UNIVERSIDAD TECNOLÓGICA DE PANAMÁ

FACULTAD DE INGENIERÍA DE SISTEMAS COMPUTACIONALES

LICENCIATURA EN DESARROLLO Y GESTIÓN DE SOFTWARE

DESARROLLO DE SOFTWARE IV

Profa. ***Mgtr. Ana Araba de Ruiz***

PROYECTO #1

Sistema de gestión integral para panadería familiar

Integrantes del Grupo 5:

Bernal,Austin. – 8-1017-760

González, Melany. – 8-1006-1163

Saavedra,Alberto. – 8-1013-2305

Zhang,Andres. – 8-1013-1433

Grupo: ___1GS125_____

27 DE NOVIEMBRE 2025

Contenido

Tabla de Ilustraciones

Tablas

Resumen

El presente proyecto final aborda la necesidad de digitalizar y optimizar los procesos operativos de *Family Bakery* (o Panadería Artesanal). Actualmente, la gestión manual de pedidos, inventario y producción resulta en errores frecuentes de cálculo, descontrol de existencias e incapacidad para generar análisis de ventas fiables. La solución propuesta es un Sistema de Gestión desarrollado en C# bajo una arquitectura de N Capas, utilizando SQL Server como motor de base de datos. La característica central es la automatización del control de inventario, implementando un módulo de **Registro de Producción transaccional** que descuenta automáticamente las materias primas (según las Recetas) e incrementa el stock de productos terminados. Esto garantiza la integridad de los datos, mejora la eficiencia operativa y proporciona la base para la toma de decisiones informadas.

Introducción

En el dinámico sector de la repostería artesanal, la eficiencia y la precisión en la gestión de recursos son cruciales para la rentabilidad y la satisfacción del cliente. *Family Bakery* es un negocio con gran potencial, pero sus métodos de operación manuales representan un cuello de botella que limita su crecimiento. Este proyecto se enfoca en superar dichas limitaciones mediante la implementación de un sistema de información robusto. El objetivo es transicionar de un modelo basado en papel y hojas de cálculo a un sistema digitalizado, modular y escalable. El sistema no solo gestionará las ventas y el catálogo, sino que atacará el problema raíz: la relación precisa entre **Recetas, Producción e Inventario**. A continuación, se presenta la documentación de la fase de análisis, que establece los objetivos, el alcance y los cimientos técnicos de la aplicación.

Objetivos

General

Desarrollar e implementar un Sistema de Gestión de Pedidos, Inventario y Producción (SGPA) integral y escalable para *Family Bakery* (Panadería Artesanal) utilizando C# y SQL Server, con el fin de automatizar procesos críticos y mejorar la eficiencia operativa.

Específicos

- Diseñar el sistema bajo una arquitectura de **N Capas (N-Tier)** para asegurar la separación de responsabilidades, facilitar el mantenimiento y garantizar la escalabilidad futura.
- Modelar e implementar la base de datos en **SQL Server**, incluyendo las estructuras clave para la gestión de Recetas (Recetas, DetalleRecetas) y el control de stock (Productos.Stock, RegistroProduccion).
- Implementar una **lógica de negocio transaccional** (mediante *Stored Procedures*) para el Módulo de Producción, garantizando la atomicidad en el descuento de ingredientes y el aumento de productos terminados.
- Desarrollar los módulos de **Administración**, **Inventario**, **Catálogo, productos** y **Pedidos** con interfaces de usuario (Windows Forms o Web) intuitivas y funcionales para los diferentes roles de usuario.

Planteamiento del problema

Family Bakery gestiona sus pedidos y productos de forma manual, lo que genera errores en el cálculo de totales, confusión en el inventario y dificultad para analizar ventas. Esto afecta la eficiencia del negocio y la experiencia del cliente. Se requiere un sistema digital que automatice estos procesos y permita tomar decisiones informadas.

Alcance

El proyecto contempla el desarrollo de un Sistema de Gestión (SGPA) completo que abarca la gestión de datos maestros (Usuarios, Categorías, Productos, Ingredientes), el control de la cadena de valor interna (Recetas, Producción) y la interfaz comercial (Catálogo y Pedidos). Se implementará la arquitectura N-Tier completa (Presentación, Negocio, Datos, Entidades) con C# para la lógica de programación y SQL Server para la persistencia de datos. El alcance incluye los mockups diseñados y la lógica de negocio crítica y transaccional del Módulo de Producción.

Justificación

La justificación de este proyecto radica en la necesidad imperante de formalizar y automatizar la gestión de recursos en un negocio artesanal. La implementación de un SGIPAN eliminará los errores humanos asociados al cálculo de pedidos y al manejo manual del inventario. La lógica transaccional de producción garantizará una precisión del 100% en el stock, permitiendo a la gerencia conocer en tiempo real la disponibilidad de productos e ingredientes. En última instancia, la digitalización con este sistema transformará la eficiencia, reducirá los costos operativos por mermas y errores, y mejorará la experiencia del cliente al ofrecer un proceso de pedido más ágil y transparente.

1 Información sobre la aplicación

1.1 Explicación general sobre la aplicación

La aplicación es un Sistema de Gestión (SGPA) diseñado para una panadería artesanal. Se basa en una **Arquitectura de N Capas**, donde la lógica del negocio está estrictamente separada de la interfaz y la capa de acceso a datos.

Módulos Clave:

1. **Administración:** Gestión de usuarios y configuración de datos maestros (categorías, precios).
2. **Inventario y Producción:** Módulo CRÍTICO que maneja el stock de Ingredientes (entradas, vencimientos) y, mediante el uso de **Recetas**, ejecuta el RegistroProduccion transaccional para actualizar el stock de Productos terminados.
3. **Catálogo y Pedidos:** Permite al cliente navegar por los productos, registrar pedidos con detalles de entrega, y seleccionar el método de pago.

1.2 Versión del CSharp utilizado (2017……, 2022)

- Versión 17.14.13 (August 2025)

1.3 Descripción de la Aplicación de las Buenas Prácticas de Programación en su proyecto

La estructura del proyecto se adhiere a las siguientes Buenas Prácticas:

1. **Separación de Responsabilidades (Arquitectura N-Tier):** La estricta división en Capa de Presentación (UI), Capa de Negocio (BLL) y Capa de Datos (DAL) aísla fallos, permite que los desarrolladores trabajen simultáneamente en diferentes capas y facilita la futura migración de la interfaz (por ejemplo, de Windows Forms a Web).
2. **Transaccionalidad en Operaciones Críticas:** Se utiliza una **TRANSACCIÓN** explícita dentro del *Stored Procedure* SP_RegistrarProduccion. Esto aplica el principio de Atomicidad (de ACID), asegurando que el descuento de ingredientes y el aumento de stock de productos **se completen ambos o fallen ambos**, manteniendo la consistencia de los datos del inventario.
3. **Uso de POCOs (Business Entities):** Las clases de la Capa de Entidades (BE) son objetos simples y puros que solo representan la estructura de los datos, mejorando la claridad y el mapeo con la base de datos.
4. **Uso de Stored Procedures (SQL Server):** La lógica de acceso a datos se centraliza en *Stored Procedures*, lo que mejora el rendimiento, la seguridad y la encapsulación de las consultas SQL.

2 Conclusiones y recomendaciones

2.1 Del equipo

El desarrollo del proyecto, que combinó la planificación y el diseño de la interfaz de usuario (UI) con la creación de una base de datos robusta, ha sido un proceso de aprendizaje intenso en equipo. Desde la parte de la UI y la implementación en C#, el mayor reto ha sido ajustar el diseño a las restricciones del framework de programación, lo que nos obligó a hacer ajustes constantes y darnos cuenta de que no conocíamos bien la arquitectura por capas desde el inicio. Hemos superado esto con mucha comunicación, haciendo ajustes conjuntos para que el código no se estanque. Mientras tanto, el equipo de la base de datos se enfocó en crear una estructura detallada y robusta, aprendiendo a usar TRY/CATCH y validaciones para evitar errores; ellos descubrieron que el verdadero desafío no es solo conectar la base de datos, sino lograr que los datos fluyan de forma precisa y exacta a cada botón y campo de la UI. En resumen, aprendimos que la flexibilidad en la planificación, el trabajo conjunto en la definición de flujos, y la precisión

milimétrica en el manejo de datos son lo más importante. Todavía nos queda trabajo pendiente, sobre todo en la integración final de la persistencia, pero la base está muy sólida y vamos superbién avanzando.

2.2 Individual

- ✓ ***Bernal,Austin:*** *El trabajo de implementación, centrado en montar la interfaz de usuario (UI) en C# y definir los flujos con mi compañera, nos ha enseñado mucho. Viéndolo bien, conectar la base de datos no es tan complejo, pero el reto real es que los datos vayan y vengan exactos para cada botón y formulario de la UI. Nuestro mayor problema hasta ahora ha sido ajustar el diseño que nos dio mi compañera a lo que el framework de C# nos permite, lo cual nos obliga a refactorizar el código constantemente. Lo estamos solucionando hablando mucho para buscar alternativas que funcionen en C# sin perder la idea original. Todo esto me demostró que ser flexible al implementar es clave y que la precisión de los datos es fundamental para que la app funcione. Aún nos queda trabajo pendiente, sobre todo en la parte de guardar datos, pero vamos bien. Para seguir mejorando, recomiendo hacer pruebas rápidas de los componentes apenas los monto, revisar los diagramas de flujos de datos antes de codificar la persistencia, y siempre dejar un espacio para limpiar el código después de terminar un flujo.*

- ✓ ***González, Melany:*** *El desarrollo del proyecto, que se centró en la planificación y el diseño de la aplicación, enfrentó desafíos. Lo más difícil al principio fue que no sabíamos bien cómo manejar el diseño por capas, que es clave para que el código no se vuelva un lío después. También, escoger los controles exactos para que mi compañero pudiera implementarlos en C# no fue fácil. Tuvimos un problema gordo al pasar el diseño al código: tocó hacer muchos ajustes juntos para que el diseño se adaptara a lo que la programación permite, pero lo estamos arreglando con reuniones rápidas. Aprendí que es vital que los planes sean flexibles y que trabajar juntos en armar los flujos de la aplicación es lo que nos hace avanzar. Todavía nos queda trabajo, pero vamos bién. Para lo que falta, propongo tres cosas: ponernos más de acuerdo en el diseño y el código, dibujar mejor los flujos antes de empezar a programar, y hacer revisiones rápidas del diseño desde el inicio.*

- ✓ ***Saavedra, Alberto*** *: En este trabajo hicimos una base de datos que fuera lo bastante detallada para evitar cualquier error que se pudiera presentar en cualquier escenario, además de poder aprender sobre lo procedimientos el try y catch para evitar cualquier error y aprender nuevas funciones gracias a las palabras del glosario.*

- ✓ ***Zhang, Andres:*** *En conclusión, Nosotros realizamos la base de dato en una estructura robusta, capaz de prevenir datos inconsistentes y manejar errores de forma controlada mediante el uso de validaciones y bloques TRY/CATCH en los procedimientos almacenados. También pude aprender varias palabras que no lo conocía cuando estaba realizando el glosario.*

3 Anexos

3.1 Cronograma Estimado del Proyecto Final

Cronograma: Proyecto Final y Proyecto # 1 (Estimado)

ACTIVIDADES	Semanas						
	8	9	10	11	12	13	14
FASE 1. ANÁLISIS							
a. Entregable 1. Situación Actual							
1. Planteamiento del problema							
b. Entregable 2. Especificación de Requerimientos							
1. Lista de Requerimientos							
c. Entregable 3. Propuesta de Solución							
1. Diseño preliminar del Prototipo							
2. Presentacion del Prototipo							
FASE 2. DISEÑO							
a. Entregable 4. Arquitectura del Sistema							
1. Diagrama general del sistema							
2. Diseño de los Formularios							
b. Entregable 5. Diseño de Base de Datos							
1. Diagrama de la Base de Datos							
2. Especificacion de la Base de Datos							
3. Presentacion del Diseño							
FASE 3. DESARROLLO							
a. Entregable 6.							
1. Revisión de la Funcionalidad							
2. Revisión de la Interfaz Gráfica							
3. Revisión de la calidad del código							
4. Documentación de las Funcionalidades							
5. Pruebas							
FASE 4. IMPLEMENTACIÓN							
a. Entregable 7. Implementación del Sistema							
1. Pruebas							
2. Guía de usuario							
3. Entrega y Sustentación del Proyecto Final							

3.2 Cronograma Real del Proyecto Final

Cronograma: Proyecto # 1 (Estimado)

