Universidad Claeh
Centro CTC

Analista Programador
Programación II

Proyecto Web
Trabajo Obligatorio

Luis Virriel
Danlee Garcia

2024

Índice

Resumen

La propuesta de este proyecto consiste en el desarrollo de una aplicación web en C# y ASP.NET para el uso de DanLuis Solutions, una empresa ficticia dedicada a la gestión de clientes y la asignación de órdenes de trabajo. Esta aplicación permite a los administradores agregar clientes y técnicos, asignarles órdenes de trabajo y gestionar de manera eficiente todas las tareas relacionadas con estos procesos.

El sistema se basa en principios de programación orientada a objetos, aplicando conceptos como las clases , los métodos y atributos que la conforman, lo que permite una gestión eficiente de los datos y las funcionalidades.Por otro lado le dimos uso a las listas Enum , para personalizar el timo de categoría del estado de una orden.

En cuanto a la estructura del sistema, se ha implementado una base de datos ficticia, que simula el almacenamiento de datos sin la necesidad de utilizar un servidor de base de datos real. Esta base de datos interna está representada por una clase denominada Base De Datos, que contiene listas de objetos como clientes, órdenes de trabajo, técnicos y otros elementos esenciales para el funcionamiento de la aplicación.

La interfaz de la aplicación está diseñada con Bootstrap, utilizando sistemas grid para garantizar una visualización responsiva y adaptativa en diferentes dispositivos. Esto asegura una experiencia de usuario fluida y atractiva, permitiendo la gestión de clientes y tareas en cualquier momento y desde cualquier lugar. Los administradores pueden agregar clientes y técnicos, asignarles órdenes de trabajo, y editar los detalles de los mismos a través de una interfaz limpia y fácil de navegar.

El uso de listas en C# para almacenar los datos de clientes, órdenes de trabajo, Técnicos y otros elementos clave del sistema hace que el proceso de búsqueda, actualización y eliminación de registros sea eficiente. Estas listas se gestionan dentro de la clase BaseDeDatos y se actualizan dinámicamente a medida que los usuarios interactúan con la aplicación, manteniendo una estructura organizada y accesible.

La aplicación también permite la asignación de tareas a los clientes, con un control detallado de las órdenes de trabajo que se generan. A través de un sencillo flujo de trabajo, el administrador puede crear nuevas órdenes, asociarlas a clientes específicos, y hacer seguimientos de su estado y progreso.

En resumen, DanLuis Solutions ha sido diseñada para ofrecer una solución completa para la gestión de clientes y órdenes de trabajo, utilizando principios de programación orientada a objetos, una estructura de base de datos interna, y tecnologías como Bootstrap , C# y ASP.NET. La aplicación no solo optimiza la gestión de datos, sino que también ofrece una experiencia de usuario intuitiva, escalable y eficiente.

1. Introducción

La programación orientada a objetos (POO) es un paradigma fundamental en el desarrollo de software, basado en la creación y manipulación de clases y objetos. Las clases actúan como moldes o plantillas que definen las características y comportamientos comunes de los objetos, permitiendo que estos se instancian con atributos específicos. Este enfoque promueve la reutilización del código y una estructura organizada, facilitando la extensión y el mantenimiento de la aplicación.

Un concepto clave en POO es la herencia, que permite que una clase derive de otra, heredando atributos y métodos de la clase base. Esto favorece la optimización del código, ya que las clases derivadas pueden modificar o extender la funcionalidad de la clase base sin necesidad de duplicar código. En este proyecto, se utiliza la herencia para estructurar eficientemente las relaciones entre diferentes entidades, como clientes y órdenes de trabajo, lo que mejora la gestión de los datos y simplifica la implementación de nuevas características.

Las listas son otro componente crucial dentro de la programación orientada a objetos. Permiten almacenar colecciones de elementos del mismo tipo y realizar diversas operaciones, como agregar, eliminar y ordenar datos. En este proyecto, las listas en C# juegan un papel fundamental en la organización y manipulación de información, desde los clientes y órdenes de trabajo hasta los usuarios y otras entidades clave del sistema.
Aunque no se aplicaron directamente los conceptos de agregación y composición en este proyecto, estos son fundamentales en POO, ya que permiten establecer relaciones más complejas entre objetos, donde un objeto puede ser parte de otro o contener otros objetos dentro de su estructura. Estas relaciones ofrecen una mayor flexibilidad y modularidad en el diseño del sistema.

En cuanto al desarrollo web, se hace uso de lenguajes como HTML y CSS dentro del marco de ASP.NET, que sirve como plantilla para la aplicación web. HTML es utilizado para estructurar el contenido en la interfaz, mientras que CSS se encarga de mejorar la apariencia visual mediante estilos y diseños.
Dentro de ASP.NET, se incorporan las bibliotecas de Bootstrap, que enriquecen la experiencia de usuario al proporcionar componentes y estilos predefinidos que facilitan la creación de una interfaz responsiva y atractiva. Bootstrap optimiza la experiencia del usuario en dispositivos móviles y de escritorio, garantizando una navegación fluida y accesible en diferentes plataformas.

1.1. Clases y Objetos

En la programación orientada a objetos (POO), las clases y objetos son dos de los conceptos fundamentales que permiten organizar y estructurar el código de manera eficiente. A continuación, te explico qué son y cómo se interrelacionan:

Una clase es una especie de "plantilla" o "molde" para crear objetos. Define un conjunto de atributos (también llamados propiedades o campos) y métodos (también llamados funciones o comportamientos) que los objetos creados a partir de esa clase compartirán. Es decir, la clase es la definición de un tipo de objeto, pero por sí misma no es un objeto real, sino una especie de modelo para generar objetos.

Por ejemplo, en una aplicación para gestionar vehículos, podrías tener una clase llamada Coche, que define qué características (atributos) y qué acciones (métodos) tendrán todos los coches de la aplicación.

1

Un objeto es una instancia concreta de una clase. Es decir, es un elemento real que se crea a partir de la plantilla definida por la clase. Un objeto tiene sus propios valores para los atributos definidos en la clase y puede ejecutar los métodos definidos en esa clase.
Siguiendo con el ejemplo de la clase Coche, un objeto de esta clase sería un coche específico, con valores particulares para su marca, modelo y año.
Para crear un objeto, se utiliza la palabra clave new en la mayoría de los lenguajes orientados a objetos (como C# o Java).

1.2. Clases Enum:

Las clases enum (enumeraciones) son tipos definidos por el usuario que consisten en un conjunto de constantes con nombres específicos. Estas son útiles para representar valores que tienen un significado limitado y definido, como días de la semana, estados de un proceso, o colores.

1. Definición: En lenguajes como C + +, Java o Kotlin, las enumeraciones permiten agrupar constantes relacionadas en una estructura organizada y legible.
2. Ventajas:
    - Restringen los valores posibles de una variable a los enumerados en la clase.
    - Mejoran la seguridad del tipo, especialmente en lenguajes como C++ donde las clases enum son más estrictas que las enumeraciones tradicionales .

1.3. Listas

Las listas en C# Forman parte de los tipos de datos nativos en el framework de .NET y nos permiten almacenar secuencias de variables. Por ejemplo, si deseamos gestionar un conjunto de nombres de personas, podemos utilizar una variable de tipo List, que se corresponde con una lista de cadenas de texto en la sintaxis de C#.
List<string> TecnicosNombres;

Esta estructura List en C# Es fuertemente tipada, lo que significa que, como se muestra en el ejemplo, al crear una lista, debemos especificar el tipo de objetos que contendrá (ya sea cadenas de texto, números u otro tipo definido por el usuario).

1.4. ASP.NET

ASP.NET es un marco de trabajo (framework) gratuito y de código abierto que permite desarrollar sitios web y aplicaciones web de alto rendimiento utilizando tecnologías como HTML, CSS y JavaScript. Además, ASP.NET facilita la creación de API web y la implementación de tecnologías en tiempo real, como WebSockets, para aplicaciones interactivas.
ASP.NET ofrece tres marcos principales para desarrollar aplicaciones web: Web Forms, ASP.NET MVC y ASP.NET Web Pages. Los tres marcos son maduros, estables y ampliamente utilizados, lo que permite crear aplicaciones web de calidad con cualquiera de ellos. Al elegir cualquiera de estos marcos, el desarrollador se beneficia de las ventajas y características comunes de ASP.NET, sin importar la opción seleccionada.
Cada uno de estos marcos está diseñado para diferentes enfoques de desarrollo. La elección del marco dependerá de factores como los recursos disponibles (conocimientos, habilidades y experiencia del equipo de desarrollo), el tipo de aplicación a desarrollar y las preferencias personales del programador en cuanto a estilo de desarrollo.

- ASP.NET Web Forms permite crear sitios web dinámicos mediante un modelo de programación basado en eventos y un enfoque de diseño de tipo "arrastrar y soltar". Esto facilita el desarrollo rápido de sitios web sofisticados con interfaces de usuario (UI) controladas y la gestión de datos a través de una amplia variedad de controles y componentes predefinidos.
- ASP.NET MVC se enfoca en el patrón de diseño Modelo-Vista-Controlador (MVC), ofreciendo un mayor control sobre la estructura y el flujo de la aplicación. Es ideal para desarrolladores que prefieren tener un control más granular sobre el código y la presentación.
- ASP.NET Web Pages es un marco más sencillo y ligero, adecuado para proyectos más pequeños o cuando se busca rapidez en el desarrollo sin la complejidad adicional de MVC o Web Forms.

Todos estos marcos están construidos sobre el .NET Framework, lo que significa que comparten una base común de funcionalidad y herramientas. Por ejemplo, los tres marcos cuentan con un sistema de autenticación basado en roles para la gestión de seguridad y acceso de usuarios, así como mecanismos para manejar solicitudes HTTP, gestionar sesiones y otros aspectos esenciales de las aplicaciones web.
En resumen, ASP.NET proporciona flexibilidad y escalabilidad a los desarrolladores, permitiendo elegir el marco que mejor se adapte a sus necesidades y estilo de trabajo, mientras aprovechan la robustez y las características compartidas del .NET Framework.

1.5. HTML(Lenguaje de etiquetas de hipertexto)

HTML (Lenguaje de Marcas de Hipertexto, por sus siglas en inglés HyperText Markup Language) es el componente fundamental de la web. Se encarga de definir la estructura y el significado del contenido en las páginas web. Aunque HTML establece la base de cualquier sitio, normalmente se complementa con otras tecnologías, como CSS para la presentación y el diseño visual, y JavaScript para agregar interactividad y comportamiento dinámico.

El término hipertexto se refiere a los enlaces que conectan páginas web entre sí, tanto dentro de un mismo sitio como entre diferentes sitios de la red. Los enlaces son una parte

esencial de la World Wide Web, ya que permiten navegar entre contenidos relacionados. Cuando subimos contenido a Internet y lo vinculamos con otros sitios, participamos activamente en la creación de la web interconectada.

HTML usa etiquetas o "marcas" para señalar y organizar el contenido dentro de una página. Estas marcas indican cómo deben mostrarse los elementos, como texto, imágenes, y otros recursos, en un navegador. Algunas de las etiquetas más comunes incluyen <head>, <title>, <body>, <header>, <footer>, <article>, <section>, <p>, , , <img>, <video>, <ul>, <ol>, <li>, entre otras.

Cada elemento HTML está definido por una etiqueta, que consiste en el nombre del elemento entre corchetes angulares (< >). Las etiquetas son insensibles a mayúsculas o minúsculas, lo que significa que puedes escribirlas de cualquier manera: por ejemplo, <title>, <TITLE> o <TiTle> son igualmente válidas.

En resumen, HTML es el lenguaje que da forma a las páginas web, estructurando y organizando el contenido que los navegadores mostrarán al usuario. A través de sus etiquetas, HTML permite la creación de un ecosistema interconectado de información, enlazando páginas y recursos en la vasta red que conocemos como la web.

1.6. CSS: Cascading Style Sheets

CSS (Cascading Style Sheets u Hojas de Estilo en Cascada) es el lenguaje utilizado para definir la presentación y el diseño de documentos HTML o XML (incluyendo otros lenguajes basados en XML como SVG, MathML o XHTML). CSS se encarga de especificar cómo se deben visualizar los elementos estructurados en una página web, ya sea en la pantalla, en papel, o en otros medios como dispositivos móviles o lectores de voz.

