UNIVERSIDAD POLITÉCNICA DE MADRID

E.T.S. DE INGENIERÍA DE SISTEMAS INFORMÁTICOS

PROYECTO FIN DE GRADO

GRADO EN INGENIERÍA DEL SOFTWARE

PANOT: Plataforma Móvil para la Gestión de la Inteligencia Relacional mediante Captura Asistida de Interacciones

Desarrollado por: Ángel Rodríguez Morán

Dirigido por: Elvira Amador Domínguez

Madrid, 13 de enero de 2026

PANOT: Plataforma Móvil para la Gestión de la Inteligencia Relacional mediante Captura Asistida de Interacciones
Desarrollado por: Ángel Rodríguez Morán
Dirigido por: Elvira Amador Domínguez
Proyecto Fin de Grado, 13 de enero de 2026

E.T.S. de Ingeniería de Sistemas Informáticos
Campus Sur UPM, Carretera de Valencia (A-3), km. 7
28031, Madrid, España

Si deseas citar este trabajo, la entrada completa en BibTeX es la siguiente:

```
@mastersthesis{2026ngelRodrguezMorn,
    title = {PANOT: Plataforma Móvil para la Gestión de la Inteligencia Relacional
mediante Captura Asistida de Interacciones},
    type = {Bachelor's Thesis},
    author = {},
    school = {E.T.S. de Ingeniería de Sistemas Informáticos},
    year = {2026},
    month = {1},
}
```

Esta obra está bajo una licencia Creative Commons «Atribución-NoComercial-CompartirIgual 4.0 Internacional». Obra derivada de https://github.com/blazaid/UPM-Report-Template.

Todo cambio respecto a la obra original es responsabilidad exclusiva del presente autor.

[Cita opcional para el proyecto]

— [Autor de la cita]

Agradecimientos

[Escribir aquí los agradecimientos del proyecto]

Resumen

mas a delante...

Palabras clave:

Abstract

[Write here the project summary in English]

Keywords:

Índice general

1 Introducción 1

1.1 Motivación 1

1.2 Descripción y Alcance del Proyecto 1

1.3 Objetivos 1

1.4 Estructura de la memoria 1

2 Estado de la Técnica y Contexto Tecnológicos 2

2.1 Gestión de la Inteligencia Relacional 2

2.2 Cambio de Paradigma en el Diseño de Producto 8

2.3 Principios de Privacidad y Eficiencia por Diseño 9

2.4 Contexto Tecnológico 15

3 Desarrollo del Proyecto 24

3.1 Filosofía y Metodología de Desarrollo 24

3.2 Definición del PMV y Especificación de Requisitos 27

3.3 Diseño del Sistema 32

3.4 Fases de la Implementación Iterativa 45

3.5 Despliegue y Lanzamiento 45

4 Verificación y Resultados 46

5 Conclusiones y Trabajo Futuro 47

5.1 Conclusiones Generales 47

5.2 Aplicación de Conocimientos Adquiridos en el Grado 47

5.3 Líneas de Trabajo Futuro 47

Glosario 49

Siglas 51

Índice de figuras

2.1 Representación orientativa en MemGraph Lab del contexto del contacto «María» en formato grafo relacional. 6

3.1 Diagrama orientativo de la estrategia de ramificación. (1) Aislamiento: Creación de una rama efímera (*feature*) a partir de desarrollo para trabajar en un *Issue* específico. (2) Implementación: Desarrollo de la funcionalidad mediante *commits* atómicos. (3) Integración: Fusión de la rama mediante un *Pull Request* validado, cerrando el ciclo de desarrollo. (4) Release: Promoción del código estable desde desarrollo a la rama principal (*main*) para su despliegue en producción. 26

3.2 Visualización de las columnas del tablero Kanban implementado en GitHub Projects en un punto del desarrollo. Se destaca el límite de WIP configurado a 1 en la columna *In Progress* (indicador 1/1), forzando un flujo continuo y eliminando el desperdicio asociado al cambio de contexto. . 27

3.3 Arquitectura general del sistema PANOT 32

3.4 Diagrama de la arquitecturadel sistema multi-agente de PANOT 36

3.5 Diagrama del modelo de datos de PANOT del schema de Supabase «*auth*». Solo se incluye la tabla *auth.users* ya que es la unica tabla de el schema «*auth*» que se ha modificado. 39

3.6 Diagrama del modelo de datos de PANOT del schema de Supabase «*public*». 40

3.7 Ejemplo de grafo semántico en PostgreSQL. En el se pueden ver dos nodos: un primer nodo A de tipo CONTACT y un segundo nodo B de tipo CONCEPT. Ambos estan conectados con una relación de tipo RELACIONADO_CON. 42

3.8 Diagrama de actividad del sistema de procesamiento de PANOT 44

3.9 Diagrama de secuencia del procesamiento de una Interacción grabada por el usuario 45

Índice de tablas

3.1 Features del Sistema y los IDs de los Requisitos Funcionales Asociados . 29

3.2 Requisitos Funcionales de la *Feature 1 - Gestión de Usuario* 29

3.3 Requisitos Funcionales de la *Feature 2 - Gestión de Contactos* 29

3.4 Requisitos Funcionales de la *Feature 3 - Gestión de Interacciones* 30

3.5 Requisitos Funcionales de la *Feature 4 - Inteligencia Relacional* 30

3.6 Requisitos Funcionales de la *Feature 5 - Soporte y Feedback* 30

3.7 Requisitos No Funcionales del sistema con ID, Tipo, Descripción y Criterio de Aceptación 31

1. Introducción

1.1. Motivación

[Escribir aquí la motivación del proyecto]

1.2. Descripción y Alcance del Proyecto

[Escribir aquí la descripción y alcance del proyecto]

1.3. Objetivos

[Escribir aquí los objetivos del proyecto]

1.4. Estructura de la memoria

[Escribir aquí la estructura de la memoria]

2. Estado de la Técnica y Contexto Tecnológicos

2.1. Gestión de la Inteligencia Relacional

Los *Sistemas de Gestión de Relaciones con Clientes* (CRM) convencionales, si bien útiles en entornos corporativos para la gestión masiva de clientes, presentan limitaciones significativas cuando se trata de capturar la complejidad inherente a las relaciones humanas. Estos sistemas operan principalmente con información descontextualizada, almacenando datos de contacto de manera estática y registrando interacciones sin considerar su evolución temporal ni el contexto en el que ocurren.

2.1.1. Aplicaciones Similares

El paradigma actual de la gestión de contactos ha evolucionado desde simples agentas digitales, a lo que ahora se denomina *Gestión de Relaciones Personales* (PRM). Sin embargo, al analizar las soluciones líderes en el mercado, se han identificado patrones de diseño que limitan la capacidad de capturar la esencia de las relaciones humanas.

Aplicaciones de código abierto como *Monica* o soluciones más comerciales como *Dex* o *Clay* operan bajo la misma premisa de transformar un contacto en una entidad más compleja y dinámica, no obstante, transfieren toda la carga cognitiva y administrativa al usuario, lo que inyecta cierta fricción a la hora de capturar información.

Algunas de estas herramientas introducen el uso de modelos de lenguaje para enriquecer automáticamente los perfiles, extrayendo datos de redes sociales y correos electrónicos. Esto supone una propuesta de valor muy fuerte, pero, aunque reduzcan la fricción de la inserción de datos, su enfoque es fundamentalmente 'archivístico'. Al operar sobre datos públicos y huellas digitales pasadas, carecen de un contexto situacional profundo, es decir, saben donde trabaja un contacto, pero no saben cómo se sienten respecto a su nuevo empleo o qué matíz emocional tuvo la última conversa-

ción.

El vacío que se considera que estas aplicaciones no logran cubrir radica en la disonancia entre la naturaleza de las relaciones humanas y la arquitectura de las bases de datos que las contienen. Es decir, PANOT busca funcionar como una extensión cognitiva, y no una agenda perfecta. Por lo que la hipótesis de valor que enmarca PANOT, se posicionaría en ese *registro del entendimiento*.

2.1.2. Inteligencia Relacional en el Contexto de la Inteligencia Artificial

La Inteligencia Relacional presenta un nuevo paradigma evolutivo en la gestión de información personal y profesional, introduciendo el concepto de dinamismo contextual, donde cada relación evoluciona continuamente reflejando cambios en intereses, preferencias y circunstancias vitales. Este enfoque reconoce que las relaciones que tenemos no son entidades fijas, sino procesos complejos que cambian según un contexto temporal y situacional.

Para poder comprender el alcance de esta premisa, es necesario enmarcar el concepto dentro del ecosistema más amplio de la Inteligencia Artificial y analizar cómo se diferencia con los paradigmas tradicionales.

La diferencia fundamental entre la Inteligencia Relacional y los paradigmas tradicionales de Inteligencia Artificial (IA) radica en que, mientras estos últimos operan principalmente mediante la aproximación estadística de distribuciones de datos y patrones —optimizando funciones de pérdida sobre grandes volúmenes de datos descontextualizados—, la Inteligencia Relacional se fundamenta en la construcción y manipulación de representaciones estructurales de relaciones que permiten la generalización cruzada y la Inferencia Analógica. La Inteligencia Relacional captura la estructura relacional subyacente que puede transferirse entre dominios aparentemente no relacionados, tal como ocurre en el razonamiento humano, creando un contexto de conocimiento más amplio y generalizable o especfíco para un dominio en concreto.

La investigación en Inteligencia Relacional demuestra capacidades que van más allá del aprendizaje estadístico tradicional. En 2022, se publicó un estudio de la Universidad de Edimburgo[1] en el que se muestra cómo un modelo computacional puede aprender representaciones relacionales estructuradas y realizar Generalización de Ce-

ro Disparos entre dominios completamente diferentes, como la transferencia de conocimiento entre videojuegos. Esta capacidad de generalización permite que el sistema aprenda a reconocer y comprender relaciones entre entidades en contextos completamente diferentes.

Traspasando la analogía de los videojuegos al contexto de PANOT, la Inteligencia Relacional como se menciona en [1] permite que el sistema aprenda a reconocer y comprender patrones relacionales estructurados entre personas, eventos y contextos, más allá de las asociaciones estadísticas superficiales. En lugar de simplemente almacenar datos estáticos de contactos, el sistema de PANOT puede construir representaciones relacionales dinámicas que capturan la estructura subyacente de las relaciones humanas —como la evolución temporal de intereses comunes, la frecuencia contextual de interacciones, o los cambios en preferencias y circunstancias vitales—.

Para ilustrar este proceso, consideremos un ejemplo práctico del flujo de procesamiento relacional en PANOT:

Input: El usuario captura una interacción mediante nota de voz: "Acabo de almorzar con María. Está muy emocionada porque ha conseguido un nuevo trabajo como diseñadora en una startup tecnológica. Le interesa especialmente el trabajo remoto y mencionó que está buscando un piso más cerca de su nueva oficina. Hablamos de proyectos de diseño colaborativo y se mostró muy receptiva a la idea de futuros proyectos juntos."

Procesamiento: PANOT procesa esta entrada multimodal extrayendo múltiples capas de información relacional estructurada:

- *Evento*: almuerzo social de contexto informal
- *Cambio de estado*: transición profesional — nuevo trabajo como diseñadora
- *Cambio de preferencias*: prioridad hacia trabajo remoto
- *Necesidad emergente*: búsqueda de vivienda
- *Relaciones*: interés común en proyectos de diseño colaborativo, receptividad a futura colaboración
- *Contexto temporal*: estado emocional positivo, momento de transición vital

El sistema construye una representación relacional estructurada que conecta estas entidades (usuario-contacto) mediante relaciones semánticas. A modo ilustrativo, esta sería la representación en formato `JSON`:

```
{
  "personal": {
    "necesidad_actual": "búsqueda de vivienda",
    "preferencia_ubicación": "cerca de la nueva oficina",
    "estado_emocional": "emocionada"
  },
  "profesional": {
    "rol": "diseñadora",
    "empresa": "startup tecnológica",
    "intereses": ["trabajo remoto"],
    "estado": "transición laboral reciente"
  },
  "relación": {
    "última_interacción": "almuerzo informal",
    "intereses_comunes": ["diseño colaborativo", "proyectos futuros"],
    "dinámica": "receptiva a colaboración"
  }
}
```

Output: PANOT actualiza dinámicamente el contexto de María, integrando la nueva información estructurada directamente en su perfil relacional:

- *Contexto Personal*: Se actualiza el estado vital reflejando la necesidad activa de "búsqueda de vivienda" y su preferencia de ubicación, además de capturar el estado emocional positivo asociado al cambio.
- *Contexto Profesional*: Se modifica la información laboral para incluir el nuevo rol de "diseñadora en startup tecnológica" y se añade el "trabajo remoto" como un interés profesional clave en esta etapa.
- *Contexto de la Relación*: Se enriquece la dinámica de la relación registrando los "proyectos de diseño colaborativo" como un nuevo eje de interés común y actualizando el estado de la relación hacia una posible colaboración futura.

Así, el contacto de María dentro de la base de datos de PANOT quedaría como un conjunto de nodos interconectados que representan eventos, gustos, situaciones, necesidades, etc. —Ver 2.1— abstrayéndo el complejo contexto de la relación en una representación más simplificada.

Figura 2.1. Representación orientativa en MemGraph Lab del contexto del contacto «María» en formato grafo relacional.

2.1.3. Arquitecturas Similares: Grafos Contextuales en Sistemas de Agentes

La representación relacional estructurada que utiliza PANOT encuentra un paralelismo arquitectónico significativo con sistemas de agentes de inteligencia artificial que emplean grafos contextuales como mecanismo de memoria persistente. Estos sistemas, inspirados en arquitecturas como GraphRAG (Graph Retrieval-Augmented Generation), implementan grafos de conocimiento que permiten a los agentes acceder de manera eficiente a información estructurada y realizar razonamientos complejos mediante la navegación de relaciones semánticas.

