

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, 1 de febrero 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, 1 de febrero 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 = {2},  
}
```

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	2
2.1	Gestión de la Inteligencia Relacional	2
2.2	Cambio de Paradigma en el Diseño de Producto	7
2.3	Principios de Privacidad y Eficiencia por Diseño	9
3	Contexto Tecnológico	15
3.1	Capa del Cliente Móvil	15
3.2	Capa de Datos y Persistencia Local	16
3.3	Servicios Externos	17
4	Desarrollo del Proyecto	23
4.1	Filosofía y Metodología de Desarrollo	23
4.2	Definición del PMV y Especificación de Requisitos	26
4.3	Diseño del Sistema	32

4.4	Implementación del Sistema	46
4.5	Despliegue y Lanzamiento de PANOT	55
5	Resultados y Verificación	59
5.1	Interfaz de Usuario y Flujos del Sistema	59
5.2	Verificación de Requisitos	69
5.3	Cumplimiento de Objetivos	70
6	Conclusiones y Trabajo Futuro	71
6.1	Conclusiones Generales	71
6.2	Aplicación de Conocimientos Adquiridos en el Grado	71
6.3	Líneas de Trabajo Futuro	71
A	Evidencias de Verificación de Requisitos No Funcionales	72
A.1	RNF-SEG-001 - Seguridad	72
A.2	RNF-REND-002 y RNF-EFI-003 - Rendimiento y Eficiencia	76
A.3	RNF-MANT-008 - Mantenibilidad	80
	Glosario	83
	Siglas	85

Índice de figuras

2.1 Representación orientativa usando MemGraph Lab del contexto del contacto «María» en formato de grafo relacional.	6
3.1 Arquitectura API Gateway simplificada	18
3.2 Comparativa de modelos mencionados de OpenAI. Tabla sacada de la documentación de OpenAI [7].	20
4.1 Diagrama orientativo de la estrategia de ramificación. (1) Aislamiento: Creación de una rama efímera (<i>feature</i>) a partir de desarrollo para trabajar en un <i>Issue</i> específico. (2) Implementación: Desarrollo de la funcionalidad mediante <i>commits</i> atómicos. (3) Integración: Fusión de la rama mediante un <i>Pull Request</i> validado, cerrando el ciclo de desarrollo. (4) Release: Promoción del código estable desde desarrollo a la rama principal (<i>main</i>) para su despliegue en producción.	25
4.2 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 <i>In Progress</i> (indicador 1/1), forzando un flujo continuo y eliminando el desperdicio asociado al cambio de contexto.	26
4.3 Arquitectura general del sistema PANOT	32
4.4 Diagrama de la arquitectura del sistema multi-agente de PANOT	36

4.5 Diagrama del modelado de datos de PANOT con los schemas de Supabase « <i>public</i> » y « <i>auth</i> ». En el diagrama del schema « <i>auth</i> » solo se ha incluido la tabla <i>auth.users</i> ya que es la unica tabla de el schema que se ha modificado. Y en el diagrama del schema « <i>public</i> » se ha obviado del diagrama que las claves foráneas <i>user_id</i> o <i>owner_id</i> de las tablas provienen del <i>id</i> de la tabla <i>users</i> del schema « <i>auth</i> ». A modo de ejemplo, solo se ha añadido en la tabla <i>profiles</i> para mantener el diagrama más legible.	40
4.6 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
4.7 Diagrama de actividad del sistema de procesamiento de PANOT	44
4.8 Diagrama de secuencia del procesamiento de una Interacción grabada por el usuario	45
4.9 Repositorios de la organización panot-hq	46
4.10 Gráfico de burndup de issues cerrados en el proyecto de PANOT	48
4.11 Issues cerradas en octubre de 2025	49
4.12 Árbol de commits de octubre (y finales de septiembre) de 2025 en repositorio Git local de panot-hq/mobile	50
4.13 Issues cerradas en noviembre de 2025	51
4.14 Árbol de commits de noviembre de 2025 en repositorio panot-hq/mobile local	51
4.15 Issues cerradas en diciembre de 2025	52
4.16 Ejemplo de traza de ejecución en LangSmith del sistema multi-agente de PANOT	53
4.17 Árbol de commits de diciembre de 2025 (y principios de enero de 2026) en repositorio Git local de panot-hq/mobile	54

4.18 Issue cerrada en enero de 2026 (captura tomada a mediados de enero de 2026)	54
4.19 Captura del mail de Apple confirmando la aprobación de la App Review	57
4.20 Captura del historial de envíos de la aplicación a App Review	57
5.1 Flujo de Registro de Usuario y Onboarding	59
5.2 Flujo de Inicio de Sesión	60
5.3 Flujo de Creación de Contacto	61
5.4 Traza de ejecución en LangSmith para la creación del contacto Pedro. . .	62
5.5 Grafo de conocimiento resultante de la creación del contacto Pedro en graph-visualizer.	62
5.6 Grafo de conocimiento resultante de la creación del contacto Pepe y la detección de nodos comunes con el contacto Pedro en graph-visualizer.	63
5.7 Flujo de Grabación y Procesamiento de Interacción (I)	64
5.8 Flujo de Grabación y Procesamiento de Interacción (II)	65
5.9 Grafo de conocimiento resultante de los contactos Pepe y Pedro tras procesar las nuevas interacciones en graph-visualizer.	65
5.10 Flujo de Edición o Eliminación de Contacto	66
5.11 Flujo de Búsqueda de Contacto	66
5.12 Flujo de Soporte y Feedback	67
5.13 Verificación de persistencia: Filas de la base de datos con los tickets de soporte enviados.	67
5.14 Flujo de Configuración y Ajustes	68
A.1 Política RLS para las tablas semantic_edges y semantic_nodes. . .	72
A.2 Política RLS para la tabla interactions.	73

A.3 Política RLS para la tabla <code>profiles</code>	73
A.4 Política RLS para la tabla <code>contacts</code>	74
A.5 Política RLS para la tabla <code>workers</code>	74
A.6 Política RLS para la tabla <code>process_queue</code>	75
A.7 Política RLS para la tabla <code>feedback_tickets</code>	75
A.8 Lista de trazas de ejecución en LangSmith.	76
A.9 Gráfica de latencia por traza en LangSmith. Listado de valores de latencia (T_{e2e}): 28.89; 37.34; 34.37; 46.75; 34.03; 40.45; 30.48; 29.06; 30.60; 31.19; 34.84; 31.96; 35.06; 32.99; 35.42; 37.17; 28.09; 45.99; 32.48; 29.69; 34.50; 7.75; 32.07; 32.42; 31.39; 34.37; 37.60	78
A.10 Gráfica de coste por traza en LangSmith. Listado de valores de coste (C_i) en USD: 0.0044; 0.0042; 0.0052; 0.0046; 0.0049; 0.0056; 0.0044; 0.0052; 0.0048; 0.0047; 0.0045; 0.0045; 0.0057; 0.0047; 0.0050; 0.0055; 0.0051; 0.0080; 0.0054; 0.0047; 0.0054; 0.0012; 0.0045; 0.0052; 0.0051; 0.0060; 0.0060.	79
A.11 Flujo de eventos recibido en el panel de PostHog.	81

Índice de tablas

4.1	Definición de las Historias de Usuario para el PMV de PANOT	28
4.2	Features del Sistema y los IDs de los Requisitos Funcionales Asociados .	28
4.3	Requisitos Funcionales de la <i>Feature 1 - Gestión de Usuario</i>	29
4.4	Requisitos Funcionales de la <i>Feature 2 - Gestión de Contactos</i>	29
4.5	Requisitos Funcionales de la <i>Feature 3 - Gestión de Interacciones</i>	29
4.6	Requisitos Funcionales de la <i>Feature 4 - Inteligencia Relacional</i>	29
4.7	Requisitos Funcionales de la <i>Feature 5 - Soporte y Feedback</i>	30
4.8	Matriz de Trazabilidad de Requisitos e Historias de Usuario	30
4.9	Requisitos No Funcionales del sistema: ID, Tipo, Descripción y Criterio de Aceptación	31
4.10	Ejes de desarrollo de PANOT	47
5.1	Tabla de verificación del cumplimiento de Requisitos Funcionales. . . .	69
5.2	Tabla de verificación del cumplimiento de Requisitos No Funcionales. .	70
A.1	Eventos de PostHog capturados en la aplicación	80

Índice de listados

2.1 Possible representación relacional estructurada del contacto «María»	5
A.1 Implementación de abstracción de eventos con anonimización de metadatos.	81

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

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 tabular sin considerar su evolución temporal ni su contexto situacional.

2.1.1. Aplicaciones Similares

El paradigma actual de la gestión de contactos ha evolucionado desde simples agencias digitales, a lo que ahora se denominan sistemas de *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 insercción de datos, su enfoque es fundamentalmente 'archivístico'. Al operar sobre este tipo de fuentes de datos, 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 matiz emocional tuvo la última conversación.

El vacío que se considera que estas aplicaciones no logran cubrir radica en la diso-

nancia 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 el registro de *Inteligencia Relacional*.

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 específico 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 Cero 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 completa-

mente 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 podría construir representaciones relacionales dinámicas que capturan la estructura subyacente de las relaciones del usuario que use la plataforma.

Para ilustrar este proceso, consideremos un ejemplo práctico del flujo de procesamiento relacional que realizaría el sistema de PANOT:

Entrada: El usuario capture la siguiente interacción mediante una 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 trabajar juntos en el futuro.”

Procesamiento: PANOT procesa esta entrada 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 colaboración en el futuro
- *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 una posible representación en formato JSON:

Listado 2.1. Posible representación relacional estructurada del contacto «María».

