



“Al mal tiempo, buena cara”

Obligatorio Programación III

Universidad ORT Centro CTC CEI

Carrera Analista Programador

Programación III

Prof. Martín Alvarez & Marciano

Facundo Cardoso, Santiago Gutierrez y Tobías Nuñez

Maldonado, 8 de julio de 2024



**ANALISTA
PROGRAMADOR**



Resumen

"Al mal tiempo, buena cara" es un sitio web para una cadena de restaurantes que ofrece una experiencia culinaria excepcional con ingredientes frescos y locales. El sitio web incluye varias secciones como una página principal, menú de clientes, reseñas, restaurantes, mesas, órdenes, pagos, reservas, usuarios y roles. La página principal está diseñada para brindar una experiencia atractiva y amigable para los usuarios, mostrando la identidad del restaurante y destacando sus principales características y servicios.

El proyecto desarrolla una aplicación web para la gestión integral del restaurante "Al mal tiempo, Buena Cara". La implementación se llevó a cabo utilizando SQL Server Management, C#, ASP.NET MVC, y tecnologías web como Bootstrap, HTML y CSS. Al acceder a la página web, los usuarios se encuentran con una pestaña de Login, donde ingresan sus datos para acceder. Una vez dentro, pueden acceder a la sección de "Reservas". Aquí, cada usuario puede gestionar sus reservas, incluyendo la asignación de mesas y la visualización de disponibilidad.

En la parte izquierda de la página, una barra de navegación facilita el acceso a diferentes secciones de la web, como "Clientes", "Menús", "Pagos", "Reseñas" y "Clima".

Índice:

Resumen.....	2
1. Introducción.....	5
1.1 ¿Qué es un I.D.E?.....	5
1.2 ¿Qué es C#?.....	5
1.3 Atributos y Métodos en Programación.....	5
1.4 Bootstrap.....	5
1.5 CSS.....	6
1.6 HTML.....	6
1.7 Herencia.....	6
1.8 Clases.....	7
1.9 SQL Server Management.....	8
1.10 API Currencylayer de Conversión de Monedas.....	8
1.11 API de Clima OpenWeatherMap.....	9
1.12 Patrón Modelo Vista Controlador.....	9
1.13 CRUD (Create, Read, Update, Delete).....	10
1.14 JSON (JavaScript Object Notation).....	10
1.15 Entity Framework.....	11
2. Desarrollo:.....	11
2.1 Arquitectura y Tecnologías Utilizadas.....	12
2.2 Patrón Modelo-Vista-Controlador (MVC).....	12
2.3 Estructura del Proyecto.....	13
2.2 Clases del Programa.....	16
Clase BaseDeDatos.....	16
Clase Cliente:.....	24
Clase Restaurante:.....	24
Clase Reseña:.....	25
Clase Pago:.....	26
Clase Reserva:.....	28
Clase Mesa:.....	29
Clase Menu:.....	29
Clase Usuario.....	31
Clases Adicionales:.....	32
Clase Rol.....	32
Clase OrdenDetalle.....	33
Clase Welcome.....	34
Clase Clima.....	36
Clase ErrorViewModel.....	37
Clase Ordene.....	38
Clase Rol.....	39
2.3 Estructura del Proyecto.....	40
Página Principal (Home).....	40
Menú de Cliente.....	40
Barra Lateral de Navegación (Sidebar).....	40

Sección de Reservas.....	40
Sección de Clientes.....	40
Sección de Pagos.....	40
Sección de Reseñas.....	40
Sección de Usuarios.....	41
Sección de Roles.....	41
3.0 Funcionalidades del Programa.....	43
4.0 Futuras Implementaciones.....	44
4.1 Mejora del Sistema de Reservas:.....	44
4.2 Aplicación Móvil Complementaria:.....	44
4.3 Soporte Multilenguaje:.....	44
Iniciar Sesión.....	45
Realizar una Reserva.....	46
Consultar el Menú.....	47
Dejar una Reseña.....	48
Gestionar Usuarios (Solo Administradores).....	49
Asignar Roles a Usuarios (Solo Administradores).....	51
6.0 Diagramas:.....	53
Diagrama UML:.....	53
Diagrama Secuencial:.....	54
7.0 Referencias:.....	55
8.0 Conclusión:.....	56

1. Introducción

1.1 ¿Qué es un I.D.E?

Un entorno de desarrollo integrado o IDE (Integrated Development Environment) es un espacio de trabajo virtual que se utiliza para el desarrollo y programación de aplicaciones de software. Las características del IDE son diversas y suelen incluir la capacidad multiplataforma, soporte para diferentes lenguajes de programación, integración con sistemas de control de versiones, reconocimiento de sintaxis, capacidad de depuración de código, entre otros. Puede utilizar cualquier editor de texto para escribir código. Sin embargo, la mayoría de los entornos de desarrollo integrado incluyen funcionalidades que van más allá de la edición de texto. Proporcionan una interfaz central para herramientas de desarrollo comunes, lo que hace que el proceso de desarrollo de software sea mucho más eficiente. Los desarrolladores pueden comenzar a programar aplicaciones nuevas rápidamente en lugar de integrar y configurar diferentes software de forma manual.[1]

1.2 ¿Qué es C#?

C# (léase C Sharp), es una evolución que Microsoft realizó de este lenguaje, tomando lo mejor de los lenguajes C y C++, y ha continuado añadiéndole funcionalidades, tomando de otros lenguajes, como java, algo de su sintaxis evolucionada. Lo orientó a objetos para toda su plataforma NET (tanto Framework como Core), y con el tiempo adaptó las facilidades de la creación de código que tenía otro de sus lenguajes más populares, Visual Basic, haciéndolo tan polivalente y fácil de aprender como éste, sin perder ni un ápice de la potencia original de C. En la versión de .NET Core, se ha reconstruido por completo su compilador, haciendo las aplicaciones un 600% más rápidas.[2]

1.3 Atributos y Métodos en Programación

Los atributos son variables que almacenan información sobre un objeto, mientras que los métodos son funciones que realizan tareas específicas. La correcta utilización de atributos y métodos permite crear programas más organizados y funcionales.[3]

1.4 Bootstrap

Bootstrap es un framework front-end utilizado para desarrollar aplicaciones web y sitios mobile first, o sea, con un layout que se adapta a la pantalla del dispositivo utilizado por el usuario.[4]

Inicialmente, se llamó Twitter Blueprint y, un poco más tarde, en 2011, se transformó en código abierto y su nombre cambió para Bootstrap. Desde entonces fue actualizado varias veces y ya se encuentra en la versión 4.4.

El framework combina CSS y JavaScript para estilizar los elementos de una página HTML. Permite mucho más que, simplemente, cambiar el color de los botones y los enlaces.

Esta es una herramienta que proporciona interactividad en la página, por lo que ofrece una serie de componentes que facilitan la comunicación con el usuario, como menús de navegación, controles de página, barras de progreso y más.

Además de todas las características que ofrece el framework, su principal objetivo es permitir la construcción de sitios web responsive para dispositivos móviles.

Esto significa que las páginas están diseñadas para funcionar en desktop, tablets y smartphones, de una manera muy simple y organizada.

Bootstrap está constituido por una serie de archivos CSS y JavaScript responsables de asignar características específicas a los elementos de la página.

Hay un archivo principal llamado bootstrap.css, que contiene una definición para todos los estilos utilizados. Básicamente, la estructura del framework se compone de dos directorios:

- css: contiene los archivos necesarios para la estilización de los elementos y una alternativa al tema original;
- js: contiene la parte posterior del archivo bootstrap.js (original y minificado), responsable de la ejecución de aplicaciones de estilo que requieren manipulación interactiva.

1.5 CSS

CSS (Cascading Style Sheets) es un lenguaje que maneja el diseño y presentación de las páginas web. Funciona junto con HTML para definir cómo lucen las páginas cuando un usuario las visita, permitiendo crear reglas de estilo que mejoran la experiencia visual del usuario.[5]

1.6 HTML

HTML (HyperText Markup Language) no es un lenguaje de programación, sino un lenguaje de marcado que define la estructura del contenido de una página web. Consiste en una serie de elementos que encierran diferentes partes del contenido para que se vean o comporten de una determinada manera.

1.7 Herencia

La herencia permite definir nuevas clases basadas en clases ya existentes para reutilizar el código, generando una jerarquía de clases dentro de una aplicación. Si

una clase deriva de otra, hereda sus atributos y métodos, y puede añadir nuevos o redefinir los heredados.

Terminología importante:

- Superclase: la clase cuyas características se heredan se conoce como superclase (o una clase base o una clase principal).
- Subclase: la clase que hereda la otra clase se conoce como subclase (o una clase derivada, clase extendida o clase hija). La subclase puede agregar sus propios campos y métodos, además de los campos y métodos de la superclase.
- Reutilización: la herencia respalda el concepto de “reutilización”, es decir, cuando queremos crear una clase nueva y ya hay una clase que incluye parte del código que queremos, podemos derivar nuestra nueva clase de la clase existente. Al hacer esto, estamos reutilizando los campos/atributos y métodos de la clase existente.

Los modificadores de acceso definen qué clases pueden acceder a un atributo o método. Esto podría servir por ejemplo para ser usados para proteger la información o mejor dicho definir cómo nuestro programa accede a ella. Es decir, los modificadores de acceso afectan a las entidades y los atributos a los que puede acceder dentro de una jerarquía de herencia.

Aunque así de pronto esto pueda parecer complejo lo mejor, para entenderlo, es resumir sus características en una descripción general rápida de los diferentes modificadores:

- Solo se puede acceder a los atributos o métodos privados (private) dentro de la misma clase.
- Se puede acceder a los atributos y métodos sin un modificador de acceso dentro de la misma clase, y por todas las demás clases dentro del mismo paquete.
- Se puede acceder a los atributos o métodos protegidos (protected) dentro de la misma clase, por todas las clases dentro del mismo paquete y por todas las subclases.
- Todas las clases pueden acceder a los atributos y métodos públicos.[6]

1.8 Clases

Una clase es un diseño que se puede utilizar para crear varios objetos individuales. Es algo parecido a un tipo de dato, y un objeto es algo parecido a una variable. Al igual que las variables son de un tipo de dato concreto, los objetos también son de una Clase concreta. Una clase define un grupo de datos (atributos) y un conjunto de comportamientos (métodos) que manipulan esos datos. La correcta gestión de clases implica el uso de abstracción, herencia y polimorfismo. Dos de los conceptos más usados en Programación Orientada a Objetos son sin lugar a dudas los de Clase y Objeto. Ahora bien, **una Clase engloba dos componentes distintos:**

- **Los atributos:** Son las propiedades que poseen los objetos de esa clase.
- **Los Métodos:** Son las acciones que los objetos de la clase pueden realizar.

1.9 SQL Server Management

SQL Server Management es una herramienta para la administración de bases de datos SQL Server. Proporciona una interfaz gráfica y herramientas avanzadas para la gestión, configuración, desarrollo y administración de bases de datos SQL Server.

SSMS permite administrar los objetos de Analysis Services, como la copia de seguridad y el procesamiento de objetos. Management Studio proporciona un proyecto de Script de Analysis Services en el que se desarrollan y guardan los scripts escritos en expresiones multidimensionales (MDX), Extensiones de minería de Datos (DMX) y XML for Analysis (XMLA). Los proyectos de Scripts de Analysis Services se usan para realizar tareas de administración o para volver a crear objetos, como bases de datos y cubos, en instancias de Analysis Services. Por ejemplo, puede desarrollar un script XMLA en un proyecto de Script de Analysis Services que cree directamente los objetos nuevos en una instancia de Analysis Services existente. Los proyectos de Scripts de Analysis Services se pueden guardar como parte de una solución e integrarlos con un control de código fuente.[7]

1.10 API Currencylayer de Conversión de Monedas

Currencylayer proporciona una API REST simple con tipos de cambio históricos y en tiempo real para 168 monedas del mundo, entregando pares de divisas en formato JSON universalmente utilizable, compatible con cualquiera de sus aplicaciones.

