Preguntas arquitectura

Fase 2: Preguntas de Arquitectura y Experiencia

Microservicios:

Si el sistema creciera y necesitará pasar de monolito a microservicios, ¿cómo dividirías los

servicios y qué consideraciones de comunicación implementarías?

Respuesta

Dividiría la aplicación en estos servicios básicos

Servicios principales

- Auth Service: Login, registro y manejo de tokens JWT
- User Service: Perfiles de usuario y configuraciones
- Debt Service: Crear, editar, eliminar y consultar deudas
- Notification Service: Emails y notificaciones push

Comunicación entre servicios

- HTTP REST para llamadas directas cuando necesito respuesta inmediata
- Message Queue, uso Redis o RabbitMQ para eventos como "deuda pagada"
- API Gateway para un solo punto de entrada que rutea las peticiones
- Service Discovery para que los servicios se encuentren entre sí
- Empezaría simple con HTTP calls y después agregaría messaging conforme crezca.

Optimización en la nube

Supongamos que tu aplicación corre en AWS. ¿Qué servicios usarías y porque para:

- Autenticación segura.

  AWS Cognito: Porque maneja todo el tema de usuarios, passwords, y tokens sin que yo tenga que programar mucho. Incluye MFA gratis.

- Base de datos.

  RDS PostgreSQL: Porque ya conozco PostgreSQL y RDS me da backups automáticos y actualizaciones sin dolores de cabeza.

- Cache y escalabilidad.

  ElastiCache Redis: Para cachear consultas frecuentes de deudas

  Auto Scaling Groups: Para que se agreguen más instancias cuando hay mucho tráfico

  CloudFront: Para servir archivos estáticos más rápido

- Balanceo de carga.

  Application Load Balancer de AWS

Buenas prácticas de seguridad

Menciona al menos 3 prácticas clave que aplicarías para garantizar seguridad en backend,

frontend y despliegue en la nube.

Backend

- alidación exhaustiva: Sanitización de inputs, validación en capas (controller + service)
- Secrets management: AWS Secrets Manager para credenciales, nunca hardcodear
- Rate limiting: Throttling de APIs para prevenir ataques DDoS

Frontend

- Content Security Policy (CSP): Headers HTTP para prevenir XSS
- Secure token storage: HTTPOnly cookies o secure localStorage con encriptación

- Input sanitization: Escapar datos antes de renderizar, usar librerías confiable

Despliegue

- Secrets Manager: Para credenciales de base de datos y APIs
- Security Groups restrictivos: Solo abrir los puertos necesarios
- SSL/TLS certificates: Usar AWS Certificate Manager para HTTPS automático

PostgreSQL vs NoSQL

¿En qué escenarios usarías PostgreSQL y en cuáles una base NoSQL? Explica con ejemplos

concretos.

Uso PostgreSQL cuando:

- Necesito transacciones: Como transferencias de dinero donde todo debe ser consistente
- Relaciones complejas: Mi app de deudas donde usuarios se relacionan con múltiples deudas
- Reportes con JOINs: Necesito consultas como "usuarios que más deben por mes"
- Datos estructurados: Esquemas fijos como User, Debt, Payment

Uso NoSQL (MongoDB/DynamoDB) cuando:

- Escalabilidad masiva: Si tuviera millones de usuarios necesitaría sharding automático
- Datos no relacionales: Logs de aplicación, eventos de usuario, métricas
- Esquema flexible: Catálogos de productos donde cada item tiene atributos diferentes
- Cache simple: Para guardar resultados de APIs que cambian poco

Despliegue

Si tuvieras que desplegar esta app en producción, ¿qué pipeline CI/CD diseñarías para

asegurar calidad, testeo y despliegue continuo?

Pipeline CI/CD

Cuando trabajo en una feature:

- Antes de hacer commit: linting y tests unitarios
- Si algo falla, no me deja subir el código
- Build local para asegurarme que compila

Al crear Pull Request:

- Otro dev tiene que revisar mi código (obligatorio)
- Se ejecutan tests de integración automáticamente
- Scan rápido de seguridad con herramientas básicas
- Si algo está rojo, no se puede mergear

Después del merge a main:

- Corren todos los tests (unitarios, integración, e2e)
- Se arma la imagen Docker con el nuevo código
- Deploy automático a ambiente de staging
- Tests básicos de smoke (endpoints principales funcionando)

Para subir a producción:

- Por ahora requiere aprobación manual mía o del tech lead
- Primero va al 50% del tráfico, el otro 50% sigue en la versión anterior
- Si las métricas se ven bien después de 15-30 min, se va al 100%
- Si algo se rompe, rollback automático a la versión anterior

Herramientas:

- GitHub Actions para todo el pipeline
- Docker para empaquetado

- AWS ECS para deployments
- CloudWatch para monitorear que todo esté bien