En arquitecturas de agentes modernas, el grafo contextual actúa como una memoria estructurada que almacena información sobre interacciones pasadas, estados del

entorno y relaciones entre diferentes entidades. Esta estructura permite a los agentes:

- *Almacenar información de forma estructurada*: Capturar y organizar datos sobre estados, eventos y relaciones de manera que preserve la estructura relacional subyacente, en lugar de almacenar información de forma descontextualizada.
- *Acceder rápidamente a información relevante*: Navegar eficientemente por el grafo para recuperar datos pertinentes según la situación actual, mejorando significativamente la eficiencia en la recuperación de información comparado con búsquedas en bases de datos relacionales tradicionales.
- *Facilitar el razonamiento complejo*: Utilizar la estructura del grafo para inferir nuevas relaciones y tomar decisiones informadas mediante razonamiento multihop —la capacidad de realizar inferencias a través de múltiples pasos siguiendo las conexiones del grafo—.
- *Adaptabilidad y aprendizaje continuo*: Actualizar y expandir el conocimiento de manera dinámica, incorporando nuevas interacciones y relaciones sin requerir reestructuración completa de la base de datos.

La eficiencia superior de las estructuras de grafo para la memoria de los agentes radica en la diferencia fundamental entre el modelo de acceso a datos relacionales y el modelo de navegación por grafos. En las bases de datos relacionales, recuperar información sobre relaciones entre entidades requiere realizar múltiples operaciones que pueden cruzar tablas diferentes, lo cual implica escanear índices y realizar comparaciones entre grandes volúmenes de datos. La complejidad de estas operaciones crece exponencialmente con el número de relaciones involucradas, resultando en tiempos de consulta que pueden ser $\mathcal{O}(n \log n)$ o peor cuando se requieren múltiples uniones de tablas anidadas.

Por el contrario, en una estructura de grafo, acceder a los vecinos directos de un nodo —es decir, recuperar todas las relaciones de una entidad— es una operación de complejidad $\mathcal{O}(1)$ en promedio, ya que las conexiones están almacenadas directamente como parte de la estructura del nodo mediante listas de adyacencia o estructuras similares. Esta diferencia es crítica para sistemas de agentes que requieren acceso frecuente y rápido a información relacional.

Como ejemplo, si queremos que nuestro sistema haga una consulta relacional para recuperar “todos los eventos relacionados con María y sus intereses comunes” podría requerir de múltiples uniones de las tablas de contactos, eventos, intereses y relaciones

en el caso en el que se trate de un sistema de relacional. Por el contrario, en un grafo contextual, esta misma información se obtiene mediante una simple navegación a través de las aristas conectadas al nodo de María, accediendo directamente a los nodos adyacentes sin necesidad de realizar búsquedas complejas.

En el contexto de PANOT, el grafo relacional que representa cada contacto y sus interacciones funciona análogamente al grafo contextual de un sistema agéntico: ambos proporcionan una estructura que permite acceso eficiente a información relevante, razonamiento sobre relaciones complejas y actualización dinámica del conocimiento. Esta arquitectura permite que PANOT no solo almacene información sobre contactos, sino que también pueda realizar inferencias relacionales, generalizar patrones entre relaciones y adaptarse continuamente a la evolución de las interacciones humanas.

2.2. Cambio de Paradigma en el Diseño de Producto

Un paradigma de diseño de producto constituye un conjunto de principios fundamentales, patrones de interacción y enfoques conceptuales que guían la creación de experiencias de usuario en productos y servicios digitales o analógicos. Representa más que una simple metodología de diseño; es una filosofía que establece cómo los usuarios perciben, interactúan y se relacionan con una aplicación. Fijese que este apartado no está orientado en principios de diseño de la arquitectura del software, sino en principios de diseño de producto que van más allá del desarrollo de la aplicación y están orientados en la experiencia del usuario.

En el contexto de las aplicaciones móviles, la relevancia de los paradigmas de diseño de producto adquiere una dimensión crítica debido a las características inherentes de estos dispositivos: limitaciones de espacio en pantalla, interacciones predominantemente táctiles y expectativas de inmediatez y estímulo por parte de los usuarios. Un paradigma de diseño adecuado no solo determina la usabilidad de una aplicación, sino que establece la base sobre la cual se construyen las expectativas del usuario, su curva de aprendizaje y, fundamentalmente, su conexión emocional con el producto.

La irrupción de la Inteligencia Artificial como tecnología dominante ha transformado radicalmente el panorama del diseño de productos digitales. La democratización de las capacidades de IA ha generado un desplazamiento del valor diferencial de los productos: ya no es suficiente ofrecer una funcionalidad única o una interfaz atractiva,

pues estas características pueden replicarse rápidamente. En este nuevo contexto, la diferenciación competitiva se desplaza hacia dimensiones más profundas y fundamentales de la experiencia humana.

2.2.1. Valores Diferenciadores en la Era de la IA

En busca de la diferenciación y lealtad a largo plazo de estos productos, se deben incorporar valores fundamentales más allá de la funcionalidad técnica. Para construir un producto que se diferencie, es esencial y crítico en la era en la vivimos generar valor desde flancos más profundos y fundamentales de la experiencia humana. Como punto de partida, el producto ha de centrarse en en los siguientes dos principios:

- *Conexión y resonancia emocional*: El producto debe generar una conexión emocional genuina con los usuarios, comprendiendo su contexto y acompañando la evolución de sus necesidades, para establecer vínculos sostenibles que trasciendan la interacción funcional.
- *Personalización adaptativa y comprensión contextual*: El sistema debe de tener la capacidad de aprender activamente de las interacciones con el usuario, infiriendo patrones, intereses y necesidades implícitas sin requerir configuraciones explícitas, y adaptando la experiencia de manera proactiva y sin fricción, asegurando una experiencia personalizada continua.

Los usuarios no solo buscan que una aplicación funcione bien; buscan que se *adapte* a ellos, que *comprenda* su contexto, que *evolucione* con sus necesidades y que establezca una conexión que trascienda la mera transacción funcional.

2.3. Principios de Privacidad y Eficiencia por Diseño

El desarrollo de sistemas de software modernos, especialmente aquellos que procesan información personal y utilizan recursos computacionales intensivos como modelos de lenguaje, requieren de la integración de principios fundamentales desde las primeras etapas del diseño. En este apartado se presentan dos marcos conceptuales esenciales: la *Privacidad desde el Diseño*, enmarcada en el contexto normativo español,

y la *Eficiencia por Diseño*, fundamentada en las mejores prácticas de arquitectura de software modernas.

2.3.1. Principio de Privacidad

La *Privacidad desde el Diseño* (Privacy by Design, PbD) constituye un enfoque proactivo que integra la protección de datos personales desde las primeras fases de desarrollo de productos, servicios o procesos. Este concepto, desarrollado inicialmente por Ann Cavoukian en la década de los 90 y reconocido internacionalmente en la 32ª Conferencia Internacional de Comisionados de Protección de Datos y Privacidad celebrada en Jerusalén en 2010, ha sido consolidado legalmente en el Reglamento General de Protección de Datos (RGPD) mediante su artículo 25, que establece la obligación de implementar medidas técnicas y organizativas apropiadas para garantizar la protección de datos desde el diseño y por defecto.

La Agencia Española de Protección de Datos (AEPD), en su *Guía de Privacidad desde el Diseño* [2], identifica siete principios fundacionales que deben guiar el desarrollo de sistemas que procesan datos personales:

1. *Proactivo, no reactivo; preventivo, no correctivo*: La privacidad debe anticiparse a los riesgos antes de que se materialicen, implementando medidas preventivas en lugar de soluciones correctivas posteriores.
2. *La privacidad como configuración predeterminada*: Los sistemas deben proteger los datos personales por defecto, sin requerir acción adicional del usuario. La configuración más privada debe ser la opción predeterminada.
3. *Privacidad incorporada en la fase de diseño*: Las medidas de protección deben integrarse desde el inicio del desarrollo, evitando soluciones añadidas posteriormente que puedan ser menos efectivas o generar fricciones en la experiencia del usuario.
4. *Funcionalidad total: pensamiento "todos ganan"*: Los objetivos de privacidad no deben comprometer la funcionalidad del sistema. El diseño debe equilibrar ambos aspectos, garantizando que la protección de datos y las funcionalidades esenciales coexistan sin sacrificios mutuos.
5. *Aseguramiento de la privacidad en todo el ciclo de vida*: La protección de datos debe mantenerse durante todas las etapas del ciclo de vida del sistema, desde su

concepción hasta su retirada, incluyendo desarrollo, puesta en producción, operación, mantenimiento y eliminación.

6. *Visibilidad y transparencia*: Los usuarios deben tener información clara y accesible sobre las prácticas de tratamiento de datos, fomentando la confianza mediante la transparencia en los procesos.

7. *Respeto por la privacidad de los usuarios: mantener un enfoque centrado en el usuario*: El diseño debe respetar las preferencias y necesidades de privacidad de los usuarios, otorgándoles control efectivo sobre sus datos personales y manteniendo un enfoque que priorice sus derechos y expectativas.

La implementación de estos principios requiere un enfoque de *Privacy Engineering* o ingeniería de la privacidad, que traduce los principios conceptuales en medidas técnicas concretas durante las diversas fases de desarrollo. La guía de la AEPD identifica estrategias de diseño específicas, entre las que destacan: *minimizar* la recopilación de datos a lo estrictamente necesario, *ocultar* información sensible mediante técnicas de ofuscación o desvinculación, *separar* datos para evitar correlaciones indebidas o perfilados del usuario, *abstraer* detalles personales limitando su detalle, *informar* a los usuarios sobre el tratamiento de sus datos, *controlar* el acceso y uso de la información, *cumplir* con qué procedimiento se están llevando a cabo con sus datos, *controlar* en sentido de otorgar control a los usuarios en relación a la recogida, tratamiento, usos y comunicaciones realizados sobre sus datos personales, *cumplir* asegurando que los tratamientos personales son compatibles y respetan los requisitos y obligaciones legales impuestos por la normativa, y por último *demostrar*, según el artículo 24 de la LGPD, el cumplimiento normativo mediante documentación y auditorías, tanto a los usuarios como a las autoridades de supervisión asociadas.

2.3.2. Principio de Eficiencia

La *Eficiencia por Diseño* (Efficiency by Design) constituye un enfoque arquitectónico que integra la optimización de recursos computacionales desde las primeras etapas del desarrollo, garantizando que los sistemas puedan gestionar su carga de trabajo utilizando la menor cantidad de recursos posibles sin comprometer la funcionalidad ni la experiencia de usuario. A diferencia de la optimización reactiva, que se aplica una vez que el sistema ya está en producción, la eficiencia por diseño requiere considerar aspectos de rendimiento, escalabilidad y consumo de recursos desde las fases de análisis y diseño arquitectónico.

En su libro *Clean Architecture*, Robert C. Martin [3] aborda esta cuestión desde una perspectiva más fundacional, enfatizando la importancia de promover la eficiencia a largo plazo mediante el diseño de sistemas flexibles y mantenibles, a través de la separación de la lógica de negocio y los detalles técnicos. Esto se consigue con un diseño de capas donde las dependencias siempre apuntan hacia el interior, de modo que los cambios en aspectos técnicos, es decir, en capas más centrales, no impacten en las capas superiores de principios y reglas del sistema. De esta manera, se favorece la creación de sistemas eficientes y sostenibles, capaces de mantener su funcionalidad y adaptabilidad en el tiempo.

En la práctica, las empresas de software modernas implementan el principio de eficiencia por diseño mediante diversas estrategias arquitectónicas. Por ejemplo, la adopción de arquitecturas serverless y funciones como servicio (FaaS) permite a empresas como Netflix optimizar el consumo de recursos computacionales, pagando únicamente por el tiempo de ejecución real en lugar de mantener servidores activos de forma continua. Otras empresas, como Spotify, han implementado arquitecturas de microservicios con auto-escalado horizontal, permitiendo que la infraestructura se adapte dinámicamente a la carga de trabajo sin sobre-provisionamiento de recursos. Asimismo, la implementación de estrategias de caching distribuido y CDN (Content Delivery Networks) en empresas como Amazon Web Services permite reducir la latencia y el consumo de ancho de banda mediante el almacenamiento de contenido frecuentemente accedido en ubicaciones geográficamente cercanas a los usuarios finales.

2.3.3. Aplicación en Sistemas con Modelos de Lenguaje

La aplicación de los principios de privacidad y eficiencia por diseño adquiere particularidades específicas cuando se trata de sistemas que integran modelos de lenguaje (LLMs), debido a la naturaleza sensible de los datos procesados y a los elevados costes computacionales asociados con el procesamiento de lenguaje natural.

En el contexto de aplicaciones que utilizan modelos de lenguaje, la protección de la privacidad presenta desafíos únicos derivados de la necesidad de procesar información personal, a menudo sensible, mediante servicios externos o infraestructura en la nube. Para abordar estos desafíos, se han desarrollado diversas *tecnologías de preservación de la privacidad* (TPP) que permiten procesar datos manteniendo su confidencialidad.

Una de las técnicas más prometedoras en este ámbito es el *cifrado totalmente homomórfico* (Fully Homomorphic Encryption, FHE), un paradigma criptográfico que permite realizar operaciones computacionales sobre datos cifrados sin necesidad de descifrarlos previamente. A diferencia del cifrado tradicional, donde los datos deben descifrarse antes de ser procesados, el cifrado homomórfico permite que un servidor o servicio externo procese información cifrada y devuelva resultados también cifrados, manteniendo la confidencialidad de los datos en todo momento. Este enfoque es especialmente relevante en sistemas que utilizan modelos de lenguaje en entornos como el sanitario [4], ya que permitiría enviar datos sensibles cifrados sobre pacientes a servicios de procesamiento de lenguaje natural sin que el proveedor del servicio pueda acceder al contenido real de la información.

Sin embargo, el cifrado totalmente homomórfico presenta limitaciones prácticas significativas en términos de rendimiento y eficiencia computacional ya que este tipo de operaciones requieren de modelos pre-entrenados con datos homomórficamente cifrados. Por esta razón, su aplicación en sistemas desplegados sigue siendo un área de investigación activa, aunque existen implementaciones experimentales que demuestran su viabilidad para casos de uso específicos.

Además del cifrado homomórfico, otras estrategias de preservación de la privacidad en sistemas con modelos de lenguaje incluyen la *minimización de datos*, enviando únicamente la información estrictamente necesaria para el procesamiento, la *ofuscación* de información sensible mediante técnicas de anonimización o pseudonimización, y el *procesamiento local* cuando es posible, utilizando modelos de lenguaje más pequeños ejecutados directamente en el dispositivo del usuario para evitar la transmisión de datos a servicios externos.

