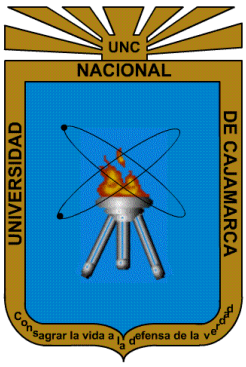


FACULTAD DE INGENIERÍA

ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMAS

DESARROLLO DE UNA SISTEMA DE GESTIÓN WEB PARA EL ÁREA DE VENTAS DEL RESTAURANTE “EL SABOR CAJABAMBINO” UTILIZANDO ASP.NET CORE MVC

Asignatura:
PROGRAMACIÓN APLICADA II

Integrantes:
Limay Rodriguez, Adriana Anthonela
Ramos Guerra, Ainhoa Jolie
Pérez Briceño, Darick André
Valdiviezo Zavaleta, Jesús Arturo

Docente:
Ing. Mendoza Santos Carlos Eduardo

Cajamarca – Perú
Agosto, 2025

Datos Informativos

Versiones

Versión	Fecha
1.0	18.08.2025

REPOSITORIO: https://github.com/Arturo969/5-Ciclo-Programacion-Aplicada-Restaurante-.git

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

RESUMEN

Este informe detalla el desarrollo y la implementación de un sistema de gestión web simplificado para el área de ventas del restaurante "El Sabor Cajabambino", una solución construida con ASP.NET Core MVC y SQL Server. El proyecto se enfoca en la implementación de operaciones CRUD (Crear, Leer, Actualizar, Borrar) para optimizar la gestión de pedidos, inventario, mesas y clientes, resolviendo los problemas derivados de los procesos manuales actuales y sentando las bases para una gestión de datos más eficiente y una mejor toma de decisiones operativas.

CAPÍTULO 1. PLANEACION DE LA SOLUCION DE APLICACIÓN WEB

1.1. Introducción

Un sistema de gestión web se define como una herramienta de software diseñada para digitalizar, optimizar y centralizar los procesos operativos de una organización. El objetivo principal de estos sistemas es transformar las tareas manuales en flujos de trabajo eficientes y precisos, proporcionando una base sólida para la toma de decisiones y la mejora continua. En este contexto, la arquitectura Modelo-Vista-Controlador (MVC) emerge como un patrón de diseño fundamental para el desarrollo de aplicaciones robustas y escalables, ya que separa la lógica de la aplicación, la interfaz de usuario y el manejo de datos, facilitando así su mantenimiento y desarrollo.

Por lo tanto, el desarrollo de sistemas de información ha evolucionado, adoptando frameworks que simplifican el proceso y promueven buenas prácticas de programación. En particular, ASP.NET Core MVC es un framework de código abierto, multiplataforma y de alto rendimiento que se ha consolidado como una opción sólida para la creación de aplicaciones web dinámicas. De este modo, la sinergia de este framework con sistemas de gestión de bases de datos relacionales como SQL Server permite una gestión de datos eficiente y segura, crucial para la integridad de la información empresarial.

Este informe, por consiguiente, detalla el desarrollo de un sistema de gestión web simplificado para el área de ventas del restaurante "El Sabor Cajabambino". Específicamente, el proyecto se centra en la implementación de las operaciones básicas de datos conocidas como CRUD (Crear, Leer, Actualizar, Borrar) para gestionar de manera integral pedidos, inventario, mesas y clientes. A lo largo del documento, se describirá la planificación, diseño y desarrollo de esta solución, demostrando cómo la aplicación de un enfoque estructurado y las tecnologías adecuadas pueden resolver problemas operacionales reales.

Finalmente, la estructura de este informe se presenta de la siguiente manera: el CAPÍTULO 1. PLANEACIÓN DEL SISTEMA DE GESTIÓN WEB establece los fundamentos del proyecto, mientras que el CAPÍTULO 2. MODELADO DEL NEGOCIO describe la empresa y su contexto. Además, el

CAPÍTULO 3. ANÁLISIS DE REQUERIMIENTOS aborda el levantamiento de información y el diseño de la base de datos. Por último, los capítulos 4, 5, 6 y 7 se centran en el marco teórico, la arquitectura, el desarrollo y las pruebas del sistema, respectivamente.

1.2. Justificación

La implementación de un sistema de gestión web simplificado en el área de ventas del restaurante "El Sabor Cajabambino" es crucial para optimizar los procesos de registro y control operacional. Actualmente, la gestión de ventas, pedidos y el inventario se realiza de manera manual o a través de herramientas poco integradas, lo que genera inconsistencias en los datos, lentitud en el servicio y dificultades para un control efectivo.

La ausencia de una plataforma centralizada y robusta dificulta la gestión de pedidos y reservas, el seguimiento del inventario en tiempo real y la correcta administración de la información de los clientes.

El desarrollo de esta solución web, utilizando tecnologías como ASP.NET Core MVC y SQL Server, permitirá digitalizar y centralizar estos procesos. El sistema transformará la gestión manual en un flujo de trabajo ágil y preciso, lo que no solo mejorará la eficiencia operativa y la satisfacción del cliente, sino que también proporcionará una base sólida para el crecimiento del negocio.

1.3. Descripción del sistema de gestión web

El sistema de gestión web es una plataforma diseñada para optimizar los procesos del área de ventas del restaurante "El Sabor Cajabambino". Desarrollado bajo la arquitectura de software Modelo-Vista-Controlador (MVC), el sistema utiliza el framework ASP.NET Core MVC como su eje tecnológico.

Asimismo, para la persistencia y gestión de los datos, se ha implementado una base de datos relacional en SQL Server. Esta base de datos almacena de manera estructurada toda la información crítica del negocio, como pedidos, inventario de cocina, estado de las mesas y datos de los clientes.

Además, el núcleo funcional del sistema se basa en la implementación de operaciones CRUD (Crear, Leer, Actualizar, Borrar) en cada uno de los módulos principales. Esto permite al personal del restaurante realizar las siguientes tareas de manera eficiente:

- Gestión de pedidos y reservas: Administrar y registrar nuevas órdenes y reservas, así como consultar y actualizar su estado.
- Control de inventario: Mantener un registro detallado del stock de ingredientes, facilitando el control y la reposición.
- Monitoreo de mesas: Visualizar en tiempo real la ocupación y disponibilidad de las mesas para una asignación ágil.
- Administración de clientes: Mantener una base de datos de clientes para mejorar la atención y personalización del servicio.
- En conjunto, la sinergia entre ASP.NET Core MVC y SQL Server, junto con la implementación de estas funcionalidades CRUD, provee una solución robusta y escalable que busca modernizar y mejorar la operación diaria del restaurante.

1.4. Objetivos

1.4.1. Objetivo General

Diseñar y desarrollar un sistema de gestión web simplificado para optimizar los procesos del área de ventas en el restaurante "El Sabor Cajabambino", utilizando el framework ASP.NET Core MVC para la gestión eficiente de datos e interacción con los usuarios.

1.4.2. Objetivos Específicos

- Diseñar e implementar un modelo de base de datos relacional que sustente los procesos de venta, inventario, pedidos y clientes, garantizando la integridad y consistencia de los datos.
- Implementar un módulo de gestión de pedidos y reservas que permita al personal del restaurante crear, consultar, modificar y eliminar registros, mejorando la organización de las ventas y la atención al cliente.
- Crear una interfaz para la gestión del inventario de cocina, que facilite el registro de ingresos y salidas de productos, permitiendo un control actualizado del stock disponible.

- Desarrollar un panel de visualización del estado de las mesas, que permita al personal monitorear en tiempo real la disponibilidad y ocupación, agilizando la asignación de mesas a los clientes.
- Desarrollar un módulo de administración de clientes que permita registrar, consultar y actualizar la información de los usuarios para una mejor personalización del servicio.

1.5. Descripción de los Stakeholders

Gerente General

El dueño de "El Sabor Cajabambino" es el Sr. José Carlos Álvarez. Empresario Experto: Con más de 15 años en el rubro de restaurantes, lo que le ha proporcionado un conocimiento profundo del mercado y las preferencias de los comensales en Cajamarca.

Estudiantes UNC

Los estudiantes de Ingeniería de Sistemas del 5to ciclo de la Universidad Nacional de Cajamarca que participan en el curso de Programación Aplicada representan un grupo de stakeholders clave en este proyecto. Poseen una sólida base en fundamentos de programación, bases de datos y análisis de sistemas, lo que les permite comprender la arquitectura y los procesos necesarios para la implementación de una solución para la gestión de datos.

CAPÍTULO 2. MODELADO DEL NEGOCIO

2.1. Descripción de la empresa u organización

El restaurante el sabor cajabambino es una empresa dedicada a transmitir el sabor típico de Cajamarca, que se esfuerza por ofrecer una experiencia culinaria que celebre la rica tradición gastronómica de la región, utilizando ingredientes frescos y recetas transmitidas de generación en generación y su guía fundamental es una visión clara y una misión enfocada en nuestro compromiso con la sociedad cajamarquina, buscando no solo deleitar paladares, sino también contribuir al desarrollo cultural y económico local.

Razón social

MEDINA LEZAMA, MARIA BERANIZ

Ubicación

Av. La Paz #720, Cajamarca, Cajamarca, Perú.

Rubro Económico

Elaboración y venta de alimentos preparados.

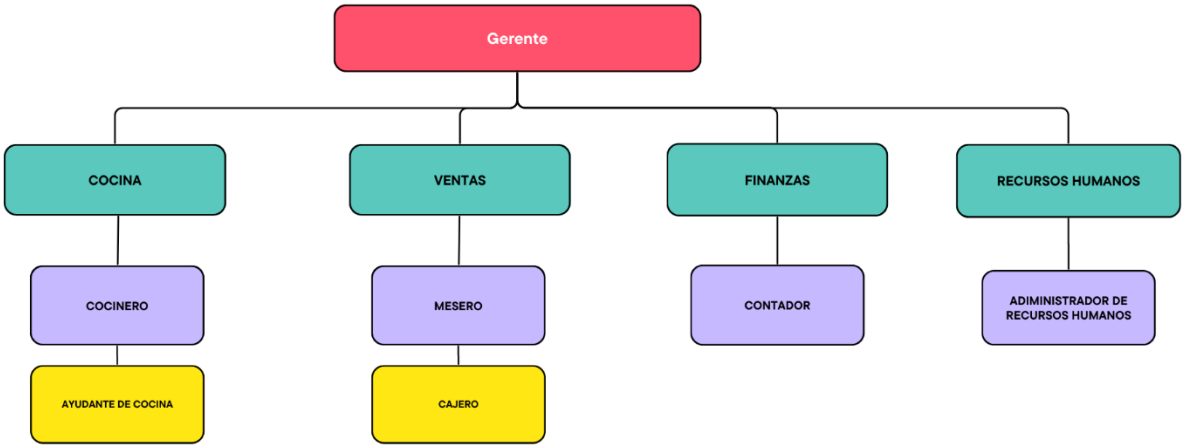
Clientes

Público diverso que incluye residentes locales, turistas nacionales e internacionales, familias, parejas, grupos de amigos y empresas. Todos ellos comparten el interés por disfrutar de la auténtica tradición culinaria cajamarquina en un ambiente acogedor que se enriquece con la experiencia de música en vivo, ideal para comidas cotidianas, celebraciones especiales y eventos corporativos.

Competidores

Otros restaurantes de comida típica cajamarquina en la ciudad u otros que ofrecen platos tradicionales peruanos en general.

2.2. Organigrama de la empresa



2.3. Recursos

2.3.1. Personal

Alumnos de la Escuela de Ingeniería de Sistemas. (4 alumnos)

2.3.2. Hardware y Software

- Laptops
- PC
- Visual Studio Community 2022
- SQL Server
- ASP.net Core MVC
- Gemini
- Bootstrap
- CSS
- HTML
- C#
- Páginas Razor
- Figma

2.3.3. Otros

- Google Forms

2.4. Alcance

El alcance de esta propuesta se limita principalmente al área de ventas y, de manera complementaria, al área de inventario del restaurante "El Sabor Cajabambino". Si bien otras áreas del restaurante pueden generar datos importantes, el enfoque primordial de esta aplicación recaerá en las transacciones de ventas y la gestión de inventario, dado que son los pilares fundamentales para el análisis de rentabilidad y la optimización de recursos en el negocio.

2.4.1. Dentro del alcance

- Plantear la administración del proyecto.
- Analizar los requerimientos.
- Diseñar la base de datos relacional.
- Los mockups con Figma.
- Desarrollar un sistema simplificado como propuesta

2.4.2. Fuera del alcance

- El proyecto no incluirá la exposición o visualización de información que pueda perjudicar o no se alinee con los objetivos estratégicos y la confidencialidad de la empresa. Asimismo, no se realizará el despliegue ni mantenimiento del sistema.

2.5. Tareas

Las tareas a realizar se enfocarán en el desarrollo de una aplicación web para el área de ventas del restaurante "El Sabor Cajabambino". El trabajo principal consistirá en la implementación de funcionalidades CRUD (Crear, Leer, Actualizar, Borrar) para una gestión eficiente de los procesos clave del negocio. Las tareas se dividirán en los siguientes módulos funcionales:

- **Gestión de Ventas y Pedidos:** Se desarrollará un módulo para registrar y administrar los pedidos realizados en el restaurante, así como para gestionar las reservas de mesas. Esto permitirá un seguimiento detallado de las transacciones y agilizará el servicio al cliente.
- **Control de Inventario:** Se implementará una funcionalidad para gestionar el inventario de cocina, lo que incluye el registro de ingresos

y salidas de productos. El objetivo es mantener un control preciso del stock y evitar desabastecimientos.

- **Administración de Mesas:** Se creará un módulo para visualizar y gestionar el estado de las mesas del restaurante (disponible, ocupada, reservada, mantenimiento). Esto facilitará la asignación de mesas y optimizará la atención en el local.
- **Gestión de Clientes:** Se desarrollará un apartado para almacenar y administrar la información de los clientes, lo que permitirá un mejor servicio y la posibilidad de futuras estrategias de fidelización.

CAPÍTULO 3. ANALISIS DE REQUERIMIENTOS

3.1. El Negocio

3.1.1. Misión

En Sabor Cajabambino, nuestra misión es celebrar la riqueza de la cocina de Cajabamba y Cajamarca, compartiendo con cada visitante una experiencia gastronómica auténtica y acogedora. Ofrecemos un viaje de sabores que abarca desde los platos típicos transmitidos por generaciones hasta opciones a la carta cuidadosamente elaboradas, todo en un ambiente que honra nuestra cultura, conecta a las personas y evoca la calidez de nuestra tierra.

3.1.2. Visión

Visualizamos a Sabor Cajabambino como un legado de sabor y hospitalidad donde cada experiencia gastronómica, arraigada en nuestra herencia culinaria de Cajabamba y Cajamarca, trasciende el presente y perdura en el tiempo. Buscamos expandir nuestro alcance sin perder la esencia que nos define: el sabor auténtico, la calidez de nuestra hospitalidad y el orgullo inquebrantable por nuestra identidad cultural.

3.1.3. Objetivos organizacionales

3.1.3.1. Generales

Crear experiencias gastronómicas memorables que celebren los sabores únicos de Cajabamba y Cajamarca, construyendo una base de clientes leales y asegurando el crecimiento sostenido de Sabor Cajabambino.

3.1.3.2. Específicos

- Fomentar un sentido de pertenencia fuerte en el equipo de trabajo.
- Construir una comunidad de comensales leales.
- Realizar alianzas estratégicas con proveedores que ofrezcan la mejor calidad de insumos.