```
{  
  "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"  
  }  
}
```

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

- *Contexto Personal:* Se actualiza la información sobre su situación personal con “búsqueda de vivienda” y su preferencia de ubicación, además de capturar el estado emocional positivo asociado al cambio de trabajo.
- *Contexto Profesional:* Se modifica la información sobre su situación profesional 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 modifica la información sobre 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. abstrayéndo el complejo contexto de la relación en una representación más simplificada 2.1.

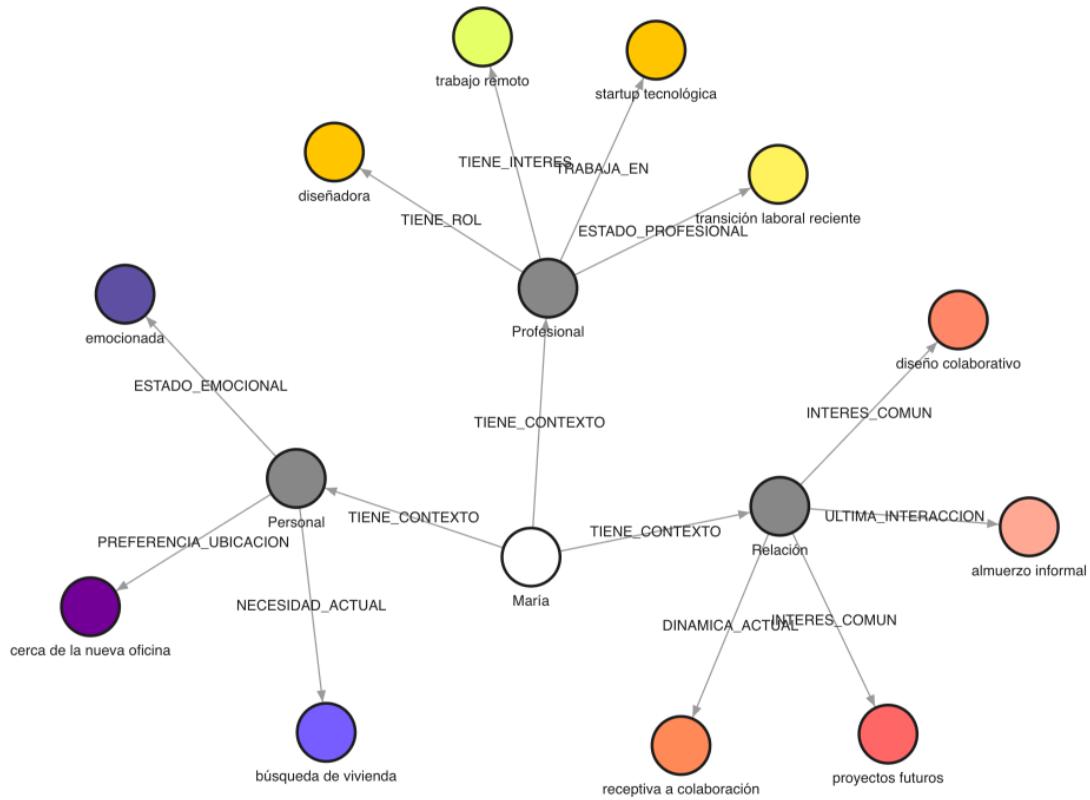


Figura 2.1. Representación orientativa usando MemGraph Lab del contexto del contacto «María» en formato de grafo relacional.

2.1.3. 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.

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. Fíjese 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 procesamiento de los modelos de lenguaje ha generado un desplazamiento del valor diferencial de los productos: ya no es suficiente ofrecer una funcionalidad única, 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.

Es decir, 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.

Con esta premisa, PANOT se diseña con una orientación centrada en el usuario y su contexto, buscando una experiencia adaptada a sus necesidades. Aunque en su versión actual no despliegue una experiencia de interfaz única para cada usuario ni perfiles de personalización explícitos, se alinea con los valores diferenciadores anteriores en dos frentes: en *conexión y resonancia emocional*, al posicionar la aplicación como extensión cognitiva de las relaciones del usuario ya que el sistema acompañaría la evolución de cada contacto del usuario y su contexto relacional, y en *comprensión contextual*, al inferir de las interacciones capturadas del usuario patrones, intereses y necesidades de sus contactos sin exigirle configuraciones manuales.

2.3. Principios de Privacidad y Eficiencia por Diseño

El desarrollo de sistemas de software modernos, especialmente aquellos que procesan información personal y utilizan 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 32^a 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 en productos y servicios de software.

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:* 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, despliegue, 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. *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 las siguientes estrategias de diseño:

1. *Minimizar:* Recopilar únicamente los datos estrictamente necesarios.
2. *Ocultar:* Proteger la información sensible empleando técnicas de ofuscación o desvinculación.
3. *Separar:* Gestionar los datos de forma que se eviten correlaciones indebidas o perfiles del usuario.
4. *Abstraer:* Limitar el nivel de detalle de los datos personales tratados.
5. *Informar:* Comunicar de manera clara a los usuarios sobre el tratamiento de sus datos.
6. *Controlar:* Garantizar un adecuado control de acceso y uso de la información, así como permitir a los usuarios gestionar la recogida, tratamiento, usos y comunicaciones de sus datos personales.
7. *Cumplir:* Asegurar que los tratamientos de datos personales se llevan a cabo conforme a los procedimientos y requisitos legales aplicables.
8. *Demostrar:* Según el artículo 24 de la LGPD, poder evidenciar el cumplimiento normativo a través de documentación y auditorías, tanto frente a los usuarios como ante las autoridades de supervisión.

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 diseño, 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, sostenibles y modulares, 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 micro-servicios 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 Network) en empresas como Amazon Web Services permite reducir la latencia y el consumo de ancho de banda mediante el almacenamiento de contenido 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, 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 infraestructuras 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* ([FHE](#)), un paradigma criptográfico que, diferencia del cifrado tradicional, donde los datos deben descifrarse antes de ser procesados, 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 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 [FHE](#) 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.

Otras [TPP](#) 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 fun-

damental es la adopción de *arquitecturas serverless* y *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*.

2.3.4. Aplicación en PANOT

El caso del sistema 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 y contextos situacionales 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 requiere de proteger la información del usuario 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, como se explicará en el siguiente capítulo, se ha utilizado *Supabase* como plataforma de backend. *Supabase* tiene la particularidad de autogestionar 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 *arquitecturas serverless* y 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 del sistema 4.1.

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 mientras se optimiza el consumo de recursos computacionales y económicos asociados con el alojamiento de la infraestructura y el uso de servicios externos.

3.

Contexto Tecnológico

La elección de tecnologías para el desarrollo del proyecto se ha realizado teniendo en cuenta una sinergia coherente entre mantenibilidad, seguridad, escalabilidad, facilidad de integración, rapidez de implementación y coste operacional.

3.1. 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 las funcionalidades. Además *Expo* es un framework basado en *React*, lo que facilita el desarrollo modular, la reutilización de componentes y el tipado de los artefactos del proyecto debido a la naturaleza de este lenguaje.
- 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, o almacenamiento local, sin requerir la complejidad del desarrollo nativo puro.

Se analizaron otras alternativas para el desarrollo del cliente móvil: en primer lugar, el desarrollo nativo con *Swift* y *SwiftUI*, que habría ofrecido un rendimiento mayor y acceso completo a las capacidades de iOS, pero se descartó por su mayor complejidad

de configuración, tiempo de desarrollo más extenso y limitación a una única plataforma, lo que no se alineaba con los objetivos de eficiencia y escalabilidad del proyecto. Asimismo, se consideró *Flutter*, un framework desarrollado por *Google* que permite el desarrollo de aplicaciones multiplataforma con *Dart* y compilación a código nativo, similar a *Expo*, pero también se descartó por la poca familiaridad con este lenguaje.

3.2. 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 contactos de manera continua independientemente de si tienen conexión o no. 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 nativas de almacenamiento local proporcionadas por *Expo*. *Legend State* proporciona un sistema de observables que permite mantener un estado global de los datos sincronizado entre los componentes de la aplicación, mientras que *Expo* ofrece [APIs](#) para el almacenamiento persistente en el dispositivo mediante *AsyncStorage*.

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*. Esta operación es instantánea y no requiere conexión a internet. Posteriormente, en segundo plano, la aplicación intenta sincronizar estos datos con el servidor de *Supabase* en cuanto se 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, como ya se ha mencionado, *Supabase* como infraestructura 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*.

- 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 mediante [WebSockets](#) (*Supabase Realtime*), permitiendo que actualizaciones realizadas localmente se reflejen automáticamente en la base de datos.
- Seguridad integrada: Como se mencionó en el capítulo anterior [2.3.4](#), *Supabase* proporciona mecanismos de seguridad a nivel de fila y columna, y encriptación de datos en reposo y en tránsito, 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 llamadas 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.

No se han considerado alternativas a *Supabase* para la persistencia, ya que ya se contaba con experiencia previa con esta plataforma y se ha optado por la simplicidad y velocidad de integración, la escalabilidad y la seguridad integrada que ofrece, además del soporte de *Expo* y *React Native*.

3.3. 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* [6], 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 y la demanda de recursos. El flujo de ejecución de una función es el siguiente:

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. De esta manera, se centralizan las comprobaciones de seguridad antes de ejecutar el código, lo que garantiza que solo solicitudes autenticadas y autorizadas accedan a los servicios.

2. Ejecución en el edge runtime: La función se ejecuta en el nodo *Edge Runtime* más cercano al usuario, minimizando la latencia de procesamiento.
3. Respuesta a través del gateway: El [API Gateway](#) reenvía la respuesta al cliente y registra los metadatos de la solicitud para análisis en logs y métricas que pueden explorarse en el Dashboard de *Supabase* o integrarse con sistemas de monitorización como se verá más adelante.

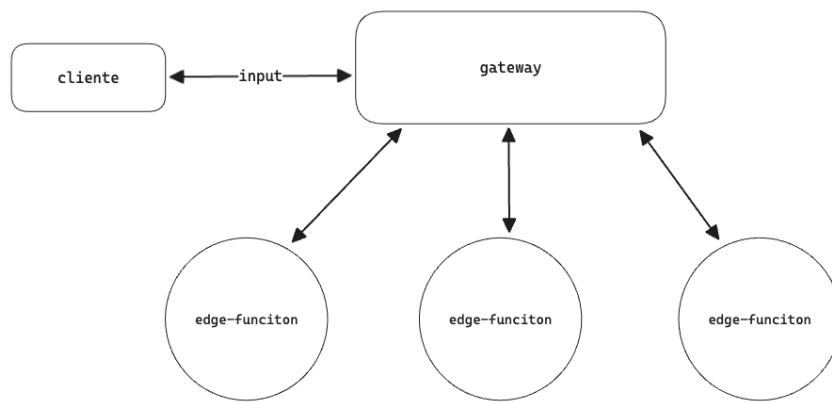


Figura 3.1. Arquitectura [API Gateway](#) simplificada

Esta arquitectura de [API Gateway 3.1](#) de las *Edge Functions* de *Supabase* 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 la latencia total de la solicitud.
- 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.

Alternativas consideradas como las *AWS Lambda* de Amazon, *Google Cloud Functions* de Google y *Azure Functions* de Microsoft se han descartado por su complejidad y coste ya que no se contaba con experiencia previa con estas plataformas y se ha optado por la simplicidad y facilidad de integración que ofrece *Supabase*. Además, elegir las *Edge Functions* de *Supabase* es una decisión resultaba la opción más orgánica y natural para el proyecto teniendo en cuenta el contexto tecnológico de la capa de datos [3.2](#).

3.3.1. Subcapa de Inteligencia Artificial

La subcapa de inteligencia artificial de PANOT, responsable del procesamiento de interacciones y la generación de relaciones contextuales, se ha implementado mediante *Supabase Edge Functions* [3.3](#) que actúan como intermediario entre la aplicación móvil y la *API de OpenAI*, utilizando el modelo gpt-5-mini 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 gpt-5-mini se fundamenta en su equilibrio entre eficiencia, calidad y coste. Este modelo ofrece tiempos de respuesta y costes menores que alternativas más grandes como gpt-5-pro o gpt-5.1, 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. 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.

gpt-5-mini	gpt-5-pro	gpt-5.1			
A faster, cost-efficient version of GPT-5 for well-defined tasks	Version of GPT-5 that produces smarter and more precise responses	The best model for coding and agentic tasks with configurable reasoning effort			
Reasoning	● ● ●	Reasoning	● ● ● ● ●	Reasoning	● ● ● ●
Speed	⚡ ⚡ ⚡ ⚡	Speed	⚡	Speed	⚡ ⚡ ⚡
PRICING	PER 1M TOKENS	PRICING	PER 1M TOKENS	PRICING	PER 1M TOKENS
Input	\$0.25	Input	\$15.00	Input	\$1.25
Cached Input	\$0.03	Cached Input	-	Cached Input	\$0.13
Output	\$2.00	Output	\$120.00	Output	\$10.00

Figura 3.2. Comparativa de modelos mencionados de OpenAI. Tabla sacada de la documentación de OpenAI [7].

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 además de una capa de observabilidad de las peticiones como se verá más adelante. El sistema de agentes que orquesta este procesamiento se ha desarrollado utilizando *LangChain* como framework y su diseño y funcionamiento se detallan en la Sección 4.3.2.

Como alternativas se consideraron los modelos *Claude* de *Anthropic*, que ofrece capacidades avanzadas de razonamiento pero con un coste de computación sustancialmente mayor. No obstante, se decidió elegir *OpenAI*, teniendo en cuenta la experiencia previa que se tenía con la plataforma.

3.3.2. Subcapa de Infraestructura de Pagos

Para la integración de funcionalidades de pago y suscripciones, se ha seleccionado *Stripe* como proveedor de los servicios de pasarela de pagos. *Stripe* es una plataforma Banking-as-a-Service (*BaaS*) 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* que gestionan la creación de intenciones de pago y el procesamiento de transacciones de forma segura, además de integrar [Webhooks](#) para manejar eventos asíncronos como confirmaciones de pago o actualizaciones de métodos de pago, sincronizando automáticamente el estado en la base de datos. Adicionalmente, *Stripe* gestiona automáticamente el cumplimiento de estándares de seguridad como [PCI DSS](#) (Payment Card Industry Data Security Standard) lo que garantiza una mayor seguridad en las transacciones del usuario.

Como alternativa a *Stripe* se evaluó *Polar*, otra plataforma [BaaS](#) 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 para aplicaciones móviles con Kit de Desarrollo de Software ([SDKs](#)) específicos para *Expo*. *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. Esto incrementaría el tiempo de desarrollo y el riesgo de problemas de compatibilidad.

3.3.3. Subcapa de Observabilidad

Una parte crítica en el desarrollo de productos digitales es disponer de un entendimiento claro de cómo los usuarios interactúan con el producto y de cómo se comportan los sistemas que lo sustentan. En PANOT, la observabilidad se aborda en dos dimensiones: el análisis de uso y comportamiento del usuario en la aplicación, y el seguimiento del sistema de agentes que orquesta el procesamiento de inteligencia relacional (Sección 4.3.2). Para cada una se ha integrado una herramienta específica: *PostHog* para la analítica de producto y *LangSmith* para la observabilidad del sistema agéntico.

Para el análisis de uso de funcionalidades y comprensión del comportamiento de los usuarios es esencial que los desarrolladores tomen decisiones informadas basadas en datos reales y conocer qué aspectos de la aplicación aportan valor al usuario y cuáles no. Para ello se ha integrado *PostHog*, una plataforma de análisis de productos de código abierto que proporciona la capacidad de hacer un seguimiento de eventos una

vez la aplicación está en manos del usuario.

La elección de *PostHog* se fundamenta en su compatibilidad con React Native y Expo, su modelo de código abierto, y su enfoque específico en análisis de producto que proporciona métricas relevantes para 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. Este seguimiento de eventos se implementa mediante llamadas a funciones específicas que permiten registrar eventos personalizados con propiedades asociadas.

Junto con la analítica de producto, se ha incorporado *LangSmith* como capa de observabilidad del sistema de agentes desarrollado con *LangChain*. *LangSmith* es la plataforma de trazabilidad y depuración del ecosistema *LangChain* y permite monitorizar en tiempo real cada ejecución del sistema de agentes: latencia de las llamadas, tokens consumidos, coste por ejecución y flujo completo de invocaciones entre cada agente. Esta visibilidad es fundamental para validar requisitos de rendimiento y eficiencia económica, depurar fallos y optimizar el uso del modelo de lenguaje.

Para la analítica de producto 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 presentan modelos más restrictivos en sus planes gratuitos comparado con *PostHog*, y requieren una configuración más compleja para su integración con Expo.

En cuanto a la observabilidad del sistema de agentes, no se ha considerado ninguna alternativa ya que *LangSmith* es la plataforma de trazabilidad por defecto del ecosistema *LangChain* y resultaba la opción más coherente.

4.

Desarrollo del Proyecto

4.1. 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* [8]. 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.

4.1.1. 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* eliminando 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.

4.1.2. 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 largas 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 4.1—. 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

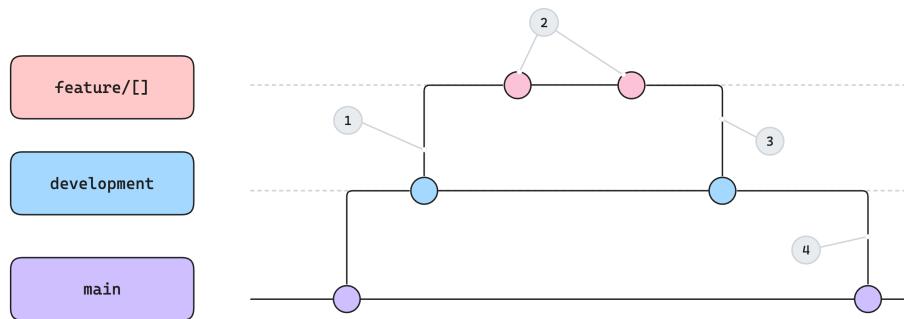