Con CSS, es posible modificar aspectos visuales y estilísticos de una página web, como la fuente, el color, el tamaño, el espaciado del texto, la disposición de los elementos en el espacio (por ejemplo, dividiendo el contenido en varias columnas), o incluso agregar animaciones y otros efectos visuales.
El principal objetivo de CSS es separar la estructura del contenido, que se define en HTML, de su presentación visual. Esto permite a los desarrolladores web controlar de manera más eficiente el diseño y la apariencia de un sitio sin alterar la estructura subyacente del contenido. Además, CSS hace que sea posible aplicar un mismo estilo a múltiples páginas, facilitando la consistencia y el mantenimiento del diseño en todo un sitio web.
Por ejemplo, mediante CSS se pueden aplicar efectos como cambio de color al pasar el mouse (hover), transiciones suaves, fondo degradado, y mucho más, permitiendo así mejorar la experiencia visual del usuario y hacer las páginas más interactivas y atractivas.
En resumen, CSS es esencial para transformar una página web básica en una experiencia visualmente atractiva y funcional, proporcionando a los desarrolladores un control total sobre la presentación del contenido.

1.7. Bootstrap

Bootstrap es un framework de código abierto diseñado para facilitar el desarrollo de sitios y aplicaciones web. Fue creado por Twitter y se ha convertido en una herramienta ampliamente utilizada gracias a su capacidad para proporcionar un diseño responsivo y profesional sin necesidad de escribir extensos códigos CSS o JavaScript. Es ideal tanto para desarrolladores principiantes como avanzados debido a su enfoque modular y su facilidad de integración.

El núcleo de Bootstrap se basa en un sistema de rejilla responsiva de 12 columnas que permite estructurar el contenido de manera flexible, ajustándose a cualquier dispositivo. Además, ofrece una amplia gama de componentes listos para usar, como botones, menús, formularios y carruseles, que se pueden personalizar según las necesidades del proyecto. También incluye utilidades para estilos CSS básicos y avanzados, así como integraciones con JavaScript que añaden interactividad. Su integración es sencilla, mediante un enlace CDN o descargando los archivos necesarios para el proyecto.

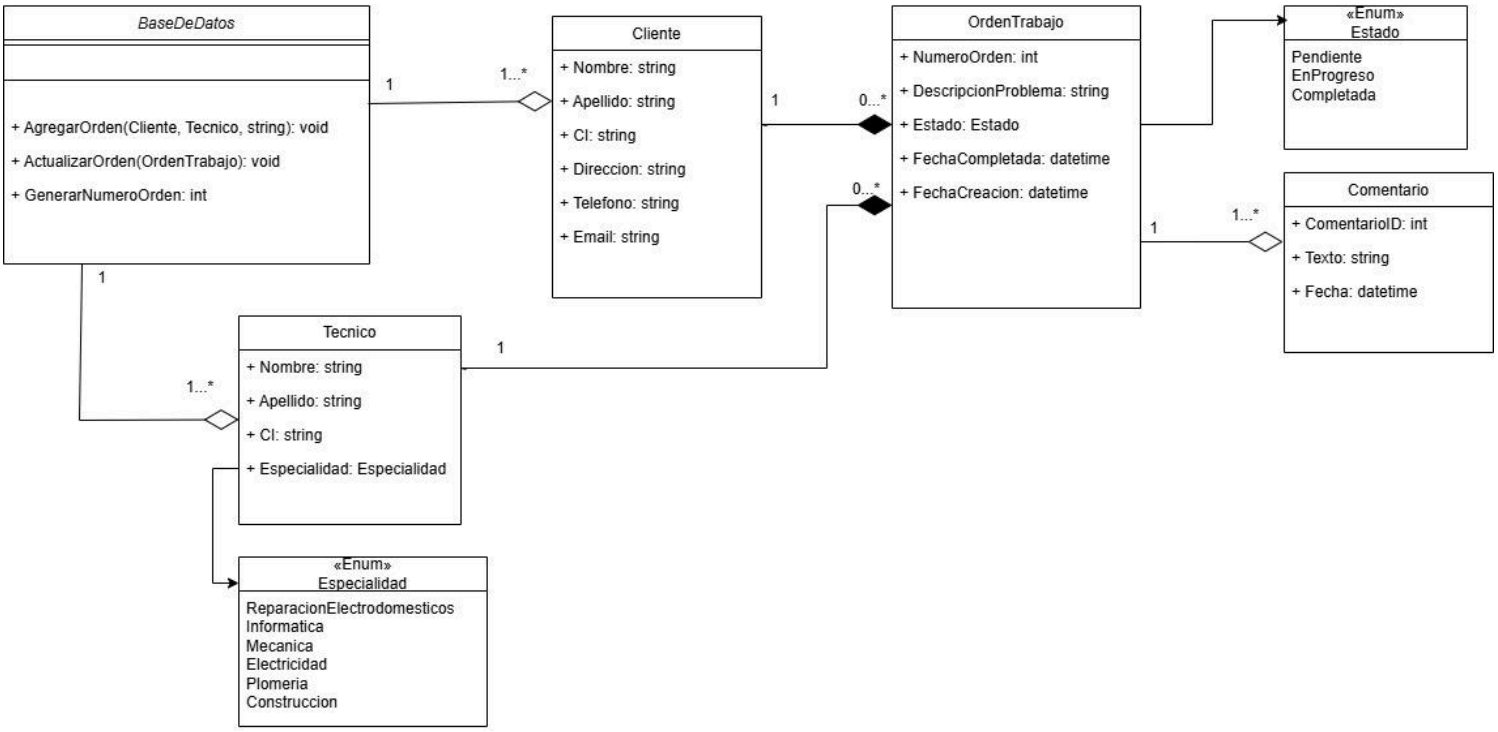
En conclusión, Bootstrap es una herramienta poderosa para el diseño y desarrollo web que ahorra tiempo y esfuerzo, garantizando resultados profesionales y adaptativos. Su popularidad y soporte continuo lo convierten en una opción confiable para proyectos de cualquier escala, desde sitios personales hasta aplicaciones corporativas.

1.8. Diagrama UML

Un diagrama UML (Lenguaje de Modelado Unificado) en programación orientada a objetos (POO) es una representación visual de las clases, objetos y sus relaciones dentro de un sistema. El diagrama de clases UML es una de las herramientas más importantes, ya que muestra cómo las clases interactúan entre sí mediante sus atributos (propiedades) y métodos (funciones), así como las relaciones entre ellas, como la herencia, la composición y la asociación.

En un diagrama de clases UML, cada clase se representa con un rectángulo dividido en tres secciones: el nombre de la clase en la parte superior, los atributos en el medio y los métodos en la parte inferior. Además, las relaciones entre clases se muestran con diferentes tipos de líneas y flechas, indicando cómo las clases se conectan y se influyen mutuamente.

Este tipo de diagrama es fundamental para estructurar y planificar un sistema en POO, ya que facilita la comprensión de la arquitectura y la interdependencia de los componentes del sistema, ayudando tanto en el diseño como en el mantenimiento del software.



1. Clase BaseDeDatos : Esta clase representa la base de datos y tiene métodos para administrar órdenes de trabajo.
    - Métodos:
        - AgregarOrden(Cliente, Tecnico, string): void:Agrega una orden de trabajo, asociándola a un cliente y a un técnico.
        - ActualizarOrden(OrdenTrabajo): void:Actualiza la orden de trabajo.
        - GenerarNumeroOrden(): int:Genera un número de orden de trabajo.
2. Clase Cliente : Representa a un cliente con información personal.
    - Atributos:
        - Nombre, Apellido, CI, Direccion, Telefono, Email(Datos del cliente).
3. Clase Tecnico : Representa a un técnico.
    - Atributos:
        - Nombre, Apellido, CI(Datos personales del técnico).
        - Especialidad:Enumeración que representa diferentes especializaciones como reparación de electrónica, TI y construcción.
4. OrdenTrabajo Clase : Representa una orden de trabajo.
    - Atributos:
        - NumeroOrden, DescripcionProblema, Estado, FechaCreacion, FechaCompletada.
        - Está relacionado con ambos Cliente, Tecnicolo que significa que cada pedido está vinculado a un cliente y un técnico.
5. Comentario Clase : Representa un comentario asociado a una orden de trabajo.
    - Atributos:
        - ComentarioID, Texto, Fecha: Comentarios relacionados con órdenes de trabajo.
6. Estado Enum : Representa el estado de la orden de trabajo, el cual puede ser:

    - “Pendiente”(Pendiente), “EnProgreso”(En progreso) o “Completada”(Completado).

El diagrama muestra cómo se relaciona cada clase:

- “BaseDeDatos” pueden gestionar múltiples “Cliente” objetos “Tecnico”.
- Cada uno OrdenTrabajo está asociado a uno “Cliente” y uno “Tecnico”(pero un cliente o técnico puede estar vinculado a varios pedidos).
- “OrdenTrabajo” Puede tener varios “Comentario” objetos, pero cada comentario está vinculado a un solo orden.

2. Clases

Como se puede pude apreciar las clases son lo que más abarca nuestro proyecto, además de ser la estructura principal de nuestro proyecto. Poseen una amplia gama de información y reflejan en sus atributos lo que caracteriza a cada una, lo que le da funcionalidad a la aplicación. Entre ellas podemos encontrar tales como Cliente y Técnico o las clases Enum. A continuación le vamos a mostrar detalladamente cada clase.

2.1. Clase BaseDeDatos

La clase "BaseDeDatos" se erige como la piedra angular del programa, albergando las listas que contienen toda la información disponible en el sistema, tales como las listas de Clientes, Técnicos y Órdenes De Trabajo . Esta clase se presenta como abstracta y estática, lo que posibilita su accesibilidad desde cualquier parte del código. Adicionalmente, se implementó la carga inicial de datos en esta clase, con el objetivo de dotar al programa de información preexistente para facilitar la gestión del sistema y la verificación de las funcionalidades

```
public abstract class BaseDeDatos
{
    13 referencias
    public static List<Cliente> Clientes { get; set; } = new List<Cliente>
    {
        new Cliente("Juan", "Pérez", "51742259", "Calle Falsa 123", "555-1234", "juan.perez@easd.com"),
        new Cliente("María", "González", "33896775", "Avenida Siempre Viva 456", "555-5678", "maria.gonzalez@dasdasd.com"),
        new Cliente("Carlos", "Sánchez", "30152485", "Calle Real 789", "555-9101", "carlos.sanchez@sefesf3.com"),
        new Cliente("Ana", "Lopez", "28886967", "Boulevard  101", "555-1213", "ana.lopez@por.com"),
        new Cliente("Pepe", "Martinez", "40579087", "Plaza 202", "555-1415", "luis.martinez@ejimplo.com")
    };
```

Figura 2.2.1: Atributos de la clase “BasedeDatos” , lista estática “Clientes”.

```
12 referencias
public static List<Tecnico> Tecnicos { get; set; } = new List<Tecnico>
{
    new Tecnico("Pedro", "Ramírez", "57211058", Especialidad.ReparacionElectrodomesticos),
    new Tecnico("Lucía", "Cruz", "32100531", Especialidad.Informatica),
    new Tecnico("Luis", "Virriel", "44952182", Especialidad.Informatica),
    new Tecnico("Jorge", "Hernandez", "4214160", Especialidad.Mecanica),
    new Tecnico("Danlee", "Garcia", "65574298", Especialidad.Informatica),
    new Tecnico("Andrés", "Martínez", "20839201", Especialidad.Plomeria)
};
19 referencias
```

Figura 2.2.1: Atributos de la clase “BasedeDatos” , lista estática “Técnicos”.

```
public static List<OrdenTrabajo> OrdenesDeTrabajo { get; set; } = new List<OrdenTrabajo>();

0 referencias
public static void AgregarOrden(Cliente cliente, Tecnico tecnico, string descripcionProblema, Estado estado)
{
    int numeroOrden = GenerarNumeroOrden();
    OrdenTrabajo nuevaOrden = new OrdenTrabajo(numeroOrden, cliente, tecnico, descripcionProblema, estado);


    OrdenesDeTrabajo.Add(nuevaOrden);
}
```

Figura 2.2.1: Atributos y Método de la clase “BasedeDatos” , lista estática “OrdenesDeTrabajo” y método para agregar una Orden.

```
2 referencias
public static int GenerarNumeroOrden()
{
    if (OrdenesDeTrabajo.Count == 0)
    {
        return 1;
    }
    else
    {
        return OrdenesDeTrabajo.Max(o => o.NumeroOrden) + 1;
    }
}
0 referencias
public static void ActualizarOrden(OrdenTrabajo ordenActualizada)
{
    var ordenExistente = OrdenesDeTrabajo.FirstOrDefault(o => o.NumeroOrden == ordenActualizada.NumeroOrden);
    if (ordenExistente != null)
    {
        ordenExistente.ClienteAsociado = ordenActualizada.ClienteAsociado;
        ordenExistente.TecnicoAsignado = ordenActualizada.TecnicoAsignado;
        ordenExistente.DescripcionProblema = ordenActualizada.DescripcionProblema;
        ordenExistente.Estado = ordenActualizada.Estado;
        ordenExistente.Comentarios = ordenActualizada.Comentarios;
        ordenExistente.FechaCompletada = ordenActualizada.FechaCompletada;
    }
}
```

Figura 2.2.1: Métodos de la clase “BasedeDatos” , método “GenerarNumeroOrden” y “ActualizarOrden”.