3.2. Descripción de los procesos del negocio

El conjunto de procesos a modelar comprenderá la operación integral del restaurante Sabor Cajabambino:

El proceso de atención al cliente: Este proceso empieza desde el momento en que el cliente llega al restaurante, incluyendo la recepción, la asignación de mesas (con o sin reserva), la presentación del menú. la toma de pedidos, el servicio de bebidas y alimentos, la atención a consultas y solicitudes, la presentación de la cuenta, el cobro y la despedida.

El proceso de gestión de pedidos para llevar/a domicilio: Este proceso empieza desde la recepción del pedido, su registro, la comunicación a la cocina, el empaquetado adecuado, la gestión del pago y la entrega al cliente, se enfocará en la precisión, la rapidez y la calidad de la entrega.

El proceso de gestión de inventario de la cocina y el almacén: Control de stock, el registro de las salidas para la preparación de los platos y la generación de pedidos de reposición.

3.3. Problemas del negocio

La insuficiente digitalización de procesos operativos como las ventas y la gestión del inventario limitan la agilidad y la eficiencia del negocio, lo que se traduce en una capacidad restringida para optimizar la experiencia del cliente, escalar las operaciones de manera efectiva ante la diversificación de la oferta gastronómica y la futura implementación de canales digitales, y mantener una ventaja competitiva en el mercado.

3.4. Selección de los entrevistados

En este caso se ha seleccionado a José Carlos Álvarez el dueño del restaurante “El sabor Cajabambino” ya que es el único con disponibilidad para realizar las entrevistas y el recojo de datos necesarios.

3.5. Entrevista

El proceso de entrevista es fundamental para comprender a fondo las necesidades y desafíos del restaurante "El Sabor Cajabambino" en relación con sus datos. Se realizaron preguntas estructuradas y abiertas con el gerente del negocio, el Sr. José Carlos Álvarez (Dueño/Gerente General). El objetivo es indagar sobre sus tareas diarias, su perspectiva sobre el desempeño en

las áreas de interés y los retos que enfrenta en la gestión y análisis de la información. Se prestó especial atención a la toman decisiones actualmente y qué tipo de información o herramientas desearía tener en el futuro para mejorar este proceso. Esta interacción directa permitió obtener una visión cualitativa de las expectativas y las áreas de mejora desde la perspectiva de quien administra el negocio.

3.5.1. Definición de requerimientos del negocio

Entrevista para obtener información que sirva de apoyo a la toma de decisiones en el área de recursos humanos de la empresa EL SABOR CAJABAMBINO

Estimado colaborador: *El propósito de esta encuesta es comprender mejor los desafíos y necesidades relacionadas con la* ***gestión de ventas e inventario*** *en "El Sabor Cajabambino".*

La información recopilada será fundamental para el diseño y el desarrollo de un sistema de gestión web para el área de ventas que permita optimizar estos procesos, mejorar la toma de decisiones y, en última instancia, beneficiar el crecimiento y la eficiencia de nuestro restaurante.

Lugar de la entrevista: Restaurant el Sabor Cajabambino

INSTRUCCIONES: Por favor, lea cuidadosamente cada pregunta y marque con una (X) la respuesta que mejor se ajuste a su realidad, o escriba su respuesta donde sea necesario. ¡Gracias por su valioso tiempo!

1. Actualmente, ¿cómo registra y controla las ventas diarias en el restaurante?

a. *Manualmente (cuadernos, hojas de cálculo básicas)*

b. A través de un sistema de punto de venta (POS)

c. Opción 4

d. *Otros: Sistema de registro de ventas alquilado a terceros*

2. ¿Qué tan fácil le resulta obtener información sobre las ventas de días, semanas o meses anteriores?

a. Muy fácil

b. Fácil

c. *Regular*

d. Difícil

e. Muy difícil

3. ¿Con qué frecuencia analiza los datos de ventas para identificar productos más vendidos o patrones de consumo?

a. Diariamente

b. Semanalmente

c. *Mensualmente*

d. Nunca

e. *Otros: No es tanto un análisis, reviso los datos sobre todo para llevar la contabilidad y no tener problemas con SUNAT.*

4. ¿Considera que la información actual sobre ventas es suficiente para tomar decisiones sobre promociones o cambios en el menú? ¿Por qué?

a. Sí

b. *No*

La información de las ventas sirve para llevar la contabilidad. No la uso regularmente para tomar decisiones

5. ¿Cómo realiza actualmente el control de inventario de materias primas e insumos en el restaurante?

a. *Manualmente (fichas, conteo físico)*

b. A través de un sistema de inventario o un módulo de POS

c. Otros:

6. ¿Qué tan fácil es saber en cualquier momento la cantidad exacta de un ingrediente o producto disponible?

a. Muy fácil

b. Fácil

c. *Regular*

d. Difícil

e. Muy difícil

7. ¿Ha experimentado problemas de desabastecimiento o exceso de inventario de algún producto o ingrediente?

a. Sí, frecuentemente

b. *Sí, ocasionalmente*

c. No, rara vez

d. Nunca

Si su respuesta es sí, ¿cuáles han sido las consecuencias (ej. pérdidas de ventas, productos vencidos)?

El encargado de abastecer el inventario y gestionarlo es el cocinero, por lo que no sé cómo maneja todo ese proceso. Pero ha existido situaciones donde no se ha logrado un abastecimiento adecuado.

8. ¿Qué dificultades encuentra en la gestión de inventario para el restaurante (ej. seguimiento de ingredientes, control de stock, pérdidas)?

No puedo llevar un seguimiento adecuado del inventario. El sistema que alquilo tiene un apartado de inventario, pero nunca lo he usado.

9. ¿Qué piensa de la automatización de los procesos de registro de ventas e inventario en "El Sabor Cajabambino" a través de un sistema de gestión web? ¿Cómo cree que le ayudaría en su trabajo?

No tengo mucho conocimiento sobre los detalles del sistema, pero según lo que me han explicado creo que sería una herramienta muy útil para llevar una gestión mucho más práctica y eficiente de las ventas y el inventario.

10. ¿Hay alguna otra información o comentario que considere relevante para el diseño de este sistema"?

A para organizar mejor a los clientes y tener un registro más claro de las actividades en el momento, quizá una forma de visualizar el estado de las mesas.

11. Indique el rango en el que se encuentra su rendimiento laboral.

a. Del 75% - 100%

b. *Del 50% - 75%*

c. Del 25% - 50%

d. Del 0% - 25%

12. Qué piensas de la gestión de los datos de ventas e inventario en la mejora del desempeño, ¿Cómo cree Ud. que le ayudaría?

Creo que sería una herramienta interesante de implementar en mi negocio. Definitivamente, sería útil.

3.6. Resumen de los requerimientos obtenidos en la entrevista

Los requerimientos recopilados según las entrevistas, encuestas y evaluación de reportes fueron:

- Gestión de Pedidos y Reservas de Mesas: Es fundamental contar con un sistema que facilite la toma de pedidos, la asignación de mesas y la gestión de reservas. Esto mejorará la organización del servicio y agilizará la atención a los clientes, reduciendo los tiempos de espera. Mejora en el acceso a la información de ventas: Urge optimizar la facilidad de acceso y consulta de la información de ventas diarias, semanales y mensuales del sistema alquilado, ya que el acceso a datos históricos es "Regular".
- Control del Inventario de Ingredientes: Se necesita un módulo para llevar un registro preciso del inventario de cocina. El objetivo es poder registrar entradas y salidas de productos de manera sencilla, lo que permitirá tener un control del stock en tiempo real y evitar problemas de desabastecimiento o mermas innecesarias. Organización y evaluación del desempeño de meseros: Se necesita organizar mejor a los meseros y tener un registro más claro de su actividad para evaluar su desempeño.
- Administración del Estado de las Mesas: Se requiere una interfaz visual y práctica que permita al personal conocer el estado actual de cada mesa (disponible, ocupada, reservada). Esto ayudará a una mejor planificación del servicio y una asignación más eficiente de los espacios.
- Registro y Administración de Clientes: Existe la necesidad de crear un registro de clientes que capture información básica como nombres y datos de contacto. Esto servirá como una base para mejorar la personalización del servicio y para futuras campañas de fidelización.

3.7. Base de Datos Relacional

La base de datos Fuente será una base de datos relacional SQL desarrollada en SQL Server, la cual cuenta con diferentes esquemas y tablas que contienen información de cinco meses de pedidos.

Esquema GENERAL

Este esquema contiene información general y de catálogo del restaurante, incluyendo productos y la disposición de las mesas. Tablas:

- Categoria
    - Id_Categoria
    - Nombre

    - Descripcion
- Mesa
    - Id_Mesa
    - Capacidad
    - Ubicación
    - Estado
- Producto
    - Id_Producto
    - Nombre
    - Precio
    - Descripcion
    - Foto
    - Id_Categoria
    - EsPreparado
- ProductoIngrediente
    - Id_Producto
    - Id_Item
    - Cantidad

Esquema CLIENTE

Este esquema gestiona la información de los clientes y sus reservas. Tablas:

- Cliente
    - Id_Cliente
    - Nombres
    - ApellidoPaterno
    - ApellidoMaterno
    - DNI
    - Telefono
    - CorreoElectronico
    - Direccion
    - FechaDeNacimiento
- Reserva

    - Id_Reserva
    - Fecha
    - Hora
    - Id_Mesa
    - Id_Cliente
    - NumeroPersonas
    - Estado
    - Comentarios

Esquema INVENTARIO

Este esquema se encarga de la gestión de inventario de materias primas e insumos.

Tablas:

- Inventario
    - Id_Item
    - ItemNombre
    - Id_ItemCategoria
    - UnidadMedida
    - Stock
    - CostoPorUnidad
    - FechaDeExpiracion
    - NivelReorden
    - CantidadReorden
    - NecesitaReorden
- ItemCategoria
    - Id_ItemCategoria
    - Categoria
    - Descripcion

Esquema PERSONAL

Este esquema almacena la información de los empleados del restaurante. Tablas:

- Empleado

    - Id_Empleado
    - NombreCompleto
    - DNI
    - FechaNacimiento
    - Direccion
    - Telefono
    - CorreoElectronico
    - Rol
    - Turno
    - FechaContratacion
    - Salario
    - Estado
    - Usuario
    - Contrasenia

Esquema TRANSACCION

Este esquema registra todos los detalles de los pedidos y sus transacciones. Tablas:

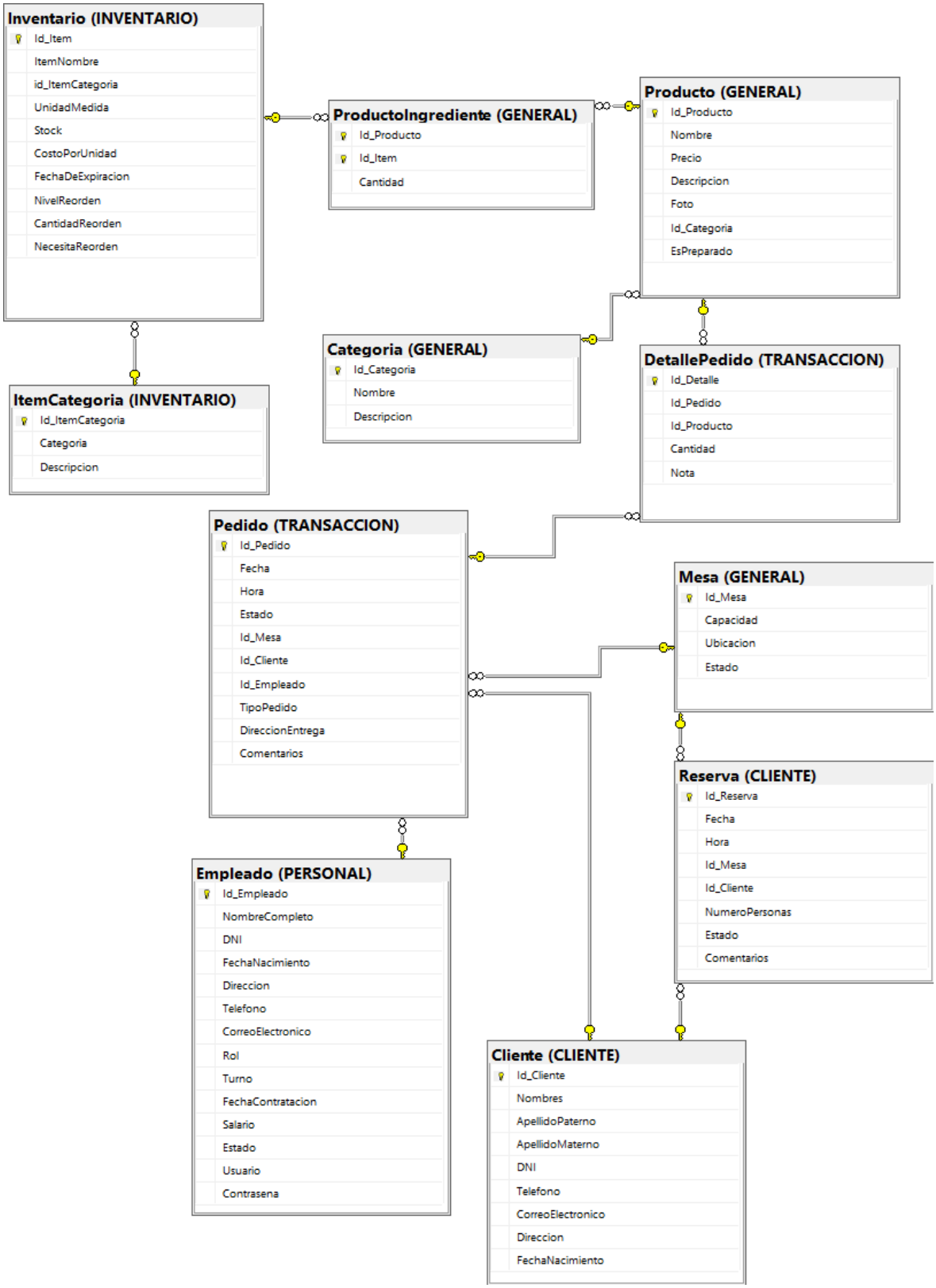
- DetallePedido
    - Id_Detalle
    - Id_Pedido
    - Id_Producto
    - Cantidad
    - PrecioUnitario
    - Nota
- Pedido
    - Id_Pedido
    - Fecha
    - Hora
    - Estado

- Id_Mesa
- Id_Cliente
- Id_Empleado
- TipoPedido
- DireccionEntrega
- Comentarios

3.8. Diagrama Relacional

Figura 1

Diagrama base de datos



CAPÍTULO 4. MARCO TEÓRICO

4.1. Sistemas de Gestión Web

Los sistemas de gestión web se definen como herramientas de software en línea diseñadas para digitalizar, optimizar y centralizar los procesos operativos de una organización, facilitando la interacción y el intercambio de información entre usuarios y empresas, estos sistemas integran hardware, software e infraestructura de red para proporcionar una experiencia en línea eficiente y segura (Silva, 2023).

En un sentido más amplio, la gestión de sitios web abarca el mantenimiento general, la seguridad y el desarrollo a largo plazo, además de la organización de la entrega de contenidos y la estrategia de marketing, las tareas esenciales de estos sistemas incluyen el mantenimiento continuo, la implementación de actualizaciones de diseño y contenido, la garantía de la seguridad de la plataforma, el soporte técnico al personal y clientes, la planificación del crecimiento futuro y la búsqueda de una experiencia de usuario consistente en todos los canales (Ibexa, 2025).