En el caso de la eficiencia en este tipo de sistemas, una estrategia arquitectónica fundamental es la adopción de *arquitecturas serverless* y *Funciones como Servicio* (FaaS), que permiten ejecutar procesamiento de modelos de lenguaje de manera escalable y bajo demanda. Los servicios serverless eliminan la necesidad de mantener servidores activos de forma continua, permitiendo que la infraestructura se active únicamente cuando se requiere procesamiento, reduciendo significativamente los costes operativos. Plataformas como AWS Lambda, Google Cloud Functions o Azure Functions permiten ejecutar funciones que invocan modelos de lenguaje, pagando únicamente por el tiempo de ejecución real y los recursos computacionales consumidos durante cada invocación.

Además de la arquitectura serverless, la *elección del modelo de lenguaje adecuado*

constituye un factor crítico para optimizar la eficiencia en sistemas que procesan lenguaje natural. La selección del modelo debe equilibrar tres dimensiones fundamentales: la *eficiencia computacional*, el *coste económico* y el *tiempo de respuesta*. Modelos de lenguaje más grandes y complejos, como GPT-5 Codex de OpenAI o Claude 4.5 Sonnet de Anthropic, ofrecen capacidades superiores de comprensión y generación de lenguaje, pero conllevan un mayor coste computacional. Por el contrario, modelos más pequeños y optimizados, como Gemma 3n E4B de Google o Ministral 3B de Mistral, aunque suponen un coste computacional menor, no ofrecen las mismas capacidades que los primeros.

[insertar tabla de costes de de los modelos mencionados en coste por nº de tokens de input]

La optimización de la eficiencia en sistemas con modelos de lenguaje requiere, por tanto, un análisis cuidadoso de los requisitos funcionales y no funcionales de la aplicación, seleccionando el modelo que mejor equilibre la calidad de las respuestas, el tiempo de respuesta y un coste económico sostenible.

2.3.4. Aplicación en PANOT

El caso del servicio que se ha desarrollado en este proyecto, se ha planteado un enfoque que respeta y aplica los principios de privacidad y eficiencia desde el diseño del sistema.

El principio de privacidad en un sistema como PANOT, adquiere una relevancia especial ya que se manejan datos sobre relaciones interpersonales, preferencias, contextos situacionales y evolución temporal de los contactos del usuario. Sin embargo, la aplicación de la privacidad desde el diseño en PANOT presenta particularidades específicas derivadas de su naturaleza como herramienta personal de gestión de contactos.

PANOT está diseñado como una aplicación personal donde el usuario gestiona y evoluciona el perfil de sus propios contactos. En este contexto, la privacidad no requiere ocultar información al usuario propietario y governador del dato, sino protegerlos frente a accesos no autorizados y cumplir con las restricciones legales sobre datos especialmente sensibles dentro de lo que es posible.

A nivel de infraestructura, se ha utilizado *Supabase* como plataforma de backend como ya se explicará en puntos posteriores. *Supabase* tiene la particularidad de auto-

gestionar estas medidas de seguridad a través de dos mecanismos: Por un lado, todos los datos se encriptan en reposo mediante AES-256 y en tránsito mediante el protocolo Transport Layer Security, garantizando la protección de la información en todas las etapas del ciclo de vida. Y por otro lado, *Supabase* proporciona Row Level Security y Column Level Security, permitiendo un control granular del acceso a los datos a nivel de fila y columna, lo que asegura que cada usuario solo pueda acceder a sus propios datos y a nada más. *Supabase* además cumple con estándares de seguridad como SOC 2 Type 2, proporcionando una base de cumplimiento normativo adicional.

La aplicación de los principios de eficiencia por diseño en PANOT se ha materializado mediante la adopción de una metodología de desarrollo inspirada en el enfoque *Lean Startup* propuesto por Eric Ries [5]. Este marco metodológico, fundamentado en el ciclo iterativo Construir-Medir-Aprender, establece que el desarrollo de productos debe priorizar la entrega de valor con el menor coste posible, en el menor tiempo posible, manteniendo la máxima calidad en las funcionalidades principales —aunque esta filosofía, que guiará todo el desarrollo, se explicará más en detalle en los capítulos siguientes—.

En este contexto, durante el diseño arquitectónico de PANOT, se han orientando las decisiones técnicas hacia la construcción de una infraestructura escalable y eficiente desde sus fundamentos. Este enfoque ha permitido priorizar la funcionalidad esencial del sistema —la gestión inteligente de relaciones interpersonales mediante modelos de lenguaje— mientras se optimiza el consumo de recursos computacionales y económicos asociados con el alojamiento de la infraestructura y el uso de servicios externos.

2.4. Contexto Tecnológico

La elección de tecnologías para el desarrollo del proyecto se ha realizado teniendo en cuenta los criterios de mantenibilidad, seguridad, escalabilidad, rapidez de implementación y coste. Tras un análisis comparativo entre múltiples opciones disponibles, se ha optado por una arquitectura cliente-servidor con aplicación móvil desarrollada con el framework *Expo* React Native, un backend entermente gestionado por *Supabase*, con PostgreSQL como motor de bases de datos, *Stripe* para la integración con pasarelas de pago y *PostHog* para el análisis de métricas de éxito y seguimiento de uso de funcionalidades.

2.4.1. Capa del Cliente Móvil

Para el desarrollo de la aplicación móvil PANOT se ha seleccionado *Expo*, un framework construido sobre *React Native* que simplifica significativamente el proceso de desarrollo de aplicaciones multiplataforma. Esta elección se fundamenta en varios factores clave:

- Rapidez de desarrollo: *Expo* proporciona un conjunto de herramientas e Interfaces de Programación de Aplicaciones (APIs) preconfiguradas que reducen la complejidad de configuración del proyecto y aceleran el tiempo de desarrollo, permitiendo enfocar los esfuerzos en la implementación de funcionalidades de valor. Además *Expo* es un framework basado en *React* y en el caso de PANOT, *Typescript* lo que facilita el desarrollo modular, la reutilización de componentes y el tipado de los artefactos del proyecto debido a la naturaleza de estos lenguajes.
- Compatibilidad multiplataforma: Aunque el proyecto se centra inicialmente en iOS, *Expo* facilita la extensión futura a Android con mínimos cambios en el código base, garantizando una base sólida para el crecimiento del producto.
- Ecosistema Open Source: *Expo* cuenta con una comunidad activa y documentación muy completa, lo que facilita el aprendizaje y la resolución de problemas.
- Acceso a funcionalidades nativas: A través de módulos nativos y APIs expuestas por *Expo*, se mantiene acceso a capacidades del dispositivo como notificaciones push, almacenamiento local y sensores, sin requerir la complejidad del desarrollo nativo puro.

Se analizaron las siguientes alternativas tecnológicas para el desarrollo del cliente móvil:

- Desarrollo nativo con *Swift* y *SwiftUI*: Esta opción habría proporcionado un rendimiento óptimo y acceso completo a todas las capacidades nativas de iOS. Sin embargo, fue descartada debido a su mayor complejidad de configuración, un tiempo de desarrollo más extenso y la limitación a una única plataforma, aspectos que no se alinean con los objetivos de eficiencia y escalabilidad del proyecto.
- Desarrollo multiplataforma con *Flutter*: También se consideró el uso de *Flutter*, un framework desarrollado por Google que permite crear aplicaciones móviles

utilizando *Dart* y un motor de renderizado propio, compilando a código nativo. *Flutter* permite construir interfaces visuales consistentes en diferentes plataformas y evita la dependencia de componentes nativos del sistema operativo, similar a *Expo*. Sin embargo, esta alternativa también fue descartada principalmente debido a la poca familiaridad con el lenguaje *Dart*, lo que habría alargado inevitablemente el tiempo de desarrollo del proyecto.

2.4.2. Capa de Datos y Persistencia Local

Uno de los requisitos fundamentales de PANOT es garantizar la funcionalidad de la aplicación incluso en ausencia de conectividad a Internet, permitiendo a los usuarios capturar interacciones y gestionar sus relaciones de manera continua independientemente de su estado de conexión. Para materializar este requisito, se ha adoptado un enfoque *local-first*, donde los datos se almacenan localmente en primer lugar, sincronizándose con el servidor cuando la conectividad está disponible.

La implementación de este patrón se ha realizado mediante la combinación de *Legend State*, una librería de gestión de estado reactiva y eficiente, junto con las capacidades de almacenamiento local proporcionadas por Expo. Legend State proporciona un sistema de observables que permite mantener un estado global sincronizado entre los componentes de la aplicación, mientras que Expo ofrece APIs para el almacenamiento persistente en el dispositivo mediante *AsyncStorage* o sistemas de base de datos locales como *SQLite*.

El flujo de trabajo implementado sigue el siguiente patrón: cuando el usuario realiza una acción (por ejemplo, registrar una interacción), los datos se almacenan inmediatamente en el estado local gestionado por Legend State y se persisten en el almacenamiento local del dispositivo. Esta operación es instantánea y no requiere conectividad. Posteriormente, en segundo plano, la aplicación intenta sincronizar estos datos con el servidor cuando detecta conectividad disponible. Si la sincronización falla temporalmente, los datos permanecen en el dispositivo y se reintenta automáticamente cuando la conexión se restablece, garantizando que ninguna información se pierda.

Para la persistencia en la nube, se ha seleccionado *Supabase* como plataforma de backend, que proporciona una base de datos *PostgreSQL* gestionada junto con APIs REST y en tiempo real. Esta elección se fundamenta en:

- Simplicidad de integración: *Supabase* proporciona un cliente JavaScript/TypeScript que se integra naturalmente con React Native y Expo, facilitando la sincronización bidireccional entre el estado local y la base de datos.
- Escalabilidad: PostgreSQL es un motor de bases de datos relacional robusto y escalable, capaz de gestionar grandes volúmenes de datos y relaciones complejas entre entidades.
- Sincronización en tiempo real: *Supabase* ofrece capacidades de suscripción a cambios en tiempo real, permitiendo que actualizaciones realizadas desde otros dispositivos se reflejen automáticamente en la aplicación.
- Seguridad integrada: Como se mencionó en la sección de privacidad, *Supabase* proporciona mecanismos de seguridad a nivel de fila y columna que garantizan el aislamiento de datos entre usuarios y encriptación de datos en todo su ciclo de vida.
- Coste de uso: el plan gratuito de *Supabase* ofrece llamas ilimitadas a su API, gestión de hasta 500.000 usuarios activos, y hasta 500MB de almacenamiento de datos, lo que supone una base sólida para hacer posible el desarrollo del proyecto.

Esta arquitectura local-first no solo garantiza la funcionalidad offline, sino que también mejora la experiencia del usuario al proporcionar respuestas instantáneas sin depender de la latencia de red, mientras mantiene la consistencia de datos a través de la sincronización automática en segundo plano.

2.4.3. Capa de Servicios Externos

Además de proporcionar la infraestructura de base de datos y autenticación, *Supabase* se ha utilizado como plataforma de backend completo para gestionar todas las conexiones con servicios externos mediante su arquitectura de *Edge Functions*, que actúa como un API Gateway centralizado. Esta decisión arquitectónica se alinea directamente con los principios de seguridad y eficiencia por diseño establecidos en el proyecto.

Las *Edge Functions* de *Supabase* son funciones serverless escritas en TypeScript y ejecutadas en un runtime basado en *Deno*. Estas funciones son distribuidas y replicadas globalmente en servidores para reducir así la latencia en función del posicionamiento geográfico del usuario.

1. Entrada de solicitud en el edge gateway: El API Gateway actúa como relay que enruta el tráfico, gestiona los headers de autenticación, valida los JSON Web Tokens (JWT) y aplica reglas de enrutamiento y control de tráfico.

2. Aplicación de autenticación y políticas: El API Gateway valida los JWT de *Supabase*, aplica rate limiting y centraliza las comprobaciones de seguridad antes de ejecutar el código, garantizando que solo solicitudes autenticadas y autorizadas accedan a los servicios externos.

3. Ejecución en el edge runtime: La función se ejecuta en un nodo de *Edge Runtime* distribuido regionalmente más cercano al usuario, minimizando la latencia de procesamiento.

4. Integraciones y acceso a datos: Las funciones comúnmente invocan APIs de *Supabase* (Auth, Postgres, Storage) o APIs de terceros, utilizando estrategias de conexión optimizadas para entornos edge/serverless.

5. Observabilidad y logs: Las invocaciones emiten logs y métricas que pueden explorarse en el dashboard de *Supabase* o integrarse con sistemas de monitorización downstream como *Sentry*.

6. Respuesta a través del gateway: El API Gateway reenvía la respuesta al cliente y registra los metadatos de la solicitud para análisis y auditoría posterior.

Esta arquitectura de API Gateway [COMO SE PUEDE VER EN LA FIG.N] proporciona múltiples beneficios que justifican su adopción:

- Seguridad centralizada: Todas las claves de API y credenciales de servicios externos se mantienen en el servidor, nunca expuestas al cliente móvil. El gateway valida automáticamente la autenticación antes de procesar cualquier solicitud, reduciendo significativamente la superficie de ataque.

- Baja latencia: La distribución global de las funciones garantiza que las solicitudes se procesen en el nodo más cercano geográficamente al usuario, reduciendo el tiempo de respuesta total.

- Escalabilidad automática: Las Edge Functions se escalan automáticamente según la demanda, eliminando la necesidad de gestionar infraestructura de servidores y optimizando los costes operativos mediante un modelo de pago por uso.

- Simplificación del desarrollo: La integración nativa con las APIs de *Supabase* permite que las funciones accedan directamente a la base de datos, autenticación y almacenamiento sin configuración adicional, reduciendo la complejidad del código y la gestión de credenciales.
- Rate limiting y control de tráfico: El gateway proporciona mecanismos integrados para limitar la frecuencia de solicitudes, protegiendo tanto los servicios externos como la infraestructura propia de sobrecargas.

[incluir brevemente alternativas consideradas]

2.4.4. Capa de Inteligencia Artificial

La capa de inteligencia artificial de PANOT, responsable del procesamiento de interacciones y la generación de recomendaciones contextuales, se ha implementado mediante *Supabase Edge Functions* que actúan como intermediario entre la aplicación móvil y la *API de OpenAI*, utilizando el modelo [TODAVIA POR COMPARAR] para el procesamiento de lenguaje natural.