Figura 4.1. Diagrama orientativo de la estrategia de ramificación. (1) Aislamiento: Creación de una rama efímera (*feature*) a partir de desarrollo para trabajar en un *Issue* específico. (2) Implementación: Desarrollo de la funcionalidad mediante *commits* atómicos. (3) Integración: Fusión de la rama 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

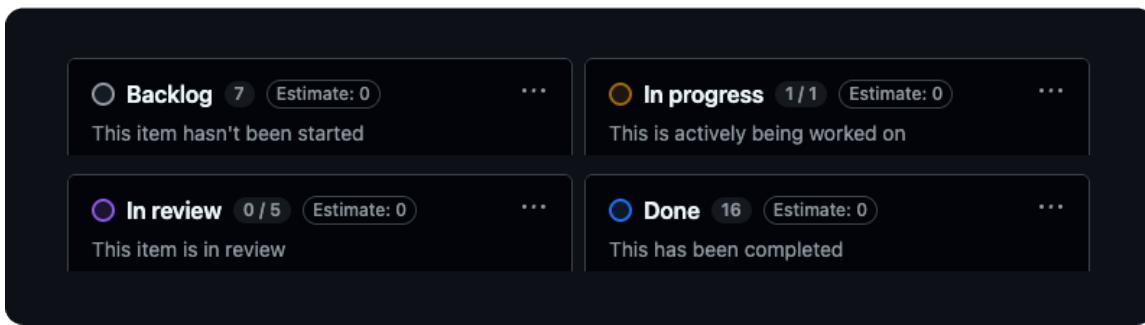


Figura 4.2. Visualización de las columnas del tablero Kanban implementado en GitHub Projects en un punto del desarrollo. Se destaca el límite de WIP configurado a 1 en la columna *In Progress* (indicador 1/1), forzando un flujo continuo y eliminando el desperdicio asociado al cambio de contexto.

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.

4.2. 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.

4.2.1. 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 Captura sin Fricción:** Se asume que los usuarios solo registrarán información de sus contactos si el esfuerzo requerido es cercano a cero. La captura mediante una interfaz *voice-first* proporciona este valor al permitir registrar matices humanos al instante y en movimiento, eliminando la barrera del teclado y permitiendo que el usuario mantenga su atención en la interacción social.
2. **Valor de la Continuidad Relacional:** Se asume que el usuario obtiene una utilidad crítica al disponer de una cronología estructurada (*timeline*) de sus encuentros. Este registro histórico permite retomar conversaciones de forma natural, eliminando la incertidumbre del *dónde lo dejamos* y transformando una lista estática de nombres en una historia relacional viva.
3. **Valor del Entendimiento Contextual:** Se asume que el valor real de un contacto no reside en sus datos demográficos, sino en su contexto único y su relación con el usuario. Es por eso que un contacto pasa de ser una entidad tabular a ser una persona con un contexto específico que va evolucionando a lo largo del tiempo.

4.2.2. Historias de Usuario

Para traducir estas hipótesis en funcionalidades concretas y mantener un enfoque centrado en el usuario, se han definido una serie de Historias de Usuario (**HU**). Estas historias actúan como unidades de trabajo que describen el *quién*, el *qué* y el *para qué* de cada capacidad del sistema.

<i>ID</i>	<i>Historia de Usuario</i>
HU-01	<i>Como usuario, quiero registrarme y acceder de forma segura para mantener mis relaciones personales organizadas y accesibles de manera segura y privada.</i>
HU-02	<i>Como usuario, quiero importar contactos de mi agenda nativa para poder usar mis contactos ya existentes.</i>
HU-03	<i>Como usuario, quiero crear un contacto dictando su descripción o escribiéndolo manualmente para permitir que ellos mismos lo hagan o hacerlo con el mínimo esfuerzo posible.</i>
HU-04	<i>Como usuario, quiero grabar una interacción por voz para poder reducir el esfuerzo de registrarla y para hacerlo a mayor velocidad que escribiendo.</i>
HU-05	<i>Como usuario, quiero validar o corregir la información procesada por el sistema para asegurar que el registro de mis relaciones sea siempre acorde a la realidad.</i>
HU-06	<i>Como usuario, quiero visualizar un historial cronológico por cada contacto para recordar instantáneamente el contexto de nuestras últimas conversaciones.</i>
HU-07	<i>Como usuario, quiero buscar contactos por palabras clave del resumen para localizarlos por su contexto y no solo por su nombre.</i>
HU-08	<i>Como usuario, quiero que el sistema extraiga datos de interés de las interacciones que registre y actualice los perfiles de mis contactos automáticamente para reducir el esfuerzo manual de edición de fichas.</i>
HU-09	<i>Como usuario, quiero enviar reportes de error o sugerencias para poder contribuir a la mejora del producto.</i>

Tabla 4.1. Definición de las Historias de Usuario para el **PMV** de PANOT

4.2.3. Requisitos Funcionales

Se han establecido una serie de requisitos obligatorios que instrumentarán al sistema con las capacidades para validar las premisas e historias de usuario planteadas. Estos requisitos de funcionalidad se han agrupado en las siguientes características principales (Features):

<i>ID</i>	<i>Nombre</i>	<i>Requisitos Funcionales Asociados (IDs)</i>
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 4.2. Features del Sistema y los IDs de los Requisitos Funcionales Asociados

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 4.3. 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 4.4. 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 transcritto 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 4.5. 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 4.6. Requisitos Funcionales de la *Feature 4 - Inteligencia Relacional*

<i>ID</i>	<i>Descripción</i>
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 la 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 4.7. Requisitos Funcionales de la *Feature 5 - Soporte y Feedback*

4.2.4. Matriz de Trazabilidad

Para garantizar la integridad del sistema y asegurar que cada desarrollo técnico responde directamente a una necesidad del usuario, se ha elaborado la siguiente matriz de trazabilidad. Esta herramienta vincula los *Requisitos Funcionales* con las *Historias de Usuario*. Esta trazabilidad permite verificar la cobertura total del **PMV** y asegura que el diseño del sistema multi-agente y la arquitectura *local-first* están plenamente justificados por el valor que aportan al usuario final.

<i>ID Requisito Funcional</i>	<i>ID Historia de Usuario</i>
FR-01	HU-01
FR-02	HU-01
FR-03	— (necesidad técnica)
FR-04	HU-03
FR-05	HU-02
FR-06	HU-03
FR-07	HU-07
FR-08	HU-03, HU-05, HU-08
FR-09	— (necesidad técnica)
FR-10	HU-04
FR-11	HU-05
FR-12	HU-04
FR-13	— (necesidad técnica)
FR-14	HU-08
FR-15	HU-08
FR-16	HU-09
FR-17	HU-09

Tabla 4.8. Matriz de Trazabilidad de Requisitos e Historias de Usuario

4.2.5. 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:

<i>ID</i>	<i>Tipo</i>	<i>Descripción</i>	<i>Criterio Aceptación</i>
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.
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 35 segundos de media para interacciones de longitud media (hasta 400 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 0,01 unidades monetarias.
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 4.9. Requisitos No Funcionales del sistema: ID, Tipo, Descripción y Criterio de Aceptación

4.3. Diseño del Sistema

4.3.1. Visión general de la Arquitectura

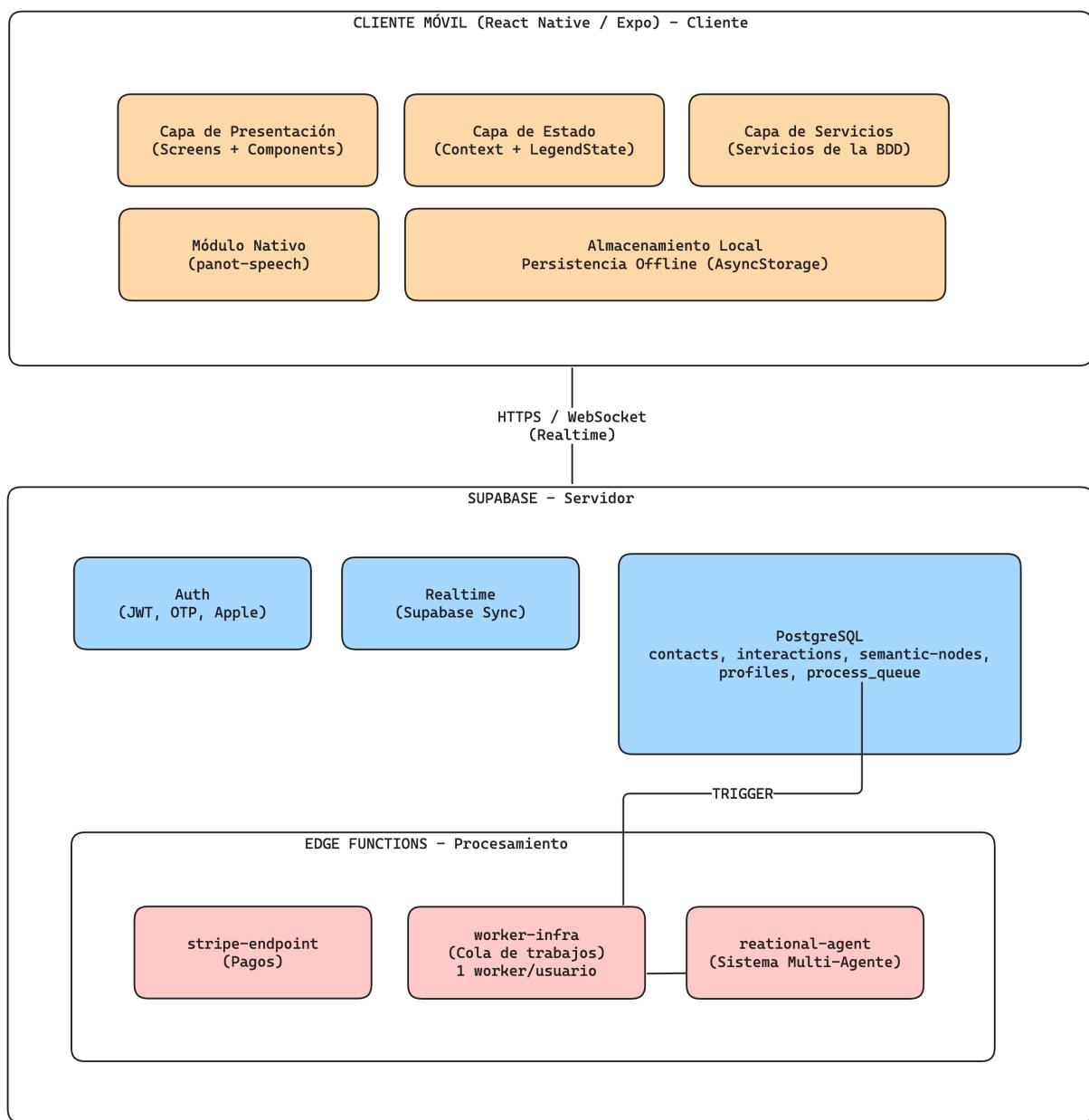


Figura 4.3. Arquitectura general del sistema PANOT

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 4.3.2.

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.

4.3.2. 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.

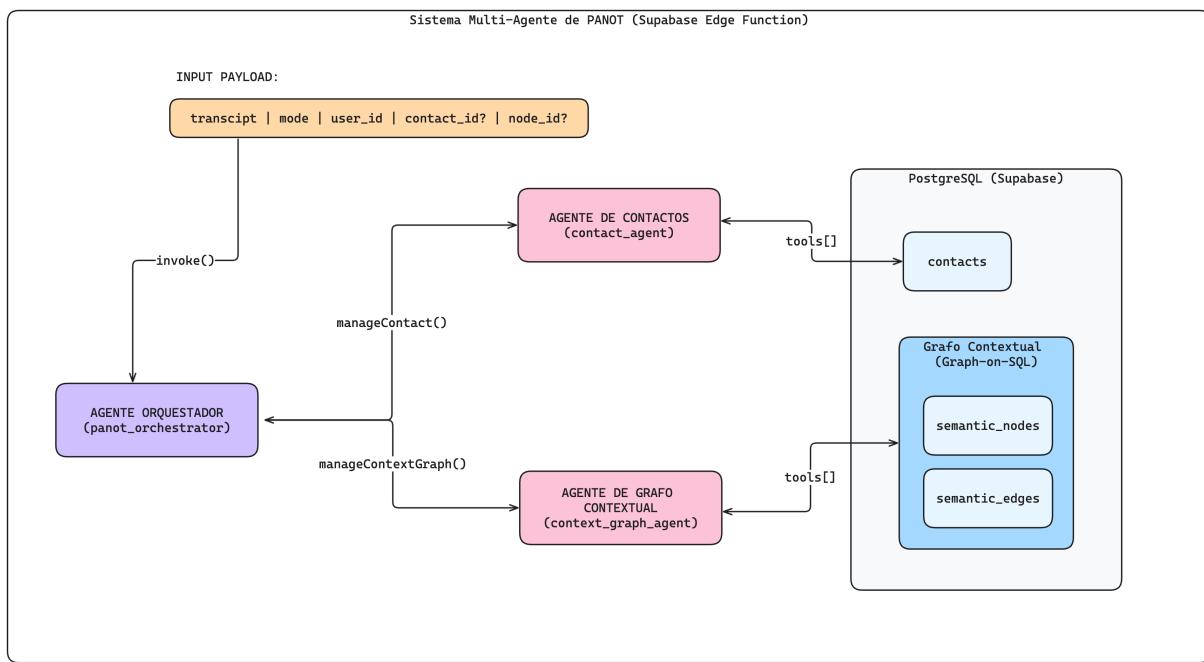


Figura 4.4. Diagrama de la arquitectura del sistema multi-agente de PANOT

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-Agente. 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.

En cuanto a la elección de herramientas, se han implementado funciones basadas en procesamiento por lotes (*batch processing*). Este enfoque permite que los agentes procesen múltiples elementos de manera eficiente en una sola operación. Como resultado, se reduce la latencia y se optimizan los costes operativos del sistema.

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 *similaridad coseno s* entre el embedding entrante y los ya almacenados. El comportamiento del matching se define formalmente mediante la función:

`find_semantic_match` : $[0, 1] \rightarrow \mathcal{A}$ donde

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

Donde $n^* = \arg \max_{n \in \mathcal{N}} \text{sim}(e_{\text{nuevo}}, e_n)$ 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.

4.3.3. Modelado de Datos

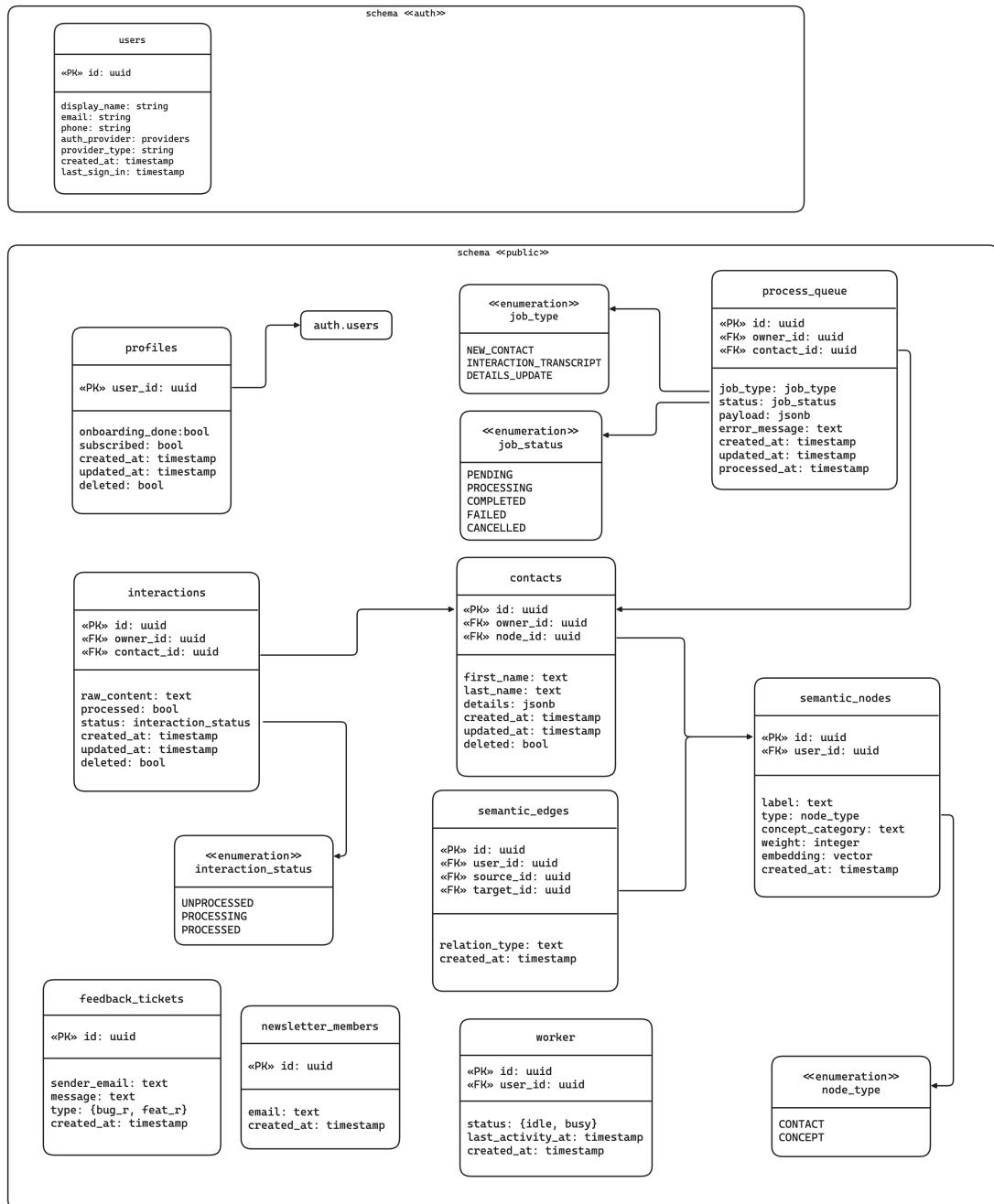


Figura 4.5. Diagrama del modelado de datos de PANOT con los schemas de Supabase «public» y «auth». En el diagrama del schema «auth» solo se ha incluido la tabla `auth.users` ya que es la única tabla de el schema que se ha modificado. Y en el diagrama del schema «public» se ha obviado del diagrama que las claves foráneas `user_id` o `owner_id` de las tablas provienen del `id` de la tabla `users` del schema «auth». A modo de ejemplo, solo se ha añadido en la tabla `profiles` para mantener el diagrama más legible.

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 automáticamente por el sistema.
- **interactions**: Transcripciones de voz grabadas por el usuario. El campo `raw_content` contiene el texto transcrita. 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.

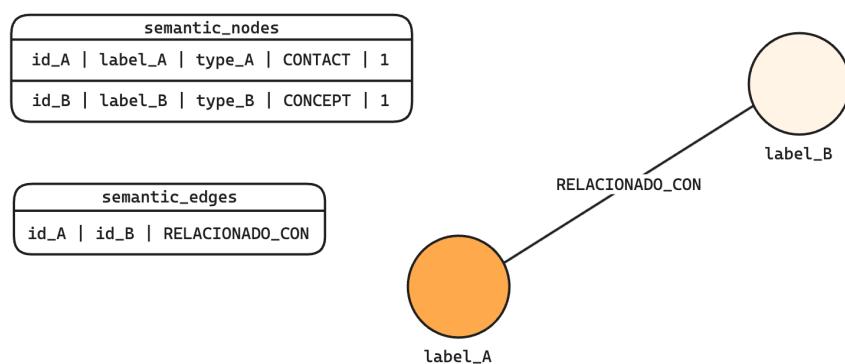


Figura 4.6. Ejemplo de grafo semántico en PostgreSQL. En el se pueden ver dos nodos: un primer nodo A de tipo `CONTACT` y un segundo nodo B de tipo `CONCEPT`. Ambos están conectados con una relación de tipo `RELACIONADO_CON`.

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 sistema de procesamiento 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 4.3.2 anterior.

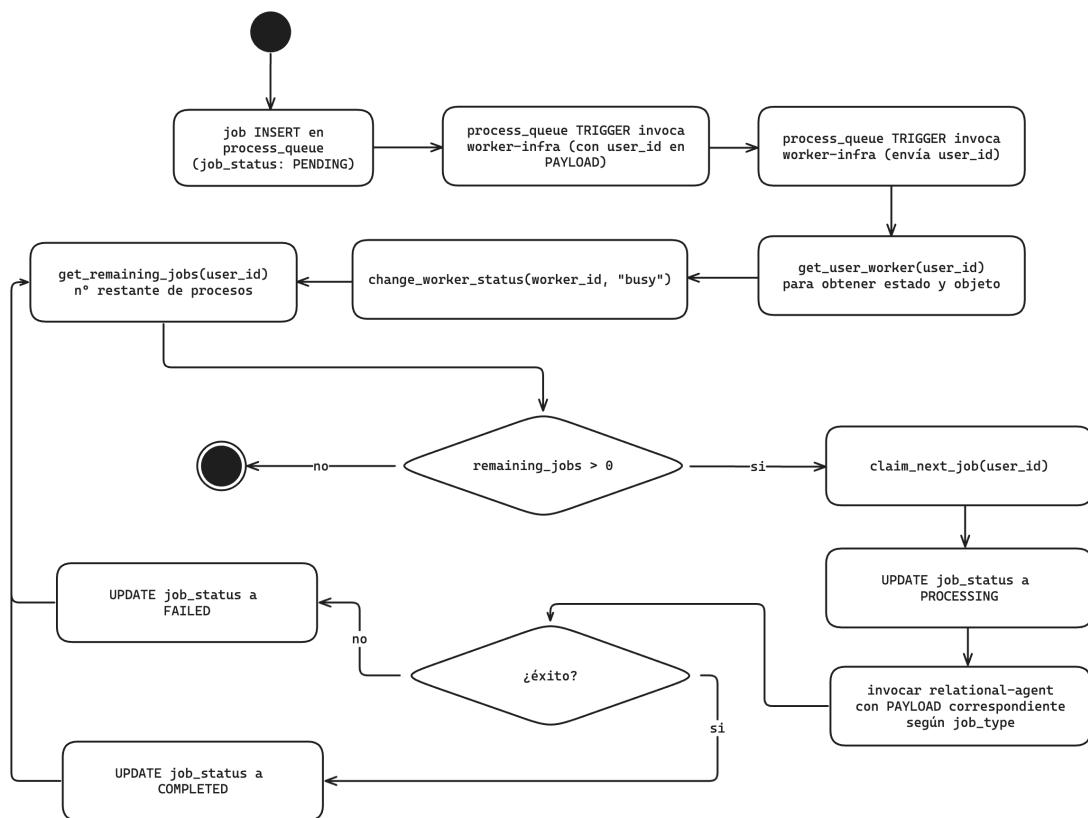


Figura 4.7. Diagrama de actividad del sistema de procesamiento de PANOT

El diagrama de actividad anterior (4.7) 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`):

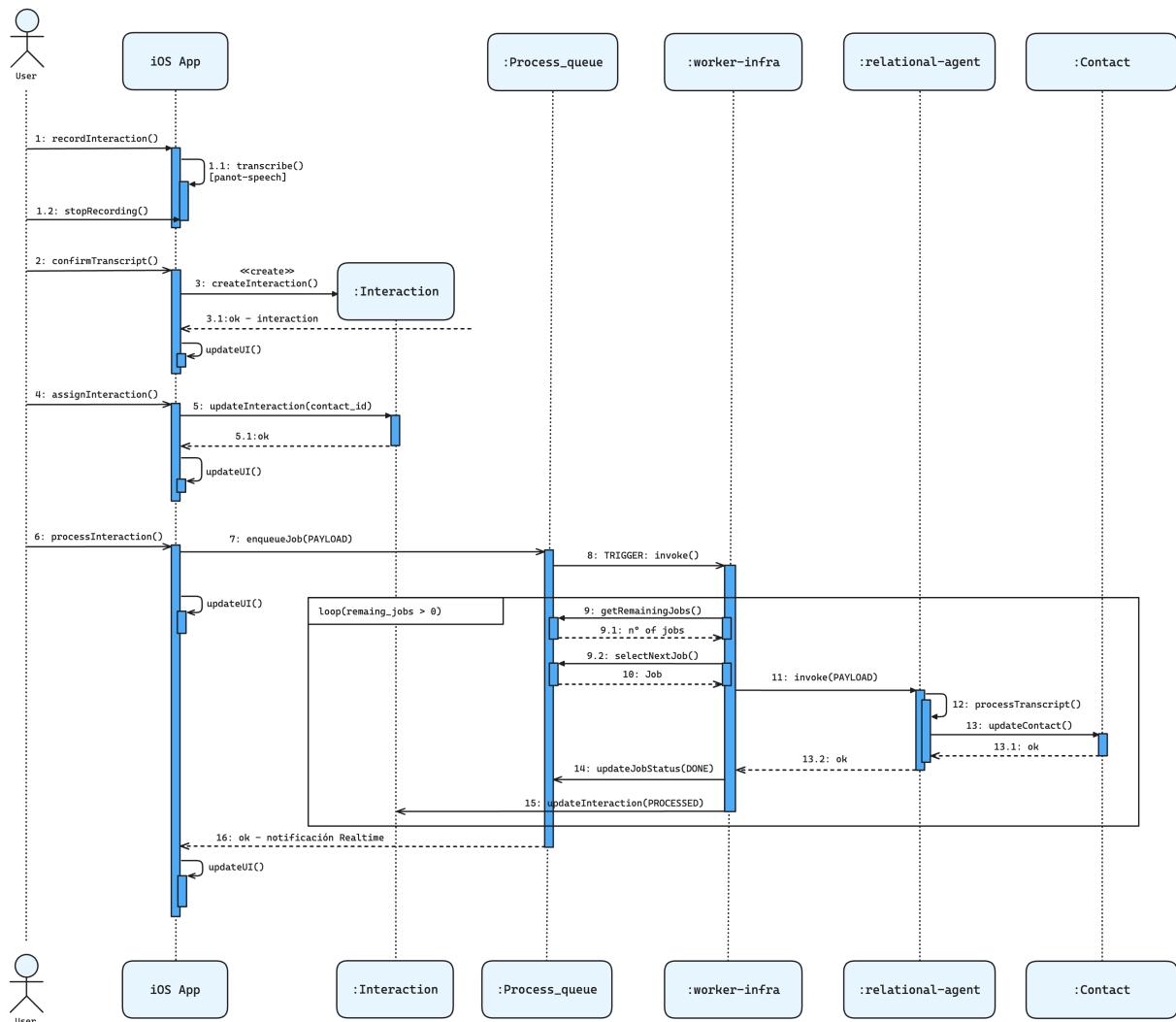


Figura 4.8. Diagrama de secuencia del procesamiento de una Interacción grabada por el usuario

4.4. Implementación del Sistema

Como se mencionó en la sección 4.1.1, el desarrollo de PANOT se centra en la producción de un artefacto que permita validar las hipótesis de valor. Es por esto que, a diferencia de un modelo incremental tradicional, la implementación de este sistema se ha llevado a cabo como un esfuerzo unificado para materializar el PMV. Bajo esta premisa, la implementación de PANOT se ha organizado en ejes de desarrollo concurrentes para maximizar la velocidad de desarrollo y permitir el trabajo paralelo sobre las diferentes capacidades del sistema.

Es esencial señalar que en este proyecto, el despliegue no es una fase posterior de la implementación, sino su culminación. La publicación de la aplicación "Semilla" en una plataforma de distribución es el evento de cierre del proceso de construcción (Fase Construir) y da comienzo a la fase de medición (Fase Medir). Por ello, la implementación incluirá tareas como la instrumentación de herramientas de medición ya que, sin ellas, el PMV no podría cumplir su propósito de validación de la hipótesis de valor.

4.4.1. Organización panot-hq

Para facilitar la gestión y organización del código del proyecto, se ha creado la organización panot-hq en *GitHub*. Esta organización centraliza todos los repositorios de desarrollo de PANOT, permitiendo una estructura clara y escalable del proyecto. Además, como ya se explicó en la sección 4.1.2, se ha configurado un proyecto dentro de la organización para agrupar las unidades de trabajo y el progreso de las mismas. Esta estructura se ha establecido para, además de los beneficios obvios de trazabilidad y seguimiento, permitir la incorporación de nuevos colaboradores en el futuro. Los repositorios levantados de la organización son los siguientes:

 mobile source code of PANOT mobile app	• TypeScript		0		0
 landing PANOT's landing page	• TypeScript		0		0
 edge-backend Supabase Edge Functions source code	• TypeScript		0		0
 graph-visualizer helper semantic graph visualization tool	• TypeScript		0		0
 panot-speech PANOT's iOS native module for speech recognition	• Swift		0		1

Figura 4.9. Repositorios de la organización panot-hq

- panot-hq/mobile: Código de la aplicación móvil.
- panot-hq/landing: Código de [incluir en el glosario def de landing] landing de PANOT.
- panot-hq/edge-backend: Código de las Edge Functions que se despliegan en Supabase.
- panot-hq/graph-visualizer: Código del entorno de pruebas para la visualización de grafos de conocimiento.
- panot-hq/panot-speech: Código del paquete *npm* del módulo nativo iOS para transcripción en tiempo real para dispositivos iOS.

4.4.2. Ejes de Desarrollo

Se define eje de desarrollo como la línea de trabajo dedicada al desarrollo de una capacidad del sistema, incluyendo su integración con el resto de los componentes con los que interactúe. En el caso de PANOT, se han llevado a cabo los siguientes ejes de desarrollo:

Eje	Repositorios Involucrados	Tecnologías Utilizadas
(1) Aplicación Móvil	panot-hq/mobile panot-hq/panot-speech	Expo, SwiftUI, TypeScript, PostHog, Stripe
(2) Base de Datos	panot-hq/mobile	Supabase, PostgreSQL, TypeScript
(3) Sistema Multi-Agente	panot-hq/edge-backend panot-hq/graph-visualizer panot-hq/mobile	Supabase, Deno, TypeScript, LangChain, LangSmith
(4) Sistema de Cola de Trabajos	panot-hq/edge-backend panot-hq/mobile	Supabase, Deno, TypeScript
(5) Landing de PANOT	panot-hq/landing	Next.js, TypeScript

Tabla 4.10. Ejes de desarrollo de PANOT

- (1) Aplicación Móvil: Desarrollo de la UI y componentes de la aplicación de PANOT.
- (2) Base de Datos: Desarrollo y despliegue de la infraestructura de datos.
- (3) Sistema Multi-Agente: Desarrollo de sistema de procesamiento de información.
- (4) Sistema de Cola de Trabajos: Desarrollo de sistema de gestión de trabajadores.
- (5) Landing de PANOT: Desarrollo de la página web.

4.4.3. Evolución Cronológica

Para asegurar una evolución estable y controlada del código, se estableció una planificación orientativa centrada en la reducción de incertidumbre técnica y basada 4 hitos principales:

1. *Sincronización y Offline-first*: Garantizar que la app fuera funcional sin conexión.
2. *Captura Nativa*: Validar la transcripción en tiempo real de forma nativa y offline.
3. *Inteligencia Relacional*: Implementar el sistema multi-agente para el procesamiento de las interacciones y creación de contactos.
4. *Cierre de Producto*: Implementar sistema de suscripciones y observabilidad para el ciclo "Medir".

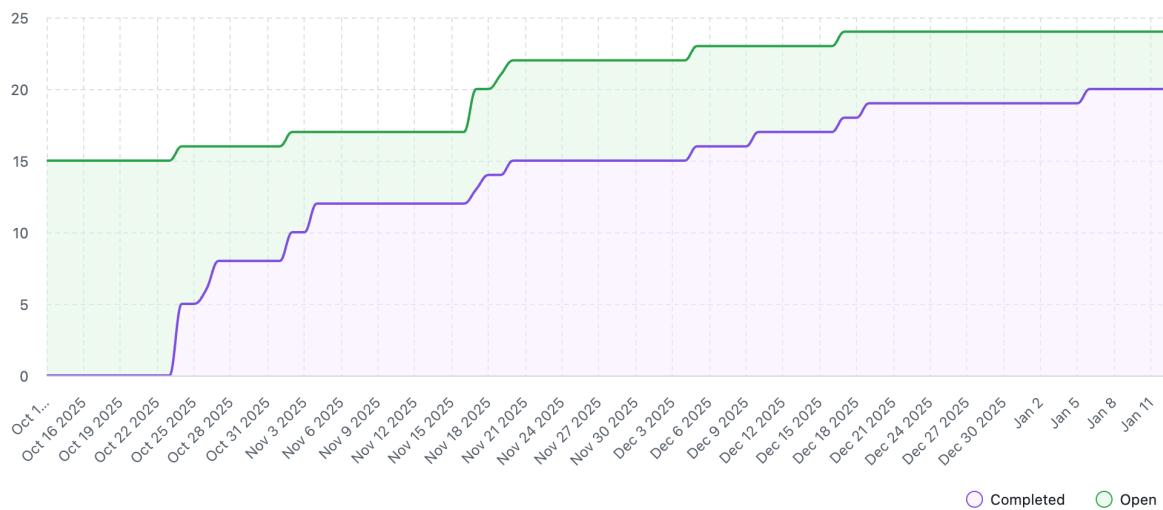


Figura 4.10. Gráfico de burndup de issues cerrados en el proyecto de PANOT

El desarrollo del [PMV](#) se ha llevado a cabo en un periodo de cuatro meses (octubre 2025 - enero 2026). Se han descartado cambios que se han implementado después de estas fechas ya que engloban tanto cambios no críticos como correcciones de errores menores o iteraciones en la interfaz de usuario. A continuación, se detalla la línea de tiempo del desarrollo de octubre de 2025 a enero de 2026, siguiendo el orden de prioridad de los hitos enumerados anteriormente.

Octubre 2025

Ejes de desarrollo: (1) Aplicación Móvil, (2) Base de Datos.

Repositorios: panot-hq/mobile, panot-hq/panot-speech.

Requisitos Funcionales implementados: primera parte del [FR-01](#), [FR-02](#), [FR-04](#), [FR-08](#), [FR-10](#), [FR-11](#), [FR-12](#), [FR-13](#)

El objetivo de octubre fue alcanzar una versión estable *offline* con las capacidades mínimas de gestión de usuario, contactos e interacciones. Además, se desarrolló e integró el paquete nativo panot-speech con la interfaz de usuario, permitiendo al usuario grabar y transcribir su voz en tiempo real y de forma totalmente offline localmente.

<input checked="" type="checkbox"/> Implementar el flujo de creación de cuenta (Email y Apple ID) #22	panot-hq/mobile · angx1 closed on Oct 1, 2025	(R)
<input checked="" type="checkbox"/> Crear un nuevo contacto desde cero #6	panot-hq/mobile · angx1 closed on Oct 24, 2025	(R)
<input checked="" type="checkbox"/> Visualizar la ficha completa de un contacto #7	panot-hq/mobile · angx1 closed on Oct 24, 2025	(R)
<input checked="" type="checkbox"/> Implementar la grabación de voz para interacciones #12	panot-hq/mobile · angx1 closed on Oct 24, 2025	(R)
<input checked="" type="checkbox"/> Mostrar el historial de interacciones en la ficha de contacto #14	panot-hq/mobile · angx1 closed on Oct 24, 2025	(R)
<input checked="" type="checkbox"/> Integrar el servicio de transcripción de voz a texto #26	panot-hq/mobile · angx1 closed on Oct 24, 2025	(R)
<input checked="" type="checkbox"/> Implementar Capa de Persistencia Local para funcionalidad offline #33	panot-hq/mobile · angx1 closed on Oct 26, 2025	(enhancement)
<input checked="" type="checkbox"/> Editar la información de un contacto #8	panot-hq/mobile · angx1 closed on Oct 27, 2025	(R)
<input checked="" type="checkbox"/> Implementar la solicitud de permisos contextuales #25	panot-hq/mobile · angx1 closed on Oct 27, 2025	(R)

Figura 4.11. Issues cerradas en octubre de 2025

Como se puede ver en la figura 4.11, se implementó el registro por Email y Apple ID (Issue #22) y se configuró la capa de persistencia local con *Legend-State* (Issue #33). Adicionalmente, se habilitó parte del **CRUD** básico manual de contactos (Issues #6, #7, #8). Se implementó también la interfaz de grabación y la transcripción en tiempo real (Issues #12, #26), gestionando los permisos contextuales necesarios y obligatorios de iOS para el uso del micrófono y acceso a la agenda de contactos local del usuario (Issue #25).

