Sistema de ventas para la empresa Encanto de azahar

Materia: Desarrollo Back-end

Docente: Noé Gutiérrez Osorno

Integrantes:
Francisco Javier Robles Bautizta
Carmen Maribel Jose Martinez

7° Semestre

12/12/2025

ENCANTO DE AZAHAR

"EXPLORA EL MUNDO A TRAVÉS DE NUESTROS AROMAS"

INDICE

INTRODUCCIÓN

El proyecto consiste en desarrollar un sistema de ventas en C# para la empresa Encanto de Azahar, con el objetivo de automatizar y optimizar sus procesos comerciales. Mediante la aplicación de patrones de diseño, se busca crear un sistema eficiente para la gestión de ventas, clientes y productos, mejorando la organización y el control de las operaciones de la empresa.

NOMBRE DEL PROYECTO

Sistema de ventas para la empresa Encanto de azahar

OBJETIVO GENERAL

Desarrollar un sistema de ventas en C# para la empresa Encanto de Azahar, que gestione usuarios, productos y pedidos, aplicando patrones de diseño para garantizar eficiencia, seguridad y funcionalidad en la venta personalizada de café.

OBJETIVOS ESPECÍFICOS

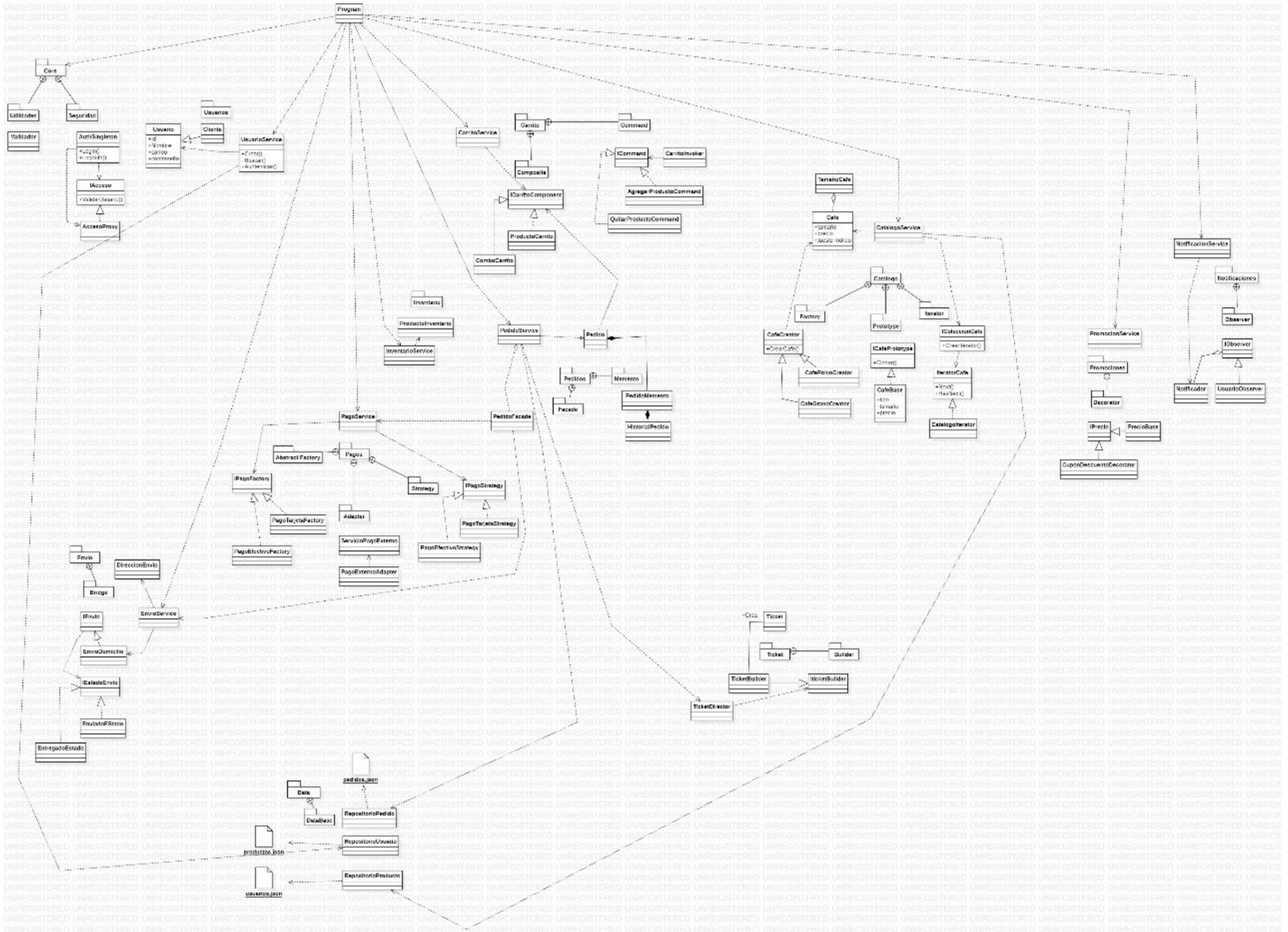
1. Implementar al menos 16 patrones de diseño (creacionales, estructurales y conductuales) en las diferentes secciones del sistema de ventas.
2. Diseñar diagramas de clases y de estructura que reflejen la arquitectura del sistema y la interacción entre sus componentes.
3. Automatizar procesos clave del negocio, como el registro de ventas y la gestión de inventario.
4. Mantener un historial de pedidos por usuario, permitiendo consultar sus compras anteriores.
5. Garantizar la persistencia de datos mediante archivos JSON para usuarios, productos y pedidos.