ACTIVIDADES	Semanas									Estatus
	8	9	10	11	12	13	14	15	16	
FASE 1. ANÁLISIS										Terminado
a. Entregable 1. Situación Actual										Terminado
1. Planteamiento del problema										Terminado
b. Entregable 2. Especificación de Requerimientos										Terminado
1. Lista de Requerimientos										Terminado
c. Entregable 3. Propuesta de Solución										Terminado
1. Diseño preliminar del Prototipo										Terminado
2. Presentacion del Prototipo										Terminado
FASE 2. DISEÑO										Terminado
a. Entregable 4. Arquitectura del Sistema										Terminado
1. Diagrama general del sistema										Terminado
2. Diseño de los Formularios										Terminado
b. Entregable 5. Diseño de Base de Datos										Terminado
1. Diagrama de la Base de Datos										Terminado
2. Especificacion de la Base de Datos										Terminado
3. Presentacion del Diseño										Terminado
FASE 3. DESARROLLO										Iniciado
a. Entregable 6.										En Desarrollo
1. Revisión de la Funcionalidad										En Desarrollo
2. Revisión de la Interfaz Gráfica										En Desarrollo
3. Revisión de la calidad del código										En Desarrollo
4. Documentación de las Funcionalidades										En Desarrollo
5. Pruebas										En Desarrollo
FASE 4. IMPLEMENTACIÓN										En Desarrollo
a. Entregable 7. Implementación del Sistema										En Desarrollo
1. Pruebas										En Desarrollo
2. Guía de usuario										En Desarrollo
3. Entrega y Sustentación del Proyecto Final										En Desarrollo

3.3 Documentación de los entregables de las fases identificadas en el cronograma real

3.3.1 Fase de Análisis (1)

3.3.1.1 Lista de requerimientos

Tabla 1. Requerimientos de la app

Tipo	Requerimiento (Funcional/No Funcional)	Descripción Detallada	Módulo Impactado
F	Gestión de Datos Maestros	CRUD completo para Usuarios, Categorías, Productos, Ingredientes y Recetas.	Administración
F	Registro de Producción Transaccional	El sistema debe descontar ingredientes automáticamente según la DetalleRecetas y aumentar el Stock de Productos dentro de una única transacción.	Inventario/Producción
F	Gestión de Pedidos	Registrar la cabecera y el detalle de los pedidos, incluyendo DireccionEntrega, MetodoPago y Observaciones.	Pedidos
F	Control de Stock de Producto Terminado	Descuento del campo Stock de la tabla Productos al concretar una venta/pedido.	Pedidos/Inventario
F	Módulo de Recetas	El sistema debe permitir definir qué Ingredientes y qué Cantidad se requiere por unidad de Producto.	Administración
NF	Arquitectura	El sistema debe implementarse obligatoriamente bajo un esquema de **N Capas**.	General
NF	Persistencia de Datos	Utilizar **SQL Server** para la base de datos.	General
NF	Auditoría de Stock	El sistema debe mantener un log de entradas/salidas de stock en la tabla RegistroProduccion.	Inventario

3.3.1.2 Diseño preliminar del prototipo

Tabla 2. Contenido del prótotipo

Módulo/Formulario	Componentes Clave	Estado Actual
formLogin	Usuario, Contraseña, Botón Iniciar Sesión.	Diseñado

frmAdmin	Menú lateral para navegación (Inventario, Productos, Categorías, Usuarios).	Diseñado
frmInventario (Ingredientes)	*Grid* con el stock actual, botones de Agregar/Editar, campos para registro de entradas (Nombre, Cantidad, Unidad, Vencimiento).	Diseñado
frmReceta	**NUEVO:** Control maestro-detalle. Selección del Producto y *Sub-Grid* para listar DetalleRecetas (Ingrediente, Cantidad Requerida).	Diseñado
frmProduccion	**NUEVO:** Selección de Producto, campo para Cantidad a Producir, Botón Registrar Producción.	Diseñado
formCliente	Vista en tarjetas de los productos, filtro por Categoría, barra de búsqueda.	Diseñado
formCliHistorial	Tabla de historial de pedidos. Muestra ID, Fecha, Total y Estado de cada transacción. Incluye un botón "Ver Detalle" por fila.	Diseñado
frmPedido	Vista de Carrito de Compras, formulario de datos de envío (DireccionEntrega, Observaciones), selección de Método de Pago.	Diseñado
frmProducto	DatagridView para Inserción de productos, catálogo de productos y recuento de stock	Diseñado
frmRegistro	Nombre de Usuario Nuevo, Correo, Contraseña y botón de crear cuenta	Diseñado