La implementación de un sistema de gestión web es crucial para las empresas, especialmente para aquellas con una presencia digital significativa, permite la preparación para el crecimiento futuro, asegura la evolución constante del sitio, mitiga los costos derivados de errores e inactividad, y promueve la eficiencia económica (Ibexa, 2025). Adoptar herramientas y prácticas de gestión web eficaces desde las etapas iniciales es fundamental para escalar un negocio de manera efectiva. Sin una gestión adecuada, los procesos pueden volverse complejos, exigentes y costosos, y los errores o la inactividad pueden tener consecuencias desastrosas, particularmente para negocios con una fuerte dependencia del comercio electrónico, la tecnología digital, a través de estas herramientas, optimiza procesos críticos, facilita la colaboración y mejora las prácticas laborales, lo que se traduce en una significativa reducción de costos operativos (Silva, 2023).

Para el restaurante "El Sabor Cajabambino", la implementación de un sistema de gestión web simplificado constituye una respuesta directa a la necesidad de optimizar procesos manuales y poco integrados en el área de ventas e inventario, estos procesos actuales generan inconsistencias en los datos y lentitud en el servicio, afectando la eficiencia operativa; este nuevo sistema transformará la gestión manual en un flujo de trabajo ágil y preciso, sentando las bases para el crecimiento futuro del negocio. La digitalización de las operaciones no es solo una

mejora operativa, sino un facilitador estratégico, al centralizar y automatizar la gestión de ventas e inventario, el restaurante no solo resuelve los problemas actuales de inconsistencia y lentitud, sino que se posiciona para manejar un mayor volumen de operaciones, diversificar su oferta y potencialmente expandir sus canales digitales, lo cual es fundamental para mantener una ventaja competitiva en un mercado dinámico, la inversión en este sistema representa, por lo tanto, una inversión en la capacidad de crecimiento y adaptación del negocio a largo plazo.

Además, la eficiencia operativa lograda mediante la digitalización sienta una base sólida para la toma de decisiones estratégicas, la información obtenida de la entrevista con el dueño del restaurante reveló que los datos de ventas se utilizaban principalmente para la contabilidad, no para tomar decisiones estratégicas sobre promociones o cambios en el menú. Esta situación revela una brecha entre la recolección de datos y su aplicación estratégica, un sistema de gestión web, al estandarizar y centralizar los datos, no solo mejora la eficiencia operativa al reducir errores y tiempos, sino que, de manera más profunda, libera el potencial de los datos. Al disponer de información precisa y accesible sobre ventas, inventario y clientes, el gerente puede evolucionar de una gestión reactiva a una proactiva, utilizando los datos para identificar patrones de consumo, optimizar el menú, planificar promociones y gestionar el personal de manera más efectiva,la eficiencia operativa es el primer paso, pero el valor real añadido reside en la capacidad de generar inteligencia de negocio que impulse decisiones estratégicas.

4.2. Bases de Datos Relacionales (RDBMS):

4.2.1. Microsoft SQL Server:

Microsoft SQL Server es un sistema de gestión de bases de datos relacionales (RDBMS) líder, ampliamente utilizado en entornos corporativos para aplicaciones de inteligencia de negocios y análisis, se basa en el lenguaje Transact-SQL y ofrece extensiones de programación propietarias para entornos locales y en la nube (Nedigital, 2024). Es reconocido como uno de los softwares de gestión de bases de datos más populares y utilizados por grandes empresas debido a sus beneficios significativos y utilidades, especialmente dada la vasta cantidad de datos e información que se maneja hoy en día (WE Educación, s.f.). SQL Server actúa como el eje central para almacenar información en bases de datos relacionales, proporcionando una gestión sencilla a través de su interfaz visual y diversas opciones y herramientas, sus funciones y características clave incluyen la

integración de datos fluida, que permite la incorporación de servicios cognitivos y Azure AI. Ofrece alta disponibilidad y resiliencia para procesos de failover más rápidos y redundancia con respaldos automáticos, minimizando el riesgo de pérdida de información, sus funcionalidades de memoria integradas aumentan la flexibilidad y simplifican la integración con la familia de servidores Microsoft, aunque es un producto de Microsoft, posee accesibilidad de código abierto, lo que lo hace accesible para desarrolladores y cuenta con el apoyo de una gran comunidad, permite la inteligencia de Big Data, consultando grandes volúmenes de datos sin replicación y es compatible con Windows, Linux y Kubernetes, lo que demuestra su compatibilidad con diversos lenguajes y plataformas (Nedigital, 2024).

Sus capacidades de base de datos inteligentes soportan memoria en memoria, memoria persistente y tempdb optimizado para memoria, en cuanto a la seguridad, cuenta con cifrado de datos y cumplimiento normativo, siendo considerado entre las plataformas más seguras, con protocolos avanzados de cifrado y autenticación multifactor, facilita el BI móvil y la escalabilidad mediante una fácil integración con dispositivos y servicios de Azure para un análisis eficiente (Nedigital, 2024). Además, proporciona herramientas de gestión como SQL Server Management Studio (SSMS) para acceder, configurar, administrar y desarrollar todos los componentes de SQL Server, ofreciendo un entorno completo con herramientas gráficas y editores de scripts, en el ámbito de funcionalidades de desarrollo, permite crear bases de datos, tablas, índices, procedimientos almacenados, vistas, y soporta tipos de datos avanzados como JSON y datos espaciales (Microsoft, 2025).

A pesar de sus múltiples ventajas, SQL Server puede presentar algunas consideraciones, los costos pueden ser variables y potencialmente elevados, especialmente en un modelo PaaS en la nube, y existe una dependencia de la conexión a internet. Para el proyecto El Sabor Cajabambino, SQL Server es el gestor de base de datos seleccionado, lo que subraya la importancia de sus características de seguridad, rendimiento y gestión de datos para el sistema del restaurante, su integración con ASP.NET Core MVC a través de Entity Framework Core es clave para la persistencia de datos, la seguridad y confiabilidad de SQL Server son cruciales para mitigar riesgos asociados a los datos sensibles del

negocio. SQL Server es reconocido como una de las plataformas más seguras, con protocolos avanzados de cifrado y autenticación multifactor (Nedigital, 2024).

Asimismo, la integración de SQL Server con el ecosistema Microsoft es un facilitador clave para el desarrollo y mantenimiento. SQL Server se integra perfectamente con herramientas de Microsoft como Azure Data Studio y Azure Portal (Nedigital, 2024). La sinergia entre SQL Server y ASP.NET Core MVC a través de Entity Framework Core permite un mapeo objeto-relacional eficiente y una interacción de datos simplificada, acelerando el desarrollo y facilitando el mantenimiento futuro del sistema, esta elección tecnológica no solo satisface los requisitos funcionales, sino que optimiza el proceso de desarrollo y la sostenibilidad del proyecto.

4.3. Operaciones CRUD

CRUD es un acrónimo fundamental en el desarrollo de *software* que representa las cuatro operaciones básicas que se realizan sobre los datos: Crear (*Create*), Leer (*Read*), Actualizar (*Update*) y Eliminar (*Delete*), estas operaciones son esenciales para la manipulación y gestión de datos en bases de datos y otros sistemas de almacenamiento (Ebis Education, 2023).

La definición de cada operación es la siguiente:

- **Crear (Create):** Se refiere a la inserción de nuevos datos o registros en una base de datos (Conecta Software, 2024). Por ejemplo, en el contexto del restaurante, esto implica añadir un nuevo cliente o un nuevo producto al sistema.
- **Leer (Read):** Permite consultar o recuperar datos existentes de una base de datos sin modificarlos (Conecta Software, 2024). Esto incluye la visualización de listas de pedidos, la consulta del stock de ingredientes o el monitoreo del estado de las mesas.
- **Actualizar (Update):** Consiste en modificar datos ya existentes en una base de datos (Conecta Software, 2024). Un ejemplo práctico sería cambiar el estado de un pedido (por ejemplo, de "en preparación" a "entregado") o actualizar la cantidad de

  stock de un ingrediente.
- **Eliminar (Delete):** Permite borrar registros o datos que ya no son necesarios de la base de datos (Conecta Software, 2024). Esto podría

aplicarse a la eliminación de un producto descontinuado o un registro de mesa que ya no es relevante.

Sus ventajas incluyen la simplicidad y facilidad de uso, ya que el concepto es intuitivo y accesible para desarrolladores de todos los niveles, lo que acelera los ciclos de desarrollo, proporcionan estandarización, ofreciendo un conjunto uniforme de operaciones para la manipulación de datos, lo que simplifica la comunicación entre desarrolladores y mejora la mantenibilidad y legibilidad del código (Conecta Software, 2024). Su flexibilidad y escalabilidad permiten aplicarlas a diversos modelos y estructuras de datos (bases de datos relacionales, NoSQL), adaptando las operaciones a diferentes requisitos del proyecto y soportando la escalabilidad de las aplicaciones, además, contribuyen a la integridad y seguridad de los datos al proporcionar formas controladas de manipular la información, permitiendo la implementación de reglas de validación y mecanismos de autenticación/autorización para proteger datos sensibles, las operaciones CRUD están diseñadas para ser eficientes y de alto rendimiento, ya que los sistemas de bases de datos están optimizados para manejarlas eficazmente, lo que contribuye al rendimiento general de las aplicaciones, promueven la reutilización de código, encapsulando la lógica de manipulación de datos y reduciendo el esfuerzo de desarrollo (Ebis Education, 2023). Finalmente, permiten la automatización de procesos, eliminando la necesidad de manipulación manual de datos, lo que reduce errores y aumenta la eficiencia (Conecta Software, 2024).

El núcleo funcional del sistema de gestión web simplificado para "El Sabor Cajabambino" se basa en la implementación de operaciones CRUD en cada uno de los módulos principales: gestión de pedidos y reservas, control de inventario, monitoreo de mesas y administración de clientes, el proyecto busca implementar estas funcionalidades para una gestión eficiente de los procesos clave del negocio, las operaciones CRUD actúan como el lenguaje universal que habilita la interoperabilidad y la automatización de procesos de negocio. No son solo un conjunto de funciones básicas, sino la base de las APIs y la integración de datos (Conecta Software, 2024). En el contexto del restaurante, esto se traduce en la capacidad del sistema para intercambiar información sin intervención manual (Conecta Software, 2024). Por ejemplo, al "Crear" un nuevo pedido, este puede automáticamente "Actualizar" el inventario de ingredientes, si se "Actualiza" el

estado de una mesa, esta información es "Leída" por el panel de visualización, esta estandarización permite que los diferentes módulos del sistema (pedidos, inventario, mesas, clientes) se comuniquen de manera coherente y predecible, esto significa que el sistema no es una colección de funciones aisladas, sino un ecosistema integrado donde los datos fluyen de manera automática y consistente, lo cual es fundamental para la eficiencia operativa de un restaurante que maneja múltiples flujos de trabajo simultáneamente.

La siguiente tabla ilustra cómo las operaciones CRUD se mapean directamente a las funcionalidades específicas implementadas en el sistema de gestión para El Sabor Cajabambino, demostrando la aplicación práctica de este concepto fundamental.

Operación CRUD	**Descripción General**	**Funcionalidad en "El Sabor Cajabambino"**	**Módulo del Sistema**
Crear (Create)	Inserción de nuevos registros de datos.	Registro de nuevos pedidos, adición de nuevos clientes, registro de ingresos de inventario.	Gestión de Pedidos, Administración de Clientes, Control de Inventario
Leer (Read)	Consulta y recuperación de datos existentes.	Visualización de la lista de pedidos, consulta de stock de ingredientes, monitoreo del estado de las mesas, búsqueda de información de clientes.	Gestión de Pedidos, Control de Inventario, Administración de Mesas, Administración de Clientes
Actualizar (Update)	Modificación de datos existentes.	Cambio de estado de un pedido (ej. "en preparación" a	Gestión de Pedidos, Control de Inventario,

		"entregado"), actualización de cantidad de stock de un ingrediente, cambio de estado de una mesa (ej. "libre" a "ocupada"), modificación de datos de un cliente.	Administración de Mesas, Administración de Clientes
Eliminar (Delete)	Borrado de registros de datos.	Cancelación de un pedido, eliminación de un producto descontinuado, remoción de un registro de cliente.	Gestión de Pedidos, Control de Pedidos, Administración de Clientes

4.4. Arquitectura de Software

4.4.1. Patrón Modelo-Vista-Controlador (MVC):

El patrón Modelo-Vista-Controlador (MVC) es un patrón arquitectónico de *software* ampliamente utilizado en el desarrollo de interfaces de usuario, que divide una aplicación en tres componentes interconectados: Modelo, Vista y Controlador (AlexHost, 2024). Esta separación de intereses es una de sus principales ventajas, permitiendo que el Modelo sea compilado y probado con independencia de la presentación visual (Microsoft Learn, 2024).

Los componentes y sus responsabilidades son los siguientes:

- **Modelo (Model):** Representa los datos de la aplicación, la lógica de negocio y las reglas, es responsable de gestionar directamente los datos, recuperarlos de la base de datos, realizar operaciones sobre ellos y notificarse a sí mismo o a la Vista cuando los datos cambian (AlexHost, 2024). También encapsula cualquier lógica de implementación para conservar el

estado de la aplicación (Microsoft Learn, 2024). En vistas fuertemente tipadas, se utilizan a menudo ViewModel para contener los datos a mostrar (Microsoft Learn, 2024).

- **Vista (View):** Es la interfaz de usuario de la aplicación, encargada de presentar el contenido, recibe los datos del Modelo y los muestra al usuario, pero no contiene lógica de negocio (AlexHost, 2024). En ASP.NET Core MVC, las vistas utilizan el motor de vistas Razor para incrustar código.NET en formato HTML, y deben contener la mínima lógica posible, enfocada solo en la presentación (Microsoft Learn, 2024).
- **Controlador (Controller):** Actúa como intermediario entre el Modelo y la Vista (AlexHost, 2024). Maneja las entradas del usuario, las procesa (llamando a las funciones apropiadas en el Modelo) y determina qué Vista debe mostrarse, recibe solicitudes HTTP, interactúa con el Modelo para actualizar datos y luego actualiza la Vista con los nuevos datos (AlexHost, 2024).

El funcionamiento de MVC se describe de la siguiente manera: cuando un usuario interactúa con la aplicación (por ejemplo, envía un formulario), el Controlador recibe la solicitud, procesa la entrada, interactúa con el Modelo para actualizar los datos, y una vez que el Modelo se actualiza, la Vista se actualiza con los nuevos datos y se muestra al usuario (AlexHost, 2024).

Las ventajas de MVC son diversas:

La separación de intereses impone una clara distinción entre datos (Modelo), capa de presentación (Vista) y lógica (Controlador), lo que facilita la gestión y modificación de la, esto permite a los desarrolladores trabajar en distintas partes de forma independiente, fomenta la

reutilización de componentes; el mismo Modelo puede usarse con diferentes Vistas, y la Vista puede cambiarse sin alterar la lógica subyacente, la separación de componentes mejora la mantenibilidad, ya que los cambios en una parte no afectan a las demás, facilitando la depuración y prueba, facilita la escalabilidad de la aplicación al permitir añadir nuevas funcionalidades sin afectar la arquitectura existente, finalmente, facilita las pruebas unitarias dado que la lógica de negocio reside en el Modelo, permitiendo probar los componentes Modelo y Controlador independientemente de la Vista (AlexHost, 2024).

Sin embargo, MVC también presenta algunas desventajas. Para aplicaciones pequeñas, puede introducir una complejidad innecesaria al requerir la gestión de múltiples archivos, la curva de aprendizaje puede ser desafiante para principiantes debido a la interacción entre componentes y conceptos como el enrutamiento, los frameworks MVC a menudo requieren una cantidad considerable de código repetitivo (*boilerplate*) para la configuración inicial, a veces, los controladores pueden acoplarse demasiado a las vistas que controlan, lo que puede limitar la flexibilidad y llevar a "controladores inflados" o acoplamiento estrecho entre Controlador y Vista, finalmente, la separación de lógica puede introducir una sobrecarga de rendimiento adicional para la comunicación entre componentes, aunque el impacto varía (AlexHost, 2024).

