Universidad Politécnica de Chiapas.

Cuatrimestre.	Septiembre- Diciembre 2025
Grupo.	Noveno B
Asignatura.	SOA
Corte	C2
Actividad.	Proyecto Integrador - Diagrama de dominio
Fecha	
Matricula.	221189 231208 221214 231198
Nombre.	Luis Alberto Batalla. Willber Hernandez. Esduardo Palomeque Roblero. Maria Fernanda Sanchez

Documentación Técnica: Sistema "Reconexión Humana"

Versión: 1.0
Fecha: 24 de octubre de 2025

1. Introducción

Este documento proporciona una guía técnica completa para el desarrollo, integración y mantenimiento del sistema "Reconexión Humana". Describe la arquitectura general, los principios de enfermedad, los dominios de negocio, los microservicios y sus responsabilidades.

El objetivo es establecer un entendimiento común y un conjunto de buenas prácticas para asegurar la construcción de un sistema robusto, escalable y mantenible.

---

2. Visita Arquitectónica

El sistema está enfermo seguido una **arquitectura de microservicios** con un enfoque **orientado a eventos**. Los principios clave hijo:

- **Separación de Responsabilidades:** Cada microservicio es dueño de una capacidad de negocio específica.
- **Bajo Acoplamiento:** Los servicios se comunican a través de APIs bien definidas y eventos asíncronos, minimizando las dependencias directas.
- **Base de Datos por Servicio:** Cada microservicio gestiona su propio esquema de base de datos, garantizando la autonomía y permitiendo la **persistencia políglota** (usar la mejor base de datos para cada tarea).
- **Punto de Entrada Único:** Un **API Gateway** actúa como fachada, simplificando el acceso desde el cliente y gestionando responsabilidades transversales como la autenticación y el enrutamiento.

---

3. Principios de Domain-Driven Design (DDD)

Aplicamos DDD para modelar el sistema en torno al negocio.

3.1. Lenguaje Ubicuo (Ubiquitous Language)

Para evitar ambigüedades, todo el equipo (desarrollo, negocio, etc.) debe usar un vocabulario común y bien definido.

- **Usuario (User):** Individuo con una cuenta en el sistema.
- **Publicación (Publication):** Contenido genérico creado por un usuario (puede ser un `Post` o una `Story`).
- **Interacción (Interaction):** Una acción de un usuario sobre una publicación, como un `Like` o un `Comment`.
- **Relación Social (Social Relationship):** Un vínculo entre dos usuarios, como `Follow` (Seguimiento) o `Block` (Bloqueo).
- **Perfil de Riesgo (Risk Profile):** Una evaluación interna y anónima del bienestar de un usuario, **totalmente aislada** del perfil social.

3.2. Dominios y Subdominios

- **Core Domain (Dominio Principal): Conexión Social y Bienestar**. Este es el corazón del negocio y donde debemos enfocar la mayor parte del esfuerzo de modelado. Incluye la creación de contenido, las interacciones sociales y la identificación proactiva de perfiles de riesgo para ofrecer ayuda.
- **Subdominios de Soporte (Supporting Subdomains):**
  - **Gestión de Identidad:** Autenticación y gestión de perfiles de usuario.
  - **Mensajería:** Comunicación en tiempo real entre usuarios.
- **Subdominios Genéricos (Generic Subdomains):**
  - **Notificaciones:** Envío de alertas push a los dispositivos.
  - **Seguridad de Contraseñas:** Verificación contra brechas de seguridad externas.

---

4. Bounded Contexts y Microservicios

Cada **Bounded Context** define un límite claro donde un modelo de dominio específico es consistente. En nuestra arquitectura, cada Bounded Context se materializa como un microservicio.

4.1. `AuthIdentity` (Contexto de Identidad)

- **Responsabilidad:** Gestionar el ciclo de vida de la identidad y el perfil del usuario. Es la fuente de la verdad para la entidad `User`.
- **Aggregate Root:** `User`. El `User` es la raíz de consistencia. Una transacción de registro debe crear tanto el `User` como su `UserProfile` de forma atómica.

- **Agregado (Aggregate):** El agregado `User` incluye la entidad `User` y la entidad `UserProfile`.
- **Entidades:** `User`, `UserProfile`.
- **Value Objects:** `Email`, `HashedPassword`, `Username`. Estos objetos encapsulan lógica de validación (ej. un `Email` debe tener un formato válido).
- **Transiciones de Estado:** Un `User` puede pasar de un estado `PendingVerification` a `Active`.