Los datos del tipo de cambio al contado se recuperan de varios proveedores importantes de datos Forex en tiempo real, se validan, se procesan y se entregan cada hora, cada 10 minutos o incluso dentro de la ventana de mercado de 60 segundos.

Al proporcionar el valor de mercado de divisas más representativo disponible (valor "punto medio") para cada solicitud de API, la API de currencylayer potencia convertidores de divisas, aplicaciones móviles, componentes de software financiero y sistemas de back-office en todo el mundo. Esta API es utilizada en nuestra aplicación web para calcular y mostrar precios en diferentes monedas, asegurando que los usuarios vean información precisa y actualizada en términos de costos. La integración de esta API se realiza a través de solicitudes HTTP, y la API devuelve los datos en formato JSON, que luego son procesados por nuestra aplicación para realizar las conversiones necesarias.

1.11 API de Clima OpenWeatherMap

Para mejorar la experiencia del usuario y proporcionar información relevante sobre el clima, se ha integrado una API de clima en la aplicación. Esta API llamada Openweathermap permite obtener datos climáticos en tiempo real para la ubicación de cada restaurante. La información sobre el clima se utiliza no solo para mostrar las condiciones climáticas actuales, sino también para aplicar descuentos basados en el clima, como descuentos en días de lluvia o frío. La integración de la API de clima se realiza mediante solicitudes HTTP y los datos se procesan en formato JSON. Esta funcionalidad se ha añadido al sistema de reservas, permitiendo registrar automáticamente las condiciones climáticas al momento de realizar una reserva.

1.12 Patrón Modelo Vista Controlador

El MVC o Modelo-Vista-Controlador es un patrón de arquitectura de software que, utilizando 3 componentes (Vistas, Models y Controladores) separa la lógica de la aplicación de la lógica de la vista en una aplicación. Es una arquitectura importante puesto que se utiliza tanto en componentes gráficos básicos hasta sistemas empresariales; la mayoría de los frameworks modernos utilizan MVC (o alguna adaptación del MVC) para la arquitectura. ¿Por qué se utiliza el MVC?, La razón es que nos permite separar los componentes de nuestra aplicación dependiendo de la responsabilidad que tienen, esto significa que cuando hacemos un cambio en alguna parte de nuestro código, esto no afecte otra parte del mismo. Por ejemplo, si modificamos nuestra Base de Datos, sólo deberíamos modificar el modelo que es quién se encarga de los datos y el resto de la aplicación debería permanecer intacta. Esto respeta el principio de la responsabilidad única. Es decir, una parte de tu código no debe de saber qué es lo que hace toda la aplicación, sólo debe de tener una responsabilidad.

- Modelo

Se encarga de los datos, generalmente (pero no obligatoriamente) consultando la base de datos. Actualizaciones, consultas, búsquedas, etc. todo eso va aquí, en el modelo.

- Vistas

Son la representación visual de los datos, todo lo que tenga que ver con la interfaz gráfica va aquí. Ni el modelo ni el controlador se preocupan de cómo se verán los datos, esa responsabilidad es únicamente de la vista.

- Controlador

Se encarga de controlar, recibe las órdenes del usuario y se encarga de solicitar los datos al modelo y de comunicárselos a la vista.[8]

1.13 CRUD (Create, Read, Update, Delete)

CRUD es un acrónimo que se refiere a las cuatro operaciones básicas que se pueden realizar en una base de datos o en una aplicación de gestión de datos: Crear (Create), Leer (Read), Actualizar (Update) y Eliminar (Delete). Estas operaciones son esenciales para el manejo de datos en cualquier aplicación.

- Create (Crear): Se refiere a la operación de agregar un nuevo registro a la base de datos.
- Read (Leer): Se refiere a la operación de recuperar datos de la base de datos.
- Update (Actualizar): Se refiere a la operación de modificar un registro existente en la base de datos.
- Delete (Eliminar): Se refiere a la operación de borrar un registro de la base de datos.
-

1.14 JSON (JavaScript Object Notation)

JSON es un formato de intercambio de datos ligero y fácil de leer para los humanos y fácil de generar y analizar para las máquinas. JSON se utiliza comúnmente para transmitir datos en aplicaciones web (por ejemplo, enviar datos desde el servidor al cliente, por ejemplo, para mostrar en una página web, o viceversa).

- Formato: JSON es un formato de texto que es completamente independiente del lenguaje pero utiliza convenciones que son familiares para los programadores de la familia de lenguajes C, incluidos C, C++, C#, Java, JavaScript, Perl, Python y muchos otros. Estos lenguajes tienen estructuras de datos que son bastante similares a JSON, lo que hace que JSON sea una opción natural para un formato de datos.

- Uso: JSON se usa principalmente para transmitir datos estructurados a través de una red, generalmente entre un servidor y una aplicación web. También se usa en muchas aplicaciones no relacionadas con la web, como archivos de configuración y almacenamiento de datos.

1.15 Entity Framework

Entity Framework (EF) es un marco de trabajo de mapeo objeto-relacional (ORM) de código abierto para ADO.NET, parte de .NET Framework. Permite a los desarrolladores trabajar con datos en forma de objetos y propiedades específicos del dominio, como clientes y direcciones de clientes, sin tener que preocuparse por las tablas y columnas de la base de datos subyacente donde se almacenan estos datos.

- Mapeo Objeto-Relacional: Entity Framework permite a los desarrolladores trabajar con datos en forma de objetos .NET utilizando el mapeo objeto-relacional (ORM) para abstraer la lógica de acceso a datos de la lógica del negocio.
- Consultas LINQ: Entity Framework soporta LINQ (Language Integrated Query) para escribir consultas de base de datos en un lenguaje de programación .NET como C# o VB.NET.
- Proveedores de Base de Datos: EF soporta varios proveedores de base de datos, lo que permite trabajar con diferentes sistemas de gestión de bases de datos (DBMS) sin cambiar el código de la aplicación.

2. Desarrollo:

El código desarrollado en C# con Visual Studio Community ofrece una aplicación web para gestionar un restaurante. Su objetivo principal es facilitar la administración de reservas, clientes, menús, pagos y más. La interfaz es fácil de usar, con una barra de navegación que permite acceder a diferentes secciones como "Clientes", "Menús", "Pagos", "Reseñas" y "Clima".

2.1 Arquitectura y Tecnologías Utilizadas

El sistema se ha desarrollado utilizando tecnologías web estándar como ASP.NET, C#, y SQL Server Management. Se ha seguido el patrón Modelo-Vista-Controlador (MVC) para una estructura organizada.

-ASP.NET y C#: Utilizados para el backend y la lógica del servidor.

-Bootstrap: Framework frontend que agiliza el diseño responsivo de la interfaz.

-HTML y CSS: Definen la estructura y estilo de las páginas web.

-SQL Server Management: Utilizado para la gestión de bases de datos.

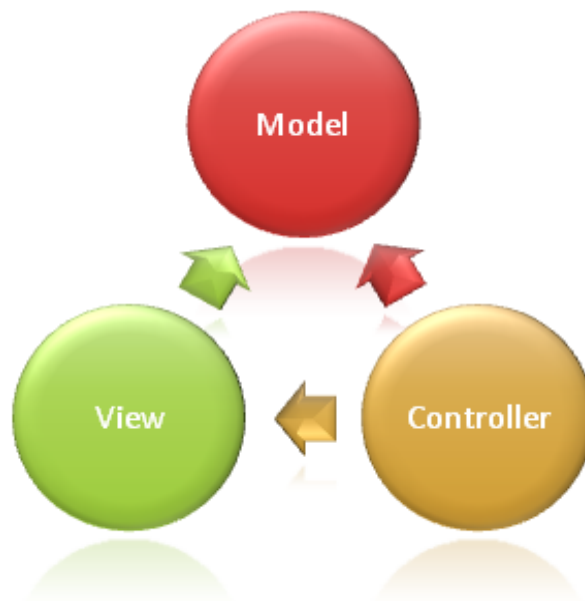
Esta combinación ofrece una base robusta para el desarrollo web, asegurando una arquitectura organizada y una interfaz moderna y atractiva.

2.2 Patrón Modelo-Vista-Controlador (MVC)

Modelo: Representa la capa de datos y lógica de negocio.

Vista: Se compone de las páginas ASPX que definen la interfaz de usuario.

Controlador: Maneja la lógica de control, recibe la entrada del usuario y coordina las interacciones entre la vista y el modelo.



2.3 Estructura del Proyecto

Página Principal (Home)

Sección de Hero: Incluye una imagen destacada del restaurante y una breve descripción.

Sección de Preferencia del Restaurante: Presenta una selección de platos y bebidas.

Slider de Imágenes: Muestra una serie de imágenes en un slider.

Acerca de Nosotros: Proporciona información sobre la historia y misión del restaurante.

Formulario de Contacto: Permite a los usuarios enviar preguntas o hacer reservaciones.

Menú de Cliente:

Categorías del Menú: Entradas, Platos Principales, Guarniciones, Postres, Menús Especiales, Bebidas, Bebidas Alcohólicas, Brunch.

Cada categoría: Despliega una lista de platos con su imagen, nombre, descripción y precio.

Barra Lateral de Navegación (Sidebar):

Incluye enlaces: A las diferentes secciones del sitio web.

Permite una navegación fácil y rápida: Entre las secciones.

Reservas:

Visualización de Reservas: Muestra una lista de todas las reservas realizadas.

Creación de Reservas: Formulario para crear nuevas reservas, asignando mesas y horarios.

Edición de Reservas: Permite modificar los detalles de una reserva existente.

Eliminación de Reservas: Funcionalidad para cancelar reservas.

Cientes:

Listado de Cientes: Muestra una lista de todos los clientes registrados.

Detalles del Cliente: Información detallada sobre un cliente específico.

Creación de Cliente: Formulario para registrar nuevos clientes.

Edición de Cliente: Permite actualizar la información de un cliente existente.

Eliminación de Cliente: Funcionalidad para eliminar un cliente del sistema.

Menús:

Listado de Menús: Visualización de todos los platos y bebidas disponibles.

Detalles del Menú: Información detallada de un plato específico.

Creación de Menú: Formulario para agregar nuevos platos al menú.

Edición de Menú: Permite modificar los detalles de un plato existente.

Eliminación de Menú: Funcionalidad para eliminar un plato del menú.

Mesas:

Listado de Mesas: Muestra todas las mesas disponibles en el restaurante.

Detalles de Mesa: Información específica sobre una mesa.

Creación de Mesa: Formulario para agregar nuevas mesas.

Edición de Mesa: Permite actualizar la información de una mesa existente.

Eliminación de Mesa: Funcionalidad para eliminar una mesa del sistema.

Órdenes:

Listado de Órdenes: Muestra todas las órdenes realizadas por los clientes.

Detalles de Órdenes: Información específica sobre una orden.

Creación de Órdenes: Formulario para registrar nuevas órdenes.

Edición de Órdenes: Permite modificar los detalles de una orden existente.

Eliminación de Órdenes: Funcionalidad para cancelar una orden.

Pagos:

Listado de Pagos: Visualización de todos los pagos realizados.

Detalles de Pagos: Información detallada de un pago específico.

Creación de Pagos: Formulario para registrar nuevos pagos.

Edición de Pagos: Permite modificar los detalles de un pago existente.

Eliminación de Pagos: Funcionalidad para eliminar un pago del sistema.

Reseñas:

Listado de Reseñas: Muestra todas las reseñas dejadas por los clientes.

Detalles de Reseñas: Información específica sobre una reseña.

Creación de Reseñas: Formulario para que los clientes puedan dejar nuevas reseñas.

Edición de Reseñas: Permite modificar una reseña existente.