4.5. ASP.NET Core MVC

ASP.NET Core MVC es un framework de desarrollo web de código abierto, modular y multiplataforma, diseñado para construir aplicaciones web modernas basadas en la nube y APIs, representa un rediseño completo del framework original ASP.NET, unificando los previamente separados ASP.NET MVC y ASP.NET Web API en un único modelo de programación, a partir de noviembre de 2020, Microsoft simplificó el nombre, eliminando Core para las nuevas versiones, que ahora se denominan simplemente .NET seguido de un número de versión (Umbraco, 2023).

Las características clave de ASP.NET Core MVC incluyen su naturaleza multiplataforma, que le permite desarrollarse y ejecutarse en Windows, macOS, Linux y Docker, a diferencia del ASP.NET original que era solo para Windows, ofrece un rendimiento mejorado gracias a su diseño modular, un menor tamaño y una arquitectura optimizada, lo que resulta en aplicaciones que manejan más solicitudes por segundo y tienen tiempos de inicio más rápidos, el desarrollo simplificado es posible gracias a un framework unificado que combina MVC y Web API en una única canalización su modularidad se evidencia al distribuirse como paquetes NuGet, permitiendo un enfoque de desarrollo más granular y flexible, posee un sistema de configuración basada en entorno listo para la nube que se adapta al entorno de despliegue, y cuenta con soporte integrado para inyección de dependencias, una característica central que simplifica la gestión de dependencias dentro de las aplicaciones, finalmente, ofrece seguridad mejorada en comparación con versiones anteriores de ASP.NET (Umbraco, 2023).

El rendimiento y la modularidad de ASP.NET Core MVC son impulsores de la eficiencia operativa y la escalabilidad del sistema. ASP.NET Core MVC ofrece rendimiento mejorado debido a su diseño modular y arquitectura optimizada, además, es un framework modular distribuido como paquetes NuGet, para el restaurante El Sabor Cajabambino, un sistema de gestión de ventas requiere rapidez en el procesamiento de pedidos y actualización de inventarios, especialmente en momentos de alta demanda, el rendimiento superior de ASP.NET Core MVC asegura que las operaciones CRUD se realicen de manera eficiente, lo cual es crítico para la agilidad y eficiencia del negocio. La modularidad, por su parte, permite que el sistema sea más ligero y que solo se incluyan las funcionalidades necesarias, lo que contribuye a un menor consumo de recursos y una mayor velocidad, esta combinación de alto rendimiento y diseño modular es un factor clave para la escalabilidad del sistema, permitiendo que el restaurante maneje un volumen creciente de operaciones sin experimentar degradación significativa en el servicio, lo que es esencial para su crecimiento.

CAPÍTULO 5. DESARROLLO DEL SISTEMA DE GESTIÓN WEB

5.1. Entorno y Tecnologías de Desarrollo

- **Lenguaje de Programación:** C#.
- **Framework:** ASP.NET Core MVC.
- **Gestor de Base de Datos:** SQL Server.
- **ORM (Mapeador Objeto-Relacional): Entity Framework Core**, que es la tecnología que te permite interactuar con la base de datos a través de tus clases de C#.
- **Entorno de Desarrollo Integrado (IDE):** Microsoft Visual Studio Community 2022 Versión 17.14.5

5.2. Implementación de la Arquitectura MVC

- **Modelos (Models):**
- *Models/Categoria.cs*

```
public partial class Categoria
{
    public int IdCategoria { get; set; }
    public string Nombre { get; set; } = null!;
    public string? Descripcion { get; set; }
    public virtual ICollection<Producto> Productos { get; set; } = new List<Producto>();
}
```

- *Models/Cliente.cs*

```
public partial class Cliente
{
    public int IdCliente { get; set; }
    public string Nombres { get; set; } = null!;
    public string ApellidoPaterno { get; set; } = null!;
    public string ApellidoMaterno { get; set; } = null!;
    public string Dni { get; set; } = null!;
    public string? Telefono { get; set; }
    public string? CorreoElectronico { get; set; }
    public string? Direccion { get; set; }
    public DateOnly? FechaNacimiento { get; set; }
    public virtual ICollection<Pedido> Pedidos { get; set; } = new List<Pedido>();
    public virtual ICollection<Reserva> Reservas { get; set; } = new List<Reserva>();
}
```

- *Models/DetallePedido.cs*

```
public partial class DetallePedido
{
    public int IdDetalle { get; set; }
    public int IdPedido { get; set; }
    public int IdProducto { get; set; }
    public int Cantidad { get; set; }
    public string? Nota { get; set; }
    public virtual Pedido IdPedidoNavigation { get; set; } = null!;
```

```
    public virtual Producto IdProductoNavigation { get; set; } = null!;
}
```

- *Models/Empleado.cs*

```
public partial class Empleado
{
    public int IdEmpleado { get; set; }
    public string NombreCompleto { get; set; } = null!;
    public string Dni { get; set; } = null!;
    public DateOnly? FechaNacimiento { get; set; }
    public string? Direccion { get; set; }
    public string? Telefono { get; set; }
    public string? CorreoElectronico { get; set; }
    public string Rol { get; set; } = null!;
    public string Turno { get; set; } = null!;
    public DateOnly? FechaContratacion { get; set; }
    public decimal? Salario { get; set; }
    public string? Estado { get; set; }
    public string Usuario { get; set; } = null!;
    public string Contrasena { get; set; } = null!;
    public virtual ICollection<Pedido> Pedidos { get; set; } = new List<Pedido>();
}
```

- *Models/Inventario.cs*

```
public partial class Inventario
{
    public int IdItem { get; set; }
    public string ItemNombre { get; set; } = null!;
    public int IdItemCategoria { get; set; }
    public string? UnidadMedida { get; set; }
    public decimal? Stock { get; set; }
    public decimal CostoPorUnidad { get; set; }
    public DateOnly? FechaDeExpiracion { get; set; }
    public decimal? NivelReorden { get; set; }
    public decimal? CantidadReorden { get; set; }
    public bool? NecesitaReorden { get; set; }
    public virtual ItemCategorium? IdItemCategoriaNavigation { get; set; }
    public virtual ICollection<ProductoIngrediente> ProductoIngredientes { get; set; } = new
List<ProductoIngrediente>();
}
```

- *Models/ItemCategorium.cs*

```
public partial class ItemCategorium
{
    public int IdItemCategoria { get; set; }
    public string Categoria { get; set; } = null!;
    public string? Descripcion { get; set; }
    public virtual ICollection<Inventario> Inventarios { get; set; } = new List<Inventario>();
}
```

- *Models/Mesa.cs*

```
public partial class Mesa
{
    public int IdMesa { get; set; }
    public int? Capacidad { get; set; }
    public string? Ubicacion { get; set; }
    public string? Estado { get; set; }
    public virtual ICollection<Pedido> Pedidos { get; set; } = new List<Pedido>();
    public virtual ICollection<Reserva> Reservas { get; set; } = new List<Reserva>();
}
```

- *Models/Pedido.cs*

```
public partial class Pedido
{
    public int IdPedido { get; set; }
    public DateOnly Fecha { get; set; }
    public TimeOnly Hora { get; set; }
    public string Estado { get; set; } = null!;
    public int? IdMesa { get; set; }
    public int? IdCliente { get; set; }
    public int? IdEmpleado { get; set; }
    public string? TipoPedido { get; set; }
    public string? DireccionEntrega { get; set; }
    public string? Comentarios { get; set; }
    public virtual ICollection<DetallePedido> DetallePedidos { get; set; } = new List<DetallePedido>();
    public virtual Cliente? IdClienteNavigation { get; set; }
    public virtual Empleado? IdEmpleadoNavigation { get; set; }
    public virtual Mesa? IdMesaNavigation { get; set; } }
```

- *Models/Producto.cs*

```
public partial class Producto
{
    public int IdProducto { get; set; }
    public string Nombre { get; set; } = null!;
    public decimal Precio { get; set; }
    public string? Descripcion { get; set; }
    public string? Foto { get; set; }
    public int IdCategoria { get; set; }
    public bool? EsPreparado { get; set; }
    public virtual ICollection<DetallePedido> DetallePedidos { get; set; } = new List<DetallePedido>();
    public virtual Categoria IdCategoriaNavigation { get; set; } = null!;
    public virtual ICollection<ProductoIngrediente> ProductoIngredientes { get; set; } = new
List<ProductoIngrediente>();
}
```

- *Models/ProductoIngrediente.cs*

```
public partial class ProductoIngrediente
{
    public int IdProducto { get; set; }
    public int IdItem { get; set; }
    public decimal Cantidad { get; set; }
    public virtual Inventario IdItemNavigation { get; set; } = null!;
    public virtual Producto IdProductoNavigation { get; set; } = null!;
}
```

- *Models/Reserva.cs*

```
public partial class Reserva
{
    public int IdReserva { get; set; }
    public DateOnly Fecha { get; set; }
    public TimeOnly Hora { get; set; }
    public int? IdMesa { get; set; }
    public int? IdCliente { get; set; }
    public int NumeroPersonas { get; set; }
    public string? Estado { get; set; }
    public string? Comentarios { get; set; }
    public virtual Cliente? IdClienteNavigation { get; set; }
    public virtual Mesa? IdMesaNavigation { get; set; }
}
```

Controladores (Controllers): Todos los controladores de la aplicación, como InventarioController, ClientesController y PedidosController, comparten una estructura y propósito comunes, lo que permite un enfoque de desarrollo modular y coherente. Cada uno es responsable de gestionar un modelo de datos específico (inventario, clientes, pedidos) y actúa como un punto de entrada para las interacciones del usuario. Esta uniformidad se basa en el patrón de diseño Model-View-Controller (MVC), donde el controlador maneja las solicitudes, utiliza un contexto de base de datos para acceder y manipular la información, y devuelve las vistas correspondientes al usuario.

La generalización de los controladores se logra a través de la implementación consistente de métodos de acción para las operaciones básicas CRUD (Crear, Leer, Actualizar, Eliminar). Por ejemplo, cada controlador tiene una acción Index para listar todos los elementos, una Details para ver uno en particular, y pares de acciones Create, Edit y Delete (tanto GET como POST) para gestionar el ciclo de vida de los datos. Esta estandarización no solo simplifica el desarrollo y mantenimiento, sino que también facilita la comprensión del código, ya que la lógica para gestionar un módulo es predecible y consistente con la de los demás.

- *Controllers/LoginController.cs*

```
public class LoginController : Controller
{
    private readonly RestauranteProgramacionIiContext _context;
    public LoginController(RestauranteProgramacionIiContext context)
    {
        _context = context;
    }
    [HttpGet]
    public IActionResult Login()
    {
        return View();
    }
    [HttpPost]
    [ValidateAntiForgeryToken]
    public IActionResult Login(LoginViewModel model)
    {
        if (ModelState.IsValid)
        {
            var empleado = _context.Empleados
                .FirstOrDefault(e => e.Usuario == model.Usuario && e.Contrasena == model.Contrasena);
            if (empleado != null)
            {
                return RedirectToAction("Index", "Home");
            }
            ModelState.AddModelError(string.Empty, "Usuario o contraseña incorrectos.");
        }
        return View(model);
    }
```

- *Controllers/CategoriaController.cs*

```
public class CategoriaController : Controller
{
    private readonly RestauranteProgramacionIiContext _context;
    public CategoriaController(RestauranteProgramacionIiContext context)
    {
        _context = context;    }
    // GET: Categoria
    public async Task<IActionResult> Index()
    {
        return View(await _context.Categoria.ToListAsync());
    }
    // GET: Categoria/Details/5
    public async Task<IActionResult> Details(int? id)
    {
        if (id == null)
        {
            return NotFound();
        }
        var categoria = await _context.Categoria
            .FirstOrDefaultAsync(m => m.IdCategoria == id);
        if (categoria == null)
        {
            return NotFound();
        }
        return View(categoria);
    }
    // GET: Categoria/Create
    public IActionResult Create()
    {
        return View();
    }
    // POST: Categoria/Create
    [HttpPost]
    [ValidateAntiForgeryToken]
    public async Task<IActionResult> Create([Bind("IdCategoria,Nombre,Descripcion")] Categoria categoria)
    {
        if (ModelState.IsValid)
        {
            _context.Add(categoria);
            await _context.SaveChangesAsync();
            return RedirectToAction(nameof(Index));
        }
        return View(categoria);
    }
    // GET: Categoria/Edit/5
    public async Task<IActionResult> Edit(int? id)
    {
        if (id == null)
        {
            return NotFound();
        }
        var categoria = await _context.Categoria.FindAsync(id);
        if (categoria == null)
        {
            return NotFound();
        }
        return View(categoria);
    }
    // POST: Categoria/Edit/5
    [HttpPost]
    [ValidateAntiForgeryToken]
    public async Task<IActionResult> Edit(int id, [Bind("IdCategoria,Nombre,Descripcion")] Categoria categoria)
    {
```

```
            if (id != categoria.IdCategoria)
            {
                return NotFound();
            }
            if (ModelState.IsValid)
            {
                try
                {
                    _context.Update(categoria);
                    await _context.SaveChangesAsync();
                }
                catch (DbUpdateConcurrencyException)
                {
                    if (!CategoriaExists(categoria.IdCategoria))
                    {
                        return NotFound();
                    }
                    else
                    {
                        throw;
                    }
                }
                return RedirectToAction(nameof(Index));
            }
            return View(categoria);
        }
        // GET: Categoria/Delete/5
        public async Task<IActionResult> Delete(int? id)
        {
            if (id == null)
            {
                return NotFound();
            }
            var categoria = await _context.Categoria
                .FirstOrDefaultAsync(m => m.IdCategoria == id);
            if (categoria == null)
            {
                return NotFound();
            }
            return View(categoria);
        }
        // POST: Categoria/Delete/5
        [HttpPost, ActionName("Delete")]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> DeleteConfirmed(int id)
        {
            var categoria = await _context.Categoria.FindAsync(id);
            if (categoria != null)
            {
                _context.Categoria.Remove(categoria);
            }
            await _context.SaveChangesAsync();
            return RedirectToAction(nameof(Index));
        }
        private bool CategoriaExists(int id)
        {
            return _context.Categoria.Any(e => e.IdCategoria == id);
        }
    }
```