JUSTIFICACIÓN

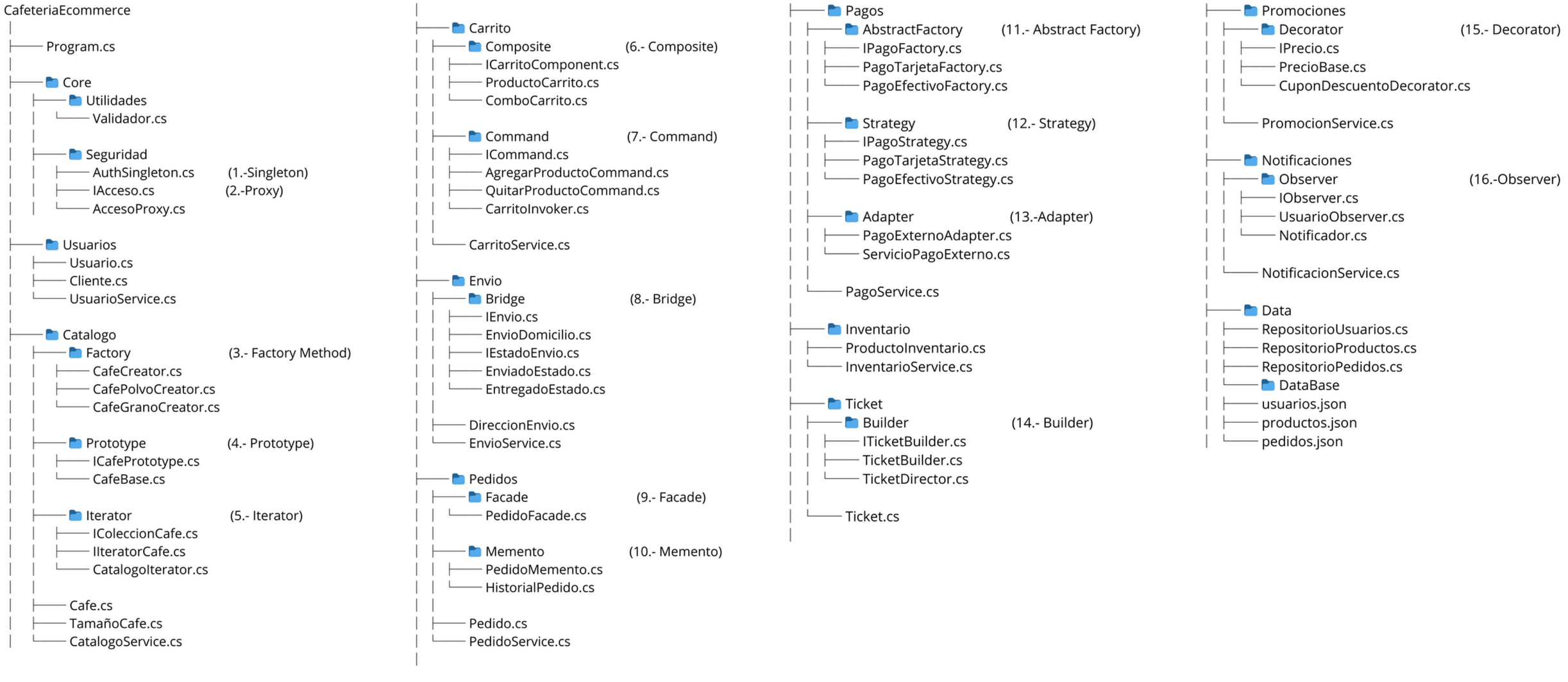
La implementación de un sistema de ventas para la empresa Encanto de Azahar es importante para optimizar sus procesos comerciales. La aplicación de patrones de diseño permite desarrollar un software más organizado, modular y fácil de mantener, facilitando la integración de nuevas funcionalidades sin afectar la estructura existente.