2.2. Clase Clase “Cliente”

La clase clientes posee atributos simples que se implementan tanto para agregar un cliente como para mostrar su información.

```
public class Cliente
{
    5 referencias
    public string Nombre { get; set; }
    5 referencias
    public string Apellido { get; set; }
    8 referencias
    public string CI { get; set; }
    3 referencias
    public string Direccion { get; set; }
    3 referencias
    public string Telefono { get; set; }
    3 referencias
    public string Email { get; set; }
    1 referencia
    public Cliente() { }

    5 referencias
    public Cliente(string nombre, string apellido, string ci, string direccion, string telefono, string email)
    {
        Nombre = nombre;
        Apellido = apellido;
        CI = ci;
        Direccion = direccion;
        Telefono = telefono;
        Email = email;
    }
```

Figura 2.2.2:Atributos de la clase”Cliente” .

2.3. Clases Enum “Comentario” , “Especialidad” y “Estados”

```
public class Comentario
{
    5 referencias
    public int ComentarioID { get; set; }
    5 referencias
    public string Texto { get; set; }
    3 referencias
    public DateTime Fecha { get; set; }

    1 referencia
    public Comentario(int comentarioID, string texto, DateTime fecha)
    {
        ComentarioID = comentarioID;
        Texto = texto;
        Fecha = fecha;
    }
```

Figura 2.2.3 Clase “Comentario”

```
13 referencias
public enum Especialidad
{
    ReparacionElectrodomesticos,
    Informatica,
    Mecanica,
    Electricidad,
    Plomeria,
    Construccion
}
```

Figura 2.2.3 Clase “Especialidad”

```
17 referencias
public enum Estado
{
    Pendiente,
    EnProgreso,
    Completada
}
```

Figura 2.2.3 Clase “Estado”

2.4. Clase “OrdenDeTrabajo”

```
public class OrdenTrabajo
{
    12 referencias
    public int NumeroOrden { get; set; }
    7 referencias
    public Cliente ClienteAsociado { get; set; }
    11 referencias
    public Tecnico TecnicoAsignado { get; set; }
    6 referencias
    public string DescripcionProblema { get; set; }
    3 referencias
    public DateTime FechaCreacion { get; set; }
    12 referencias
    public Estado Estado { get; set; }
    15 referencias
    public List<Comentario> Comentarios { get; set; }
    7 referencias
    public DateTime? FechaCompletada { get; set; }

    2 referencias
    public OrdenTrabajo(int numeroOrden, Cliente clienteAsociado, Tecnico tecnicoAsignado, string descripcionProblema, Estado estado)
    {
        NumeroOrden = numeroOrden;
        ClienteAsociado = clienteAsociado;
        TecnicoAsignado = tecnicoAsignado;
        DescripcionProblema = descripcionProblema;
        FechaCreacion = DateTime.Now;
        Estado = estado;
        Comentarios = new List<Comentario>();
        FechaCompletada = null;

    }
```

Figura 2.2.4 Clase “OrdenTrabajo”

2.5. Clase Tecnico

```
public class Tecnico
{
    7 referencias
    public string Nombre { get; set; }
    6 referencias
    public string Apellido { get; set; }
    14 referencias
    public string CI { get; set; }
    4 referencias
    public Especialidad Especialidad { get; set; }
    1 referencia
    public Tecnico() { }
    6 referencias
    public Tecnico(string nombre, string apellido, string ci, Especialidad especialidad)
    {
        Nombre = nombre;
        Apellido = apellido;
        CI = ci;
        Especialidad = especialidad;
    }
```

3. HTML , Estructura de las páginas, Estilos(CSS y Bootstrap) y Código C#

Tanto el HTML , CSS y Bootstrap fueron esenciales para la estructura y la recepción de información detrás de cada interacción de un usuario para con la página. A continuación mostraremos dichas estructuras .

3.1 Página Inicio(Default solo accesible por administrador)

Este código pertenece a una página ASP.NET que permite gestionar órdenes de trabajo. Está estructurado en tres secciones principales que corresponden a los estados de las órdenes: "Pendientes", "En Progreso" y "Completadas". Cada sección muestra el número total de órdenes en ese estado y proporciona un botón para ver más detalles. Los detalles se visualizan en una tabla organizada dentro de un GridView, que está inicialmente oculto. Los GridView muestran información clave sobre las órdenes, como el número de orden, el cliente asociado, el técnico asignado, la descripción del problema, la fecha de creación, el estado de la orden y los comentarios relacionados. Cada tabla tiene campos de datos que se vinculan dinámicamente con los objetos de la base de datos, lo que permite mostrar la información actualizada según el estado de cada orden. Además, el diseño es responsive, utilizando clases de Bootstrap para una disposición en columnas, y se asegura de que las tablas sean legibles y accesibles, con un estilo oscuro y botones interactivos.

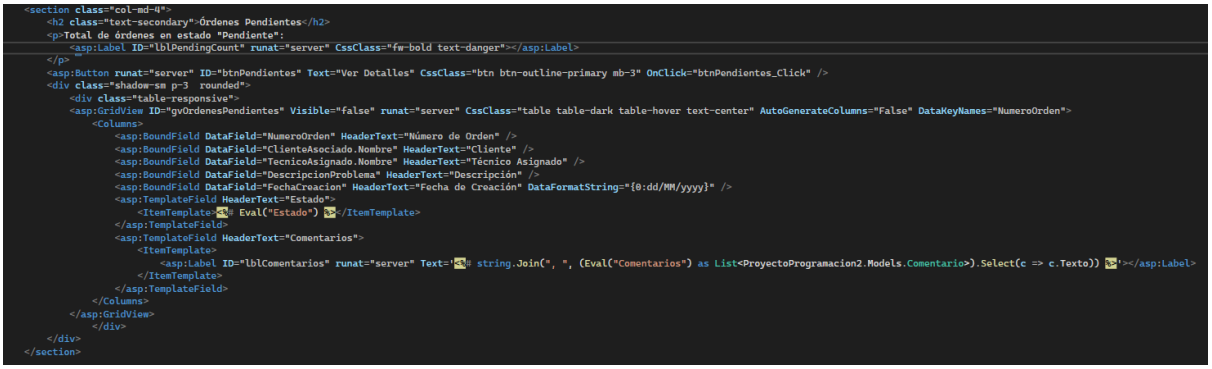


Figura 2.3.1 GridView de Ordenes Pendientes

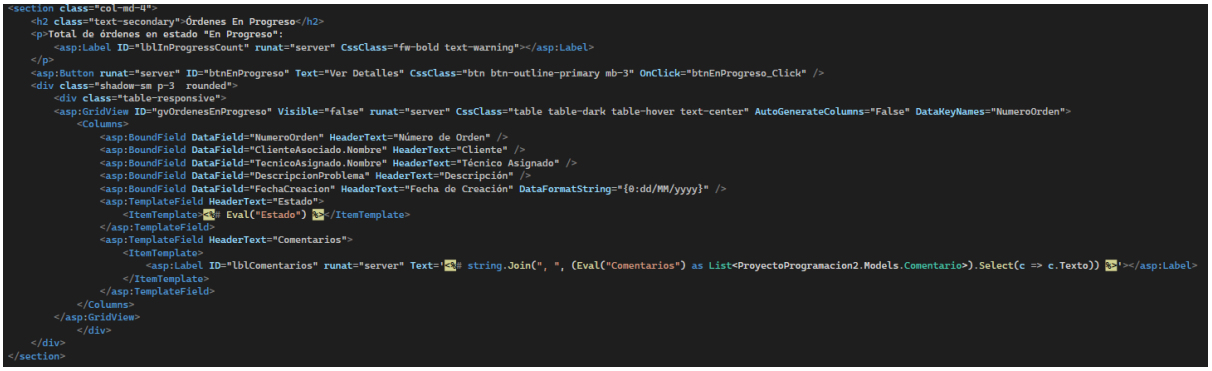


Figura 2.3.1 GridView de Órdenes en Progreso

Figura 2.3.1 GridView de Órdenes Completadas

Este código es una implementación de un formulario web en ASP.NET que gestiona órdenes de trabajo, mostrando las órdenes según su estado: pendientes, en progreso y completadas.

1. Page_Load (Evento de Carga de Página):
    - El método “Page_Load” se ejecuta cuando se carga la página, pero solo si no es una solicitud de postback (es decir, la primera vez que se carga la página).
    - Si no es un postback, se llama a “CargarDatos()”, un método que carga las órdenes de trabajo filtradas por su estado.
2. CargarDatos (Carga de Datos):
    - Se filtran las órdenes de trabajo por su estado: pendiente, en progreso y completada, usando el método FiltrarPorEstado.
    - Para cada estado, se asignan los resultados a diferentes GridView (controles que muestran las órdenes) y se actualizan las etiquetas (lblPendingCount, lblInProgressCount, lblCompletedCount) con la cantidad de órdenes correspondientes.

```
0 referencias
protected void Page_Load(object sender, EventArgs e)
{
    if (!IsPostBack)
    {
        CargarDatos();
    }
}

1 referencia
private void CargarDatos()
{
    var ordenesPendientes = FiltrarPorEstado(Estado.Pendiente);
    gvOrdenesPendientes.DataSource = ordenesPendientes;
    gvOrdenesPendientes.DataBind();

    var ordenesEnProgreso = FiltrarPorEstado(Estado.EnProgreso);
    gvOrdenesEnProgreso.DataSource = ordenesEnProgreso;
    gvOrdenesEnProgreso.DataBind();

    var ordenesCompletadas = FiltrarPorEstado(Estado.Completada);
    gvOrdenesCompletadas.DataSource = ordenesCompletadas;
    gvOrdenesCompletadas.DataBind();

    lblPendingCount.Text = ordenesPendientes.Count.ToString();
    lblInProgressCount.Text = ordenesEnProgreso.Count.ToString();
    lblCompletedCount.Text = ordenesCompletadas.Count.ToString();
}
```

3. FiltrarPorEstado:
    - Este método toma un estado como parámetro (por ejemplo, Estado.Pendiente) y devuelve una lista de órdenes de trabajo que coinciden con ese estado usando LINQ.
4. Eventos de Botones:
    - Los métodos btnPendientes_Click, btnEnProgreso_Click, y btnCompletadas_Click controlan la visibilidad de los diferentes GridView, permitiendo al usuario alternar entre las órdenes de trabajo de diferentes estados.

Este enfoque permite mostrar las órdenes según su estado y alternar entre ellas de manera dinámica sin recargar toda la página.

```
3 referencias
public List<OrdenTrabajo> FiltrarPorEstado(Estado estado)
{

    return BaseDeDatos.OrdenesDeTrabajo.Where(o => o.Estado == estado).ToList();
}

0 referencias
protected void btnPendientes_Click(object sender, EventArgs e)
{
    gvOrdenesPendientes.Visible = true;
    gvOrdenesEnProgreso.Visible = false;
    gvOrdenesCompletadas.Visible = false;
}

0 referencias
protected void btnEnProgreso_Click(object sender, EventArgs e)
{
    gvOrdenesEnProgreso.Visible = true;
    gvOrdenesPendientes.Visible = false;
    gvOrdenesCompletadas.Visible = false;
}

0 referencias
protected void btnCompletadas_Click(object sender, EventArgs e)
{
    gvOrdenesEnProgreso.Visible = false;
    gvOrdenesPendientes.Visible = false;
    gvOrdenesCompletadas.Visible = true;
}
```

En última instancia tenemos la interfaz del administrador donde podemos apreciar la homogeneidad del proyecto en progreso. Este está conformado por tres columnas con tablas (GridView) las cuales reciben datos filtrados dependiendo del estado de una orden.

