Documento Técnico - Arquitectura de MGI-ORDER-API

1. Arquitectura General

La API se desarrolló utilizando NestJS como framework principal, con TypeORM como ORM para la gestión de datos y PostgreSQL como base de datos. La aplicación garantiza la modularidad y escalabilidad del proyecto.

Componentes principales:

- NestJS (Backend Framework): proporciona una arquitectura modular basada en controladores, servicios y módulos.
- TypeORM (ORM): se encarga de la comunicación con la base de datos relacional, manejando entidades, repositorios y transacciones.
- PostgreSQL (Base de Datos Relacional): almacena los datos de usuarios, productos y órdenes con soporte completo para ACID.

Flujo de creación de una orden:

1. El cliente envía un request POST /orders con el usuario y productos.
2. El OrderService inicia una transacción con TypeORM.
3. Se valida la existencia del usuario.
4. Se valida y reserva stock de los productos.
5. Se crea la orden y sus relaciones.
6. En caso de error, la transacción realiza un rollback automático.

2. Ventajas de crear interfaces para definir repositorios

Se implementó un patrón de interfaces para repositorios, en lugar de usar directamente los repositorios concretos de TypeORM. Esto aporta varias ventajas:

1. Desacoplamiento: la capa de aplicación no depende directamente de TypeORM.
2. Testabilidad: es posible inyectar mocks o stubs de los repositorios en pruebas unitarias.
3. Claridad en el contrato: las interfaces definen claramente qué operaciones están disponibles.
4. Mantenibilidad: cambios en persistencia no afectan la capa de negocio.

3. Aplicación de los principios SOLID

1. SRP: Cada servicio tiene una responsabilidad única.
2. OCP: La lógica de negocio está abierta a extensión pero cerrada a modificación.
3. LSP: Las interfaces aseguran que cualquier implementación pueda sustituirse.

4. ISP: Las interfaces de repositorio definen solo los métodos necesarios.
5. DIP: Los servicios dependen de abstracciones, no implementaciones concretas.

4. Beneficios Generales

- Escalabilidad: arquitectura modular.
- Mantenibilidad: separación de responsabilidades.
- Flexibilidad: facilidad para cambiar de motor de base de datos.
- Calidad de código: principios SOLID aseguran diseño limpio.

5. Conclusión

El proyecto está diseñado siguiendo buenas prácticas de arquitectura de software, aplicando principios SOLID y patrones de diseño que garantizan un código desacoplado, mantenible y escalable. El uso de interfaces para repositorios y la adopción de NestJS permiten un desarrollo flexible y preparado para entornos modernos.