4.2. `SocialConnect` (Contexto Social y de Contenido)

- **Responsabilidad:** Orquesta toda la interacción social: publicaciones, comentarios, likes, y el grafo social (seguidores/bloqueos). Es el **Core Domain**.
- **Aggregate Roots:**
  - `Publication`: La raíz para todas las interacciones relacionadas. Para añadir un `Comment` o un `Like`, se debe cargar el agregado `Publication`. Esto asegura que no se pueda comentar una publicación borrada.
  - `UserNode`: En el contexto del grafo social, el usuario es el nodo principal. Las relaciones (`Follows`, `Blocks`) son aristas.
- **Agregados:**
  - El agregado `Publication` incluye la entidad `Publication` y sus colecciones de `Media` y `Comments`.
- **Entidades:** `Publication`, `Media`, `Comment`.
- **Value Objects:** `Location`, `MediaType`.

4.3. `MessagingService` (Contexto de Mensajería)

- **Responsabilidad:** Gestionar conversaciones y mensajes en tiempo real.
- **Aggregate Root:** `Conversation`. Todos los mensajes (`Message`) y participantes (`Participant`) pertenecen a una `Conversation`. Para enviar un mensaje, primero se debe validar el estado de la conversación y si el emisor es un participante válido.
- **Agregado:** El agregado `Conversation` incluye la entidad `Conversation`, su lista de `Participants` y la colección de `Messages`.
- **Entidades:** `Conversation`, `Message`, `Participant`.
- **Value Objects:** `MessageStatus` (ej. `SENT`, `DELIVERED`, `READ`).

4.4. `RiskMitigation` (Contexto de Análisis de Riesgo)

- **Responsabilidad:** Consumir eventos del sistema (ej. `UserInteracted`, `PostCreated`) de forma asíncrona para analizar patrones y construir un `RiskProfile`. **Este contexto es de solo lectura respecto a los datos sociales y su modelo está completamente aislado**.
- **Aggregate Root:** `RiskProfile`. Cada perfil es una unidad atómica de análisis asociada a un `user_id`.

---

5. Estrategia de Datos y Persistencia

Seguimos el patrón **Database per Service** con un enfoque de **persistencia políglota**:

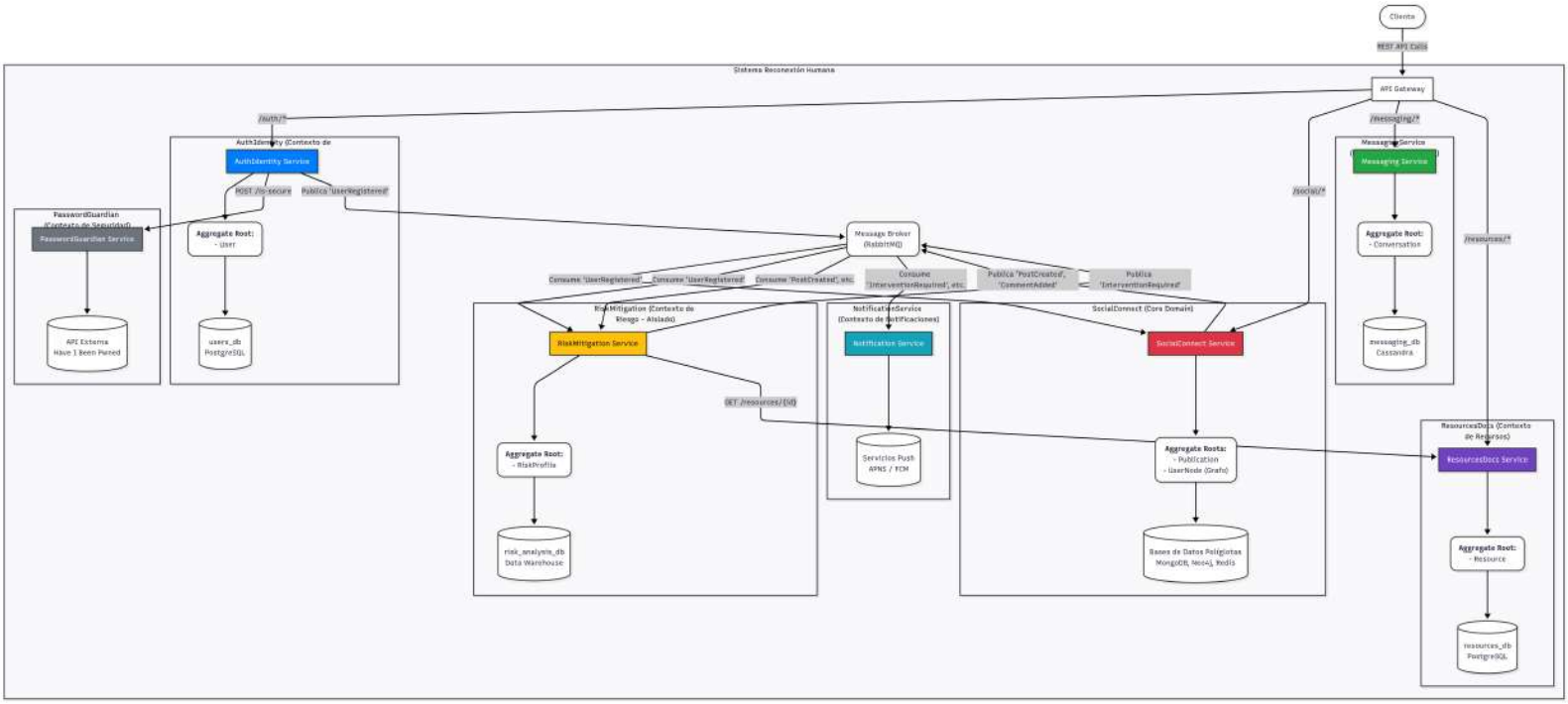
- `AuthIdentity` **-> PostgreSQL:** Necesitamos consistencia fuerte (ACID) y transacciones para el registro y la gestión de perfiles. Un modelo relacional es ideal.
- `SocialConnect` **-> Múltiples Tecnologías:**
  - **Contenido (`publications`, `media`):** Una **base de datos documental (ej. MongoDB)** es ideal para manejar la estructura flexible y anidada de las publicaciones.
  - **Grafo Social (`follows`, `blocks`):** Una **base de datos de grafos (ej. Neo4j)** es la más eficiente para consultas complejas como "amigos de mis amigos" o "contenido de personas que sigo".
  - **Interacciones (`likes`):** Una base de datos **Clave-Valor (ej. Redis)** puede usarse para contadores rápidos y volátiles.
- `MessagingService` **-> Base de Datos de Columna Ancha (ej. Cassandra):** Optimizada para un volumen de escritura masivo y consultas por rango de tiempo, perfecto para historiales de chat.