Eliminación de Reseñas: Funcionalidad para eliminar una reseña del sistema.

Clima:

Visualización de Clima: Muestra las condiciones climáticas actuales.

Integración de Descuentos: Aplica descuentos basados en las condiciones climáticas.

Configuración de Clima: Permite ajustar las condiciones climáticas que activan los descuentos.

Usuarios:

Listado de Usuarios: Muestra todos los usuarios registrados en el sistema.

Detalles del Usuario: Información detallada de un usuario específico.

Creación de Usuario: Formulario para registrar nuevos usuarios.

Edición de Usuario: Permite modificar la información de un usuario existente.

Eliminación de Usuario: Funcionalidad para eliminar un usuario del sistema.

Roles:

Listado de Roles: Muestra todos los roles disponibles en el sistema.

Detalles de Roles: Información específica sobre un rol.



ANALISTA
PROGRAMADOR



Creación de Roles: Formulario para agregar nuevos roles.

Edición de Roles: Permite modificar los detalles de un rol existente.

Eliminación de Roles: Funcionalidad para eliminar un rol del sistema.

2.2 Clases del Programa

Clase BaseDeDatos

Esta clase abstracta gestiona los datos de la aplicación web y cuenta con varias listas como atributos, cada una correspondiente a una entidad del sistema. Las listas de Restaurantes, Menús, Clientes, Reservas, Pagos y Reseñas contienen objetos de sus respectivas clases. Además, posee el atributo usuarioLogeado, un objeto estático de la clase Usuario que representa al usuario autenticado. La clase cuenta con métodos para interactuar con estas listas, como agregar, eliminar y actualizar elementos.

```
32 referencias
public partial class Obligatorio2024Context : DbContext
{
    0 referencias
    public Obligatorio2024Context()
    {
    }

    0 referencias
    public Obligatorio2024Context(DbContextOptions<Obligatorio2024Context> options)
        : base(options)
    {
    }

    14 referencias
    public virtual DbSet<Cliente> Clientes { get; set; }

    8 referencias
    public virtual DbSet<Clima> Climas { get; set; }

    21 referencias
    public virtual DbSet<Menu> Menus { get; set; }

    18 referencias
```

fig 1 (1-15):

18 referencias

```
public virtual DbSet<Mesa> Mesas { get; set; }
```

7 referencias

```
public virtual DbSet<OrdenDetalle> OrdenDetalles { get; set; }
```

11 referencias

```
public virtual DbSet<Ordene> Ordenes { get; set; }
```

9 referencias

```
public virtual DbSet<Pago> Pagos { get; set; }
```

18 referencias

```
public virtual DbSet<Reserva> Reservas { get; set; }
```

7 referencias

```
public virtual DbSet<Reseña> Reseñas { get; set; }
```

20 referencias

```
public virtual DbSet<Restaurante> Restaurantes { get; set; }
```

15 referencias

```
public virtual DbSet<Role> Roles { get; set; }
```

7 referencias

```
public virtual DbSet<RolesPermiso> RolesPermisos { get; set; }
```

fig 1 (2-15):

8 referencias

```
public virtual DbSet<Usuario> Usuarios { get; set; }
```

fig 1 (3-15):

0 referencias

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<Cliente>(entity =>
    {
        entity.HasKey(e => e.Id).HasName("PK__Clientes__3214EC27D1BE8A99");

        entity.HasIndex(e => e.Email, "UQ__Clientes__A9D10534DA6CAAD6").IsUnique();

        entity.Property(e => e.Id).HasColumnName("ID");
        entity.Property(e => e.Apellido)
            .HasMaxLength(100)
            .IsUnicode(false);
        entity.Property(e => e.Email)
            .HasMaxLength(100)
            .IsUnicode(false);
        entity.Property(e => e.Nombre)
            .HasMaxLength(100)
            .IsUnicode(false);
        entity.Property(e => e.Telefono)
            .HasMaxLength(15)
            .IsUnicode(false);
        entity.Property(e => e.TipoCliente)
            .HasMaxLength(50)
            .IsUnicode(false);
    });
}
```

fig 1 (4-15):

```
modelBuilder.Entity<Clima>(entity =>
{
    entity.HasKey(e => e.Id).HasName("PK__Clima__3214EC2732B5A9A0");

    entity.ToTable("Clima");

    entity.Property(e => e.Id).HasColumnName("ID");
    entity.Property(e => e.DescripciónClima)
        .HasMaxLength(255)
        .IsUnicode(false);
    entity.Property(e => e.Fecha).HasColumnType("datetime");
    entity.Property(e => e.ReservaId).HasColumnName("ReservaID");
    entity.Property(e => e.Temperatura).HasColumnType("decimal(5, 2)");

    entity.HasOne(d => d.Reserva).WithMany(p => p.Climas)
        .HasForeignKey(d => d.ReservaId)
        .OnDelete(DeleteBehavior.Cascade)
        .HasConstraintName("FK_Clima_Reservas");
});
```

fig 1 (5-15):

```

modelBuilder.Entity<Menu>(entity =>
{
    entity.HasKey(e => e.Id).HasName("PK__Menu__3214EC2734CD0CE4");

    entity.ToTable("Menu");

    entity.Property(e => e.Id).HasColumnName("ID");
    entity.Property(e => e.Categoria)
        .HasMaxLength(100)
        .IsUnicode(false);
    entity.Property(e => e.Descripción)
        .HasMaxLength(255)
        .IsUnicode(false);
    entity.Property(e => e.Disponible)
        .HasMaxLength(2)
        .IsUnicode(false)
        .HasDefaultValue("Si");
    entity.Property(e => e.ImagenUrl)
        .HasMaxLength(255)
        .IsUnicode(false)
        .HasColumnName("ImagenURL");
    entity.Property(e => e.NombrePlato)
        .HasMaxLength(100)
        .IsUnicode(false);
    entity.Property(e => e.Precio).HasColumnType("decimal(10, 2)");
});

```

fig 1 (6-15):

```

modelBuilder.Entity<Mesa>(entity =>
{
    entity.HasKey(e => e.Id).HasName("PK__Mesas__3214EC271657CC8A");

    entity.Property(e => e.Id).HasColumnName("ID");
    entity.Property(e => e.Estado)
        .HasMaxLength(50)
        .IsUnicode(false);
    entity.Property(e => e.RestauranteId).HasColumnName("RestauranteID");

    entity.HasOne(d => d.Restaurante).WithMany(p => p.Mesas)
        .HasForeignKey(d => d.RestauranteId)
        .OnDelete(DeleteBehavior.Cascade)
        .HasConstraintName("FK_Mesas_Restaurantes");
});

```

fig 1 (7-15):

```

modelBuilder.Entity<OrdenDetalle>(entity =>
{
    entity.HasKey(e => e.Id).HasName("PK__OrdenDet__3214EC2706636105");

    entity.Property(e => e.Id).HasColumnName("ID");
    entity.Property(e => e.MenuId).HasColumnName("MenuID");
    entity.Property(e => e.OrdenId).HasColumnName("OrdenID");

    entity.HasOne(d => d.Menu).WithMany(p => p.OrdenDetalles)
        .HasForeignKey(d => d.MenuId)
        .HasConstraintName("FK_OrdenDetalles_Menu");

    entity.HasOne(d => d.Orden).WithMany(p => p.OrdenDetalles)
        .HasForeignKey(d => d.OrdenId)
        .HasConstraintName("FK_OrdenDetalles_Ordenes");
});

```

fig 1 (8-15):

```

modelBuilder.Entity<Ordene>(entity =>
{
    entity.HasKey(e => e.Id).HasName("PK__Ordenes__3214EC2760D86D40");

    entity.Property(e => e.Id).HasColumnName("ID");
    entity.Property(e => e.ReservaId).HasColumnName("ReservaID");
    entity.Property(e => e.Total).HasColumnType("decimal(10, 2)");

    entity.HasOne(d => d.Reserva).WithMany(p => p.Ordenes)
        .HasForeignKey(d => d.ReservaId)
        .OnDelete(DeleteBehavior.Cascade)
        .HasConstraintName("FK_Ordenes_Reservas");
});

```

fig 1 (9-15):

```

modelBuilder.Entity<Pago>(entity =>
{
    entity.HasKey(e => e.Id).HasName("PK__Pagos__3214EC27B323BD92");

    entity.Property(e => e.Id).HasColumnName("ID");
    entity.Property(e => e.FechaPago).HasColumnType("datetime");
    entity.Property(e => e.MetodoPago)
        .HasMaxLength(50)
        .IsUnicode(false);
    entity.Property(e => e.Moneda)
        .HasMaxLength(50)
        .IsUnicode(false);
    entity.Property(e => e.Monto).HasColumnType("decimal(10, 2)");
    entity.Property(e => e.ReservaId).HasColumnName("ReservaID");
    entity.Property(e => e.TipoCambio).HasColumnType("decimal(10, 4)");

    entity.HasOne(d => d.Reserva).WithMany(p => p.Pagos)
        .HasForeignKey(d => d.ReservaId)
        .HasConstraintName("FK_Pagos_Reservas");
});

```

fig 1 (10-15):

```

modelBuilder.Entity<Reserva>(entity =>
{
    entity.HasKey(e => e.Id).HasName("PK__Reservas__3214EC2717DB00C4");

    entity.Property(e => e.Id).HasColumnName("ID");
    entity.Property(e => e.ClienteId).HasColumnName("ClienteID");
    entity.Property(e => e.Estado)
        .HasMaxLength(50)
        .IsUnicode(false);
    entity.Property(e => e.FechaReserva).HasColumnType("datetime");
    entity.Property(e => e.MesaId).HasColumnName("MesaID");

    entity.HasOne(d => d.Cliente).WithMany(p => p.Reservas)
        .HasForeignKey(d => d.ClienteId)
        .OnDelete(DeleteBehavior.Cascade)
        .HasConstraintName("FK_Reservas_Clientes");

    entity.HasOne(d => d.Mesa).WithMany(p => p.Reservas)
        .HasForeignKey(d => d.MesaId)
        .OnDelete(DeleteBehavior.Cascade)
        .HasConstraintName("FK_Reservas_Mesas");
});

```

fig 1 (11-15):

```

modelBuilder.Entity<Reseña>(entity =>
{
    entity.HasKey(e => e.Id).HasName("PK__Reseñas__3214EC27CC87EA5E");

    entity.Property(e => e.Id).HasColumnName("ID");
    entity.Property(e => e.ClienteId).HasColumnName("ClienteID");
    entity.Property(e => e.Comentario)
        .HasMaxLength(1000)
        .IsUnicode(false);
    entity.Property(e => e.FechaReseña).HasColumnType("datetime");
    entity.Property(e => e.RestauranteId).HasColumnName("RestauranteID");

    entity.HasOne(d => d.Cliente).WithMany(p => p.Reseñas)
        .HasForeignKey(d => d.ClienteId)
        .OnDelete(DeleteBehavior.Cascade)
        .HasConstraintName("FK_Reseñas_Clientes");

    entity.HasOne(d => d.Restaurante).WithMany(p => p.Reseñas)
        .HasForeignKey(d => d.RestauranteId)
        .OnDelete(DeleteBehavior.Cascade)
        .HasConstraintName("FK_Reseñas_Restaurantes");
});

```

fig 1 (12-15):

```

modelBuilder.Entity<Restaurante>(entity =>
{
    entity.HasKey(e => e.Id).HasName("PK__Restaura__3214EC2787A97561");

    entity.Property(e => e.Id).HasColumnName("ID");
    entity.Property(e => e.Dirección)
        .HasMaxLength(255)
        .IsUnicode(false);
    entity.Property(e => e.Nombre)
        .HasMaxLength(100)
        .IsUnicode(false);
    entity.Property(e => e.Teléfono)
        .HasMaxLength(15)
        .IsUnicode(false);
});

modelBuilder.Entity<Role>(entity =>
{
    entity.HasKey(e => e.Id).HasName("PK__Roles__3214EC2792EFD105");

    entity.Property(e => e.Id).HasColumnName("ID");
    entity.Property(e => e.Nombre)
        .HasMaxLength(50)
        .IsUnicode(false);
});

```

fig 1 (13-15):