```

* 40194a1 - Merge pull request #36 from panot-hq/feature/core-interaction-loop (28/10/2025) <angel>
| \
| * cce29fa - (origin/feature/core-interaction-loop, feature/core-interaction-loop) error fix: supabase data not syncing (28/10/2025) <angel>
* 23b4e08 - Merge pull request #35 from panot-hq/feature/core-interaction-loop (27/10/2025) <angel>
| \
| * bf4e1cf - feat(interaction): implement real-time voice capture and transcription (27/10/2025) <angel>
| * 2b9569c - (origin/feature/import-local-contacts, feature/import-local-contacts) feat(contacts): allow importing from native phonebook (26/10/2025) <angel>
* e75cf85 - Merge pull request #34 from panot-hq/feature/local-storage (24/10/2025) <angel>
| \
| * 8cb649d - (origin/feature/local-storage, feature/local-storage) feature(persistence): implement local storage with Legend-State (24/10/2025) <angel>
* 2003b6c - Merge pull request #32 from panot-hq/feature/core-interaction-loop (09/10/2025) <angel>
| \
| * 825a91a - feat(core): interaction list refresh after interaction deletion (09/10/2025) <angel>
* 5509c63 - Merge pull request #31 from panot-hq/feature/core-interaction-loop (09/10/2025) <angel>
| \
| * 52807db - feat(core): one line adjustment on gesture availability on contacts/[id] (09/10/2025) <angel>
* 0346a8b - Merge pull request #30 from panot-hq/feature/core-interaction-loop (09/10/2025) <angel>
| \
| * ba32f83 - feat(core): implement the core user workflow (09/10/2025) <angel>
* 6b2f5ed - Merge pull request #29 from panot-hq/feature/account-signup (28/09/2025) <angel>
| \
| * 1a58560 - (origin/feature/account-signup, feature/account-signup) feat(auth): implement otp email verification process for sign up (28/09/2025) <angel>
* 2d32c95 - Merge pull request #28 from panot-hq/feature/account-signup (28/09/2025) <angel>
| \
| * 818d843 - feat(auth): implement email and Apple sign up flows (28/09/2025) <angel>
* 946b537 - Remove mobile header from README (26/09/2025) <angel>
| \
| * 3b57f48 - first commit (26/09/2025) <angel>
* 8926ee5 - Initial commit (25/09/2025) <angel>
(END)

```

Figura 4.12. Árbol de commits de octubre (y finales de septiembre) de 2025 en repositorio Git local de panot-hq/mobile

Noviembre 2025

Ejes de desarrollo: (1) Aplicación Móvil, (3) Sistema Multi-Agente (Inicio).

Repositorios: panot-hq/mobile.

Requisitos Funcionales implementados: segunda parte del [FR-01](#), [FR-05](#), [FR-06](#), [FR-07](#), [FR-09](#)

Noviembre se centró en completar las funcionalidades restantes de gestión de contactos además de preparar el sistema para las capacidades de inteligencia relacional. El objetivo fue alcanzar un sistema funcional que permitiera la actualización automática de los detalles de los contactos mediante el procesamiento de interacciones. Como dato importante, en este punto del proyecto, todavía no se contaba con el sistema multi-agente ni la clase de grafo semántico, es decir, todavía no existía un sistema completo de procesamiento relacional ni en el procesamiento de interacciones ni en la creación de contactos "hablando sobre ellos". Lo implementado fue una solución temporal.

Como se ve en la Figura 4.13, en noviembre se implementó la eliminación de contactos (Issue #9), su creación mediante lenguaje natural (Issue #37), la importación desde la agenda nativa (Issue #10) y la búsqueda de contactos (Issue #11), reduciendo drásticamente la fricción de uso de la aplicación. Por último, se completó el flujo de acceso de usuarios (Issue #23) lo que desbloqueó por completo que el sistema contara con un mecanismo de autenticación robusto y funcional tanto para el *sign-up* (registro) como para el *log-in* (inicio de sesión), y se incorporó el apartado de ajustes de la



Figura 4.13. Issues cerradas en noviembre de 2025

aplicación (Issue #42), lo que estableció la base para futuras capacidades y el apartado de feedback.

```
* 4735cd2 - minor sync changes (30/11/2025) <angel>
* 79785be - Merge pull request #51 from panot-hq/feature/paywall (27/11/2025) <angel>
| \
| * 711b747 - (origin/feature/paywall, feature/paywall) feature(account): paywall logic and stripe integration (27/11/2025) <angel>
| / \
| * 4e0792d - Merge pull request #49 from panot-hq/feature/relational-intelligence (20/11/2025) <angel>
| \
| * ec4dff4 - (origin/feature/relational-intelligence, feature/relational-intelligence) feat(contacts): contact's context update with interaction #48 (20/11/2025) <angel>
| / \
| * ec1a4bc - Merge pull request #46 from panot-hq/feature/account-login (18/11/2025) <angel>
| \
| * 57ea651 - (origin/feature/account-login, feature/account-login) feat(auth): log in with email and apple implementation closing #23 (18/11/2025) <angel>
| / \
| * 24d0be3 - Merge pull request #45 from panot-hq/feature/settings (17/11/2025) <angel>
| \
| * 3a8d309 - (origin/feature/settings, feature/settings) feat(settings): issue #42 closing (for now) (17/11/2025) <angel>
| * ccf05a6 - Merge pull request #44 from panot-hq/feature/settings (17/11/2025) <angel>
| \
| * 390803c - feat(settings): voice language configuration (17/11/2025) <angel>
| /
| * dd7ae5e - UI fixes: small ui changes (17/11/2025) <angel>
| * 3702fc4 - feat(contact): api calling migration to supabase edge functions (05/11/2025) <angel>
| * fc00ae3 - UI fixes: some ui improvements on interaction card (05/11/2025) <angel>
| * d19fac5 - UI fixes: contacts search bar animation fix (04/11/2025) <angel>
| * 5f2194d - Merge pull request #40 from panot-hq/feature/contact-voice-capture (04/11/2025) <angel>
| \
| * 37975ee - (origin/feature/contact-voice-capture, feature/contact-voice-capture) UI fixes: implemented audio visualization (04/11/2025) <angel>
| /
| * 7c1496c - feat(contact): fix gut conflict and feature implementation (04/11/2025) <angel>
| * cde48c6 - Stopped tracking .env File (02/11/2025) <angel>
| * 9977e15 - feat(contacts): implement contact voice capture (02/11/2025) <angel>
```

Figura 4.14. Árbol de commits de noviembre de 2025 en repositorio panot-hq/mobile local

Diciembre 2025

Ejes de desarrollo: (1) Aplicación Móvil, (2) Base de Datos, (3) Sistema Multi-Agente, (4) Sistema de Cola de Trabajos.

Repositorios: [panot-hq/mobile](#), [panot-hq/edge-backend](#), [panot-hq/graph-visualizer](#).

Requisitos Funcionales implementados: [FR-14](#), [FR-15](#), [FR-16](#), [FR-17](#)

Diciembre representó el mes con mayor densidad técnica, y se centró en completar las funcionalidades esenciales para el lanzamiento del [PMV](#), incluyendo el sistema de suscripciones, el sistema multi-agente, el sistema de cola de procesamiento, la capa de observabilidad y los mecanismos de soporte.



Figura 4.15. Issues cerradas en diciembre de 2025

En primer lugar, se completó la lógica del *paywall* y la integración con *Stripe* para suscripciones mensuales y acceso a funcionalidades premium (Issue #50) con el desarrollo de la Edge Function `stripe-infra`. Esta implementación permitió establecer el modelo de monetización del sistema y controlar el acceso a las capacidades de procesamiento relacional de la aplicación.

En segundo lugar, se implementó el [sistema multi-agente](#) completo para el procesamiento de información (Issue #52):

- Se desarrolló la edge function `relational-agent` en [panot-hq/edge-backend](#) utilizando *LangChain* para la orquestación de agentes e integración con la aplicación móvil.
- Se crearon las tablas necesarias en Supabase para los nodos y las aristas semánticas.
- Se desarrolló la herramienta [panot-hq/graph-visualizer](#) para facilitar el desarrollo, pruebas y validación de la estructura del grafo semántico del usuario.

4. Y se realizó la integración con *LangSmith* como capa de observabilidad del sistema agéntico, permitiendo así monitorizar el flujo de procesamiento con información como la latencia, los tokens utilizados o el coste de la ejecución de cada llamada al sistema [4.16](#).

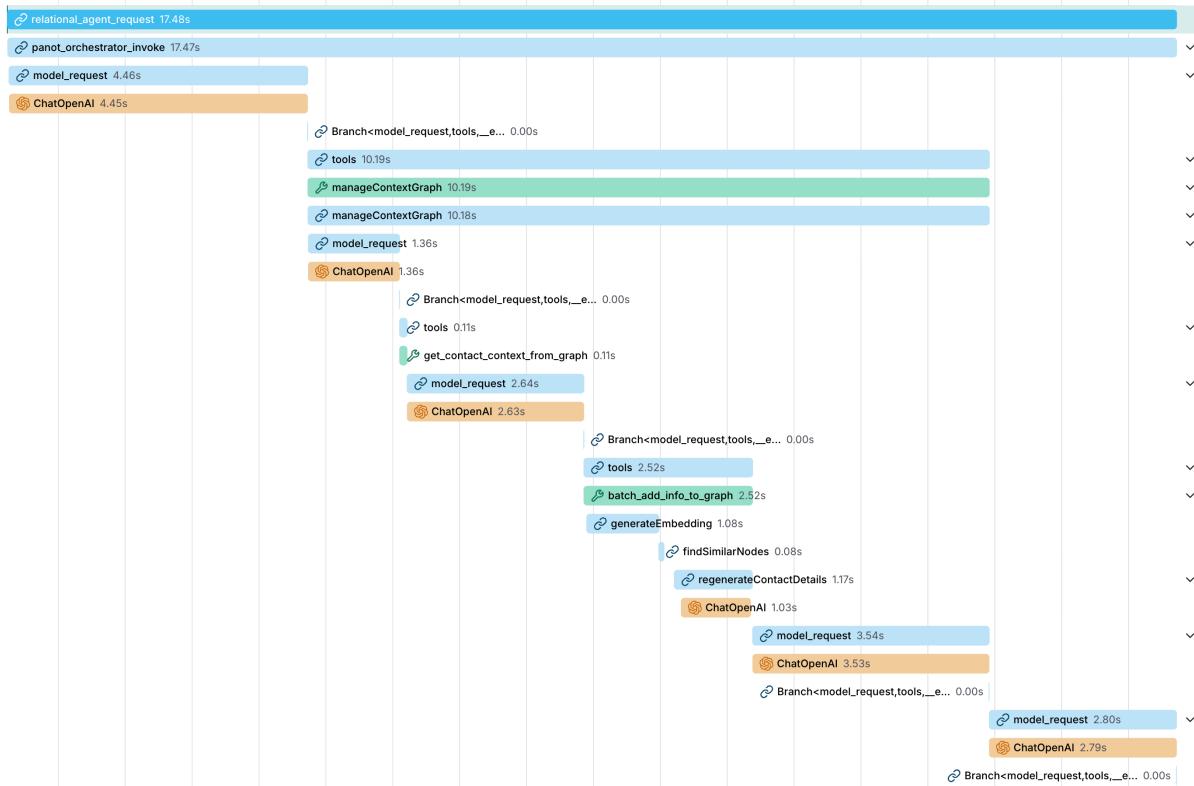


Figura 4.16. Ejemplo de traza de ejecución en *LangSmith* del sistema multi-agente de PANOT

Para la gestión de la ejecución asíncrona de procesos entre usuarios, se implementó el sistema de cola de procesos, como puede verse en la Figura [4.7](#). Entre los principales sub-hitos de esta implementación se encuentran el desarrollo de `worker-infra` en `panot-hq/edge-backend`, su integración con la aplicación móvil, así como la creación de las tablas necesarias en Supabase tanto para la cola de procesos como para los `workers` de los usuarios.

Por último, se instrumentó la aplicación para el análisis de comportamiento de usuarios utilizando *PostHog* (Epic #55). Esta implementación permitió la captura de métricas para validar las hipótesis de valor definidas en el proyecto. Y se añadió el mecanismo de soporte y recolección de *tickets* para reportes de errores o solicitudes de funcionalidades desde la aplicación (Issue #43) lo que instauró el canal de comunicación entre los usuarios y el equipo de desarrollo (por ahora unipersonal).

