INSTRUMENTO DE EVALUACIÓN – RETO 1: GESTION DE PEDIDOS

Duración: 8 horas

Stack tecnológico: .NET + Angular + SQL Server

Arquitectura sugerida: Clean Architecture

1. Descripción del reto

En una compañía comercial mediana, el proceso de **gestión de pedidos** se realiza de forma manual, usando hojas de cálculo y documentos físicos. Los vendedores registran pedidos en formatos distintos, lo que genera inconsistencias entre áreas: se confunden cantidades, se repiten pedidos, y el inventario no refleja el estado real de la bodega. Esto produce pérdidas por sobreventa, retrasos en entregas y errores en facturación.

La gerencia busca iniciar una **modernización progresiva de su ERP**, comenzando por el módulo de **Pedidos**, el núcleo del flujo O2C (*Order to Cash*). Este módulo debe estandarizar la captura, control y seguimiento de pedidos, además de mantener coherencia con el inventario y preparar la base para futuras etapas de facturación y reportes.

El reto técnico consiste en diseñar e implementar, en un entorno mixto de tecnologías heredadas (**Visual FoxPro**) y modernas (**.NET + SQL Server**), un sistema funcional que permita crear, confirmar, anular y cerrar pedidos, controlando automáticamente el stock disponible y registrando cada transición de estado. El sistema debe ser robusto ante errores de validación, simple de operar por vendedores y extensible hacia los próximos módulos del ERP.

Objetivos funcionales:

- Digitalizar el proceso de registro de pedidos con flujo completo de estados: *Nuevo → Confirmado → Cerrado/Anulado.*
- Aplicar control automático de inventario: reserva y liberación de stock.
- Registrar auditoría (usuario, fecha, acción).
- Exponer API REST estándar para futuras integraciones.
- Implementar cliente en Visual FoxPro para interacción del usuario final.
- Usar herramientas de IA (Copilot o ChatGPT) para acelerar análisis y desarrollo, documentando prompts y resultados.

INSTRUMENTO DE EVALUACIÓN – RETO 1: GESTION DE PEDIDOS

Desafío de integración:
El equipo deberá garantizar la interoperabilidad entre **Visual FoxPro** (cliente legado) y el **API .NET**, definiendo un contrato claro (JSON) para los flujos de pedidos, autenticación y errores. Se valorará la capacidad de resolver problemas de compatibilidad y mantener la coherencia entre capa de presentación y servidor.

2. Historias de usuario

ID	Historia	Criterios de aceptación
US-AUTH-1	Como **usuario del sistema**, quiero **iniciar sesión con usuario y contraseña** para acceder al módulo de pedidos.	Autenticación JWT válida; el token se usa en cada solicitud; error 401 con mensaje claro si las credenciales no son válidas o el usuario está inactivo.
US-CUST-1	Como **vendedor**, quiero **seleccionar un cliente activo** desde la lista para asociar correctamente el pedido.	La API solo retorna clientes con IsActive = true; no se permite crear pedido con cliente inactivo; el pedido almacena el CustomerId válido.
US-ORD-1	Como **vendedor**, quiero **crear un nuevo pedido con sus líneas de productos** para registrar una venta pendiente.	Estado inicial *NUEVO*; mínimo una línea con Qty > 0 y UnitPrice ≥ 0; el servidor calcula LineTotal y totales generales; la edición solo es posible mientras el pedido está en *NUEVO*.
US-ORD-2	Como **vendedor**, quiero **confirmar un pedido** para **reservar el stock** y garantizar la disponibilidad.	Solo pedidos *NUEVO* pueden confirmarse; se valida OnHand − Reserved ≥ Qty por producto; si hay disponibilidad se actualiza Reserved y el estado pasa a *CONFIRMADO*; 409 si no hay stock suficiente.
US-ORD-3	Como **vendedor**, quiero **anular un pedido confirmado**	Solo pedidos *CONFIRMADO* pueden anularse; se revierten las reservas; el

	para **liberar el stock reservado**.	estado cambia a *ANULADO* y se registra auditoría (usuario, fecha, acción).
US-ORD-5	Como **usuario**, quiero **listar y filtrar los pedidos existentes** para consultar el historial de ventas.	Endpoint GET /orders con filtros por estado y fecha; paginación con metadatos {items,page,pageSize,total} y encabezado X-Total-Count; 401 si no hay token.