```

modelBuilder.Entity<RolesPermiso>(entity =>
{
    entity.HasKey(e => e.Id).HasName("PK__RolesPer__3214EC277425FBF0");

    entity.Property(e => e.Id).HasColumnName("ID");
    entity.Property(e => e.Permiso)
        .HasMaxLength(255)
        .IsUnicode(false);
    entity.Property(e => e.RolId).HasColumnName("RolID");

    entity.HasOne(d => d.Rol).WithMany(p => p.RolesPermisos)
        .HasForeignKey(d => d.RolId)
        .OnDelete(DeleteBehavior.Cascade)
        .HasConstraintName("FK_RolesPermisos_Roles");
});

```

fig 1 (14-15):

```

modelBuilder.Entity<Usuario>(entity =>
{
    entity.HasKey(e => e.Id).HasName("PK__Usuarios__3214EC27B642DCEA");

    entity.HasIndex(e => e.Email, "UQ__Usuarios__A9D1053468104A9F").IsUnique();

    entity.Property(e => e.Id).HasColumnName("ID");
    entity.Property(e => e.Apellido)
        .HasMaxLength(100)
        .IsUnicode(false);
    entity.Property(e => e.Contraseña)
        .HasMaxLength(100)
        .IsUnicode(false);
    entity.Property(e => e.Email)
        .HasMaxLength(100)
        .IsUnicode(false);
    entity.Property(e => e.Nombre)
        .HasMaxLength(100)
        .IsUnicode(false);
    entity.Property(e => e.RolId).HasColumnName("RolID");
    entity.Property(e => e.Telefono)
        .HasMaxLength(15)
        .IsUnicode(false);

    entity.HasOne(d => d.Rol).WithMany(p => p.Usuarios)
        .HasForeignKey(d => d.RolId)
        .OnDelete(DeleteBehavior.Cascade)
        .HasConstraintName("FK_Usuarios_Roles");
});

OnModelCreatingPartial(modelBuilder);
}

1 referencia
partial void OnModelCreatingPartial(ModelBuilder modelBuilder);

```

fig 1 (15-15):

Clase Cliente:

La clase Cliente representa un cliente en el sistema y contiene información relacionada con su identificación, nombre, apellido, dirección y número de teléfono.

Propiedades:

- Cedula: Identificación única del cliente.
- Nombre: Nombre del cliente.
- Apellido: Apellido del cliente.
- Direccion: Dirección del cliente.
- Telefono: Número de teléfono del cliente.
- TipoCliente: Tipo de cliente (Nuevo, Frecuente, VIP).
- Constructor: Constructor predeterminado. Constructor que acepta todos los parámetros necesarios para inicializar las propiedades del cliente.

Métodos de Acceso: Métodos de acceso (get y set) para cada una de las propiedades del cliente.

```
6 referencias
public partial class Cliente
{
    10 referencias
    public int Id { get; set; }

    1 referencia
    public string Nombre { get; set; } = null!;

    1 referencia
    public string Apellido { get; set; } = null!;

    6 referencias
    public string Email { get; set; } = null!;

    2 referencias
    public string? Telefono { get; set; }

    3 referencias
    public string TipoCliente { get; set; } = null!;

    1 referencia
    public virtual ICollection<Reserva> Reservas { get; set; } = new List<Reserva>();

    1 referencia
    public virtual ICollection<Reseña> Reseñas { get; set; } = new List<Reseña>();
}
```

fig 2:

Clase Restaurante:

La clase Restaurante representa un restaurante en el sistema y contiene información relacionada con su identificación, nombre y dirección.

Propiedades:

- Id: Identificación única del restaurante.
- Nombre: Nombre del restaurante.
- Dirección: Dirección del restaurante.
- Constructor: Constructor predeterminado. Constructor que acepta todos los parámetros necesarios para inicializar las propiedades del restaurante.

Métodos de Acceso: Métodos de acceso (get y set) para cada una de las propiedades del restaurante.

```

6 referencias
public partial class Restaurante
{
    8 referencias
    public int Id { get; set; }

    1 referencia
    public string Nombre { get; set; } = null!;

    7 referencias
    public string Dirección { get; set; } = null!;

    1 referencia
    public string Teléfono { get; set; } = null!;

    1 referencia
    public string Ciudad { get; set; } = null!;

    1 referencia
    public virtual ICollection<Mesa> Mesas { get; set; } = new List<Mesa>();

    1 referencia
    public virtual ICollection<Reseña> Reseñas { get; set; } = new List<Reseña>();
}

```

fig 3:

Clase Reseña:

La clase Reseña modela una reseña hecha por un cliente sobre un restaurante.

Propiedades:

- Id: Identificación única de la reseña.
- ClientId: Identificación del cliente que realiza la reseña.
- RestaurantId: Identificación del restaurante que es reseñado.
- Puntaje: Puntuación otorgada en la reseña.
- Comentario: Comentario escrito en la reseña. FechaReseña: Fecha en que se realiza la reseña.
- Constructor: Constructor predeterminado. Constructor que acepta todos los parámetros necesarios para inicializar las propiedades de la reseña.

Métodos de Acceso: Métodos de acceso (get y set) para cada propiedad, siguiendo convenciones de estilo de programación.

```

8 referencias
▼ public partial class Reseña
{
    7 referencias
    public int Id { get; set; }

    5 referencias
    public int? ClienteId { get; set; }

    7 referencias
    public int? RestauranteId { get; set; }

    0 referencias
    public int Puntaje { get; set; }

    1 referencia
    public string? Comentario { get; set; }

    1 referencia
    public DateTime FechaReseña { get; set; }

    4 referencias
    public virtual Cliente? Cliente { get; set; }

    4 referencias
    public virtual Restaurante? Restaurante { get; set; }
}

```

fig 4:

Clase Pago:

La clase Pago modela un pago realizado por un cliente.

Propiedades:

- Id: Identificación única del pago.
- Reservald: Identificación de la reserva asociada al pago.
- Monto: Monto del pago.
- MetodoPago: Método de pago utilizado.
- Moneda: Moneda en la que se realizó el pago.
- FechaPago: Fecha en la que se realizó el pago.
- Constructor: Constructor predeterminado. Constructor que acepta todos los parámetros necesarios para inicializar las propiedades del pago.

Métodos de Acceso: Métodos de acceso (get y set) para cada propiedad, siguiendo convenciones de estilo de programación.

```

10 referencias
public partial class Pago
{
    9 referencias
    public int Id { get; set; }
    10 referencias
    public int? ReservaId { get; set; }
    11 referencias
    public decimal Monto { get; set; }
    5 referencias
    public DateTime FechaPago { get; set; }
    6 referencias
    public string MetodoPago { get; set; } = null!;
    6 referencias
    public string Moneda { get; set; } = null!;
    5 referencias
    public decimal? TipoCambio { get; set; }
    7 referencias
    public virtual Reserva? Reserva { get; set; }
}

```

fig 5 (1-2):

```

1 referencia
public static async Task<Pago> CrearPagoAsync(decimal monto, string moneda, string metodoPago, CurrencyService currencyService)
{
    var pago = new Pago
    {
        Monto = monto,
        FechaPago = DateTime.Now,
        MetodoPago = metodoPago,
        Moneda = moneda
    };

    if (moneda != "UYU")
    {
        decimal? tipoCambio = await currencyService.GetExchangeRate("UYU", moneda);
        pago.TipoCambio = tipoCambio;
        if (tipoCambio.HasValue)
        {
            pago.Monto = monto * tipoCambio.Value; // Calcula el monto en UYU
        }
    }
    else
    {
        pago.TipoCambio = 1; // El tipo de cambio es 1 para UYU
    }

    return pago;
}
}

```

fig 5 (2-2):

Clase Reserva:

La clase Reserva representa una reserva de mesa en el restaurante.

Propiedades:

- Id: Identificación única de la reserva.
- Clienteld: Identificación del cliente que realiza la reserva.
- Restauranteld: Identificación del restaurante donde se realiza la reserva.
- Mesald: Identificación de la mesa reservada.
- FechaReserva: Fecha y hora de la reserva.
- NumeroPersonas: Número de personas para la reserva.
- Constructor: Constructor predeterminado. Constructor que acepta todos los parámetros necesarios para inicializar las propiedades de la reserva.

Métodos de Acceso: Métodos de acceso (get y set) para cada una de las propiedades de la reserva.

```
public partial class Reserva
{
    11 referencias
    public int Id { get; set; }

    5 referencias
    public int? ClienteId { get; set; }

    15 referencias
    public int? MesaId { get; set; }

    3 referencias
    public DateTime FechaReserva { get; set; }

    1 referencia
    public string Estado { get; set; } = null!;

    14 referencias
    public virtual Cliente? Cliente { get; set; }

    1 referencia
    public virtual ICollection<Clima> Climas { get; set; } = new List<Clima>();

    9 referencias
    public virtual Mesa? Mesa { get; set; }

    1 referencia
    public virtual ICollection<Ordene> Ordenes { get; set; } = new List<Ordene>();

    1 referencia
    public virtual ICollection<Pago> Pagos { get; set; } = new List<Pago>();
}
```

fig 6:

Clase Mesa:

La clase Mesa representa una mesa en un restaurante.

Propiedades:

- Id: Identificación única de la mesa.
- NumeroMesa: Número de la mesa.
- Capacidad: Capacidad de la mesa. Estado:
- Estado de la mesa (Disponibile, Ocupada). RestauranteId: Identificación del restaurante al que pertenece la mesa.
- Constructor: Constructor predeterminado. Constructor que acepta todos los parámetros necesarios para inicializar las propiedades de la mesa.

Métodos de Acceso: Métodos de acceso (get y set) para cada una de las propiedades de la mesa.

```
7 referencias
public partial class Mesa
{
    18 referencias
    public int Id { get; set; }

    12 referencias
    public int NumeroMesa { get; set; }

    0 referencias
    public int Capacidad { get; set; }

    10 referencias
    public string Estado { get; set; } = null!;

    13 referencias
    public int? RestauranteId { get; set; }

    1 referencia
    public virtual ICollection<Reserva> Reservas { get; set; } = new List<Reserva>();

    18 referencias
    public virtual Restaurante? Restaurante { get; set; }
}
```

fig 7:

Clase Menu:

La clase Menu representa un plato o conjunto de platos ofrecidos por el restaurante.

Propiedades:

- Id: Identificación única del menú.
- NombrePlato: Nombre del plato.
- Descripcion: Descripción del plato.
- Precio: Precio del plato.
- ImagenUrl: URL de la imagen del plato.
- Categoria: Categoría a la que pertenece el plato (Entradas, Platos Principales, etc.). Disponible: Indica si el plato está disponible.

- Constructor: Constructor predeterminado. Constructor que acepta todos los parámetros necesarios para inicializar las propiedades del menú.

Métodos de Acceso: Métodos de acceso (get y set) para cada propiedad, siguiendo convenciones de estilo de programación.

```
6 referencias
public partial class Menu
{
    7 referencias
    public int Id { get; set; }

    2 referencias
    public string NombrePlato { get; set; } = null!;

    1 referencia
    public string Descripción { get; set; } = null!;

    2 referencias
    public decimal Precio { get; set; }

    2 referencias
    public string? ImagenUrl { get; set; }

    9 referencias
    public string Categoria { get; set; } = null!;

    1 referencia
    public string Disponible { get; set; } = null!;

    1 referencia
    public virtual ICollection<OrdenDetalle> OrdenDetalles { get; set; } = new List<OrdenDetalle>();
}
```

fig 8:

Clase Usuario

La clase Usuario representa a un usuario del sistema y contiene información asociada a su identificación, nombre de usuario, contraseña y permisos específicos para visualizar distintas secciones de la aplicación. A continuación, se presenta un análisis detallado:

Propiedades:

- Id: Identificación única del usuario.
- NombreUsuario: Nombre del usuario.
- Contraseña: Contraseña del usuario.
- Rol: Rol del usuario (Administrador, Cliente).
- Constructor: Constructor predeterminado. Constructor que acepta todos los parámetros necesarios para inicializar las propiedades del usuario.

Métodos de Acceso: Métodos de acceso (get y set) para cada una de las propiedades del usuario.