- *Controllers/ClienteController.cs*

```
public class ClienteController : Controller
 {
```

```
    private readonly RestauranteProgramacionIiContext _context;

    public ClienteController(RestauranteProgramacionIiContext context)
    {
        _context = context;
    }
    // GET: Cliente
    public async Task<IActionResult> Index()
    {
        return View(await _context.Clientes.ToListAsync());
    }
    // GET: Cliente/Details/5
    public async Task<IActionResult> Details(int? id)
    {
        if (id == null)
        {
            return NotFound();
        }

        var cliente = await _context.Clientes
            .FirstOrDefaultAsync(m => m.IdCliente == id);
        if (cliente == null)
        {
            return NotFound();
        }

        return View(cliente);
    }

    // GET: Cliente/Create
    public IActionResult Create()
    {
        return View();
    }
    // POST: Cliente/Create
    [HttpPost]
    [ValidateAntiForgeryToken]
    public async Task<IActionResult>
Create([Bind("IdCliente,Nombres,ApellidoPaterno,ApellidoMaterno,Dni,Telefono,CorreoElectronico,Direccion,FechaNacimie
nto")] Cliente cliente)
    {
        if (ModelState.IsValid)
        {
            _context.Add(cliente);
            await _context.SaveChangesAsync();
            return RedirectToAction(nameof(Index));
        }
        return View(cliente);
    }

    // GET: Cliente/Edit/5
    public async Task<IActionResult> Edit(int? id)
    {
        if (id == null)
        {
            return NotFound();
        }

        var cliente = await _context.Clientes.FindAsync(id);
        if (cliente == null)
        {
            return NotFound();
        }
        return View(cliente);
    }
    // POST: Cliente/Edit/5
    [HttpPost]
    [ValidateAntiForgeryToken]
    public async Task<IActionResult> Edit(int id,
[Bind("IdCliente,Nombres,ApellidoPaterno,ApellidoMaterno,Dni,Telefono,CorreoElectronico,Direccion,FechaNacimiento")]
Cliente cliente)
    {
        if (id != cliente.IdCliente)
        {
            return NotFound();
        }

        if (ModelState.IsValid)
        {
            try
            {
                _context.Update(cliente);
                await _context.SaveChangesAsync();
            }
            catch (DbUpdateConcurrencyException)
            {
```

```
                if (!ClienteExists(cliente.IdCliente))
                {
                    return NotFound();
                }
                else
                {
                    throw;
                }
            }
            return RedirectToAction(nameof(Index));
        }
        return View(cliente);
    }

    // GET: Cliente/Delete/5
    public async Task<IActionResult> Delete(int? id)
    {
        if (id == null)
        {
            return NotFound();
        }

        var cliente = await _context.Clientes
            .FirstOrDefaultAsync(m => m.IdCliente == id);
        if (cliente == null)
        {
            return NotFound();
        }

        return View(cliente);
    }
    // POST: Cliente/Delete/5
    [HttpPost, ActionName("Delete")]
    [ValidateAntiForgeryToken]
    public async Task<IActionResult> DeleteConfirmed(int id)
    {
        var cliente = await _context.Clientes.FindAsync(id);
        if (cliente != null)
        {
            _context.Clientes.Remove(cliente);
        }

        await _context.SaveChangesAsync();
        return RedirectToAction(nameof(Index));
    }

    private bool ClienteExists(int id)
    {
        return _context.Clientes.Any(e => e.IdCliente == id);
    }
}
```

- *Controllers/EmpleadoController.cs*

```
    public class EmpleadoController : Controller
    {
            private readonly RestauranteProgramacionIiContext _context;

            public EmpleadoController(RestauranteProgramacionIiContext context)
            {
                    _context = context;
            }
            // GET: Empleado
            public async Task<IActionResult> Index()
            {
                    return View(await _context.Empleados.ToListAsync());
            }
            // GET: Empleado/Details/5
            public async Task<IActionResult> Details(int? id)
            {
                    if (id == null)
                    {
                    return NotFound();
                    }

                    var empleado = await _context.Empleados
                    .FirstOrDefaultAsync(m => m.IdEmpleado == id);
```

```
            if (empleado == null)
            {
            return NotFound();
            }

            if (Request.Headers["X-Requested-With"] == "XMLHttpRequest")
            {
            return PartialView("_Details", empleado); // Retorna una vista parcial para solicitudes AJAX
            }
            // si no es AJAX, devolver vista completa
            return View(empleado);
        }
        // GET: Empleado/Create
        public IActionResult Create()
        {
            var roles = _context.Empleados.Select(e => e.Rol).Distinct().ToList();
            ViewData["Roles"] = new SelectList(roles);
            ViewData["Turnos"] = new SelectList(new[] { "Completo", "Medio tiempo" });
            ViewData["Estados"] = new SelectList(new[] { "Activo", "Vacaciones", "Despedido" });
            return View();
        }
        // POST: Empleado/Create
        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult>
        Create([Bind("IdEmpleado,NombreCompleto,Dni,FechaNacimiento,Direccion,Telefono,CorreoElectronico,
        Rol,Turno,FechaContratacion,Salario,Estado,Usuario,Contrasena")] Empleado empleado)
        {
            if (ModelState.IsValid)
            {
            _context.Add(empleado);
            await _context.SaveChangesAsync();
            return RedirectToAction(nameof(Index));
            }
            var roles = _context.Empleados.Select(e => e.Rol).Distinct().ToList();
            ViewData["Roles"] = new SelectList(roles);
            ViewData["Turnos"] = new SelectList(new[] { "Completo", "Medio tiempo" });
            ViewData["Estados"] = new SelectList(new[] { "Activo", "Vacaciones", "Despedido" });
            return View(empleado);
        }
        // GET: Empleado/Edit/5
        public async Task<IActionResult> Edit(int? id)
        {
            if (id == null)
            {
            return NotFound();
            }

            var empleado = await _context.Empleados.FindAsync(id);
            if (empleado == null)
            {
            return NotFound();
            }
            var roles = await _context.Empleados.Select(e => e.Rol).Distinct().ToListAsync();
            ViewData["Roles"] = new SelectList(roles, empleado.Rol);
            ViewData["Turnos"] = new SelectList(new[] { "Completo", "Medio tiempo" }, empleado.Turno);
            ViewData["Estados"] = new SelectList(new[] { "Activo", "Vacaciones", "Despedido" },
            empleado.Estado);
            return View(empleado);
        }
        // POST: Empleado/Edit/5
        [HttpPost]
        [ValidateAntiForgeryToken]
```

```
public async Task<IActionResult> Edit(int id,
[Bind("IdEmpleado,NombreCompleto,Dni,FechaNacimiento,Direccion,Telefono,CorreoElectronico,Rol,Turn
o,FechaContratacion,Salario,Estado,Usuario,Contrasena")] Empleado empleado)
{
        if (id != empleado.IdEmpleado)
        {
        return NotFound();
        }

        if (ModelState.IsValid)
        {
        try
        {
                _context.Update(empleado);
                await _context.SaveChangesAsync();
                return RedirectToAction(nameof(Index)); // Redirigir a la acción Index después de
                guardar los cambios
        }
        catch (DbUpdateConcurrencyException)
        {
                if (!EmpleadoExists(empleado.IdEmpleado))
                {
                        return NotFound();
                }
                else
                {
                        throw;
                }
        }
        }
        var roles = _context.Empleados.Select(e => e.Rol).Distinct().ToList();
        ViewData["Roles"] = new SelectList(roles, empleado.Rol);
        ViewData["Turnos"] = new SelectList(new[] { "Completo", "Medio tiempo" }, empleado.Turno);
        ViewData["Estados"] = new SelectList(new[] { "Activo", "Vacaciones", "Despedido" },
        empleado.Estado);
        return View(empleado);
}
// GET: Empleado/Delete/5
public async Task<IActionResult> Delete(int? id)
{
        if (id == null)
        {
        return NotFound();
        }

        var empleado = await _context.Empleados
        .FirstOrDefaultAsync(m => m.IdEmpleado == id);
        if (empleado == null)
        {
        return NotFound();
        }

        return View(empleado);
}
// POST: Empleado/Delete/5
[HttpPost, ActionName("Delete")]
[ValidateAntiForgeryToken]
public async Task<IActionResult> DeleteConfirmed(int id)
{
        var empleado = await _context.Empleados.FindAsync(id);
        if (empleado != null)
        {
        _context.Empleados.Remove(empleado);
        }
```

```
                    await _context.SaveChangesAsync();
                    return RedirectToAction(nameof(Index));
                }
                // POST: Empleado/Buscar
                [HttpGet]
                public async Task<IActionResult> Buscar(string searchTerm)
                {
                    if (string.IsNullOrWhiteSpace(searchTerm))
                    {
                    return Json(await _context.Empleados.ToListAsync());
                    }

                    var empleados = await _context.Empleados
                    .Where(e => e.NombreCompleto.Contains(searchTerm) ||
                                e.Dni.Contains(searchTerm) ||
                                e.Rol.Contains(searchTerm) ||
                                (e.Telefono != null && e.Telefono.Contains(searchTerm)))
                    .ToListAsync();

                    if (Request.Headers["X-Requested-With"] == "XMLHttpRequest")
                    {
                    return Json(empleados);
                    }

                    return View("Index", empleados);
                }
                private bool EmpleadoExists(int id)
                {
                    return _context.Empleados.Any(e => e.IdEmpleado == id);
                }
}
```

- *Controllers/InventarioController.cs*

```
public class InventarioController : Controller
{
    private readonly RestauranteProgramacionIiContext _context;
    public IActionResult inventario_prueba()
    {
        return View();
    }
    public InventarioController(RestauranteProgramacionIiContext context)
    {
        _context = context;
    }
    // GET: Inventario
    public async Task<IActionResult> Index()
    {
        var restauranteProgramacionIiContext = _context.Inventarios.Include(i => i.IdItemCategoriaNavigation);
        return View(await restauranteProgramacionIiContext.ToListAsync());
    }
    // GET: Inventario/Details/5
    public async Task<IActionResult> Details(int? id)
    {
        if (id == null)
        {
            return NotFound();
        }
        var inventario = await _context.Inventarios
            .Include(i => i.IdItemCategoriaNavigation)
            .FirstOrDefaultAsync(m => m.IdItem == id);
        if (inventario == null)
        {
```

```
                return NotFound();
            }
            if (Request.Headers["X-Requested-With"] == "XMLHttpRequest")
            {
                return PartialView("_Details", inventario);
            }

            return View(inventario);
        }
        // GET: Inventario/Create
        public IActionResult Create()
        {
            ViewData["IdItemCategoria"] = new SelectList(_context.ItemCategoria, "IdItemCategoria", "Categoria");
            return View();
        }

        // POST: Inventario/Create
        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult>
Create([Bind("IdItem,ItemNombre,IdItemCategoria,UnidadMedida,Stock,CostoPorUnidad,FechaDeExpiracion,NivelRe
orden,CantidadReorden,NecesitaReorden")] Inventario inventario)
        {
            if (ModelState.IsValid)
            {
                _context.Add(inventario);
                await _context.SaveChangesAsync();
                return RedirectToAction(nameof(Index));
            }
            //
            ViewData["IdItemCategoria"] = new SelectList(_context.ItemCategoria, "IdItemCategoria", "Categoria",
inventario.IdItemCategoria);
            return View(inventario);
        }
        // GET: Inventario/Edit/5
        public async Task<IActionResult> Edit(int? id)
        {
            ModelState.Remove("IdItemCategoriaNavigation");
            if (id == null)
            {
                return NotFound();
            }
            var inventario = await _context.Inventarios
        .Include(i => i.IdItemCategoriaNavigation)
        .FirstOrDefaultAsync(m => m.IdItem == id);
            if (inventario == null)
            {
                return NotFound();
            }
            // Modificar para mostrar el nombre de la categoría en lugar del ID
            //esto cambio
            ViewData["IdItemCategoria"] = new SelectList(_context.ItemCategoria, "IdItemCategoria", "Categoria",
inventario.IdItemCategoria);
            return View(inventario);
        }
        // POST: Inventario/Edit/5
        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> Edit(int id,
[Bind("IdItem,ItemNombre,IdItemCategoria,UnidadMedida,Stock,CostoPorUnidad,FechaDeExpiracion,NivelReorden,C
antidadReorden,NecesitaReorden")] Inventario inventario)
        {
            ModelState.Remove("IdItemCategoriaNavigation");
            if (id != inventario.IdItem)
```

```
            {
                return NotFound();
            }

            if (ModelState.IsValid)
            {
                try
                {
                    _context.Update(inventario);
                    await _context.SaveChangesAsync();
                }
                catch (DbUpdateConcurrencyException)
                {
                    if (!InventarioExists(inventario.IdItem))
                    {
                        return NotFound();
                    }
                    else
                    {
                        throw;
                    }
                }
                return RedirectToAction(nameof(Index));
            }
            //esto cambio
            ViewData["IdItemCategoria"] = new SelectList(_context.ItemCategoria, "IdItemCategoria", "Categoria",
inventario.IdItemCategoria);
            return View(inventario);
        }
        // GET: Inventario/Delete/5
        public async Task<IActionResult> Delete(int? id)
        {
            if (id == null)
            {
                return NotFound();
            }

            var inventario = await _context.Inventarios
                .Include(i => i.IdItemCategoriaNavigation)
                .FirstOrDefaultAsync(m => m.IdItem == id);
            if (inventario == null)
            {
                return NotFound();
            }

            return View(inventario);
        }
        // POST: Inventario/Delete/5
        [HttpPost, ActionName("Delete")]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> DeleteConfirmed(int id)
        {
            var inventario = await _context.Inventarios.FindAsync(id);
            if (inventario != null)
            {
                _context.Inventarios.Remove(inventario);
            }

            await _context.SaveChangesAsync();
            return RedirectToAction(nameof(Index));
        }
        // GET: Inventario/Buscar
        [HttpGet]
```

```
        public async Task<IActionResult> Buscar(string searchTerm)
        {
            if (string.IsNullOrWhiteSpace(searchTerm))
            {
                var todos = await _context.Inventarios
                    .Include(i => i.IdItemCategoriaNavigation)
                    .ToListAsync();
                return Json(todos);
            }

            var inventarios = await _context.Inventarios
                .Include(i => i.IdItemCategoriaNavigation)
                .Where(i => i.ItemNombre.Contains(searchTerm) ||
                            i.UnidadMedida.Contains(searchTerm) ||
                            (i.IdItemCategoriaNavigation != null &&
                             i.IdItemCategoriaNavigation.Categoria.Contains(searchTerm)))
                .ToListAsync();

            if (Request.Headers["X-Requested-With"] == "XMLHttpRequest")
            {
                return Json(inventarios);
            }

            return View("Index", inventarios);
        }

        private bool InventarioExists(int id)
        {
            return _context.Inventarios.Any(e => e.IdItem == id);
        }
    }
```

- *Controllers/MesaController.cs*

```
    public class MesaController : Controller
    {
        private readonly RestauranteProgramacionIiContext _context;

        public MesaController(RestauranteProgramacionIiContext context)
        {
            _context = context;
        }
        // GET: Mesa
        public async Task<IActionResult> Index()
        {
            return View(await _context.Mesas.ToListAsync());
        }
        // GET: Mesa/Details/5
        public async Task<IActionResult> Details(int? id)
        {
            if (id == null)
            {
                return NotFound();
            }

            var mesa = await _context.Mesas
                .FirstOrDefaultAsync(m => m.IdMesa == id);
            if (mesa == null)
            {
                return NotFound();
            }
            return View(mesa);
        }
        // GET: Mesa/Create
```

```
        public IActionResult Create()
        {
            return View();
        }
        // POST: Mesa/Create
        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> Create([Bind("IdMesa,Capacidad,Ubicacion,Estado")] Mesa mesa)
        {
            if (ModelState.IsValid)
            {
                _context.Add(mesa);
                await _context.SaveChangesAsync();
                return RedirectToAction(nameof(Index));
            }
            return View(mesa);
        }
        // GET: Mesa/Edit/5
        public async Task<IActionResult> Edit(int? id)
        {
            if (id == null)
            {
                return NotFound();
            }
            var mesa = await _context.Mesas.FindAsync(id);
            if (mesa == null)
            {
                return NotFound();
            }
            return View(mesa);
        }
        // POST: Mesa/Edit/5
        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> Edit(int id, [Bind("IdMesa,Capacidad,Ubicacion,Estado")] Mesa mesa)
        {
            if (id != mesa.IdMesa)
            {
                return NotFound();
            }
            if (ModelState.IsValid)
            {
                try
                {
                    _context.Update(mesa);
                    await _context.SaveChangesAsync();
                }
                catch (DbUpdateConcurrencyException)
                {
                    if (!MesaExists(mesa.IdMesa))
                    {
                        return NotFound();
                    }
                    else
                    {
                        throw;
                    }
                }
                return RedirectToAction(nameof(Index));
            }
            return View(mesa);
        }
        // GET: Mesa/Delete/5
```

```
        public async Task<IActionResult> Delete(int? id)
        {
            if (id == null)
            {
                return NotFound();
            }

            var mesa = await _context.Mesas
                .FirstOrDefaultAsync(m => m.IdMesa == id);
            if (mesa == null)
            {
                return NotFound();
            }

            return View(mesa);
        }
        // POST: Mesa/Delete/5
        [HttpPost, ActionName("Delete")]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> DeleteConfirmed(int id)
        {
            var mesa = await _context.Mesas.FindAsync(id);
            if (mesa != null)
            {
                _context.Mesas.Remove(mesa);
            }

            await _context.SaveChangesAsync();
            return RedirectToAction(nameof(Index));
        }
        private bool MesaExists(int id)
        {
            return _context.Mesas.Any(e => e.IdMesa == id);
        }
    }
```