```
* a439463 - (origin/feature/account-settings, feature/account-settings) feat(settings): implemented membership management and account settings (06/01/2026) <angel>
* e6ece16 - UI fix: minor text changes (29/12/2025) <angel>
* 19ef578 - Merge pull request #59 from panot-hq/feature/feedback-collection (19/12/2025) <angel>
  \ 
  * 99665995 - (origin/feature/feedback-collection, feature/feedback-collection) feat(feedback): feedback ticket collection logic and UI implementation #43 (19/12/2025) <angel>
  / 
  * 45e4c95 - fixed user creation errors (19/12/2025) <angel>
  * 8claae9 - Merge pull request #57 from panot-hq/feature/observability (17/12/2025) <angel>
    \ 
    * 8879e59 - (origin/feature/observability, feature/observability) feat(observability): API key privatization (17/12/2025) <angel>
    * 563b781 - feat(observability): implementation of observability layer with PostHog #55 (17/12/2025) <angel>
    / 
    * a8fe15b - UI fixes: changes some translation texts and minor UI changes (16/12/2025) <angel>
    * 7f581e1 - Merge pull request #54 from panot-hq/feature/app-localisation (16/12/2025) <angel>
      \ 
      * c1f9d2e - (origin/feature/app-localisation, feature/app-localisation) feat(setup): app localisation to english and spanish (16/12/2025) <angel>
      / 
      * e354939 - UI fixes: cambios en UI y UX (16/12/2025) <angel>
      * ca02502 - Merge pull request #53 from panot-hq/feature/info-processing (10/12/2025) <angel>
        \ 
        * 96e8520 - (origin/feature/info-processing, feature/info-processing) info processing integration with edge-back (10/12/2025) <angel>
        / 
        * 1ef491d - db sync upgrade for semantic graph compatibility (05/12/2025) <angel>
```

Figura 4.17. Árbol de commits de diciembre de 2025 (y principios de enero de 2026) en repositorio Git local de panot-hq/mobile

Enero 2026

Ejes de desarrollo: (1) Aplicación Móvil (Cierre), (5) Landing de PANOT.

Repositorios: panot-hq/mobile, panot-hq/landing.

Requisitos Funcionales implementados: [FR-03](#)

Habiendo finalizado el desarrollo completo del [PMV](#), principios de enero (del 6 al 8) de 2026 se dedicaron al desarrollo de la página web landing de PANOT tanto en inglés como en español además de incorporar la sección de gestión de cuenta en la aplicación, donde el usuario puede gestionar su suscripción o eliminar permanentemente su cuenta.

ⓘ Crear la pantalla de "Gestionar Cuenta"

Task #24 · by angx1 was closed 2 weeks ago

Figura 4.18. Issue cerrada en enero de 2026 (captura tomada a mediados de enero de 2026)

4.5. Despliegue y Lanzamiento de PANOT

Tras culminar con las fases de análisis, diseño e implementación, el proyecto entra en su fase clave de puesta en producción. Este proceso no se limita únicamente a transferir el código a un servidor, sino que engloba una serie de verificaciones de seguridad, la preparación de artefactos nativos y la gestión administrativa ante las plataformas de distribución (en este caso la App Store de Apple). El objetivo final es garantizar que el PMV sea accesible para los usuarios finales bajo los estándares de seguridad, rendimiento y diseño definidos en los capítulos anteriores (Tabla 4.9) y los impuestos por las políticas de Apple.

Dada la naturaleza académica de este proyecto y el marco normativo de la Universidad Politécnica de Madrid, el despliegue está orientado a la materialización de un artefacto funcional ajustando ciertas capacidades comerciales para alinearse con los derechos de explotación de la institución. [VER BIEN ESTO Y MODIFICARLO SI NO ES CORRECTO]

4.5.1. Preparación del Entorno y Adaptación Académica

La transición hacia una versión publicable ha requerido una configuración específica de la infraestructura y la aplicación móvil de PANOT. A diferencia de una versión comercial abierta, y con el objetivo de garantizar la sostenibilidad del proyecto a futuro, se ha desplegado una versión que prioriza las capacidades de almacenamiento local y gestión de usuarios, contactos e interacciones descritas en la especificación del sistema. En este sentido, se ha optado por una configuración que prescinde de los módulos de pago recurrente y de las llamadas activas al sistema multi-agente en el entorno desplegado. Esta decisión técnica permite que tanto el evaluador y como futuros usuarios interactúen con una aplicación estable y fluida, centrada en la experiencia de usuario y la arquitectura local-first, sin riesgos operativos o económicos.

En cuanto al cliente móvil, el proceso de empaquetado se realizó de forma manual utilizando *Xcode*. Este flujo de trabajo se inicia con la generación del proyecto nativo de iOS mediante el comando de pre-construcción (`npx expo prebuild --platform ios`) de Expo, integrando todos los módulos que componen el proyecto. Una vez dentro de *Xcode*, se procedió a la firma del código mediante certificados de distribución oficiales, paso indispensable para generar el archivo con extensión `.ipa` (iOS App Store Package) denominado binario (*binary*). Este archivo binario, constituye la entrega

final y representaría el esfuerzo de ingeniería realizado listo para su distribución.

4.5.2. Cumplimiento Normativo y Directrices de Apple

La subida del binario a la plataforma *App Store Connect* marca el inicio de una auditoría técnica y de contenido por parte de Apple. Para superar este proceso de revisión (App Review), el desarrollo de PANOT ha tenido que alinearse con las *App Review Guidelines* [9], un conjunto de reglas estrictas que garantizan la seguridad y calidad de las aplicaciones en el ecosistema iOS.

Uno de los puntos críticos ha sido el cumplimiento de la directiva sobre Privacidad y Protección de Datos (*Legal - 5.1 Privacy*). Dado que PANOT solicita acceso a información sensible como la agenda de contactos y el micrófono del dispositivo, se han tenido que implementar mensajes de propósito claros y concisos que expliquen al usuario exactamente por qué y para qué se requieren dichos permisos. Paralelamente, el diseño de PANOT se ha trazado siguiendo las Human Interface Guidelines ([HIG](#)) [10] para asegurar una experiencia basada en los pilares de *Jerarquía, Armonía y Consistencia*.

Adicionalmente, la decisión de omitir la pasarela de pagos en esta versión académica de PANOT simplifica el cumplimiento de las directrices de negocio (*Business - 3.1 Payments*), evitando la complejidad de las compras integradas obligatorias y permitiendo que el proyecto se presente como un producto de software estable y gratuito. Esta aproximación valida el sistema bajo las normativas de seguridad y diseño del ecosistema de Apple, reduciendo simultáneamente la incertidumbre legal derivada de los derechos de explotación de la Universidad Politécnica de Madrid con respecto a este proyecto.

4.5.3. Estrategia de Lanzamiento y Visibilidad

La finalización de este proceso de despliegue se alcanza con la aprobación de la aplicación por parte del equipo de revisión de Apple. Este hito no solo representa un éxito a nivel administrativo, sino que actúa como una certificación de que el software cumple con los estándares de seguridad, rendimiento y diseño exigidos por Apple. La confirmación de entrada en la *App Store* permite comenzar la fase de validación (*Medir*).

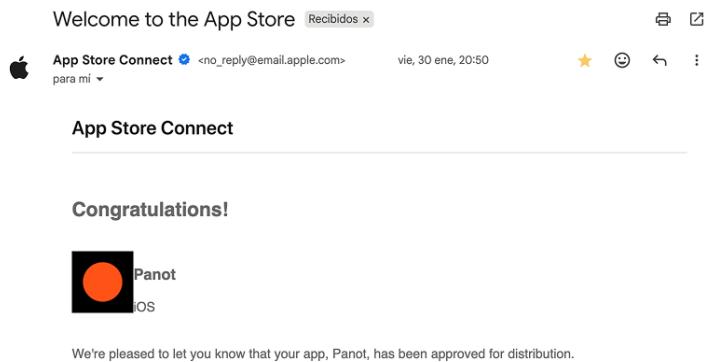


Figura 4.19. Captura del mail de Apple confirmando la aprobación de la App Review

Antes de la aprobación, la aplicación fue rechazada tres veces por incumplir la directriz de Privacidad y Protección de Datos (*Legal - 5.1 Privacy*), debido a la falta de claridad en las descripciones para el usuario (Fig. 4.20).

Version 1.0		
ACTIVITY	USER	DATE
Ready for Distribution	angel.rodriguez3rm@gmail.com	Feb 1, 2026 at 4:42 PM
Pending Developer Release	Apple	Jan 30, 2026 at 8:49 PM
In Review	Apple	Jan 30, 2026 at 8:42 PM
Waiting for Review	angel.rodriguez3rm@gmail.com	Jan 29, 2026 at 9:06 AM
Ready for Review	angel.rodriguez3rm@gmail.com	Jan 29, 2026 at 9:06 AM
Prepare for Submission	angel.rodriguez3rm@gmail.com	Jan 29, 2026 at 9:02 AM
Rejected	Apple	Jan 29, 2026 at 8:23 AM
In Review	Apple	Jan 29, 2026 at 7:57 AM
Waiting for Review	angel.rodriguez3rm@gmail.com	Jan 27, 2026 at 8:46 PM
Ready for Review	angel.rodriguez3rm@gmail.com	Jan 27, 2026 at 8:46 PM
Prepare for Submission	angel.rodriguez3rm@gmail.com	Jan 27, 2026 at 8:46 PM
Rejected	Apple	Jan 27, 2026 at 8:06 PM
In Review	Apple	Jan 27, 2026 at 7:53 PM
Waiting for Review	angel.rodriguez3rm@gmail.com	Jan 26, 2026 at 11:13 AM
Ready for Review	angel.rodriguez3rm@gmail.com	Jan 26, 2026 at 11:11 AM
Prepare for Submission	angel.rodriguez3rm@gmail.com	Jan 26, 2026 at 10:48 AM
Rejected	Apple	Jan 26, 2026 at 8:48 AM
In Review	Apple	Jan 26, 2026 at 4:56 AM
Waiting for Review	angel.rodriguez3rm@gmail.com	Jan 23, 2026 at 11:37 AM
Ready for Review	angel.rodriguez3rm@gmail.com	Jan 23, 2026 at 11:36 AM
Prepare for Submission	angel.rodriguez3rm@gmail.com	Jan 6, 2026 at 12:10 PM

Figura 4.20. Captura del historial de envíos de la aplicación a App Review

El lanzamiento principal se ha planificado en la plataforma *Product Hunt*. Al tratarse de un sistema con un alto componente de innovación, resulta el escenario más adecuado para cerrar el primer ciclo de la metodología *Lean Startup*. La visibilidad de este canal no solo busca funcionar como embudo de captación de usuarios, sino que además, permitirá recopilar métricas de comportamiento de usuarios reales que habilitan la validación de la hipótesis de valor del proyecto. De este modo, la publicación oficial en la App Store y la subsiguiente campaña de visibilidad en *Product Hunt* cierran el bloque de desarrollo del proyecto.

[INCLUIR CAPTURA DE LAUNCH EN PRODUCT HUNT]

5.

Resultados y Verificación

5.1. Interfaz de Usuario y Flujos del Sistema

