E-commerce BE-MASTER - Backend

Arquitectura General

1. Lenguaje de Programación:
    - Utilizar TypeScript como el lenguaje principal, aprovechando la capacidad de tipado estático y otras características modernas que ofrece.
2. Framework:
    - Emplear NestJS como el framework principal, ya que facilita la creación de microservicios y sigue una arquitectura basada en módulos, controladores y servicios.

Microservicios

1. División de Servicios:
    - Diseñar microservicios independientes para funciones específicas como usuarios, productos, carrito de compras y pedidos. Cada microservicio se enfocará en una tarea específica para promover la escalabilidad y el mantenimiento, para la comunicación con el frontend se usará BFF para comunicar los servicios y devolver de una forma eficiente la data solicitada.

Base de Datos:

1. Almacenamiento de Datos:
    - Utilizar una combinación de bases de datos según los requisitos específicos. PostgreSQL para datos transaccionales y MongoDB para datos más flexibles o información de productos.

Autenticación y Autorización:

1. Autenticación Centralizada:
    - Implementar un servicio de autenticación centralizado utilizando JWT para gestionar la autenticación y autorización. AWS Cognito puede ser una opción para gestionar usuarios de manera segura.

Almacenamiento de Archivos:

1. Recursos Estáticos:
    - Almacenar recursos estáticos como imágenes de productos en Amazon S3 para aprovechar su escalabilidad y capacidad de gestión de archivos a gran escala.

Patrones de Diseño:

1. Arquitectura Modular:
    - Seguir el patrón de arquitectura modular de NestJS con módulos, controladores y servicios para mantener un código organizado y fácil de entender.
2. Inyección de Dependencias:
    - Aplicar el patrón de Inyección de Dependencias nativo en NestJS para facilitar la gestión de dependencias y mejorar la modularidad.
3. BFF (Backend For Frontend):
    - Teniendo en cuenta que es un e-commerce, el proyecto requerirá algunas vistas que muestren datos de distintos servicios, la implementación de este patrón ayudará a una respuesta óptima sin mucha transformación donde se servirá la data de una mejor manera sin consultar microservicios innecesariamente dando como resultado una eficiencia de carga a nivel general.

Seguridad:

1. SSL/TLS:
    - Implementar conexiones seguras mediante SSL/TLS para proteger la transferencia de datos sensibles.
2. Validación de Datos:
    - Realizar validación de datos en el servidor para prevenir ataques como la inyección de SQL y XSS.
3. Políticas de Seguridad:
    - Configurar políticas de seguridad para garantizar la autenticación y autorización adecuadas
4. OWASP:
    - Seguir la metodología OWASP permitirá mitigar riesgos de ataques e incrementar la seguridad con transacciones financieras, gestionar las sesiones de los usuarios de una forma más segura, poder evitar ataques SQL y XSS de una forma más segura.

Rendimiento y Escalabilidad:

1. CDN:
    - Considerar el uso de servicios como Amazon CloudFront para distribuir contenido estático globalmente y mejorar la velocidad de carga de recursos.
2. Escalabilidad Automática:
    - Aprovechar la escalabilidad automática proporcionada por servicios como AWS Elastic Beanstalk o AWS Fargate para manejar automáticamente el aumento o disminución de la carga de trabajo.

Monitoreo y Registro:

1. Monitoreo:
    - Habilitar el monitoreo y registro para analizar el tráfico y detectar problemas.
2. Registros de Auditoría:
    - Implementar registros de auditoría para rastrear actividades y detectar posibles amenazas de seguridad.