- *Controllers/ProductoController.cs*

```
    public class ProductoController : Controller
    {
        private readonly RestauranteProgramacionIiContext _context;

        public ProductoController(RestauranteProgramacionIiContext context)
        {
            _context = context;
        }
        // GET: Producto
        public async Task<IActionResult> Index()
        {
            var restauranteProgramacionIiContext = _context.Productos.Include(p => p.IdCategoriaNavigation);
            return View(await restauranteProgramacionIiContext.ToListAsync());
        }
        // GET: Producto/Details/5
        public async Task<IActionResult> Details(int? id)
        {
            if (id == null)
            {
                return NotFound();
            }
            var producto = await _context.Productos
                .Include(p => p.IdCategoriaNavigation)
                .FirstOrDefaultAsync(m => m.IdProducto == id);
            if (producto == null)
            {
```

```
            return NotFound();
        }
        if (Request.Headers["X-Requested-With"] == "XMLHttpRequest")
        {
            return PartialView("_Details", producto);
        }
        return View(producto);
    }
    // GET: Producto/Create
    public IActionResult Create()
    {
        ViewData["IdCategoria"] = new SelectList(
            _context.Categoria.Select(c => new
            {
                c.IdCategoria,
                c.Nombre // Usar el nombre en lugar del ID
            }),
            "IdCategoria",
            "Nombre", // Cambiar a Nombre
            null
        );
        return View();
    }
    // POST: Producto/Create

    [HttpPost]
    [ValidateAntiForgeryToken]
    public async Task<IActionResult>
Create([Bind("IdProducto,Nombre,Precio,Descripcion,Foto,IdCategoria,EsPreparado")] Producto producto)
    {
        if (ModelState.IsValid)
        {
            _context.Add(producto);
            await _context.SaveChangesAsync();
            return RedirectToAction(nameof(Index));
        }
        ViewData["IdCategoria"] = new SelectList(
            _context.Categoria.Select(c => new
            {
                c.IdCategoria,
                c.Nombre // Usar el nombre en lugar del ID
            }),
            "IdCategoria",
            "Nombre", // Cambiar a Nombre
            producto.IdCategoria
        );

        return View(producto);
    }
    // GET: Producto/Edit/5
    public async Task<IActionResult> Edit(int? id)
    {
        if (id == null)
        {
            return NotFound();
        }
        var producto = await _context.Productos.FindAsync(id);
        if (producto == null)
        {
            return NotFound();
        }
        ViewData["IdCategoria"] = new SelectList(
            _context.Categoria.Select(c => new
```

```
                {
                    c.IdCategoria,
                    c.Nombre // Usar el nombre en lugar del ID
                }),
                "IdCategoria",
                "Nombre", // Cambiar a Nombre
                producto.IdCategoria
            );
        return View(producto);
    }
    // POST: Producto/Edit/5
    [HttpPost]
    [ValidateAntiForgeryToken]
    public async Task<IActionResult> Edit(int id,
[Bind("IdProducto,Nombre,Precio,Descripcion,Foto,IdCategoria,EsPreparado")] Producto producto)
    {
        if (id != producto.IdProducto)
        {
            return NotFound();
        }
        if (ModelState.IsValid)
        {
            try
            {
                _context.Update(producto);
                await _context.SaveChangesAsync();
            }
            catch (DbUpdateConcurrencyException)
            {
                if (!ProductoExists(producto.IdProducto))
                {
                    return NotFound();
                }
                else
                {
                    throw;
                }
            }
            return RedirectToAction(nameof(Index));
        }
        ViewData["IdCategoria"] = new SelectList(
                _context.Categoria.Select(c => new
                {
                    c.IdCategoria,
                    c.Nombre
                }),
                "IdCategoria",
                "Nombre",
                producto.IdCategoria
            );
        return View(producto);
    }
    // GET: Producto/Delete/5
    public async Task<IActionResult> Delete(int? id)
    {
        if (id == null)
        {
            return NotFound();
        }
        var producto = await _context.Productos
            .Include(p => p.IdCategoriaNavigation)
            .FirstOrDefaultAsync(m => m.IdProducto == id);
        if (producto == null)
```

```
            {
                return NotFound();      }

            return View(producto);
        }
        // POST: Producto/Delete/5
        [HttpPost, ActionName("Delete")]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> DeleteConfirmed(int id)
        {
            var producto = await _context.Productos.FindAsync(id);
            if (producto != null)
            {
                _context.Productos.Remove(producto);      }
            await _context.SaveChangesAsync();
            return RedirectToAction(nameof(Index));   }
        private bool ProductoExists(int id)
        {
            return _context.Productos.Any(e => e.IdProducto == id);   }
    }
```

- *Controllers/ReservaController.cs*

```
    public class ReservaController : Controller
    {
        private readonly RestauranteProgramacionIiContext _context;

        public ReservaController(RestauranteProgramacionIiContext context)
        {
            _context = context;
        }
        // GET: Reserva
        public async Task<IActionResult> Index()
        {
            var restauranteProgramacionIiContext = _context.Reservas.Include(r => r.IdClienteNavigation).Include(r =>
r.IdMesaNavigation);
            return View(await restauranteProgramacionIiContext.ToListAsync());
        }
        // GET: Reserva/Details/5
        public async Task<IActionResult> Details(int? id)
        {
            if (id == null)
            {
                return NotFound();
            }
            var reserva = await _context.Reservas
                .Include(r => r.IdClienteNavigation)
                .Include(r => r.IdMesaNavigation)
                .FirstOrDefaultAsync(m => m.IdReserva == id);
            if (reserva == null)
            {
                return NotFound();
            }
            if (Request.Headers["X-Requested-With"] == "XMLHttpRequest")
            {
                return PartialView("_Details", reserva);
            }
            return View(reserva);
        }
        // GET: Reserva/Create
        public IActionResult Create()
        {
            ViewData["IdCliente"] = new SelectList(_context.Clientes, "IdCliente", "IdCliente");
            ViewData["IdMesa"] = new SelectList(_context.Mesas, "IdMesa", "IdMesa");
            return View();
```

```
        }
        // POST: Reserva/Create
        // To protect from overposting attacks, enable the specific properties you want to bind to.
        // For more details, see http://go.microsoft.com/fwlink/?LinkId=317598.
        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult>
    Create([Bind("IdReserva,Fecha,Hora,IdMesa,IdCliente,NumeroPersonas,Estado,Comentarios")] Reserva reserva)
        {
            if (ModelState.IsValid)
            {
                _context.Add(reserva);
                await _context.SaveChangesAsync();
                return RedirectToAction(nameof(Index));
            }
            ViewData["IdCliente"] = new SelectList(_context.Clientes, "IdCliente", "IdCliente", reserva.IdCliente);
            ViewData["IdMesa"] = new SelectList(_context.Mesas, "IdMesa", "IdMesa", reserva.IdMesa);
            return View(reserva);
        }
        // GET: Reserva/Edit/5
        public async Task<IActionResult> Edit(int? id)
        {
            if (id == null)
            {
                return NotFound();
            }
            var reserva = await _context.Reservas.FindAsync(id);
            if (reserva == null)
            {
                return NotFound();
            }
            ViewData["IdCliente"] = new SelectList(_context.Clientes, "IdCliente", "IdCliente", reserva.IdCliente);
            ViewData["IdMesa"] = new SelectList(_context.Mesas, "IdMesa", "IdMesa", reserva.IdMesa);
            return View(reserva);
        }
        // POST: Reserva/Edit/5
        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> Edit(int id,
    [Bind("IdReserva,Fecha,Hora,IdMesa,IdCliente,NumeroPersonas,Estado,Comentarios")] Reserva reserva)
        {
            if (id != reserva.IdReserva)
            {
                return NotFound();
            }
            if (ModelState.IsValid)
            {
                try
                {
                    _context.Update(reserva);
                    await _context.SaveChangesAsync();
                }
                catch (DbUpdateConcurrencyException)
                {
                    if (!ReservaExists(reserva.IdReserva))
                    {
                        return NotFound();
                    }
                    else
                    {
                        throw;
                    }
                }
```

```
            return RedirectToAction(nameof(Index));
        }
        ViewData["IdCliente"] = new SelectList(_context.Clientes, "IdCliente", "IdCliente", reserva.IdCliente);
        ViewData["IdMesa"] = new SelectList(_context.Mesas, "IdMesa", "IdMesa", reserva.IdMesa);
        return View(reserva);
    }
    // GET: Reserva/Delete/5
    public async Task<IActionResult> Delete(int? id)
    {
        if (id == null)
        {
            return NotFound();
        }
        var reserva = await _context.Reservas
            .Include(r => r.IdClienteNavigation)
            .Include(r => r.IdMesaNavigation)
            .FirstOrDefaultAsync(m => m.IdReserva == id);
        if (reserva == null)
        {
            return NotFound();
        }
        return View(reserva);
    }
    // POST: Reserva/Delete/5
    [HttpPost, ActionName("Delete")]
    [ValidateAntiForgeryToken]
    public async Task<IActionResult> DeleteConfirmed(int id)
    {
        var reserva = await _context.Reservas.FindAsync(id);
        if (reserva != null)
        {
            _context.Reservas.Remove(reserva);
        }
        await _context.SaveChangesAsync();
        return RedirectToAction(nameof(Index));
    }
    // Agregar el método de búsqueda
    [HttpGet]
    public async Task<IActionResult> Buscar(string searchTerm)
    {
        if (string.IsNullOrWhiteSpace(searchTerm))
        {
            var todos = await _context.Reservas
                .Include(r => r.IdClienteNavigation)
                .Include(r => r.IdMesaNavigation)
                .ToListAsync();
            return Json(todos);
        }
        var reservas = await _context.Reservas
            .Include(r => r.IdClienteNavigation)
            .Include(r => r.IdMesaNavigation)
            .Where(r => r.IdClienteNavigation.Nombres.Contains(searchTerm) ||
                        r.Estado.Contains(searchTerm) ||
                        r.Comentarios.Contains(searchTerm))
            .ToListAsync();
        if (Request.Headers["X-Requested-With"] == "XMLHttpRequest")
        {
            return Json(reservas);
        }

        return View("Index", reservas);
    }
    private bool ReservaExists(int id)
```

```
        {
            return _context.Reservas.Any(e => e.IdReserva == id);
        }
    }
```

- **Vistas (Views):** Las vistas de la aplicación se implementaron con Razor, con una estructura específica para implementar las funcionalidades CRUD. Para mas detalle revisar el repositorio del proyecto.

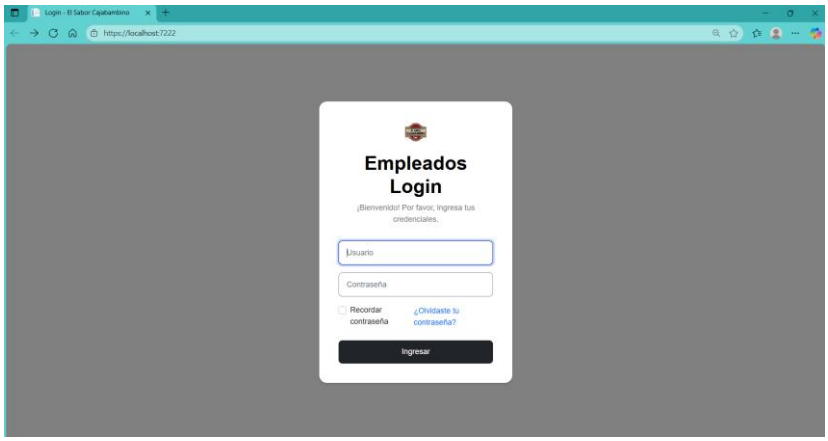
CAPÍTULO 6. PRUEBA DEL SISTEMA DE GESTIÓN WEB

6.1. Objetivo de las Pruebas

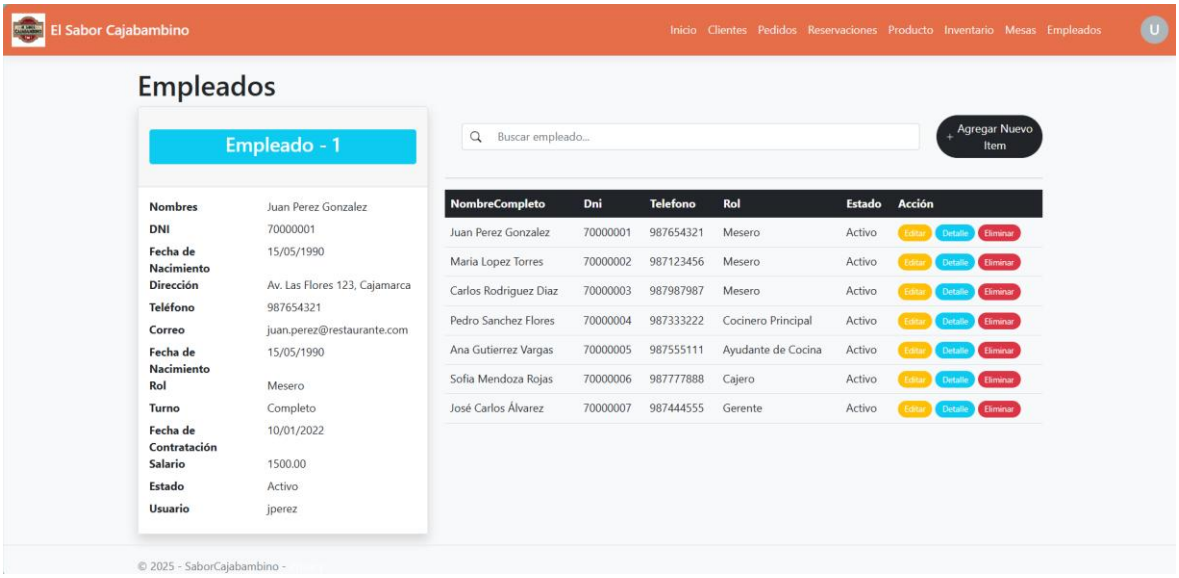
- **Objetivo:** El objetivo principal de esta etapa fue validar que las funcionalidades desarrolladas en el sistema de gestión web operen de acuerdo con los requerimientos establecidos. Se buscó asegurar que todas las operaciones CRUD (Crear, Leer, Actualizar, Borrar) funcionen correctamente en cada uno de los módulos.
- **Enfoque de las Pruebas:** Se utilizó un enfoque de **pruebas funcionales (caja negra)**, lo que significa que se validó el comportamiento de la aplicación desde la perspectiva del usuario, sin inspeccionar el código interno.