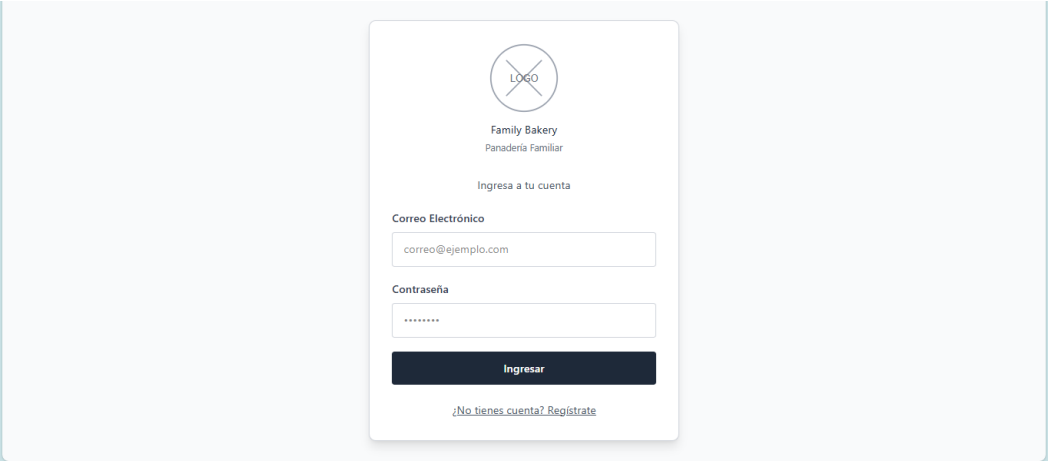

Ilustración 1. Inicio de sesión con usuario conocido

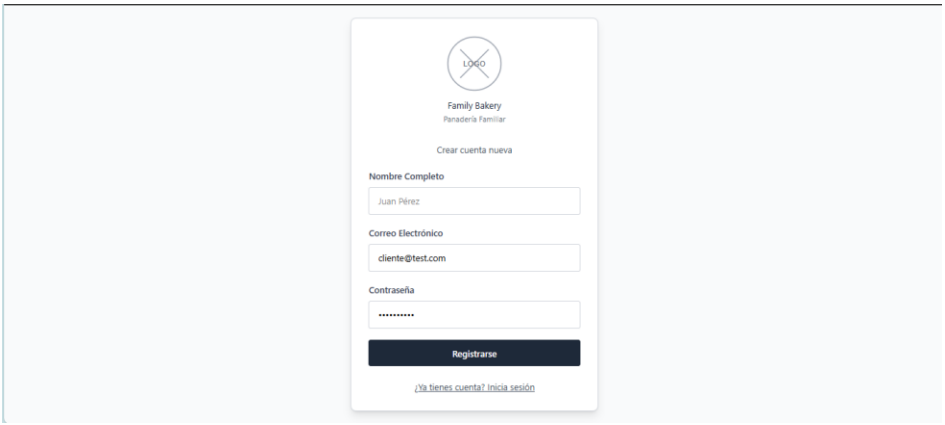

Ilustración 2. Creación de cuenta como nuevo usuario

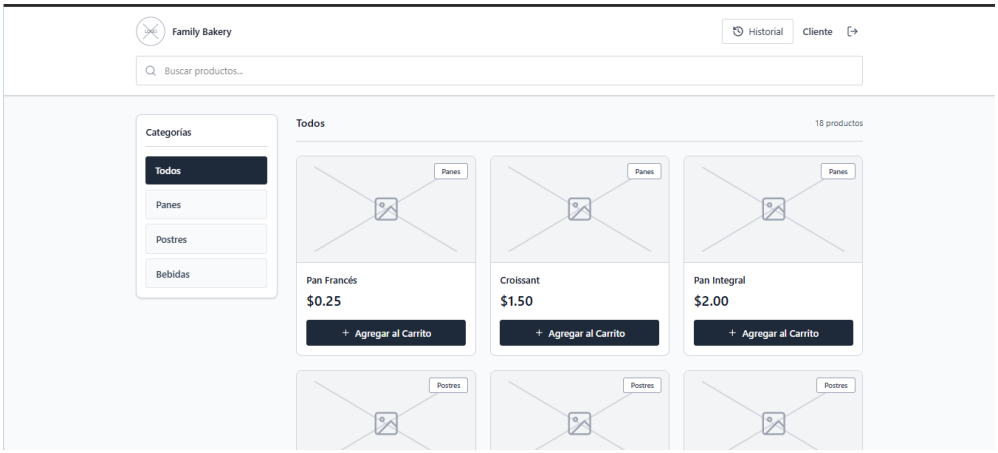

Ilustración 3. Interfaz del cliente para realizar una compra

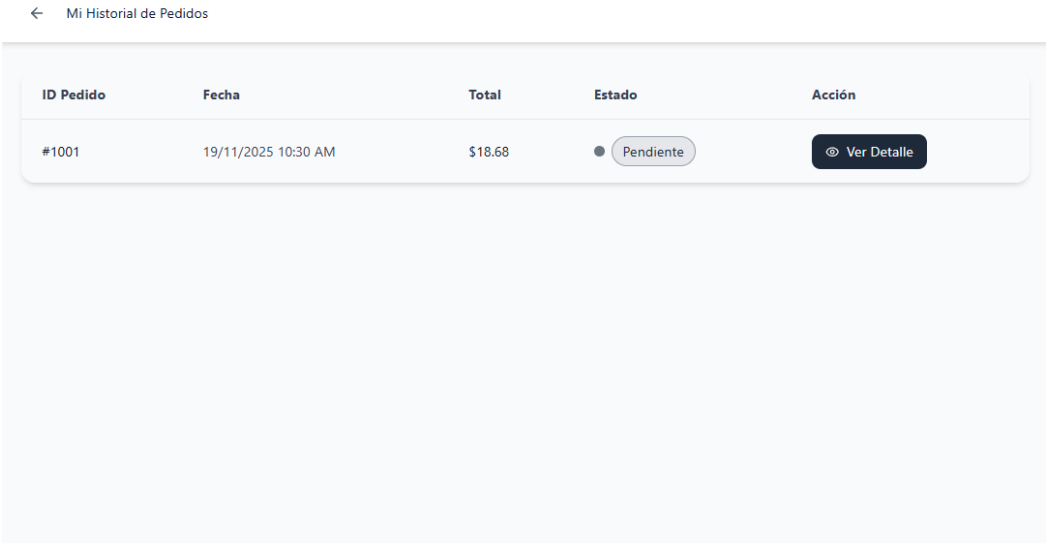

Ilustración 4. Historial de pedidos del cliente

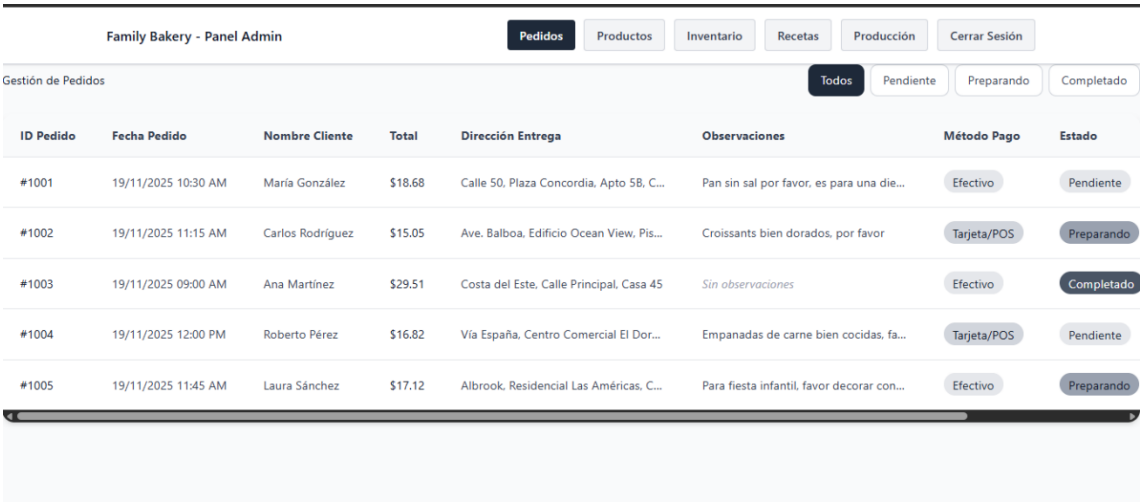

Ilustración 5. Interfaz de pedidos del admin

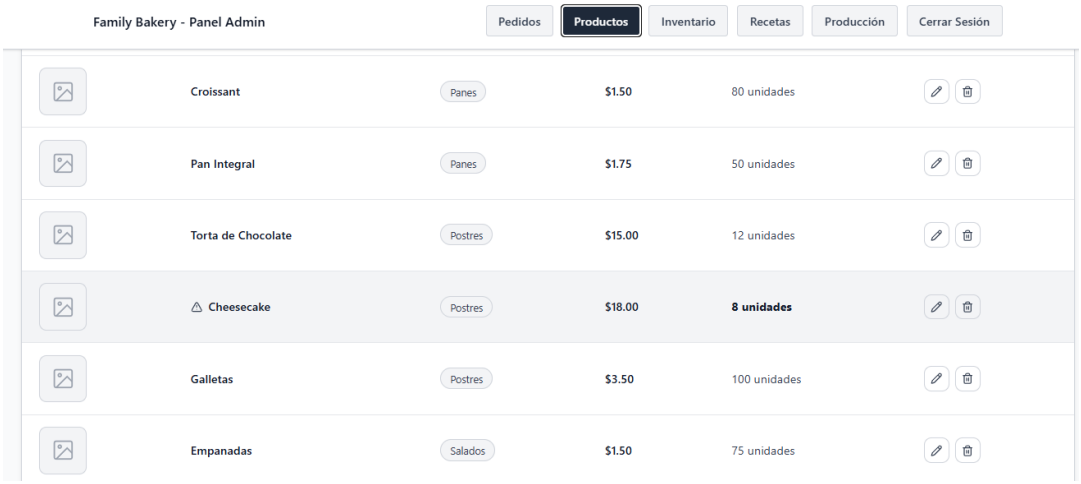

Ilustración 6. Modificación de productos del admin

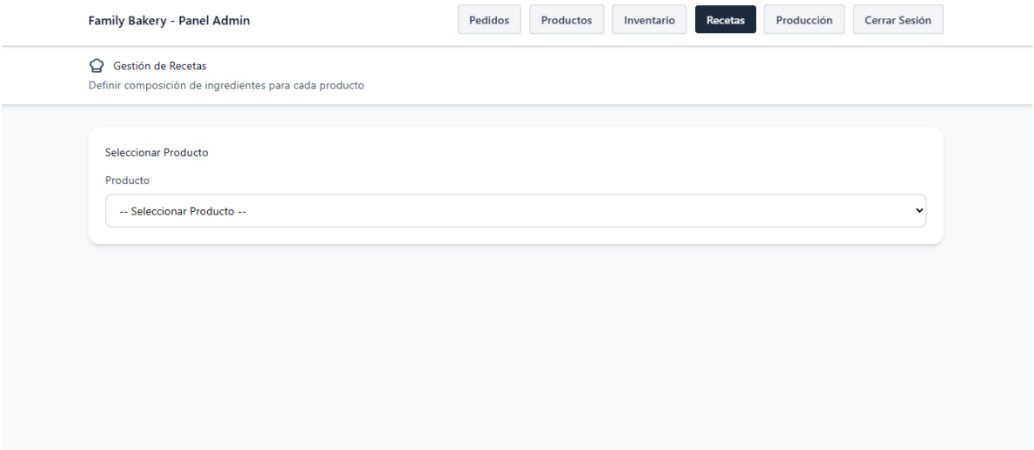

Ilustración 7. Interfaz de recetas del admin

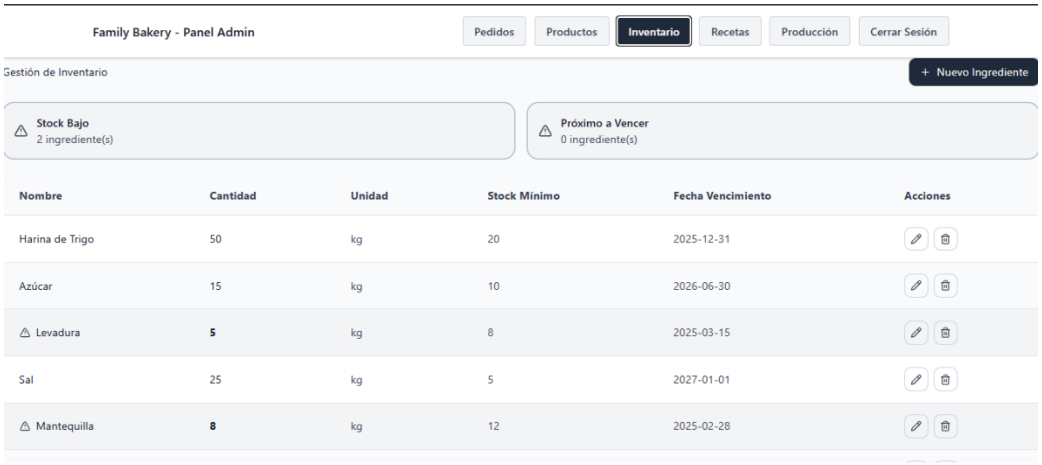

Ilustración 8. Interfaz de inventario del admin

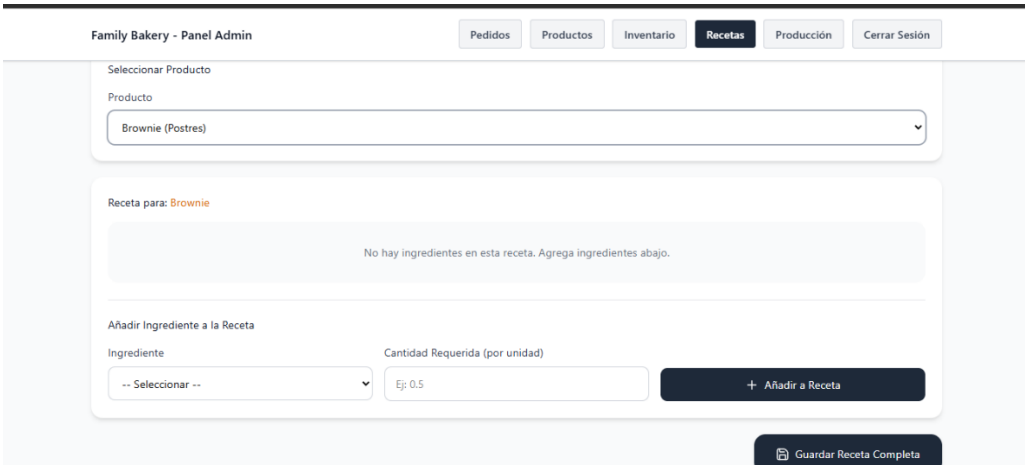

Ilustración 9. Despliegue del form de rectas luego de la selección del admin

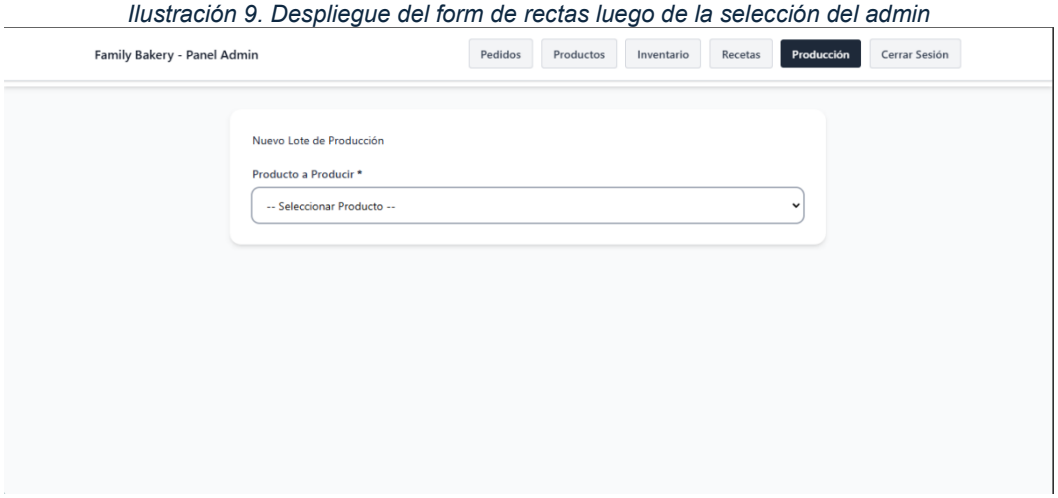

Ilustración 10. Interfaz de producción del admin

3.3.2 Fase de Diseño (2)

3.3.2.1 Diagrama general del sistema

Principales Procesos del Sistema

1. Proceso de Autenticación y Autorización

El sistema inicia con un módulo de login que diferencia entre dos tipos de usuarios:

- **Usuario Cliente:** Accede al catálogo de productos y funcionalidades de compra.
- **Usuario Administrador:** Accede al panel de gestión empresarial tipo ERP.

La autenticación se realiza mediante correo electrónico, y el sistema redirige automáticamente al usuario a su módulo correspondiente según su tipo.

2. Proceso de Compra (Módulo Cliente)

2.1 Navegación del Catálogo:

- Visualización de productos organizados en 4 categorías (Panes, Bebidas, Postres, salados)
- Sistema de búsqueda por nombre de producto
- Filtrado dinámico por categoría
- Tarjetas de producto con imagen, nombre, descripción y precio

2.2 Gestión del Carrito de Compras:

- Agregar productos con selección de cantidad
- Modificar cantidades de productos en el carrito
- Eliminar productos del carrito
- Cálculo automático de subtotales y total general
- Validación de stock disponible

2.3 Finalización del Pedido:

- Selección obligatoria de método de pago (Efectivo, Tarjeta,)
- Campo opcional para observaciones especiales
- Confirmación del pedido con generación de número único
- Registro automático en el historial del cliente
- Descuento automático de stock del inventario

2.4 Historial de Pedidos:

- Visualización de todos los pedidos realizados por el cliente
- Información detallada: fecha, número de pedido, productos, cantidades, precios
- Estado del pedido (Pendiente, En Preparación, Completado, Cancelado)
- Método de pago utilizado
- Total pagado

3. Proceso de Gestión de Inventario (Módulo Administrador)

3.1 Administración de Productos:

- Registro de nuevos productos con: nombre, descripción, precio, categoría, stock, imagen
- Edición de información de productos existentes
- Eliminación de productos del catálogo
- Sistema de alertas visuales para productos con bajo stock (< 10 unidades)

3.2 Administración de Ingredientes:

- Registro de ingredientes con: nombre, cantidad disponible, unidad de medida, precio unitario
- Edición de cantidades y precios de ingredientes
- Eliminación de ingredientes
- Sistema de alertas críticas para ingredientes con stock bajo (< 50 unidades para líquidos/gramos, < 10 para unidades unidades contables)
- Indicadores visuales de nivel de stock (Verde: suficiente, Amarillo: bajo, Rojo: crítico)

4. Proceso de Gestión de Pedidos (Módulo Administrador)

4.1 Visualización y Administración:

- Panel unificado con todos los pedidos del sistema
- Información completa: número, cliente, fecha, productos, cantidades, estado, método de pago, total
- Funcionalidad de búsqueda por número de pedido o nombre de cliente
- Filtrado por estado del pedido

4.2 Gestión de Estados:

- Cambio de estado de pedidos mediante sistema de tabs
- Estados disponibles: Pendiente, En Preparación, Completado, Cancelado
- Actualización en tiempo real del estado visible para clientes en su historial

4.3 Eliminación de Pedidos:

- Capacidad de eliminar pedidos del sistema
- Confirmación de acción para prevenir eliminaciones accidentales

5. Proceso de Gestión de Recetas (Módulo Administrador)

5.1 Creación de Recetas:

- Asociación de productos del catálogo con sus ingredientes necesarios
- Definición de cantidades específicas de cada ingrediente
- Especificación de rendimiento (cantidad de productos que produce la receta)
- Validación de ingredientes disponibles en el inventario

5.2 Consulta de Recetas:

- Visualización de todas las recetas registradas
- Detalle completo de ingredientes y cantidades por receta
- Cálculo automático de costos de producción basado en precios de ingredientes
- Información de rendimiento por receta

5.3 Edición y Eliminación:

- Modificación de ingredientes y cantidades en recetas existentes
- Eliminación de recetas del sistema

6. Proceso de Registro de Producción (Módulo Administrador)

6.1 Registro de Lotes de Producción:

- Selección del producto a producir (solo productos con receta registrada)
- Especificación de cantidad a producir
- Validación automática de disponibilidad de ingredientes
- Sistema de alertas si hay ingredientes insuficientes