El diseño de la interfaz de PANOT se ha construido, como ya se ha comentado, siguiendo las [HIG](#) de Apple. La *Jerarquía* visual permite que el usuario identifique instantáneamente las acciones críticas, como la captura de voz; la *Armonía* asegura una paleta de colores y tipografías que reducen la fatiga visual; y la *Consistencia* garantiza que los patrones de navegación (como la barra de navegación o los botones de ajustes, añadir o aceptar) sean familiares para cualquier usuario de la plataforma.

A continuación, se ilustran los flujos de la aplicación siguiendo las Historias de Usuario principales:

Flujo de Registro de Usuario y Onboarding - HU-01

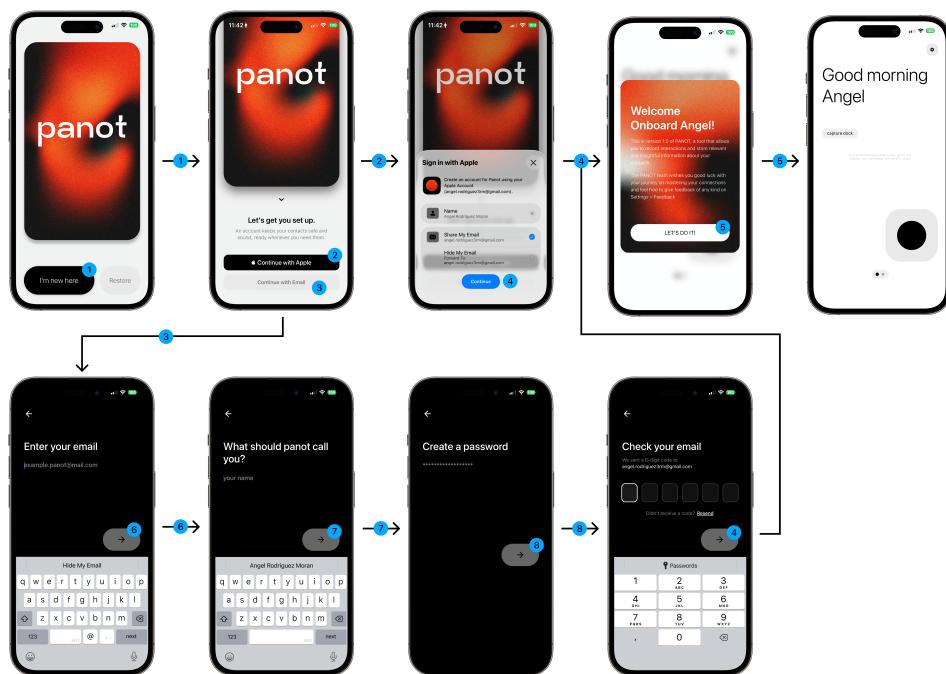


Figura 5.1. Flujo de Registro de Usuario y Onboarding

Flujo de Inicio de Sesión - HU-01

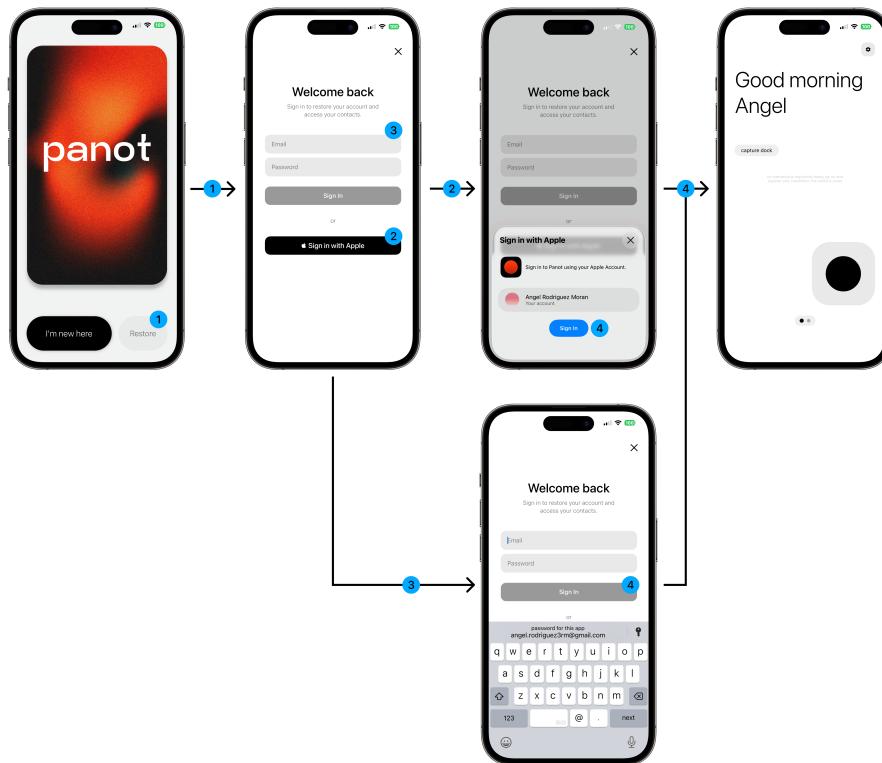


Figura 5.2. Flujo de Inicio de Sesión

Ambos flujos de Registro 5.1 e Inicio de Sesión 5.2 recogen la posibilidad de que el usuario use su *Correo electrónico* o su *Apple ID* como proveedor de autenticación.

Flujo de Creación de Contacto - HU-02 y HU-03

El siguiente flujo de ususario abarca las tres posibilidades de creación de un contacto: creación manual (camino 1 - 2 - 4 - 9), creación con dictado (camino 1 - 2 - 5 - 10 - 11) e importación del contacto (camino 1 - 2 - 6 - 7- 8).

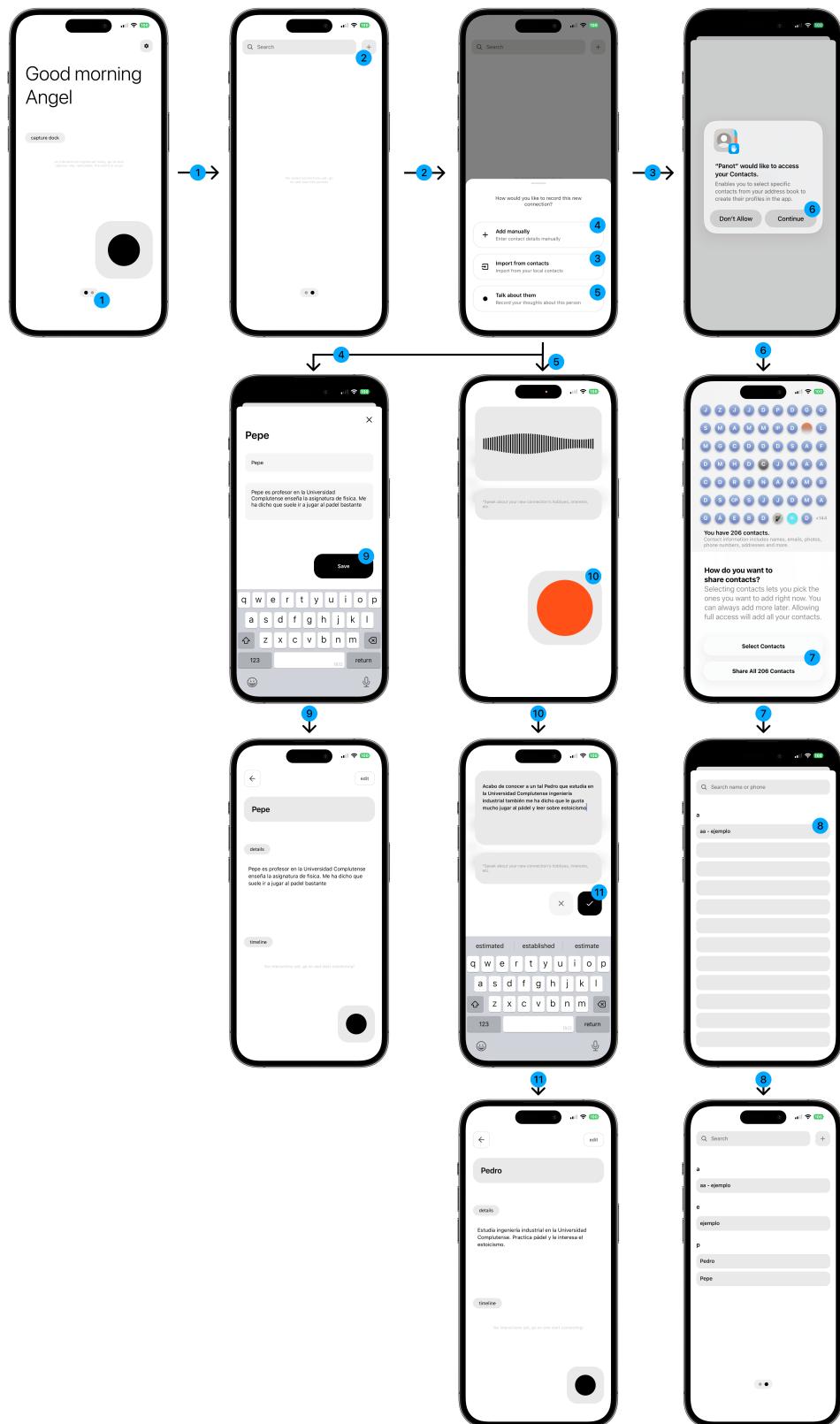


Figura 5.3. Flujo de Creación de Contacto

Para validar el funcionamiento de este proceso, se ha analizado la orquestación interna del sistema tras la creación del contacto Pedro. Como se observa en la Figura 5.4, el motor multi-agente inicia una traza de ejecución para crear el nuevo contacto con su grafo contextual correspondiente.

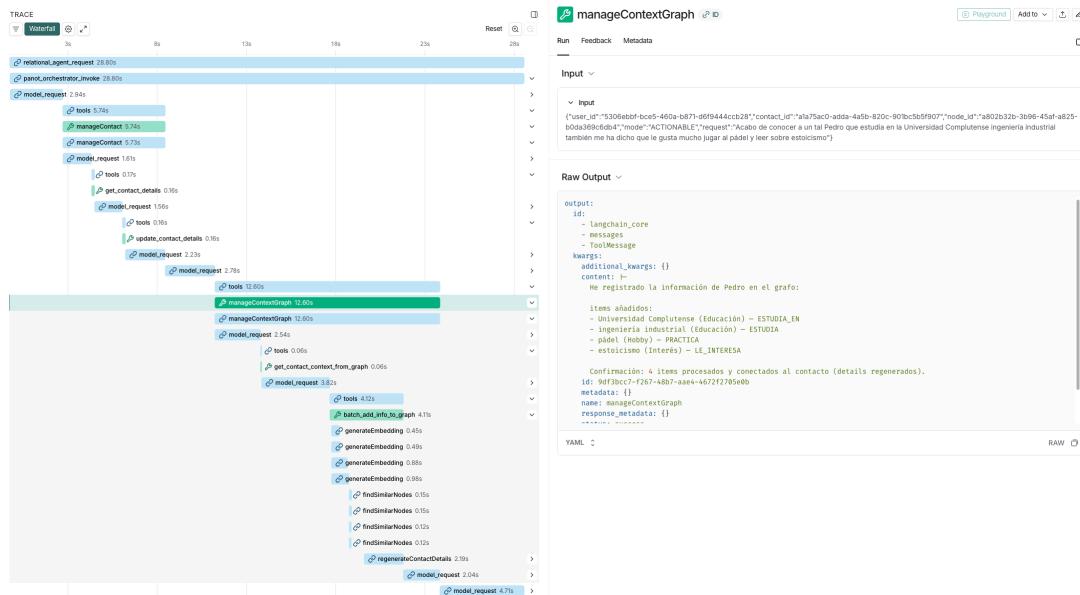


Figura 5.4. Traza de ejecución en LangSmith para la creación del contacto Pedro.

El resultado es el siguiente grafo semántico (ver Figura 5.5) donde el contacto no es una entrada tabular, sino el nodo central de una red de conceptos.

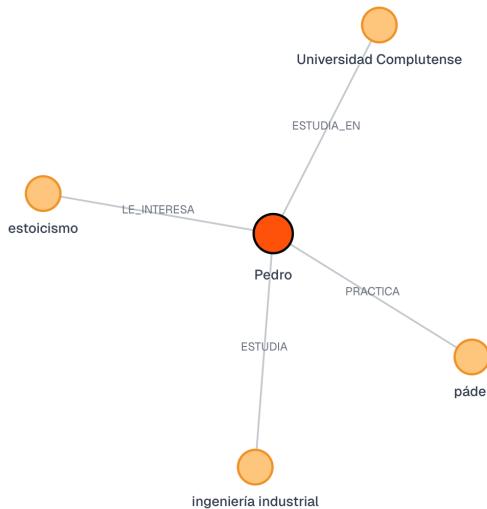


Figura 5.5. Grafo de conocimiento resultante de la creación del contacto Pedro en graph-visualizer.

Posteriormente, al crear el contacto Pepe, el sistema activa el *matching semántico* para identificar nodos comunes entre los contactos. Como se evidencia en la Figura 5.6, el sistema ha identificado de forma autónoma que ambos tienen los nodos comunes Universidad Complutense y pádel, estableciendo una conexión relacional entre los dos contactos sin intervención del usuario.

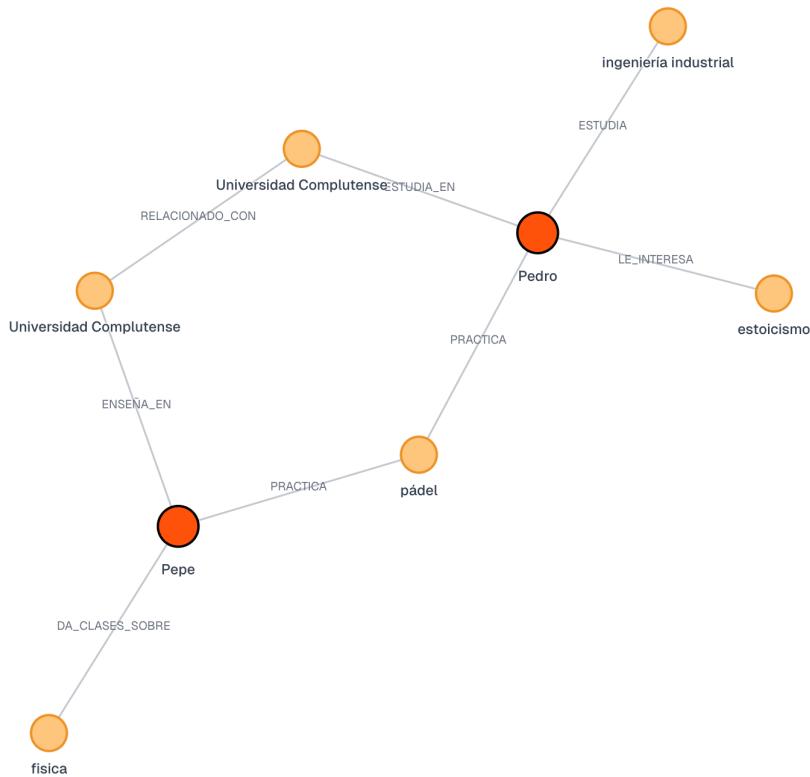


Figura 5.6. Grafo de conocimiento resultante de la creación del contacto Pepe y la detección de nodos comunes con el contacto Pedro en graph-visualizer.

Flujo de Grabación y Procesamiento de Interacción - HU-04, HU-06 y HU-08

Los siguientes flujos de creación y procesamiento de una interacción que se exponen a continuación corresponden dos variantes: la primera (**I**) corresponde a asignación de la interacción a un contacto previo a aceptar la creación de la interacción. Mientras que la segunda (**II**) corresponde a la creación de una nueva interacción sin asignarla a un contacto previamente.

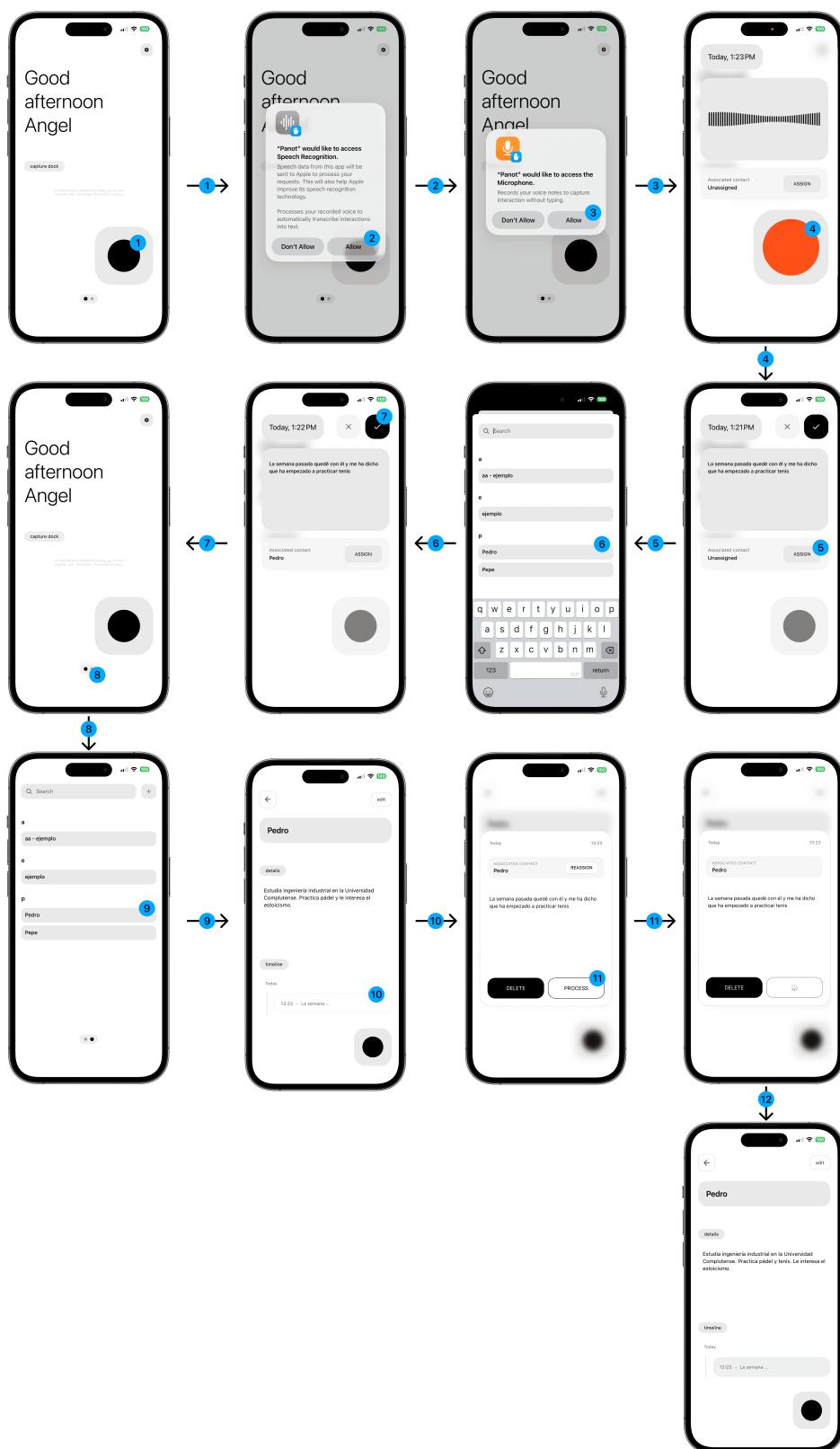


Figura 5.7. Flujo de Grabación y Procesamiento de Interacción (I)

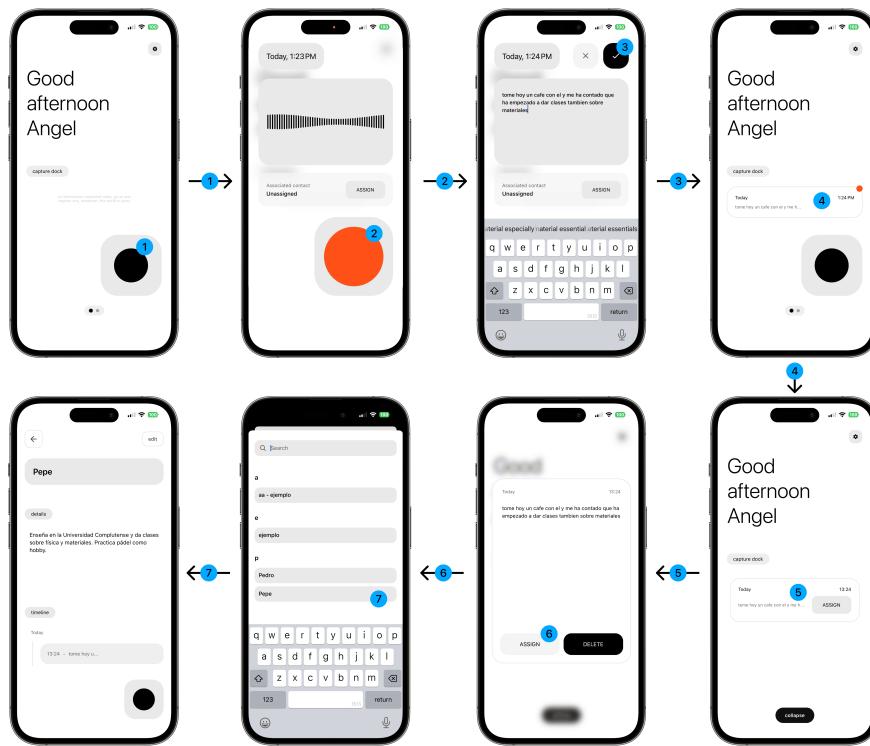


Figura 5.8. Flujo de Grabación y Procesamiento de Interacción (II)

La Figura 5.9 muestra el grafo tras procesar las interacciones reflejando la actualización e incorporación de los nuevos nodos concepto en ambos contactos.

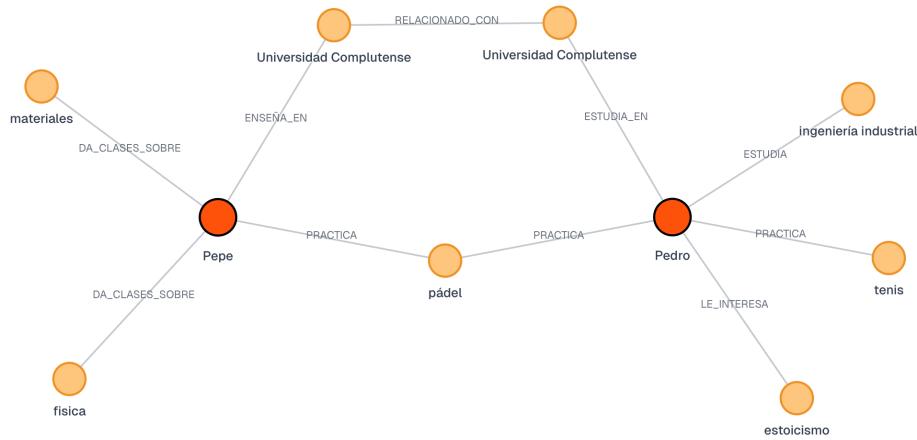


Figura 5.9. Grafo de conocimiento resultante de los contactos Pepe y Pedro tras procesar las nuevas interacciones en graph-visualizer.

Flujo de Edición o Eliminación de Contacto - HU-05

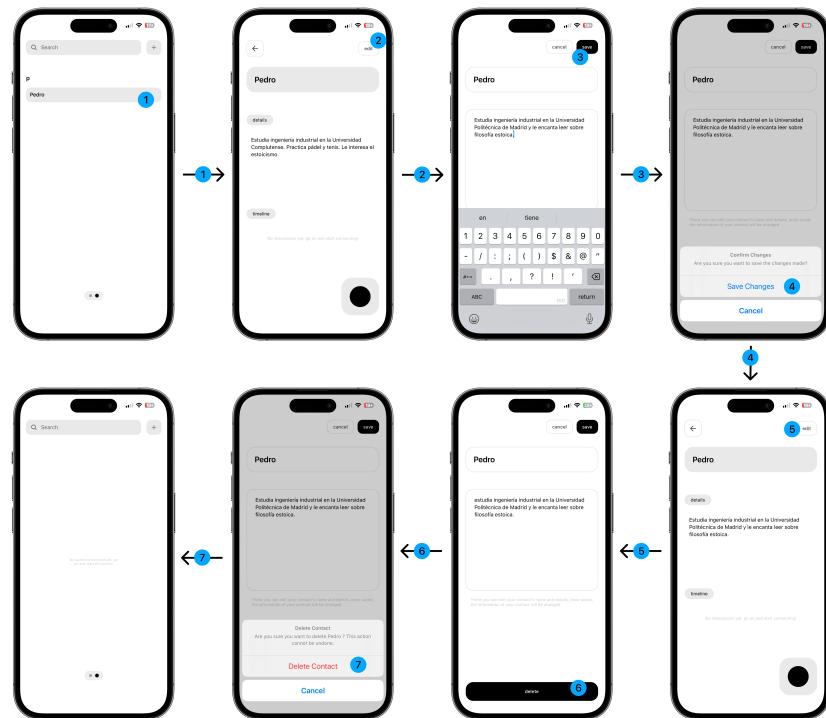


Figura 5.10. Flujo de Edición o Eliminación de Contacto

Flujo de Búsqueda de Contacto - HU-07

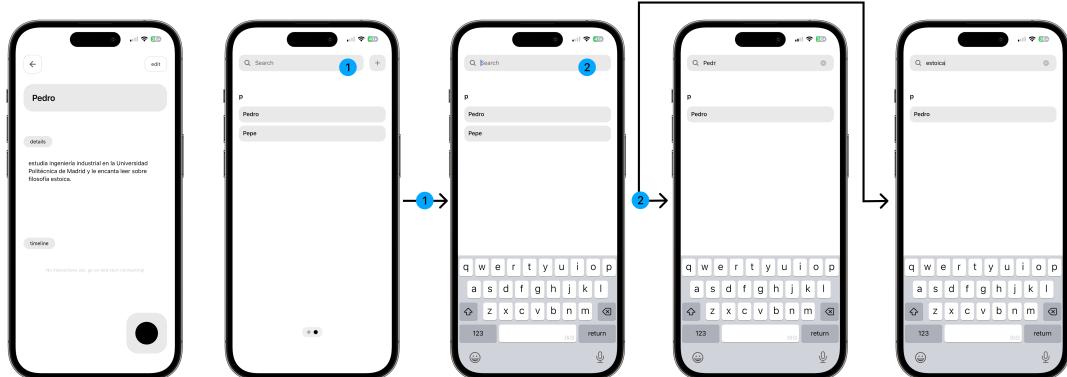


Figura 5.11. Flujo de Búsqueda de Contacto

Flujo de Soporte y Feedback - HU-09

El siguiente flujo corresponde al envío de un ticket de soporte para un bug (camino 1 - 2 - 4 - 6 - 7) o una sugerencia (camino 1 - 2 - 3 - 5 - 6 - 7).

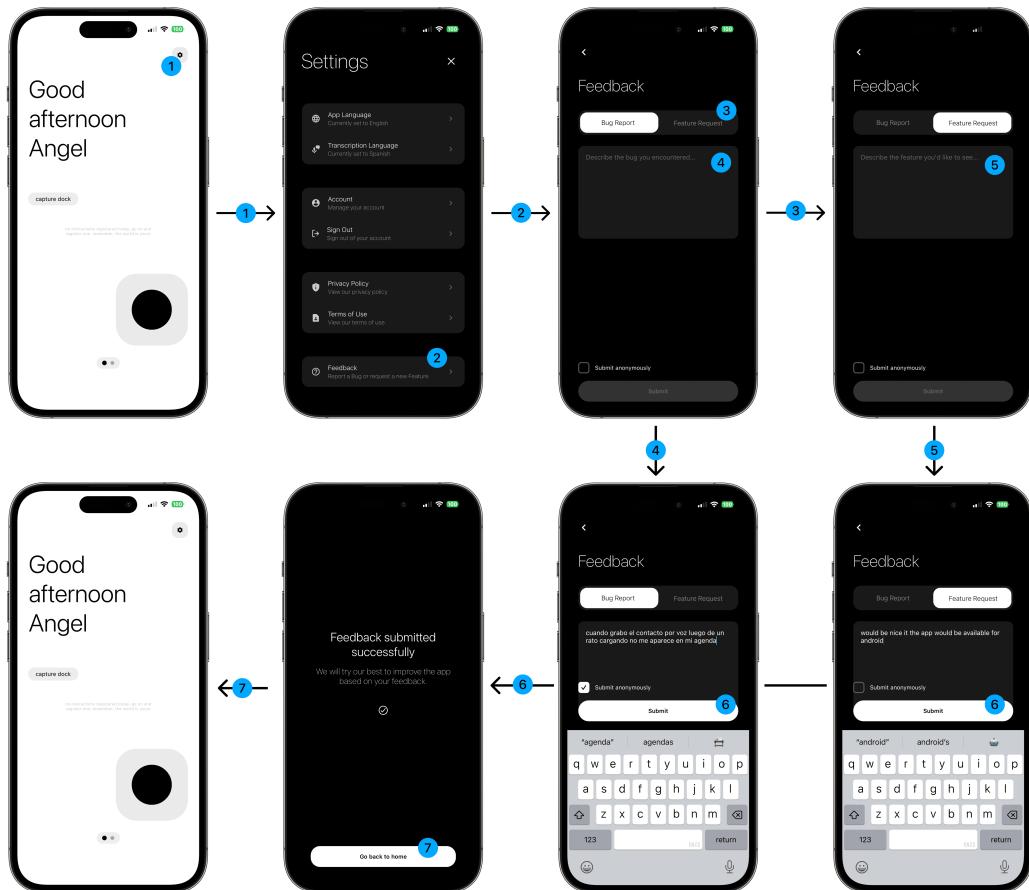


Figura 5.12. Flujo de Soporte y Feedback

Para cerrar el ciclo de soporte, se ha verificado la recepción de estos tickets en la base de datos. La Figura 5.13 muestra los registros reales en la tabla `feedback_tickets` de Supabase, confirmando que el reporte de errores y sugerencias llega correctamente al equipo de desarrollo para su gestión.

id	sender_email	feedback_type	message	created_at
21	anonymus	bug	cuento grabo el contacto por voz luego de un rato cargando no me aparece en mi agenda	2026-01-26 14:10:23.156373+00
22	angel.rodriguez3rm@gmail.com	feature	would be nice if the app would be available for android	2026-01-26 14:11:05.717141+00

Figura 5.13. Verificación de persistencia: Filas de la base de datos con los tickets de soporte enviados.

Flujo de Configuración y Ajustes

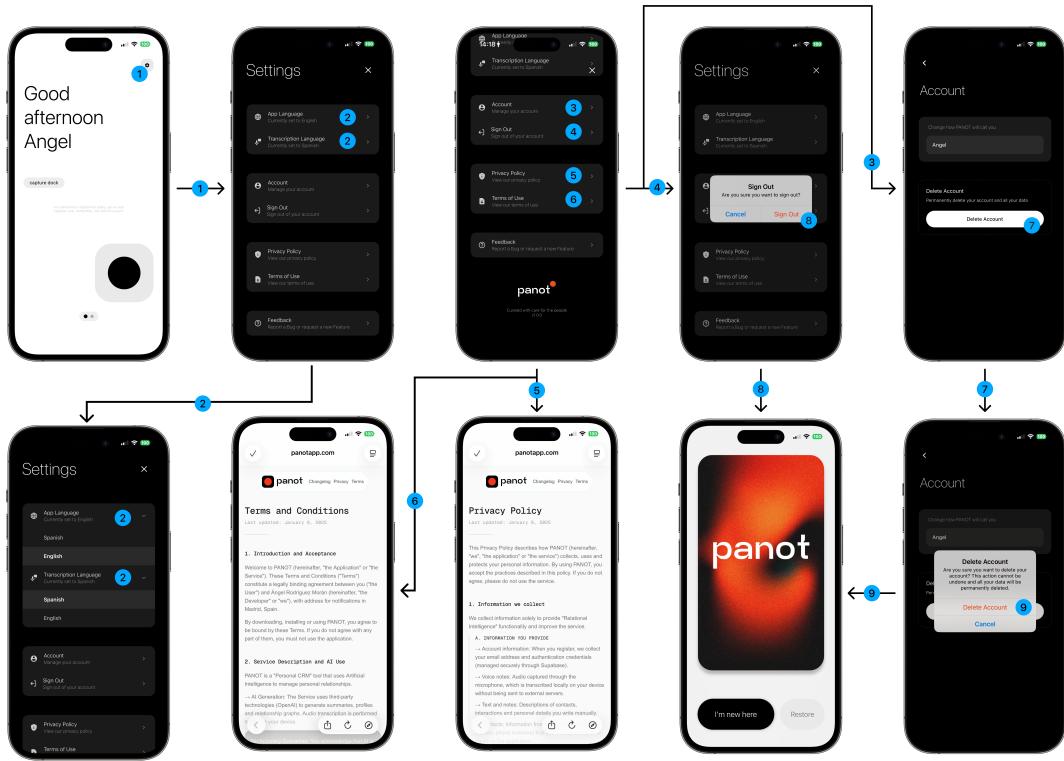


Figura 5.14. Flujo de Configuración y Ajustes

5.2. Verificación de Requisitos

Una vez expuestos los resultados visuales y técnicos del sistema, se procede a la verificación formal de los requisitos definidos en la fase de análisis. Este proceso asegura que el artefacto desarrollado cumple con la especificación original y garantiza los niveles de calidad exigidos.

5.2.1. Verificación de Requisitos Funcionales

Para validar la funcionalidad del sistema, se ha realizado un contraste entre cada requisito funcional y la evidencia generada en los flujos de usuario. En la siguiente tabla se detalla el estado de cumplimiento de cada requisito, vinculándolos con los flujos o figuras de la sección anterior.

<i>ID</i>	<i>Descripción del Requisito</i>	<i>Estado</i>	<i>Evidencia / Fuente</i>
FR-01	Registro e inicio de sesión seguro.	Logrado	Flujos 5.1 y 5.2
FR-02	Cerrar sesión del usuario de forma segura.	Logrado	Flujo 5.14
FR-03	Eliminación completa de cuenta y datos.	Logrado	Flujo 5.14
FR-04	Creación de un contacto manualmente.	Logrado	Flujo 5.3
FR-05	Importación de contactos desde agenda nativa.	Logrado	Flujo 5.3
FR-06	Creación de contacto mediante dictado de voz.	Logrado	Flujo 5.3 , Traza 5.4 y Fig. 5.5
FR-07	Búsqueda de un contacto por contexto o por nombre.	Logrado	Flujo 5.11
FR-08	Edición de un contacto.	Logrado	Flujo 5.10
FR-09	Eliminación de un contacto.	Logrado	Flujo 5.10
FR-10	Interfaz dedicada de grabación y aceptación de permisos de hardware.	Logrado	Flujos 5.7 y 5.8
FR-11	Validación de texto transcritto previo a aceptación.	Logrado	Flujos 5.7 y 5.8
FR-12	Conversión de audio a texto en tiempo real.	Logrado	Módulo panot-speech
FR-13	Descartar una interacción.	Logrado	Flujos 5.7 y 5.8
FR-14	Actualización del grafo semántico tras interacción.	Logrado	Figuras 5.6 y 5.9
FR-15	Generación de relaciones (matching semántico).	Logrado	Fig. 5.9 (Nodos comunes)
FR-16	Envío de reportes de errores o bugs.	Logrado	Flujo 5.12 y Fig. 5.13
FR-17	Envío de propuestas de mejora o solicitudes de nuevas funcionalidades.	Logrado	Flujo 5.12 y Fig. 5.13

Tabla 5.1. Tabla de verificación del cumplimiento de Requisitos Funcionales.

5.2.2. Verificación de Requisitos No Funcionales

La verificación de los requisitos no funcionales asegura que el sistema no solo sea funcional, sino que opere bajo los estándares de seguridad, rendimiento y usabilidad definidos en el apartado de [Requisitos No Funcionales](#). A continuación se detallan las pruebas y evidencias técnicas para cada indicador de calidad.

Para validar la calidad y robustez del sistema, se han sometido los indicadores de calidad a pruebas de estrés, auditorías de seguridad y análisis de observabilidad. En la siguiente tabla se sintetizan los resultados:

ID	Tipo	Resultado de la Prueba	Estado	Evidencia
RNF-SEG-001	Seguridad	Validación de políticas RLS. El acceso es denegado si el id de usuario del JWT no coincide con el <code>owner_id</code> o el <code>user_id</code> .	Logrado	Apéndice A.1
RNF-REND-002	Rendimiento	Resultados del análisis de latencia en Apéndice A.2.1 .	Logrado	Apéndice A.2
RNF-EFI-003	Eficiencia	Resultados del análisis de eficiencia económica en Apéndice A.2.2 .	Logrado	Apéndice A.2
RNF-DISP-004	Disponibilidad (I)	Operatividad completa de operaciones CRUD en modo avión verificada.	Logrado	User Acceptance Testing
RNF-DISP-005	Disponibilidad (II)	Sincronización automática tras recuperación de red en <10 segundos.	Logrado	User Acceptance Testing
RNF-USAB-006	Usabilidad (I)	Inicio de grabación accesible en 1 o 2 <i>movimientos</i> desde la <i>home</i> .	Logrado	Flujos 5.7 y 5.8
RNF-USAB-007	Usabilidad (II)	Feedback visual mediante <i>spinners</i> y etiquetas de estado en tareas largas.	Logrado	Flujo 5.7 paso 11
RNF-MANT-008	Mantenibilidad	Registro del 100 % de eventos en PostHog.	Logrado	Apéndice A.3

Tabla 5.2. Tabla de verificación del cumplimiento de Requisitos No Funcionales.

5.3. Cumplimiento de Objetivos

[CUMPLIMIENTO DE OBJETIVOS IMPUESTOS EN INTRO]

6. Conclusiones y Trabajo Futuro

6.1. Conclusiones Generales

[Escribir aquí las conclusiones generales]

6.2. Aplicación de Conocimientos Adquiridos en el Grado

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

6.3. Líneas de Trabajo Futuro

[Escribir aquí las líneas de trabajo futuro]

A. Evidencias de Verificación de Requisitos No Funcionales

A.1. RNF-SEG-001 - Seguridad

The screenshot shows two tables of RLS policies:

NAME	COMMAND	APPLIED TO
Users can insert their own edges	INSERT	public
Users can view their own edges	SELECT	public

NAME	COMMAND	APPLIED TO
Users can insert their own nodes	INSERT	public
Users can view their own nodes	SELECT	public

Below each table are the corresponding SQL alter policy commands:

```
1 alter policy "Users can insert their own edges"
2 on "public"."semantic_edges"
5 to public
6 with check (
7 | [auth.uid() = user_id]
8 );
```

```
1 alter policy "Users can view their own edges"
2 on "public"."semantic_edges"
5 to public
6 using (
7 | [auth.uid() = user_id]
8 );
```



```
1 alter policy "Users can insert their own nodes"
2 on "public"."semantic_nodes"
5 to public
6 with check (
7 | [auth.uid() = user_id]
8 );
```

```
1 alter policy "Users can view their own nodes"
2 on "public"."semantic_nodes"
5 to public
6 using (
7 | [auth.uid() = user_id]
8 );
```

Figura A.1. Política RLS para las tablas `semantic_edges` y `semantic_nodes`.

Interactions		
NAME	COMMAND	APPLIED TO
Users can delete their own interactions	DELETE	public
Users can insert their own interactions	INSERT	public
Users can update their own interactions	UPDATE	public
Users can view their own interactions	SELECT	public
Users can view their own non-deleted interactions	SELECT	authenticated


```
1 alter policy "Users can delete their own interactions"
2 on "public"."interactions"
3 to public
4 using (
5   | [auth.uid() = owner_id]
6 );
7
8 )
```



```
1 alter policy "Users can insert their own interactions"
2 on "public"."interactions"
3 to public
4 with check (
5   | [auth.uid() = owner_id]
6 );
7
8 );
```



```
1 alter policy "Users can update their own interactions"
2 on "public"."interactions"
3 to public
4 using (
5   | (auth.uid() = owner_id)
6 );
7
8 ) with check (
9   | (auth.uid() = owner_id)
10 );
```



```
1 alter policy "Users can view their own interactions"
2 on "public"."interactions"
3 to public
4 using (
5   | ((auth.uid() = owner_id) AND ((deleted = false) OR (deleted IS
6   NULL)))
7 );
8 )
```



```
1 alter policy "Users can view their own non-deleted interactions"
2 on "public"."interactions"
3 to public
4 using (
5   | ((auth.uid() = owner_id) AND ((deleted = false) OR (deleted IS
6   NULL)))
7 );
8 );
```

Figura A.2. Política RLS para la tabla interactions.

NAME	COMMAND	APPLIED TO
profiles_delete_policy	DELETE	public
profiles_insert_policy	INSERT	public
profiles_select_policy	SELECT	public
profiles_update_policy	UPDATE	public

```

1 alter policy "profiles_delete_policy"
2 on "public"."profiles"
5 to public
6 using (
7 | [user_id = auth.uid()]
8 );

```

```

1 alter policy "profiles_select_policy"
2 on "public"."profiles"
5 to public
6 using (
7 | [user_id = auth.uid()]
8 );

```

```

1 alter policy "profiles_insert_policy"
2 on "public"."profiles"
5 to public
6 with check (
7 | [(user_id = auth.uid()) OR ((auth.uid() IS NULL) AND
8 | (EXISTS ( SELECT 1
9 | FROM auth.users
10 | WHERE (users.id = profiles.user_id)))])
11 );

```

```

1 alter policy "profiles_update_policy"
2 on "public"."profiles"
5 to public
6 using (
7 | (user_id = auth.uid())
8 ) with check (
9 | (user_id = auth.uid())
10 );

```

Figura A.3. Política RLS para la tabla `profiles`.

NAME	COMMAND	APPLIED TO
Users can delete their own contacts	DELETE	public
Users can insert their own contacts	INSERT	public
Users can update their own contacts	UPDATE	public
Users can view their own contacts	SELECT	public

```
1 alter policy "Users can delete their own contacts"
2 on "public"."contacts"
3 to public
4 using (
5   | (auth.uid() = owner_id)
6 );
7
8
9
10

1 alter policy "Users can insert their own contacts"
2 on "public"."contacts"
3 to public
4 with check (
5   | (auth.uid() = owner_id)
6 );
7
8
9
10

1 alter policy "Users can view their own contacts"
2 on "public"."contacts"
3 to public
4 using (
5   | (auth.uid() = owner_id)
6 );
7
8
9
10
```

Figura A.4. Política RLS para la tabla contacts.

NAME	COMMAND	APPLIED TO
workers_delete_policy	DELETE	public
workers_insert_policy	INSERT	public
workers_select_policy	SELECT	public
workers_update_policy	UPDATE	public

```

1 alter policy "workers_delete_policy"
2 on "public"."workers"
3 to public
4 using (
5   | [user_id = auth.uid()]
6 );

```

