**MicroERP**

**📘 Documento de Diseño de Arquitectura**

**🎯 Objetivo**

Definir la planificación técnica del sistema **MicroERP**, basado en microservicios con **Spring Boot y Spring Cloud**, autenticación centralizada con **Keycloak**, mensajería con **RabbitMQ/Kafka**, y un stack de **observabilidad** para métricas, logs y trazas.

**🏗️ Arquitectura General**

**🔑 Infraestructura**

* **Config Server**: Centraliza la configuración de todos los microservicios.
* **Eureka Server**: Registro y descubrimiento de servicios.
* **API Gateway**: Entrada única al sistema, maneja enrutamiento, seguridad y filtros (Spring Cloud Gateway).
* **Keycloak**: Autenticación y autorización (OAuth2 + JWT).
* **Message Broker**: RabbitMQ o Kafka para eventos asincrónicos entre microservicios.

**⚙️ Microservicios de Negocio**

1. **User Service**
   * Maneja datos de usuario de negocio (perfil, teléfono, dirección, etc.).
   * Integración con Keycloak para credenciales y roles.
2. **Customer Service**
   * Gestión de clientes.
   * CRUD con persistencia independiente.
3. **Product Service**
   * CRUD de productos.
   * Expone métricas para Prometheus.
4. **Stock Service**
   * Control de inventario.
   * Se actualiza en base a eventos de ventas recibidos desde el broker.
5. **Sales Service**
   * Registro de ventas y facturación.
   * Publica eventos al broker para actualizar stock y notificar.
6. **Notification Service**
   * Escucha eventos de ventas confirmadas.
   * Envía correos/notificaciones a los clientes.

**🔍 Observabilidad**

* **Prometheus**: Recolección de métricas expuestas por Actuator.
* **Grafana**: Dashboards para visualización.
* **ELK Stack** (opcional): Logs centralizados.
* **Jaeger/Zipkin**: Trazas distribuidas entre servicios.

**🔧 Tecnologías**

* **Backend**: Spring Boot, Spring Cloud, Spring Data JPA, Spring Mail.
* **Seguridad**: Keycloak, OAuth2, JWT.
* **Mensajería**: RabbitMQ o Kafka.
* **Persistencia**: MySQL/PostgreSQL (cada microservicio con su propia base de datos).
* **Observabilidad**: Prometheus, Grafana, ELK, Jaeger/Zipkin.
* **Testing**: JUnit 5, Mockito, Testcontainers (bases y brokers en contenedores).
* **Logging**: Slf4j + Logback (estructurado).

**📌 Flujos principales**

1. **Autenticación**
   * Todo request ingresa por el **API Gateway**.
   * El Gateway valida el token JWT emitido por **Keycloak**.
2. **Venta**
   * Cliente → Gateway → **Sales Service**.
   * Sales Service registra venta y publica evento en **RabbitMQ/Kafka**.
   * **Stock Service** consume evento → descuenta inventario.
   * **Notification Service** consume evento → envía email al cliente.
3. **Consultas (REST síncronas)**
   * Usadas para CRUD de usuarios, clientes, productos, etc.
4. **Eventos (mensajería asíncrona)**
   * Usados para actualizar stock y enviar notificaciones sin bloquear la venta.

**🛡️ Resiliencia**

* Uso de **Resilience4j** para:
  + Circuit breakers.
  + Retries con backoff.
  + Rate limiting.

**📑 Documentación de APIs**

* Cada microservicio expone su propia **documentación Swagger/OpenAPI**.
* El Gateway puede centralizar el acceso a esa documentación.

**🚀 DevOps / CI-CD**

* **Contenerización**: Cada microservicio empaquetado en **Docker**.
* **Orquestación**:
  + **Docker Compose** para desarrollo local.
  + **Kubernetes** (opcional) para producción.
* **CI/CD**: Integración continua con GitHub Actions / GitLab CI.

**✅ Beneficios de esta arquitectura**

* Seguridad centralizada.
* Desacoplamiento de servicios.
* Escalabilidad horizontal.
* Observabilidad completa.
* Pruebas y logging desde el inicio.