3. PROM sugerido (estructural general)

Actúa como un **arquitecto de software full stack** con experiencia en **.NET, Visual FoxPro y SQL Server**.

Diseña la solución técnica completa del **módulo de Pedidos del ERP**, que debe permitir **crear, confirmar, anular y listar pedidos**, asegurando la integridad del inventario, la trazabilidad de estados y la interoperabilidad entre sistemas modernos y legados.

Objetivo técnico general
Implementar un flujo robusto y coherente entre el cliente VFP, el backend .NET y la base de datos SQL Server, garantizando:

- Consistencia transaccional entre creación, confirmación y anulación de pedidos.
- Validación de disponibilidad de stock antes de confirmar.
- Persistencia de auditoría y control de cambios.
- Exposición de endpoints REST seguros y documentados.
- Comunicación fluida mediante JSON entre VFP y la API .NET.

Entregables esperados del diseño

1. **Modelo relacional** con tablas normalizadas: *Customers, Products, Orders, OrderLines, Stocks, Users, AuditLog*.
2. **Arquitectura API** basada en controladores REST, DTOs y repositorios con validaciones en servidor.
3. **Manejo de estados** (Nuevo, Confirmado, Anulado) mediante endpoints dedicados (/confirm, /cancel).

4. **Auditoría y trazabilidad**: tabla AuditLog que registre usuario, acción, entidad, fecha y payload.
5. **Cliente VFP funcional** con pantallas: *Login*, *Pedidos*, *Formulario de Pedido* y *Listado con filtros/paginación*.
6. **Scripts SQL (schema + seeds)** con datos iniciales y restricciones CHECK, UNIQUE y FK.

Requisitos técnicos mínimos

- **Backend (.NET):**
  - API RESTful con autenticación JWT, validaciones con *FluentValidation*, documentación con *Swagger* y manejo de errores con *ProblemDetails*.
  - Transacciones atómicas para confirmación y anulación.
  - Logs estructurados y respuesta estándar {success, data, errors}.
- **Frontend (Visual FoxPro):**
  - Consumo de API vía HTTP.
  - Validaciones locales de campos y manejo de errores del servidor.
  - Grids con filtros por estado y paginación.
- **Base de datos (SQL Server):**
  - Integridad referencial estricta (FK y ON UPDATE RESTRICT).
  - Índices en CustomerId, Status, CreatedAt.
  - Scripts reproducibles (schema.sql, seeds.sql).

Líneas guía del desarrollo

- Mantener *Clean Code* y principios *SOLID*.
- Usar *DTOs* para transferencia de datos entre capas.
- Asegurar idempotencia de los endpoints /confirm y /cancel.
- Validar todos los datos en backend (no solo en cliente).
- Implementar rollback en fallos de stock o estado.
- Documentar en prompts.md los usos de IA (análisis, generación de código, refactorización, validaciones).

4. PROM sugerido para Frontend (Visual FoxPro)

Actúa como desarrollador frontend en **Visual FoxPro**. Construye la interfaz del **módulo de Pedidos** y conéctala a la **API .NET** vía HTTP (JSON).

Objetivo
Entregar un cliente VFP usable que permita: login, crear pedidos con líneas, confirmar, anular y listar con filtros y paginación.

Estructura propuesta

- Carpetas: /ui (formularios), /services (HTTP), /models (DTO), /state (token, usuario), /logs.
- Formularios: frmLogin, frmPedidos, frmPedidoEdit.
- Módulos: HttpClient.prg, AuthService.prg, OrderService.prg, CustomerService.prg, Paging.prg.