La elección de *OpenAI* como proveedor de servicios de inteligencia artificial se fundamenta en su alta adopción por parte de la comunidad de desarrolladores y su extensa documentación, que facilita, de nuevo, la integración y resolución de problemas durante el desarrollo. *OpenAI* mantiene una hoja de ruta activa con actualizaciones frecuentes de sus modelos y mejoras continuas en la *API de OpenAI*, garantizando la evolución y mantenibilidad a largo plazo de la integración.

La selección del modelo específico [TODAVIA POR COMPARAR] se fundamenta en su equilibrio entre eficiencia, calidad y coste. Este modelo ofrece tiempos de respuesta menores que alternativas más grandes como [TODAVIA POR COMPARAR], permitiendo una experiencia de usuario más fluida en aplicaciones móviles donde la latencia es crítica. A pesar de ser más ligero, mantiene capacidades de comprensión y generación de lenguaje natural suficientes para las tareas requeridas en PANOT, como el análisis semántico de interacciones y la generación de recomendaciones personalizadas. Además, su coste por token sustancialmente menor es fundamental para la sostenibilidad económica del proyecto, considerando que las operaciones de inteligencia artificial se ejecutan frecuentemente en segundo plano.

[TABLA COMPARATIVA DE LOS TRES MODELOS DE OPENAI]

Las *Edge Functions* encapsulan la lógica de comunicación con la *API de OpenAI*, incluyendo manejo de errores, gestión de timeouts y validación de la estructura de respuesta, abstrayéndose así de acoplar la lógica de comunicación con el cliente móvil.

Como alternativas se consideraron los modelos *Claude de Anthropic*, que ofrece capacidades avanzadas de razonamiento pero con un coste de computación sustancialmente mayor.[VER A VE ESTO PORQUE REALMENTE SE HA ELEGIDO OPENAI POR ELECCION PERSONAL]

[TABLA COMPARATIVA DE AMBOS MODELOS]

2.4.5. Capa de Sistema de Pagos

Para la integración de funcionalidades de pago y suscripciones, se ha seleccionado *Stripe* como proveedor de servicios de pasarela de pagos. *Stripe* es una plataforma líder que proporciona APIs robustas y seguras para procesar pagos, gestionar suscripciones recurrentes y manejar la facturación.

La integración con *Stripe* se realiza también mediante *Supabase Edge Functions*. Las *Edge Functions* gestionan la creación de intenciones de pago, la validación de tarjetas y el procesamiento de transacciones de forma segura, además de recibir y procesar webhooks para manejar eventos asíncronos como confirmaciones de pago o actualizaciones de métodos de pago, sincronizando automáticamente el estado en la base de datos. Además, *Stripe* gestiona automáticamente el cumplimiento de estándares de seguridad como PCI DSS (Payment Card Industry Data Security Standard).

Como alternativa a *Stripe* se evaluó *Polar*, una plataforma de pasarela de pagos que ofrece funcionalidades similares para el procesamiento de transacciones. La principal diferencia entre ambas plataformas radica en que *Polar*, aunque ofrece una mejor experiencia de desarrollo y una simplicidad de implementación mayor a *Stripe*, está desarrollada fundamentalmente para aplicaciones web, mientras que *Stripe* proporciona un soporte nativo y maduro para aplicaciones móviles con Kit de Desarrollo de Software (SDKs) específicos para iOS y Android. *Polar* no cuenta con suficiente adopción en el ecosistema de aplicaciones móviles, lo que se traduce en documentación limitada, menos ejemplos de integración y una comunidad de desarrolladores más reducida para este tipo de aplicaciones. Esta orientación hacia aplicaciones web añadiría una complejidad adicional al proyecto, ya que requeriría adaptaciones y soluciones personalizadas para integrar correctamente los pagos en la aplicación móvil desarrollada con

Expo, incrementando el tiempo de desarrollo y el riesgo de problemas de compatibilidad. Por estas razones, se descartó *Polar* en favor de *Stripe*, que ofrece una integración más directa y documentada para aplicaciones móviles, reduciendo la complejidad del desarrollo y garantizando una experiencia de usuario más fluida.

2.4.6. Capa de Observabilidad

Una parte crítica en el desarrollo de productos digitales, es tener un claro entendimiento de cómo los usuarios interactúan con el producto. Para ello, es esencial que los desarrolladores tengan documentadas métricas de éxito y uso de funcionalidades, con el objetivo de poder tomar decisiones informadas basadas en datos reales, teniendo en cuenta qué aspectos de la aplicación están aportando valor al usuario y cuales no.

Para llevar a cabo este análisis de uso de funcionalidades y comprensión del comportamiento de los usuarios, se ha integrado una capa de observabilidad con *PostHog*, una plataforma de análisis de productos de código abierto que proporciona lwa capacidad de hacer un seguimiento de eventos una vez la aplicación está en manos del usuario.

La selección de *PostHog* se fundamenta en su compatibilidad con React Native y Expo, su modelo de código abierto que permite desplegar la plataforma de forma autohospedada si se requiere mayor control sobre los datos, y su enfoque específico en análisis de producto que proporciona métricas relevantes para, como ya hemos comentado, la toma de decisiones basadas en datos. Además, *PostHog* ofrece una capa gratuita que permite comenzar el seguimiento de eventos sin coste inicial, lo que se ajusta al contexto de este proyecto.

El seguimiento de eventos se implementa mediante llamadas a funciones especcíficas, que permiten registrar eventos personalizados con propiedades asociadas. Por ejemplo, cuando un usuario realiza una acción específica en la aplicación, como completar una interacción o visualizar una recomendación, se registra un evento con identificadores únicos y metadatos relevantes. *PostHog* también proporciona funcionalidades adicionales como la identificación de usuarios, que permite asociar eventos a usuarios específicos y realizar análisis de comportamiento individual, y el seguimiento automático de eventos del sistema como inicios de sesión o actualizaciones de la aplicación.

Como alternativas se consideraron *Mixpanel*, que ofrece herramientas avanzadas

para el análisis de productos y retención de usuarios con un SDK robusto para React Native, y *Amplitude*, una plataforma líder en análisis de productos que proporciona capacidades avanzadas de Análisis de Cohorte y Embudos. Sin embargo, ambas alternativas presentan modelos de precios más restrictivos en sus planes gratuitos comparado con *PostHog*, y requieren una configuración más compleja para su integración con Expo. También se evaluó *Google Analytics*, una solución gratuita y ampliamente utilizada, pero su enfoque está más orientado al análisis de tráfico web que al análisis de producto específico para aplicaciones móviles.

3. Desarrollo del Proyecto

3.1. Filosofía y Metodología de Desarrollo

El desarrollo de PANOT se ha planteado desde una perspectiva que prioriza la validación empírica y la eficiencia en la entrega de valor, alejándose de los enfoques tradicionales de planificación predictiva en cascada. Dado el carácter innovador de la propuesta —la gestión de la inteligencia relacional mediante IA —, el proyecto se enfrenta a un alto grado de incertidumbre, no tanto en la viabilidad técnica, sino en la validación del producto por parte del usuario final.

Para mitigar este riesgo, se ha adoptado una filosofía fundamentada en los principios de *Lean Startup* [5] y *Lean Thinking* [6]. Bajo este prisma, el desarrollo de software no se entiende como la ejecución de una especificación cerrada, sino como un proceso de descubrimiento orientado a minimizar el desperdicio (Muda) —entendido como cualquier esfuerzo o característica que no genere valor para el usuario.

En consecuencia, la metodología de trabajo implementada en este Trabajo Fin de Grado no tiene como objetivo la finalización de un producto comercial definitivo, sino la construcción y despliegue de un Producto Mínimo Viable. Este PMV constituye el punto de partida necesario para ejecutar la primera fase del ciclo Construir-Medir-Aprender, permitiendo someter a prueba las hipótesis conceptuales detalladas en el capítulo anterior y generar un aprendizaje que guíe la evolución futura de la plataforma.

3.1.1. Enfoque Lean

Para el desarrollo del sistema, como ya hemos comentado, se ha descartado la adopción de modelos tradicionales en cascada e incluso de marcos ágiles rígidos como Scrum, en favor de una filosofía fundamentada en los principios de *Lean Software Development*. Respondiendo así a la necesidad inherente de gestionar la alta incertidumbre que los productos de características similares a PANOT suponen.

Esta decisión cobra especial relevancia al considerar la naturaleza unipersonal del equipo de desarrollo. Mientras que metodologías como Scrum imponen una sobrecarga de gestión mediante reuniones y artefactos diseñados para la coordinación grupal, el enfoque Lean permite aplicar el principio de Optimizar el Todo, eliminando la burocracia innecesaria. Esto maximiza el ancho de banda cognitivo disponible para tareas de ingeniería y diseño, permitiendo mantener un flujo de entrega continuo sin las interrupciones artificiales de los *sprints* temporales.

Bajo esta premisa, el trabajo presentado en esta memoria no debe entenderse como un producto final inmutable, sino como la materialización de la fase *Construir* del ciclo Construir-Medir-Aprender. El PMV de PANOT actuará como el artefacto necesario para validar la hipótesis de *Inteligencia Relacional* elimanando todo el desarrollo superfluo que no contribuya a esta validación, cumpliendo así con el principio Lean de Eliminar Desperdicio.

Así mismo, el éxito del proyecto no se cuantifica mediante el número de funcionalidades entregadas o el cumplimiento estricto de un cronograma predictivo, sino a través de la capacidad del sistema para generar *Aprendizaje Validado*. El objetivo técnico es instrumentar la aplicación (capa de observabilidad) para que, en iteraciones futuras, sea posible medir con datos reales el compromiso del usuario y la utilidad de la propuesta.

3.1.2. Metodología y Flujo de Desarrollo

Para organizar de manera robusta y estable la carga de trabajo del proyecto, se ha optado por la adopción de la metodología de ramificación *GitFlow*. Este modelo se fundamenta en la segregación de responsabilidades, estableciendo dos ramas longevas principales: `main`, que contiene exclusivamente código estable de producción, y `development`, que actúa como entorno de integración.

El desarrollo de nuevas características se realiza de manera aislada en ramas efímeras (`feature/*`), que solo se fusionan con la rama de integración tras ser completadas y validadas —Ver 3.1—. Esta estrategia garantiza la estabilidad del sistema al evitar que el código en desarrollo comprometa la funcionalidad base.

Para la orquestación de las tareas, se ha utilizado GitHub Projects, implementando un tablero Kanban automatizado que permite visualizar el flujo de valor y limitar el trabajo en curso. La trazabilidad entre la planificación y la ejecución se articula mediante

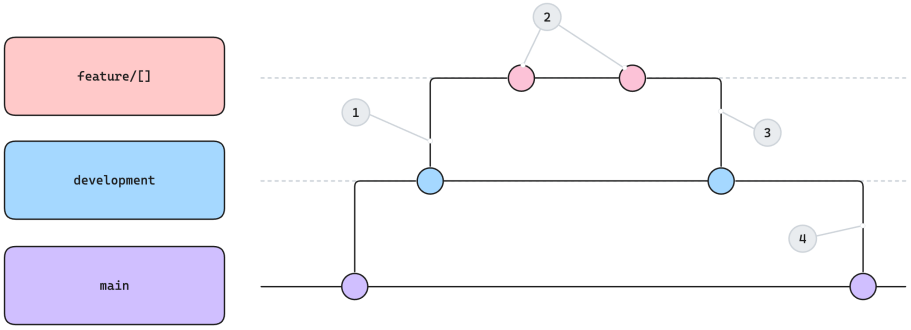

Figura 3.1. Diagrama orientativo de la estrategia de ramificación. (1) Aislamiento: Creación de una rama efímera (*feature*) a partir de desarrollo para trabajar en un *Issue* específico. (2) Implementación: Desarrollo de la funcionalidad mediante *commits* atómicos. (3) Integración: Fusión de la rama mediante un *Pull Request* validado, cerrando el ciclo de desarrollo. (4) Release: Promoción del código estable desde desarrollo a la rama principal (*main*) para su despliegue en producción.

dos artefactos clave:

- Issues: Representan las unidades de trabajo o requisitos del sistema. Cada *Issue* posee un identificador único #N y, para organizar la granularidad del proyecto, se ha establecido una jerarquía de dos niveles:
  - [Epic] Issues: Representan las grandes funcionalidades o módulos del sistema (Features).
  - FR (Functional Requirements): Actúan como sub-tareas atómicas vinculadas a un Epic, definiendo requisitos concretos implementables en una sola iteración.
- Pull Requests (PRs): Constituyen el mecanismo de control de calidad y fusión de código. Una vez finalizado el desarrollo en una rama `feature/*`, se abre un PR hacia `development`. En el contexto de este proyecto, el PR cumple una doble función: actúa como una etapa de revisión de código obligatoria —permitiendo verificar diferencias (*diffs*) y detectar errores antes de la integración— y sirve como disparador automático para cerrar los *Issues* asociados, garantizando así la trazabilidad completa entre el requisito definido y el código implementado.

Finalmente, estas tareas transitan por el tablero Kanban siguiendo un flujo de estados que refleja el ciclo de vida del desarrollo:

- Backlog: Contiene el conjunto de *Issues* pendientes de implementación. Dado que el alcance del proyecto se ha limitado estrictamente al PMV, todas las tareas

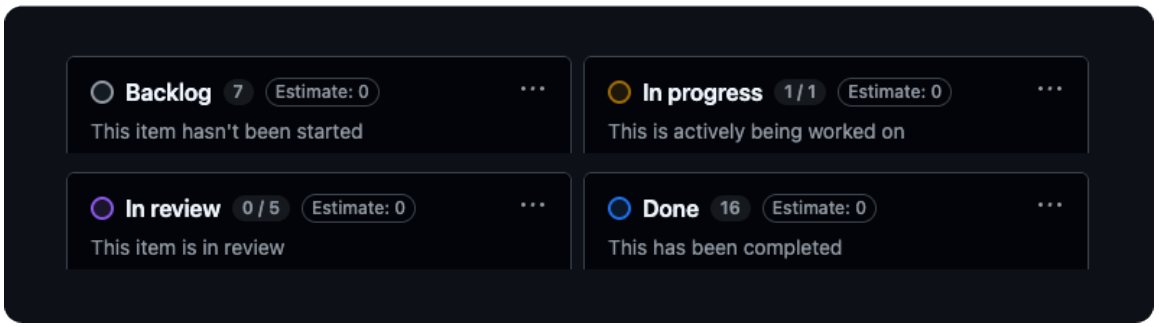

Figura 3.2. Visualización de las columnas del tablero Kanban implementado en GitHub Projects en un punto del desarrollo. Se destaca el límite de WIP configurado a 1 en la columna *In Progress* (indicador 1/1), forzando un flujo continuo y eliminando el desperdicio asociado al cambio de contexto.

presentes en esta columna se consideran críticas y obligatorias para el funcionamiento del sistema, habiéndose descartado previamente cualquier funcionalidad accesoria.