```
6 referencias
public partial class Usuario
{
    8 referencias
    public int Id { get; set; }

    1 referencia
    public string Nombre { get; set; } = null!;

    1 referencia
    public string Apellido { get; set; } = null!;

    4 referencias
    public string Email { get; set; } = null!;

    2 referencias
    public string Contraseña { get; set; } = null!;

    1 referencia
    public string? Telefono { get; set; }

    5 referencias
    public int? RolId { get; set; }

    4 referencias
    public virtual Role? Rol { get; set; }
}
```

fig 9:

Clases Adicionales:

Clase Rol

La clase Rol representa los diferentes roles que un usuario puede tener dentro del sistema.

Aquí está el análisis detallado de la implementación:

Propiedades:

- Id: Identificación única del rol.
- NombreRol: Nombre del rol (Administrador, Cliente).
- Constructor: Constructor predeterminado. Constructor que acepta todos los parámetros necesarios para inicializar las propiedades del rol.

Métodos de Acceso: Métodos de acceso (get y set) para cada propiedad.

```
6 referencias
public partial class Role
{
    7 referencias
    public int Id { get; set; }

    1 referencia
    public string Nombre { get; set; } = null!;

    1 referencia
    public virtual ICollection<RolesPermiso> RolesPermisos { get; set; } = new List<RolesPermiso>();

    1 referencia
    public virtual ICollection<Usuario> Usuarios { get; set; } = new List<Usuario>();
}
```

fig 10:

Clase OrdenDetalle

La clase OrdenDetalle representa los detalles de una orden realizada en el restaurante.

Aquí está el análisis detallado de la implementación:

Propiedades:

- Id: Identificación única del detalle de la orden.
- OrdenId: Identificación de la orden asociada.
- MenuId: Identificación del menú asociado.
- Cantidad: Cantidad de ítems ordenados.
- Precio: Precio de los ítems ordenados.
- Constructor: Constructor predeterminado. Constructor que acepta todos los parámetros necesarios para inicializar las propiedades del detalle de la orden.

Métodos de Acceso: Métodos de acceso (get y set) para cada propiedad.

```
8 referencias
public partial class OrdenDetalle
{
    7 referencias
    public int Id { get; set; }

    5 referencias
    public int? OrdenId { get; set; }

    5 referencias
    public int? MenuId { get; set; }

    0 referencias
    public int Cantidad { get; set; }

    4 referencias
    public virtual Menu? Menu { get; set; }

    4 referencias
    public virtual Ordene? Orden { get; set; }
}
```

fig 11:

Clase Welcome

La clase Welcome representa la pantalla de bienvenida de la aplicación. Aquí está el análisis detallado de la implementación:

Propiedades:

- Mensaje: Mensaje de bienvenida mostrado a los usuarios.
- Constructor: Constructor predeterminado. Constructor que acepta todos los parámetros necesarios para inicializar las propiedades de la clase Welcome.

Métodos de Acceso: Métodos de acceso (get y set) para cada propiedad.

```
0 referencias
public long? Dt { get; set; }

[JsonProperty("sys", NullValueHandling = NullValueHandling.Ignore)]
0 referencias
public Sys Sys { get; set; }

[JsonProperty("timezone", NullValueHandling = NullValueHandling.Ignore)]
0 referencias
public long? Timezone { get; set; }

[JsonProperty("id", NullValueHandling = NullValueHandling.Ignore)]
0 referencias
public long? Id { get; set; }

[JsonProperty("name", NullValueHandling = NullValueHandling.Ignore)]
0 referencias
public string Name { get; set; }

[JsonProperty("cod", NullValueHandling = NullValueHandling.Ignore)]
0 referencias
public long? Cod { get; set; }
}
```

fig 12 (1-3):

```

public partial class Welcome
{
    [JsonProperty("coord", NullValueHandling = NullValueHandling.Ignore)]
    0 referencias
    public Coord Coord { get; set; }

    [JsonProperty("weather", NullValueHandling = NullValueHandling.Ignore)]
    1 referencia
    public List<Weather> Weather { get; set; }

    [JsonProperty("base", NullValueHandling = NullValueHandling.Ignore)]
    0 referencias
    public string Base { get; set; }

    [JsonProperty("main", NullValueHandling = NullValueHandling.Ignore)]
    1 referencia
    public Main Main { get; set; }

    [JsonProperty("visibility", NullValueHandling = NullValueHandling.Ignore)]
    0 referencias
    public long? Visibility { get; set; }

    [JsonProperty("wind", NullValueHandling = NullValueHandling.Ignore)]
    0 referencias
    public Wind Wind { get; set; }

    [JsonProperty("clouds", NullValueHandling = NullValueHandling.Ignore)]
    0 referencias
    public Clouds Clouds { get; set; }
}

```

fig 12 (2-3):

```

0 referencias
public long? Dt { get; set; }

[JsonProperty("sys", NullValueHandling = NullValueHandling.Ignore)]
0 referencias
public Sys Sys { get; set; }

[JsonProperty("timezone", NullValueHandling = NullValueHandling.Ignore)]
0 referencias
public long? Timezone { get; set; }

[JsonProperty("id", NullValueHandling = NullValueHandling.Ignore)]
0 referencias
public long? Id { get; set; }

[JsonProperty("name", NullValueHandling = NullValueHandling.Ignore)]
0 referencias
public string Name { get; set; }

[JsonProperty("cod", NullValueHandling = NullValueHandling.Ignore)]
0 referencias
public long? Cod { get; set; }
}

```

fig 12 (3-3):

Clase Clima

La clase Clima representa la información del clima, utilizada para ofrecer descuentos basados en el clima en la aplicación. Aquí está el análisis detallado de la implementación:

Propiedades:

- Id: Identificación única de la información del clima.
- Temperatura: Temperatura actual. Humedad: Humedad actual. Condiciones: Condiciones climáticas actuales (Soleado, Lluvioso, etc.).
- Constructor: Constructor predeterminado. Constructor que acepta todos los parámetros necesarios para inicializar las propiedades de la clase Clima.

Métodos de Acceso: Métodos de acceso (get y set) para cada propiedad.

```
7 referencias
public partial class Clima
{
    7 referencias
    public int Id { get; set; }

    2 referencias
    public DateTime Fecha { get; set; }

    2 referencias
    public decimal Temperatura { get; set; }

    2 referencias
    public string DescripciónClima { get; set; } = null!;

    6 referencias
    public int? ReservaId { get; set; }

    4 referencias
    public virtual Reserva? Reserva { get; set; }
}
```

fig 13:

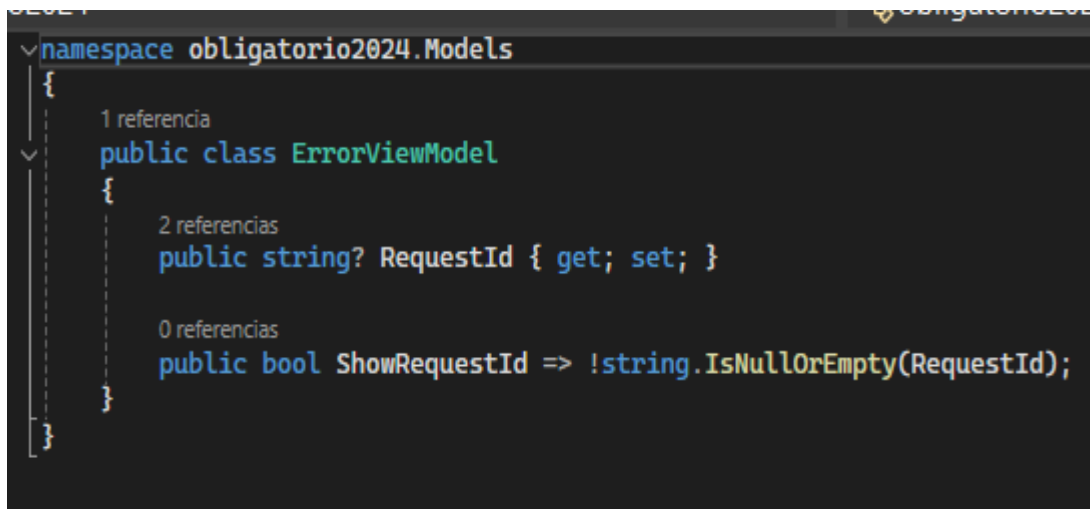
Clase ErrorViewModel

La clase ErrorViewModel representa la información sobre los errores que ocurren en la aplicación.

Propiedades:

- RequestId: Identificación única de la solicitud que causó el error.
MensajeError: Mensaje descriptivo del error.
- Constructor: Constructor predeterminado. Constructor que acepta todos los parámetros necesarios para inicializar las propiedades de la clase ErrorViewModel.

Métodos de Acceso: Métodos de acceso (get y set) para cada propiedad.



```
namespace obligatorio2024.Models
{
    1 referencia
    public class ErrorViewModel
    {
        2 referencias
        public string? RequestId { get; set; }

        0 referencias
        public bool ShowRequestId => !string.IsNullOrEmpty(RequestId);
    }
}
```

fig 14:

Clase Ordene

La clase Ordene representa una orden realizada en el restaurante.

Propiedades:

- Id: Identificación única de la orden.
- Reservald: Identificación de la reserva asociada.
- Total: Total de la orden.
- Constructor predeterminado.
- Constructor que acepta todos los parámetros necesarios para inicializar las propiedades de la orden.

Métodos de Acceso: Métodos de acceso (get y set) para cada propiedad.

```
7 referencias
public partial class Ordene
{
    7 referencias
    public int Id { get; set; }

    5 referencias
    public int? ReservaId { get; set; }

    1 referencia
    public decimal Total { get; set; }

    1 referencia
    public virtual ICollection<OrdenDetalle> OrdenDetalles { get; set; } = new List<OrdenDetalle>();

    4 referencias
    public virtual Reserva? Reserva { get; set; }
}
```

fig 15:

Clase Rol

La clase Rol representa los diferentes roles que un usuario puede tener dentro del sistema.

Propiedades:

- Id: Identificación única del rol.
- NombreRol: Nombre del rol (Administrador, Cliente).
- Constructor predeterminado.
- Constructor que acepta todos los parámetros necesarios para inicializar las propiedades del rol.

Métodos de Acceso: Métodos de acceso (get y set) para cada propiedad.