3.2 Pagina Clientes (Accesible para el administrador)

Incluye un formulario de entrada con validaciones para capturar datos como nombre, apellido, cédula de identidad, dirección, teléfono y correo electrónico. Las validaciones aseguran que los campos obligatorios sean completados y que el correo electrónico tenga un formato válido. Los datos ingresados pueden procesarse mediante el botón "Crear Cliente".
Además, la página presenta una GridView para mostrar, editar y eliminar clientes en una lista. Cada fila de la lista contiene datos organizados en columnas correspondientes a los campos mencionados. También se integran eventos como edición, actualización, eliminación y cancelación de edición para interactuar con los registros de clientes, gestionando el estado del sistema dinámicamente.
Finalmente, los estilos y clases CSS aplican un diseño moderno y profesional, utilizando un esquema de colores oscuros con elementos claros para resaltar contenido.

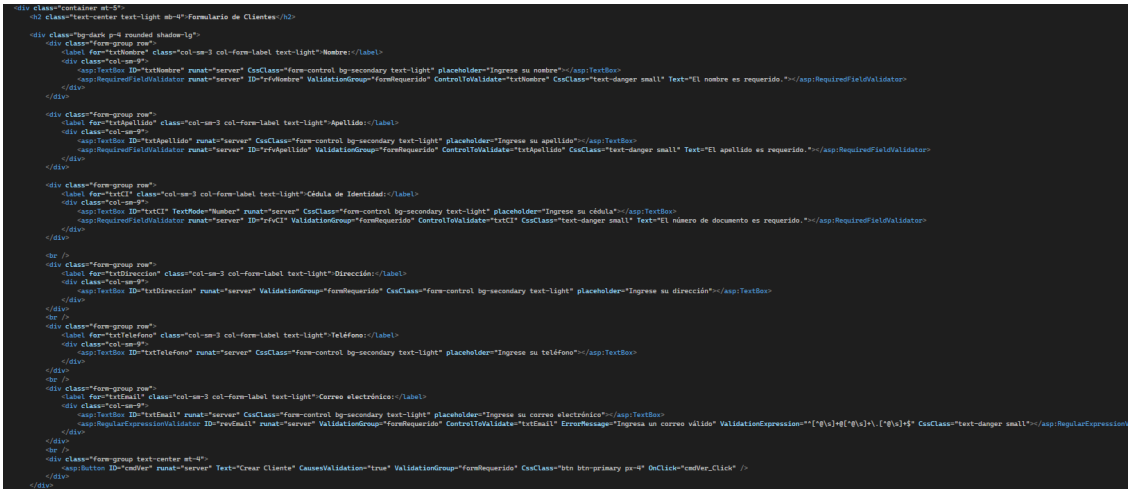
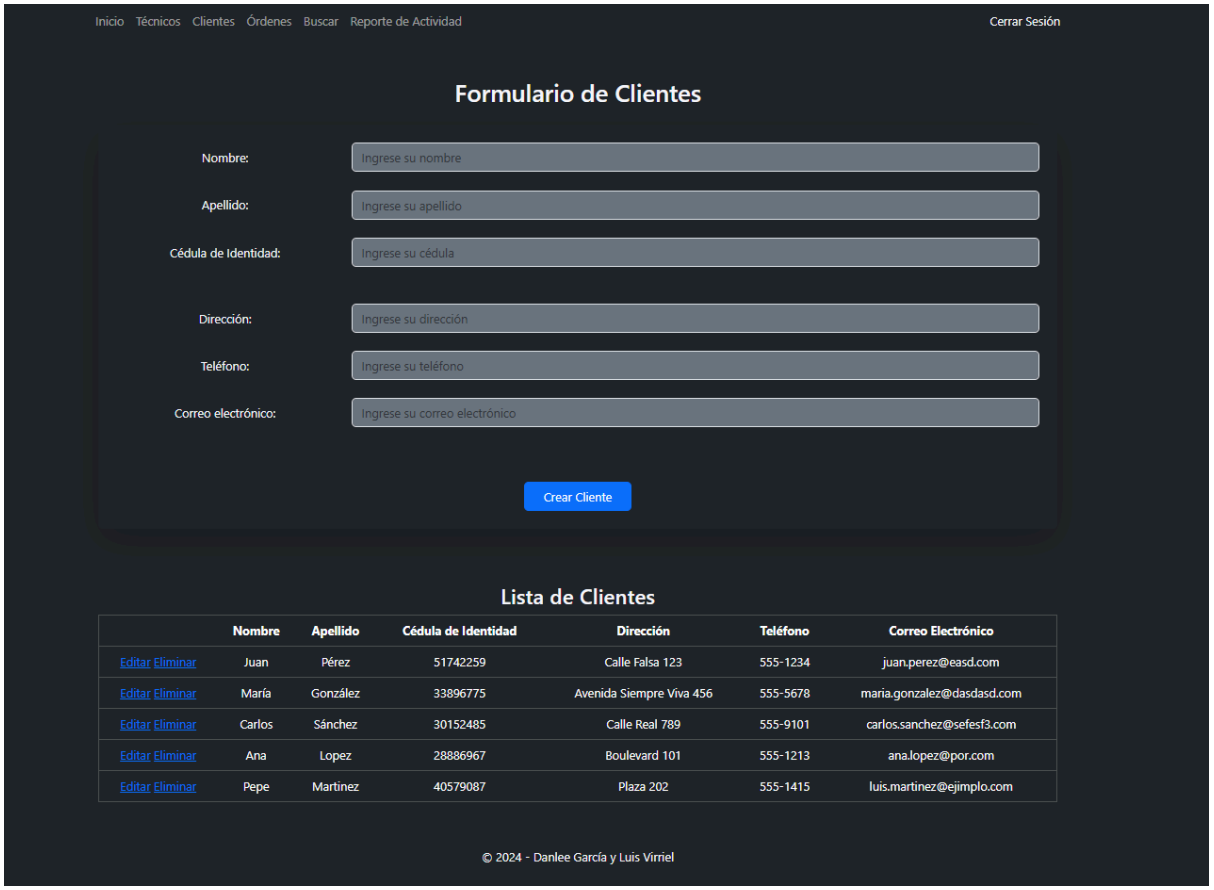


Figura 2.3.2.

```
    <h3 class="text-center text-light mt-5">Lista de Clientes</h3>
    <div class="table-responsive">
        <asp:GridView ID="gvClientes" runat="server" CssClass="table table-dark table-hover text-center" AutoGenerateColumns="False" DataKeyNames="CI"
            AutoGenerateEditButton="True" AutoGenerateDeleteButton="True"
            OnRowEditing="EditarClientes" OnRowUpdating="ActualizarClientes"
            OnRowDeleting="BorrarClientes" OnRowCancelingEdit="CanceloEditarClientes">
            <Columns>
                <asp:BoundField DataField="Nombre" HeaderText="Nombre" />
                <asp:BoundField DataField="Apellido" HeaderText="Apellido" />
                <asp:BoundField DataField="CI" HeaderText="Cédula de Identidad" />
                <asp:BoundField DataField="Direccion" HeaderText="Dirección" />
                <asp:BoundField DataField="Telefono" HeaderText="Teléfono" />
                <asp:BoundField DataField="Email" HeaderText="Correo Electrónico" />
            </Columns>
        </asp:GridView>
    </div>

    <asp:Label ID="lblError" runat="server" Visible="false" CssClass="text-danger"></asp:Label>
</div>
```

Figura 2.3.2

Como resultado nos queda algo bastante estético , con una versión oscura como preferencia. Por una parte el formulario que le permite al administrador ingresar un cliente cómodamente con todas las validaciones. Por otro lado una Grid(**grilla**) que muestra una tabla que muestra en sus categorías los atributos de la clase Cliente que anteriormente mostramos. Además de una columna extra que tiene la finalidad de poder eliminar y editar con sus respectivos botones.



3.3 Pagina Tecnicos (Accesible para el administrador)

El formulario principal permite a los administradores ingresar información sobre los técnicos, incluyendo su nombre, apellido, cédula de identidad y especialidad, con validaciones requeridas para cada campo utilizando los controles RequiredFieldValidator de ASP.NET. La especialidad se selecciona a través de un DropDownList que incluye opciones como reparación de electrodomésticos, informática, mecánica, electricidad, plomería y construcción.

Una vez que el formulario es completado, los administradores pueden hacer clic en el botón "Crear Técnico" para agregar la información del técnico al sistema. Además, se incluye una tabla (GridView) que muestra una lista de técnicos registrados, con la capacidad de editar, actualizar o eliminar registros existentes mediante botones de acción en cada fila de la tabla. Los técnicos pueden ser editados mediante un DropDownList que permite cambiar la especialidad, y los cambios se validan y se actualizan en la base de datos. En resumen, este código facilita la gestión de técnicos, con funcionalidades de creación, visualización, edición y eliminación de registros.

```
<h3 class="text-center text-light mt-5">Lista de Técnicos</h3>
<div class="table-responsive">
    <asp:GridView ID="gvTecnicos" runat="server" CssClass="table table-dark table-hover text-center" AutoGenerateColumns="False" DataKeyNames="CI"
        OnRowEditing="EditarTecnicos" OnRowUpdating="ActualizarTecnicos" OnRowDeleting="BorrarTecnicos" OnRowCancelingEdit="CanceloEditarTecnicos">
        <Columns>
            <asp:BoundField DataField="CI" HeaderText="Cédula de Identidad" />
            <asp:BoundField DataField="Nombre" HeaderText="Nombre" />
            <asp:BoundField DataField="Apellido" HeaderText="Apellido" />
            <asp:TemplateField HeaderText="Especialidad">
                <ItemTemplate>
                    <%# Eval("Especialidad") %>
                </ItemTemplate>
                <EditItemTemplate>
                    <asp:DropDownList ID="ddlEspecialidad" runat="server" CssClass="form-select bg-secondary text-light"
                        SelectedValue='<%# Eval("Especialidad") %>'>
                        <asp:ListItem Text="ReparacionElectrodomesticos" Value="ReparacionElectrodomesticos"></asp:ListItem>
                        <asp:ListItem Text="Informatica" Value="Informatica"></asp:ListItem>
                        <asp:ListItem Text="Mecanica" Value="Mecanica"></asp:ListItem>
                        <asp:ListItem Text="Electricidad" Value="Electricidad"></asp:ListItem>
                        <asp:ListItem Text="Plomeria" Value="Plomeria"></asp:ListItem>
                        <asp:ListItem Text="Construccion" Value="Construccion"></asp:ListItem>
                    </asp:DropDownList>
                </EditItemTemplate>
            </asp:TemplateField>
            <asp:CommandField ShowEditButton="True" ShowDeleteButton="True" />
        </Columns>
    </asp:GridView>
</div>
```

```
<div class="container mt-5">
    <h2 class="text-center text-light mb-4">Formulario Técnicos</h2>

    <div class="bg-dark p-4 rounded shadow-lg">
        <div class="form-group row">
            <label for="txtNombre" class="col-sm-3 col-form-label text-light">Nombre:</label>
            <div class="col-sm-9">
                <asp:TextBox ID="txtNombre" runat="server" CssClass="form-control bg-secondary text-light" placeholder="Ingrese su nombre"></asp:TextBox>
                <asp:RequiredFieldValidator runat="server" ID="RequiredFieldValidator2" ValidationGroup="formRequerido" ControlToValidate="txtNombre" CssClass="text-danger small" Text="El nombre es requerido."></asp:RequiredFieldValidator>
            </div>
        </div>

        <div class="form-group row">
            <label for="txtApellido" class="col-sm-3 col-form-label text-light">Apellido:</label>
            <div class="col-sm-9">
                <asp:TextBox ID="txtApellido" runat="server" CssClass="form-control bg-secondary text-light" placeholder="Ingrese su apellido"></asp:TextBox>
                <asp:RequiredFieldValidator runat="server" ID="RequiredFieldValidator3" ValidationGroup="formRequerido" ControlToValidate="txtApellido" CssClass="text-danger small" Text="El apellido es requerido."></asp:RequiredFieldValidator>
            </div>
        </div>

        <div class="form-group row">
            <label for="txtCI" class="col-sm-3 col-form-label text-light">Cédula de Identidad:</label>
            <div class="col-sm-9">
                <asp:TextBox ID="txtCI" TextMode="Number" runat="server" CssClass="form-control bg-secondary text-light" placeholder="Ingrese su cédula"></asp:TextBox>
                <asp:RequiredFieldValidator runat="server" ID="RequiredFieldValidator4" ValidationGroup="formRequerido" ControlToValidate="txtCI" CssClass="text-danger small" Text="El número de documento es requerido."></asp:RequiredFieldValidator>
            </div>
        </div>

        <div class="form-group row">
            <label for="txtEspecialidad" class="col-sm-3 col-form-label text-light">Especialidad:</label>
            <div class="col-sm-9">
                <asp:DropDownList runat="server" ID="txtEspecialidad" class="form-select bg-secondary text-light">
                    <asp:ListItem runat="server" Selected="True">Seleccione la especialidad</asp:ListItem>
                    <asp:ListItem runat="server" Value="ReparacionElectrodomesticos">Reparación de Electrodomésticos</asp:ListItem>
                    <asp:ListItem runat="server" Value="Informatica">Informática</asp:ListItem>
                    <asp:ListItem runat="server" Value="Mecanica">Mecánica</asp:ListItem>
                    <asp:ListItem runat="server" Value="Electricidad">Electricidad</asp:ListItem>
                    <asp:ListItem runat="server" Value="Plomeria">Plomería</asp:ListItem>
                    <asp:ListItem runat="server" Value="Construccion">Construcción</asp:ListItem>
                </asp:DropDownList>
                <asp:RequiredFieldValidator runat="server" ID="RequiredFieldValidator5" ValidationGroup="formRequerido" ControlToValidate="txtEspecialidad" CssClass="text-danger small" Text="La especialidad es requerida."></asp:RequiredFieldValidator>
            </div>
        </div>
```