6.2 Descuento Automático de Inventario:

- Cálculo proporcional de ingredientes necesarios según cantidad a producir
- Descuento automático de ingredientes del inventario al confirmar producción
- Actualización en tiempo real del stock de ingredientes
- Registro de la fecha y hora de producción

6.3 Actualización de Stock de Productos:

- Incremento automático del stock del producto terminado
- Sincronización con el módulo de inventario de productos
- Disponibilidad inmediata para venta en el catálogo cliente

6.4 Historial de Producción:

- Registro completo de todos los lotes producidos
- Información detallada: producto, cantidad, ingredientes consumidos, fecha de producción
- Trazabilidad completa del proceso productivo

Flujo de Datos General

1. **Flujo Cliente → Sistema:**

- Cliente selecciona productos → Carrito → Confirma pedido con método de pago → Sistema registra pedido → Descuenta stock de productos→ Actualiza historial del cliente

2. **Flujo Administrador → Inventario:**

- Admin registra productos/ingredientes → Sistema almacena datos → Productos disponibles en catálogo cliente

3. **Flujo Administrador → Producción:**

- Admin crea receta → Asocia ingredientes → Registra producción → Sistema descuenta ingredientes → Incrementa stock de producto → Producto disponible para venta

4. **Flujo Bidireccional Pedidos:**

- Cliente genera pedido → Admin visualiza en panel → Admin actualiza estado → Cliente ve actualización en historial

3.3.2.2 Diagrama de la Base de Datos

Tabla 3. Información de la BD

Nombre	Proveedor	Especificación	Servidor utilizado
PanaderiaDB	**Microsoft SQL Server** (Usando T-SQL)	Base de datos Relacional (RDBMS), **3FN**, incluye **Transacciones** y **Procedimientos Almacenados**.	**MELANY**

ER de la BD

Tabla 4. Relaciones en la BD

Objeto que se Crea (El "Hijo")	Identificador de Enlace	Objeto que lo Autoriza (El "Dueño")	Conexión (Regla de Cantidad)	Regla de Funcionamiento de la Panadería
Productos	IdCategoria	**Categorías**	**1 autoriza a Muchos** (1, N)	Un producto nuevo **siempre** necesita un permiso (el código de Categoría) para entrar al catálogo.
Pedidos	IdUsuario	**Usuarios**	**1 autoriza a Muchos** (0, N)	El **Usuario** es el cliente. Puede registrarse (1) pero hacer **Cero o Muchas** compras.
Recetas	IdProducto	**Productos**	**1 autoriza a Máximo Uno** (0, 1)	Un producto terminado **puede** tener una receta, pero **nunca dos**. La receta necesita el código de un producto existente.
RegistroProduccion	IdProducto	**Productos**	**1 autoriza a Muchos** (0, N)	El registro de horneado (hijo) es el **historial** de producción del producto (dueño).

				Puede no tener historial aún.
DetallePedidos	IdPedido	**Pedidos**	**Es obligatorio** (1, N)	La línea de producto existe **solo como parte** del ticket de compra (el Pedido). El ticket no puede estar vacío.
DetallePedidos	IdProducto	**Productos**	**Cero o Muchos** (0, N)	El detalle de la línea apunta al producto que fue vendido, enlazándolo con el catálogo de productos.
DetalleRecetas	IdReceta	**Recetas**	**Es obligatorio** (1, N)	El detalle de ingrediente existe **solo como parte** de una receta. Una receta *debe* tener al menos un ingrediente.
DetalleRecetas	IdIngrediente	**Ingredientes**	**Cero o Muchos** (1, N)	La línea apunta al ingrediente específico. Un ingrediente (el dueño) se puede usar en muchas recetas.

