**Universidad Politécnica de Chiapas.**

| | |
|---|---|
| **Cuatrimestre.** | Septiembre- Diciembre 2025 |
| **Grupo.** | Noveno B |
| **Asignatura.** | SOA |
| **Corte** | C2 |
| **Actividad.** | Analicis de Microservicio basado en Responsabilidades.s |
| **Fecha** | |
| **Matricula.** | 221189<br>231208<br>221214<br>231198 |
| **Nombre.** | Luis Alberto Batalla.<br>Willber Hernandez.<br>Esduardo Palomeque Roblero.<br>Maria Fernanda Sanchez |

# Diseño de Arquitectura de Microservicios: Reconexión Humana

**Versión:** 2.1
**Fecha:** 23 de Octubre de 2025

## 1. Introducción

Este documento detalla la arquitectura de microservicios para el proyecto "Reconexión Humana". La división se basa en el análisis previo de **Bounded Contexts** (Contextos Delimitados) y tiene como objetivo crear un sistema escalable, mantenible y seguro.

Se definen las responsabilidades de cada servicio, sus APIs, los eventos que emiten y consumen, los modelos de datos que gestionan y los patrones de comunicación para asegurar un **bajo acoplamiento** y una **alta cohesión**.

---

## 2. Definición de Microservicios

### 2.1. Microservicio: `AuthIdentity`

- **Bounded Context Asociado:** Gestión de Identidad y Autenticación.
- **Responsabilidad Principal:** Actuar como el **proveedor de identidad centralizado (Identity Provider)**. Es la única fuente de verdad sobre la identidad, autenticación y datos de perfil básicos de un usuario.
- **Justificación:** Centralizar la autenticación en un solo servicio reduce la duplicación de código, simplifica la gestión de la seguridad y permite tener un único punto para implementar políticas de acceso globales (ej. autenticación de dos factores).
- **Datos que Gestiona:**
    - **Entidades:** `User`, `UserProfile`. (Los tokens son efímeros y pueden no ser entidades persistentes).
    - **Value Objects:** `Email`, `PasswordHash`, `Username`. Estos objetos encapsulan reglas de validación (ej. formato de email, complejidad de la contraseña).

- **Tecnología Sugerida:** Base de datos relacional (PostgreSQL, MySQL) para garantizar consistencia y transacciones ACID.
- **Esquema de Datos:**

```
-- users_db
CREATE TABLE Users (
    user_id UUID PRIMARY KEY,
    username VARCHAR(50) UNIQUE NOT NULL,
    email VARCHAR(255) UNIQUE NOT NULL,
    password_hash VARCHAR(255) NOT NULL,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);

CREATE TABLE UserProfiles (
    user_id UUID PRIMARY KEY REFERENCES Users(user_id),
    full_name VARCHAR(150),
    bio TEXT,
    profile_picture_url VARCHAR(512)
);
```

- **API Sincrónica (REST)**

| Endpoint | Método | Descripción | Request Body (Schema) |
|---|---|---|---|
| `/register` | `POST` | Registra un nuevo usuario en el sistema. | `{ "username", "email", "password" }` |
| `/login` | `POST` | Autentica a un usuario y devuelve un token JWT. | `{ "email", "password" }` |
| `/token/validate` | `POST` | **Endpoint interno.** Valida un token JWT. Usado por otros servicios. | `{ "token": "..." }` |
| `/users/me` | `GET` | Devuelve el perfil del usuario autenticado. | (Autenticado con Bearer Token) |
| `/users/me` | `PATCH` | Actualiza el perfil del usuario autenticado (nombre, bio, etc.). | `{ "full name"?, "bio"?, "profile_picture_url"? }` |

- **Comunicación Asincrónica (Eventos Publicados)**
  - **Broker:** RabbitMQ
  - **Exchange:** `identity_events` (tipo `fanout`)

| Evento | Payload (Schema) | Propósito |
|---|---|---|

| Evento | Payload (Schema) | Propósito |
| --- | --- | --- |
| `UserRegistered` | `{ "user_id", "username", "email" }` | Notifica al resto del sistema que se ha creado una nueva identidad. |
| `UserProfileUpdated` | `{ "user_id", "updated_fields": ["..."] }` | Informa de cambios en el perfil para que otros servicios actualicen su caché. |