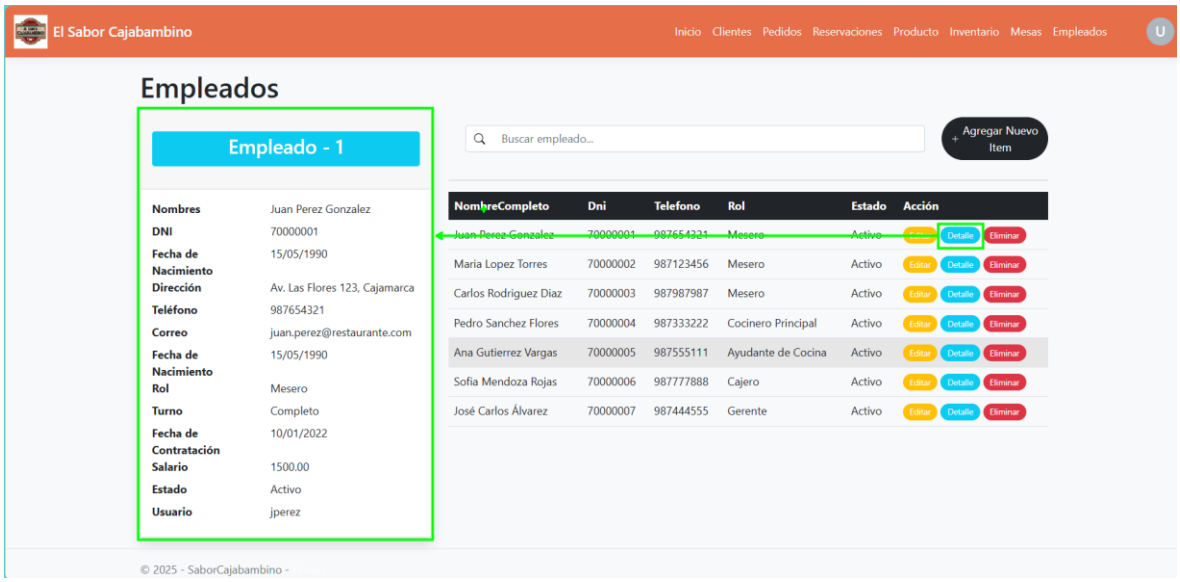
6.2. Pruebas

6.2.1. Login

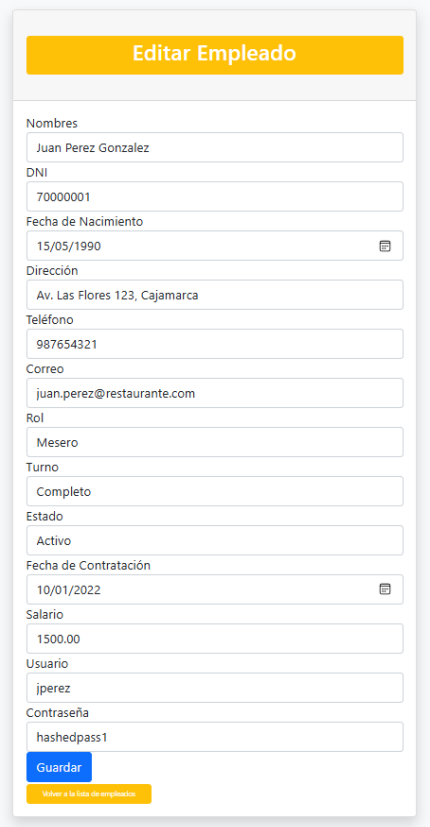


6.2.2. Empleados

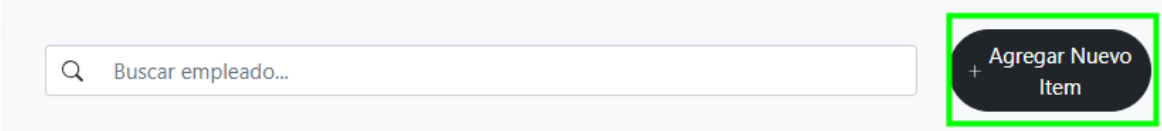




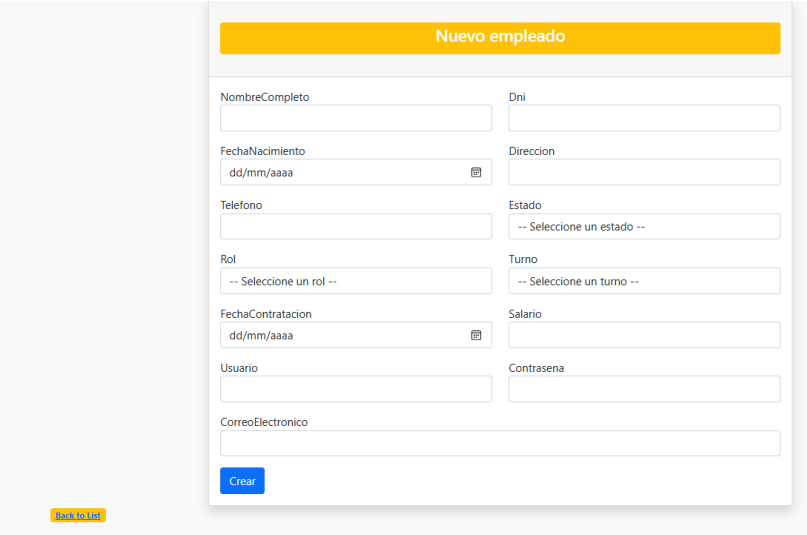
NombreCompleto	Dni	Telefono	Rol	Estado	Acción
Juan Perez Gonzalez	70000001	987654321	Mesero	Activo	Editar Detalle Eliminar
Maria Lopez Torres	70000002	987123456	Mesero	Activo	Editar Detalle Eliminar
Carlos Rodriguez Diaz	70000003	987987987	Mesero	Activo	Editar Detalle Eliminar
Pedro Sanchez Flores	70000004	987333222	Cocinero Principal	Activo	Editar Detalle Eliminar
Ana Gutierrez Vargas	70000005	987555111	Ayudante de Cocina	Activo	Editar Detalle Eliminar
Sofia Mendoza Rojas	70000006	987777888	Cajero	Activo	Editar Detalle Eliminar
José Carlos Álvarez	70000007	987444555	Gerente	Activo	Editar Detalle Eliminar



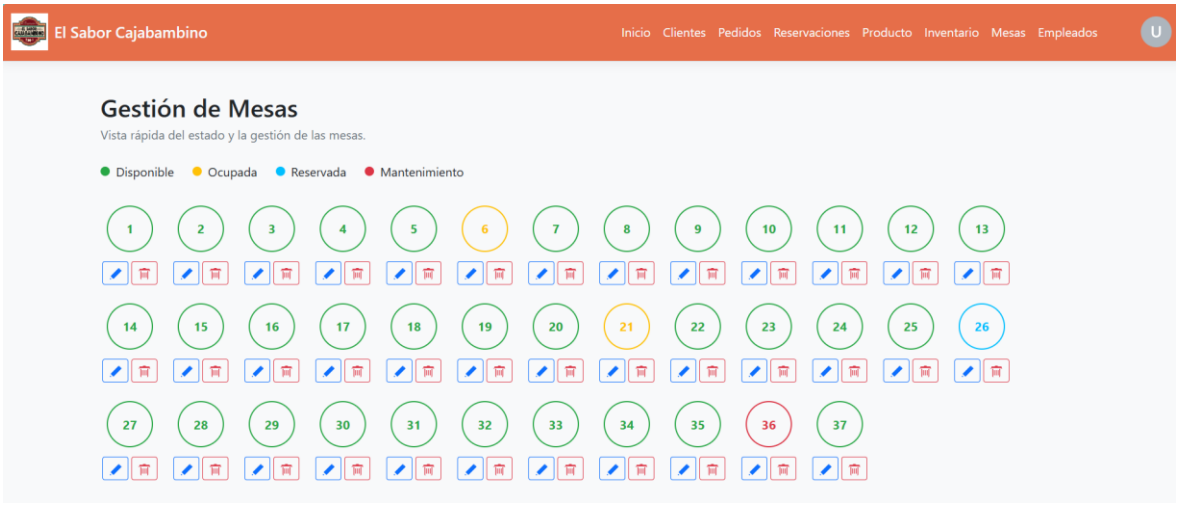
NombreCompleto	Dni	Telefono	Rol	Estado	Acción
Juan Perez Gonzalez	70000001	987654321	Mesero	Activo	Editar Detalle Eliminar
Maria Lopez Torres	70000002	987123456	Mesero	Activo	Editar Detalle Eliminar
Carlos Rodriguez Diaz	70000003	987987987	Mesero	Activo	Editar Detalle Eliminar
Pedro Sanchez Flores	70000004	987333222	Cocinero Principal	Activo	Editar Detalle Eliminar
Ana Gutierrez Vargas	70000005	987555111	Ayudante de Cocina	Activo	Editar Detalle Eliminar
Sofia Mendoza Rojas	70000006	987777888	Cajero	Activo	Editar Detalle Eliminar
José Carlos Álvarez	70000007	987444555	Gerente	Activo	Editar Detalle Eliminar

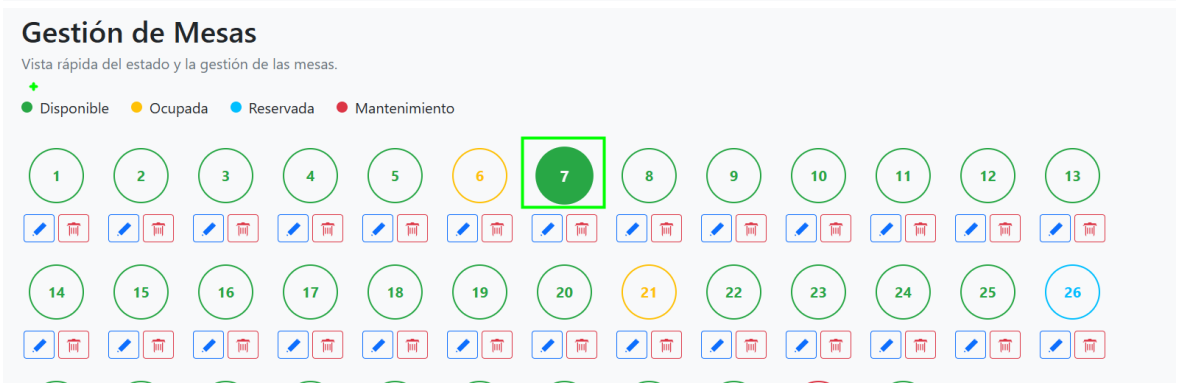


NombreCompleto	Dni	Telefono	Rol	Estado	Acción
Juan Perez Gonzalez	70000001	987654321	Mesero	Activo	Editar Detalle Eliminar
Maria Lopez Torres	70000002	987123456	Mesero	Activo	Editar Detalle Eliminar
Carlos Rodriguez Diaz	70000003	987987987	Mesero	Activo	Editar Detalle Eliminar
Pedro Sanchez Flores	70000004	987333222	Cocinero Principal	Activo	Editar Detalle Eliminar
Ana Gutierrez Vargas	70000005	987555111	Ayudante de Cocina	Activo	Editar Detalle Eliminar
Sofia Mendoza Rojas	70000006	987777888	Cajero	Activo	Editar Detalle Eliminar
José Carlos Álvarez	70000007	987444555	Gerente	Activo	Editar Detalle Eliminar

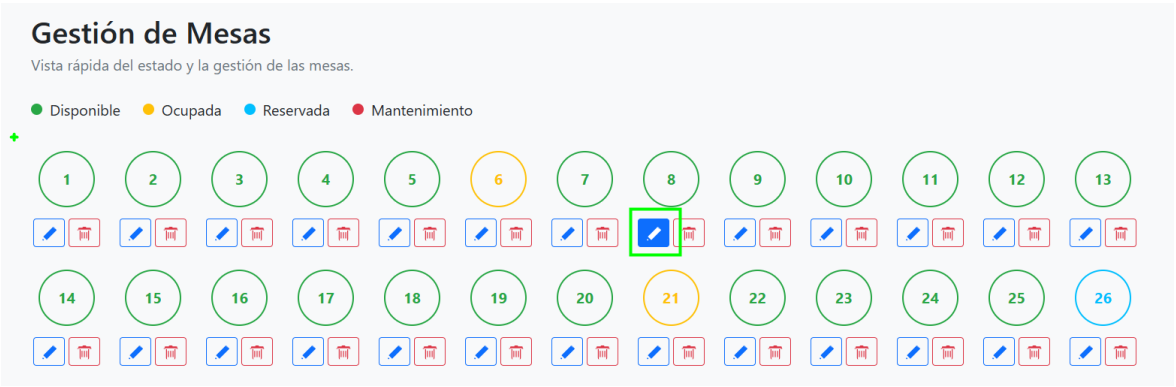

Nuevo empleado
NombreCompleto
Dni
FechaNacimiento
dd/mm/aaaa
Direccion
Telefono
Estado
-- Seleccione un estado --
Rol
-- Seleccione un rol --
Turno
-- Seleccione un turno --
FechaContratacion
dd/mm/aaaa
Salario
Usuario
Contrasena
CorreoElectronico
Crear
Back to List

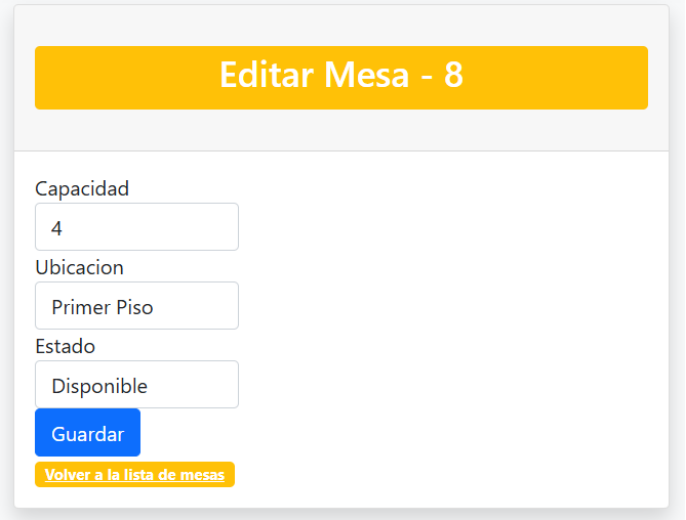
6.2.1. Mesas

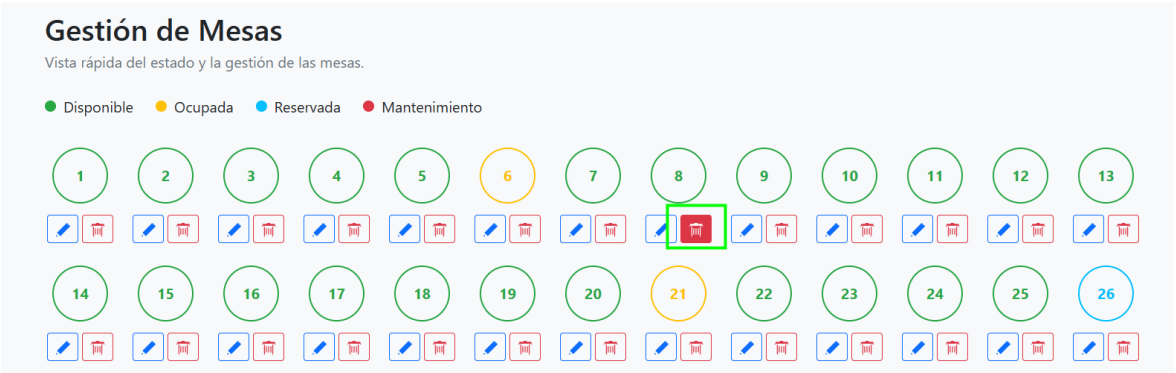

El Sabor Cajabambino
Inicio Clientes Pedidos Reservaciones Producto Inventario Mesas Empleados
U
Gestión de Mesas
Vista rápida del estado y la gestión de las mesas.
Disponible Ocupada Reservada Mantenimiento
1 2 3 4 5 6 7 8 9 10 11 12 13
14 15 16 17 18 19 20 21 22 23 24 25 26
27 28 29 30 31 32 33 34 35 36 37


