ser limpiados.
- **PictureBox pictureBox:** Recibe un objeto PictureBox, que se limpiará (se eliminará la imagen) si existe.

Inicialización de Acción

_accion = "insert";

- Aquí se está asignando el valor **"insert"** a una variable llamada _**accion.**
- Aunque en el fragmento no vemos la declaración de _**accion**, nos indica que es un campo de la clase que indica el estado del formulario (por ejemplo, si está en modo insertar o editar).

Importancia:

- Esta inicialización es útil para controlar la lógica del formulario: al limpiar los campos, el sistema sabe que está listo para insertar un nuevo registro.
- **Analogía práctica:** Piensa en _accion como el "modo" de un electrodoméstico. Si seleccionas "insert", la máquina (el formulario) sabe que debe comportarse de cierta forma (aceptar nuevos datos).

Limpieza de Cajas de Texto

```
foreach (var textbox in listaTextBoxes)
{
    textbox.Clear();
}
```

- Se usa un bucle foreach para recorrer cada TextBox en la lista.
- El método **.Clear()** elimina cualquier texto dentro del TextBox, dejándolo vacío.

Limpieza de la Imagen

```
if (pictureBox != null)
{
    pictureBox.Image = null;
}
```

- **Validación:** Se comprueba si pictureBox no es null para evitar errores de ejecución.
- **Acción:** Si existe, su propiedad .Image se establece en null, eliminando la imagen mostrada.

Analogía: Imagina que tienes un marco con una fotografía. **pictureBox.Image = null** sería como quitar la foto del marco, dejándolo vacío.

Cómo Invocar el Método

Una vez que el método LimpiarCampos está creado, se puede utilizar en cualquier parte del código, especialmente después de realizar operaciones como registrar datos.
Por ejemplo, al final de un método registrarDatosFormulario(), lo invocamos así:

LimpiarCampos(listaTextBoxes, pictureBox);

- **listaTextBoxes:** Se refiere a la lista que contiene todos los TextBox que queremos limpiar.
- **pictureBox:** Es el control donde el usuario pudo haber cargado una imagen.

Resultado esperado: El formulario queda completamente vacío y listo para una nueva entrada de datos.

Creación de un Método para Limpiar Controles desde un Formulario Principal

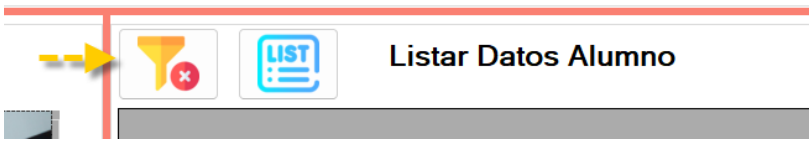
Limpiar los controles de un formulario es una práctica común en aplicaciones de escritorio (Windows Forms), sobre todo cuando queremos reiniciar el formulario tras una operación como **guardar**, **actualizar** o **cancelar**.

¿Por qué limpiar un formulario?
Al interactuar con una interfaz gráfica, los usuarios ingresan datos en cuadros de texto, cargan imágenes, o visualizan información en tablas. Después de ciertas acciones, como guardar los datos, es necesario limpiar la interfaz para permitir un nuevo ingreso sin interferencias visuales.

Caso de uso real:
Imagina una aplicación escolar donde se registran estudiantes. Una vez que se guarda un estudiante, se desea dejar el formulario limpio para registrar uno nuevo. Si los datos anteriores permanecen, se puede generar confusión o errores.

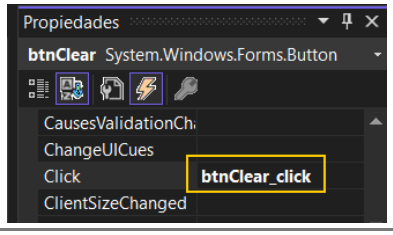
Preparando el botón para limpiar
Establecer el botón en el formulario

1. Abre el diseñador del formulario (**Form1.cs** [Diseñador]).
2. Arrastra un botón desde la Caja de herramientas hacia el formulario.
3. Cambia su Name a **btnClear** y su propiedad Text dejarla vacía



Asociar el evento Click del botón

- El evento Click es el más común y se dispara cuando el usuario hace clic sobre el botón.
- Para asociar el método de limpieza, nos dirigiremos hacia los eventos y seleccionaremos el evento **“Click”**