Explicación Detallada:

1. Método Page_Load:
    - Este método se ejecuta cuando la página es cargada. Si no es una solicitud posterior (es decir, es la primera carga), llama al método CargarTecnicos() para cargar los datos de los técnicos y mostrarlos en un GridView.
2. Método CargarTecnicos:
    - Asigna los datos de los técnicos a la propiedad DataSource del GridView (gvTecnicos) y luego los muestra utilizando DataBind().
3. Método convertirAespecialidad:
    - Convierte un texto (txtEspecialidad) a un valor del enum Especialidad mediante Enum.TryParse(). Esto permite trabajar con una especialidad seleccionada por el usuario de manera más estructurada.

```
private void CargarTecnicos()
{
    gvTecnicos.DataSource = BaseDeDatos.Tecnicos;
    gvTecnicos.DataBind();
}
1 referencia
private Especialidad convertirAespecialidad(string txtEspecialidad)
{

    Enum.TryParse(txtEspecialidad, true, out Especialidad especialidad);

    return especialidad;


}
```

4. Método cmdCrearTecnico_Click:
    - Al hacer clic en el botón "Crear Técnico", se validan los datos ingresados: nombre, apellido, cédula, etc. Si los datos son válidos, se crea un nuevo objeto Tecnico, se agrega a la base de datos (BaseDeDatos.Tecnicos) y se actualiza la vista con CargarTecnicos().
    - Si los datos son incorrectos, se muestran mensajes de error.

```
protected void cmdCrearTecnico_Click(object sender, EventArgs e)
{
    lblError.Visible = false;
    string nombre = txtNombre.Text.Trim();
    string apellido = txtApellido.Text.Trim();
    string cedula = txtCI.Text.Trim();

    string verificoNombreYapellido = @"^[a-zA-ZáéíóúÁÉÍÓÚñÑ\s]+$";
    Regex regex = new Regex(verificoNombreYapellido);

    if (string.IsNullOrEmpty(txtNombre.Text))
    {
        lblError.Text = "Debe agregar un nombre";
        lblError.Visible = true;
        return;
    }

    if (!regex.IsMatch(nombre))
    {
        lblError.Text = "Debe escribir un nombre valido.";
        lblError.Visible = true;
        return;
    }
    if (!regex.IsMatch(apellido))
    {
        lblError.Text = "Debe escribir un apellido valido.";
        lblError.Visible = true;
        return;
    }

    if (string.IsNullOrEmpty(txtApellido.Text))
    {
        lblError.Text = "Debe agregar un apellido";
        lblError.Visible = true;
        return;
    }
```

5. Método LimpiarCampos:
    - Limpia los campos del formulario (nombre, apellido, cédula, especialidad) después de crear un técnico.
6. Método btnActualizar_Click:
    - Muestra un mensaje de éxito cuando un técnico ha sido actualizado correctamente.
7. Métodos EditarTecnicos y CancelarEditarTecnicos:
    - Permiten editar los datos de un técnico desde el GridView. El primero habilita el modo de edición, y el segundo lo cancela.

```
protected void btnActualizar_Click(object sender, EventArgs e)
{

    lblError.Text = "Tecnico actualizado correctamente.";
    lblError.Visible = true;
}
0 referencias
protected void EditarTecnicos(object sender, GridViewEditEventArgs e)
{
    gvTecnicos.EditIndex = e.NewEditIndex;
    CargarTecnicos();
}


0 referencias
protected void CanceloEditarTecnicos(object sender, GridViewCancelEditEventArgs e)
{
    gvTecnicos.EditIndex = -1;
    gvTecnicos.DataSource = BaseDeDatos.Tecnicos;
    gvTecnicos.DataBind();
}
```

8. Método ActualizarTecnicos:
    - Actualiza la información del técnico seleccionado en el GridView después de realizar cambios. Si la especialidad seleccionada no es válida, muestra un error.

```
protected void ActualizarTecnicos(object sender, GridViewUpdateEventArgs e)
{
    int index = e.RowIndex;

    if (index >= 0 && index < BaseDeDatos.Tecnicos.Count)
    {
        GridViewRow row = gvTecnicos.Rows[index];
        Tecnico tecnico = BaseDeDatos.Tecnicos[index];

        tecnico.setNombre(((TextBox)row.Cells[1].Controls[0]).Text);
        tecnico.setApellido(((TextBox)row.Cells[2].Controls[0]).Text);
        tecnico.setCI(((TextBox)row.Cells[0].Controls[0]).Text);

        DropDownList ddlEspecialidad = (DropDownList)row.FindControl("ddlEspecialidad");
        if (ddlEspecialidad != null && Enum.TryParse(ddlEspecialidad.SelectedValue, out Especialidad especialidad))
        {
            tecnico.Especialidad = especialidad;
            lblError.Text = "Técnico actualizado correctamente.";
            lblError.CssClass = "text-success";
            lblError.Visible = true;
        }
        else
        {
            lblError.Text = "Especialidad seleccionada no válida.";
            lblError.CssClass = "text-danger";
            lblError.Visible = true;
            return;
        }

        gvTecnicos.EditIndex = -1;
        CargarTecnicos();
    }
}
```

9. Método BorrarTecnicos:
    - Elimina un técnico de la base de datos al hacer clic en el botón "Eliminar" del GridView.
10. Método EsCedulaUruguaya:
    - Valida si una cédula ingresada es válida según el formato y las reglas de validación de cédulas uruguayas. Este método utiliza coeficientes específicos para calcular un dígito verificador y comprobar si la cédula es correcta.

```
protected void BorrarTecnicos(object sender, GridViewDeleteEventArgs e)
{
    int index = e.RowIndex;
    if (index >= 0 && index < BaseDeDatos.Tecnicos.Count)
    {
        BaseDeDatos.Tecnicos.RemoveAt(index);
        CargarTecnicos();
    }
}
1 referencia
private bool EsCedulaUruguaya(string ci)
{

    ci = ci.Replace(".", "").Replace("-", "");
    if (ci.Length < 7 || ci.Length > 8) return false;

    int[] coeficientes = { 2, 9, 8, 7, 6, 3, 4 };
    int suma = 0;
    if (ci.Length == 7)
        ci = "0" + ci;
    for (int i = 0; i < 7; i++)
    {
        suma += (ci[i] - '0') * coeficientes[i];
    }
    int digitoVerificador = (10 - (suma % 10)) % 10;
    return digitoVerificador == (ci[7] - '0');
}
```

Funcionalidad General:

- Este código gestiona el registro, actualización y eliminación de técnicos en una aplicación web en ASP.NET, asegurándose de que los datos ingresados sean correctos y validados. Utiliza un GridView para mostrar los técnicos y permite interactuar con ellos (crear, editar, eliminar) de forma eficiente.

3.4 Página Ordenes

En primer lugar, se presenta un formulario para crear nuevas órdenes de trabajo. El formulario incluye tres campos principales: uno para seleccionar el cliente mediante un DropDownList (**ddlCliente**), otro para seleccionar el técnico asignado a la orden (**ddlTecnico**), y un campo de texto para describir el problema relacionado con la orden (txtDescripcion). Este último tiene una validación para asegurar que el campo no quede vacío. Al final del formulario, hay un botón que, al hacer clic, activa el proceso de creación de la orden (**btnCrearOrden**).
Luego, la página muestra una lista de órdenes de trabajo en un GridView (**gvOrdenes**). Este **GridView** está configurado para mostrar varios campos de la orden, como el número de orden, el nombre del cliente, el técnico asignado, la descripción del problema, y la fecha de creación. Además, incluye un campo para editar el estado de la orden, que es editable a través de un DropDownList dentro del GridView. También tiene un botón de edición para cada fila, lo que permite modificar la información de las órdenes existentes.
Finalmente, se incluyen etiquetas de error (**lblError** y **Label1**) que se muestran si se presenta algún problema con la creación o edición de órdenes.

```
<h2 class="text-light mb-4 text-center">Gestión de Órdenes de Trabajo</h2>

<div class="bg-dark p-4 rounded shadow-lg">
    <div class="form-group row">
        <label for="ddlCliente" class="col-sm-3 col-form-label text-light font-weight-bold">Cliente:</label>
        <div class="col-sm-9">
            <asp:DropDownList ID="ddlCliente" runat="server" CssClass="form-control bg-secondary text-light" Required="true"></asp:DropDownList>
        </div>
    </div>

    <div class="form-group row">
        <label for="ddlTecnico" class="col-sm-3 col-form-label text-light font-weight-bold">Técnico Asignado:</label>
        <div class="col-sm-9">
            <asp:DropDownList ID="ddlTecnico" runat="server" CssClass="form-control bg-secondary text-light" Required="true"></asp:DropDownList>
        </div>
    </div>

    <div class="form-group row">
        <label for="txtDescripcion" class="col-sm-3 col-form-label text-light font-weight-bold">Descripción del Problema:</label>
        <div class="col-sm-9">
            <asp:TextBox ID="txtDescripcion" runat="server" CssClass="form-control bg-secondary text-light" placeholder="Ingrese la descripción del problema"></asp:TextBox>
            <asp:RequiredFieldValidator runat="server" ID="rfvDescripcion" ValidationGroup="formRequerido" ControlToValidate="txtDescripcion" CssClass="text-danger small" Text="La descripción es requerida."></asp:RequiredFieldValidator>
        </div>
    </div>

    <div class="form-group row">
        <div class="col-sm-9 offset-sm-3 text-center">
            <asp:Button ID="btnCrearOrden" runat="server" Text="Crear Orden" CausesValidation="true" ValidationGroup="formRequerido" CssClass="btn btn-danger" OnClick="btnCrearOrden_Click" />
        </div>
    </div>
    <asp:Label ID="Label1" runat="server" Visible="false" CssClass="text-danger"></asp:Label>
</div>

<br />
```

```
    <h3 class="text-light text-center mb-4">Lista de Órdenes de Trabajo
        <asp:Label ID="lblNombre" runat="server"></asp:Label>
    </h3>
    <asp:GridView ID="gvOrdenes" runat="server" CssClass="table table-dark table-bordered table-striped text-center" AutoGenerateColumns="False" DataKeyNames="NumeroOrden"
        OnRowEditing="gvOrdenes_RowEditing" OnRowUpdating="gvOrdenes_RowUpdating" OnRowCancelingEdit="gvOrdenes_RowCancelingEdit">
        <Columns>
            <asp:BoundField DataField="NumeroOrden" HeaderText="Número de Orden" />
            <asp:BoundField DataField="ClienteAsociado.Nombre" HeaderText="Cliente" />
            <asp:BoundField DataField="TecnicoAsignado.Nombre" HeaderText="Técnico Asignado" />
            <asp:BoundField DataField="DescripcionProblema" HeaderText="Descripción" />
            <asp:BoundField DataField="FechaCreacion" HeaderText="Fecha de Creación" DataFormatString="{0:dd/MM/yyyy}" />
            <asp:TemplateField HeaderText="Comentarios">
                <ItemTemplate>
                    <%# string.Join("<br />", ((List<ProyectoProgramacion2.Models.Comentario>)Eval("Comentarios")).Select(c => c.Texto)) %>
                </ItemTemplate>
            </asp:TemplateField>
            <asp:TemplateField HeaderText="Estado">
                <ItemTemplate>
                    <%# Eval("Estado") %>
                </ItemTemplate>
                <EditItemTemplate>
                    <asp:DropDownList ID="ddlEstado" runat="server" CssClass="form-select bg-secondary text-light" SelectedValue='<%# Eval("Estado") %>'>
                        <asp:ListItem Text="Pendiente" Value="Pendiente"></asp:ListItem>
                        <asp:ListItem Text="EnProgreso" Value="EnProgreso"></asp:ListItem>
                        <asp:ListItem Text="Completada" Value="Completada"></asp:ListItem>
                    </asp:DropDownList>
                </EditItemTemplate>
            </asp:TemplateField>
            <asp:CommandField ShowEditButton="True" />
        </Columns>
    </asp:GridView>

    <asp:Label ID="lblError" runat="server" Visible="false" CssClass="text-danger"></asp:Label>
</div>
```

Page_Load: Este método se ejecuta cuando la página se carga. La condición if (!IsPostBack) asegura que las acciones de carga solo se realicen una vez, es decir, solo en la primera carga de la página, no cuando se envíe el formulario o se recargue la página.

