



Enunciado postulantes Fullstack Dev Sr Bistrosoft

Objetivo del proyecto

Medir las capacidades analíticas de los postulantes para el puesto Fullstack dev Sr. dentro del marco de evaluaciones del proceso de selección.

Pautas del trabajo

- El trabajo debe realizarse de forma remota por el postulante, con un tiempo máximo a definir por el reclutador.
- El trabajo consiste en desarrollar una aplicación, con algunos features básicos que nos permitirán medir tus skills técnicos.
- La aplicación debe desarrollarse en una computadora, que no será provista por Bistrosoft, utilizando el IDE que el postulante desee. El framework a utilizar será .NET 7/8. Para la app web, usar [Vue.js](#) 3. Con respecto al IDE se sugiere Visual Studio Code, Cursor o Windsurf.
- Se permite y se recomienda **utilizar IA integrada al IDE para hacerlo en el menor lapso posible**.
- Posterior a la finalización de la aplicación, el postulante deberá ponerse en contacto con el evaluador que le entregó el link del examen, para poder defenderlo en una etapa de coloquio.
- Durante la etapa de coloquio, el postulante queda sujeto a defender el trabajo realizado, con posibilidad de reprenguntas por parte de los evaluadores.
- Al momento de evaluar lo entregado se considerará:
 - Legibilidad del código
 - Buenas prácticas
 - Deseable el uso de Arquitectura Limpia (Clean Architecture) o alguna otra (Hex, cebolla, capas, etc), solo si la maneja (NO EXCLUSIVO)
 - Nivel de conocimiento sobre los lenguajes de programación y frameworks usados.
 - Inyección de dependencias (deseable)
 - Uso de librerías de Logging (deseable)
 - Uso de librerías de Mapping (deseable)



Enunciado del trabajo

Objetivo

Desarrollar una API REST en **.NET 7/8/9** para la gestión de pedidos de una tienda online, y una app en [vue.js](#) que permita generar los pedidos de la API.

Requisitos Técnicos

- Utilizar **.NET 7** o superior.
 - Utilizar [Vue.js](#)³
 - Usar **Entity Framework Core** con una base de datos en memoria (opcionalmente SQL Server).
 - Implementar el patrón **Repository**.
 - Utilizar **MediatR** para la capa de aplicación.
 - Incluir **pruebas unitarias** con **xUnit** y **Moq**.
 - Exponer endpoints en **Swagger** con documentación.
-

Descripción del Problema

La tienda online necesita un sistema para gestionar pedidos con las siguientes entidades:

1. Entidades Principales

- **Cliente (Customer)**:
 - **Id** (Guid)
 - **Name** (string, requerido)
 - **Email** (string, único y requerido)
 - **PhoneNumber** (string)
 - **Orders** (lista de pedidos del cliente)
- **Pedido (Order)**:
 - **Id** (Guid)
 - **CustomerId** (Guid, requerido)
 - **TotalAmount** (decimal, requerido)
 - **CreatedAt** (DateTime, requerido)
 - **Status** (enum: Pending, Paid, Shipped, Delivered, Cancelled)
 - **OrderItems** (lista de productos en el pedido)
- **Producto (Product)**:



- `Id` (Guid)
 - `Name` (string, requerido)
 - `Price` (decimal, requerido)
 - `StockQuantity` (int, requerido)
 - **Item del Pedido (OrderItem):**
 - `Id` (Guid)
 - `OrderId` (Guid, requerido)
 - `ProductId` (Guid, requerido)
 - `Quantity` (int, requerido)
 - `UnitPrice` (decimal, requerido)
-

Requerimientos de la API

1. Crear un Cliente

- Endpoint: `POST /api/customers`
- Crea un cliente nuevo.

2. Obtener un Cliente por ID

- Endpoint: `GET /api/customers/{id}`
- Retorna la información del cliente y sus pedidos.

3. Crear un Pedido

- Endpoint: `POST /api/orders`
- Recibe un `customerId` y una lista de productos (`productId, quantity`).
- Valida stock antes de confirmar el pedido.

4. Actualizar el Estado de un Pedido

- Endpoint: `PUT /api/orders/{id}/status`
- Permite cambiar el estado (`Pending → Paid → Shipped → Delivered`).

5. Listar Pedidos de un Cliente

- Endpoint: `GET /api/customers/{id}/orders`
- Devuelve todos los pedidos de un cliente con sus productos.

6. Pruebas Unitarias

- Agregar pruebas unitarias para la capa de aplicación y repositorio.
-

Extras (Opcionales, pero Valorados)



- Uso de **CQRS** con **MediatR**.
 - Manejo de **excepciones globales**.
 - Implementación de **logging y caching**.
 - Seguridad con **JWT**.
 - Dockerización del proyecto.
-

Criterios de Evaluación

- Código limpio y organizado** (SOLID, buenas prácticas).
- Correcta implementación de patrones** como Repository y CQRS.
- Uso adecuado de asincronía** (`async/await`).
- Pruebas unitarias bien estructuradas**.
- Buena gestión de errores y validaciones**.