Explicación del Diagrama de Arquitectura de Microservicios

Versión: 1.0
Fecha: 24 de Octubre de 2025

Este documento describe en detalle el diagrama de arquitectura general del sistema "Reconexión Humana", que consolida los principios de diseño, los patrones de comunicación y la estrategia de datos.

---

1. Propósito del Diagrama

El diagrama tiene como objetivo principal ofrecer una **visión holística y unificada** de la arquitectura del sistema. Sirve como un mapa conceptual y técnico para:

- **Visualizar los Bounded Contexts:** Muestra cómo el dominio del negocio se ha descompuesto en contextos delimitados, cada uno gestionado por un microservicio.
- **Ilustrar los Principios de DDD:** Expone conceptos clave como los Aggregate Roots dentro de cada contexto.
- **Aclarar los Patrones de Comunicación:** Diferencia claramente entre la comunicación síncrona (API calls) y la asíncrona (eventos).
- **Mostrar la Estrategia de Persistencia:** Refleja el enfoque de "Database per Service" y la persistencia políglota.

---

2. Interpretación de los Componentes Visuales

2.1. Bounded Contexts y Microservicios

- Cada **subgrafo rectangular** (ej. "AuthIdentity (Contexto de Identidad)") representa un **Bounded Context** de Domain-Driven Design.
- Dentro de cada subgrafo, el servicio (ej. `AuthIdentity Service`) es la **implementación concreta** de ese contexto como un microservicio.
- El **Core Domain** (`SocialConnect`) está resaltado en rojo para denotar su importancia estratégica y complejidad. Los demás son subdominios de soporte o genéricos.

2.2. Conceptos de DDD

- **Aggregate Root (Raíz de Agregado):** La etiqueta `<b>Aggregate Root:</b>` dentro de cada contexto (ej. `User`, `Publication`) identifica la entidad principal que actúa como punto de entrada y guardián de la consistencia para todas las operaciones dentro de ese agregado. Por ejemplo, para crear un comentario, se debe pasar a través del agregado `Publication`.

2.3. Persistencia de Datos

- El icono de base de datos (`(nombre_db)`) dentro de cada contexto ilustra el patrón **Database per Service**.
- Las etiquetas (`PostgreSQL`, `MongoDB`, `Neo4j`, `Cassandra`) demuestran la estrategia de **persistencia políglota**, donde se elige la tecnología de base de datos más adecuada para la tarea específica del microservicio.

---

3. Flujos de Comunicación

El diagrama utiliza dos tipos de líneas para representar los flujos de comunicación:

3.1. Comunicación Síncrona (Líneas continuas con flecha)

- Representan **llamadas directas de API (REST/HTTP)** que esperan una respuesta inmediata.
- **Flujo Cliente-Sistema:** El `Cliente` solo se comunica con el `API Gateway`. Este actúa como una fachada que enruta las peticiones al microservicio interno correspondiente. Esto simplifica la lógica del cliente y centraliza la seguridad.
- **Flujo Servicio-Servicio:** En algunos casos, un servicio puede llamar a otro directamente para obtener una respuesta necesaria en el momento (ej. `AuthSvc` llamando a `GuardianSvc` para validar una contraseña durante el registro). Este tipo de acoplamiento se usa con moderación.

3.2. Comunicación Asíncrona (Líneas hacia y desde el Message Broker)

- Representa un **patrón orientado a eventos** que utiliza un `Message Broker (RabbitMQ)` para desacoplar los servicios.
- **Publicación de Eventos:** Un servicio publica un evento de negocio (ej. `AuthSvc` publica `UserRegistered`) sin saber quién lo consumirá.
- **Consumo de Eventos:** Otros servicios se suscriben a los eventos que les interesan (ej. `SocialSvc` y `RiskSvc` consumen `UserRegistered` para realizar sus propias tareas).

- **Ventaja Principal:** Este desacoplamiento aumenta la **resiliencia**. Si `SocialSvc` está caído, el registro de usuarios en `AuthSvc` sigue funcionando, y el evento se procesará cuando `SocialSvc` se recupere.