Estructura Backend para una Aplicación de E-Commerce

1. Introducción

Este documento describe la arquitectura de software para una aplicación de e-commerce basada en microservicios. El objetivo es proporcionar una guía detallada sobre la estructura del proyecto, tecnologías utilizadas, patrones de diseño implementados y la estructura de archivos. La arquitectura está diseñada para ser escalable, mantenible y eficiente.

2. Arquitectura de Microservicios

La arquitectura de la aplicación se basa en microservicios, donde cada servicio es independiente. La gestión del enrutamiento se haría a través de API Gateway y existiría un load balancer antes de cada microservicio para distribuir la carga y permitir la alta disponibilidad de los mismos. Los microservicios sugeridos son:

- Productos: Manejo de productos disponibles para la venta.
- Login: Autenticación de usuarios.
- Auth: Gestión de autorizaciones y roles de usuarios.
- Pedidos: Gestión de pedidos realizados por los clientes.
- Clientes: Información y gestión de clientes.
- Payments: Procesamiento de pagos.
- Inventario: Gestión del inventario de productos.
- Notificaciones: Envío de notificaciones a los clientes.

3. Tecnologías Utilizadas

- **Backend Framework:** Node.js ofrece una arquitectura basada en eventos y no bloqueante que es ideal para aplicaciones concurrentes. NestJS agrega una estructura modular y características avanzadas como la inyección de dependencias y soporte para TypeScript, lo que mejora la mantenibilidad y escalabilidad del código. TypeScript aporta tipado estático y herramientas de desarrollo mejoradas, lo que reduce los errores y mejora la calidad del código.
- **Base de Datos:** Se utilizará una combinación de bases de datos SQL y NoSQL según las necesidades de cada microservicio.

4. Diseño de Base de Datos

SQL (PostgreSQL): Ideal para datos transaccionales y relaciones complejas. Se utilizará en microservicios como:

- Pedidos: Manejo de transacciones y consistencia.
- Clientes: Gestión de la información estructurada de los clientes.
- Payments: Registro de transacciones de pago.

NoSQL (MongoDB): Adecuado para datos no estructurados y consultas flexibles. Se utilizará en microservicios como:

- Productos: Almacenamiento de datos flexibles y catálogo de productos.
- Inventario: Gestión dinámica del stock de productos.
- Notificaciones: Manejo de datos no estructurados y envío de notificaciones.

5. Patrones de Diseño

- **Repository Pattern:** Para la abstracción de la capa de acceso a datos.
- **Singleton:** Para la conexión a la base de datos y manejo de logs.
- **Factory Pattern:** Para la creación de objetos de manera controlada.
- **Observer Pattern:** Para la implementación de notificaciones en tiempo real.

6. Estructura de Carpetas

Para cada microservicio se sugiere la utilización de la distribución de carpetas por dominio, por ejemplo para el microservicio de login existiría 2 dominios, uno para manejar las peticiones de login y otro para las de logout.

```
/src
  /common
    /decorators
    /filters
    /interceptors
  /config
  /modules
    /login
      /controllers
      /dto
      /entities
      /services
      /login.module.ts
    /logout
      /controllers
      /dto
      /entities
      /services
      /logout.module.ts
  /main.ts
  /app.module.ts
/test
  /e2e
  /unit
```