PRUEBA TÉCNICA DESARROLLADOR FULLSTACK

Jaime Enrique Barrera Sandoval

Correo: jaenba10@gmail.com
Teléfono: 3123702377

Portafolio: Portafolio
Repositorio: GitHub

3. Diseño de Arquitectura de Microservicios para una Plataforma de Comercio Electrónico Agrícola

1. Servicios Considerados y su Responsabilidad

Para garantizar una arquitectura eficiente y escalable, la plataforma debe dividirse en varios microservicios con responsabilidades bien definidas:

- **Servicio de Gestión de Productos**: Maneja la creación, actualización y eliminación de productos agrícolas. También almacena detalles como precio, disponibilidad y características.
- **Servicio de Autenticación y Seguridad**: Gestiona el registro, autenticación y autorización de usuarios mediante JWT y OAuth2.
- **Servicio de Inventario**: Controla el stock de productos en tiempo real, permitiendo actualización automática y alertas de escasez.
- **Servicio de Pedidos y Pagos**: Administra las compras, procesamientos de pagos y generación de facturas. Puede integrarse con pasarelas de pago externas.
- **Servicio de Comunicación y Notificaciones**: Envía alertas sobre estados de pedidos, promociones y mensajes de soporte a clientes y proveedores.

2. Comunicación entre Microservicios

Para garantizar una interacción eficiente y confiable entre los microservicios, es importante elegir la estrategia adecuada:

- **REST API**: Ideal para comunicación síncrona entre servicios con un menor nivel de complejidad. Se utiliza para operaciones como consulta de productos y procesamiento de pedidos.
- **gRPC**: Ofrece alto rendimiento y menor consumo de ancho de banda en comparación con REST. Es útil cuando se requiere comunicación rápida entre servicios internos.

- **Mensajería Asíncrona (Kafka, RabbitMQ)**: Para eventos de alto volumen, como actualizaciones de inventario o procesamiento de pagos en segundo plano. Permite desacoplamiento entre servicios y una mayor escalabilidad.

Cada opción tiene sus ventajas y la mejor estrategia dependerá de las necesidades específicas de la plataforma.

3. Seguridad en la Comunicación

La seguridad es un aspecto clave en una arquitectura de microservicios. Se pueden implementar varias estrategias para garantizar la integridad y confidencialidad de los datos:

- **Autenticación y Autorización**: Uso de **JWT** (JSON Web Tokens) para proteger las APIs, garantizando que solo usuarios autenticados accedan a recursos específicos.
- **Cifrado de Datos**: Implementación de protocolos de seguridad como **TLS/SSL** para cifrar la comunicación entre microservicios y prevenir ataques de intermediarios.
- **Gestión de Permisos**: Aplicación de **Spring Security** y OAuth2 para manejar roles y permisos dentro del sistema, asegurando que cada usuario tenga acceso solo a los datos necesarios.
- **Monitorización y Auditoría**: Uso de herramientas como **ELK Stack** (Elasticsearch, Logstash, Kibana) para detectar y responder a posibles amenazas en tiempo real.

Este enfoque proporciona una arquitectura modular, escalable y segura para el comercio electrónico en el sector agrícola.

4. Optimización de Rendimiento en Aplicaciones Web con Angular, React y Vue

1. Minimización y Compresión de Recursos

Reducir el tamaño de los archivos de la aplicación ayuda a mejorar los tiempos de carga. Se pueden aplicar las siguientes técnicas:

- **Compresión Gzip/Brotli**: Configurar el servidor para comprimir archivos estáticos (HTML, CSS, JS) antes de enviarlos al navegador.
- **Minificación de código**: Herramientas como Webpack y esbuild permiten eliminar espacios y caracteres innecesarios en archivos JS y CSS.
- **Carga diferida de recursos (Lazy Loading)**: Permite cargar módulos de la aplicación solo cuando se necesitan, evitando que toda la aplicación se descargue de inmediato.

2. Optimización de Imágenes y Recursos Multimedia

Las imágenes y videos pueden afectar significativamente la velocidad de carga. Para optimizarlos:

- **Uso de formatos modernos**: WebP ofrece mejor compresión que PNG y JPEG sin perder calidad.
- **Carga condicional de imágenes**: Implementar carga adaptable según la resolución del dispositivo y el ancho de banda disponible.
- **Uso de CDN**: Los sistemas de distribución de contenido (CDN) permiten almacenar imágenes y videos en múltiples servidores cercanos al usuario.

3. Optimización del Renderizado y Estado de la Aplicación

El rendimiento en aplicaciones SPA (Single Page Application) puede verse afectado por una mala gestión del estado y el renderizado:

- **Virtual DOM y Renderizado Eficiente**: React y Vue usan Virtual DOM para reducir el número de actualizaciones en la UI. Evitar renders innecesarios optimiza la velocidad.

- **Uso de NgRx (Angular) y Redux (React)**: Estas librerías ayudan a manejar el estado global eficientemente, evitando cálculos repetitivos y re-renderizados costosos.
- **Evitar re-renders excesivos**: Usar useMemo y useCallback en React, junto con trackBy en Angular, para optimizar el renderizado de componentes.

Con estas estrategias, la aplicación mejorará su velocidad de carga en dispositivos móviles, garantizando una mejor experiencia de usuario (UX/UI).

5. Medidas de Seguridad en APIs y Despliegue en la Nube

3. Diferencia entre OAuth y JWT

- **OAuth** es un marco de autorización que permite a las aplicaciones acceder a recursos en nombre de un usuario. Con **OAuth 2.0**, las aplicaciones obtienen tokens de acceso para interactuar de forma segura con APIs.
- **JWT (JSON Web Token)** es un formato de token compacto que transporta datos de autenticación y autorización, ideal para autenticación sin estado en APIs.

2. Protección contra ataques de fuerza bruta en una API en Azure

- Implementar **limitación de tasa** con Azure API Management para restringir solicitudes excesivas.
- Usar **Azure Web Application Firewall (WAF)** para detectar y mitigar patrones de ataque.
- Aplicar **autenticación fuerte** con OAuth, OpenID Connect o claves API.
- Configurar **bloqueo de cuentas** tras múltiples intentos fallidos de inicio de sesión.

3. Medidas de seguridad para datos sensibles en MongoDB

- Activar **cifrado TLS/SSL** para proteger los datos en tránsito.
- Usar **cifrado a nivel de campo** para información crítica.
- Aplicar **control de acceso basado en roles (RBAC)** para restringir el acceso no autorizado.

- Mantener actualizados los **parches de seguridad** de MongoDB.

4. Gestión del despliegue de microservicios en la nube

- Utilizar **Kubernetes** o **Docker** para desplegar contenedores de forma escalable.
- Implementar **descubrimiento de servicios** y **balanceo de carga** con Istio o Azure Service Fabric.
- Asegurar la **comunicación entre microservicios** con TLS mutuo o autenticación OAuth.
- Monitorear servicios con **herramientas de logging y observabilidad** como Azure Monitor.