```
6 referencias
public partial class Role
{
    7 referencias
    public int Id { get; set; }

    1 referencia
    public string Nombre { get; set; } = null!;

    1 referencia
    public virtual ICollection<RolesPermiso> RolesPermisos { get; set; } = new List<RolesPermiso>();

    1 referencia
    public virtual ICollection<Usuario> Usuarios { get; set; } = new List<Usuario>();
}
```

fig 16:

2.3 Estructura del Proyecto

Página Principal (Home)

Incluye una sección de Hero con una imagen destacada del restaurante y una breve descripción. La sección de Preferencia del Restaurante presenta una selección de platos y bebidas. El Slider de Imágenes muestra una serie de imágenes en un slider. La sección Acerca de Nosotros proporciona información sobre la historia y misión del restaurante. El Formulario de Contacto permite a los usuarios enviar preguntas o hacer reservaciones.

Menú de Cliente

Categorías del Menú incluyen Entradas, Platos Principales, Guarniciones, Postres, Menús Especiales, Bebidas, Bebidas Alcohólicas, y Brunch. Cada categoría despliega una lista de platos con su imagen, nombre, descripción y precio.

Barra Lateral de Navegación (Sidebar)

Incluye enlaces a las diferentes secciones del sitio web y permite una navegación fácil y rápida entre las secciones.

Sección de Reservas

El Formulario de Reservas permite a los usuarios reservar una mesa en un restaurante específico. La Lista de Reservas muestra las reservas actuales y permite editarlas o cancelarlas.

Sección de Clientes

La Lista de Clientes muestra todos los clientes registrados en el sistema. El Formulario de Cliente permite agregar, editar o eliminar clientes.

Sección de Pagos

La Lista de Pagos muestra todos los pagos realizados. El Formulario de Pago permite registrar nuevos pagos y editarlos.

Sección de Reseñas

La Lista de Reseñas muestra todas las reseñas dejadas por los clientes. El Formulario de Reseña permite crear, editar y eliminar reseñas.

Sección de Usuarios

La Lista de Usuarios muestra todos los usuarios registrados en el sistema. El Formulario de Usuario permite agregar, editar o eliminar usuarios. La Asignación de Roles permite asignar roles específicos a los usuarios.

Sección de Roles

La Lista de Roles muestra todos los roles disponibles en el sistema. El Formulario de Rol permite crear, editar y eliminar roles.

1. Introducción a los Roles

Los roles en el sistema son fundamentales para gestionar los permisos y accesos de los usuarios. Un rol define un conjunto de permisos que un usuario puede tener, permitiendo así un control granular sobre lo que cada usuario puede ver y hacer dentro del sistema. Esto es esencial para mantener la seguridad y eficiencia en la gestión del restaurante.

2. Base de Datos

La estructura de la base de datos para gestionar los roles y permisos se compone de varias tablas: Usuarios, Roles, Permisos, y RolPermiso.

```
CREATE TABLE Usuarios (  
    ID INT PRIMARY KEY IDENTITY(1,1)  
    , Nombre VARCHAR(100) NOT NULL,  
    Apellido VARCHAR(100) NOT NULL, -- Añadido para apellido  
    Email VARCHAR(100) NOT NULL UNIQUE,  
    Contraseña VARCHAR(100) NOT NULL,  
    Telefono VARCHAR(15),  
    RolID INT,  
    CONSTRAINT FK_Usuarios_Roles FOREIGN KEY (RolID) REFERENCES Roles(ID)  
);  
  
CREATE TABLE Roles (  
    ID INT PRIMARY KEY IDENTITY(1,1),  
    Nombre VARCHAR(50) NOT NULL );
```

```

CREATE TABLE Permisos (

id INT PRIMARY KEY IDENTITY(1,1),

nombre_permiso VARCHAR(50) NOT NULL UNIQUE,

descripcion VARCHAR(255)

);

CREATE TABLE RolPermiso (

idRol INT, idPermisos INT,

PRIMARY KEY (idRol, idPermisos),

FOREIGN KEY (idRol) REFERENCES Roles(ID),

FOREIGN KEY (idPermisos) REFERENCES Permisos(id)

);

INSERT INTO RolPermiso (

idRol,

idPermisos)

VALUES (1, 2), -- Permiso 'ver ordenes'

(1, 3), -- Permiso 'ver roles'

(1, 4), -- Permiso 'ver permisos'

(1, 5), -- Permiso 'ver reservas'

(1, 6), -- Permiso 'ver restaurantes'

(1, 7), -- Permiso 'ver mesas' (1, 8), -- Permiso 'ver pagos'

(1, 9), -- Permiso 'ver clientes'

(1, 10); -- Permiso 'ver climas'