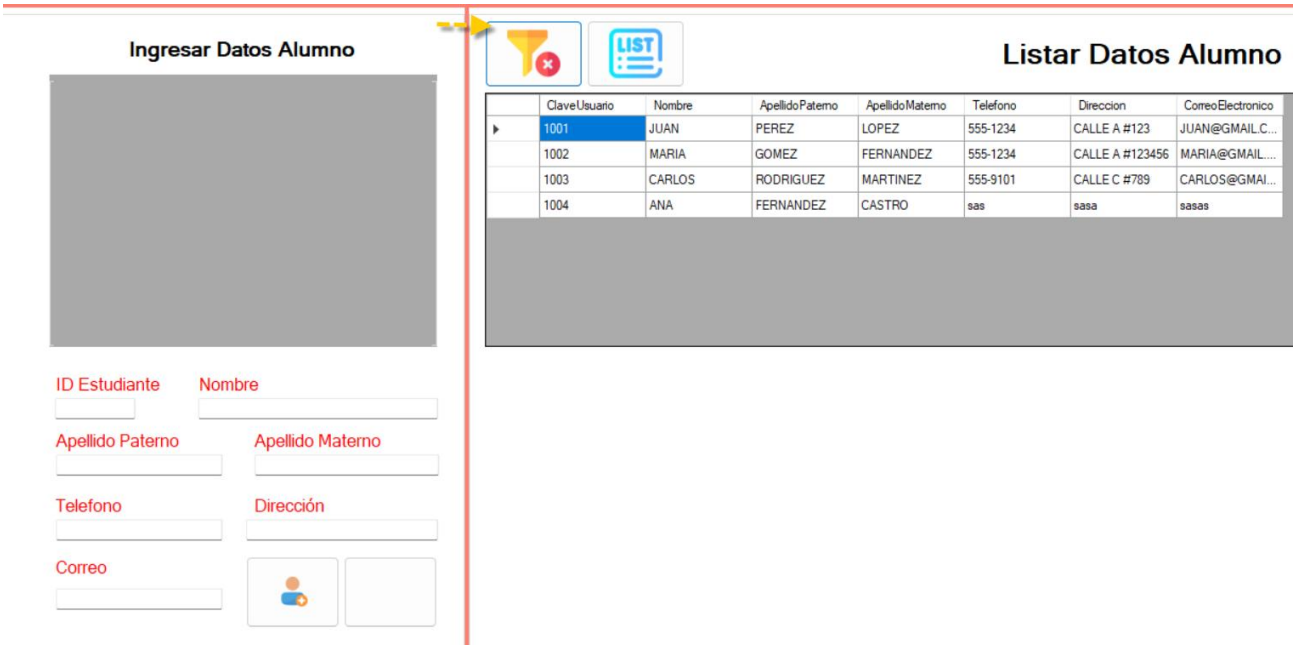
- Una vez que estamos colocados en el evento, limpiamos los controles con el siguiente código

```
private void btnLimpiarGrid_Click(object sender, EventArgs e)
{
    //Cambiamos
    logicaEstudiante._accion = "insert";

    //limpiar
    for (int i=0; i<=6; i++)
    {
        logicaEstudiante.listaTextBoxes[i].Text = "";
    }

    if (logicaEstudiante.pictureBox.Image!=null)
    {
        logicaEstudiante.pictureBox.Image = null;
    }

}
```

Logrando un resultado como el siguiente, en donde podremos apreciar que al momento de pulsar un click sobre el botón, nos limpiar el Formulario

¿Qué hace cada parte?

- **Text = "":** Borra el texto en los cuadros de entrada.
- **Image = null:** Elimina cualquier imagen cargada en el control.



RESULTADOS ESPERADOS

- El estudiante detecta correctamente la selección de un registro en el DataGridView mediante eventos como CellDoubleClick o SelectionChanged.
- Implementa un método que recupera y muestra la información del estudiante seleccionado en

controles de texto e imagen.

- Edita y actualiza los datos de un estudiante existente en la base de datos, conservando su imagen o reemplazándola según sea necesario.
- Integra el control de acciones (insert o update) mediante una variable de estado para gestionar operaciones de forma eficiente.
- Utiliza un método de limpieza que reinicia el formulario para permitir nuevos registros sin interferencias de datos anteriores.
- Garantiza la correcta conversión entre imágenes y arreglos de bytes para su almacenamiento y recuperación.
- Aplica buenas prácticas de diseño como encapsulamiento, separación por capas y uso de estructuras reutilizables.

ANÁLISIS DE RESULTADOS

- ¿Qué diferencia existe entre los eventos CellDoubleClick y SelectionChanged al seleccionar un registro en el DataGridView?
- ¿Cuál es la función de la variable _accion y por qué es importante para controlar el flujo entre registrar y actualizar estudiantes?
- ¿Qué ventajas ofrece el uso de listas de TextBox para asignar valores a los campos del formulario?
- ¿Qué ocurre si se intenta cargar una imagen no válida o ausente desde la base de datos? ¿Cómo maneja esto el bloque try-catch?
- ¿Por qué es necesario ocultar la columna que contiene la imagen (byte[]) en el DataGridView? ¿Qué implicaciones tendría dejarla visible?
- ¿Cuál es el objetivo de convertir imágenes a arreglos de bytes para almacenarlas en la base de datos? ¿Existe una alternativa técnica?
- ¿Qué aspectos debes considerar para asegurar que los datos editados sean consistentes antes de realizar la operación Update?
- ¿Qué mejoras podrías implementar al método LimpiarCampos para que sea más flexible o reutilizable?
- ¿Cómo contribuye esta práctica al desarrollo de habilidades como la atención al detalle y la resolución de problemas?