Diagrama Ilustrativo

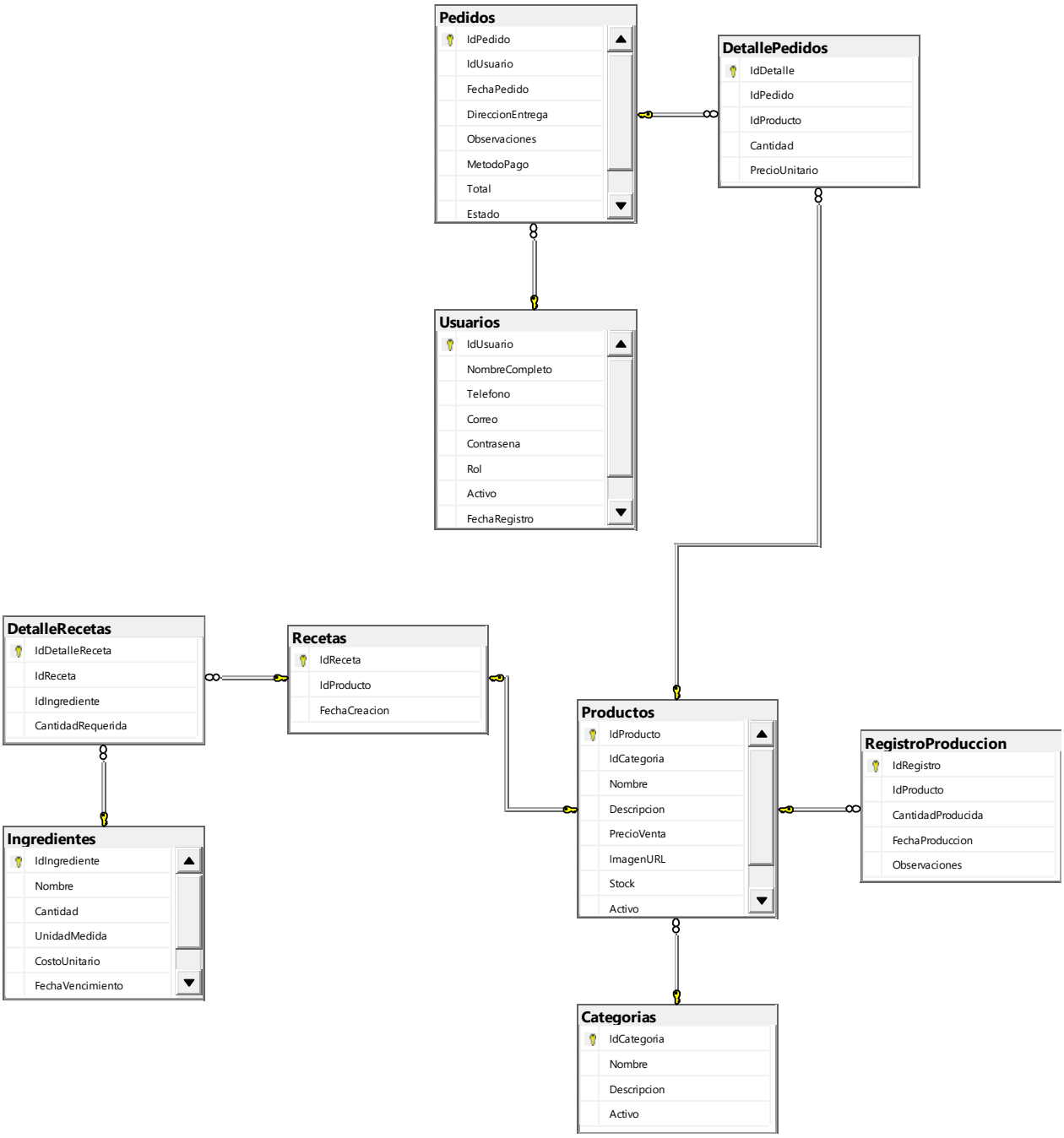

Ilustración 11. Diagrama de la BD

Script de la BD

```
-- 1. CREACIÓN DE BASE DE DATOS Y USO

IF DB_ID('PanaderiaDB') IS NULL
 BEGIN
   CREATE DATABASE PanaderiaDB;
 END;
```

```
GO -- El GO debe estar en su propia línea
USE PanaderiaDB;
GO


-- 2. DDL - Definición de Estructura de Tablas
-- (Uso de NVARCHAR para texto y ajustes en Pedidos/Estados)


-- 2.1. Tabla de Usuarios
CREATE TABLE Usuarios (
    IdUsuario       INT PRIMARY KEY IDENTITY(1,1),
    NombreCompleto  NVARCHAR(100) NOT NULL,
    Telefono        NVARCHAR(20) NULL,
    Correo          NVARCHAR(100) NOT NULL UNIQUE,
    Contrasena      VARCHAR(100) NOT NULL,
    Rol             NVARCHAR(20) NOT NULL DEFAULT 'Cliente',
    Activo          BIT NOT NULL DEFAULT 1,
    FechaRegistro   DATETIME NOT NULL DEFAULT GETDATE(),
    CONSTRAINT CK_Usuarios_Rol CHECK (Rol IN ('Administrador', 'Cliente'))
 );

-- 2.2. Tabla de Categorías
CREATE TABLE Categorias (
    IdCategoria INT PRIMARY KEY IDENTITY(1,1),
    Nombre      NVARCHAR(50) NOT NULL,
    Descripcion NVARCHAR(100) NULL,
    Activo      BIT NOT NULL DEFAULT 1
 );

-- 2.3. Tabla de Productos
CREATE TABLE Productos (
    IdProducto   INT PRIMARY KEY IDENTITY(1,1),
    IdCategoria  INT NOT NULL,
    Nombre       NVARCHAR(50) NOT NULL,
    Descripcion  NVARCHAR(200) NULL,
    PrecioVenta  DECIMAL(10,2) NOT NULL,
    ImagenURL    VARCHAR(255) NULL,
    Stock        INT NOT NULL DEFAULT 0,
    Activo       BIT NOT NULL DEFAULT 1,
    CONSTRAINT FK_Productos_Categorias
       FOREIGN KEY (IdCategoria) REFERENCES Categorias(IdCategoria),
    CONSTRAINT CK_Productos_PrecioVenta CHECK (PrecioVenta >= 0),
    CONSTRAINT CK_Productos_Stock CHECK (Stock >= 0)
 );
```

```
-- 2.4. Tabla de Ingredientes (Inventario)
CREATE TABLE Ingredientes (
    IdIngrediente INT PRIMARY KEY IDENTITY(1,1),
    Nombre        NVARCHAR(50) NOT NULL,
    Cantidad      DECIMAL(10,2) NOT NULL,
    UnidadMedida  NVARCHAR(20) NOT NULL,
    CostoUnitario DECIMAL(10,2) NULL,
    FechaVencimiento DATE NULL,
    CONSTRAINT CK_Ingredientes_Cantidad CHECK (Cantidad >= 0),
    CONSTRAINT CK_Ingredientes_CostoUnitario
        CHECK (CostoUnitario IS NULL OR CostoUnitario >= 0)
 );

-- 2.5. Tabla de Recetas (Cabecera)
CREATE TABLE Recetas (
    IdReceta      INT PRIMARY KEY IDENTITY(1,1),
    -- **CORRECCIÓN: Se mantiene UNIQUE (1 producto = 1 receta) según tu diseño original**
    IdProducto    INT NOT NULL UNIQUE,
    FechaCreacion DATETIME NOT NULL DEFAULT GETDATE(),
    CONSTRAINT FK_Recetas_Productos
        FOREIGN KEY (IdProducto) REFERENCES Productos(IdProducto)
 );

-- 2.6. Tabla Detalle de Recetas
CREATE TABLE DetalleRecetas (
    IdDetalleReceta   INT PRIMARY KEY IDENTITY(1,1),
    IdReceta          INT NOT NULL,
    IdIngrediente     INT NOT NULL,
    CantidadRequerida DECIMAL(10,4) NOT NULL,
    CONSTRAINT FK_DetalleRecetas_Recetas
        FOREIGN KEY (IdReceta) REFERENCES Recetas(IdReceta),
    CONSTRAINT FK_DetalleRecetas_Ingredientes
        FOREIGN KEY (IdIngrediente) REFERENCES Ingredientes(IdIngrediente),
    CONSTRAINT UQ_DetalleRecetas UNIQUE (IdReceta, IdIngrediente),
    CONSTRAINT CK_DetalleRecetas_Cantidad CHECK (CantidadRequerida > 0)
 );

-- 2.7. Registro de Producción (log de horneadas)
CREATE TABLE RegistroProduccion (
    IdRegistro        INT PRIMARY KEY IDENTITY(1,1),
    IdProducto        INT NOT NULL,
    CantidadProducida INT NOT NULL,
    FechaProduccion   DATETIME NOT NULL DEFAULT GETDATE(),
    Observaciones     NVARCHAR(500) NULL,
```

```
    CONSTRAINT FK_RegistroProduccion_Productos
        FOREIGN KEY (IdProducto) REFERENCES Productos(IdProducto),
    CONSTRAINT CK_RegistroProduccion_Cantidad CHECK (CantidadProducida > 0)
);

-- 2.8. Tabla de Pedidos (Cabecera)
CREATE TABLE Pedidos (
    IdPedido        INT PRIMARY KEY IDENTITY(1,1),
    IdUsuario       INT NOT NULL,
    FechaPedido     DATETIME NOT NULL DEFAULT GETDATE(),
    DireccionEntrega NVARCHAR(255) NULL,
    Observaciones   NVARCHAR(500) NULL,
    MetodoPago      NVARCHAR(50) NOT NULL,
    Total           DECIMAL(10,2) NOT NULL,
    Estado          NVARCHAR(20) NOT NULL DEFAULT 'Pendiente',
    CONSTRAINT FK_Pedidos_Usuarios
        FOREIGN KEY (IdUsuario) REFERENCES Usuarios(IdUsuario),
    CONSTRAINT CK_Pedidos_Total CHECK (Total >= 0),
    CONSTRAINT CK_Pedidos_Estado
        CHECK (Estado IN ('Pendiente','En Preparacion','Completado','Cancelado'))
);

-- 2.9. Detalle de Pedidos
CREATE TABLE DetallePedidos (
    IdDetalle      INT PRIMARY KEY IDENTITY(1,1),
    IdPedido       INT NOT NULL,
    IdProducto     INT NOT NULL,
    Cantidad       INT NOT NULL,
    PrecioUnitario DECIMAL(10,2) NOT NULL,
    CONSTRAINT FK_DetallePedidos_Pedidos
        FOREIGN KEY (IdPedido)REFERENCES Pedidos(IdPedido),
    CONSTRAINT FK_DetallePedidos_Productos
        FOREIGN KEY (IdProducto) REFERENCES Productos(IdProducto),
    CONSTRAINT CK_DetallePedidos_Cantidad CHECK (Cantidad > 0),
    CONSTRAINT CK_DetallePedidos_Precio CHECK (PrecioUnitario >= 0),
    CONSTRAINT UQ_DetallePedidos UNIQUE (IdPedido, IdProducto)
);
GO

--- 3. SPs de Producción y Venta (Gestión de Stock)

-- 3.1. Registrar Producción (Descarga ingredientes, aumenta stock)
CREATE PROCEDURE SP_RegistrarProduccion
    @IdProducto       INT,
    @CantidadProducida INT,
```

```
    @Observaciones    NVARCHAR(500) = NULL
AS
BEGIN
    SET NOCOUNT ON;

    -- Validaciones
    IF @IdProducto IS NULL OR @IdProducto <= 0
        RETURN;
    IF @CantidadProducida IS NULL OR @CantidadProducida <= 0
        RETURN;
    IF NOT EXISTS (SELECT 1 FROM Productos WHERE IdProducto = @IdProducto AND Activo =
1)
        RETURN;
    IF NOT EXISTS (SELECT 1 FROM Recetas WHERE IdProducto = @IdProducto)
        RETURN;

    -- **CORRECCIÓN: Obtener @IdReceta antes de la transacción**
    DECLARE @IdReceta INT;
    SELECT @IdReceta = IdReceta FROM Recetas WHERE IdProducto = @IdProducto;

    BEGIN TRANSACTION;
    BEGIN TRY

        -- PASO 1: Calcular los requerimientos totales (SET-BASED)
        SELECT
            dr.IdIngrediente,
            dr.CantidadRequerida * @CantidadProducida AS CantidadConsumir
        INTO
            #Requerimientos
        FROM
            DetalleRecetas dr
        WHERE
            dr.IdReceta = @IdReceta;

        -- PASO 2: Validación de Stock Insuficiente
        IF EXISTS (
            SELECT 1
            FROM #Requerimientos req
            JOIN Ingredientes i WITH (UPDLOCK) ON req.IdIngrediente = i.IdIngrediente
            WHERE i.Cantidad < req.CantidadConsumir
        )
        BEGIN
            RAISERROR('Stock insuficiente de uno o más ingredientes para la producción solicitada.',
16, 1);
            -- Se lanza error para ser capturado por el CATCH y hacer ROLLBACK
```

```
            -- Se añade un RETURN para salir del TRY en caso de error
            RETURN;
        END;

        -- PASO 3: Descontar Inventario de Ingredientes (SET-BASED)
        UPDATE i
        SET i.Cantidad = i.Cantidad - req.CantidadConsumir
        FROM Ingredientes i
        JOIN #Requerimientos req ON i.IdIngrediente = req.IdIngrediente;

        -- PASO 4: Aumentar stock de producto terminado
        UPDATE Productos
          SET Stock = Stock + @CantidadProducida
        WHERE IdProducto = @IdProducto;

        -- PASO 5: Registrar en log de producción
        INSERT INTO RegistroProduccion (IdProducto, CantidadProducida, FechaProduccion,
Observaciones)
        VALUES (@IdProducto, @CantidadProducida, GETDATE(), @Observaciones);

        -- Limpieza
        DROP TABLE #Requerimientos;

        COMMIT TRANSACTION;
        SELECT 1 AS Resultado, 'Producción registrada y stock actualizado correctamente.' AS
Mensaje;

    END TRY
    BEGIN CATCH
        IF OBJECT_ID('tempdb..#Requerimientos') IS NOT NULL
            DROP TABLE #Requerimientos;

        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;

        SELECT 0 AS Resultado, 'Fallo en la Producción: ' + ERROR_MESSAGE() AS Mensaje;
    END CATCH
END;
GO

-- 3.2. Registrar Venta (Descuento de stock del producto terminado)
CREATE PROCEDURE SP_RegistrarVenta
    @IdProducto     INT,
    @CantidadVendida INT
AS
```

```
BEGIN

    SET NOCOUNT ON;

    IF @IdProducto IS NULL OR @IdProducto <= 0
        RETURN;
    IF @CantidadVendida IS NULL OR @CantidadVendida <= 0
        RETURN;
    IF NOT EXISTS (SELECT 1 FROM Productos WHERE IdProducto = @IdProducto AND Activo =
1)
        RETURN;

    BEGIN TRANSACTION;
    BEGIN TRY
        DECLARE @StockActual INT;

        SELECT @StockActual = Stock
        FROM Productos WITH (UPDLOCK)
        WHERE IdProducto = @IdProducto;

        IF @StockActual < @CantidadVendida
        BEGIN
            RAISERROR('Stock de producto terminado insuficiente para la venta.', 16, 1);
            -- Se lanza error para ser capturado por el CATCH y hacer ROLLBACK
            RETURN;
        END;

        UPDATE Productos
        SET Stock = Stock - @CantidadVendida
        WHERE IdProducto = @IdProducto;

        COMMIT TRANSACTION;
        SELECT 1 AS Resultado, 'Venta descontada de stock correctamente.' AS Mensaje;
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;
