El desarrollo de un proyecto como lo es una aplicación de comercio electrónico, tiene una complejidad alta, debido a una cantidad increíble de opciones que debe brindar tanto para vendedores, como para compradores. Tales como seguridad, concurrencia, pasarelas de pagos, notificaciones, servicio al cliente, entre otras muchas cosas. Debido a esto, ahora se va a plantear una estructura que comprende, organización de archivos, arquitectura, tecnologías, entre otras cosas a tener en cuenta para la realización de un proyecto de tal magnitud.

1. Tecnologías a utilizar

    1.1 Lenguajes de programación.

    - Javascript/Typescript: Es una tecnología que creció de una manera increíble, y que se suele usar tanto en el FrontEnd como en el BackEnd, lo cual permitiría usarlo en todo el stack de desarrollo.

    1.2 Frameworks y librerías

    - Nest js: Es un framework de Node js que ha ganado mucha popularidad debido al planteamiento modular que implementa. También, al hacer uso de Typescript por defecto, ayuda que los datos deban ser tipados, evitando errores en producción.

    1.3 Bases de datos

    - SQL(PostgreSQL, MySQL): Siempre se recomienda el uso de bases de datos relacionales, ya que permiten mantener una integración referencial muy fuerte, y en este caso que los datos serán estructurados de una manera muy puntual. De igual forma, no se debe descartar el uso de NoSQL (MongoDB), ya que, existe una posibilidad alta de tener que leer altos volúmenes de datos para mostrar al usuario.

    1.4 Autenticación y autorización

    - JWT (JSON Web Tokens): Para manejar autenticación y autorización.
    - OAuth 2.0: Para autenticación externa (Si llegase a ser necesario).

1.5 Extras

- Redis: Mejorar el rendimiento a través del uso de su sistema de aprovechar la cache.
- Docker: Portabilidad, disponibilidad y despliegue.
- Kubernetes: Para orquestación de contenedores en entornos de producción.
- NGINX: Como servidor web y proxy inverso.

2. Arquitectura del sistema

Con relación a este punto cabe aclarar que existen las posibilidades de monolito y microservicios. Siempre es recomendado empezar realizando un monolito muchas veces, y una vez el proyecto vaya escalando, aquellos módulos que tienden a crecer y ser más demandados, empieza la necesidad de orientar la solución a microservicios y escalar sin afectar el resto de módulos.

Esto último es lo más recomendable, ya que muchas veces el hacer uso desde le inicio de múltiples servicios para manejar la aplicación, devenga en costos altos, y esto algo lo cual se debe tener en cuenta también al momento de desarrollar una solución. Entre los componentes principales, se podrían encontrar:

- Usuarios
- Productos
- Pedidos
- Pagos
- Notificaciones

3. Organización de archivos

```
/src
  |__  /commons
              |_____  /guards
              |_____  /interceptors
              |_____  /filters
  |__  /config
  |__  /modules
              |_____  /auth
              |_____  /user
              |_____  /products
              |_____  /orders
              |_____  /payments
  |__  /utils
       /tests
```

- /common: Contiene código compartido entre múltiples módulos, como decoradores, guards, interceptors y filtros.

- /config: Archivos de configuración para la base de datos, aplicación y otros servicios.

- /modules: Cada subdirectorio representa un módulo con su propia funcionalidad (auth, users, products, orders, payments). Cada módulo tiene:
-
    - module.ts: Define el módulo y sus dependencias.
    - controller.ts: Maneja las solicitudes HTTP y responde a los clientes.
    - service.ts: Contiene la lógica de negocio.
    - entity.ts: Define las entidades del modelo de datos.
    - dto/: Contiene los Data Transfer Objects para validar y transferir datos entre capas.

- /utils: Utilidades y funciones auxiliares usadas en diferentes partes de la aplicación.

- app.module.ts: Módulo raíz que importa y organiza todos los módulos de la aplicación.

- main.ts: Punto de entrada principal de la aplicación.

4. Patrones de Diseño en NestJS

   NestJS utiliza varios patrones de diseño integrados que ayudan a mantener una arquitectura limpia y escalable:

   - Inversión de Control (IoC): NestJS usa un contenedor IoC para administrar la inyección de dependencias.

   - Dependency Injection: Proporciona dependencias a través de constructores o decoradores, mejorando la modularidad y facilitando las pruebas.

   - DTOs (Data Transfer Objects): Utilizados para definir y validar los datos que se transfieren entre las capas de la aplicación.

   - Facade(Fachada): Un servicio que actúe como una capa de abstracción sobre varios otros servicios.

   - Decorator(Decorador): Utilizado para enriquecer y modificar el comportamiento de las clases y sus métodos de una manera declarativa.

   - Interceptor: Permite añadir lógica antes y después de la ejecución de un método manejador de una ruta (controller)

5. Implementación de Seguridad

   5.1 Autenticación y Autorización

   - Uso de JWT para sesiones.
   - Roles y permisos claramente definidos para recursos sensibles.

   5.2 Validación de datos

   - Validar las entradas del usuario para prevenir inyecciones SQL y otros ataques.

   5.3 Cifrado

   - Cifrado de datos sensibles (por ejemplo, contraseñas, información de tarjetas de crédito).

6. Pruebas y Despliegue

El apartado de pruebas es algo que no se puede esquivar al momento de realizar una aplicación empresarial de tal magnitud. Esto es debido a que debe existir una integridad en todo lo que concierne al desarrollo, tales como bloquear el ***merge*** de un **PR** si las validaciones(cobertura, escáneres de seguridad, code smells, etc), no cumplen de forma satisfactoria lo requerido.

6.1 Pruebas

- Unitarias: Probar funciones individuales.
- Integración: Probar la interacción entre diferentes módulos.
- E2E (End-to-End): Simular el comportamiento del usuario para probar el flujo completo.

6.2 Despliegue

- CI/CD: Usar herramientas como Jenkins, GitHub Actions, o GitLab CI para automatizar el proceso de pruebas y despliegue.
- Contenedorización: Uso de Docker para crear entornos de despliegue consistentes.
- Orquestación: Kubernetes para manejar la escala y la alta disponibilidad.