);

```

3. Descripción de las Tablas:

Usuarios: Almacena la información de los usuarios del sistema, incluyendo su rol.

Roles: Define los diferentes roles disponibles en el sistema (por ejemplo, Administrador, Gerente, Empleado).

Permisos: Define los diferentes permisos que se pueden asignar a los roles (por ejemplo, ver órdenes, ver roles).

RolPermiso: Asocia los permisos con los roles específicos.

4. Funcionalidades de Roles:

- **Creación de Roles:** Permite agregar nuevos roles al sistema.
- **Asignación de Permisos:** Cada rol puede tener múltiples permisos asignados, definidos en la tabla RolPermiso.
- **Asignación de Roles a Usuarios:** Al crear o editar un usuario, se le asigna un rol específico, determinando así los permisos que tendrá.
- **Control de Acceso:** Las vistas y acciones están protegidas mediante políticas de autorización basadas en los permisos del rol del usuario. Por ejemplo, solo los usuarios con el permiso "ver órdenes" pueden acceder a la vista de órdenes.

3.0 Funcionalidades del Programa

La gestión de Clientes permite registrar, editar y eliminar clientes, pueden ser clasificados como nuevos, frecuentes o VIP, y esta clasificación se utiliza para aplicar descuentos en los pagos. La gestión de Reservas permite a los usuarios realizar reservas para mesas en los restaurantes, incluyen información sobre la fecha, la mesa y el número de personas. Los administradores pueden ver, editar y cancelar reservas.

La gestión de Mesas permite administrar las mesas disponibles en cada restaurante. Se pueden agregar nuevas mesas, editar la información de las mesas existentes y eliminarlas si ya no están en uso. Las mesas pueden estar en estado Disponible u Ocupada.

La gestión de Menús permite a los restaurantes gestionar sus menús, incluyendo la adición de nuevos platos, la edición de platos existentes y la eliminación de platos. Los menús están categorizados en Entradas, Platos Principales, Guarniciones, Postres, Menús Especiales, Bebidas, Bebidas Alcohólicas y Brunch. Cada plato tiene un nombre, descripción, precio, imagen y disponibilidad.

La gestión de Pagos permite registrar, editar y eliminar pagos. Los pagos están asociados a reservas y pueden realizarse en diferentes monedas y métodos de pago. La aplicación calcula automáticamente los descuentos para clientes frecuentes y VIP. La gestión de Reseñas permite a los clientes dejar reseñas sobre los restaurantes.

Las reseñas incluyen un puntaje, un comentario y la fecha de la reseña. Los administradores pueden ver, editar y eliminar reseñas.

La gestión de Usuarios permite a los administradores gestionar los usuarios del sistema, incluyendo la adición de nuevos usuarios, la edición de la información de

los usuarios y la eliminación de usuarios. Los usuarios tienen roles que determinan sus permisos en la aplicación.

La Asignación de Roles permite a los administradores asignar roles a los usuarios, lo que les otorga

4.0 Futuras Implementaciones

En el desarrollo continuo del proyecto "Al mal tiempo, buena cara", se han identificado varias áreas que podrían beneficiarse de futuras mejoras e implementaciones. A continuación, se detallan algunas de las proyecciones y mejoras planificadas para el sistema.

4.1 Mejora del Sistema de Reservas:

Optimizar el sistema de reservas para incluir una vista de calendario, permitiendo a los usuarios seleccionar fechas y horas disponibles de manera más intuitiva.

Además, se podría agregar una función de notificación automática para recordar a los clientes sobre sus próximas reservas a través de correo electrónico o SMS.

4.2 Aplicación Móvil Complementaria:

Crear una aplicación móvil para Android e iOS que permita a los usuarios realizar reservas, consultar el menú, realizar pedidos y recibir notificaciones directamente desde sus dispositivos móviles. La aplicación también incluiría funcionalidades para clientes frecuentes y el sistema de recompensas.

4.3 Soporte Multilenguaje:

Ampliar el soporte del sitio web para incluir múltiples idiomas, facilitando el acceso a una audiencia más amplia y mejorando la experiencia de usuarios que no hablen el idioma principal del sitio.

5.0 Manual de Usuario

Iniciar Sesión

1. Navegue a la página de inicio de sesión desde la barra de navegación.
2. Ingrese su correo electrónico y contraseña.
3. Haga clic en "Login" para acceder a su cuenta.

The image shows a login form titled "Login" with the instruction "Por favor, ingresa tus datos para acceder a tu cuenta." Below this, there are two input fields: "Email:" with the value "lpctobinz@gmail.com" and "Contraseña:" with masked characters "*****". A red "Login" button is positioned at the bottom of the form.

fig 15:



fig 16:

Realizar una Reserva

1. Navegue a la sección de Reservas desde la barra de navegación.
2. Haga clic en "Crear Reserva".
3. Complete el formulario con los detalles de su reserva.
4. Haga clic en "Guardar" para confirmar su reserva.

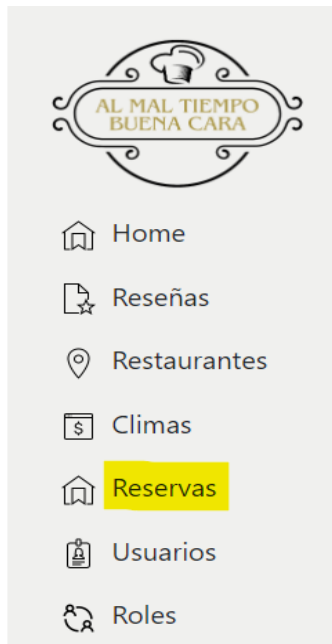
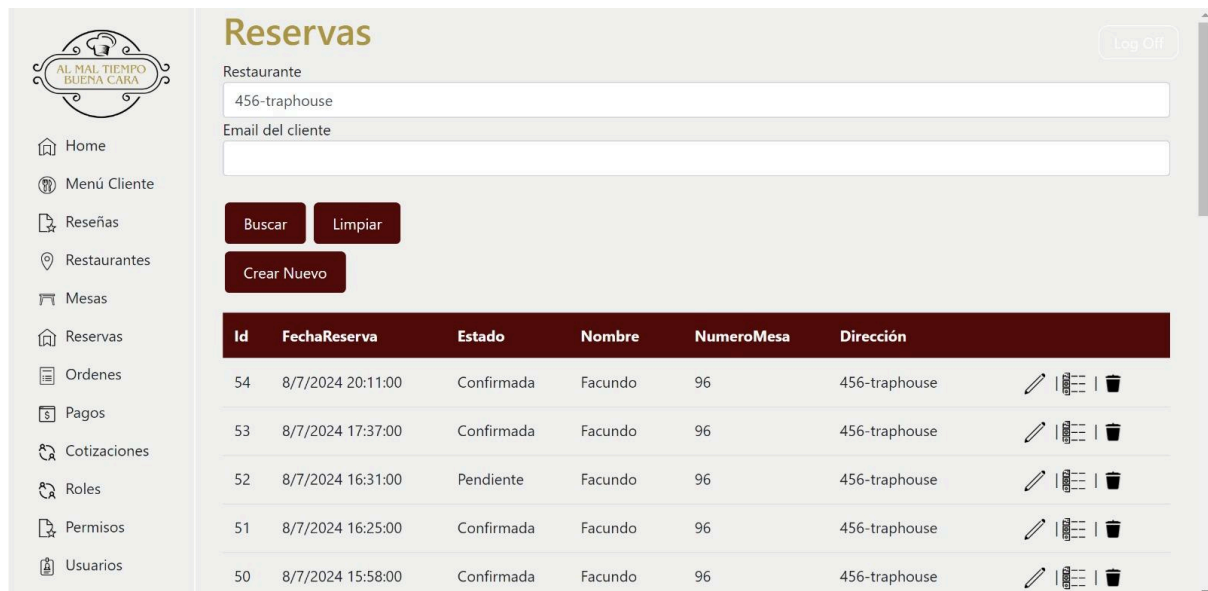