Pantallas y flujos

- frmLogin: usuario/contraseña → AuthService.Login() → guardar {token, expiresAt} en /state.
- frmPedidos (grid): filtros estado, fechaFrom, fechaTo, botones Nuevo, Editar, Confirmar, Anular, paginación Anterior/Siguiente.
- frmPedidoEdit: campos Cliente, líneas [Sku, Producto, Qty, UnitPrice, LineTotal], total general. Guardar crea o actualiza si estado=Nuevo.

Integración HTTP

- Base URL configurable. Cabecera Authorization: Bearer <token>.
- GET /customers?active=true para combo de clientes.
- GET /orders?status=&from=&to=&page=&pageSize= con respuesta {items, page, pageSize, total} y encabezado X-Total-Count.
- GET /orders/{id} para detalle.
- POST /orders crear. PUT /orders/{id} editar si Status=Nuevo.
- POST /orders/{id}/confirm y /orders/{id}/cancel.
- Reintento 1 vez en 502/503, con backoff simple.

Validaciones UI

- Campos obligatorios: cliente, ≥1 línea.
- Qty > 0, UnitPrice ≥ 0.

- Bloquear edición si Status <> Nuevo.
- Confirmar operación con diálogos (Are you sure?) en confirmar/anular.

Manejo de errores

- 400: mostrar mensaje del servidor en barra de estado.
- 401: forzar relogin y limpiar token.
- 404: “Pedido no encontrado”.
- 409: “Sin stock” o “Estado inválido”.
- Log en /logs/app.log con fecha, endpoint, payload truncado, código y mensaje.

Modelos (DTO mínimo)

- OrderDto: id, number, customerId, status, total, createdAt, confirmedAt, canceledAt.
- OrderLineDto: productId, sku, name, qty, unitPrice, lineTotal.
- PagedResult<T>: items[], page, pageSize, total.

Paginación

- Controles lblPage, btnPrev, btnNext.
- Estado persiste filtros y página actual.
- Deshabilitar btnPrev en page=1 y btnNext cuando page*pageSize >= total.

UX mínima

- Teclas rápidas: Ctrl+N nuevo, Ctrl+S guardar, Ctrl+F filtrar.
- Spinners durante llamadas HTTP.
- Formatos numéricos y monetarios consistentes.

Pruebas funcionales

- Login ok y falla.
- Crear pedido con línea inválida (esperar 400).
- Confirmar sin stock (409) y con stock (200).
- Anular confirmado (200) y verificar en grid.

- Paginación con 30+ pedidos.

Entrega

- Fuente VFP con BUILD EXE opcional.
- app.config con BaseUrl.
- Script para registrar OCX si aplica.
- README de ejecución y credenciales demo.

Si tienes dudas, dime; si no, continúa.

5. PROM sugerido para Backend (.NET)

Actúa como desarrollador backend senior en **.NET**. Crea una **API REST** para el módulo **Pedidos** que atienda a un cliente **Visual FoxPro** y use **SQL Server**.

Objetivo
Entregar endpoints seguros para crear, confirmar, anular y listar pedidos, con control transaccional de stock y trazabilidad.

Arquitectura y componentes
- Web API .NET 8, Controllers + Services + Repositories.
- EF Core + Migrations. DTOs para IO. AutoMapper.
- FluentValidation. Swagger. CORS habilitado para cliente VFP.
- Logging estructurado. ProblemDetails para errores.

Entidades núcleo
- Customer(Id, Name, TaxId, IsActive)
- Product(Id, Sku, Name, Price)
- Stock(ProductId, OnHand, Reserved)
- Order(Id, Number, CustomerId, Status, CreatedAt, ConfirmedAt, CanceledAt)
- OrderLine(Id, OrderId, ProductId, Qty, UnitPrice, LineTotal)
- User(Id, Username, PasswordHash, IsActive, Role)
- AuditLog(Id, UserId, Entity, EntityId, Action, At, Payload)

