

Sistema de Pagos Orientado a Eventos

PARTE 1: DESAFÍO DE DISEÑO DE ARQUITECTURA

Resumen

Diseñar una arquitectura integral de sistema de procesamiento de pagos orientada a eventos que maneje el pago de un servicio, gestión de billetera, recolección de métricas y respuesta de pago asincrónica con manejo de errores.

Requisitos del Negocio

Funcionalidad Principal

- **Procesamiento de Pagos:** Los usuarios pueden pagar servicios usando el saldo de su billetera.
- **Gestión de Billetera:** Rastrear saldos de usuarios, deducir fondos, manejar reembolsos.
- **Recolección de Métricas:** Análisis en tiempo real de patrones de pago y tasas de éxito.
- **Procesamiento Asincrónico:** Manejar respuestas de pasarelas de pago externas.
- **Recuperación de errores:** Manejo integral de fallas y compensación.

Entregables del Diseño de Arquitectura

1. Diagrama de Arquitectura del Sistema

Crear diagramas que muestren:

- **Arquitectura de alto nivel** con todos los servicios y componentes.
- **Diagramas de flujo de eventos** para diferentes escenarios.
- **Flujo de datos** entre servicios.
- **Puntos de integración** con sistemas externos.

2. Documento de Diseño de Servicios

Documentar cada servicio, incluyendo:

- **Límites del servicio** y responsabilidades.
- **Esquemas de eventos** para publicar/suscribir (opcional).
- **Dependencias de servicios** y patrones de comunicación.

3. Especificación de Diseño de Eventos

Definir la arquitectura orientada a eventos:

- **Catálogo de eventos** con todos los tipos de eventos utilizados.
- **Estructura de temas/colas** y convenciones de nombres.
- **Orden de eventos** y garantías de entrega (opcional).
- Consideraciones de **event sourcing** (opcional).
- **Patrones Saga** para flujos de trabajo complejos (opcional si se usa).

4. Recomendación de Stack Tecnológico

Justificar las elecciones tecnológicas para:

- **Corredores de mensajes** (Kafka, RabbitMQ, etc.).
- **Bases de datos** (relacionales, NoSQL, series de tiempo).
- **Soluciones de caché** (opcional si se usan).

5. Estrategia de Manejo de Errores

Gestión integral de fallas:

- Identificación de **escenarios de falla**.
- **Estrategias de reintento** y políticas de retroceso exponencial.
- Manejo de **colas de mensajes fallidos (dead letter queue)**.
- **Transacciones compensatorias**.
- Implementaciones de **circuit breaker** (opcional).

6. Plan de Escalabilidad (opcional)

Diseño para la escala:

- Estrategias de **escalamiento horizontal**.
- Enfoques de **balanceo de carga**.
- **Particionamiento de bases de datos** (si es necesario).
- **Estrategias de caché**.
- Análisis de **cuellos de botella de rendimiento**.

Escenarios Específicos a Abordar

Escenarios de Flujo de Pago

1. **Ruta Feliz**: Pago exitoso de principio a fin.
 2. **Saldo Insuficiente**: Flujo de rechazo inmediato.
 3. **Tiempo de Espera de Pasarela Externa**: Reintento y fallback.
 4. **Pagos Concurrentes**: Manejo de condiciones de carrera.
 5. **Recuperación del Sistema**: Reinicio y reconstrucción del estado.
-

PARTE 2: DESAFÍO DE IMPLEMENTACIÓN

Contexto

Implementar un sistema de pagos orientado a eventos basado en la arquitectura diseñada en la Parte 1. El foco está en demostrar habilidades de diseño, código limpio y manejo de eventos, **no en crear una aplicación funcional**.

Tiempo estimado: 4 horas

Entregables Requeridos

Implementación

- **Estructura de proyecto** bien organizada
- Código que demuestre los patrones y principios aplicados
- **Pruebas unitarias** representativas

Componentes Imprescindibles

- **Diseño de base de datos:** Event store y read models
 - **Plan de escalabilidad:** Estrategias de crecimiento y performance
 - **Observabilidad:** Logging, métricas, health checks
-

Criterios de Evaluación

- ✓ **Arquitectura orientada a eventos** bien implementada
 - ✓ **Código limpio** siguiendo principios SOLID
 - ✓ **Separación de responsabilidades** clara
 - ✓ **Documentación** técnica completa
 - ✓ **Consideraciones de escalabilidad** realistas
-

Notas Importantes

- **No se requiere funcionalidad operativa** en cloud
 - Usar mocks/simulaciones para integraciones externas
 - Enfocarse en demostrar **comprensión arquitectónica**
 - Si usas IA (Cursor, Windsurf, etc.), **incluir historial de conversación**
-

Entrega

Subir código a repositorio Git con README detallado explicando decisiones técnicas y cómo ejecutar las pruebas.