CONCLUSIONES Y REFLEXIONES

Esta práctica permitió comprender y aplicar técnicas clave para la edición de registros en sistemas de escritorio, integrando eventos del DataGridView, manejo de imágenes y control de estados mediante variables. Se reforzó la importancia de mantener una estructura modular y reutilizable del código, así como la necesidad de validar y limpiar adecuadamente los datos. Además, se promovió la reflexión sobre el diseño lógico de interfaces y la interacción fluida con bases de datos, fortaleciendo competencias técnicas y habilidades de resolución de problemas en entornos reales de desarrollo.

ACTIVIDADES COMPLEMENTARIAS

1. Validación de datos antes de guardar:
   - Implementa un método que verifique que todos los campos obligatorios del formulario estén completos antes de permitir la inserción o actualización del estudiante. Incluye mensajes de advertencia específicos por campo.

2. Carga inicial de imagen por defecto:
   - Modifica el formulario para que, al iniciar o limpiar los campos, el PictureBox muestre una imagen predeterminada si el usuario no ha cargado una.
3. Implementación de botón de “Cancelar edición”:
   - Agrega un botón que permita al usuario cancelar la edición de un registro en curso y restablezca el formulario al modo de inserción (_accion = "insert"), sin borrar los datos cargados en el DataGridView.

EVALUACIÓN Y EVIDENCIAS DE APRENDIZAJE	
Criterios de evaluación	• Detecta correctamente la selección de un estudiante desde el DataGridView utilizando eventos adecuados. • Implementa el método obtenerEstudianteSeleccionado() para cargar los datos del registro en los controles del formulario. • Utiliza adecuadamente la variable _accion para controlar el flujo entre inserción y edición. • Aplica correctamente la conversión de imágenes a byte[] y viceversa para su almacenamiento y visualización. • Desarrolla el método guardar() considerando los casos de inserción y actualización, validando la existencia del registro. • Integra el método LimpiarCampos() para restablecer el formulario tras completar una operación. • Oculta correctamente la columna de imágenes en el DataGridView para mejorar la interfaz de usuario. • Estructura el código de manera organizada, reutilizable y conforme a principios de programación orientada a objetos. • Responde con claridad y fundamento a las preguntas de análisis de resultados. • Demuestra responsabilidad, atención al detalle y capacidad de solución de problemas durante el desarrollo de la práctica.
Rúbricas o listas de cotejo para valorar desempeño	Lista de cotejo con siete criterios clave que evalúan la creación del proyecto, el diseño de la interfaz, la funcionalidad del código, el análisis reflexivo y la presentación del reporte. Cada ítem se evalúa con las opciones: “Cumple / No cumple” y espacio para observaciones.
Formatos de reporte de prácticas	**Formato libre estructurado que incluya:** 1. Datos del alumno. 2. Nombre de la práctica. 3. Objetivo. 4. Capturas de pantalla del proyecto. 5. Código implementado. 6. Respuestas al análisis. 7. Conclusiones y Reflexión.

NOMBRE DE LA PRÁCTICA No.14	**Implementación del Método Eliminar con Acceso a Base de Datos**
COMPETENCIA DE LA PRÁCTICA	Implementa un método para eliminar registros de estudiantes de la base de datos, con el fin de mantener actualizada y depurada la información del sistema, previa validación de existencia y confirmación del usuario, utilizando el lenguaje C# en un entorno Windows Forms con acceso a datos mediante LINQ, demostrando responsabilidad y toma de decisiones conscientes en la gestión de datos.

FUNDAMENTO TÉORICO

Esta práctica se basa en el uso de programación orientada a objetos, manipulación de estructuras de datos mediante LINQ y gestión de operaciones en bases de datos relacionales. Se aplican principios de interacción con el usuario a través de controles visuales y confirmaciones seguras, así como técnicas de conversión de imágenes a formato binario (byte[]) para su manejo y eliminación como parte integral del registro.