Endpoints requeridos
- POST /api/v1/auth/login → {token, expiresAt}. 401 si inválido.
- GET /api/v1/customers?active=true
- GET /api/v1/orders?status=&from=&to=&page=&pageSize= → {items, page, pageSize, total} + X-Total-Count.
- GET /api/v1/orders/{id}
- POST /api/v1/orders (estado inicial Nuevo)
- PUT /api/v1/orders/{id} (solo si Nuevo)
- POST /api/v1/orders/{id}/confirm (reserva stock, pasa a Confirmado)

- POST /api/v1/orders/{id}/cancel (libera reserva, pasa a Anulado)

Reglas de negocio
- Confirmar: por cada línea, verificar OnHand - Reserved ≥ Qty. Si falla, 409.
- Anular: solo si Confirmado. Revertir Reserved exactamente.
- Edición: solo en Nuevo.
- Idempotencia: repetir confirm o cancel no duplica efectos.

Transacciones y concurrencia
- En confirm y cancel: transacción ACID.
- Concurrencia optimista con RowVersion en Order y Stock. Conflicto → 409.

Validaciones
- Pedido: ≥1 línea, Qty > 0, UnitPrice ≥ 0. LineTotal = Qty*UnitPrice calculado en servidor.
- Cliente debe estar IsActive = true.
- Productos y FKs válidos.

Seguridad
- JWT Bearer. Roles opcionales.
- Requerir Authorization: Bearer en todos los endpoints salvo /auth/login.

Respuestas y errores
- Ok: {success:true, data}.
- Error: ProblemDetails con type, title, status, detail, traceId.
- Códigos: 400 validación, 401 auth, 404 no existe, 409 conflicto/estado/stock.

Paginación y filtros
- Parámetros page, pageSize, status, from, to.
- Orden por CreatedAt desc. Devolver X-Total-Count.

Auditoría
- Registrar create/confirm/cancel/update en AuditLog con usuario, timestamps y payload mínimo.

DTOs mínimos
- OrderCreateDto {customerId, lines[]:{productId, qty, unitPrice}}
- OrderDto {id, number, customerId, status, totals, createdAt, confirmedAt, canceledAt, lines[]}
- PagedResult<T> {items, page, pageSize, total}

Consideraciones VFP
- Tolerar encabezados ausentes o minúsculas.

- Mensajes de error simples y cortos.
- Timeouts razonables y respuestas JSON compactas.

Pruebas

- Unit tests: servicios de confirmación/anulación.
- Integration tests: POST /orders, confirm, cancel, paginación y 409 por stock.
- Seed de datos para escenarios.

Entrega

- Proyecto API con appsettings.json y migrations.
- Swagger habilitado en Development.
- Script init-db.ps1 para aplicar migraciones.
- README con variables, conexión y ejemplos curl.

Si tienes dudas, dime; si no, continúa.

6. PROM sugerido para Base de Datos (SQL Server)

Actúa como diseñador de base de datos en **SQL Server**. Crea el modelo relacional del módulo **Pedidos (ERP)** asegurando integridad y claridad entre entidades.

Objetivo
Definir la estructura mínima para soportar los flujos de creación, confirmación y anulación de pedidos, incluyendo control de stock y auditoría.
Estructura de la base de datos
Esquema sec (seguridad y usuarios)

- **Users**
  - Id INT IDENTITY PRIMARY KEY
  - Username NVARCHAR(80) UNIQUE
  - PasswordHash VARBINARY(256)
  - Role NVARCHAR(40)
  - IsActive BIT NOT NULL DEFAULT(1)
  - CreatedAt DATETIME2 NOT NULL DEFAULT(SYSUTCDATETIME())

Esquema erp (lógica del negocio ERP)