CargarClientes, CargarTecnicos, y CargarDatos: Son métodos que se llaman en la primera carga para llenar los controles DropDownList con los datos de los clientes, técnicos y órdenes de trabajo.

```
0 referencias
protected void Page_Load(object sender, EventArgs e)
{
    if (!IsPostBack)
    {
        CargarClientes();
        CargarTecnicos();
        CargarDatos();
    }
}
```

CargarClientes: Carga los datos de los clientes en un DropDownList (ddlCliente). Se configura el campo que se mostrará (Nombre) y el valor (CI).

CargarTecnicos: Similar al anterior, carga los técnicos en el DropDownList (ddlTecnico).

CargarDatos: Este método carga las órdenes de trabajo en un GridView (gvOrdenes) utilizando los datos disponibles en BaseDeDatos.OrdenesDeTrabajo.

```
1 referencia
private void CargarClientes()
{
    ddlCliente.DataSource = BaseDeDatos.Clientes;
    ddlCliente.DataTextField = "Nombre";
    ddlCliente.DataValueField = "CI";
    ddlCliente.DataBind();
    ddlCliente.Items.Insert(0, new ListItem("Seleccione un cliente", ""));
}

1 referencia
private void CargarTecnicos()
{
    ddlTecnico.DataSource = BaseDeDatos.Tecnicos;
    ddlTecnico.DataTextField = "Nombre";
    ddlTecnico.DataValueField = "CI";
    ddlTecnico.DataBind();
    ddlTecnico.Items.Insert(0, new ListItem("Seleccione un técnico", ""));
}

5 referencias
public void CargarDatos()
{
    gvOrdenes.DataSource = BaseDeDatos.OrdenesDeTrabajo;
    gvOrdenes.DataBind();
}
```

btnCrearOrden_Click: Este método se ejecuta cuando se hace clic en el botón para crear una nueva orden.
Realiza varias validaciones, como verificar que se haya ingresado una descripción, que se haya seleccionado un cliente y un técnico, y que ambos existan en la base de datos.
Si todo es correcto, genera un nuevo número de orden, crea una nueva OrdenTrabajo y la agrega a la lista de órdenes. Luego, recarga los datos en el GridView y limpia los campos del formulario.

```
0 referencias
protected void btnCrearOrden_Click(object sender, EventArgs e)
{
    lblError.Visible = false;
    if (string.IsNullOrEmpty(txtDescripcion.Text))
    {
        lblError.Text = "La descripción es requerida.";
        lblError.Visible = true;
        return;
    }

    if (ddlCliente.SelectedIndex == 0 || ddlTecnico.SelectedIndex == 0)
    {
        lblError.Text = "Debe seleccionar un cliente y un técnico.";
        lblError.Visible = true;
        return;
    }

    Cliente clienteSeleccionado = BaseDeDatos.Clientes.FirstOrDefault(c => c.CI == ddlCliente.SelectedValue);
    Tecnico tecnicoSeleccionado = BaseDeDatos.Tecnicos.FirstOrDefault(t => t.CI == ddlTecnico.SelectedValue);

    if (clienteSeleccionado == null || tecnicoSeleccionado == null)
    {
        lblError.Text = "Error en la selección de cliente o técnico.";
        lblError.Visible = true;
        return;
    }
    int nuevoNumeroOrden = BaseDeDatos.GenerarNumeroOrden();
    OrdenTrabajo nuevaOrden = new OrdenTrabajo(
        nuevoNumeroOrden,
        clienteSeleccionado,
        tecnicoSeleccionado,
        txtDescripcion.Text,
        Estado.Pendiente
    );
    BaseDeDatos.OrdenesDeTrabajo.Add(nuevaOrden);
    CargarDatos();
    ddlCliente.SelectedIndex = 0;
    ddlTecnico.SelectedIndex = 0;
    txtDescripcion.Text = "";
    lblError.Visible = false;
}
```

gvOrdenes_RowUpdating: Este método se ejecuta cuando se actualiza una orden en el GridView. Actualiza el estado de la orden seleccionada.

```
0 referencias
protected void gvOrdenes_RowUpdating(object sender, GridViewUpdateEventArgs e)
{
    lblError.Visible = false;

    int numeroOrden = (int)gvOrdenes.DataKeys[e.RowIndex].Value;
    var orden = BaseDeDatos.OrdenesDeTrabajo.FirstOrDefault(o => o.NumeroOrden == numeroOrden);
    if (orden != null)
    {
        var ddlEstado = (DropDownList)gvOrdenes.Rows[e.RowIndex].FindControl("ddlEstado");
        if (ddlEstado != null)
        {

            if (Enum.TryParse<Estado>(ddlEstado.SelectedValue, out var nuevoEstado))
            {

                orden.Estado = nuevoEstado;
                if (nuevoEstado == Estado.Completada)
                {
                    orden.FechaCompletada = DateTime.Now;
                }
            }
            else
            {
                lblError.Text = "El estado seleccionado no es válido.";
                lblError.Visible = true;
                return;
            }
        }
        gvOrdenes.EditIndex = -1;
        CargarDatos();
    }
}
```

gvOrdenes_RowEditing: Permite que una fila del GridView entre en modo de edición.

gvOrdenes_RowCancelingEdit: Cancela el modo de edición de una fila del GridView.

```
protected void gvOrdenes_RowEditing(object sender, GridViewEditEventArgs e)
{

    gvOrdenes.EditIndex = e.NewEditIndex;

    CargarDatos();
}

0 referencias
protected void gvOrdenes_RowCancelingEdit(object sender, GridViewCancelEditEventArgs e)
{
    gvOrdenes.EditIndex = -1;
    CargarDatos();
}
```

En varios métodos se utiliza el control lblError para mostrar mensajes de error en caso de que no se cumplan ciertos requisitos (como la selección de un cliente y un técnico, o un estado inválido).
En resumen, este código gestiona la creación, edición y actualización de órdenes de trabajo, permitiendo la asignación de clientes y técnicos, la validación de los datos ingresados y la actualización de los estados de las órdenes en una interfaz de usuario basada en formularios y GridView.

Como resultado del código tenemos una interfaz de usuario bastante intuitiva y práctica para el usuario. La cual está compuesta por dos contenedores , el primero se encuentra un formulario el cual le permite al administrador agregar una orden de trabajo a un técnico que el igual que un cliente se encuentran cargados en las DropdownList. En el segundo contenedor podemos visualizar la GridView con los datos cargados , esta posee dos funciones especiales , tales como , editar y eliminar una orden.

Figura

3.5. Página “PaginaTécnicos” (Accesible solo por el técnico registrado)

Esta página es específica para cada cliente que se encuentra registrado e inicia una secion.Se encuentra su información de ordenes de trabajo filtrada exclusivamente para uso propio, tiene permitido editar el estado de la orden.
El control principal es un GridView, que organiza y presenta datos de las órdenes en una tabla con columnas como número de orden, cliente, descripción, fecha de creación (formateada) y estado. El estado se puede editar a través de un menú desplegable con opciones como "Pendiente", "En Progreso" y "Completada". Además, cada fila incluye botones para agregar o mostrar comentarios y una opción para editar directamente. Los comentarios se gestionan mediante un modal de Bootstrap que se activa al pulsar el botón correspondiente. Este modal permite al usuario escribir un comentario en un área de texto y confirmar o cancelar la acción a través de botones.

```aspx
<div class="text-center mb-4">
    <h1 class="text-light">
        Bienvenido, <asp:Literal ID="litNombreTecnico" runat="server" />
    </h1>
    <h1><asp:Label ID="lblNoOrdenes" Visible="False" CssClass="text-danger" Text="No tienes ordenes pendientes" runat="server" /></h1>

</div>

<asp:GridView ID="gvOrdenes" runat="server" AutoGenerateColumns="False"
    OnRowCommand="gvOrdenes_RowCommand" DataKeyNames="NumeroOrden" EmptyDataText="No hay órdenes disponibles."
    CssClass="table table-dark">
    <Columns>
        <asp:BoundField DataField="NumeroOrden" HeaderText="Número de Orden" SortExpression="NumeroOrden" HeaderStyle-CssClass="text-center" />
        <asp:BoundField DataField="DescripcionProblema" HeaderText="Descripción" SortExpression="Descripcion" />
        <asp:BoundField DataField="Estado" HeaderText="Estado" HeaderStyle-CssClass="text-center" />
        <asp:TemplateField>
            <ItemTemplate>
                <asp:Button ID="btnAbrirComentario" runat="server" CommandName="AbrirComentario" Text="Abrir Comentarios"
                    CommandArgument='<%# Eval("NumeroOrden") %>' CssClass="btn btn-info" />
            </ItemTemplate>
        </asp:TemplateField>
    </Columns>
</asp:GridView>

<asp:Panel ID="comentariosSection" runat="server" CssClass="card bg-secondary text-light mt-4" Visible="false">
    <div class="card-body">
        <asp:Repeater ID="rptComentarios" runat="server" OnItemCommand="rptComentarios_ItemCommand">
            <ItemTemplate>
                <p>
                    <%# Eval("Texto") %> - <%# Eval("Fecha", "{0:dd/MM/yyyy HH:mm}") %>
                    <asp:Button ID="btnEditar" runat="server" CommandName="Editar" CommandArgument='<%# Eval("ComentarioID") %>'
                        Text="Editar" CssClass="btn btn-secondary btn-sm" />
                    <asp:Button ID="btnEliminar" runat="server" CommandName="Eliminar" CommandArgument='<%# Eval("ComentarioID") %>'
                        Text="Eliminar" CssClass="btn btn-danger btn-sm" />
                </p>
            </ItemTemplate>
        </asp:Repeater>

        <asp:TextBox ID="txtComentario" runat="server" CssClass="form-control bg-dark text-light border-secondary" Placeholder="Escribe un comentario"></asp:TextBox>
        <asp:Button ID="btnGuardarComentario" runat="server" Text="Guardar Comentario" CssClass="btn btn-primary mt-2" OnClick="btnGuardarComentario_Click" />
    </div>
</asp:Panel>

<asp:Panel ID="PanelSalir" runat="server" class="mt-5 text-center">
    <asp:LinkButton ID="LinkButton1" runat="server" CssClass="btn btn-danger" OnClick="btnSalir_Click">Cerrar Sesión</asp:LinkButton>
</asp:Panel>
```

Figura

- Método Page_Load:
    - Este método se ejecuta al cargar la página.
    - Si la página se carga por primera vez (!IsPostBack), llama al método CargarDatos para inicializar los datos del técnico y sus órdenes.
- Botón btnSalir:
    - Al hacer clic en el botón, la sesión del usuario se termina eliminando la variable Session["VariableUsuario"].
    - Redirige al usuario a la página de inicio de sesión (Login.aspx).
    -

```code
2 referencias
public partial class PaginaTecnicos : System.Web.UI.Page
{
    0 referencias
    protected void Page_Load(object sender, EventArgs e)
    {
        if (!IsPostBack)
        {
            CargarDatos();

        }
    }

    0 referencias
    protected void btnSalir_Click(object sender, EventArgs e)
    {
        Session["VariableUsuario"] = null;
        Response.Redirect("~/Login.aspx");
    }
```

- Método CargarDatos:
    - Obtiene el identificador del técnico almacenado en la sesión (Session["VariableUsuario"]). Si no existe, redirige al usuario al login.
    - Busca al técnico en la base de datos y muestra su nombre en la interfaz mediante el control litNombreTecnico.
    - Filtra las órdenes de trabajo asignadas al técnico, basándose en su identificador.
    - Si no hay órdenes, muestra un mensaje de "sin órdenes". Si existen, las muestra en un control GridView.

```
private void CargarDatos()
{
    string tecnicoCI = Session["VariableUsuario"]?.ToString();
    if (string.IsNullOrEmpty(tecnicoCI))
    {
        Response.Redirect("Login.aspx");
        return;
    }

    var tecnico = BaseDeDatos.Tecnicos.FirstOrDefault(t => t.CI == tecnicoCI);
    if (tecnico != null)
    {
        litNombreTecnico.Text =  tecnico.Nombre;
    }
    else
    {
        litNombreTecnico.Text = "";
    }

    var ordenes = BaseDeDatos.OrdenesDeTrabajo;


    var ordenesFiltradas = ordenes
        .Where(o => o.TecnicoAsignado?.CI == tecnicoCI)
        .ToList();

    if (ordenesFiltradas.Count == 0)
    {
        lblNoOrdenes.Visible = true;
    }
    else
    {
        lblNoOrdenes.Visible = false;
        gvOrdenes.DataSource = ordenesFiltradas;
        gvOrdenes.DataBind();
    }
}
```

- Manejo de eventos en GridView (gvOrdenes_RowCommand):
    - Este evento detecta interacciones en el GridView.
    - Si el comando es "AbrirComentario", obtiene el número de la orden seleccionada y llama al método AbrirComentarios para cargar sus comentarios.