        SELECT 0 AS Resultado, 'Fallo al registrar la venta: ' + ERROR_MESSAGE() AS Mensaje;
    END CATCH
END;
GO

--- 4. SPs de Gestión de Productos

-- 4.1. Listar Productos
```

```
CREATE PROCEDURE SP_ListarProductos
AS
BEGIN
    SET NOCOUNT ON;

    SELECT
        p.IdProducto,
        p.IdCategoria,
        c.Nombre AS NombreCategoria,
        p.Nombre,
        p.Descripcion,
        p.PrecioVenta,
        p.ImagenURL,
        p.Stock,
        p.Activo
    FROM
        Productos p
    JOIN
        Categorias c ON p.IdCategoria = c.IdCategoria
    ORDER BY
        p.Nombre;
END;
GO

-- 4.2. Insertar Producto
CREATE PROCEDURE SP_InsertarProducto
    @IdCategoria INT,
    @Nombre NVARCHAR(50),
    @Descripcion NVARCHAR(200) = NULL,
    @PrecioVenta DECIMAL(10,2),
    @ImagenURL VARCHAR(255) = NULL,
    @Stock INT = 0
AS
BEGIN
    SET NOCOUNT ON;

    IF @IdCategoria IS NULL OR @IdCategoria <= 0 OR NOT EXISTS (SELECT 1 FROM Categorias
WHERE IdCategoria = @IdCategoria AND Activo = 1)
        RETURN;

    IF @Nombre IS NULL OR LTRIM(RTRIM(@Nombre)) = ''
        RETURN;

    IF @PrecioVenta IS NULL OR @PrecioVenta < 0
        RETURN;
```

```
    BEGIN TRY
        INSERT INTO Productos (IdCategoria, Nombre, Descripcion, PrecioVenta, ImagenURL, Stock,
Activo)
        VALUES (@IdCategoria, @Nombre, @Descripcion, @PrecioVenta, @ImagenURL, @Stock, 1);

        SELECT 1 AS Resultado, 'Producto insertado correctamente.' AS Mensaje,
SCOPE_IDENTITY() AS IdGenerado;
    END TRY
    BEGIN CATCH
        SELECT 0 AS Resultado, 'Fallo al insertar el producto: ' + ERROR_MESSAGE() AS Mensaje,
NULL AS IdGenerado;
    END CATCH
END;
GO

-- 4.3. Actualizar Producto
CREATE PROCEDURE SP_ActualizarProducto
    @IdProducto INT,
    @IdCategoria INT,
    @Nombre NVARCHAR(50),
    @Descripcion NVARCHAR(200) = NULL,
    @PrecioVenta DECIMAL(10,2),
    @ImagenURL VARCHAR(255) = NULL,
    @Activo BIT
AS
BEGIN
    SET NOCOUNT ON;

    IF @IdProducto IS NULL OR @IdProducto <= 0 OR NOT EXISTS (SELECT 1 FROM Productos
WHERE IdProducto = @IdProducto)
        RETURN;

    IF @IdCategoria IS NULL OR @IdCategoria <= 0 OR NOT EXISTS (SELECT 1 FROM Categorias
WHERE IdCategoria = @IdCategoria)
        RETURN;

    IF @Nombre IS NULL OR LTRIM(RTRIM(@Nombre)) = ''
        RETURN;

    IF @PrecioVenta IS NULL OR @PrecioVenta < 0
        RETURN;

    BEGIN TRY
        UPDATE Productos
```

```
        SET
            IdCategoria = @IdCategoria,
            Nombre = @Nombre,
            Descripcion = @Descripcion,
            PrecioVenta = @PrecioVenta,
            ImagenURL = @ImagenURL,
            Activo = @Activo
        WHERE
            IdProducto = @IdProducto;

        SELECT 1 AS Resultado, 'Producto actualizado correctamente.' AS Mensaje;
    END TRY
    BEGIN CATCH
        SELECT 0 AS Resultado, 'Fallo al actualizar el producto: ' + ERROR_MESSAGE() AS Mensaje;
    END CATCH
END;
GO

-- 4.4. Eliminar Producto (Eliminación Lógica)
CREATE PROCEDURE SP_EliminarProducto
    @IdProducto INT
AS
BEGIN
    SET NOCOUNT ON;

    IF @IdProducto IS NULL OR @IdProducto <= 0 OR NOT EXISTS (SELECT 1 FROM Productos
WHERE IdProducto = @IdProducto)
        RETURN;

    BEGIN TRY
        UPDATE Productos
        SET Activo = 0
        WHERE IdProducto = @IdProducto;

        SELECT 1 AS Resultado, 'Producto deshabilitado (eliminación lógica) correctamente.' AS
Mensaje;
    END TRY
    BEGIN CATCH
        SELECT 0 AS Resultado, 'Fallo al deshabilitar el producto: ' + ERROR_MESSAGE() AS
Mensaje;
    END CATCH
END;
GO

---5. SPs de Gestión de Ingredientes
```

```
-- 5.1. Listar ingredientes
CREATE PROCEDURE SP_ListarIngredientes
AS
BEGIN
   SET NOCOUNT ON;

   SELECT
      IdIngrediente, Nombre, Cantidad, UnidadMedida, CostoUnitario, FechaVencimiento
   FROM Ingredientes
   ORDER BY Nombre;
END;
GO

-- 5.2. Insertar ingrediente
CREATE PROCEDURE SP_InsertarIngrediente
   @Nombre       NVARCHAR(50),
   @Cantidad     DECIMAL(10,2),
   @UnidadMedida  NVARCHAR(20),
   @CostoUnitario DECIMAL(10,2),
   @FechaVencimiento DATE = NULL
AS
BEGIN
   SET NOCOUNT ON;
   IF @Nombre IS NULL OR LTRIM(RTRIM(@Nombre)) = '' RETURN;
   IF @Cantidad IS NULL OR @Cantidad < 0 RETURN;
   IF @CostoUnitario IS NOT NULL AND @CostoUnitario < 0 RETURN;

   BEGIN TRY
      INSERT INTO Ingredientes (Nombre, Cantidad, UnidadMedida, CostoUnitario,
FechaVencimiento)
      VALUES (@Nombre, @Cantidad, @UnidadMedida, @CostoUnitario, @FechaVencimiento);
      SELECT 1 AS Resultado, 'Ingrediente insertado correctamente.' AS Mensaje,
SCOPE_IDENTITY() AS IdGenerado;
   END TRY
   BEGIN CATCH
      SELECT 0 AS Resultado, 'Error al insertar el ingrediente: ' + ERROR_MESSAGE() AS
Mensaje, NULL AS IdGenerado;
   END CATCH
END;
GO

-- 5.3. Actualizar ingrediente
CREATE PROCEDURE SP_ActualizarIngrediente
   @IdIngrediente INT,
```

```
    @Nombre        NVARCHAR(50),
    @Cantidad      DECIMAL(10,2),
    @UnidadMedida  NVARCHAR(20),
    @CostoUnitario DECIMAL(10,2),
    @FechaVencimiento DATE = NULL
AS
BEGIN
    SET NOCOUNT ON;
    IF @IdIngrediente IS NULL OR @IdIngrediente <= 0 RETURN;
    IF NOT EXISTS (SELECT 1 FROM Ingredientes WHERE IdIngrediente = @IdIngrediente)
RETURN;
    IF @Nombre IS NULL OR LTRIM(RTRIM(@Nombre)) = '' RETURN;
    IF @Cantidad IS NULL OR @Cantidad < 0 RETURN;
    IF @CostoUnitario IS NOT NULL AND @CostoUnitario < 0 RETURN;

    BEGIN TRY
        UPDATE Ingredientes
        SET Nombre = @Nombre, Cantidad = @Cantidad, UnidadMedida = @UnidadMedida,
CostoUnitario = @CostoUnitario, FechaVencimiento = @FechaVencimiento
        WHERE IdIngrediente = @IdIngrediente;
        SELECT 1 AS Resultado, 'Ingrediente actualizado correctamente.' AS Mensaje;
    END TRY
    BEGIN CATCH
        SELECT 0 AS Resultado, 'Error al actualizar el ingrediente: ' + ERROR_MESSAGE() AS
Mensaje;
    END CATCH
END;
GO

-- 5.4. Eliminar ingrediente
CREATE PROCEDURE SP_EliminarIngrediente
    @IdIngrediente INT
AS
BEGIN
    SET NOCOUNT ON;
    IF @IdIngrediente IS NULL OR @IdIngrediente <= 0 RETURN;
    IF NOT EXISTS (SELECT 1 FROM Ingredientes WHERE IdIngrediente = @IdIngrediente)
RETURN;

    IF EXISTS (SELECT 1 FROM DetalleRecetas WHERE IdIngrediente = @IdIngrediente)
    BEGIN
        SELECT 0 AS Resultado, 'Error: No se puede eliminar. El ingrediente está siendo usado en una
o más recetas.' AS Mensaje;
        RETURN;
    END;
```

```
    BEGIN TRY
        DELETE FROM Ingredientes WHERE IdIngrediente = @IdIngrediente;
        SELECT 1 AS Resultado, 'Ingrediente eliminado correctamente.' AS Mensaje;
    END TRY
    BEGIN CATCH
        SELECT 0 AS Resultado, 'Error al eliminar el ingrediente: ' + ERROR_MESSAGE() AS
Mensaje;
    END CATCH
END;
GO

---6. SPs de Gestión de Recetas

-- 6.1. Insertar receta completa (cabecera + detalle)
CREATE PROCEDURE SP_InsertarRecetaCompleta
    @IdProducto INT,
    @DetalleRecetasXML XML
AS
BEGIN
    SET NOCOUNT ON;
    IF @IdProducto IS NULL OR @IdProducto <= 0 RETURN;
    IF NOT EXISTS (SELECT 1 FROM Productos WHERE IdProducto = @IdProducto AND Activo =
1) RETURN;
    IF EXISTS (SELECT 1 FROM Recetas WHERE IdProducto = @IdProducto) RETURN;
    IF @DetalleRecetasXML IS NULL OR @DetalleRecetasXML.exist('/Detalles/Detalle') = 0
RETURN;

    BEGIN TRANSACTION;
    BEGIN TRY
        INSERT INTO Recetas (IdProducto, FechaCreacion)
        VALUES (@IdProducto, GETDATE());

        DECLARE @NuevoIdReceta INT = SCOPE_IDENTITY();

        INSERT INTO DetalleRecetas (IdReceta, IdIngrediente, CantidadRequerida)
        SELECT @NuevoIdReceta, Tbl.Col.value('(IdIngrediente)[1]', 'INT'),
Tbl.Col.value('(CantidadRequerida)[1]','DECIMAL(10,4)')
        FROM @DetalleRecetasXML.nodes('/Detalles/Detalle') AS Tbl(Col);

        COMMIT TRANSACTION;
        SELECT 1 AS Resultado, 'Receta registrada completamente con éxito.' AS Mensaje,
@NuevoIdReceta AS IdReceta;
    END TRY
    BEGIN CATCH
```

```
        IF @@TRANCOUNT > 0 ROLLBACK TRANSACTION;
        SELECT 0 AS Resultado, 'Fallo al guardar la receta: ' + ERROR_MESSAGE() AS Mensaje,
NULL AS IdReceta;
    END CATCH
END;
GO

-- 6.2. Obtener detalle de receta por producto
CREATE PROCEDURE SP_ObtenerDetalleRecetaPorProducto
    @IdProducto INT
AS
BEGIN
    SET NOCOUNT ON;

    SELECT
        dr.IdDetalleReceta, r.IdReceta, dr.IdIngrediente, i.Nombre AS NombreIngrediente,
dr.CantidadRequerida, i.UnidadMedida
    FROM Recetas r
    JOIN DetalleRecetas dr ON r.IdReceta = dr.IdReceta
    JOIN Ingredientes i ON dr.IdIngrediente = i.IdIngrediente
    WHERE r.IdProducto = @IdProducto;
END;
GO

-- 6.3. Eliminar receta completa
CREATE PROCEDURE SP_EliminarReceta
    @IdProducto INT
AS
BEGIN
    SET NOCOUNT ON;
    IF @IdProducto IS NULL OR @IdProducto <= 0 RETURN;

    DECLARE @IdReceta INT;
    SELECT @IdReceta = IdReceta FROM Recetas WHERE IdProducto = @IdProducto;

    IF @IdReceta IS NULL RETURN;

    BEGIN TRANSACTION;
    BEGIN TRY
        DELETE FROM DetalleRecetas WHERE IdReceta = @IdReceta;
        DELETE FROM Recetas WHERE IdReceta = @IdReceta;
        COMMIT TRANSACTION;
        SELECT 1 AS Resultado, 'Receta eliminada correctamente.' AS Mensaje;
    END TRY
    BEGIN CATCH
```

```
        IF @@TRANCOUNT > 0 ROLLBACK TRANSACTION;
        SELECT 0 AS Resultado, 'Fallo al eliminar la receta: ' + ERROR_MESSAGE() AS Mensaje;
    END CATCH
END;
GO
```