MATERIALES, EQUIPAMIENTO Y/O REACTIVOS

Equipo de cómputo

- Computadora personal o de laboratorio con sistema operativo Windows 10 o superior
- Procesador mínimo: Intel Core i3 o equivalente
- Memoria RAM mínima: 8 GB
- Espacio disponible en disco: al menos 10 GB

Software

- Visual Studio 2022 (versión Community, Professional o Enterprise) con los siguientes componentes:
  - Desarrollo de escritorio con .NET
  - Windows Forms .NET Framework o .NET 6/7, según elección del estudiante
  - .NET SDK correspondiente a la versión seleccionada (recomendado: .NET 6 o .NET 7)

Recursos digitales

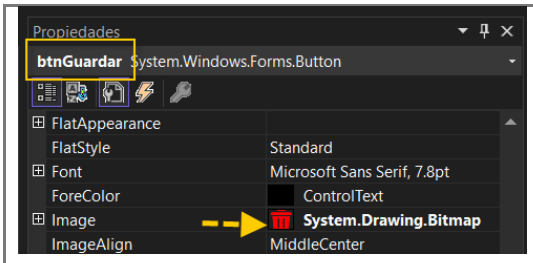
- Acceso a Internet para descarga de herramientas (en caso de ser necesario)
- Carpeta asignada o unidad compartida para guardar el proyecto generado

Cuenta de Microsoft (opcional, para sincronización de Visual Studio)

PROCEDIMIENTO O METODOLOGÍA

Creación del metodo que permite eliminar registros de la Tabla de la Base de Datos

Lo primero que haremos, será modificar el icono del botón, desde la propiedad **“Image”**



¿Qué hace este método?

- El método **eliminarRegistro()** permite eliminar un registro de estudiante desde una base de datos cuando el usuario lo confirma explícitamente mediante un cuadro de diálogo (MessageBox). Utiliza una capa lógica llamada LogicaEstudiante y una clase llamada Conexion, que actúa como intermediaria para acceder a la base de datos.

Estructura general del método

```
public void eliminarRegistro()
{
    // 1. Convertir imagen a arreglo de bytes
    // 2. Instanciar conexión a la base de datos
    // 3. Buscar si el registro existe
    // 4. Preguntar al usuario si desea eliminarlo
    // 5. Si acepta, crear objeto con los datos del estudiante
    // 6. Eliminarlo de la base de datos
    // 7. Notificar al usuario y actualizar la tabla
}
```

Conversión de imagen a arreglo de bytes

```
var imgConvertidaArray = subirImagen.ImgToByte(pictureBox.Image);
```

¿Qué significa esto?

Una imagen en un **PictureBox** es un objeto gráfico. Sin embargo, las bases de datos no almacenan imágenes directamente; en su lugar, las almacenan como datos binarios **(byte[]).** Aquí, se utiliza un método llamado **ImgToByte** que convierte la imagen en **bytes**.

Ejemplo práctico: Si tienes una foto de estudiante, esta línea permite guardar esa foto como una secuencia de bytes en la base de datos.

Conexión a la base de datos

```
Conexion conexion = new Conexion();
```

Aquí se instancia un objeto de la clase Conexion, la cual encapsula la lógica para interactuar con la base de datos para poder realizar algunas de las siguientes operaciones (consultas, inserciones, eliminaciones, etc.).

Analogía: Puedes imaginar Conexion como un mensajero que lleva instrucciones desde tu aplicación hacia la base de datos.

Verificar si el registro existe

```
var registroExistente = conexion.GetTable<Estudiante>()
    .FirstOrDefault(e => e.ClaveUsuario == _idAlumno.ToString());
```

Se utiliza LINQ para consultar si el estudiante con un determinado **ClaveUsuario** (lo que representa una clave primaria) indicando su existencia.

- **GetTable<Estudiante>():** Obtiene la tabla de estudiantes.
- **.FirstOrDefault(...):** Devuelve el primer estudiante que cumpla con el criterio (si existe), o null si no hay ninguno.
- **Importancia:** Esto evita errores por intentar eliminar un registro que no existe.

Confirmación del usuario

```
if (MessageBox.Show("Realmente deseas eliminar el registro", "Eliminar",
MessageBoxButtons.YesNo) == DialogResult.Yes)
```

- Esto muestra una ventana emergente (diálogo de confirmación) preguntando al usuario si realmente quiere eliminar el registro.
- **Buena práctica:** Siempre pedir confirmación antes de una acción destructiva es clave en cualquier sistema confiable.

Crear objeto del estudiante a eliminar

```
Estudiante estudianteParaEliminar = new Estudiante
{
    ClaveUsuario = listaTextBoxes[0].Text,
    Nombre = listaTextBoxes[1].Text,
    ApellidoPaterno = listaTextBoxes[2].Text,
    ApellidoMaterno = listaTextBoxes[3].Text,
    Telefono = listaTextBoxes[4].Text,
    Direccion = listaTextBoxes[5].Text,
    CorreoElectronico = listaTextBoxes[6].Text,
    Image = imgConvertidaArray
};
```

Aquí se reconstruye un objeto Estudiante con los datos extraídos de varios controles de tipo TextBox.

Eliminar el registro

```
conexion.Delete<Estudiante>(estudianteParaEliminar);
```

- Esta línea llama al método **Delete<T>()** de la clase **Conexion**, pasando el objeto **Estudiante**. Este método realiza la eliminación física del registro de la base de datos.