- In Progress: Indica la tarea que se está codificando activamente en ese momento. Siguiendo la filosofía Lean de Eliminar Desperdicio por cambio de contexto, se mantiene un límite de trabajo en curso (WIP) estricto de una única tarea simultánea.
- In Review: Estado en el que la funcionalidad ha sido completada y existe un *Pull Request* abierto. Actúa como fase de control de calidad y auto-revisión de las diferencias de código (*diff*) antes de su integración.
- Done: Agrupa las tareas finalizadas cuyo código ha sido validado y fusionado (*merged*) exitosamente en la rama `develop`, cerrando así el *Issue* asociado y marcando la entrega de valor.

3.2. Definición del PMV y Especificación de Requisitos

Siguiendo el marco teórico expuesto por *Eric Ries* [5] el desarrollo de PANOT se fundamenta en la identificación y validación de *Asunciones de Salto Fe*. Estas son las premisas de mayor incertidumbre y riesgo que, de resultar falsas, invalidarían la totalidad del proyecto.

Las dos asunciones fundamentales en este marco son:

- Hipótesis de Valor: Determina si un producto o servicio proporciona valor real a los usuarios una vez que lo utilizan. Se valida analizando la retención y el uso voluntario y respondería a la pregunta: *¿El usuario percibe suficiente utilidad en el sistema como para incorporarlo a su vida diaria?*
- Hipótesis de Crecimiento: Determina cómo los nuevos usuarios descubrirán el producto o servicio. Se valida analizando los motores de crecimiento (viral, pegajoso o remunerado). Responde a la pregunta: *¿Existe un mecanismo sostenible para adquirir nuevos usuarios y escalar el producto?*

A modo aclarativo, el esfuerzo de ingeniería de este Proyecto Fin de Grado se ha focalizado en la validación de la *Hipótesis de Valor*, dejando la validación de la *Hipótesis de Crecimiento* para estados futuros del ciclo de vida de la aplicación.

3.2.1. Hipótesis de Valor

Teniendo en cuenta lo anterior, el artefacto para validar la *Hipótesis de Valor* es un PMV que representa la versión mínima funcional de PANOT. La hipótesis concreta es que los usuarios adoptarán el sistema porque resuelve la carga cognitiva y el esfuerzo manual que supone mantener relaciones personales. Esta hipótesis general se desglosa en tres premisas clave:

1. Valor de la Inmediatez: Se asume que los usuarios serán constantes en el registro de información solo si el esfuerzo requerido y la fricción son cercanos a cero. La captura por voz ofrece un valor superior a la entrada de texto tradicional al permitir registrar interacciones en movimiento y sin necesidad de una atención visual exhaustiva.
2. Valor de la Mantenibilidad Pasiva: Se asume que los usuarios perciben valor en un sistema que mantiene actualizada la información de sus contactos (cambios de trabajo, mudanzas, gustos) de forma autónoma, liberándoles de la tarea administrativa de editar fichas de contacto manualmente.
3. Valor de la Memoria Aumentada: Se asume que los usuarios obtienen utilidad real al disponer de resúmenes contextuales precisos antes de una interacción, mejorando así la calidad de sus relaciones sociales gracias a una ”memoria externa”que procesa el contexto del contacto por ellos.

Se han establecido una serie de requisitos obligatorios que instrumentarán al sistema con las capacidades para validar las premisas planteadas. Estos requisitos de fun-

cionalidad se han agrupado en las siguientes características principales (Features):

ID	*Nombre*	*Requisitos Funcionales Asociados (IDs)*
F1	Gestión de Usuario	FR-01, FR-02, FR-03
F2	Gestión de Contactos	FR-04, FR-05, FR-06, FR-07, FR-08, FR-09
F3	Gestión de Interacciones	FR-10, FR-11, FR-12, FR-13
F4	Inteligencia Relacional	FR-14, FR-15
F5	Soporte y Feedback	FR-16, FR-17

Tabla 3.1. Features del Sistema y los IDs de los Requisitos Funcionales Asociados

3.2.2. Requisitos Funcionales

ID	*Descripción*
FR-01	El sistema debe permitir el registro e inicio de sesión de usuarios de forma segura.
FR-02	El sistema debe permitir cerrar sesión al usuario de forma segura.
FR-03	El sistema debe permitir al usuario la eliminación completa de su cuenta y toda la información asociada a ella, lo que debe desencadenar el borrado de sus datos tanto en el dispositivo local como en el servidor remoto.

Tabla 3.2. Requisitos Funcionales de la *Feature 1 - Gestión de Usuario*

ID	*Descripción*
FR-04	El sistema debe permitir dar de alta un contacto manualmente introduciendo, al menos, un nombre identificativo y adicionalmente, un campo de detalles/descripción inicial.
FR-05	El sistema debe permitir al usuario importar contactos existentes desde la agenda nativa del dispositivo móvil para poblar la base de datos inicial sin entrada manual.
FR-06	El sistema debe permitir la creación de un nuevo perfil de contacto mediante dictado de voz en lenguaje natural. El usuario describirá a la persona y el sistema procesará el audio para inferir automáticamente el nombre y poblar el campo de detalles iniciales sin necesidad de escritura manual.
FR-07	El sistema debe permitir localizar contactos mediante una barra de búsqueda que filtre resultados tanto por coincidencia de nombre como por palabras clave contenidas en el campo de detalles/resumen del contacto.
FR-08	El sistema debe habilitar la modificación directa de los campos de un contacto (nombre, resumen, detalles) mediante entrada de texto estándar, permitiendo al usuario corregir posibles inexactitudes de la inferencia de IA o actualizar datos específicos bajo su propio criterio.
FR-09	El sistema debe permitir el borrado permanente de la ficha de un contacto existente. Esta acción debe desencadenar la eliminación del nodo correspondiente y asegurar que los datos asociados dejan de estar accesibles en la interfaz de usuario.

Tabla 3.3. Requisitos Funcionales de la *Feature 2 - Gestión de Contactos*

ID	*Descripción*
FR-10	El sistema debe disponer de una interfaz dedicada para iniciar, detener y gestionar la grabación de audio de una nueva interacción o evento. Este módulo actuará como la fuente de entrada principal para el motor de procesamiento, gestionando los permisos de hardware y el flujo de audio del dispositivo.
FR-11	El sistema debe permitir al usuario editar manualmente el texto transcrito antes de confirmar su envío al motor de procesamiento, garantizando la corrección de errores de transcripción o la eliminación de datos que no se deseen procesar.
FR-12	El sistema debe capturar audio a través del micrófono del dispositivo y convertirlo a texto en tiempo real, mostrando el resultado en pantalla para la revisión inmediata del usuario.
FR-13	El sistema debe ofrecer la funcionalidad explícita de descartar/eliminar una interacción. Esto permite al usuario eliminar tomas erróneas o accidentales sin que estas consuman recursos de procesamiento ni afecten al historial del contacto.

Tabla 3.4. Requisitos Funcionales de la *Feature 3 - Gestión de Interacciones*

ID	*Descripción*
FR-14	El sistema debe procesar la información de la interacción para actualizar los detalles del contacto almacenados en su grafo semántico, agregando la nueva información o eliminando la información obsoleta sin modificar el contexto histórico importante.
FR-15	El sistema debe ser capaz de generar relaciones entre los nodos semánticos de los contactos usando métodos de matching semántico.

Tabla 3.5. Requisitos Funcionales de la *Feature 4 - Inteligencia Relacional*

ID	*Descripción*
FR-16	El sistema debe disponer de una interfaz accesible desde el menú de configuración que permita al usuario enviar un reporte de error o sugerencia, incluyendo opcionalmente una captura de pantalla o descripción del problema.
FR-17	El sistema debe proporcionar un mecanismo dedicado en la interfaz de configuración que permita al usuario enviar propuestas de mejora o solicitar nuevas capacidades funcionales.

Tabla 3.6. Requisitos Funcionales de la *Feature 5 - Soporte y Feedback*

3.2.3. Requisitos No Funcionales

Para complementar a la funcionalidad, es esencial que el sistema de PANOT mantenga cierto nivel de calidad. A continuación se enumeran los indicadores de calidad establecidos para la plataforma:

ID	*Tipo*	*Descripción*	*Criterio Aceptación*
RNF-SEG-001	Seguridad	El sistema debe garantizar que un usuario autenticado solo pueda acceder, leer o modificar los registros asociados a su propio ID de usuario en la base de datos remota.	El 100 % de las consultas a Supabase deben estar protegidas por políticas Row Level Security (RLS) activas y validadas mediante tests de seguridad
RNF-REND-002	Rendimiento	El tiempo total de ejecución para el análisis semántico de una interacción (extracción de entidades y actualización de los nodos del grafo en la base de datos) debe mantenerse dentro de límites tolerables para una operación en segundo plano.	El ciclo completo de procesamiento de inteligencia no debe exceder los 20 segundos para interacciones de longitud media (hasta 500 palabras).
RNF-EFI-003	Eficiencia / Coste	El sistema debe optimizar la gestión del contexto y la selección del modelo de lenguaje para garantizar la viabilidad económica del proyecto a escala.	El coste medio de procesamiento de operaciones individuales no debe superar las 0,002 unidades monetarias, validado mediante el cálculo de consumo de tokens por llamada.
RNF-DISP-004	Disponibilidad	El sistema debe permitir la creación, lectura y edición de contactos e interacciones sin necesidad de conexión a internet.	Disponibilidad del 100 % de las funcionalidades Core (Crear Interacción, Ver Contacto, Grabar Interacción, Asociar Interacción) con el dispositivo en ”Modo Avión”.
RNF-DISP-005	Disponibilidad	El sistema debe asegurar la consistencia eventual de los datos entre el dispositivo local y el servidor remoto tras periodos de desconexión.	Los datos creados offline deben reflejarse en el servidor en un tiempo <30 segundos tras la recuperación de una conexión estable.
RNF-USAB-006	Usabilidad	El proceso para crear un contacto o interacción debe requerir el mínimo esfuerzo cognitivo y físico por parte del usuario	El usuario debe poder iniciar una grabación desde la pantalla de inicio en máximo 2 acciones.
RNF-USAB-007	Usabilidad	El sistema debe informar al usuario sobre el estado de procesos largos para evitar incertidumbre.	Cualquier operación que supere los 500 ms debe mostrar un indicador visual de carga.
RNF-MANT-008	Mantenibilidad	El sistema debe capturar métricas de uso para validar las hipótesis de valor definidas en el proyecto.	El 100 % de los eventos críticos definidos (Login, Interacción Creada, Error de API) deben registrarse correctamente en la plataforma de analítica.

Tabla 3.7. Requisitos No Funcionales del sistema con ID, Tipo, Descripción y Criterio de Aceptación

3.3. Diseño del Sistema

3.3.1. Visión general de la Arquitectura

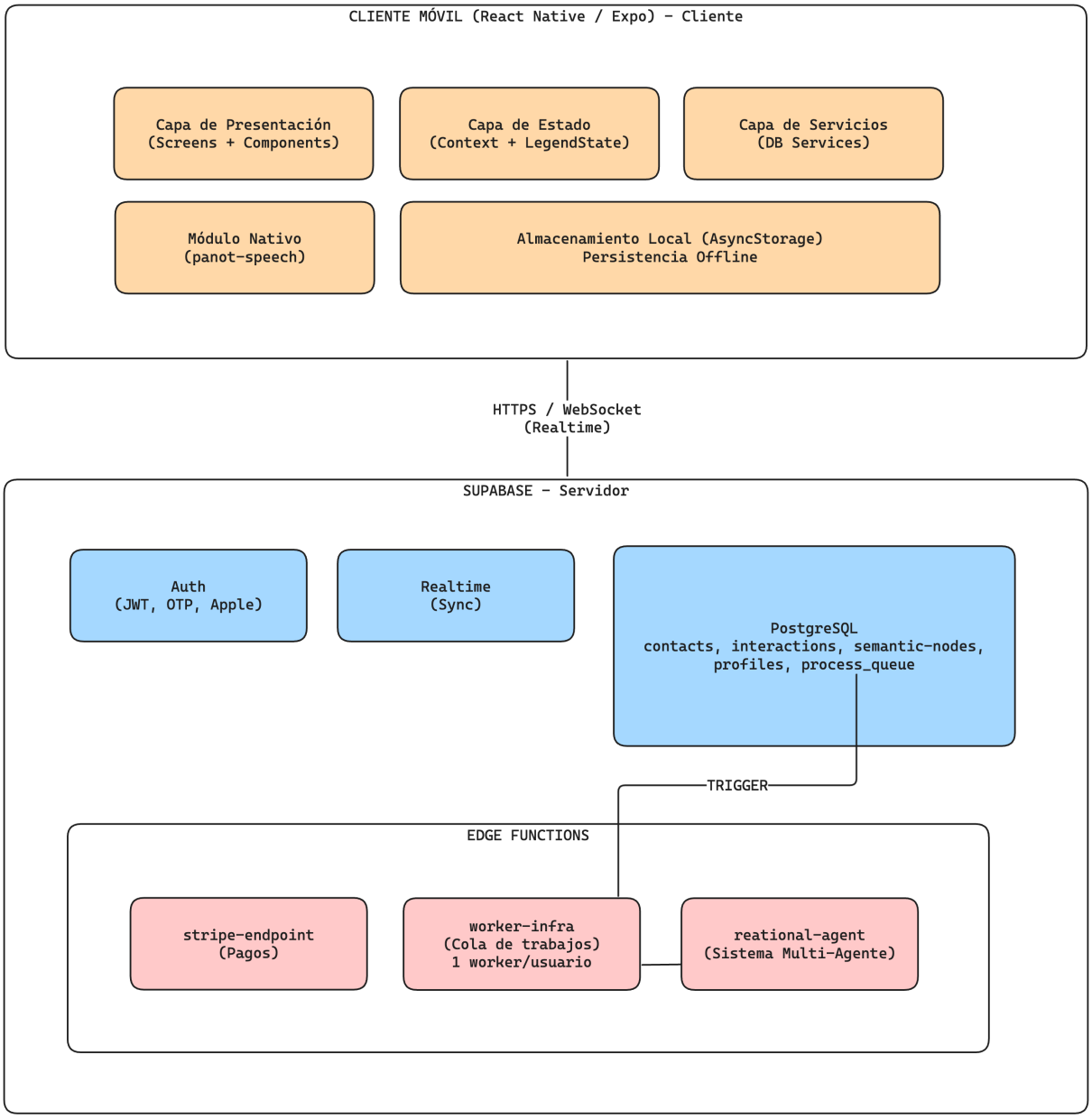

Figura 3.3. Arquitectura general del sistema PANOT

La arquitectura de PANOT sigue un estilo cliente-servidor con las siguientes características distintivas:

- Arquitectura Offline-First: El cliente móvil mantiene una copia local completa de los da-

tos, permitiendo operar sin conexión. La sincronización con el servidor se realiza automáticamente cuando hay conectividad disponible, garantizando disponibilidad del sistema en condiciones de red intermitente.