Este proyecto proporciona una herramienta confiable que centraliza la información de ventas, clientes y productos, mejorando el control y la organización de las operaciones comerciales. Además, permite poner en práctica los conocimientos adquiridos sobre patrones de diseño, reforzando la comprensión de su aplicación en el desarrollo de sistemas.

DIAGRAMA DE CLASES



ESTRUCTURA DEL PROYECTO



METODOLOGÍA DE DESARROLLO

Rol	Persona asignada	Responsabilidad principal
Product Owner (PO)	Carmen Maribel	Gestiona el Product Backlog y define las prioridades del proyecto.
Scrum Master (SM)	Francisco Robles	Asegura el cumplimiento del flujo Scrum, organiza reuniones, elimina bloqueos y coordina las entregas.
Development Team	Carmen Maribel Francisco Robles	Realiza la codificación, pruebas, documentación e integración del sistema.

CALENDARIO DE ENTREGAS

Sprint	Fechas	Entregables / Actividades	Responsable
Sprint 1	8 nov – 20 nov	Módulo de gestión de usuarios y productos: diseño y codificación de la estructura de usuarios y productos, creación de clases y métodos iniciales.	Carmen Maribel, Francisco Robles
Sprint 2	22 nov – 6 dic	Modulo de registro de ventas y pedidos: implementación de la funcionalidad de ventas, control de stock y generación de pedidos.	Carmen Maribel, Francisco Robles
Sprint 3	7 dic – 10 dic	Modulo de Historial de pedidos y persistencia de datos: registro de transacciones, almacenamiento en JSON y consultas de historial.	Carmen Maribel, Francisco Robles
Sprint 4	11 dic 2025	Pruebas finales y documentación: pruebas de funcionamiento, corrección de errores y elaboración de la documentación final del proyecto.	Carmen Maribel, Francisco Robles

IMPLEMENTACIÓN DE PATRONES

1. Patrón Singleton:

Módulo: Core/Seguridad.cs

Uso: Garantiza que solo exista una única instancia del manejador de autenticación (AuthSingleton), asegurando que todos los servicios accedan al mismo estado de sesión o permisos.

(Shvets, 2021, Pág. 148).

2. Patrón Factory Method:

Módulo: Catálogo/Factory

Clases: CafeCreator.cs, CafePolvoCreator.cs

Uso: Permite crear distintos tipos de productos (CafePolvo, CafeGrano) delegando la responsabilidad a subclases (CafePolvoCreator, CafeGranoCreator), sin que el código principal (CatalogoService) sepa qué tipo exacto de producto se está creando.

(Shvets, 2021, Pág. 78).

3. Patrón Prototype:
Módulo: Catálogo/Prototype
Clases: ICafePrototype.cs, CafeBase.cs
Uso: Permite clonar objetos de café preconfigurados rápidamente (p. ej., un "Café Estándar 250g" completo), en lugar de reconstruirlos, mejorando la eficiencia al agregar múltiples unidades al carrito. (Shvets, 2021, Pág. 132).

4. Patrón Abstract Factory:
Módulo: Pagos/AbstractFactory
Clases: IPagoFactory.cs, PagoTarjetaFactory.cs
Uso: Permitir que el sistema de Pagos cree las estrategias correctas para procesar una transacción (Tarjeta, Efectivo, etc.) sin acoplarse directamente a la clase concreta de la estrategia
(Shvets, 2021, Pág. 95).

5. Patrón Builder:
Módulo: Ticket/Builder
Clases: TicketDirector.cs, ITicketBuilder.cs
Uso: Separa la construcción del objeto (el ticket de compra). Permite construir el ticket paso a paso (añadir encabezado, lista de productos, descuento, total) de forma controlada y legible.
(Shvets, 2021, Pág. 111).

6. Patrón Proxy:
Módulo: Core/Seguridad
Clases: IAcceso.cs, AccesoProxy.cs
Uso: Proporciona un sustituto o intermediario (AccesoProxy) para controlar el acceso al objeto real (IAcceso). Se usa para aplicar la lógica de seguridad y permisos.
(Shvets, 2021, Pág. 252).