Confirmación y actualización de interfaz

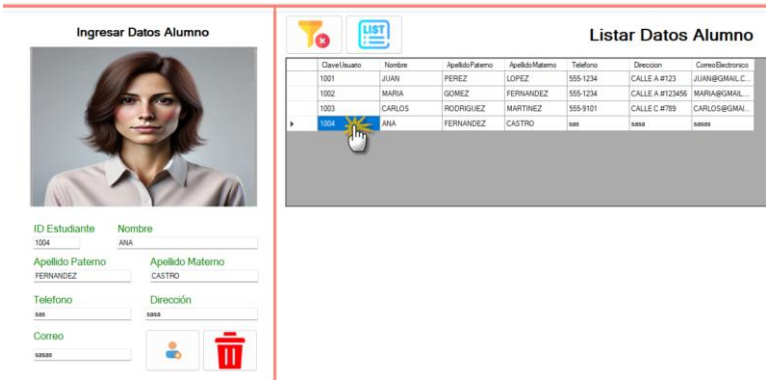
```
MessageBox.Show ("El registro se ha
```

```
eliminado correctamente!");
 //Invocamos el metodo para desplegar
registros
 ListarRegistrosTabla();
 //modificamos la variable
 _accion = "insert";
```

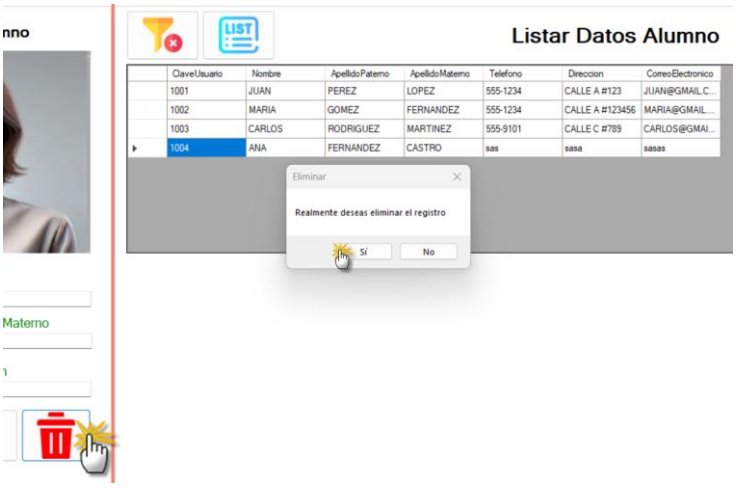
- Se informa al usuario del éxito de la operación.
- Se actualiza la interfaz, **ListarRegistrosTabla()** recarga la lista de registros en la tabla.
- Se modifica una variable de estado **(_accion)** para preparar la interfaz para una nueva inserción, no para una edición.

Ejecutamos y comprobamos la eliminación

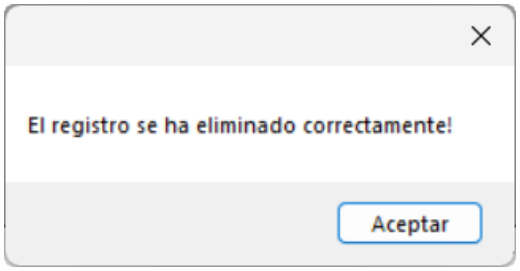
- Seleccionaremos el registro
  - El usuario debe identificar y seleccionar el registro que desea eliminar.



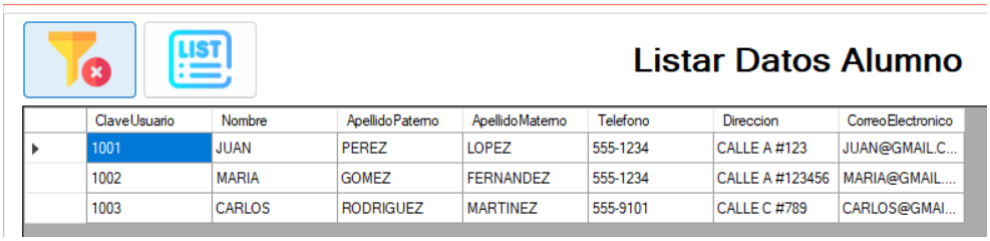
- Posteriormente a pulsar un click sobre el botón que está identificado con el icono "**Eliminar**", nos mostrara el mensaje para confirmar la eliminación del registro.
  - Se hace clic en un botón de la interfaz que está claramente identificado, con un icono de basura o una palabra como Eliminar, Borrar o Delete.



- Nos enviara un mensaje, en donde nos indicara que el registro se ha eliminado de manera correcta
    - Una vez que confirmamos la eliminación, el sistema responde con un mensaje de éxito.
    - Este mensaje puede decir, por ejemplo:
    - “El registro ha sido eliminado correctamente.”
    - Este paso es importante porque confirma que la operación fue ejecutada sin errores.



- Podremos comprobar que en este momento el registro ya no se visualiza
    - Se vuelve a observar la tabla o lista original, y el registro eliminado ya no aparece. Esto permite verificar visualmente que el cambio se hizo.