```

1 alter policy "workers_select_policy"
2 on "public"."workers"
3 to public
4 using (
5   | [user_id = auth.uid()]
6 );

```

```

1 alter policy "workers_insert_policy"
2 on "public"."workers"
3 to public
4 with check (
5   | [(user_id = auth.uid()) OR ((auth.uid() IS NULL) AND
6     (EXISTS ( SELECT 1
7       | FROM auth.users
8       WHERE users.id = workers.user_id)))]
9 );

```

```

1 alter policy "workers_update_policy"
2 on "public"."workers"
3 to public
4 using (
5   | true
6 );

```

Figura A.5. Política RLS para la tabla workers.

process_queue			Disable RLS	Create policy	⋮
NAME	COMMAND	APPLIED TO			
Users can insert their own jobs	INSERT	public	⋮		
Users can update their process queue	UPDATE	public	⋮		
Users can view their own jobs	SELECT	public	⋮		

```
1 alter policy "Users can insert their own jobs"
2 on "public"."process_queue"
3 to public
4 with check (
5   | [auth.uid() = user_id]
6 );
7
```

```
1 alter policy "Users can update their process queue"
2 on "public"."process_queue"
3 to public
4 using (
5   | (auth.uid() = user_id)
6 ) with check (
7   | (auth.uid() = user_id)
8 );
9
10 );
```

```
1 alter policy "Users can view their own jobs"
2 on "public"."process_queue"
3 to public
4 using (
5   | [auth.uid() = user_id]
6 );
7 
```

Figura A.6. Política RLS para la tabla process_queue.

NAME	COMMAND	APPLIED TO	⋮
Allow auth users to select tickets	SELECT	authenticated	⋮
Enable insert for authenticated users only	INSERT	authenticated	⋮

```
1 alter policy "Enable insert for authenticated users only"
2 on "public"."feedback_tickets"
3 to authenticated
4 with check (
5     true
6 );
7
```

```
1 alter policy "Allow auth users to select tickets"
2 on "public"."feedback_tickets"
3 to authenticated
4 using (
5     | true
6 );
7 
```

Figura A.7. Política RLS para la tabla feedback_tickets.

A.2. RNF-REND-002 y RNF-EFI-003 - Rendimiento y Eficiencia

Para validar los requisitos de rendimiento y eficiencia económica del sistema multi-agente de PANOT, se ha realizado una evaluación de carga consistente en 27 ejecuciones consecutivas del sistema (`relational_agent`) utilizando transcripciones reales de longitud media (aprox. 300 palabras). Los datos de los resultados han sido extraídos de la plataforma de observabilidad *LangSmith*.

<input checked="" type="checkbox"/>	Name	Latency	Cost	Tokens
✓	relational_agent_request	⌚ 28.89s	\$0.00443725	22,035
✓	relational_agent_request	⌚ 37.34s	\$0.0042836	22,020
✓	relational_agent_request	⌚ 34.37s	\$0.00522335	23,168
✓	relational_agent_request	⌚ 46.75s	\$0.0046925	22,258
✓	relational_agent_request	⌚ 34.03s	\$0.00499875	22,496
✓	relational_agent_request	⌚ 40.45s	\$0.00564295	25,082
✓	relational_agent_request	⌚ 30.48s	\$0.0044586	22,251
✓	relational_agent_request	⌚ 29.06s	\$0.00522855	21,953
✓	relational_agent_request	⌚ 30.60s	\$0.0048044	22,078
✓	relational_agent_request	⌚ 31.19s	\$0.004756	22,281
✓	relational_agent_request	⌚ 34.84s	\$0.00457225	22,036
✓	relational_agent_request	⌚ 31.96s	\$0.00459985	22,389
✓	relational_agent_request	⌚ 35.08s	\$0.0057934	22,959
✓	relational_agent_request	⌚ 32.99s	\$0.00477375	22,331
✓	relational_agent_request	⌚ 35.42s	\$0.0050775	22,328
✓	relational_agent_request	⌚ 37.17s	\$0.00553835	25,687
✓	relational_agent_request	⌚ 28.09s	\$0.00519965	17,932
✓	relational_agent_request	⌚ 45.99s	\$0.00800515	34,964
✓	relational_agent_request	⌚ 32.48s	\$0.0054369	22,485
✓	relational_agent_request	⌚ 29.69s	\$0.00473625	22,118
✓	relational_agent_request	⌚ 34.50s	\$0.00543175	22,821
✓	relational_agent_request	⌚ 7.75s	\$0.0012627	5,533
✓	relational_agent_request	⌚ 32.07s	\$0.00458885	22,149
✓	relational_agent_request	⌚ 32.42s	\$0.0052383	22,443
✓	relational_agent_request	⌚ 31.39s	\$0.00516955	21,915
✓	relational_agent_request	⌚ 34.37s	\$0.006084	22,134
✓	relational_agent_request	⌚ 37.60s	\$0.00608665	21,967

Figura A.8. Lista de trazas de ejecución en LangSmith.

A.2.1. Análisis de Latencia - RNF-REND-002

Dada la naturaleza estocástica de los modelos de lenguaje, el tiempo de respuesta (T) de una solicitud presentan una alta varianza debida a la carga del modelo, el tamaño de la ventana de contexto y la profundidad de la inferencia. Para evaluar el cumplimiento del requisito **RNF-REND-002**, se analizan las siguientes métricas estadísticas sobre la muestra de las $n = 27$ trazas:

1. Latencia End-to-End (E2E) (T_{e2e}): Define el tiempo total desde el inicio del proceso hasta la respuesta final. Para cada observación i , se calcula como:

$$T_{e2e,i} = t_{final,i} - t_{inicio,i} \quad (\text{A.1})$$

2. Tiempo de Respuesta Medio (\bar{T}): La media aritmética de los tiempos de respuesta. Esta métrica es el indicador principal de rendimiento esperado bajo una carga constante.

$$\bar{T} = \frac{1}{n} \sum_{i=1}^n T_{e2e,i} \quad (\text{A.2})$$

3. Mediana (M): Representa el valor central de la muestra. En sistemas agenticos, se utiliza para neutralizar el sesgo de los valores atípicos (*outliers*) provocados por la latencia variable de las **API** de terceros. Para una muestra ordenada $x_{(1)} \leq x_{(2)} \leq \dots \leq x_{(n)}$:

$$M = \begin{cases} x_{((n+1)/2)} & \text{si } n \text{ es impar (nuestro caso: } n = 27) \\ \frac{x_{(n/2)} + x_{(n/2+1)}}{2} & \text{si } n \text{ es par} \end{cases} \quad (\text{A.3})$$

Resultados

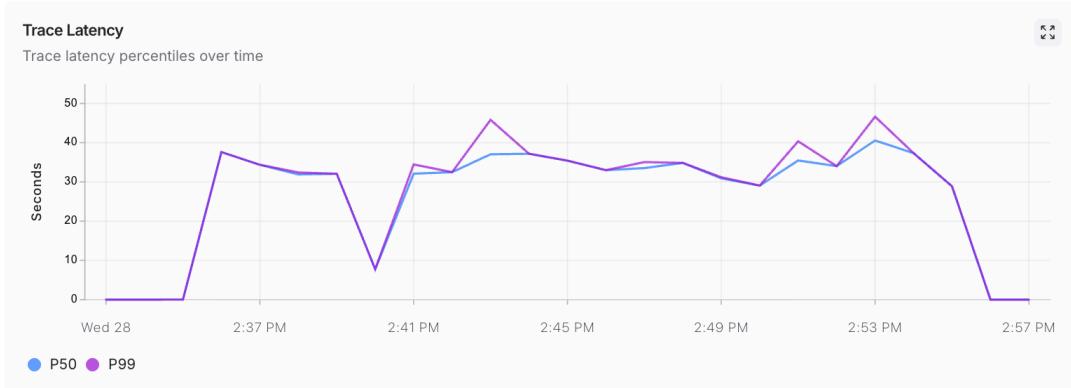


Figura A.9. Gráfica de latencia por traza en LangSmith. Listado de valores de latencia (T_{e2e}): 28.89; 37.34; 34.37; 46.75; 34.03; 40.45; 30.48; 29.06; 30.60; 31.19; 34.84; 31.96; 35.06; 32.99; 35.42; 37.17; 28.09; 45.99; 32.48; 29.69; 34.50; 7.75; 32.07; 32.42; 31.39; 34.37; 37.60

Tras el procesamiento de la muestra (Fig. A.8), los resultados arrojan una Latencia Media (\bar{T}) de 33,22 segundos y una Mediana (M) de 32,99 segundos. Ambos valores se sitúan por debajo del *threshold* de 35 segundos establecido como indicador de éxito.

A.2.2. Análisis de Eficiencia Económica - [RNF-EFI-003](#)

La eficiencia económica del sistema se evalúa en función del coste operativo directo por cada ejecución del sistema agéntico. Este coste está determinado por el consumo de recursos computacionales del procesamiento de lenguaje natural, el cual se tarifica mediante el uso de *tokens* de entrada y salida. Para validar el cumplimiento del requisito [RNF-EFI-003](#), se analizan las siguientes métricas sobre la muestra de las $n = 27$ trazas:

- Coste por Ejecución (C_i): Representa el gasto monetario de una traza individual, derivado del volumen de información procesada (I : *tokens* de entrada, O : *tokens* de salida) y el coste por *token* del modelo (τ):

$$C_i = (I_i \cdot \tau_{in}) + (O_i \cdot \tau_{out}) \quad (\text{A.4})$$

2. Coste Medio (\bar{C}): El promedio aritmético del coste de todas las ejecuciones de la muestra, utilizado para proyectar la viabilidad económica a largo plazo:

$$\bar{C} = \frac{1}{n} \sum_{i=1}^n C_i \quad (\text{A.5})$$

Resultados



Figura A.10. Gráfica de coste por traza en LangSmith. Listado de valores de coste (C_i) en USD: 0.0044; 0.0042; 0.0052; 0.0046; 0.0049; 0.0049; 0.0056; 0.0044; 0.0052; 0.0048; 0.0047; 0.0045; 0.0045; 0.0057; 0.0047; 0.0050; 0.0055; 0.0051; 0.0080; 0.0054; 0.0047; 0.0054; 0.0012; 0.0045; 0.0052; 0.0051; 0.0060; 0.0060.

Tras analizar los datos de facturación de las $n = 27$ trazas (Fig. A.10), el Coste Medio (\bar{C}) resultante es de \$0,0049 (aproximadamente 0,005 unidades monetarias). Comparando este resultado con el *threshold* de 0,01, se evidencia que el sistema opera con un margen de eficiencia del 50 % respecto al límite máximo permitido.

A.3. RNF-MANT-008 - Mantenibilidad

Para validar las hipótesis de valor planteadas en la sección 4.2.1 y asegurar la mantenibilidad del sistema a través de la toma de decisiones basada en datos, se ha instrumentado la aplicación mediante la plataforma de analítica de producto *PostHog*.

Eventos de Captura

Esta instrumentación se ha diseñado para medir el éxito y la fricción en los flujos críticos de la aplicación. Se han definido los siguientes eventos específicos que permiten monitorizar el flujo del usuario desde que inicia una acción hasta que esta se completa con éxito:

<i>Nombre de evento</i>	<i>Descripción</i>
START_MANUAL_NEW_CONTACT	Inicio de creación manual de un contacto.
CANCEL_MANUAL_NEW_CONTACT	Cancelación del flujo de creación manual de un contacto.
MANUAL_NEW_CONTACT_SUCCESS	Contacto creado correctamente de forma manual.
START_RECORDING_NEW_CONTACT	Inicio de grabación por voz para nuevo contacto.
EDIT_TEXT_RECORDING_NEW_CONTACT	Edición del texto transcrita del dictado.
CANCEL_RECORDING_NEW_CONTACT	Cancelación de la creación por dictado de creación de un contacto.
RECORDING_NEW_CONTACT_SUCCESS	Contacto creado correctamente por dictado.
START_IMPORTING_NEW_CONTACT	Inicio de importación de un contacto.
IMPORTING_NEW_CONTACT_SUCCESS	Contacto importado correctamente.
START_MANUAL_EDIT_CONTACT	Inicio de edición manual de un contacto.
CANCEL_MANUAL_EDIT_CONTACT	Cancelación de la edición del contacto.
MANUAL_EDIT_CONTACT_SUCCESS	Contacto editado correctamente.
DELETE_MANUAL_CONTACT	Eliminación de un contacto.
VIEW_CONTACT	Visualización de la ficha de un contacto.
START_INTERACTION_RECORDING	Inicio de grabación de una interacción (voz).
CANCEL_INTERACTION_RECORDING	Cancelación de la grabación de interacción.
EDIT_TEXT_RECORDING_INTERACTION	Edición del texto de la interacción grabada.
INTERACTION_RECORDING_SUCCESS	Interacción grabada correctamente.
ASSIGN_UNASSIGNED_INTERACTION	Asignación de una interacción sin asignar.
SELECT_CONTACT_TO_ASSIGN_INTERACTION	Selección de contacto para asignar la interacción.
ASSIGN_INTERACTION_SUCCESS	Interacción asignada correctamente.
VIEW_INTERACTION	Visualización del detalle de una interacción.
DELETE_INTERACTION	Eliminación de una interacción.
MANUAL_PROCESS_INTERACTION	Procesamiento manual de una interacción.

Tabla A.1. Eventos de PostHog capturados en la aplicación

Anonimización de Datos

En cumplimiento con los principios de *Privacidad desde el Diseño*, se ha implementando una capa de abstracción en el envío de métricas para que se desabiliten los campos de información sensible o geolocalización del usuario. Como se observa en el siguiente fragmento de código, se han desactivado explícitamente los campos de *GeoIP* (ciudad, país, latitud/longitud), la dirección IP, la zona horaria y la configuración de idioma local (*locale*). Esta medida asegura que las métricas obtenidas sean puramente operacionales y anónimas.

Listado A.1. Implementación de abstracción de eventos con anonimización de metadatos.

```
posthog.capture(event_name, {
  $geoip_disable: true,
  $ip: "",
  $locale: false,
  $timezone: false,
  // Desactivación de todos los campos de geolocalización
  $geoip_city_name: false,
  $geoip_country_code: false,
  ...properties,
});
```

Verificación del Flujo de Datos

La Figura A.11 muestra el flujo de eventos en tiempo real capturado durante una prueba de uso. Se confirma la recepción de los eventos lo que permite validar que el sistema de monitorización es plenamente funcional y capaz de impulsar el ciclo de aprendizaje validado de la metodología *Lean Startup* (fases *Medir-Aprender*).

EVENT ...	PERSON ...	URL / SCREEN ...	LIBRARY ...	TIME ...
VIEW_INTERACTION	019c04eb-b707-71e4-adcf-389fcc9def9a	—	posthog-react-native	a minute ago
VIEW_CONTACT	019c04eb-b707-71e4-adcf-389fcc9def9a	—	posthog-react-native	a minute ago
INTERACTION_RECORDING_SUCCESS	019c04eb-b707-71e4-adcf-389fcc9def9a	—	posthog-react-native	a minute ago
SELECT_CONTACT_TO_ASSIGN_INTERACTION	019c04eb-b707-71e4-adcf-389fcc9def9a	—	posthog-react-native	a minute ago
ASSIGN_UNASSIGNED_INTERACTION	019c04eb-b707-71e4-adcf-389fcc9def9a	—	posthog-react-native	a minute ago
EDIT_TEXT_RECORDING_INTERACTION	019c04eb-b707-71e4-adcf-389fcc9def9a	—	posthog-react-native	a minute ago
START_INTERACTION_RECORDING	019c04eb-b707-71e4-adcf-389fcc9def9a	—	posthog-react-native	a minute ago
START_INTERACTION_RECORDING	019c04eb-b707-71e4-adcf-389fcc9def9a	—	posthog-react-native	2 minutes ago
MANUAL_NEW_CONTACT_SUCCESS	019c04eb-b707-71e4-adcf-389fcc9def9a	—	posthog-react-native	2 minutes ago
START_MANUAL_NEW_CONTACT	019c04eb-b707-71e4-adcf-389fcc9def9a	—	posthog-react-native	2 minutes ago

Showing all 10 entries

Figura A.11. Flujo de eventos recibido en el panel de PostHog.

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] Supabase, *Edge Functions*, 2026. dirección: <https://supabase.com/docs/guides/functions>.
- [7] OpenAI, *Compare Models*, 2026. dirección: <https://platform.openai.com/docs/models/compare>.
- [8] D. T. J. y James P. Womack, *Lean Thinking*. Gestión 2000, 2012.
- [9] Apple, *App Review Guidelines*, 2025. dirección: <https://developer.apple.com/app-store/review/guidelines/>.
- [10] Apple, *Human Interface Guidelines*, 2025. dirección: <https://developer.apple.com/design/human-interface-guidelines>.

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. [13](#)

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. [22](#)

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. [17](#), [18](#), [III](#)

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. [14](#)

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. [14](#), [23](#), [24](#)

Eliminar Desperdicio Principio de Lean Development que se basa en eliminar cualquier esfuerzo o característica que no genere valor para el usuario. [24](#), [26](#)

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. [22](#)

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. [24](#)

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](#)

Relay Componente especializado que actúa como intermediario, permitiendo la comunicación entre un iniciador y un socio, generalmente a través de firewalls o proxies. [17](#)

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. [14](#)

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. [14](#)

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. [13](#)

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](#)

Webhook Mecanismo de comunicación basado en eventos que permite a una aplicación enviar datos automáticamente a otra en tiempo real vía HTTP, eliminando la necesidad de consultas constantes (polling) [21](#)

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. [17](#), [34](#)

Siglas

API Interfaz de Programación de Aplicaciones [15–18](#), [20](#), [33](#), [35](#), [77](#)

BaaS Banking-as-a-Service [20](#), [21](#)

CDN Content Delivery Network [11](#)

CRM Sistema de Gestión de Relaciones con Clientes [2](#)

CRUD Crear (Create), Leer (Read), Actualizar (Update), Eliminar (Delete) [49](#), [70](#)

FaaS Functions-as-a-Service [11](#), [13](#)

FHE Fully Homomorphic Encryption [12](#)

HIG Human Interface Guidelines de Apple [56](#), [59](#)

HU Historias de Usuario [28](#)

IA Inteligencia Artificial [3](#), [23](#), [35](#)

JWT JSON Web Token [17](#), [34](#), [70](#)

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

PMV Producto Mínimo Viable [23–25](#), [27](#), [28](#), [30](#), [46](#), [48](#), [52](#), [54](#), [55](#), [VII](#)

PRM Gestión de Relaciones Personales [2](#)

SDK Kit de Desarrollo de Software [21](#), [22](#)

TPP Tecnologías de Preservación de la Privacidad [12](#)

WIP Work In Progress [26](#), [III](#)