## 2.2. Microservicio: `SocialConnect`

- **Bounded Context Asociado:** Conexión Social y Contenido.
- **Responsabilidad Principal:** Gestionar toda la funcionalidad de la red social, incluyendo publicaciones, interacciones, relaciones entre usuarios y mensajería.
- **Justificación:** Agrupar estas funcionalidades, que son altamente cohesivas y colaboran constantemente, simplifica el desarrollo. El uso de persistencia políglota permite optimizar cada sub-dominio (ej. grafos para seguidores, series temporales para mensajes) sin crear una sobrecarga de microservicios separados.
- **Datos que Gestiona:**
  - **Entidades:** `Post`, `Media`, `Story`, `Comment`, `Conversation`, `Message`. Las relaciones (`Like`, `Follow`, `Block`) se modelan como aristas en el grafo.
  - **Value Objects:** `Location` (en un `Post`), `AudioTrack` (en `Media` de tipo video).
  - **Tecnología Sugerida:** Un enfoque políglota:
    - `posts_db` (Documental - MongoDB): Ideal para la estructura flexible y anidada de las publicaciones y sus metadatos.
    - `interactions_db` (NoSQL Clave-Valor - DynamoDB/Redis): Optimizado para escrituras y lecturas ultra rápidas de contadores (likes, vistas).
    - `graph_db` (Grafo - Neo4j): La herramienta perfecta para modelar y consultar relaciones complejas como "quién sigue a quién" o "quién bloquea a quién" de manera eficiente.
    - `messaging_db` (Columna ancha - Cassandra): Excelente para cargas de trabajo de series temporales como los chats, donde las escrituras son constantes y las lecturas se hacen por rangos de tiempo.
- **API Sincrónica (REST) - Ejemplo**

| Endpoint | Método | Descripción |
| --- | --- | --- |
| `/posts` | `POST` | Crea una nueva publicación. |
| `/posts/{postId}/comments` | `POST` | Añade un comentario a una publicación. |

| Endpoint | Método | Descripción |
| --- | --- | --- |
| `/posts/{postId}/likes` | `POST` | Da "me gusta" a una publicación. |
| `/users/{userId}/follow` | `POST` | Sigue a otro usuario. |
| `/feed` | `GET` | Obtiene el feed de contenido del usuario. |

- **Comunicación Asincrónica (Eventos)**
  - **Eventos Consumidos:**
    - **Exchange:** `identity_events` (de `AuthIdentity`)
    - **Acción:** Al recibir `UserRegistered`, crea el nodo de usuario en la base de datos de grafos.
  - **Eventos Publicados:**
    - **Broker:** RabbitMQ
    - **Exchange:** `social_events` (tipo `topic`)

| Evento | Routing Key | Payload (Schema) | Propósito |
| --- | --- | --- | --- |
| `PostCreated` | `social.post.created` | `{ "user id", "post_id", "media count", "timestamp" }` | Informar que se ha creado contenido nuevo. Clave para el análisis. |
| `CommentAdded` | `social.comment.added` | `{ "user id", "post_id", "comment id", "timestamp" }` | Registrar una interacción social. |
| `MessageSent` | `social.message.sent` | `{ "sender id", "conversation_id", "timestamp" }` | Registrar un evento de comunicación privada (sin contenido). |
| `FollowCreated` | `social.follow.created` | `{ "follower id", "following id", "timestamp" }` | Registrar un cambio en el grafo social. |

## 2.3. Microservicio: `RiskMitigation`

- **Bounded Context Asociado:** Análisis de Riesgo y Salud Pública.
- **Responsabilidad Principal:** Es el cerebro analítico del sistema. Procesa datos de comportamiento de forma asíncrona y anonimizada para identificar patrones de riesgo, ejecutar modelos de ML y generar planes de acción. **Este servicio no debe exponer ninguna API pública para evitar cualquier acceso directo a su lógica sensible.**

- **Justificación:** El aislamiento es crítico. Separar este servicio garantiza que los datos de salud y los modelos de riesgo estén en un entorno controlado y seguro. La comunicación asincrónica lo desacopla del resto del sistema, haciéndolo resiliente a fallos en otros servicios.

- **Datos que Gestiona:**

  - **Entidades:** `RiskProfile`, `HealthPlan`, `PlanIntervention`.
  - **Value Objects:** `RiskScore` (podría encapsular valor y categoría), `DateRange` (para `HealthPlan`).
  - **Tecnología Sugerida:** Data Warehouse o Data Lake (BigQuery, Redshift) para análisis de grandes volúmenes de datos.
  - **Esquema de Datos:**

```
-- risk_analysis_db
CREATE TABLE RiskProfiles (
    profile_id UUID PRIMARY KEY,
    user_id UUID UNIQUE NOT NULL, -- Único enlace al usuario
    risk_score FLOAT,
    risk_category VARCHAR(50),
    last_analysis_at TIMESTAMP WITH TIME ZONE
);

CREATE TABLE HealthPlans (
    plan_id UUID PRIMARY KEY,
    risk_profile_id UUID REFERENCES RiskProfiles(profile_id),
    status VARCHAR(50),
    start_date DATE
);
```

