Tarea 2 - 2025: Diseño de un Sistema de Microservicios para una Plataforma de Eventos

En esta tarea, diseñarán e implementarán un sistema de microservicios para **"EventFlow"**, una plataforma ficticia de gestión de eventos. El objetivo es aplicar los conceptos de bases de datos políglotas, consistencia, escalabilidad y patrones de diseño, justificando cada decisión.

Contexto de la Realidad Inventada

"EventFlow" es una empresa ficticia que gestiona la venta de entradas y la organización de eventos. Para manejar el alto volumen de transacciones y consultas, se ha optado por una arquitectura de microservicios.

Los microservicios clave son:

- **Servicio de Usuarios**: Gestiona los perfiles de los usuarios y su historial de compras.
- **Servicio de Eventos**: Maneja la información de los eventos, como la fecha, el lugar, el aforo total y las entradas disponibles.
- **Servicio de Reservas y Pagos**: Procesa las transacciones de compra de entradas y se comunica con otros servicios para completar la compra.

Requerimientos Clave y Retos del Diseño

Para cada uno de los siguientes puntos, deben justificar sus decisiones de diseño basándose en los requerimientos de la plataforma.

1. **Requerimientos de Consistencia y Lectura/Escritura**:
    - **Lectura de eventos y usuarios**: Debe ser extremadamente rápida y escalable. Se priorizará la **disponibilidad** y la **tolerancia a particiones**, aceptando la **consistencia eventual**.
    - **Escritura de reservas y pagos**: Requiere alta consistencia. Una reserva debe ser única y no puede haber dobles ventas. Se priorizará la consistencia sobre la velocidad.
2. **Modelado de Datos (NoSQL)**:
    - Deben elegir al menos dos bases de datos **NoSQL** y asignarlas a los microservicios.

- Justifiquen cómo modelaron los datos de eventos y usuarios. Decidan entre los patrones de datos `embebido` o `referencia` para relacionar la información.

3. **Patrones de Microservicios para Transacciones y Lógica de Negocio**:
   - `Patrón SAGA`: El proceso de compra de entradas es una transacción distribuida que involucra múltiples microservicios. Deben implementar el patrón **SAGA** con el enfoque de **Orquestación** para asegurar la consistencia.
     - **Implementación**: El Servicio de Reservas y Pagos debe actuar como un Orquestador Central que coordina la secuencia de pasos de la transacción. Detallen los pasos de la transacción exitosa y las transacciones de compensación en caso de fallo.
   - `Patrón Chain of Responsibility`: La lógica de negocio dentro del Servicio de Reservas y Pagos es secuencial. La solicitud de reserva debe pasar por una serie de validaciones y pasos. Usen el patrón **Chain of Responsibility** para estructurar esta lógica.
     - **Implementación**: Diseñen una cadena de "manejadores" para la solicitud de reserva. Los manejadores podrían incluir: ValidadorDeDatos, ValidadorDeInventario, ProcesadorDePago, etc.
4. **Patrones Avanzados de Bases de Datos (Opcional)**:
   - **Event Sourcing y CQRS**: Expliquen el concepto de **Event Sourcing** (almacenar eventos en lugar de estados) y cómo el patrón **CQRS** (separación de lecturas y escrituras) podría aplicarse a esta arquitectura.
     - **Uso**: Planteen un escenario en el que estos patrones serían beneficiosos para "EventFlow" y cómo se diferenciaría de la solución con MongoDB y Redis.
5. **Patrones de Privacidad de Datos (Opcional)**:
   - **Anonimización y Seudonimización**: Expliquen el propósito de la **anonimización** y la **seudonimización** de datos. Detallen cuándo y cómo aplicarían la **seudonimización** en este proyecto.
   - **Cifrado de Datos Personales**: Identifiquen los datos personales en la solución y expliquen cómo se podría aplicar la encriptación para protegerlos en la base de datos.
6. **Requerimientos Adicionales (Opcional)**:
   - **Exportación de Datos Anonimizados**: Deben agregar un *endpoint* que permita exportar los datos de usuarios o de eventos de forma masiva para análisis. La exportación debe garantizar que los datos personales estén **anonimizados irreversiblemente** y que la información del análisis (como el historial de eventos) se mantenga.

7. **Tecnologías de Despliegue**:
    - Se valorará usar Docker y Docker Compose y/o minikube. Se puede utilizar los lenguajes de programación y frameworks: Python (FastAPI), Spring Boot, Node con Express.
8. **Se valorará la realización de pruebas con JMeter (Opcional)**
9. **Se valorará la encriptación de tablas y/o base de datos. (Opcional)**

Descripción Detallada de los Endpoints a Diseñar

- **Servicio de Usuarios**:
    - `POST /api/usuarios`: Crea un nuevo usuario.
    - `GET /api/usuarios/{usuario_id}`: Recupera un usuario.
    - `GET /api/usuarios/exportar`: **Nuevo**. Exporta datos de usuarios en un formato adecuado para análisis (ej. JSON, CSV). Los datos personales deben estar **anonimizados**. (Opcional)
- **Servicio de Eventos**:
    - `POST /api/eventos`: Crea un nuevo evento.
    - `GET /api/eventos/{evento_id}`: Recupera la información de un evento y su aforo disponible.
- **Servicio de Reservas y Pagos**:
    - `POST /api/reservar`: Inicia la transacción **SAGA** y el proceso **Chain of Responsibility**.

Descripción de Datos de Usuario

El usuario, debe contener al menos:

```
{
   "tipo_documento": "string" (ej. "DNI", "Pasaporte"),
   "nro_documento": "string",
   "nombre": "string",
   "apellido": "string",
   "email": "string"
…
```

}

Guía de Entrega

Entrega de la Documentación (Git)

Se debe entregar un único repositorio de Git con un archivo `README.md` que contenga:

- **Justificación del Diseño**: Un apartado que explique las decisiones de diseño.
- **Diagrama de Arquitectura**: Un diagrama que muestre los microservicios, las bases de datos y las interacciones.
- **Diagrama de Flujo del Patrón SAGA**: Un diagrama que ilustre el flujo completo de la transacción.
- **Diagrama del Patrón Chain of Responsibility**: Un diagrama que muestre la secuencia de los manejadores dentro del Servicio de Reservas y Pagos.
- **Justificación del Requisito Adicional**: Si se implementa, justifiquen la estrategia de anonimización para la exportación de datos.

Entrega del Código (Git)

El código fuente de los microservicios y los archivos de configuración de Docker (o Kubernetes si es el caso) deben estar organizados en el mismo repositorio.

- **Estructura de Directorios**: Los microservicios deben estar en directorios separados.
- **Archivos de Configuración**: Se deben incluir los archivos `docker-compose.yml`.
- **Comandos de Ejecución**: En el `README.md` deben figurar los comandos para construir, levantar y probar la aplicación, y toda la documentación de la solución.

Fecha de entrega: 10 de noviembre