UNIVERSIDAD
CATÓLICA
BOLIVIANA
COCHABAMBA

Departamento de Ingeniería y Ciencias Exactas

Carrera de Ingeniería de Sistemas

Arquitectura backend y Diseño del Sistema de Gestión de Pedidos y Pagos (C# / .NET Core)

Estudiante(s):

Matias Diego Moron Vallejo

Cochabamba – Bolivia

08/11/2025

Introducción

Este documento explica el funcionamiento general del **Sistema de Gestión de Pedidos y Pagos** desarrollado en **.NET (C#)**.
La idea de este proyecto es ofrecer una solución completa (y escalable) para manejar el flujo completo de compras en un entorno de e-commerce.
Desde que un usuario crea su cuenta y su carrito, hasta que realiza el pago y se actualiza el inventario automáticamente.

El sistema busca ser modular, limpio y seguro, pero también simple de mantener. Está construido pensando en buenas prácticas de arquitectura y en un desarrollo orientado a servicios RESTful.

Arquitectura del Sistema

Capas:

El proyecto está organizado en tres capas principales, como dicta la escuela "clean architecture" pero con un toque práctico:

- **Capa API** Controladores RESTful que manejan las solicitudes HTTP, devuelven respuestas JSON limpias y consistentes. Esta compuesta por los CONTROLLERS, RESPONSES Y PROGRAM.

- **Capa Core** Aquí vive la lógica de negocio, las entidades del dominio, validaciones y servicios principales. Esta compuesta por CustomEntities (request), Entities, Interface y Services

- **Capa Infrastructure** Encargada de la persistencia, acceso a datos, validaciones técnicas y configuración de dependencias. Esta compuesta por Data → Configuration (Traductor de la base de datos) y AmazonContext (Base de datos) ,Dtos (Clases para proteger la informacion), Mapping (Mapeo de Dto y la entidad), validators (Regular las entradas si son validas) y filters (Aplica las validaciones).

Flujo de Datos

El flujo de ejecución sigue más o menos este camino:

Cliente → API Controller → Service → Repository → Base de Datos

Este flujo asegura que las responsabilidades estén bien separadas, cada capa hace "lo que debe hacer" y nada más sin el core tener una interacción directa con la infraestructura. esto nos permite corregir errores mas rapido y eficiente.

Componentes Principales

1. Gestión de Usuarios (UserController)

Responsable de manejar toda la parte de usuarios: creación, consultas de billetera, gestion de dinero (Billetera), etc.

Endpoints clave:

- POST /api/users → Crea un nuevo usuario (automáticamente genera su carrito).
- GET /api/users/{id}/billetera → Consulta saldo disponible.
- POST /api/users/{id}/billetera/llenar/{amount} → Recarga billetera del usuario.

Características:

- Se genera un carrito al crear usuario y solo tiene uno. Al momento de pagar se genera uno completamente vacio.
- Validación de email único en toda la BD.
- El usuario tiene una "billetera virtual" para simular pagos en linea.

2. Catálogo de Productos (ProductsController)

Gestión completa del inventario y catálogo de productos.

Endpoints principales:

- GET /api/products → Lista todos los productos.
- POST /api/products → Crea un nuevo producto.
- GET /api/products/category/{category} → Filtra por categoría.
- PATCH /api/products/{id}/stock/{quantity} → Actualiza stock manualmente.

3. Sistema de Pedidos (OrdersController)

Esta parte es el corazón del sistema gestiona el flujo completo de compra.

Estados posibles del pedido:

- Cart → Carrito activo del usuario.
- Pending → Pedido confirmado, esperando pago.
- Paid → Pago procesado y orden cerrada.

Flujo general:

1. Agregar producto al carrito →
   POST /api/orders/{userId}/products/{productId}/quantity/{quantity}

2. Ver carrito →
   GET /api/orders/user/{userId}/cart

3. Procesar pago →
   POST /api/orders/user/{userId}/process-payment

Cada endpoint tiene su propia capa de validaciones, tanto a nivel DTO como lógica de negocio.

4. Sistema de Pagos (PaymentsController)

Encargado de registrar, consultar y validar todas las transacciones.

Estados del pago:

- Pending → Pago iniciado.
- Completed → Pago realizado con éxito.
- Failed → Pago rechazado (por falta de fondos o error interno).

5. Casos de Uso Descritos

5.1 CU01: Gestión de Usuarios

Descripción: Registrar y administrar usuarios del sistema e-commerce
Actor: Administrador, Cliente
Flujo Principal:

1. Cliente se registra en el sistema
2. Sistema valida datos únicos (email)
3. Sistema almacena información del usuario
4. Cliente puede actualizar su perfil

5.2 CU02: Catálogo de Productos

Descripción: Gestionar el inventario y visualización de productos
Actor: Administrador, Cliente
Flujo Principal:

1. Administrador agrega nuevos productos
2. Sistema valida stock y precios
3. Cliente consulta productos disponibles
4. Sistema muestra detalles del producto

5.3 CU03: Proceso de Compra

Descripción: Realizar órdenes de compra de productos
Actor: Cliente
Flujo Principal:

1. Cliente selecciona productos al carrito
2. Sistema calcula total y valida stock
3. Cliente confirma orden
4. Sistema genera orden con estado "Pending"

5.4 CU04: Procesamiento de Pagos

Descripción: Gestionar transacciones financieras de las órdenes
Actor: Cliente, Sistema de Pagos
Flujo Principal:

1. Cliente selecciona método de pago
2. Sistema procesa transacción
3. Sistema valida fondos/autorización
4. Sistema actualiza estado de orden a "Paid"

5.5 CU05: Dashboard de Reportes

Descripción: Generar reportes analíticos del negocio
Actor: Administrador
Flujo Principal:

1. Administrador solicita reportes
2. Sistema consulta datos consolidados
3. Sistema genera métricas de ventas
4. Sistema muestra reportes en dashboard

6. Criterios de Aceptación por Caso de Uso

6.1 CU01 - Gestión de Usuarios

- Usuario puede registrarse con email único
- Sistema valida formato de email válido
- Usuario puede actualizar información personal
- Sistema previene duplicación de emails
- Credenciales se almacenan de forma segura

6.2 CU02 - Catálogo de Productos

- Administrador puede crear/editar productos
- Sistema valida precios positivos
- Stock no puede ser negativo
- Cliente puede filtrar/buscar productos
- Productos inactivos no son visibles

6.3 CU03 - Proceso de Compra

- Orden debe contener al menos un producto
- Sistema valida stock antes de crear orden
- Total se calcula automáticamente
- Orden genera número único de referencia

6.4CU04 - Procesamiento de Pagos

- Pago debe asociarse a una orden existente
- Monto del pago debe coincidir con total de orden
- Sistema registra método de pago utilizado
- Orden cambia a estado "Paid" tras pago exitoso
- Transacciones fallidas se registran con motivo

6.5 CU05 - Dashboard de Reportes

- Reporte de productos más vendidos
- Métricas de ventas mensuales
- Identificación de productos con bajo stock
- Top clientes por volumen de compras
- Estadísticas generales del board

7. Caso de Uso Trabajado en Esta Etapa de Evaluación

CASO DE USO PRINCIPAL: CU03 - Proceso de Compra
CASOS DE USO SECUNDARIOS: CU04 - Procesamiento de Pagos y CU05 - Dashboard de Reportes

Detalle del Trabajo Realizado:

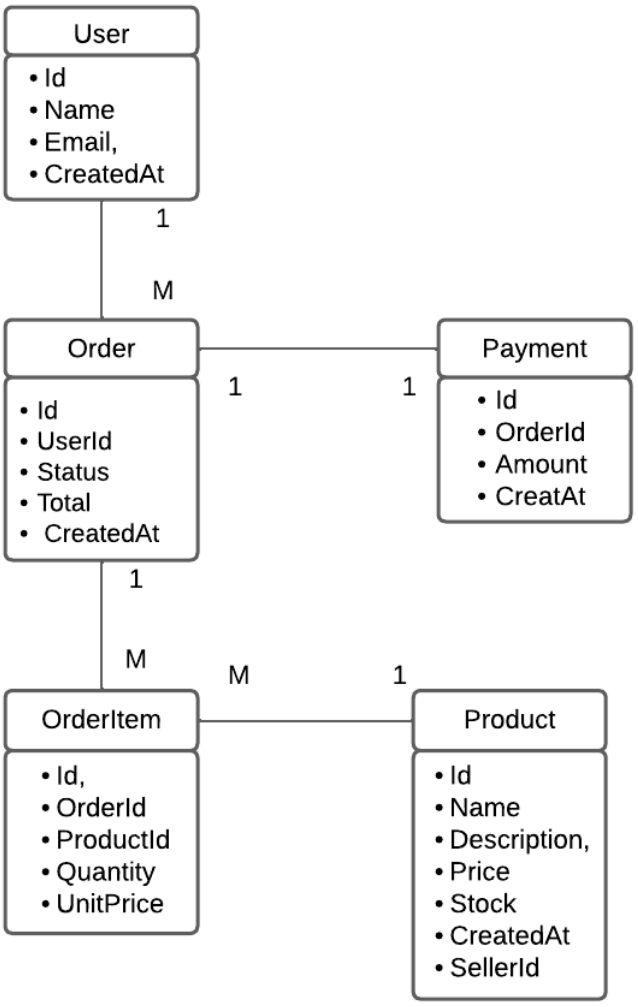
En esta etapa de evaluación, nos hemos enfocado principalmente en el **CU03 - Proceso de Compra**, implementando:

1. **Endpoints completos para gestión de órdenes** (OrdersController)
2. **Sistema de procesamiento de pagos** (PaymentsController)
3. **Servicios de negocio para órdenes y pagos** (OrderService, PaymentService)
4. **Repositorios de datos** (OrderRepository, PaymentRepository)
5. **Sistema de reportes y dashboard** con métricas de negocio

Funcionalidades Implementadas:

- Creación de órdenes con validación de stock
- Cálculo automático de totales
- Procesamiento de transacciones de pago
- Consulta de historial de órdenes por usuario
- Generación de reportes analíticos:
    - Productos más vendidos
    - Ventas mensuales
    - Productos con bajo stock
    - Top clientes por gasto
    - Estadísticas generales

8. Diagrama Entidad Relacion



User → Order → 1:N
Order → Order_Item → 1:N
Product → Order_Item → 1:N
Order → Payment → 1:1

CONCLUSION

Este proyecto representa la implementación exitosa de un sistema de e-commerce completo desarrollado en .NET con arquitectura limpia. Se ha construido una plataforma robusta que gestiona el ciclo completo de compras: desde la selección de productos en el carrito hasta el procesamiento seguro de pagos y la actualización en tiempo real del inventario. La arquitectura separa claramente las responsabilidades entre el core de negocio, la infraestructura y la API, utilizando patrones como Repository y Service Layer para mantener el código mantenible y escalable. Se implementaron validaciones sólidas con FluentValidation, transacciones atómicas para garantizar la consistencia de datos en operaciones críticas, y endpoints RESTful bien estructurados que proporcionan respuestas limpias y seguras. El sistema maneja eficientemente la gestión de usuarios con billetera virtual, control de stock, procesamiento de órdenes y un flujo transaccional completo que asegura la integridad en cada compra, estableciendo una base sólida para un sistema de e-commerce confiable y de alto rendimiento.