- **Comunicación Sincrónica (Saliente)**

| Petición | Destino | Justificación |
| --- | --- | --- |
| `GET /resources/{resId}` | `Resources Docs` | Para obtener detalles (título, URL) de un recurso antes de sugerirlo. |

- **Comunicación Asincrónica (Eventos)**

  - **Eventos Consumidos:**

    - **Exchange:** `social_events` (de `SocialConnect`)
    - **Binding Key:** `social.#` (se suscribe a todos los eventos sociales).
    - **Acción:** Ingiere los eventos en el Data Lake para su posterior análisis por los jobs de ML.

- **Eventos Publicados:**
  - **Broker:** RabbitMQ
  - **Exchange:** `interventions_exchange` (tipo `direct`)

| Evento | Routing Key | Payload (Schema) | Propósito |
|---|---|---|---|
| `InterventionRequired` | `notification.suggest` | `{ "user_id", "action_type": "SUGGEST_RESOURCE", "resource_id" }` | Solicita al `NotificationService` que envíe una sugerencia al usuario. |

## 2.4. Microservicio: `ResourcesDocs`

- **Bounded Context Asociado:** Gestión de Documentos y Recursos.
- **Responsabilidad Principal:** Funcionar como un Sistema de Gestión de Contenidos (CMS) para los artículos, guías y materiales de apoyo que se usan en los planes de salud.
- **Justificación:** Separar la gestión de contenido estático del resto de la lógica de la aplicación permite usar herramientas y flujos de trabajo optimizados para CMS (ej. editores de texto enriquecido, control de versiones de documentos) sin complicar los otros servicios.
- **Datos que Gestiona:**
  - **Entidades:** `Resource`, `ResourceCategory`.
  - **Value Objects:** `Metadata` (conjunto de pares clave-valor para un recurso).
  - **Tecnología Sugerida:** Base de datos relacional con buen soporte para JSON (PostgreSQL con JSONB).
- **API Sincrónica (REST)**

| Endpoint | Método | Descripción |
|---|---|---|
| `/resources` | `GET` | Lista recursos, con filtros por categoría. |
| `/resources` | `POST` | **(Admin)** Crea un nuevo recurso. |
| `/resources/{resId}` | `GET` | Obtiene los detalles de un recurso específico. |
| `/resources/{resId}` | `PUT` | **(Admin)** Actualiza un recurso existente. |
| `/categories` | `GET` | Lista todas las categorías de recursos. |

- **Comunicación Asincrónica (Eventos Publicados)**
  - **Broker:** RabbitMQ

- **Exchange:** `resources_events` (tipo `fanout`)

| Evento | Payload (Schema) | Propósito |
|---|---|---|
| `ResourcePublished` | `{ "resource id", "title", "category_id" }` | Informa que hay nuevo contenido disponible. |
| `ResourceUpdated` | `{ "resource id", "updated_fields": ["..."] }` | Permite a otros servicios invalidar cachés relacionadas con este recurso. |

## 2.5. Microservicio: `NotificationService` (Nuevo)

- **Bounded Context Asociado:** Notificaciones y Comunicaciones Salientes.
- **Responsabilidad Principal:** Gestionar el envío de todas las comunicaciones al usuario (notificaciones push, emails, mensajes in-app). Actúa como un router centralizado para las notificaciones.
- **Justificación:** Centralizar las notificaciones en un solo lugar evita que cada servicio (`SocialConnect`, `RiskMitigation`) tenga que implementar la lógica para conectarse con proveedores de push (APNS, FCM) o de email. Simplifica la gestión de plantillas y las preferencias de notificación del usuario.
- **Datos que Gestiona:**
  - **Entidades:** `NotificationLog`.
  - **Value Objects:** `DeviceToken` (encapsula el token y el tipo de plataforma, ej. APNS, FCM).
  - **Tecnología Sugerida:** Base de datos relacional (MySQL) o una caché (Redis) para los tokens de dispositivo, que cambian con frecuencia.
- **Comunicación Asincrónica (Eventos Consumidos)**

| Evento Consumido | Exchange Origen | Binding Key | Acción |
|---|---|---|---|
| `InterventionRequired` | `interventions_exchange` | `notification.suggest` | Recibe la solicitud de `RiskMitigation` y envía una notificación push con la sugerencia de recurso. |
| `CommentAdded` | `social_events` | `social.comment.added` | (Opcional) Notifica al autor del post que ha recibido un nuevo comentario. |
| `FollowCreated` | `social_events` | `social.follow.created` | (Opcional) Notifica al usuario que tiene un nuevo seguidor. |