3.3.3 Fase de Desarrollo (3)

3.3.3.1 Pruebas (Ejecución y Resultados)

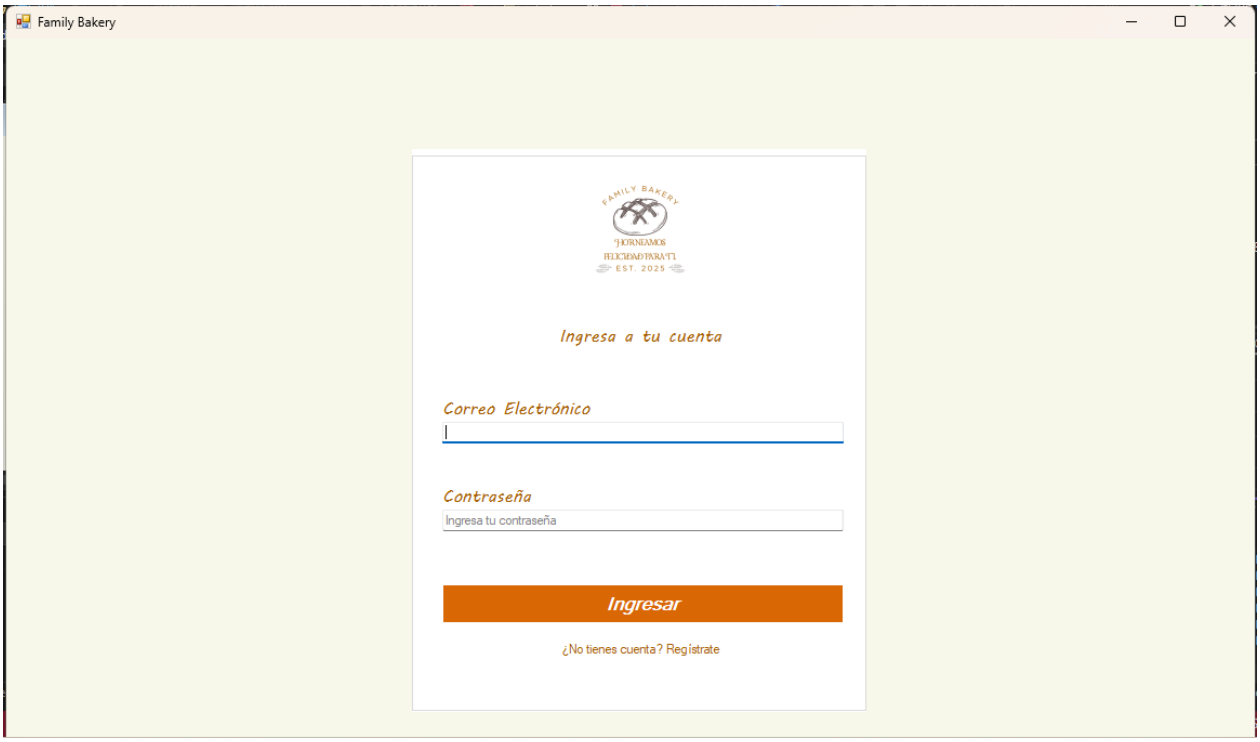

Ilustración 12. Formulario de Inicio de sesión

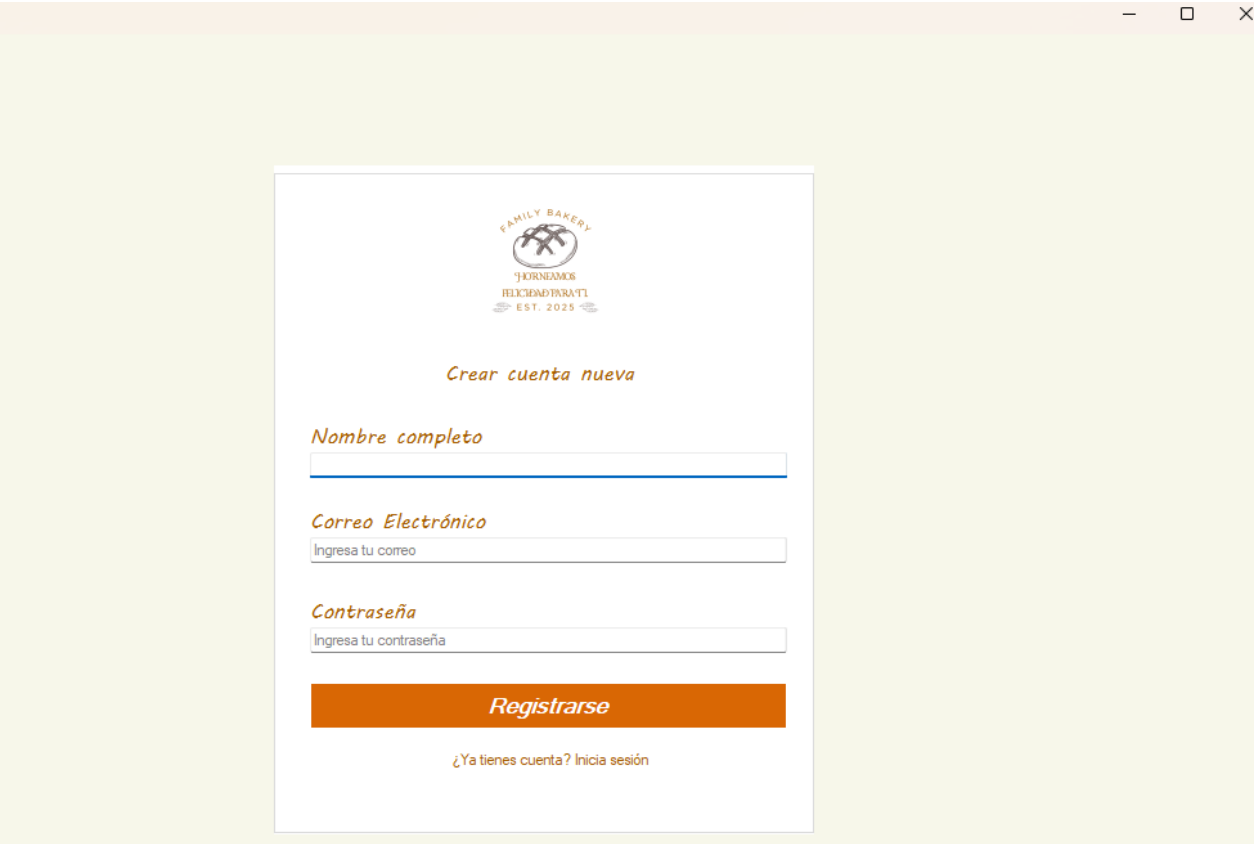

Ilustración 13. Formulario de Registro

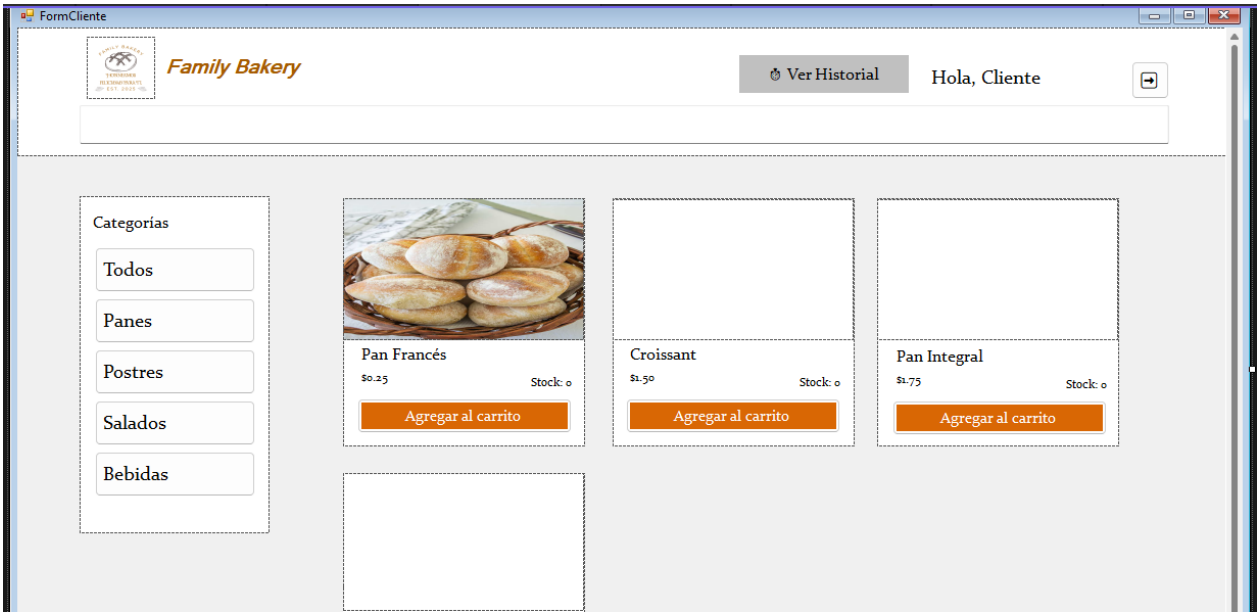

Ilustración 14. Formulario de Interfaz del cliente

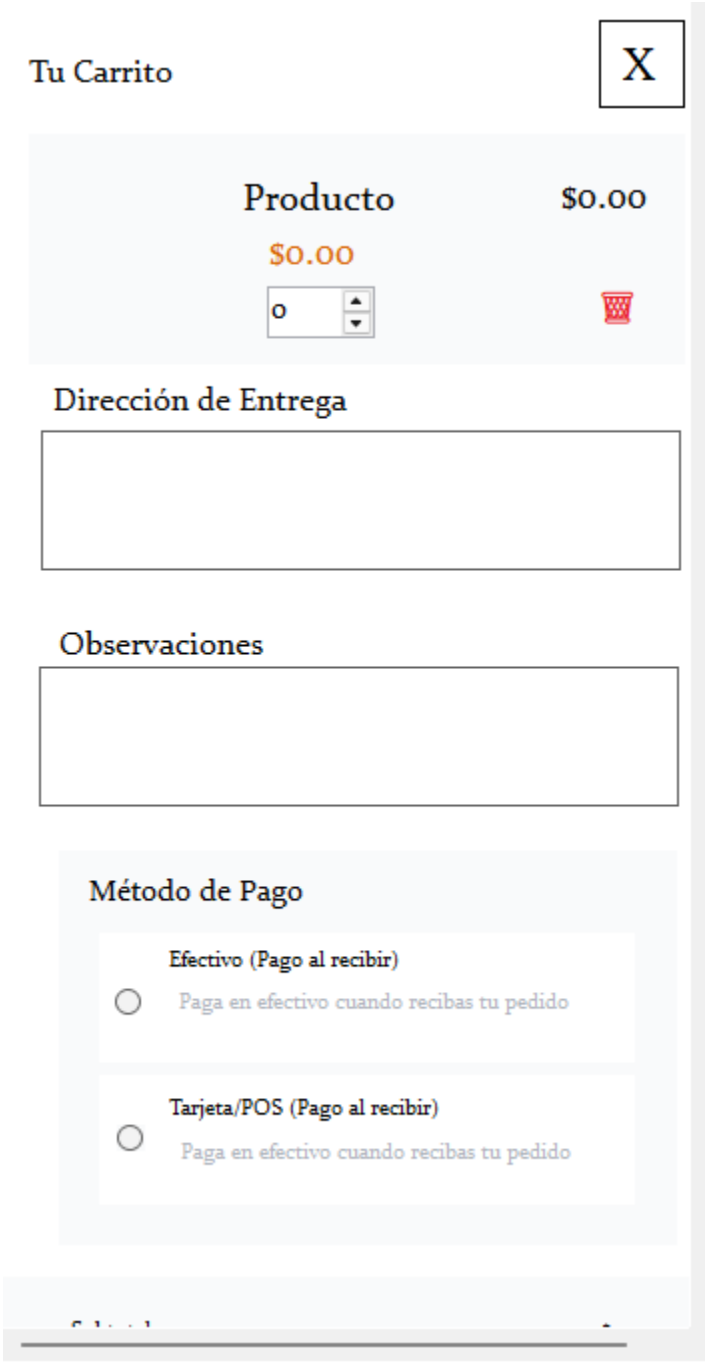

Ilustración 15. Panel de Carrito del cliente

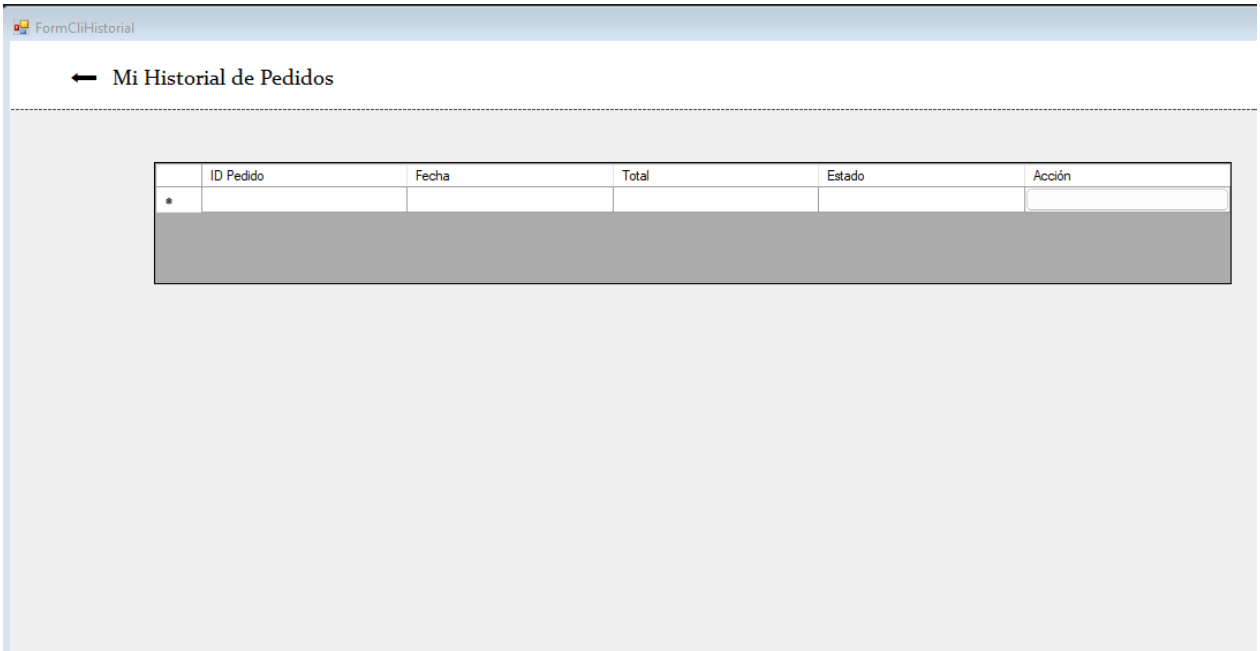

Ilustración 16. Historial de pedido del cliente

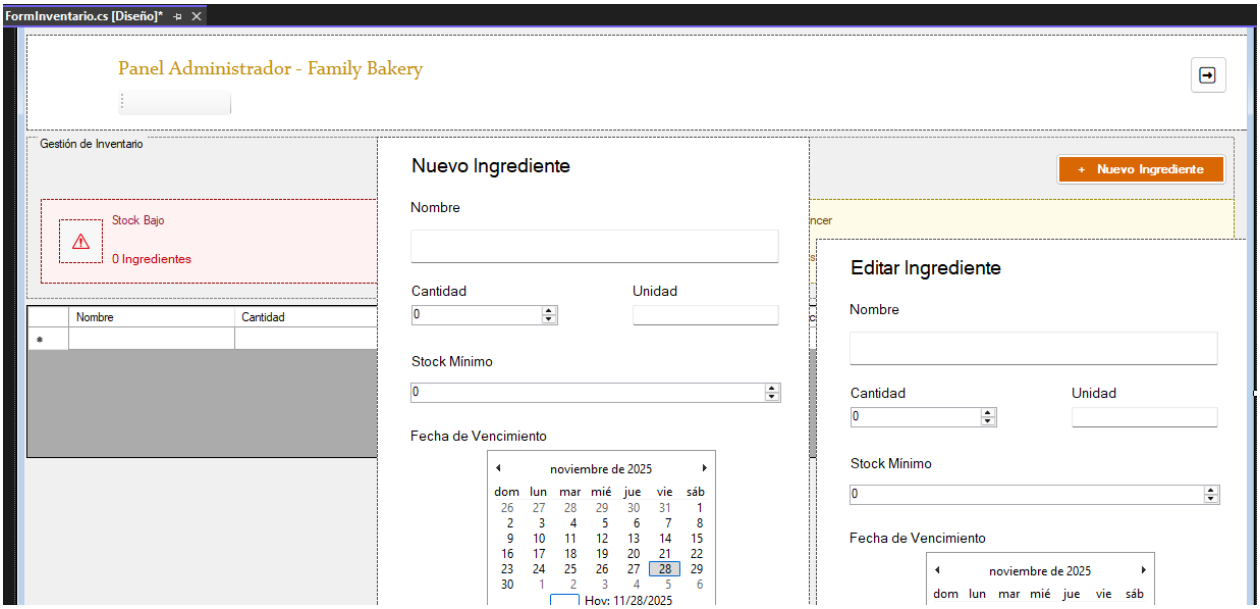

Ilustración 17. Formulario de Inventario

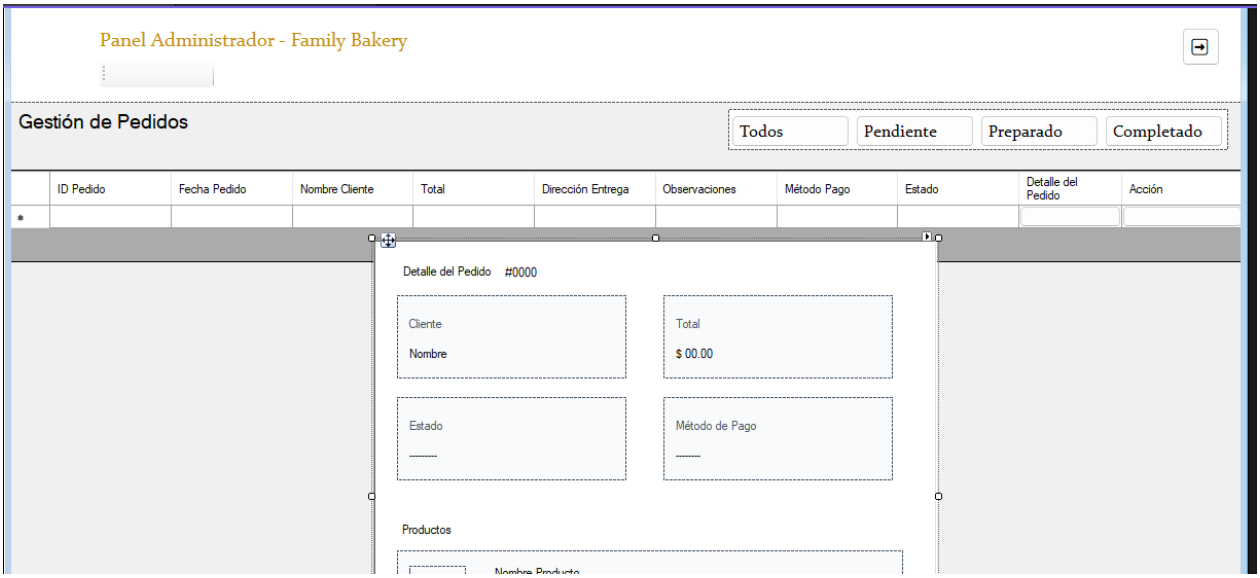

Ilustración 18. Formulario de Pedido

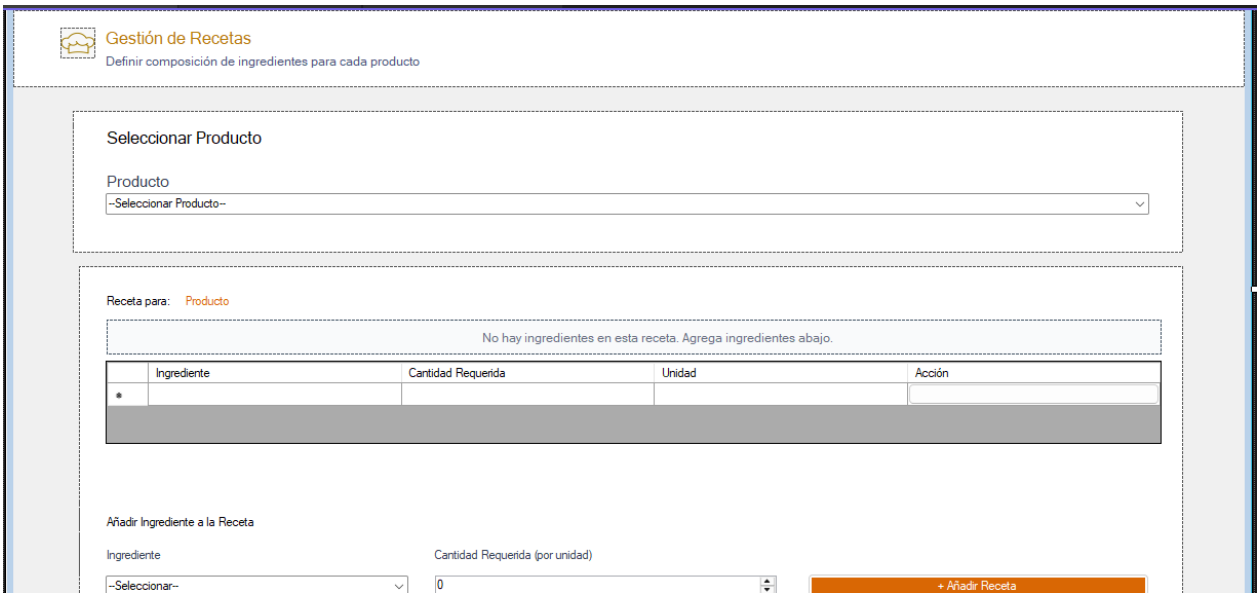

Ilustración 19. Formulario de Recetas

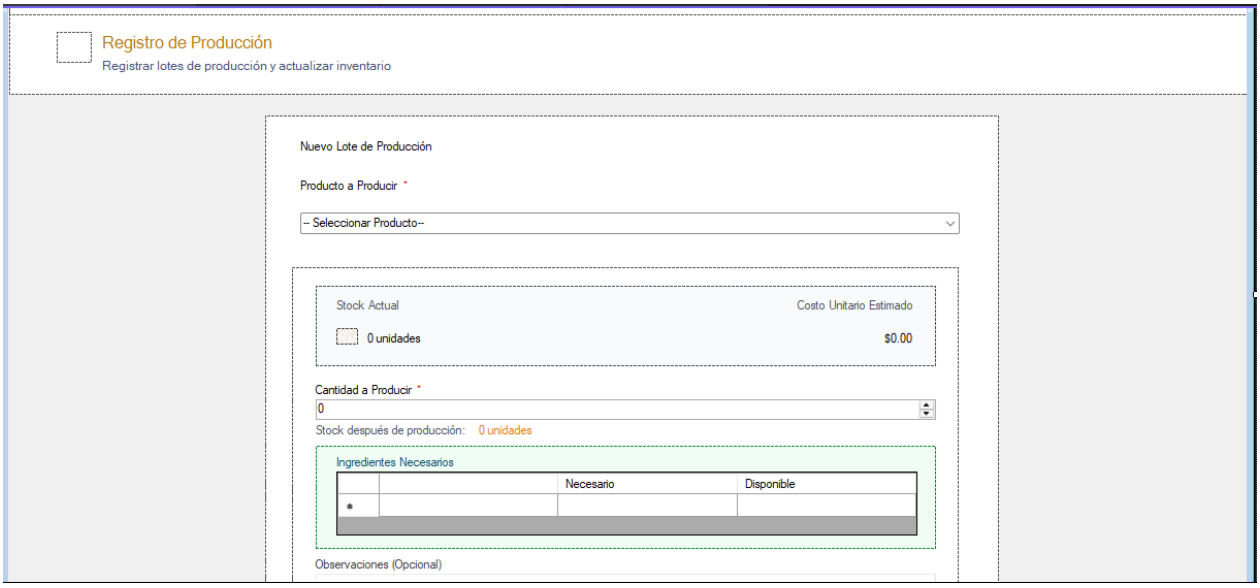

Ilustración 20. Formulario de Producción,

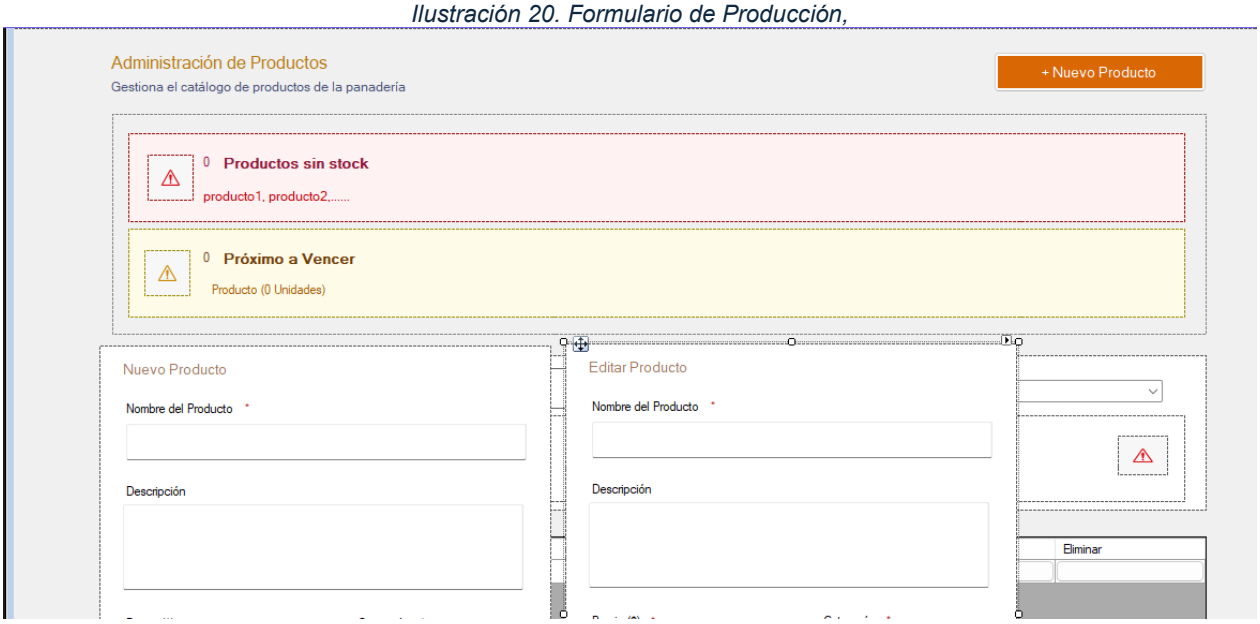

Ilustración 21. Formulario de Productos

4 Glosario

3

3FN
Tercera forma normal de normalización, aplicada al diseño de la base de datos del proyecto para reducir redundancias y evitar anomalías de inserción, actualización o eliminación. 20

A

ACID
Conjunto de propiedades de las transacciones que, como la atomicidad, aseguran que las operaciones críticas mantengan la consistencia de los datos. 8

B

Base de datos Relacional (RDBMS)
Modelo de base de datos basado en tablas relacionadas mediante claves primarias y foráneas, utilizado para organizar la información de productos, recetas, pedidos, inventario y usuarios del sistema. 20
Business Entities
Clases que representan los datos del negocio (usuario, producto, pedido, etc.). 8

C

C#
Lenguaje de programación empleado en la materia y en el proyecto para desarrollar la aplicación que implementa el Sistema de Gestión de Pedidos, Inventario y Producción. 6
CREATE DATABASE
Instrucción SQL que crea una base de datos en el servidor. 22
CREATE TABLE
Instrucción SQL que crea tablas con sus columnas y tipos de datos.....23
CRUD
Conjunto de operaciones básicas sobre datos Crear, Leer, Actualizar y Eliminar registros en las tablas del sistema.....10

D

DATETIME
Es un tipo de dato que almacena fecha y hora juntos, típicamente en el formato YYYY-MM-DD. 23

F

Filtrado dinámico