	ClaveUsuario	Nombre	ApellidoPaterno	ApellidoMaterno	Telefono	Direccion	CorreoElectronico
▸	1001	JUAN	PEREZ	LOPEZ	555-1234	CALLE A #123	JUAN@GMAIL.C...
	1002	MARIA	GOMEZ	FERNANDEZ	555-1234	CALLE A #123456	MARIA@GMAIL....
	1003	CARLOS	RODRIGUEZ	MARTINEZ	555-9101	CALLE C #789	CARLOS@GMAI...

    - Confirma que la base de datos ya no contiene ese elemento, lo cual es fundamental para la integridad del sistema.

JULIAN.escolares - dbo.Estudiante

	ID	ClaveUsuar...	Nombre	ApellidoPa...	ApellidoM...	Telefono	Direccion	CorreoElec...	Image
▸	6	1001	JUAN	PEREZ	LOPEZ	555-1234	CALLE A #1...	JUAN@GM...	<Binary dat...
	7	1002	MARIA	GOMEZ	FERNANDEZ	555-1234	CALLE A #1...	MARIA@G...	<Binary dat...
	8	1003	CARLOS	RODRIGUEZ	MARTINEZ	555-9101	CALLE C #7...	CARLOS@G...	<Binary dat...
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

RESULTADOS ESPERADOS

- El estudiante identifica correctamente el registro seleccionado en el DataGridView para su posible eliminación.
- Verifica la existencia del registro en la base de datos antes de ejecutar la operación de borrado.
- Implementa un cuadro de diálogo que solicita confirmación del usuario antes de eliminar el registro.
- Construye correctamente el objeto Estudiante con base en los datos del formulario para su eliminación.
- Utiliza el método Delete<T>() de la clase Conexion para realizar la eliminación desde la base de datos.
- Actualiza de manera automática la interfaz gráfica tras completar la operación, mediante la recarga de datos.

- Mantiene el control de flujo del sistema asignando la acción "insert" al finalizar el proceso de eliminación.
- Aplica buenas prácticas en el tratamiento de operaciones críticas, como validación previa y retroalimentación al usuario.

ANÁLISIS DE RESULTADOS

- ¿Por qué es importante verificar la existencia del registro antes de intentar eliminarlo de la base de datos?
- ¿Qué función cumple la ventana de confirmación antes de ejecutar la eliminación, y qué problemas podría evitar?
- ¿Cuál es el propósito de reconstruir un objeto Estudiante con los datos del formulario antes de eliminarlo?
- ¿Qué consecuencias tendría eliminar un registro sin actualizar la vista del DataGridView?
- ¿Qué riesgos se presentan al eliminar un registro de forma directa sin validaciones ni confirmación del usuario?
- ¿Cómo influye el uso de la variable _accion en el control del flujo después de eliminar un registro?
- ¿Qué beneficios ofrece encapsular la operación de eliminación en un método dentro de la capa lógica?
- ¿De qué manera esta práctica contribuye a fortalecer la toma de decisiones responsables en el desarrollo de software?

CONCLUSIONES Y REFLEXIONES

Esta práctica permitió al estudiante comprender la importancia de validar y confirmar operaciones críticas como la eliminación de registros en una base de datos. Se reforzó el uso de estructuras de control, la integración con interfaces gráficas y la interacción responsable con el usuario. Además, se aplicaron buenas prácticas de programación orientada a objetos y separación por capas, favoreciendo el desarrollo de soluciones seguras, limpias y sostenibles.

ACTIVIDADES COMPLEMENTARIAS

1. Implementar una papelera de reciclaje lógica:
   - Modifica el sistema para que, en lugar de eliminar permanentemente el registro, se marque con un estado de "inactivo" o "eliminado", permitiendo una futura recuperación.
2. Agregar validación para impedir eliminación sin selección:
   - Integra una validación que impida ejecutar la eliminación si no hay un registro seleccionado en el DataGridView, mostrando un mensaje adecuado.
3. Registrar acciones de eliminación en un archivo de log:
   - Crea un mecanismo que registre cada eliminación realizada, incluyendo fecha, hora, ID del estudiante eliminado y usuario que ejecutó la acción, con fines de auditoría.

EVALUACIÓN Y EVIDENCIAS DE APRENDIZAJE

Criterios de evaluación	• Verifica correctamente la existencia del registro antes de ejecutar la eliminación.