---

### 2.6. Microservicio: `PasswordGuardian` (Nuevo)

- **Bounded Context Asociado:** Seguridad de Credenciales.
- **Responsabilidad Principal:** Validar la fortaleza de una contraseña contra un banco de datos de contraseñas conocidas y comprometidas (ej. Have I Been Pwned).
- **Justificación:** Este es un requerimiento de seguridad crítico. Aislar esta funcionalidad en un servicio dedicado permite actualizar y mantener la lógica de validación y la base de datos de contraseñas comprometidas de forma independiente, sin afectar al servicio de identidad.
- **Datos que Gestiona:**
  - No gestiona datos de usuario. Mantiene o consulta una base de datos (posiblemente externa) de hashes de contraseñas comprometidas.
- **API Sincrónica (REST)**

| Endpoint | Método | Descripción | Request Body (Schema) |
|---|---|---|---|
| `/passwords/is-secure` | `POST` | **Endpoint interno.** Verifica si una contraseña es segura y no está comprometida. | `{ "password": "..." }` |

---

## 3. Flujos de Comunicación Detallados

A continuación, se describen flujos de ejemplo que ilustran la comunicación entre servicios, incluyendo la topología de RabbitMQ.

### Flujo 1: Registro de un Nuevo Usuario

1. **Cliente** -> `AuthIdentity` (Sincrónico, REST): El usuario envía `{ "username", "email", "password" }` al endpoint `POST /register`.
2. `AuthIdentity` -> `PasswordGuardian` (Sincrónico, REST): `AuthIdentity` llama internamente a `POST /passwords/is-secure` con la contraseña del usuario.
3. `PasswordGuardian`: Verifica la contraseña contra su base de datos.
   - **Si no es segura:** Devuelve un error `400 Bad Request`. `AuthIdentity` propaga el error al cliente. **El flujo termina.**
   - **Si es segura:** Devuelve una respuesta `200 OK`.
4. `AuthIdentity`: Procede a validar el resto de los datos (ej. formato de email, unicidad de username). Si todo es correcto, hashea la contraseña y crea el `User` en su base de datos

PostgreSQL.

5. `AuthIdentity` -> **RabbitMQ** (Asincrónico): Publica un evento `UserRegistered` con el payload `{ "user_id", "username", "email" }` en el exchange `identity_events` (tipo `fanout`).
6. **RabbitMQ** -> **Consumidores**: El exchange `fanout` duplica el mensaje y lo enruta a las colas de todos los servicios suscritos.
    - `SocialConnect`: Recibe el evento y ejecuta `CREATE (u:User {userId: event.user_id, username: event.username})` en su base de datos Neo4j.
    - `RiskMitigation`: Recibe el evento y ejecuta `INSERT INTO RiskProfiles (user_id, risk_score) VALUES (event.user_id, 0.0)`.

## Flujo 2: Sugerencia de un Plan de Salud

1. `SocialConnect` -> **RabbitMQ** (Asincrónico): Un usuario publica un post. El servicio publica un evento `PostCreated` en el exchange `social_events` (tipo `topic`) con la routing key `social.post.created`.
2. `RiskMitigation` (Consumidor): Su cola está vinculada al exchange `social_events` con el patrón de binding `social.#`. Recibe el evento y lo ingiere en su Data Lake.
3. `RiskMitigation` (Proceso Interno): Periódicamente, un job de análisis procesa los datos acumulados, recalcula el `risk_score` de un usuario y determina que se necesita una intervención.
4. `RiskMitigation` -> `ResourcesDocs` (Sincrónico, REST): El job determina que el recurso `res-123` es adecuado. Llama a `GET /resources/res-123` para obtener su título: `"Guía para una vida digital saludable"`.
5. `RiskMitigation` -> **RabbitMQ** (Asincrónico): Publica un evento `InterventionRequired` en el exchange `interventions_exchange` (tipo `direct`) con la routing key `notification.suggest`. El payload es `{ "user_id": "...", "resource_id": "res-123" }`.
6. `NotificationService` (Consumidor): Su cola está vinculada con la routing key `notification.suggest`. Recibe el evento. Opcionalmente, puede llamar a `AuthIdentity` o `SocialConnect` para obtener el nombre del usuario y el token del dispositivo. Finalmente, construye y envía la notificación push: *"Hola @username, creemos que este recurso podría interesarte: Guía para una vida digital saludable"*.

Este flujo garantiza que `SocialConnect` y el usuario final nunca conozcan la lógica de riesgo que originó la sugerencia, protegiendo así la privacidad.