7. **Patrón Composite:**
Módulo: Carrito/Composite
Clases: ICarritoComponent.cs, ComboCarrito.cs
Uso: Permite tratar tanto a productos individuales (ProductoCarrito) como a grupos de productos (ComboCarrito) de forma uniforme dentro del carrito. Esto simplifica el cálculo del total, ya que la lógica es la misma para ambos.
(Shvets, 2021, Pág. 192).

8. **Patrón Adapter:**
Módulo: Pagos/Adapter
Clases: TicketDirector.cs, ITicketBuilder.cs
Uso: Convierte la interfaz de clase ServicioPagoExterno en otra interfaz que el cliente espera PagoExternoAdapter. Esto permite integrar servicios de pago externos sin modificar el núcleo del sistema de pagos.
(Shvets, 2021, Pág. 161).

9. Patrón Bridge:
Módulo: Envio/Bridge
Clases: IEnvio.cs, EnvioDomicilio.cs
Uso: Para añadir nuevos tipos de seguimiento (CanceladoEstado, EnRepartoEstado) sin modificar la clase principal EnvioDomicilio, haciendo el sistema de seguimiento altamente flexible
(Shvets, 2021, Pág. 175).

10. Patrón Decorator:
Módulo: Promociones/Decorator
Clases: IPrecio.cs, CuponDescuentoDecorator.cs
Uso: Añade funcionalidades (descuentos, promociones) de manera dinámica y flexible a un objeto base (PrecioBase). Permite encadenar múltiples cupones o descuentos sin modificar el código del precio original.
(Shvets, 2021, Pág. 206).

11. Patrón Facade:

Módulo: Pedidos/Facade

Clases: PedidoFacade.cs

Uso: Proporciona una interfaz simplificada (PedidoFacade) para interactuar con el subsistema de pedidos (PedidoService). Permite confirmar pedidos, actualizar estados u obtener historial sin conocer la complejidad interna del servicio, centralizando y simplificando las operaciones relacionadas con pedidos.

(Shvets, 2021, Pág. 226).

12. Patrón Command:

Módulo: Carrito/Command

Clases: ICommand.cs, CarritoInvoker.cs

Uso: Permite que acciones como “agregar un producto al carrito” se manejen de manera independiente y ordenada. CarritoInvoker se encarga de ejecutar la acción sin preocuparse de los detalles internos, haciendo más fácil añadir, cambiar o repetir acciones en el carrito.

(Shvets, 2021, Pág. 289).

13. **Patrón Memento:**

Módulo: Pedidos/Memento

Clases: PedidoMemento.cs, HistorialPedido.cs

Uso: Permite guardar el estado de un pedido en distintos momentos (PedidoMemento). Gracias a esto, se puede volver atrás y recuperar el último estado o revisar el historial de cambios sin alterar la información original del pedido.

(Shvets, 2021, Pág. 340).

14. **Patrón Observer:**

Módulo: Notificaciones/Observer

Clases: Notificador.cs, UsuarioObserver.cs

Uso: Permite que los usuarios reciban notificaciones automáticamente cuando ocurre algo nuevo, como una oferta o el estado de un pedido. Notificador avisa a todos los usuarios suscritos (UsuarioObserver) sin que ellos tengan que estar revisando constantemente.

(Shvets, 2021, Pág. 358).

15. PatrónIterator:
Módulo: Catálogo/Iterator
Clases: IIteratorCafe.cs, CatalogoIterator.cs
Uso: Permite recorrer la lista de cafés del catálogo uno por uno (CatalogoIterator) sin exponer cómo está organizada la colección. Así, se puede ver o procesar cada café de manera ordenada sin complicarse con la estructura interna de la lista.
(Shvets, 2021, Pág. 310).

16. Patrón Strategy:
Módulo: Pagos/Strategy
Clases: IPagoStrategy.cs
Uso: Define una familia de algoritmos (PagoTarjetaStrategy, PagoEfectivoStrategy), encapsula cada uno y los hace intercambiables. Permite al cliente seleccionar dinámicamente el método de pago sin modificar la clase de Pago principal.
(Shvets, 2021, Pág. 391).

CONCLUSIÓN

El sistema de ventas en C# para consola permitió demostrar la utilidad de los patrones de diseño y la facilidad de integrar diferentes módulos, como la gestión de archivos, el catálogo de productos y el procesamiento de pagos, haciendo el desarrollo más organizado y eficiente.

BIBLIOGRAFIA

Alexander Shvets (v2021-1-7). Sumérgete en los PATRONES DE DISEÑO. Jorge F. Ramírez Ariza.

Blancarte Iturralde, O.J. (2016). Introducción a los patrones de diseño. Un enfoque Práctico (1° ed.). Oscar Blancarte Blog.