- Backend as a Service (BaaS): Se utiliza Supabase como plataforma de backend, proporcionando autenticación, base de datos PostgreSQL con Row Level Security, y sincronización en tiempo real, sin necesidad de gestionar infraestructura propia.
- Computación Serverless: Toda la lógica de backend se ejecuta en Supabase Edge Functions, funciones serverless basadas en Deno que se invocan bajo demanda. Esto incluye la integración con servicios externos (Stripe) y el sistema de procesamiento de IA.
- Procesamiento mediante Cola de Trabajos: Las tareas complejas se encolan en la base de datos y se procesan mediante triggers que invocan las Edge Functions. El modelo de ejecución combina dos niveles:
  - *Asíncrono entre usuarios*: Las peticiones de distintos usuarios se procesan en paralelo, sin bloquearse mutuamente.
  - *Secuencial por usuario*: Cada usuario tiene un worker dedicado que procesa sus tareas en orden, una tras otra, evitando condiciones de carrera sobre sus propios datos.

Capa Cliente (React Native / Expo) La aplicación móvil se organiza internamente en tres subcapas diferenciadas:

- Presentación: Pantallas y componentes React Native con navegación declarativa mediante Expo Router.
- Estado: La gestión del estado se divide en dos capas. Los React Contexts manejan el estado de la interfaz (grabación activa, búsquedas, preferencias de usuario), mientras que *LegendState* gestiona los datos de negocio (contactos, interacciones, perfiles) con persistencia local mediante *AsyncStorage* y sincronización automática con el servidor.
- Servicios: Clases especializadas (*ContactsService*, *InteractionsService*, *SemanticNodesService*, *ProcessQueueService* y *ProfilesService*) que encapsulan las operaciones CRUD sobre Supabase, proporcionando una interfaz para consultas y modificaciones de datos.

Adicionalmente, el módulo nativo y propietario `panot-speech` expone las APIs de reconocimiento de voz de iOS para transcripción en tiempo real, garantizando operabilidad *offline* e independencia de servicios externos.

Capa de Servidor (Supabase) El backend aprovecha los servicios gestionados de Supabase descritos en el anterior Capítulo:

- Autenticación: Email con OTP y Apple Sign-In, con gestión automática de tokens JWT.
- Base de datos: Persistencia de datos en PostgreSQL con políticas RLS para aislamiento de datos por usuario y encriptación de datos en transito y en reposo.
- Realtime: Suscripciones WebSocket para sincronización entre dispositivos.
- Triggers de procesamiento: Disparan Edge Functions al insertar registros en `process_queue`, implementando el patrón de cola de mensajes.

Capa de Procesamiento (Edge Functions) Tres funciones *serverless* implementan la lógica de backend:

- `stripe-endpoint`: Procesa webhooks de Stripe gestionando el ciclo de vida de suscripciones mediante Intenciones de Pago, Claves Efímeras y Clientes Temporales.
- `worker-infra`: Sistema de cola de trabajos que instancia un *worker* por usuario, garantizando procesamiento secuencial y evitando condiciones de carrera.
- `relational-agent`: Orquesta el sistema multi-agente para procesamiento semántico, detallado en la Sección 3.3.2.

La arquitectura del sistema se fundamenta en cinco decisiones clave o ADRs (Architectural Decision Records) que condicionan el diseño general:

ADR-1 Arquitectura Offline-First. Dado que la aplicación se utiliza frecuentemente en contextos de networking (eventos, conferencias, parkings, etc.) con conectividad intermitente, se optó por implementar sincronización *offline-first* mediante *LegendState* y persistencia en *AsyncStorage*. Esta decisión garantiza disponibilidad del 100 % y elimina la latencia percibida al operar con datos locales, aunque introduce complejidad en la resolución de conflictos de sincronización.

ADR-2 Supabase como BaaS. Tratándose de un proyecto individual con recursos limitados que requiere autenticación, base de datos y comunicación en tiempo real, se seleccionó Supabase frente a un backend personalizado. Esta elección reduce significativamente el tiempo de desarrollo, aunque se asume un Vendor Lock-In parcial como contrapartida.

ADR-3 Un Worker por Usuario. El procesamiento de transcripciones modifica datos del usuario (contactos, grafo semántico), lo que podría generar inconsistencias si múltiples trabajos se ejecutan en paralelo. Por ello, cada usuario dispone de un *worker* dedicado que procesa sus jobs de forma secuencial, garantizando la consistencia de datos a costa de mayor latencia cuando hay muchos trabajos encolados.

ADR-4 Sistema Multi-Agente. El procesamiento semántico requiere capacidades diferenciadas: gestión de contactos y gestión del grafo de conocimiento. Se implementó un orquestador que delega en agentes especializados (*ContactAgent*, *GraphAgent*), permitiendo separación de responsabilidades, prompts especializados y facilidad de extensión, con el Overhead de coordinación entre agentes como principal desventaja.

ADR-5 Edge Functions para Lógica de Backend. El sistema requiere ejecutar lógica de backend para procesamiento de pagos, gestión de cola de trabajos y procesamiento con IA. Frente a alternativas como AWS Lambda, Vercel Serverless o Cloudflare Workers, se optó por Supabase Edge Functions (basadas en Deno) por su integración nativa con la base de datos, gateway unificado de API, capacidad de invocación desde *triggers* de PostgreSQL y facturación por invocación sin costes de servidor inactivo. Como contrapartida, el runtime Deno presenta un ecosistema más reducido que Node.js, existe un límite de ejecución de 150 segundos, y se profundiza el Vendor Lock-In con Supabase.

3.3.2. Sistema Multi-Agente para Procesamiento de Información

El procesamiento semántico de las interacciones presenta un reto particular. Una misma interacción podría contener información de naturaleza muy diversa (datos de contacto, intereses personales, relaciones profesionales, etc.), lo que hace imposible predecir qué acciones tomar en cada caso concreto. Para abordar este problema, se ha optado por utilizar un sistema Multi-Agente.

A diferencia de un flujo de trabajo lineal, este sistema es capaz de razonar sobre el contenido y escoger qué operaciones realizar sobre la información. Además, el sistema de Multi-Agente de PANOT, ofrece ventajas a nivel de *modularidad*, ya que cada agente del sistema es especializado en un dominio concreto, y *extensibilidad*, ya que habilita la incorporación de nuevas capacidades en forma de herramientas o agentes sin necesidad de modificar el flujo y diseño general.

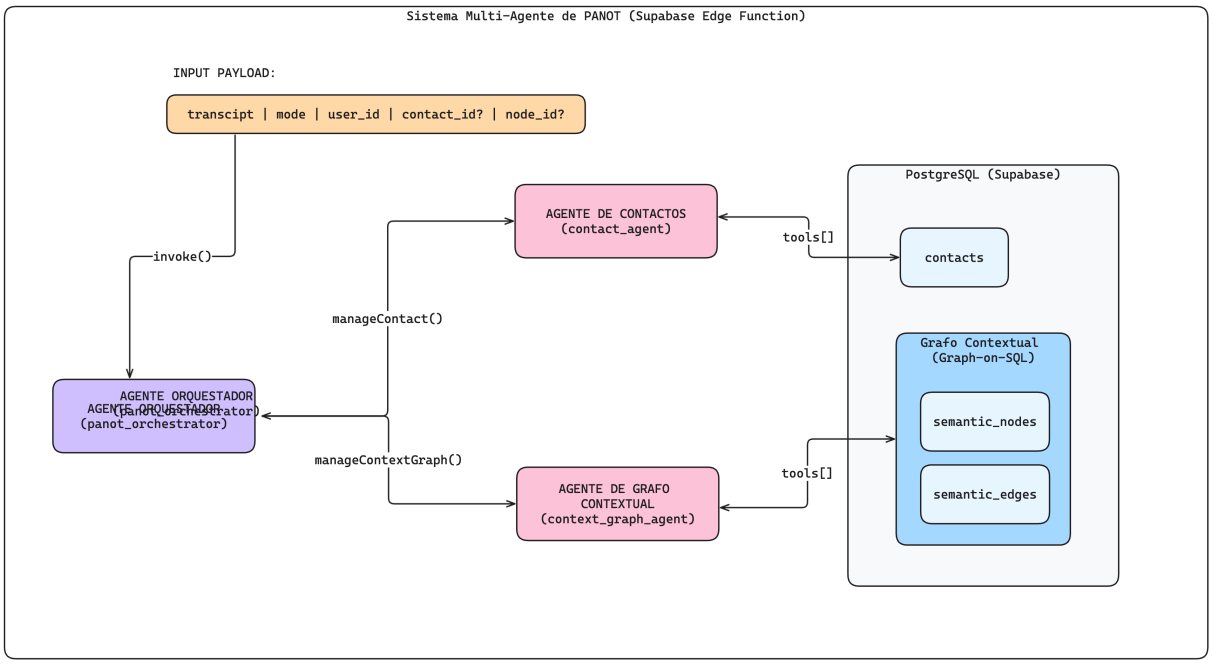

Figura 3.4. Diagrama de la arquitecturadel sistema multi-agente de PANOT

El sistema Multi-Agente de PANOT sigue una arquitectura vertical[INCLUIR REFERENCIA], donde un agente orquestador actúa como coordinador central, delegando tareas a agentes especializados que reportan sus resultados de vuelta. Esta estructura proporciona control centralizado sobre el flujo de ejecución, aislando, como ya se ha comentado, a cada agente en su dominio específico.

Agente Orquestador El agente orquestador constituye el punto de entrada del sistema Multi-Angente. El sistema recibe como entrada o bien la transcripción de una interacción o bien la nueva descripción de un contacto y, junto con el contexto necesario (`user_id`, `contact_id`, `node_id`), determina qué operaciones realizar sobre esta información. El sistema opera en tres modos diferenciados:

- ACTIONABLE: Modo predeterminado para el procesamiento de transcripciones. Permite ejecutar acciones de creación y actualización sobre contactos y el grafo contextual.
- CONTACT_DETAILS_UPDATE: Modo especializado para actualizar únicamente los datos del grafo contextual. Este modo está reservado a aquellos casos en los que el usuario modifica manualmente la descripción de un contacto.
- CONVERSATIONAL: Modo reservado para consultas sobre la red de contactos del usuario (por ejemplo, *¿qué sé de este contacto?*). En este modo, la entrada es una pregunta en

lenguaje natural del usuario y la salida, de igual manera, es una respuesta en lenguaje natural por parte del sistema, priorizando la recuperación de información existente.

El orquestador de PANOT analiza el contenido de la entrada y decide qué agentes invocar según la naturaleza de la información. Sus herramientas principales son los propios agentes especializados: `manageContact` para operaciones sobre contactos y `manageContextGraph` para gestión del grafo semántico.

Agente de Contactos - **`manageContact`** Este agente gestiona el ciclo de vida casi completo de los contactos del usuario. Su dominio abarca las operaciones de Crear, Leer y Actualizar sobre el artefacto *Contact* y dispone de las siguientes herramientas:

- `create_contact`: Crea un nuevo contacto en la base de datos. Cabe destacar que la creación del nodo semántico asociado (de tipo `CONTACT`) se delega a un *trigger* de PostgreSQL sobre inserción en la tabla de contactos, manteniendo así la separación de responsabilidades.
- `get_contact_data`: Recupera los datos almacenados de un contacto específico, permitiendo al orquestador contextualizar las decisiones posteriores.
- `update_contact_details`: Actualiza la información básica del contacto.

Agente de Grafo Contextual - **`manageContextGraph`** El agente de grafo contextual gestiona la estructura de conocimiento que representa la información semántica de los contactos del usuario. Esta estructura adopta la forma de un grafo donde los nodos representan entidades o conceptos y las aristas codifican el tipo de relaciones entre ellos. Como se analiza en la Sección 2.1.3, la elección de un grafo como modelo de datos permite representar relaciones complejas y multidimensionales que serían difíciles de capturar en un modelo relacional tradicional, además de ofrecer ventajas significativas en cuanto a eficiencia.

El grafo contiene dos tipos principales de nodos: nodos `CONTACT` (uno por cada contacto del usuario) y nodos `CONCEPT` que representan entidades abstractas o concretas. Ambos tipos de nodos tienen una etiqueta asociada que representa el tipo de concepto (Hobby, Empresa, Interés, Ubicación, etc.). Las aristas conectan estos nodos mediante relaciones semánticamente tipadas (afiliación, preferencia, espacial, social, etc.)