- Manejo de eventos en Repeater (rptComentarios_ItemCommand):
    - Gestiona las acciones de eliminar o editar comentarios.

- Si es "Eliminar", remueve el comentario seleccionado de la lista asociada a la orden.
- Si es "Editar", carga el texto del comentario en un cuadro de texto y guarda el ID en la sesión para identificarlo posteriormente.

```
protected void gvOrdenes_RowCommand(object sender, GridViewCommandEventArgs e)
{
    if (e.CommandName == "AbrirComentario")
    {
        int numeroOrden = Convert.ToInt32(e.CommandArgument.ToString());
        AbrirComentarios(numeroOrden);
    }
}
0 referencias
protected void rptComentarios_ItemCommand(object source, RepeaterCommandEventArgs e)
{
    int numeroOrden = Convert.ToInt32(Session["NumeroOrden"]);
    var orden = BaseDeDatos.OrdenesDeTrabajo.FirstOrDefault(o => o.NumeroOrden == numeroOrden);

    if (orden != null)
    {
        int comentarioID = Convert.ToInt32(e.CommandArgument);

        if (e.CommandName == "Eliminar")
        {
            var comentario = orden.Comentarios.FirstOrDefault(c => c.ComentarioID == comentarioID);
            if (comentario != null)
            {
                orden.Comentarios.Remove(comentario);
                AbrirComentarios(numeroOrden);
            }
        }
        else if (e.CommandName == "Editar")
        {
            var comentario = orden.Comentarios.FirstOrDefault(c => c.ComentarioID == comentarioID);
            if (comentario != null)
            {
                txtComentario.Text = comentario.Texto;
                Session["ComentarioID"] = comentarioID;
            }
        }
    }
}
```

- Método AbrirComentarios:
  - Busca la orden seleccionada por su número.
  - Si existen comentarios asociados, los muestra en el control Repeater.
  - Si no hay comentarios, muestra una lista vacía.
  - Activa la sección de comentarios y guarda el número de la orden en la sesión.

- Botón btnGuardarComentario:
  - Permite agregar o editar comentarios asociados a una orden.
  - Si el texto del comentario está vacío, muestra una alerta al usuario.
  - Si se está editando, actualiza el texto del comentario correspondiente.
  - Si es un nuevo comentario, genera un ID único y lo añade a la lista de comentarios de la orden.
  - Limpia el área de texto y actualiza los comentarios y la lista de órdenes.

```
0 referencias
protected void btnGuardarComentario_Click(object sender, EventArgs e)
{
    string comentarioTexto = txtComentario.Text.Trim();

    if (string.IsNullOrWhiteSpace(comentarioTexto))
    {
        ClientScript.RegisterStartupScript(this.GetType(), "error", "alert('Comentario vacio.');", true);
        return;
    }

    int numeroOrden = Convert.ToInt32(Session["NumeroOrden"]);
    var orden = BaseDeDatos.OrdenesDeTrabajo.FirstOrDefault(o => o.NumeroOrden == numeroOrden);

    if (orden != null)
    {
        int comentarioID = Session["ComentarioID"] != null ? Convert.ToInt32(Session["ComentarioID"]) : 0;

        if (comentarioID > 0)
        {
            var comentario = orden.Comentarios.FirstOrDefault(c => c.ComentarioID == comentarioID);
            if (comentario != null)
            {
                comentario.Texto = comentarioTexto;
            }
            Session["ComentarioID"] = null;
        }
        else
        {
            comentarioID = orden.Comentarios.Count > 0 ? orden.Comentarios.Max(c => c.ComentarioID) + 1 : 1;
            orden.Comentarios.Add(new Comentario(comentarioID, comentarioTexto, DateTime.Now));
        }

        txtComentario.Text = string.Empty;
        AbrirComentarios(numeroOrden);
        CargarDatos();

    }
}
```

Este código organiza la lógica para cargar información del usuario, manejar sus órdenes asignadas y gestionar comentarios de forma intuitiva, empleando controles de servidor ASP.NET y estructuras de datos basadas en clases.

3.6 Pagina “Login” (Accesible tanto para quien tenga el URL de la página)

La página de inicio de sesión en ASP.NET, diseñada para ofrecer una interfaz centrada y moderna. La página utiliza un diseño de contenedor (login-container) con estilo CSS para centrar vertical y horizontalmente el contenido en la ventana. La estructura principal está dividida en dos secciones: un formulario de inicio de sesión que incluye campos para el usuario y la contraseña, además de un botón para enviar la información, y un área de imagen que muestra un logo decorativo. Se aplican estilos personalizados para crear una apariencia profesional, con un esquema de colores oscuros, bordes redondeados, y sombreado para el cuadro de inicio de sesión. La interacción del botón de inicio de sesión está vinculada a un evento btnLogin_Click, mientras que los mensajes de error se muestran mediante un control Label.

```
<div class="login-container">
    <div class="login-box">

        <div class="login-form">
            <h2 class="text-light text-center mb-4">Bienvenidos a DanLuis Solutions</h2>

            <div class="form-group">
                <label for="txtUsername" class="col-form-label text-light">Usuario:</label>
                <asp:TextBox ID="txtUsername" runat="server" CssClass="form-control bg-secondary text-light" placeholder="Ingrese su usuario" />
            </div>

            <div class="form-group">
                <label for="txtPassword" class="col-form-label text-light">Contraseña:</label>
                <asp:TextBox ID="txtPassword" runat="server" TextMode="Password" CssClass="form-control bg-secondary text-light" placeholder="Ingrese su contraseña" />
            </div>

            <div class="form-group text-center mt-4">
                <asp:Button ID="btnLogin" runat="server" Text="Iniciar Sesión" CssClass="btn btn-primary px-4" OnClick="btnLogin_Click" />
            </div>

            <div class="form-group text-center">
                <asp:Label ID="lblMessage" runat="server" ForeColor="Red" />
            </div>
        </div>

        <div class="login-logo">
            <div class="login-logo">
                <img src="../Imagenes/MITAD_BARBA.png" alt="Logo DanLuis Solutions" />
            </div>
        </div>
    </div>
</div>
```

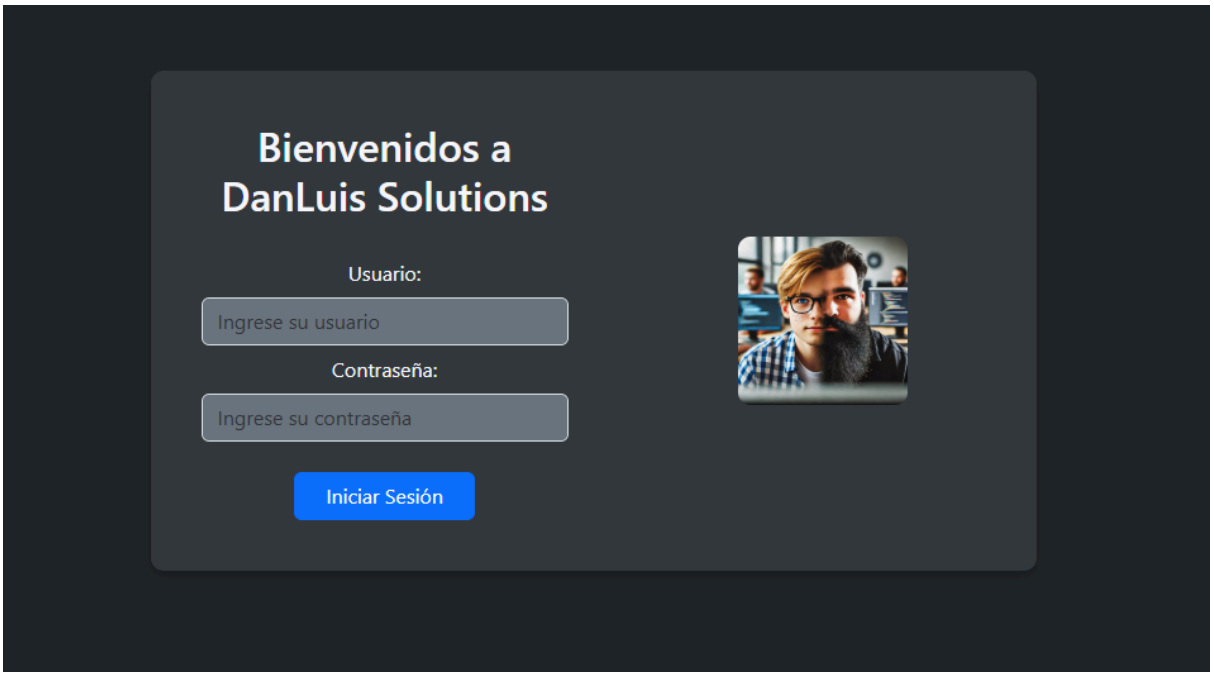
```
public partial class Login : Page
{
    0 referencias
    protected void Page_Load(object sender, EventArgs e)
    {

    }
    0 referencias
    protected void btnLogin_Click(object sender, EventArgs e)
    {
        string username = txtUsername.Text.Trim();
        string password = txtPassword.Text.Trim();

        if (username == "ADMIN" && password == "ADMIN")
        {
            Response.Redirect("Default.aspx");
        }
        else
        {
            var tecnico = Models.BaseDeDatos.Tecnicos
                .Find(t => t.CI.Equals(username, StringComparison.OrdinalIgnoreCase));
            if (tecnico != null && tecnico.CI == password)
            {
                Session["VariableUsuario"] = password;

                Response.Redirect("PaginaTecnicos.aspx");
            }
            else
            {
                lblMessage.Text = "Usuario o contraseña incorrectos.";
            }
        }
    }
}
```

Este código maneja la autenticación en una página de inicio de sesión en ASP.NET. Aquí está el flujo de la lógica:

1. Obtención de credenciales:
   Se obtienen el nombre de usuario y la contraseña de los controles de texto (txtUsername y txtPassword), eliminando los espacios extra.
2. Validación de administrador:
   Si el nombre de usuario y la contraseña son ambos "ADMIN", se redirige a Default.aspx.
3. Validación de técnicos:
   Si no es administrador, busca en la lista Técnicos de la base de datos (Models.BaseDeDatos) para encontrar un técnico cuyo CI coincida con el nombre de usuario. Si se encuentra y la contraseña es correcta, guarda la contraseña en la sesión y redirige a PaginaTecnicos.aspx.
4. Manejo de error:
   Si las credenciales son incorrectas, muestra un mensaje de error en lblMessage.

Este código distingue entre administradores y técnicos, redirigiendo a las páginas correspondientes si las credenciales son válidas. Sin embargo, carece de medidas de seguridad como el cifrado de contraseñas.

Finalmente podemos ver una interfaz de usuario estéticamente atractiva y con un eje de profesionalismo bastante acertado.