	• Solicita confirmación del usuario mediante un cuadro de diálogo antes de eliminar el registro. • Construye de forma adecuada el objeto Estudiante con los datos del formulario para su eliminación. • Utiliza el método Delete<T>() de la clase Conexion para eliminar el registro de la base de datos. • Actualiza el contenido del DataGridView después de completar la eliminación. • Restablece el estado del formulario para nuevas inserciones asignando el valor "insert" a la variable _accion. • Muestra mensajes claros al usuario durante el proceso de eliminación y al finalizar la operación. • Aplica buenas prácticas de codificación, organización por capas y control de errores. • Demuestra atención al detalle y responsabilidad al manejar información sensible. • Responde con claridad y fundamento a las preguntas del análisis de resultados.
Rúbricas o listas de cotejo para valorar desempeño	Lista de cotejo con siete criterios clave que evalúan la creación del proyecto, el diseño de la interfaz, la funcionalidad del código, el análisis reflexivo y la presentación del reporte. Cada ítem se evalúa con las opciones: “Cumple / No cumple” y espacio para observaciones.
Formatos de reporte de prácticas	**Formato libre estructurado que incluya:** 1. Datos del alumno. 2. Nombre de la práctica. 3. Objetivo. 4. Capturas de pantalla del proyecto. 5. Código implementado. 6. Respuestas al análisis. 7. Conclusiones y Reflexión.

FUENTES DE INFORMACIÓN

Esposito, D. (2020). Architecting modern web applications with ASP.NET Core and Microsoft Azure. Microsoft Learn. https://learn.microsoft.com/en-us/dotnet/architecture/modern-web-apps-azure/

Flaticon. (2023). Free icons and stickers for your projects [Sitio web]. https://www.flaticon.com/

Fowler, M. (2002). Patterns of enterprise application architecture. Addison-Wesley. https://martinfowler.com/books/eaa.html

Google Fonts. (2023). Material icons [Sitio web]. https://fonts.google.com/icons

Microsoft. (2023). App.config file in C#. Microsoft Learn. https://learn.microsoft.com/en-us/dotnet/framework/configure-apps/

Microsoft. (2023). Entity Framework overview. Microsoft Learn. https://learn.microsoft.com/en-us/ef/

Microsoft. (2023). SqlConnection class. Microsoft Learn. https://learn.microsoft.com/en-us/dotnet/api/system.data.sqlclient.sqlconnection

Microsoft. (2023). Use the Visual Studio IDE for C#. Microsoft Learn. https://learn.microsoft.com/en-us/visualstudio/ide/

Microsoft. (2023). Windows Forms overview. Microsoft Learn. https://learn.microsoft.com/en-us/dotnet/desktop/winforms/overview/

Pexels. (2023). Free stock photos & videos [Sitio web]. https://www.pexels.com/

ResizePixel. (2023). Resize image online [Sitio web]. https://www.resizepixel.com/

Stack Overflow. (2023). Best practices in Windows Forms applications [Sitio web]. Stack Overflow. https://stackoverflow.com/questions/tagged/winforms

W3Schools. (2023). C# tutorial. W3Schools. https://www.w3schools.com/cs/

UES
Universidad Estatal de Sonora
La Fuerza del Saber Estimulará mi Espíritu

ANEXOS

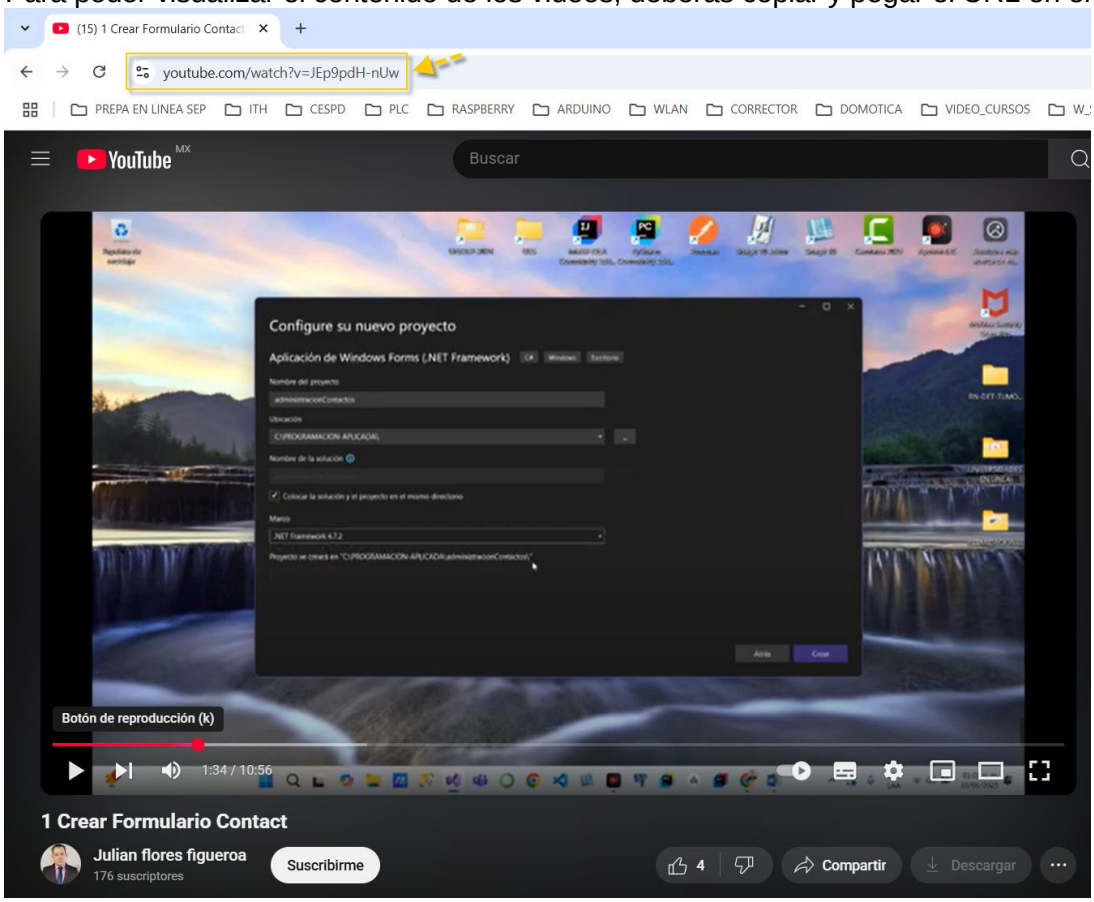
VIDEOS: FORMULARIO, MODELO, CAPA LOGICA, CAPA DE CONEXIÓN	
1-Paso a Paso: Crear un Formulario de Contacto	https://youtu.be/JEp9pdH-nUw?si=nMbEz9sypRkFeXMm
2-Cómo Crear un Formulario de Detalles de Contacto	https://youtu.be/SudiBSJzKSQ?si=GHiE4e98niSrC5_w
3-Diseño de la Capa Modelo y la Lógica de Negocio	https://youtu.be/gpZcVB8dZvc?si=JlBltSGI888YoG-P
4-Integración y Gestión de la Capa de Acceso a Datos	https://youtu.be/1Rmg4sNx2zQ?si=nWmzUj91q4ZTgk8J