- `batch_add_info_to_graph`: Permite añadir múltiples conceptos (intereses, hobbies, empresas, etc.) al grafo de un contacto en una única invocación. Para cada concepto, aplica la lógica de matching semántico para determinar si el concepto debe reutilizar un nodo ya existente o crear uno nuevo.
- `batch_delete_semantic_nodes`: Elimina múltiples nodos del grafo en una sola operación. Las aristas asociadas se eliminan automáticamente por integridad referencial. Solo opera sobre nodos pertenecientes al usuario especificado.
- `get_contact_context_from_graph`: Recupera el grafo completo de conocimiento de un contacto, incluyendo todos los nodos conectados y sus tipos de relación. Indica además si cada nodo es compartido con otros contactos mediante un campo específico denominado `is_shared`.
- `find_shared_connections_for_contact`: Identifica todos los contactos que comparten contexto con un contacto específico. Busca nodos v tales que $weight(v) > 1$ (es decir, su peso es mayor que uno, lo que indica que están conectados a múltiples contactos) y retorna qué otros contactos comparten cada nodo. Esta es la funcionalidad que permite descubrir conexiones no evidentes entre contactos.
- `search_semantic_nodes`: Permite buscar nodos existentes en el grafo del usuario filtrando por categoría (Hobby, Empresa, Interés, etc.) o por coincidencia parcial en la etiqueta. Útil para explorar el grafo o verificar la existencia de conceptos antes de añadirlos.

Matching Semántico

El sistema utiliza búsqueda vectorial para determinar si existe relación semántica con conceptos ya existentes, o si el concepto nuevo debe reutilizar un nodo ya existente o crear uno nuevo.

Para cada concepto entrante, se genera un *embedding* combinando tres atributos: la etiqueta del nodo, su categoría y el tipo de relación (usando el modelo *text-embedding-3-small* de OpenAI). Este vector se almacena en PostgreSQL y se compara con los nodos existentes mediante *pgvector*, calculando la *similitud coseno* s entre el embedding entrante y los ya almacenados. El comportamiento del matching se define formalmente mediante la función:

$$\texttt{find\_semantic\_match} : [0,1] \to \mathcal{A} \quad \text{donde}$$

$$\texttt{find\_semantic\_match}(s) = \begin{cases} \text{REUTILIZAR}(n^*) & \text{si } s \geq 0{,}90 \\ \text{CREAR} + \text{RELACIONAR}(n^*) & \text{si } 0{,}47 \leq s < 0{,}90 \\ \text{CREAR} & \text{si } s < 0{,}47 \end{cases}$$

Donde $n^* = \arg\max_{n \in \mathcal{N}} \text{sim}(\text{e}_{\text{nuevo}}, \text{e}_n)$ es el nodo con mayor similitud, $\mathcal{N}$ es el conjunto de nodos semánticos existentes, y $\mathcal{A} = \{\text{REUTILIZAR}, \text{CREAR}, \text{RELACIONAR}\}$ es el conjunto de acciones. En el primer caso (*match exacto*), se incrementa el peso del nodo n^* y se establece la relación desde el contacto. En el segundo caso (*match relacionado*), se crea un nodo nuevo con una arista `RELACIONADO_CON` hacia n^*, preservando la conexión semántica. Y en el tercer caso, se crea un nodo nuevo sin conexiones adicionales.

Esta estrategia permite descubrir conexiones compartidas entre contactos mediante el análisis de caminos en el grafo: dos contactos C_1 y C_2 están conectados cuando comparten un nodo adyacente común (escenario 1, $d(C_1, C_2) = 2$) o cuando sus nodos están enlazados mediante una arista `RELACIONADO_CON` (escenario 2, $d(C_1, C_2) = 3$), priorizando las conexiones de menor distancia como indicadores de mayor afinidad. Siendo $d(C_1, C_2)$ la distancia o camino más corto entre los contactos en el grafo.

3.3.3. Modelado de Datos

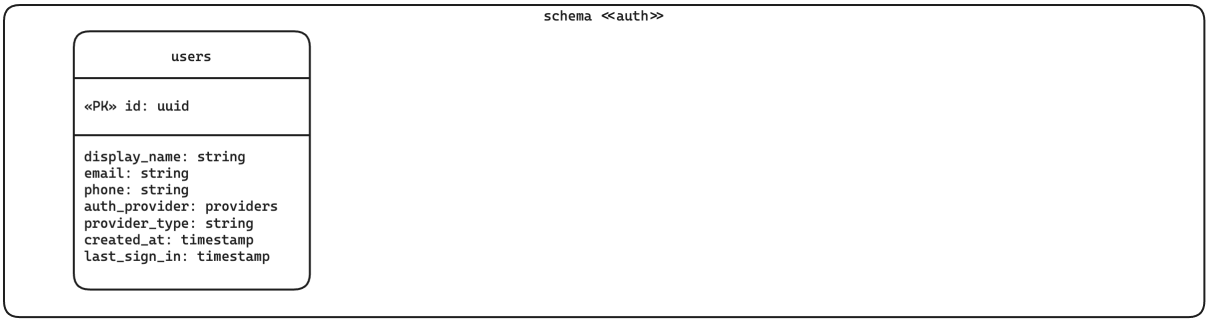

Figura 3.5. Diagrama del modelo de datos de PANOT del schema de Supabase *«auth»*. Solo se incluye la tabla *auth.users* ya que es la unica tabla de el schema *«auth»* que se ha modificado.

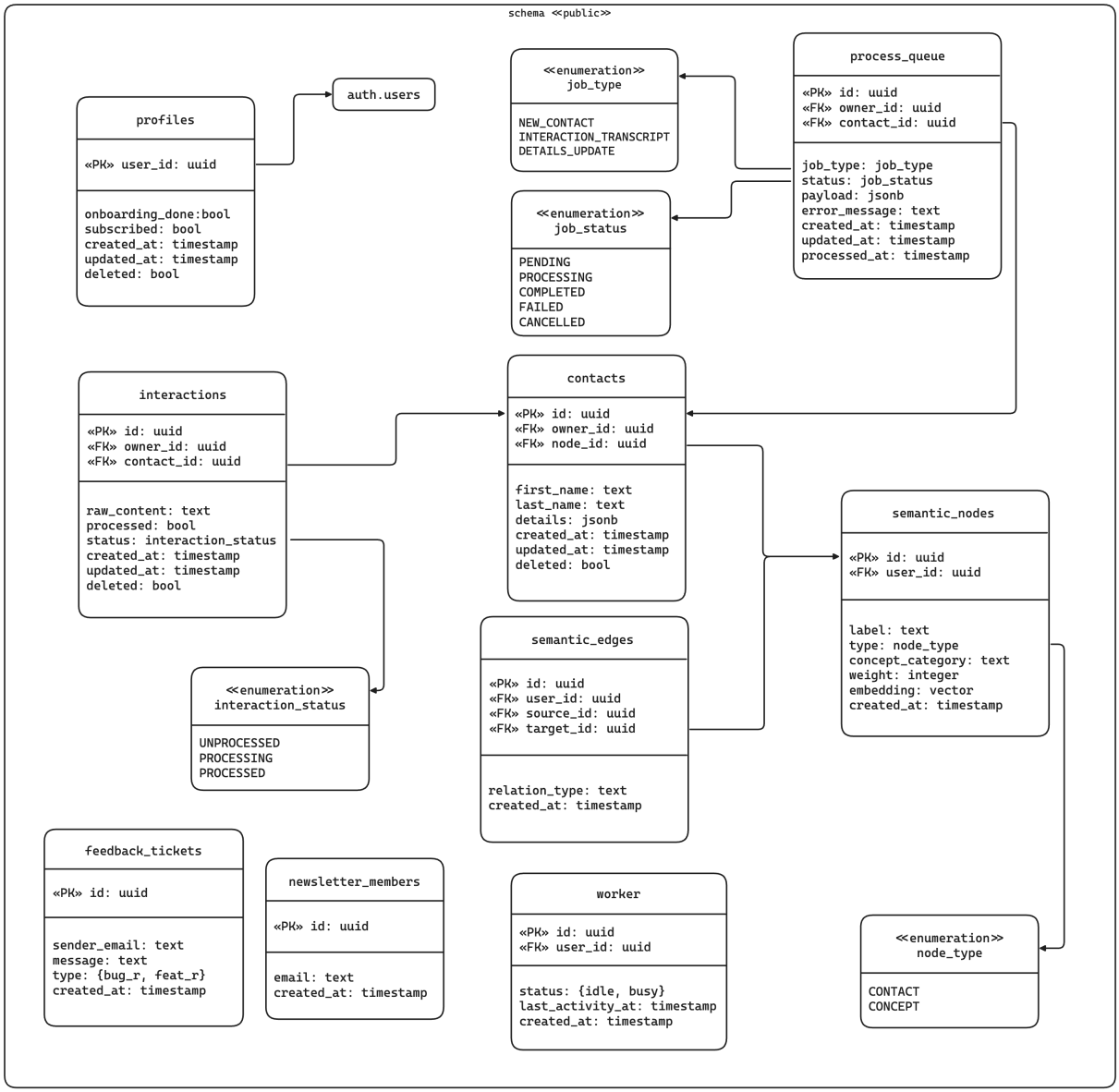

Figura 3.6. Diagrama del modelo de datos de PANOT del schema de Supabase *«public»*.

El modelo de datos de PANOT se organiza en dos schemas: *«auth»* (gestionado por Supabase) y *«public»* (lógica de negocio). A continuación, se explicarán las entidades del schema *«public»*:

Entidades Principales

- `profiles`: Información del perfil de usuario de la aplicación. Cada registro está vinculado a un usuario del schema *«auth»* mediante `user_id`. Contiene el estado del onboarding y si el usuario tiene suscripción activa o no.

- `contacts`: Contactos creados por el usuario. Cada contacto tiene un `owner_id` que indica su propietario (usuario creador) y un `node_id` que lo vincula con su nodo correspondiente en el grafo semántico correspondiente. El campo `details` almacena en formato JSON la descripción y detalles del contacto, generado manualmente por el usuario o automaticamente por el sistema.
- `interactions`: Transcripciones de voz grabadas por el usuario. El campo `raw_content` contiene el texto transcrito. Cada interacción puede estar asociada a un contacto mediante `contact_id` o permanecer sin asignar. El campo `status` indica el estado de procesamiento, que puede ser:
  - `UNPROCESSED`: pendiente de procesar,
  - `PROCESSING`: en proceso,
  - `PROCESSED`: completado.

Adicionalmente, se ha incorporado en todas las entidades un campo `deleted` que implementa el patrón *soft delete*. Este mecanismo permite marcar los registros como eliminados de manera lógica en lugar de borrarlos físicamente, lo que facilita la recuperación ante eliminaciones accidentales y preserva la integridad referencial entre entidades relacionadas. Cabe añadir que este mecanismo ha sido implementado de cara a futuras iteraciones.

Grafo Semántico

El conocimiento contextual de los contactos se modela como un grafo almacenado en PostgreSQL, compuesto por dos entidades:

- `semantic_nodes`: Nodos del grafo. El campo `type` distingue entre dos tipos: `CONTACT` (nodo raíz creado automáticamente para cada contacto) y `CONCEPT` (información contextual como intereses, hobbies, ubicaciones o emociones). El campo `weight` indica cuántas conexiones tiene el nodo; un valor mayor a uno señala que es compartido por múltiples contactos. El campo `embedding` almacena el vector generado para búsqueda por similitud semántica. Y el campo `concept_category` clasifica el tipo de concepto (Hobby, Interés, Empresa, Emoción, etc.).
- `semantic_edges`: Aristas del grafo. Representa la relación semántica entre dos nodos. Conecta un nodo origen con un nodo destino (`source_id` → `target_id`) mediante un tipo de relación (`relation_type`).

Se ha optado por implementar el grafo directamente en PostgreSQL (*graph-on-SQL*) en lugar de utilizar una base de datos de grafos dedicada. La razón principal de esta decisión es evitar la complejidad operativa de mantener múltiples motores de bases de datos: un único sistema simplifica el despliegue, las copias de seguridad, la monitorización y el mantenimiento general de la infraestructura. Además, para grafos de escala personal (cientos a miles de nodos por usuario), PostgreSQL ofrece un rendimiento más que suficiente.

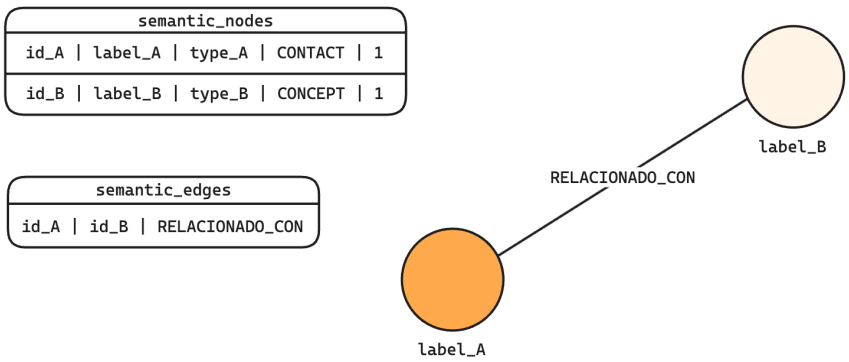

Figura 3.7. Ejemplo de grafo semántico en PostgreSQL. En el se pueden ver dos nodos: un primer nodo A de tipo CONTACT y un segundo nodo B de tipo CONCEPT. Ambos estan conectados con una relación de tipo RELACIONADO_CON.

Esta elección trae consigo ventajas adicionales: la integración nativa con Supabase y el resto del modelo de datos, el soporte de búsqueda vectorial mediante la extensión `pgvector` (necesaria para el matching semántico), y la posibilidad de ejecutar operaciones sobre el grafo y las entidades relacionales en una misma transacción, garantizando la consistencia de los datos.

Sistema de Procesamiento

Como se describió en la visión general de la arquitectura, PANOT implementa un modelo de ejecución basado en cola de trabajos que combina paralelismo entre usuarios con procesamiento secuencial por usuario. Este patrón se materializa en dos entidades:

- `process_queue`: Cola de trabajos pendientes. Cada registro representa una tarea a procesar, identificada por su `job_type`:
  - `NEW_CONTACT`: procesar nuevo contacto.
  - `INTERACTION_TRANSCRIPT`: analizar transcripción.
  - `DETAILS_UPDATE`: actualizar resumen del contacto.

  El campo `status` indica el estado del trabajo:
  - `PENDING`: pendiente de procesar.
  - `PROCESSING`: en proceso.
  - `COMPLETED`: completado.
  - `FAILED`: fallido.
  - `CANCELLED`: cancelado.

