Patrón de Arquitectura

Patrón Empleado: Domain-Driven Design (DDD)

Este proyecto utiliza el enfoque **Domain-Driven Design** (DDD) para dividir responsabilidades y mantener un diseño modular y escalable.

Backend

Estructura de Carpetas

- **auth**: Módulo para autenticación y autorización (incluye JWT).
- **users**: Gestión de usuarios.
- **favorites**: Módulo para gestionar los favoritos de cada usuario.
- **marvel**: Comunicación con la API de Marvel.

Capas

1. **Entities**:
   Representan los modelos de datos y reflejan las tablas de la base de datos (ORM con TypeORM).
2. **Services**:
   Contienen la lógica de negocio. Implementan principios SOLID.
3. **Controllers**:
   Definen las rutas y los endpoints que los usuarios y el frontend pueden consumir.
4. **DTOs**:
   Gestionan la transferencia de datos y validan inputs.

Mejoras Propuestas

- Implementar interfaces de repositorio para desacoplar la lógica del acceso a datos.
- Dividir en capas adicionales como **domain** (entidades y lógica de negocio) y **application** (casos de uso).

Frontend

Estructura de Carpetas

1. **components**:
   Dividido en módulos como auth, comics y favorites.
   Cada módulo tiene:
   - Componentes específicos (por ejemplo, login, register).
   - Servicios para consumir datos relacionados.

2. **shared**:
   Contiene componentes reutilizables como navbar, modales y guards.
3. **Interceptors**:
   Manejan errores globales y gestionan encabezados HTTP automáticamente.

Flujo General

1. El usuario interactúa con la interfaz de Angular.
2. Angular realiza solicitudes HTTP al backend desarrollado en NestJS.
3. El backend maneja las solicitudes, consulta la base de datos y devuelve las respuestas.