---

6. Patrones de Comunicación e Integración

- **Comunicación Síncrona (Cliente -> Sistema):** El cliente interactúa con el sistema a través del **API Gateway** usando una API RESTful. El Gateway enruta las peticiones al microservicio correspondiente.
- **Comunicación Asíncrona (Servicio <-> Servicio):** Los microservicios se comunican entre sí a través de un **Broker de Mensajes (RabbitMQ)**.
  - **Ejemplo de Flujo (Registro):**
    1. `AuthIdentity` guarda el nuevo usuario en su base de datos.
    2. Publica un evento `UserRegistered` en el broker.
    3. `SocialConnect` consume este evento para crear el "nodo" de usuario en su base de datos de grafos.
    4. `RiskMitigation` consume el evento para inicializar un `RiskProfile` vacío.

- **Ventajas:** Esto desacopla los servicios. Si `SocialConnect` está caído, el registro de usuarios sigue funcionando y el evento se procesará cuando el servicio se recupere.


Cliente
REST API Calls
Sistema Reconexión Humana
API Gateway
/auth/*
/messaging/*
/social/*
/resources/*
Auth1dentity (Contexto de
Auth1dentity Service
POST /to-secure
Publica 'UserRegistered'
PasswordGuardian
(Contexto de Seguridad)
PasswordGuardian Service
Aggregate Root:
- User
APIs Externas
Have I Been Pwned
users_db
PostgreSQL
Message Broker
(RabbitMQ)
Consume 'UserRegistered'
Consume 'UserRegistered'
Consume 'PostCreated', etc.
Consume
'InterventionRequired', etc.
Publica 'PostCreated',
'CommentAdded'
Publica
'InterventionRequired'
MessagingService
Messaging Service
Aggregate Root:
- Conversation
messaging_db
Cassandra
RiskMitigation (Contexto de
Riesgo - Aislado)
RiskMitigation Service
Aggregate Root:
- RiskProfile
risk_analysis_db
Data Warehouse
NotificationService
(Contexto de Notificaciones)
Notification Service
Servicios Push
APNS / FCM
SocialConnect (Core Domain)
SocialConnect Service
Aggregate Roots:
- Publication
- UserNode (Grafo)
Bases de Datos Poliglotas
MongoDB, Neo4j, Redis
REST /resources/{id}
ResourcesDocs (Contexto
de Recursos)
ResourcesDocs Service
Aggregate Root:
- Resource
resources_db
PostgreSQL

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.