Gestión de Mesas
Vista rápida del estado y la gestión de las mesas.
Disponible Ocupada Reservada Mantenimiento
1 2 3 4 5 6 7 8 9 10 11 12 13
14 15 16 17 18 19 20 21 22 23 24 25 26

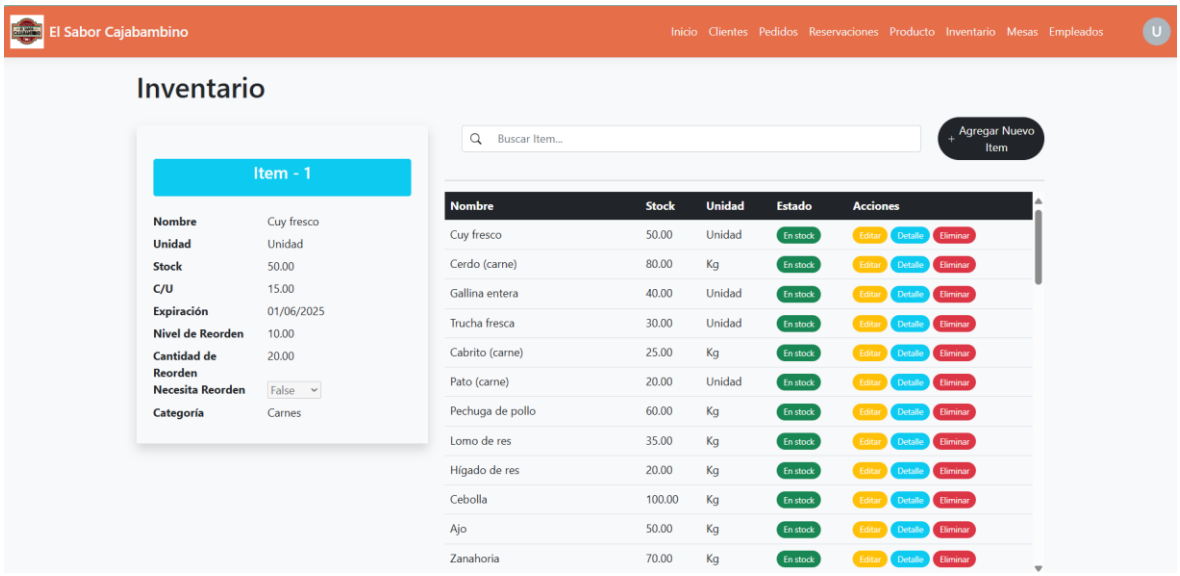

Mesa - 7
Capacidad
4
Ubicacion
Primer Piso
Estado
Disponible
Editar
Volver a la lista de mesas


Gestión de Mesas
Vista rápida del estado y la gestión de las mesas.
Disponible
Ocupada
Reservada
Mantenimiento
1 2 3 4 5 6 7 8 9 10 11 12 13
14 15 16 17 18 19 20 21 22 23 24 25 26

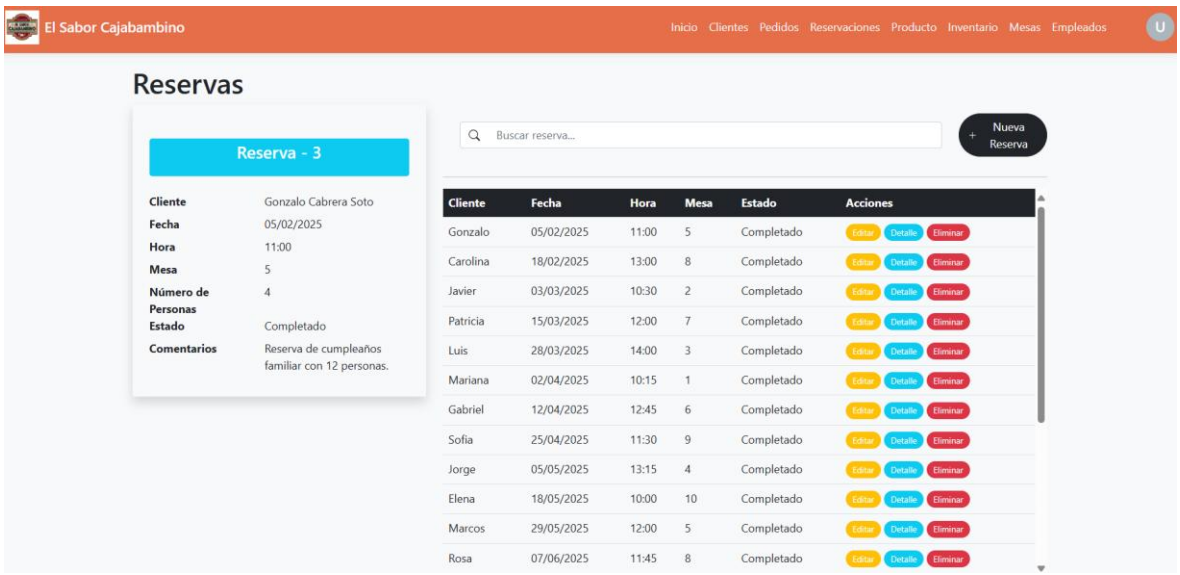

Editar Mesa - 8
Capacidad
4
Ubicacion
Primer Piso
Estado
Disponible
Guardar
Volver a la lista de mesas


Gestión de Mesas
Vista rápida del estado y la gestión de las mesas.
Disponible
Ocupada
Reservada
Mantenimiento
1 2 3 4 5 6 7 8 9 10 11 12 13
14 15 16 17 18 19 20 21 22 23 24 25 26

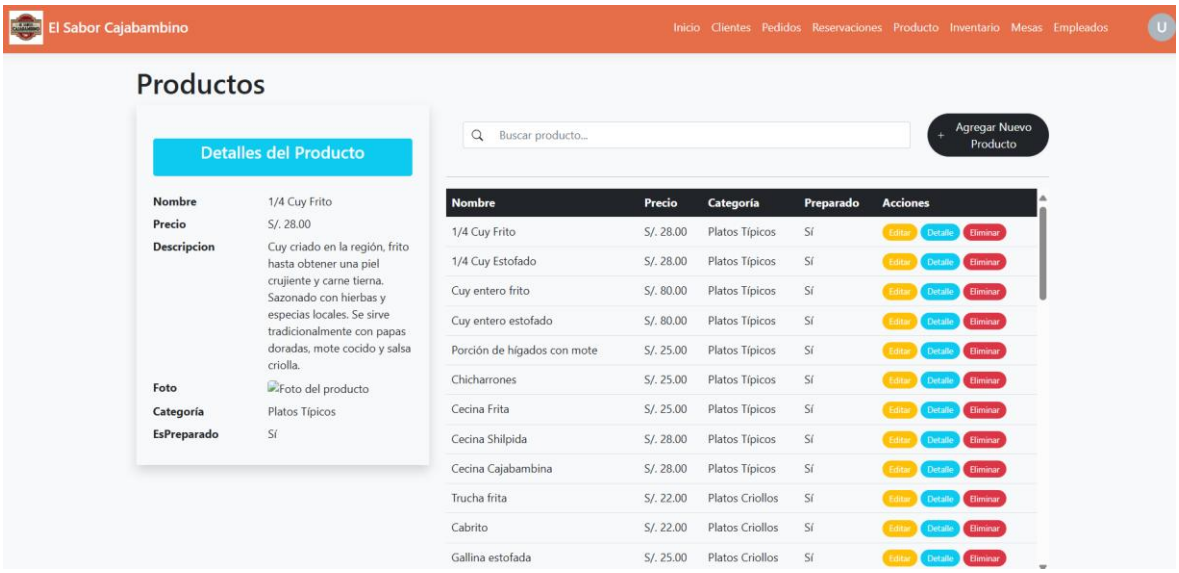
6.2.1. Inventario



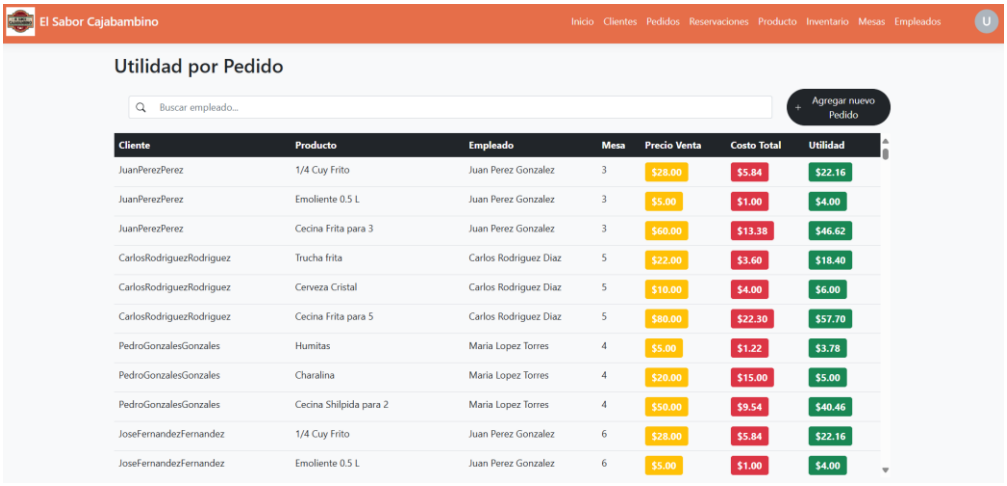
6.2.1. Reservaciones



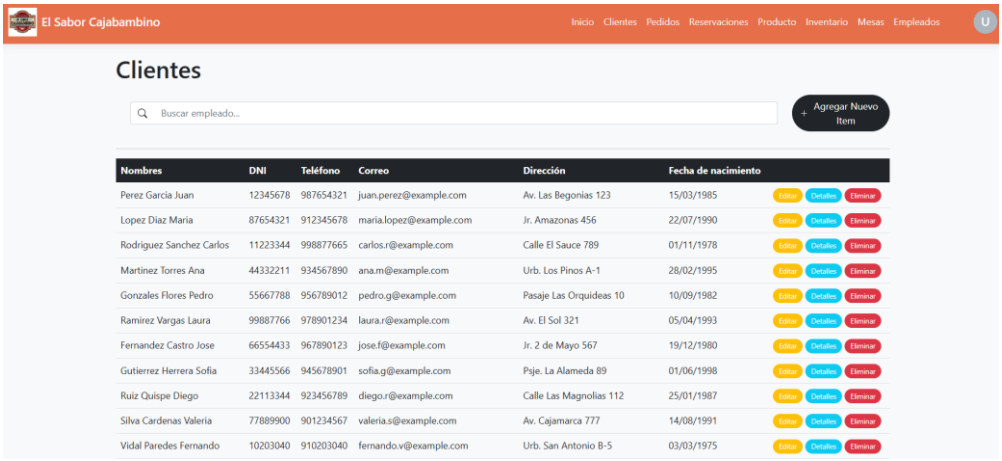
6.2.1. Productos



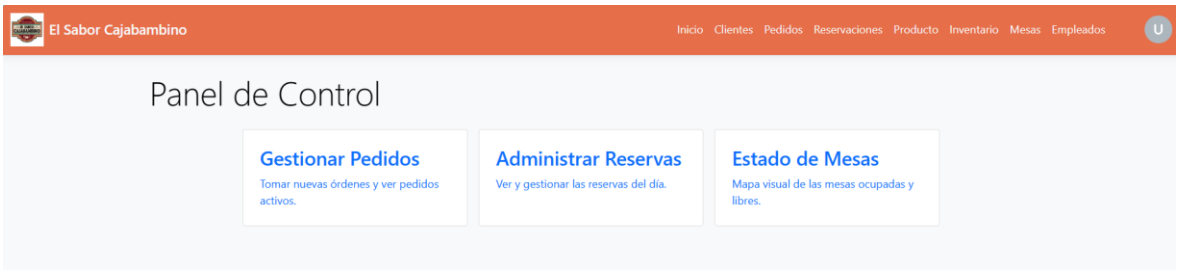
6.2.1. Pedidos



6.2.1. Cliente



6.2.1. Panel de inicio



CONCLUSIONES

El sistema de gestión web diseñado y desarrollado para "El Sabor Cajabambino" logra su objetivo principal al optimizar los procesos críticos del área de ventas. Al digitalizar y centralizar la gestión de pedidos, inventario, mesas y clientes, la solución elimina la dependencia de procesos manuales propensos a errores, lo que se traduce en una mayor eficiencia operativa y un servicio al cliente más ágil y confiable. Este enfoque estructurado resuelve directamente los problemas de inconsistencia de datos y lentitud en el servicio identificados en la etapa inicial del proyecto.

La adopción de la arquitectura Modelo-Vista-Controlador (MVC) ha demostrado ser una elección acertada, ya que proporciona una base sólida, escalable y mantenible para el sistema. La separación de la lógica del negocio (Modelo), la interfaz de usuario (Vista) y el manejo de solicitudes (Controlador) facilita futuras mejoras y la adición de nuevas funcionalidades sin comprometer la integridad del sistema. Esta robusta arquitectura asegura que la aplicación pueda crecer y adaptarse a las necesidades cambiantes del restaurante.

Finalmente, la implementación exitosa de las operaciones CRUD (Crear, Leer, Actualizar, Eliminar) en cada módulo principal satisface plenamente los requerimientos identificados en la fase de análisis. Estas funcionalidades básicas, pero esenciales, permiten al personal del restaurante gestionar de forma integral su información crítica, sentando las bases para una gestión de datos más eficiente. El proyecto no solo ha entregado una herramienta funcional, sino que también ha establecido un marco tecnológico que permitirá al negocio seguir modernizando sus operaciones en el futuro.

REFERENCIAS

AlexHost. (2024, 6 de diciembre). *¿Qué es MVC? Ventajas y desventajas de MVC*. https://alexhost.com/es/faq/que-es-mvc-ventajas-y-desventajas-de-mvc/

Conecta Software. (2024, 14 de septiembre). *¿Qué es CRUD y su papel en la integración de datos y APIs?*. https://www.conectasoftware.com/magazine/glosario/crud/

Ebis Education. (2023, 22 de diciembre). *Definición y ventajas de las operaciones CRUD*. https://www.ebiseducation.com/crud-que-es-para-que-sirve-como-funciona-y-ejemplos

Ibexa. (2025). *¿Qué es la gestión de sitios web?*. https://www.ibexa.co/es/recursos/insights-y-articulos/what-is-website-management

Microsoft Learn. (2024, 6 de noviembre). *Información general de ASP.NET Core MVC*. https://learn.microsoft.com/es-es/aspnet/core/mvc/overview?view=aspnetcore-9.0

Microsoft. (2025). *Novedades de SQL Server 2022*. https://learn.microsoft.com/en-us/sql/sql-server/what-s-new-in-sql-server-2022?view=sql-server-ver17

Nedigital. (2024). *Características y relevancia de Microsoft SQL Server para la gestión de datos*. https://www.nedigital.com/es/blog/microsoft-sql-server-ventajas-y-desventajas

Silva, I. (2023). *Sistemas web 101: comprensión de los fundamentos y beneficios*. https://scriptcaseblog.net/es/scriptcase-es/sistemas-web-101-comprension-de-los-fundamentos-y-beneficios/#:~:text=Es%20un%20software%20en%20l%C3%ADnea,en%20l%C3%ADnea%20eficiente%20y%20segura.

Umbraco. (2023). *What is ASP.NET Core?*. https://umbraco.com/knowledge-base/asp-dot-net-core/

WE Educación. (s.f.). *4 razones por las que SQL Server es usado en las empresas*. https://we-educacion.com/sql-server