fig 17 (1-2):



Reservas

Restaurante: 456-traphouse

Email del cliente:

Buscar Limpiar

Crear Nuevo
















Id	FechaReserva	Estado	Nombre	NumeroMesa	Dirección	
54	8/7/2024 20:11:00	Confirmada	Facundo	96	456-traphouse	  
53	8/7/2024 17:37:00	Confirmada	Facundo	96	456-traphouse	  
52	8/7/2024 16:31:00	Pendiente	Facundo	96	456-traphouse	  
51	8/7/2024 16:25:00	Confirmada	Facundo	96	456-traphouse	  
50	8/7/2024 15:58:00	Confirmada	Facundo	96	456-traphouse	  

fig 17 (2-2):

Consultar el Menú

1. Navegue a la sección de Menú desde la barra de navegación.
2. Seleccione una categoría (Entradas, Platos Principales, etc.).
3. Explore la lista de platos disponibles con sus detalles.



fig 18 (1-4):

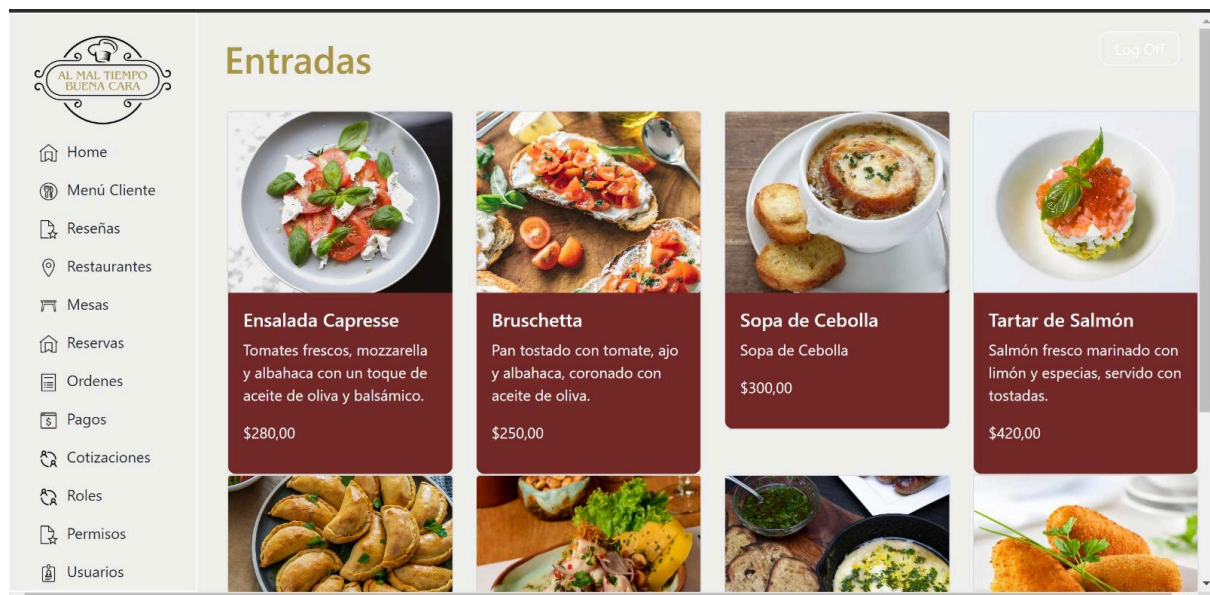


fig 18 (2-4):



fig 18 (3-3):

Dejar una Reseña

1. Navegue a la sección de Reseñas desde la barra de navegación.
2. Haga clic en "Crear Reseña".
3. Complete el formulario con su puntaje y comentario.
4. Haga clic en "Guardar" para publicar su reseña.

fig 19(1-2):

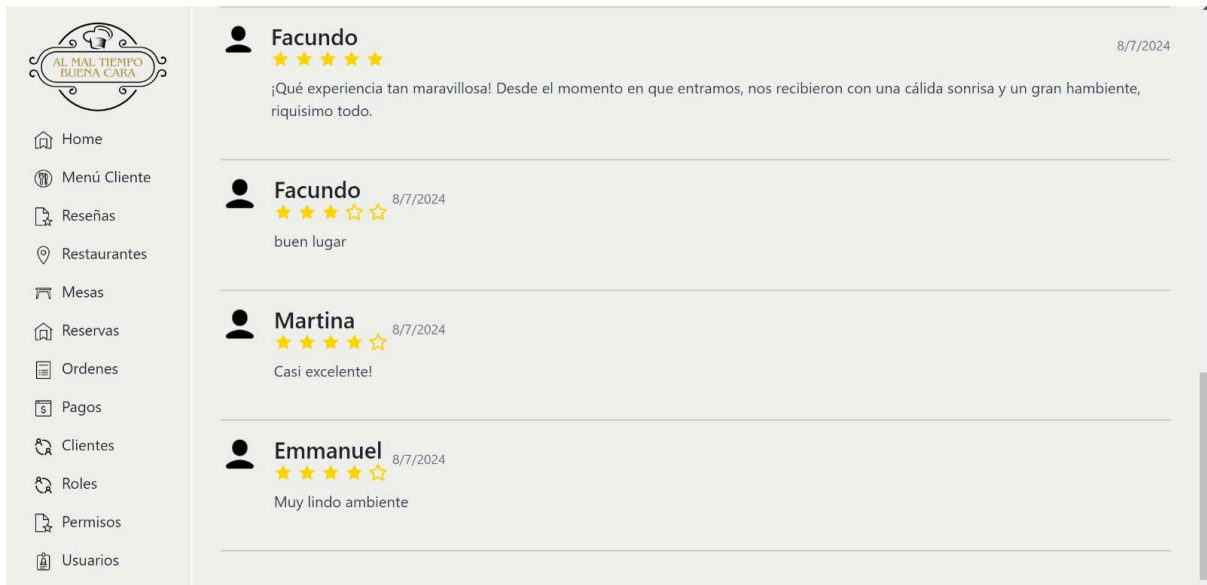


fig 19(2-2):

Gestionar Usuarios (Solo Administradores)

1. Navegue a la sección de Usuarios desde la barra de navegación.
2. Para agregar un nuevo usuario, haga clic en "Crear Usuario" y complete el formulario.
3. Para editar un usuario existente, seleccione el usuario de la lista y haga clic en "Editar".
4. Para eliminar un usuario, seleccione el usuario de la lista y haga clic en "Eliminar".

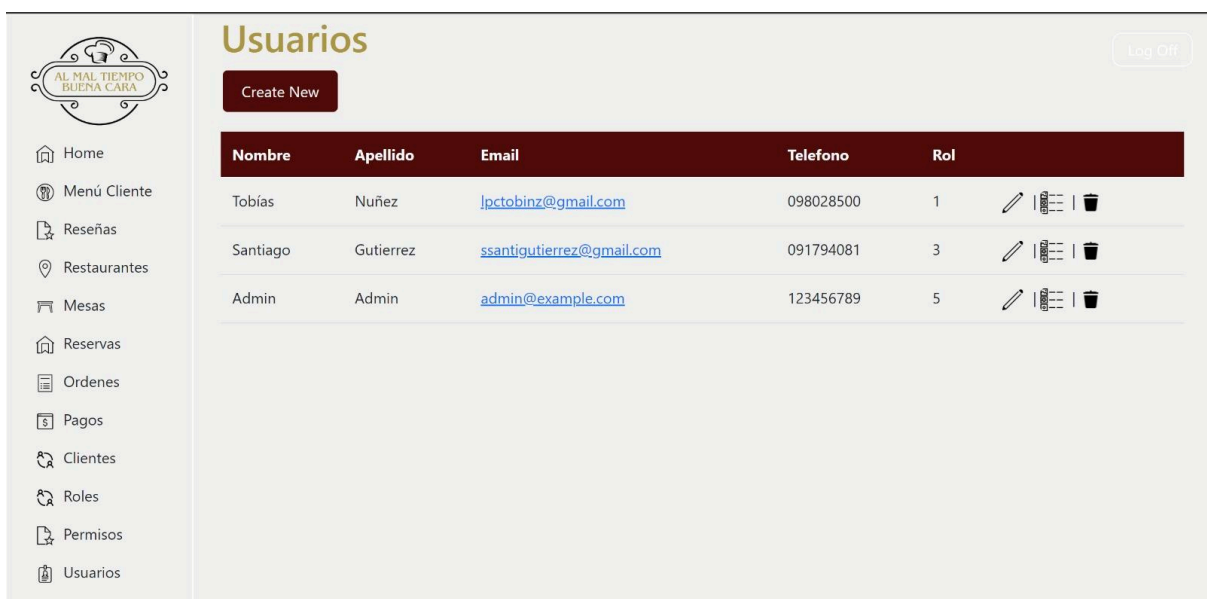



fig 20(1-4):



- Home
- Menú Cliente
- Reseñas
- Restaurantes
- Mesas
- Reservas
- Ordenes
- Pagos
- Cientes
- Roles
- Permisos

Create

Log Off

Usuario

Nombre

Julieta

Apellido

Araujo

Email

romeoyjulieta@gmail.com

Contraseña

Telefono


091641844

Rol

Funcionario

Create

fig 20(2-4):



- Home
- Menú Cliente
- Reseñas
- Restaurantes
- Mesas
- Reservas
- Ordenes
- Pagos
- Cientes
- Roles
- Permisos
- Usuarios

Edit

Log Off

Usuario

Nombre

Julieta

Apellido

Araujo

Email

romeoyjulieta@gmail.com

Contraseña

Telefono

091641844

Rol

Moza

Save

Back to List

fig 20(3-4):

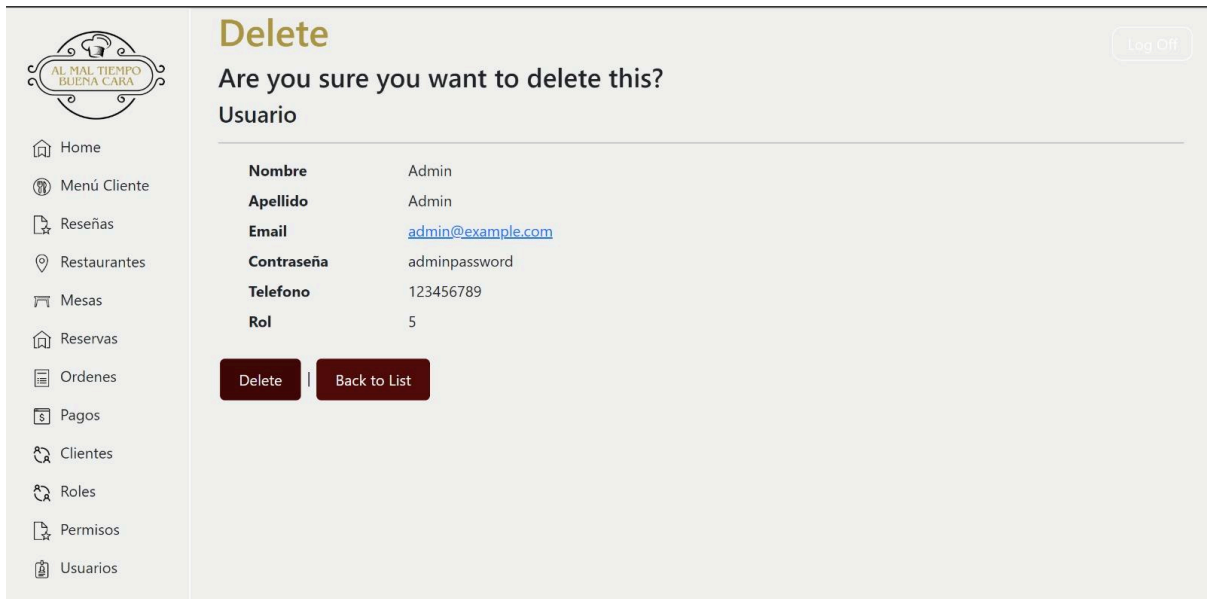


fig 20(4-4):

Asignar Roles a Usuarios (Solo Administradores)

1. Navegue a la sección de Roles desde la barra de navegación.
2. Seleccione un usuario de la lista y haga clic en "Asignar Rol".
3. Seleccione el rol deseado y haga clic en "Guardar".

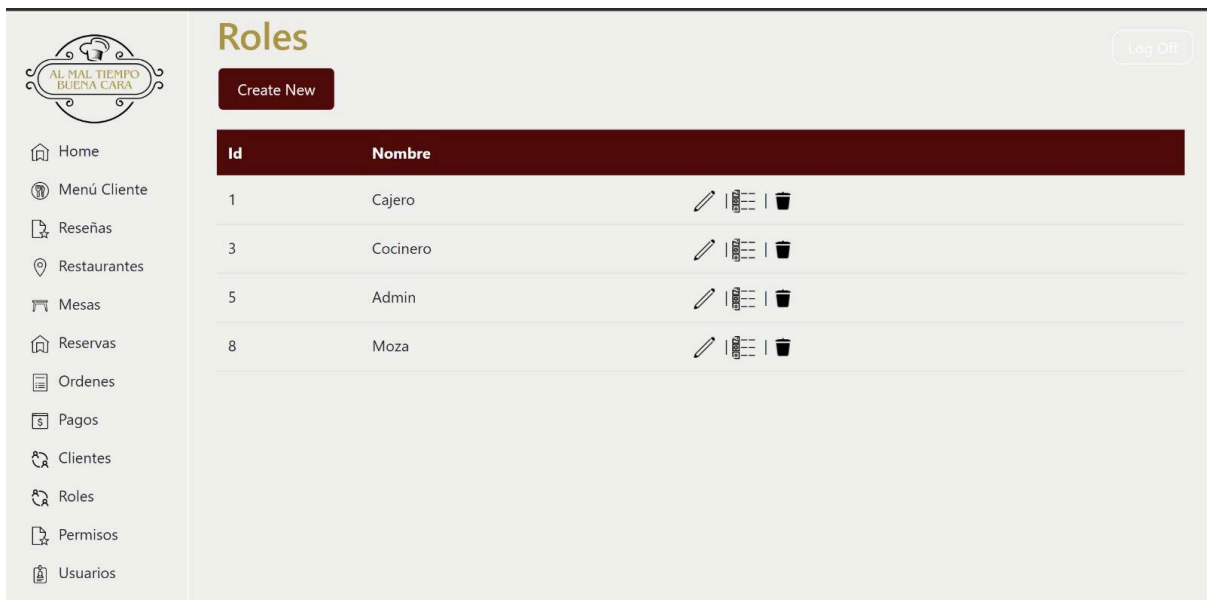



fig 21(1-4):



Home

Menú Cliente

Reseñas

Restaurantes

Mesas

Reservas

Ordenes

Pagos

Cientes

Roles

Permisos

Usuarios

Create

Role

Nombre

barman

Permisos:

☒ ver usuarios

☐ ver ordenes

☐ ver roles

☐ ver permisos

☐ ver reservas

☐ ver restaurantes

☒ ver mesas

☐ ver pagos

☒ ver clientes

☐ ver climas

Create

Back to List

Log Off

fig 21(2-4):

6.0 Diagramas:

Diagrama UML:

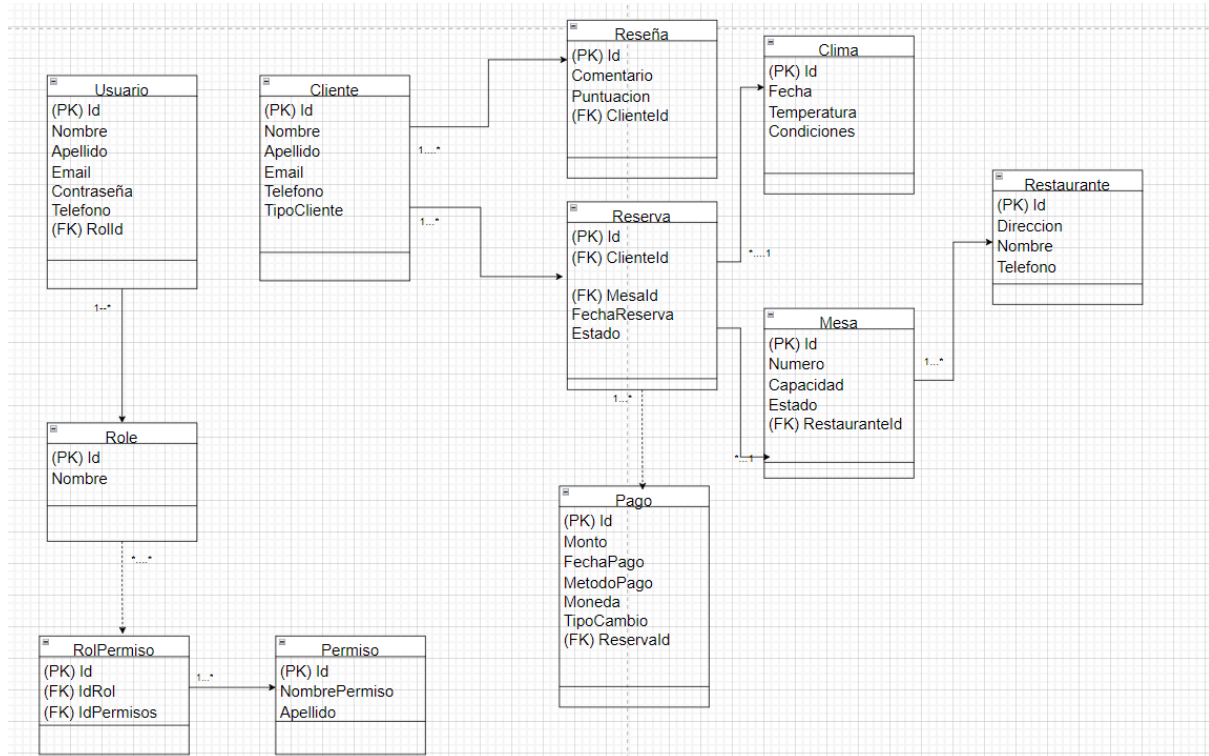
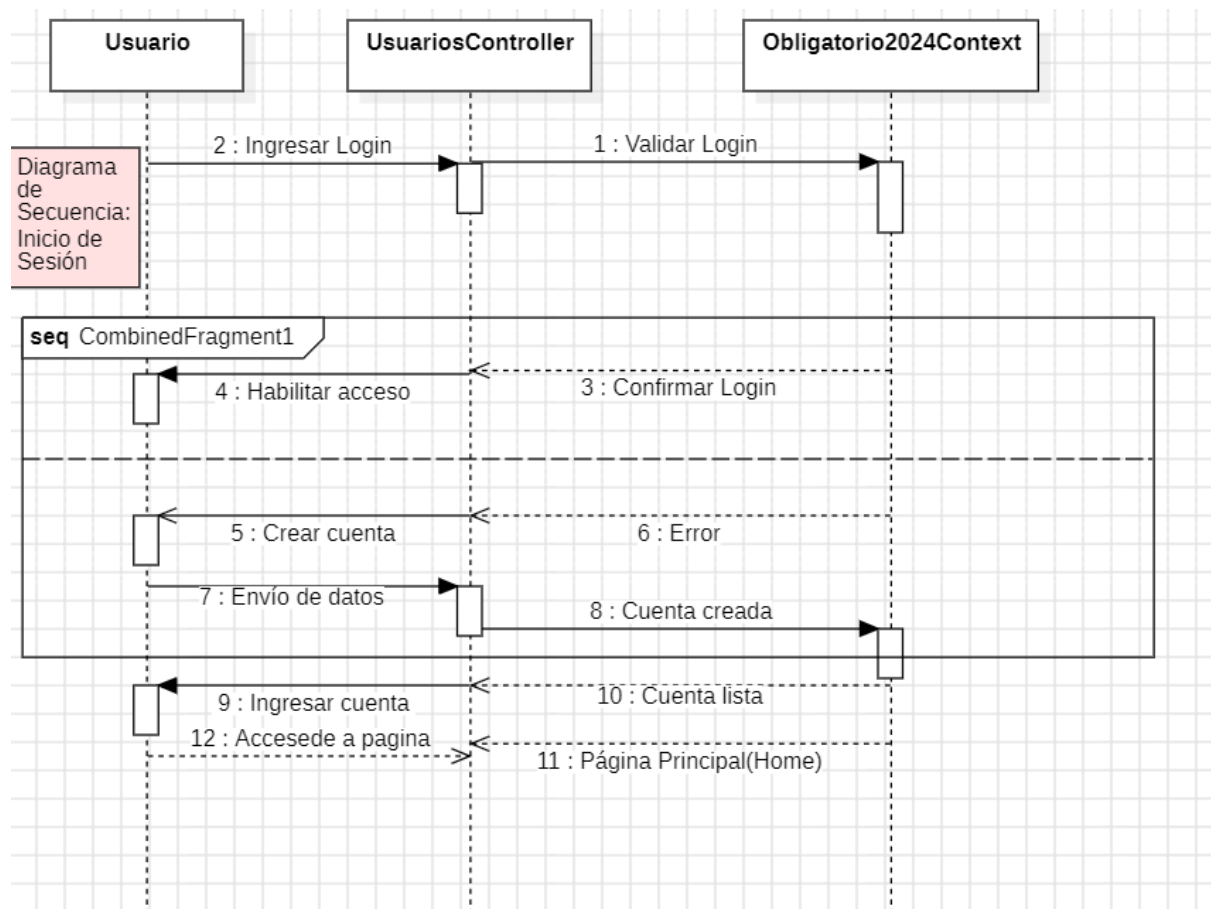


Diagrama Secuencial:



7.0 Referencias:

1. <https://aws.amazon.com/es/what-is/ide/>
2. <https://bsw.es/que-es-c/>
3. https://edea.juntadeandalucia.es/bancorecursos/file/cb77d5d9-b0a7-4c50-98f4-28073b221392/1/es-an_2021062412_9203322.zip/41_clases_y_objetos_atributos_y_mtodos.html?temp.hn=true&temp.hb=true
4. <https://rockcontent.com/es/blog/bootstrap/>
5. https://developer.mozilla.org/es/docs/Learn/CSS/First_steps/What_is_CSS
6. <https://ifgeekthen.nttdata.com/s/post/herencia-en-programacion-orientada-objetos-MCPV3PCZDNBFHSROCCU3JMI7UIJQ?language=es>
7. <https://learn.microsoft.com/es-es/sql/ssms/sql-server-management-studio-ssms?view=sql-server-ver16>
8. <https://codigofacilito.com/articulos/mvc-model-view-controller-explicado>

8.0 Conclusión:

El proyecto "Al mal tiempo, buena cara" ha demostrado ser un desarrollo significativo en la gestión de restaurantes, proporcionando una plataforma web integral que mejora tanto la operación interna como la experiencia del cliente. Utilizando tecnologías avanzadas como SQL Server Management, C#, ASP.NET MVC, Bootstrap, HTML y CSS, el sistema permite una administración eficiente de reservas, clientes, menús, pagos, reseñas y roles, ofreciendo una interfaz amigable y funcional.

La implementación de futuras mejoras, como un sistema de recompensas para clientes frecuentes, una aplicación móvil, soporte multilenguaje, integración con redes sociales y herramientas avanzadas de análisis, asegurará que el sistema se mantenga relevante y competitivo en un entorno dinámico. Estas mejoras están diseñadas para optimizar las operaciones del restaurante y aumentar la satisfacción y fidelización de los clientes.

"Al mal tiempo, buena cara" no solo facilita la gestión efectiva de un restaurante, sino que también enriquece la experiencia del cliente, posicionando al restaurante para el éxito continuo en un mercado competitivo. La planificación cuidadosa y la implementación gradual de nuevas funcionalidades garantizarán una integración efectiva y sin interrupciones, manteniendo el alto estándar de servicio que caracteriza al restaurante.