Funcionalidad que actualiza en tiempo real la lista de productos mostrados según la categoría o criterio elegido por el usuario. 16

G

Grid
Componente de interfaz en forma de tabla que muestra filas y columnas de datos, como el stock actual de ingredientes. 11

L

Login
Formulario de inicio de sesión basado en usuario y contraseña que autentica a los usuarios antes de permitirles acceder a las funcionalidades del sistema. 11

M

Módulo
Parte funcional del sistema que agrupa pantallas y procesos relacionados, como Administración, Inventario, Producción o Pedidos. 16

N

N Capas (N-Tier)
Estilo de arquitectura utilizado en el proyecto, donde la aplicación se divide en capas lógicas separadas (presentación, negocio y datos) para facilitar el mantenimiento y la escalabilidad..... 6

P

R

S

T

5 Referencias bibliográficas

DataCamp. (2025, febrero 14). ¿Qué es la Tercera Forma Normal (3NF)? Guía para principiantes.

https://www.datacamp.com/es/tutorial/third-normal-form

freeCodeCamp. (2024, febrero 9). Operaciones CRUD – ¿Qué es CRUD?. freeCodeCamp en Español.

https://www.freecodecamp.org/espanol/news/operaciones-crud-que-es-crud/

Maldonado, R. (2025, abril 11). ¿Qué es ACID en bases de datos?: Guía completa. KeepCoding.

https://keepcoding.io/blog/que-es-acid-bases-datos/

Microsoft. (s.f.). Consulta y modificación de datos con Transact-SQL. Microsoft Learn.

https://learn.microsoft.com/es-es/training/paths/get-started-querying-with-transact-sql/

Microsoft. (s.f.). Creación de un procedimiento almacenado - SQL Server. Microsoft Learn.

https://learn.microsoft.com/es-es/sql/relational-databases/stored-procedures/create-a-stored-procedure?view=sql-server-ver17

Microsoft. (s.f.). Documentación técnica de SQL Server. Microsoft Learn.

https://learn.microsoft.com/es-es/sql/sql-server/?view=sql-server-ver17

Microsoft. (s.f.). Guía de C#: lenguaje administrado de .NET. Microsoft Learn.

https://learn.microsoft.com/es-es/dotnet/csharp/

Oracle América Latina. (2021, junio 18). Qué es una base de datos relacional. Oracle.

https://www.oracle.com/latam/database/what-is-a-relational-database/

Sqlserverdb. (s.f.). Manuales SQL. Sqlserverdb.

https://sqlserverdb.com/manuales-sql/