VIDEOS-ELEMENTO-1: DISEÑO DE FORMULARIO	
5 Diseño de Formularios Video 1 Creacion de Formulario y controles en formulario	https://youtu.be/nOOFthD_pk4?si=9t0EjtP0Xz86W8IX
6 Diseño de Formularios Video 2 Incorporacion de controles y definicion de propiedades	https://youtu.be/IQMdZChScyk?si=IHEvB3sWXU12Z73N
7 Diseño de Formularios Video 3 Creación del Modelo, Creacion de la capa Logica y capa conexion	https://youtu.be/SbxHGmE0CnQ?si=vtQGrOB3s6SpYRjV
8 Diseño de Formularios Video 4 Detallar Capa de Conexion e Integrar Capas en la Vista	https://youtu.be/_QCspe5SH3s?si=I9HQUikThCeGHaQ-
9 Diseño de Formularios Video 5 Establecer propiedades para valores adicionales y creación de lista	https://youtu.be/LfK3sKunI0M?si=CM_-xX4xpgitvRt3

VIDEOS: DISEÑO DE FORMULARIO	
10 Formulario Sistema Escolar Video 1 Control GroupBox y manejo de propiedades	https://youtu.be/yjMb3NqdbMc?si=AJqw4RF0dmYprxNI
11 Formulario Sistema Escolar Video 2 Control GroupBox y manejo de propiedades	https://youtu.be/bzUHq2T0RV0?si=1eNbEnxCFNmh_GLv
12 Formulario Sistema Escolar Video 3 Creación del Método que carga la imagen en el control PictureBox	https://youtu.be/4mn-m-Qn4Mw?si=RtFVuh2f0qj_nT-r
13 Formulario Sistema Escolar Video 4 Aplicación de la Herencia y utilización del método Clic en el Control PictureBox	https://youtu.be/S_A72-0H67g?si=Gfye2cQ4Waia74_I
14 Formulario Sistema Escolar Video 5 Trabajamos con el evento Text Changed	https://youtu.be/YoMcMGNof30
15 Formulario Sistema Escolar Video 6 Trabajamos con Validación de solo letras, espacio y retroceso	https://youtu.be/AXm4TMTar6E

16 Formulario Sistema Escolar Video 7 Trabajamos con Validación de correo electrónico	https://youtu.be/5KrZtyH_-5w
17 Formulario Sistema Escolar Video 8 Colección de datos y constructor	https://youtu.be/GwNl7dP_29g
18 Formulario Sistema Escolar Video 9 Validar TextBox utilizando la colección de datos P1	https://youtu.be/7NtZICWCIxk
19 Formulario Sistema Escolar Video 9 Validar TextBox utilizando la colección de datos P2	https://youtu.be/M6kl63GZECg

NOTA: Para poder visualizar el contenido de los videos, deberás copiar y pegar el URL en el navegador




UES
Universidad Estatal de Sonora
La Fuerza del Saber Estimulará mi Espíritu