3.7 Página “ReporteDeActividad” (Accesible para el administrador)

En esta página se muestra un reporte de actividad de los técnicos. La estructura incluye:
Un título principal "Reporte de Actividad" centrado y con estilo, dentro de un contenedor con márgenes superiores.
Un panel con fondo oscuro que contiene otro título "Resumen por Técnico", también centrado.
Dentro del panel, hay una tabla responsive (GridView), configurada para no generar columnas automáticamente, con clases para darle un estilo de tabla oscura, bordeada y estriada.
La tabla tiene cuatro columnas: "Nombre Técnico", "Pendientes", "En Progreso" y "Completadas", cada una con su respectivo campo de datos (DataField) y posibilidad de ordenar por cada uno.
Este código está diseñado para mostrar, de manera estructurada, el resumen de actividades de los técnicos en una interfaz interactiva.

```aspx
<asp:Content ID="Content1" ContentPlaceHolderID="MainContent" runat="server">
    <div class="container mt-5">
        <h2 class="text-center text-light mb-4">Reporte de Actividad</h2>

        <div class="bg-dark p-4 rounded shadow-lg">

            <h3 class="text-center text-light mb-4">Resumen por Técnico</h3>
            <div class="table-responsive">
                <asp:GridView ID="gvResumenTecnico" runat="server" AutoGenerateColumns="False" CssClass="table table-dark table-bordered table-striped text-center">
                    <Columns>
                        <asp:BoundField DataField="NombreTecnico" HeaderText="Nombre Técnico" SortExpression="NombreTecnico" />
                        <asp:BoundField DataField="Pendientes" HeaderText="Pendientes" SortExpression="Pendientes" />
                        <asp:BoundField DataField="EnProgreso" HeaderText="En Progreso" SortExpression="EnProgreso" />
                        <asp:BoundField DataField="Completadas" HeaderText="Completadas" SortExpression="Completadas" />
                    </Columns>
                </asp:GridView>
            </div>
        </div>


        <div class="bg-dark p-4 rounded shadow-lg mt-5">
            <h3 class="text-center text-light mb-4">Órdenes Completadas</h3>
            <div class="table-responsive">
                <asp:GridView ID="gvOrdenesCompletadas" runat="server" AutoGenerateColumns="False" CssClass="table table-dark table-bordered table-striped text-center">
                    <Columns>
                        <asp:BoundField DataField="NumeroOrden" HeaderText="Número de Orden" />
                        <asp:BoundField DataField="ClienteNombre" HeaderText="Cliente" />
                        <asp:BoundField DataField="TecnicoNombre" HeaderText="Técnico" />
                        <asp:BoundField DataField="Descripcion" HeaderText="Descripción" />
                        <asp:BoundField DataField="FechaCompletada" HeaderText="Fecha Completada" DataFormatString="{0:dd/MM/yyyy}" />
                    </Columns>
                </asp:GridView>
            </div>
        </div>
    </div>
</asp:Content>
```

```cs
0 referencias
protected void Page_Load(object sender, EventArgs e)
{
    if (!IsPostBack)
    {
        CargarResumenTecnicos();
        CargarOrdenesCompletadas();
    }
}

1 referencia
private void CargarResumenTecnicos()
{
    var resumenTecnico = BaseDeDatos.Tecnicos.Select(t => new
    {
        NombreTecnico = $"{t.Nombre} {t.Apellido}",
        Pendientes = BaseDeDatos.OrdenesDeTrabajo.Count(o => o.TecnicoAsignado.CI == t.CI && o.Estado == Estado.Pendiente),
        EnProgreso = BaseDeDatos.OrdenesDeTrabajo.Count(o => o.TecnicoAsignado.CI == t.CI && o.Estado == Estado.EnProgreso),
        Completadas = BaseDeDatos.OrdenesDeTrabajo.Count(o => o.TecnicoAsignado.CI == t.CI && o.Estado == Estado.Completada)
    }).ToList();

    gvResumenTecnico.DataSource = resumenTecnico;
    gvResumenTecnico.DataBind();
}

1 referencia
private void CargarOrdenesCompletadas()
{
    var ordenesCompletadas = BaseDeDatos.OrdenesDeTrabajo
        .Where(o => o.Estado == Estado.Completada)
        .Select(o => new
        {
            NumeroOrden = o.NumeroOrden,
            ClienteNombre = $"{o.ClienteAsociado.Nombre} {o.ClienteAsociado.Apellido}",
            TecnicoNombre = $"{o.TecnicoAsignado.Nombre} {o.TecnicoAsignado.Apellido}",
            Descripcion = o.DescripcionProblema,
            FechaCompletada = o.FechaCompletada.HasValue ? o.FechaCompletada.Value.ToString("dd/MM/yyyy") : "N/A"
        })
        .ToList();

    gvOrdenesCompletadas.DataSource = ordenesCompletadas;
    gvOrdenesCompletadas.DataBind();
}
```

1. Page_Load:
    - Este es el método que se ejecuta cuando se carga la página. Si no es una recarga posterior a una solicitud (IsPostBack), invoca los métodos CargarResumenTecnicos y CargarOrdenesCompletadas para obtener y mostrar datos.
2. CargarResumenTecnicos:
    - Este método obtiene un resumen de las actividades de los técnicos, consultando la base de datos para contar las órdenes de trabajo en tres estados: Pendiente, En Progreso y Completada, para cada técnico.

- Se genera una lista de objetos anónimos con el nombre completo del técnico y los conteos de las órdenes en cada estado.
- Luego, la lista se asigna a un GridView (control de tabla) denominado gvResumenTecnico, y se llama a DataBind para actualizar la vista.

3. CargarOrdenesCompletadas:
   - Este método obtiene las órdenes de trabajo que están marcadas como "Completadas" en la base de datos.
   - Para cada orden completada, obtiene información relevante como el número de orden, el nombre del cliente y técnico, una descripción del problema y la fecha en que se completó la orden.
   - Similar al método anterior, los datos se asignan a un GridView llamado gvOrdenesCompletadas y luego se actualiza la vista con DataBind.

3.8 Pagina “BuscarOrdenes” (Accesible para el administrador)

Esta página te permite buscar órdenes de trabajo mediante un número de orden. En la interfaz, se presenta un campo de texto <asp:TextBox> donde el usuario ingresa el número de orden y un botón <asp:Button> que, al hacer clic, ejecuta el evento btnBuscar_Click para realizar la búsqueda. Los resultados se muestran en una tabla <asp:GridView> con columnas predefinidas como Número de Orden, Cliente, Técnico Asignado, Descripción, Fecha de Creación y Estado. Además, la propiedad AutoGenerateColumns="False" asegura que solo se muestran las columnas definidas. Si ocurre un error durante la búsqueda, se muestra un mensaje de error en un <asp:Label> con un estilo de texto rojo. El código crea una interfaz sencilla para buscar y visualizar las órdenes de trabajo de un sistema, mostrando los datos de manera organizada.

```
<asp:Content ID="Content1" ContentPlaceHolderID="MainContent" runat="server">
    <div class="container mt-5">
        <h2 class="text-center text-light mb-4">Buscar Orden de Trabajo</h2>

        <div class="bg-dark p-4 rounded shadow-lg">
            <div class="form-group row">
                <label for="txtNumeroOrden" class="col-sm-3 col-form-label text-light font-weight-bold">Número de Orden:</label>
                <div class="col-sm-9">
                    <asp:TextBox ID="txtNumeroOrden" runat="server" CssClass="form-control bg-secondary text-light" placeholder="Ingrese el número de orden"></asp:TextBox>
                </div>
            </div>

            <div class="form-group text-center mt-4">
                <asp:Button ID="btnBuscar" runat="server" Text="Buscar Orden" CssClass="btn btn-danger px-4" OnClick="btnBuscar_Click" />
            </div>
        </div>

        <br />

        <h3 class="text-center text-light mt-5">Resultados de la Búsqueda</h3>
        <div class="table-responsive">
            <asp:GridView ID="gvResultadoBusqueda" runat="server" CssClass="table table-dark table-hover text-center" AutoGenerateColumns="False" DataKeyNames="NumeroOrden">
                <Columns>
                    <asp:BoundField DataField="NumeroOrden" HeaderText="Número de Orden" />
                    <asp:BoundField DataField="ClienteAsociado.Nombre" HeaderText="Cliente" />
                    <asp:BoundField DataField="TecnicoAsignado.Nombre" HeaderText="Técnico Asignado" />
                    <asp:BoundField DataField="DescripcionProblema" HeaderText="Descripción" />
                    <asp:BoundField DataField="FechaCreacion" HeaderText="Fecha de Creación" DataFormatString="{0:dd/MM/yyyy}" />
                    <asp:BoundField DataField="Estado" HeaderText="Estado" />
                </Columns>
            </asp:GridView>
        </div>

        <asp:Label ID="lblError" runat="server" Visible="false" CssClass="text-danger"></asp:Label>
    </div>
</asp:Content>
```

```
public partial class BuscarOrden : System.Web.UI.Page
{
    0 referencias
    protected void Page_Load(object sender, EventArgs e)
    {
        if (!IsPostBack)
        {

            gvResultadoBusqueda.DataSource = BaseDeDatos.OrdenesDeTrabajo;
            gvResultadoBusqueda.DataBind();
        }
    }

    0 referencias
    protected void btnBuscar_Click(object sender, EventArgs e)
    {
        if (int.TryParse(txtNumeroOrden.Text.Trim(), out int numeroOrden))
        {
            var ordenEncontrada = BaseDeDatos.OrdenesDeTrabajo.FirstOrDefault(o => o.NumeroOrden == numeroOrden);

            if (ordenEncontrada != null)
            {
                gvResultadoBusqueda.DataSource = new[] { ordenEncontrada };
                gvResultadoBusqueda.DataBind();
            }
            else
            {
                gvResultadoBusqueda.DataSource = null;
                gvResultadoBusqueda.DataBind();

                lblError.Visible = true;
                lblError.Text = "Orden no encontrada.";
            }
        }
        else
        {
            lblError.Visible = true;
            lblError.Text = "Por favor, ingrese un número de orden válido.";

        }
    }
}
```

btnBuscar_Click: Este método maneja el clic en el botón de búsqueda. Primero, se intenta convertir el texto ingresado en el TextBox txtNumeroOrden a un número entero usando int.TryParse. Si la conversión es exitosa, se busca la orden de trabajo que coincida con el número de orden ingresado en la base de datos. Si la orden es encontrada, se muestra en el GridView. Si no se encuentra, se limpia el GridView y se muestra un mensaje de error diciendo "Orden no encontrada". Si el número de orden ingresado no es válido, también se muestra un mensaje de error pidiendo un número de orden válido.
El código maneja tanto la carga inicial de datos como la búsqueda de órdenes y la gestión de errores mediante etiquetas.

Por último obtenemos la interfaz para el administrador , con un diseño dark sencillo fácil de manipular.

4. Conclusiones

La conclusión final del proyecto se centra en resaltar los aspectos clave de la implementación de la aplicación web desarrollada para DanLuis Solutions. A lo largo del proceso, se aplicaron principios fundamentales de programación orientada a objetos, lo que permitió crear una estructura robusta y escalable, basada en clases, métodos y atributos bien definidos. Esto asegura una gestión eficiente y flexible de la información, desde la creación de clientes hasta la asignación de órdenes de trabajo y la gestión de técnicos.

El uso de listas Enum para personalizar el estado de las órdenes de trabajo muestra cómo el sistema no solo facilita la administración de datos, sino que también se adapta a las necesidades del negocio, permitiendo un control más preciso y detallado sobre las tareas asignadas. Esta característica es clave, ya que facilita la categorización de las órdenes y su seguimiento a lo largo del ciclo de vida de cada tarea.

La simulación de una base de datos interna a través de la clase Base De Datos permite la creación de un entorno controlado en el que se almacenan las entidades principales del sistema sin la necesidad de depender de un servidor de base de datos real, lo que hace que el proyecto sea fácilmente gestionable y portable. Esta implementación es útil para fines de prueba y desarrollo, aunque se contempla que en un entorno real, una base de datos externa o en la nube sería la solución adecuada para escalar y asegurar la persistencia de los datos a largo plazo.

Por último, la interfaz de usuario se diseñó con una visión clara de usabilidad, empleando tecnologías como ASP.NET y C# para crear una experiencia sencilla e intuitiva. El sistema permite a los administradores gestionar eficientemente todas las operaciones relacionadas con clientes, técnicos y órdenes de trabajo, optimizando los procesos internos y mejorando la productividad de la empresa. Con estos avances, el proyecto no solo cumple con los requisitos iniciales, sino que sienta las bases para futuras expansiones y mejoras, convirtiéndose en una herramienta valiosa para la gestión de tareas en empresas del sector.

5. Referencias

Librerías BootsTrap: https://getbootstrap.com/
Microsoft Learn : https://learn.microsoft.com/es-es/
Stack OverFlow : https://stackoverflow.com/