  El campo `contact_id` almacena opcionalmente el identificador del contacto cuando la operación actúa sobre uno específico. El campo `payload` contiene en formato JSON los datos específicos necesarios para que el sistema Multi-Agente ejecute la tarea. Y en caso de error, el campo `error_message` almacena la descripción del fallo.
- `worker`: Representa el worker dedicado de cada usuario. Cada usuario tiene exactamente un worker (restricción `UNIQUE` en `user_id`). El campo `status` indica el estado:
  - `idle`: disponible.
  - `busy`: procesando una tarea.

El campo `last_activity_at` registra la última actividad, permitiendo detectar workers bloqueados o inactivos.

A continuación, se presenta el diagrama de actividad de este sistea de procesamienton por cola. Para mantener la simplicidad se han obviado datos como actualización de tiempos (`last_activity_at`, `processed_at`, `created_at` y `updated_at`) y el tipo de `PAYLOAD` usado para invocar a `relational-agent` en función del tipo de trabajo (`job_type`). Para ver cómo funciona el sistema Multi-Agente, se puede consultar el apartado 3.3.2 anterior.

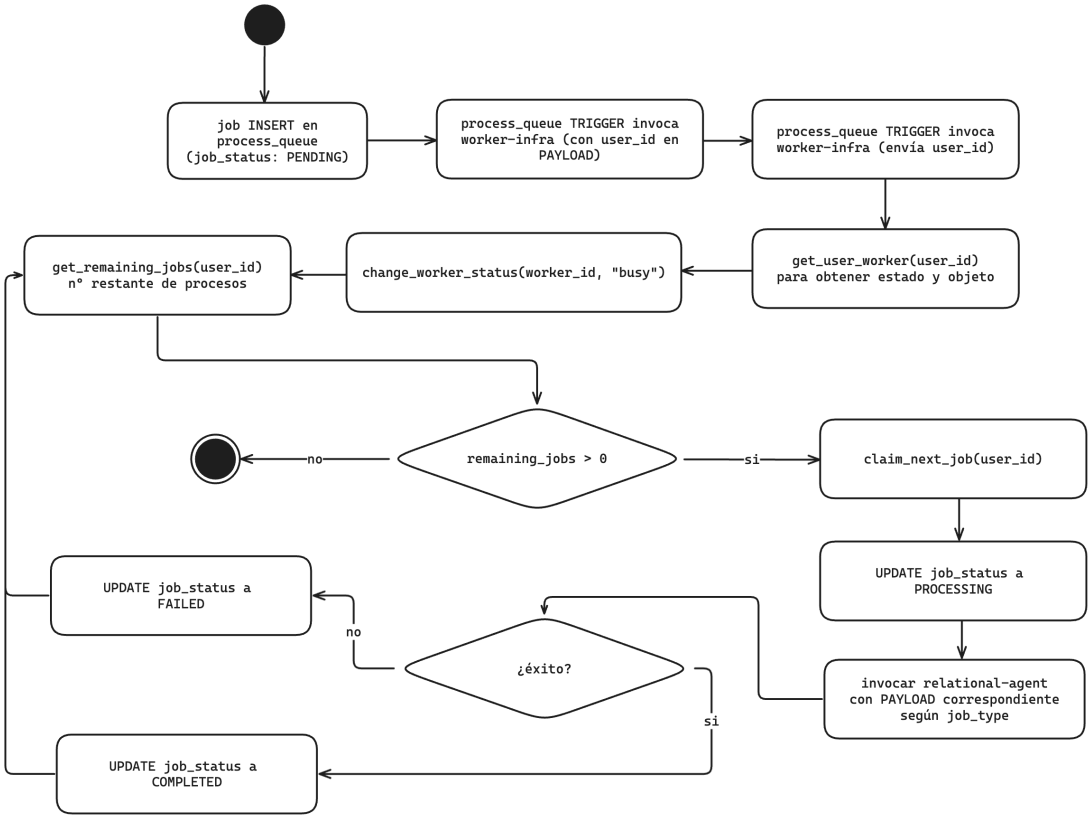

Figura 3.8. Diagrama de actividad del sistema de procesamiento de PANOT

El diagrama de actividad anterior (3.8) se centra en el flujo interno de la cola de trabajos. Para comprender el flujo completo de procesamiento, es necesario considerar todas las entidades y artefactos involucrados en el sistema. A continuación, se presenta un diagrama de secuencia que muestra la interacción entre el cliente móvil, la base de datos, el sistema de cola de trabajos (`process_queue` y `worker-infra`) y el sistema Multi-Agente (`relational-agent`):

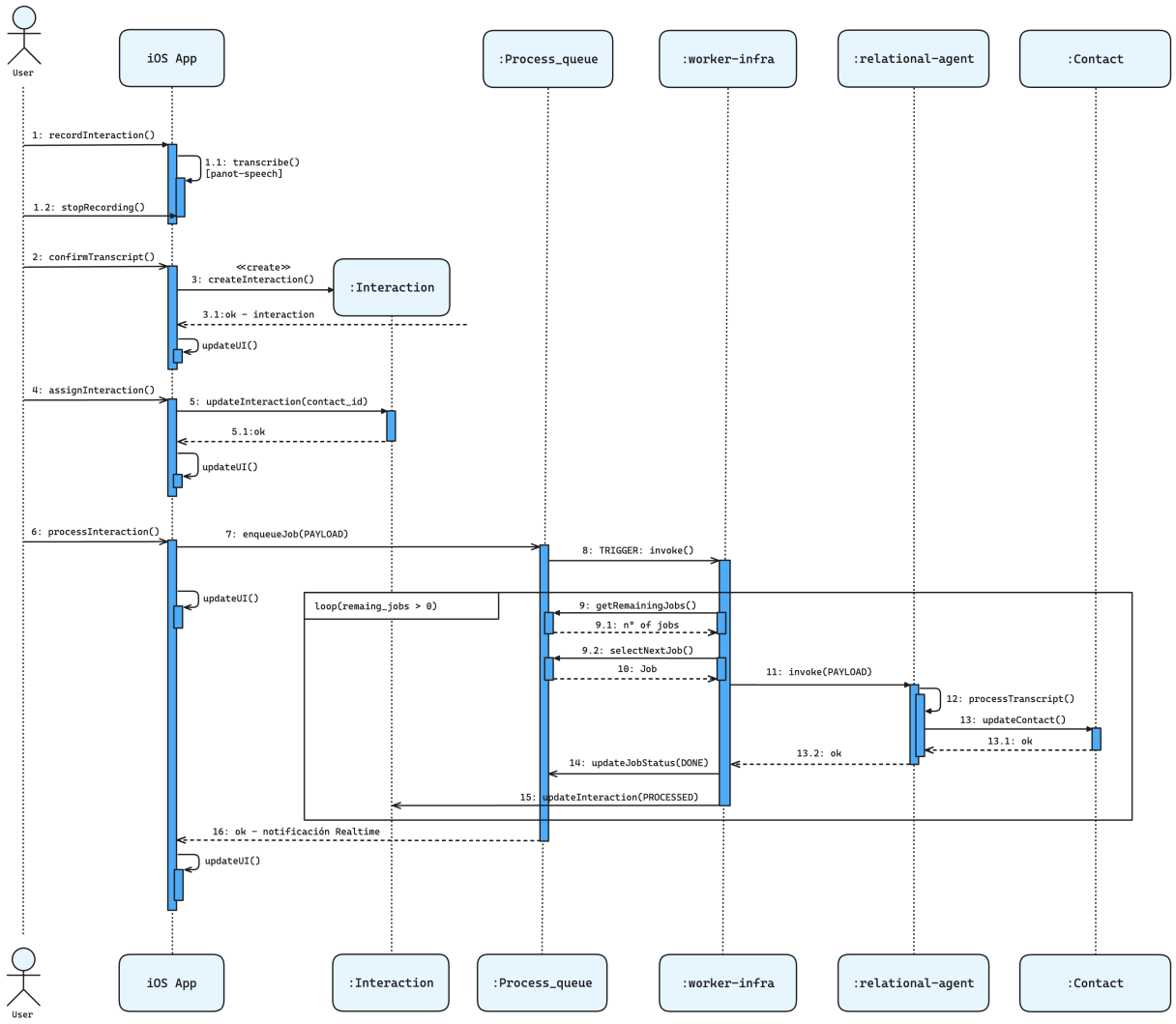

Figura 3.9. Diagrama de secuencia del procesamiento de una Interacción grabada por el usuario

3.4. Fases de la Implementación Iterativa

[Escribir aqui las fases de la implementacion]

3.5. Despliegue y Lanzamiento

[Escribir aqui el despliegue y lanzamiento de PANOT]

4. Verificación y Resultados

[Escribir aquí los resultados y evaluación del proyecto]

5. Conclusiones y Trabajo Futuro

5.1. Conclusiones Generales

[Escribir aquí las conclusiones generales]

5.2. Aplicación de Conocimientos Adquiridos en el Grado

[Escribir aquí la aplicación de conocimientos adquiridos en el grado]

5.3. Líneas de Trabajo Futuro

[Escribir aquí las líneas de trabajo futuro]

Referencias

[1] L. A. A. Doumas, G. Puebla, A. E. Martin y J. E. Hummel, «A theory of relation learning and cross-domain generalization,» *Edinburgh Research Explorer*, 2022.

[2] Agencia Española de Protección de Datos, *Guía de Privacidad desde el Diseño*, Agencia Española de Protección de Datos, oct. de 2019.

[3] R. C. Martin, *Clean Architecture*. Addison-Wesley, 2017.

[4] E. Sicilia García M.A. y Amador Domínguez, «PROVISIÓN DE MODELOS PREDICTIVOS EN EL SECTOR SALUD CON PRESERVACIÓN DE LA PRIVACIDAD,» E.T.S. de Ingeniería de Sistemas Informáticos, Proyecto Fin de Grado, jul. de 2025.

[5] E. Ries, *The Lean Startup*. Crown Currency, 2011.

[6] D. T. J. y James P. Womack, *Lean Thinking*. Gestión 2000, 2012.

Índice de términos

Glosario

AES-256 Estándar de Cifrado Avanzado (Advanced Encryption Standard) que utiliza una longitud de clave de 256 bits. Es uno de los algoritmos de cifrado simétrico más seguros y utilizados actualmente para proteger información clasificada y datos sensibles. 15

Análisis de Cohorte Técnica de análisis de datos que divide a un grupo de individuos en subgrupos basados en características comunes y observa cómo evolucionan estos individuos a lo largo del tiempo. 23

API Gateway Patrón de diseño que actúa como punto de entrada único (entry point) para un sistema distribuido, encargándose de desacoplar al cliente de la implementación interna mediante la gestión centralizada del enrutamiento, la seguridad y la composición de protocolos. 18, 19

Claves Efímeras Clave temporal de corta duración que permite a aplicaciones cliente acceder de forma segura a datos limitados de un cliente temporal (Customer) sin exponer claves secretas. 34

Clientes Temporales Objeto que representa a un cliente en Stripe, permitiendo asociar métodos de pago guardados, historial de transacciones y suscripciones. 34

Column Level Security Mecanismo de seguridad que permite controlar el acceso a las columnas de una tabla de base de datos según el usuario que realiza la consulta. 15

Construir-Medir-Aprender Ciclo de retroalimentación fundamental de la metodología *Lean Startup* diseñado para transformar ideas en productos (Construir), evaluar la reacción de los clientes mediante datos (Medir) y validar o refutar hipótesis para decidir si perseverar o pivotar (Aprender), con el objetivo de acelerar el aprendizaje validado y minimizar el desperdicio. 15, 24, 25

Eliminar Desperdicio Principio de Lean Development que se basa en eliminar cualquier esfuerzo o característica que no genere valor para el usuario. 25, 27

Embudos Técnica de análisis de datos que permite visualizar cómo los usuarios interactúan con diferentes partes de un producto a lo largo del tiempo, ayudando a identificar patrones de retención y conversión. 23

Generalización de Cero Disparos Escenario de aprendizaje automático en el que se entrena un modelo de IA para reconocer y categorizar objetos o conceptos sin haber visto ningún ejemplo de esas categorías o conceptos de antemano. 3

Inferencia Analógica Transmisión de conocimientos de una situación a otra. 3

Intenciones de Pago Objeto de Stripe que representa una intención de cobro. Contiene el monto, moneda y estado del pago, gestionando todo el ciclo de vida de la transacción. 34

Optimizar el Todo Principio de Lean Development que se basa en la idea de enfocarse en mejorar el proceso completo en lugar de solo partes aisladas. 25

Overhead Sobrecarga o coste adicional en tiempo, recursos computacionales o complejidad que introduce un sistema, proceso o abstracción más allá del trabajo estrictamente necesario para completar una tarea. 35

PCI DSS Standard de seguridad que recoge una serie de normas obligatorias para cualquier organización que procese, acepte, almacene o transite datos de tarjetas de crédito. 21

Row Level Security Mecanismo de seguridad que permite controlar el acceso a las filas de una tabla de base de datos según el usuario que realiza la consulta. 15

SOC 2 Type 2 Estándar de auditoría desarrollado por el AICPA (Instituto Americano de Contadores Públicos Certificados) que revisa y verifica tanto el diseño como el funcionamiento efectivo de los controles de seguridad implementados por una organización para salvaguardar los datos de sus clientes durante un intervalo determinado, generalmente comprendido entre 3 y 12 meses. 15

Transport Layer Security Protocolo de seguridad que proporciona autenticación, integridad y confidencialidad de los datos en tránsito entre dos puntos, generalmente entre un cliente y un servidor. 15

Vendor Lock-In Situación en la que un cliente se vuelve dependiente de un proveedor de productos o servicios, siendo incapaz de migrar a otro proveedor sin incurrir en costes sustanciales de cambio, incompatibilidades técnicas o pérdida de funcionalidad. 34, 35

WebSocket Protocolo de comunicación bidireccional y full-duplex sobre una única conexión TCP que permite el intercambio de mensajes en tiempo real entre cliente y servidor sin necesidad de realizar múltiples peticiones HTTP. 34

Siglas

API Interfaz de Programación de Aplicaciones 16–21, 33, 35

CRM Sistema de Gestión de Relaciones con Clientes 2

IA Inteligencia Artificial 3, 24, 35

JWT JSON Web Token 19, 34

OTP Contraseña de Un Solo Uso (One-Time Password) 34

PMV Producto Mínimo Viable 24–26, 28

PRM Gestión de Relaciones Personales 2

SDK Kit de Desarrollo de Software 21, 23

WIP Work In Progress 27, III