- **Customers**
  - Id INT IDENTITY PRIMARY KEY
  - Name NVARCHAR(120)
  - TaxId NVARCHAR(32) UNIQUE
  - IsActive BIT NOT NULL DEFAULT(1)
  - CreatedAt DATETIME2 NOT NULL DEFAULT(SYSUTCDATETIME())
- **Products**
  - Id INT IDENTITY PRIMARY KEY
  - Sku NVARCHAR(40) UNIQUE
  - Name NVARCHAR(120)
  - Price DECIMAL(18,2) NOT NULL CHECK(Price >= 0)
  - CreatedAt DATETIME2 NOT NULL DEFAULT(SYSUTCDATETIME())
- **Stocks**
  - ProductId INT PRIMARY KEY FOREIGN KEY REFERENCES erp.Products(Id)
  - OnHand INT NOT NULL CHECK(OnHand >= 0)
  - Reserved INT NOT NULL CHECK(Reserved >= 0)
- **Orders**
  - Id INT IDENTITY PRIMARY KEY
  - Number NVARCHAR(30) UNIQUE
  - CustomerId INT NOT NULL FOREIGN KEY REFERENCES erp.Customers(Id)
  - Status TINYINT NOT NULL /* 0 = Nuevo, 1 = Confirmado, 2 = Anulado */

    - CreatedAt DATETIME2 NOT NULL DEFAULT(SYSUTCDATETIME())
    - ConfirmedAt DATETIME2 NULL
    - CanceledAt DATETIME2 NULL

- **OrderLines**
    - Id INT IDENTITY PRIMARY KEY
    - OrderId INT NOT NULL FOREIGN KEY REFERENCES erp.Orders(Id)
    - ProductId INT NOT NULL FOREIGN KEY REFERENCES erp.Products(Id)
    - Qty INT NOT NULL CHECK(Qty > 0)
    - UnitPrice DECIMAL(18,2) NOT NULL CHECK(UnitPrice >= 0)
    - LineTotal AS (Qty * UnitPrice) PERSISTED

- **AuditLog**
    - Id BIGINT IDENTITY PRIMARY KEY
    - UserId INT FOREIGN KEY REFERENCES sec.Users(Id)
    - Entity NVARCHAR(40)
    - EntityId INT
    - Action NVARCHAR(30)
    - At DATETIME2 NOT NULL DEFAULT(SYSUTCDATETIME())
    - Payload NVARCHAR(MAX) NULL

Notas generales

- Todas las fechas en UTC.
- Relaciones 1:N:
    - Cliente → Pedidos
    - Pedido → Líneas
    - Producto → Stock y Líneas
- Restricciones CHECK en cantidades y precios.
- Estados de pedido definidos numéricamente (0=Nuevo, 1=Confirmado, 2=Anulado).

Si tienes dudas, dime; si no, continúa.

7. Rúbrica de evaluación

Criterio	Descripción	Nivel Bajo (1-2)	Nivel Alto (4-5)
Solución técnica funcional	Cumple con las historias y opera sin errores críticos.	Falla en ejecución o lógica.	Funciona completa, estable y coherente.
Calidad del código y documentación	Organización, legibilidad, README y comentarios claros.	Código desordenado o sin documentación.	Código limpio, documentado y coherente con estándares.
Capacidad de análisis y decisión	Justifica decisiones técnicas y prioriza correctamente.	Decisiones arbitrarias.	Elecciones óptimas y fundamentadas.
Participación y trazabilidad (Git)	Uso correcto de GitHub (commits descriptivos, estructura ordenada).	Repositorio incompleto.	Historial completo, mensajes claros y trazabilidad total.
Aplicación de herramientas IA	Uso real de Copilot/ChatGPT con prompts documentados.	Sin evidencia de IA.	Prompts efectivos, iterativos y aplicados con criterio.

8. Entregables esperados

El participante debe entregar un único repositorio GitHub con la siguiente estructura:

```
/comunicacion-anuncios
├── /frontend      → Proyecto VFP
├── /backend       → Solución .NET Web API (Clean Architecture)
├── /database      → schema.sql + seeds.sql
├── README.md      → Instrucciones de ejecución
└── prompts.md     → Historial de prompts IA usados
```

Condiciones:

- El repositorio debe crearse con una cuenta corporativa @ofirma.com.
- No se aceptarán repositorios asociados a correos personales.
- El README.md debe incluir pasos para crear la base de datos, ejecutar backend y frontend, dependencias, puertos y credenciales de prueba.