UNIVERSIDAD POLITÉCNICA DE MADRID

E.T.S. DE INGENIERÍA DE SISTEMAS INFORMÁTICOS

PROYECTO FIN DE GRADO

GRADO EN INGENIERÍA DE COMPUTADORES

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, 7 de noviembre de 2025

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, 7 de noviembre de 2025

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{2025ngelRodrguezMorn,
    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 = {2025},
    month = {11},
}
```

[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. Motivación

[Escribir aquí la motivación del proyecto]

1.2. Descripción y Alcance del Proyecto

[Escribir aquí la descripción y alcance del proyecto]

1.3. Objetivos

[Escribir aquí los objetivos del proyecto]

1.4. Estructura de la memoria

[Escribir aquí la estructura de la memoria]

2. Estado de la Técnica y Contexto Tecnológicos

2.1. Gestión de la Inteligencia Relacional

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

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.

2.1.1. Inteligencia Relacional en el Contexto de la Inteligencia Artificial

Para comprender el alcance de la Inteligencia Relacional, 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 IA radica en que, mientras estos últimos operan mediante asociaciones estadísticas entre patrones de entrada y salida —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[1]. La Inteligencia Relacional captura la estructura relacional subyacente que puede transferirse entre dominios aparentemente no relacionados, tal como ocurre en el razonamiento humano, creando un contexto de conocimiento más amplio y generalizable o especfíco para un dominio en concreto.

La investigación en Inteligencia Relacional demuestra capacidades que van más allá del aprendizaje estadístico tradicional. [INTRODUCIR CITACIÓN PAPER] muestran cómo un modelo computacional puede aprender representaciones relacionales estructuradas y realizar generalización de cero disparos[2] entre dominios completamente diferentes, como la transferencia de conocimiento entre videojuegos. Esta capacidad de generalización permite que el sistema aprenda a reconocer y comprender relaciones entre entidades en contextos completamente diferentes.

Traspasando la analogía de los videojuegos al contexto de PANOT, la Inteligencia Relacional como se menciona en doumas2022theory 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, PANOT puede construir representaciones relacionales dinámicas que capturan la estructura subyacente de las relaciones humanas —como la evolución temporal de intereses comunes, la frecuencia contextual de interacciones, o los cambios en preferencias y circunstancias vitales—.

Por ejemplo, el sistema puede reconocer que ciertos patrones de comunicación efectivos en relaciones profesionales pueden generalizarse a nuevas relaciones profesionales, o que cambios detectados en el contexto de una relación personal pueden aplicarse para comprender dinámicas similares en otras relaciones. Esta generalización relacional es lo que permite que PANOT evolucione continuamente cada contacto, reflejando no solo quién es esa persona en términos estáticos, sino cómo ha evolucionado y continúa evolucionando la relación según el contexto temporal y situacional.

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

Input: El usuario captura una interacción mediante nota de voz: “Acabo de almorzar con María. Está muy emocionada porque ha conseguido un nuevo trabajo como diseñadora en una startup tecnológica. Le interesa especialmente el trabajo remoto y

[1] Transmisión de conocimientos de una situación a otra

[2] 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

mencionó que está buscando un piso más cerca de su nueva oficina. Hablamos de proyectos de diseño colaborativo y se mostró muy receptiva a la idea de futuros proyectos juntos."

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

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

El sistema construye una representación relacional estructurada que conecta estas entidades (usuario-contacto) mediante relaciones semánticas representadas en formato `JSON`:

```
{
  "ha-cambiado-preferencia": {
      "preferencia" : "trabajo-remoto"
  },
  "interés-común": {
      "entidades" : [ "Usuario", "María" ],
      "tema": [ "diseño", "startup-tecnológica", "diseño-colaborativo" ]
  },
  "contexto-temporal-situacional": {
      "evento" : "almuerzo-informal",
      "estado" : "transición-profesional"
  }
}
```

Output: PANOT actualiza dinámicamente el contacto de María, generando múltiples outputs contextuales:

- *Actualización automática del perfil*: se añade "Diseñadora en startup tecnológica" como situación laboral actual y se marca "Trabajo remoto" como preferencia. Se registra también el cambio de estado como una nueva etapa profesional.
- *Recordatorio contextual*: en la ficha de María se crea podría crear un recordatorio automático para preguntar sobre la búsqueda de piso en futuras interacciones.

- *Recomendaciones de conversación*: el sistema podría sugerir, si el usuario lo desea, abordar temas de "diseño colaborativo" y "startups tecnológicas" en próximas reuniones, reforzando el interés común detectado en el `JSON`.

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.

[introducir figura de grafo de Maria]

2.1.2. Arquitecturas Similares: Grafos Contextuales en Sistemas de Agentes

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

En arquitecturas de agentes modernas, el grafo contextual actúa como una memoria estructurada que almacena información sobre interacciones pasadas, estados del entorno y relaciones entre diferentes entidades. Esta estructura permite a los agentes:

- *Almacenar información de forma estructurada*: Capturar y organizar datos sobre estados, eventos y relaciones de manera que preserve la estructura relacional subyacente, en lugar de almacenar información de forma descontextualizada.
- *Acceder rápidamente a información relevante*: Navegar eficientemente por el grafo para recuperar datos pertinentes según la situación actual, mejorando significativamente la eficiencia en la recuperación de información comparado con búsquedas en bases de datos relacionales tradicionales.
- *Facilitar el razonamiento complejo*: Utilizar la estructura del grafo para inferir nuevas relaciones y tomar decisiones informadas mediante razonamiento multihop

—la capacidad de realizar inferencias a través de múltiples pasos siguiendo las conexiones del grafo—.

- *Adaptabilidad y aprendizaje continuo*: Actualizar y expandir el conocimiento de manera dinámica, incorporando nuevas interacciones y relaciones sin requerir reestructuración completa de la base de datos.

La eficiencia superior de las estructuras de grafo para la memoria de los agentes radica en la diferencia fundamental entre el modelo de acceso a datos relacionales y el modelo de navegación por grafos. En las bases de datos relacionales, recuperar información sobre relaciones entre entidades requiere realizar múltiples operaciones que pueden cruzar tablas diferentes, lo cual implica escanear índices y realizar comparaciones entre grandes volúmenes de datos. La complejidad de estas operaciones crece exponencialmente con el número de relaciones involucradas, resultando en tiempos de consulta que pueden ser 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 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,

aunque veremos más adelante que el enfoque de PANOT es algo diferente.

2.2. Cambio de Paradigma en el Diseño de Producto

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

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

La irrupción de la Inteligencia Artificial como tecnología dominante ha transformado radicalmente el panorama del diseño de productos digitales. La democratización de las capacidades de IA ha generado un desplazamiento del valor diferencial de los productos: ya no es suficiente ofrecer una funcionalidad única o una interfaz atractiva, pues estas características pueden replicarse rápidamente. En este nuevo contexto, la diferenciación competitiva se desplaza hacia dimensiones más profundas y fundamentales de la experiencia humana.

2.2.1. Valores Diferenciadores en la Era de la IA

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

de partida, el producto ha de centrarse en en los siguientes dos principios:

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

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

2.3. Principios de Privacidad y Eficiencia por Diseño

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

2.3.1. Principio de Privacidad

La *Privacidad desde el Diseño* (Privacy by Design, PbD) constituye un enfoque proactivo que integra la protección de datos personales desde las primeras fases de desarrollo de productos, servicios o procesos. Este concepto, desarrollado inicialmente por Ann Cavoukian en la década de los 90 y reconocido internacionalmente en la 32ª Conferencia Internacional de Comisionados de Protección de Datos y Privacidad celebrada en Jerusalén en 2010, ha sido consolidado legalmente en el Reglamento General de

Protección de Datos (RGPD) mediante su artículo 25, que establece la obligación de implementar medidas técnicas y organizativas apropiadas para garantizar la protección de datos desde el diseño y por defecto.

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

1. *Proactivo, no reactivo; preventivo, no correctivo*: La privacidad debe anticiparse a los riesgos antes de que se materialicen, implementando medidas preventivas en lugar de soluciones correctivas posteriores.

2. *La privacidad como configuración predeterminada*: Los sistemas deben proteger los datos personales por defecto, sin requerir acción adicional del usuario. La configuración más privada debe ser la opción predeterminada.

3. *Privacidad incorporada en la fase de diseño*: Las medidas de protección deben integrarse desde el inicio del desarrollo, evitando soluciones añadidas posteriormente que puedan ser menos efectivas o generar fricciones en la experiencia del usuario.

4. *Funcionalidad total: pensamiento "todos ganan"*: Los objetivos de privacidad no deben comprometer la funcionalidad del sistema. El diseño debe equilibrar ambos aspectos, garantizando que la protección de datos y las funcionalidades esenciales coexistan sin sacrificios mutuos.

5. *Aseguramiento de la privacidad en todo el ciclo de vida*: La protección de datos debe mantenerse durante todas las etapas del ciclo de vida del sistema, desde su concepción hasta su retirada, incluyendo desarrollo, puesta en producción, operación, mantenimiento y eliminación.

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

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

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

2.3.2. Principio de Eficiencia

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

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

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

2.3.3. Aplicación en Sistemas con Modelos de Lenguaje

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

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

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

sin que el proveedor del servicio pueda acceder al contenido real de la información.

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

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

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

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

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

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

2.3.4. Aplicación en PANOT

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

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

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

A nivel de infraestructura, se ha utilizado Supabase como plataforma de backend como ya se explicará en puntos posteriores. Supabase tiene la particularidad de 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 TLS (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 (RLS) y Column Level Security (CLS), 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 [incluir citación o explicación de estándares], proporcionando una base de cumplimiento normativo adicional.

La aplicación de los principios de eficiencia por diseño en PANOT se ha materializado mediante la adopción de una metodología de desarrollo inspirada en el enfoque *Lean Startup* propuesto por Eric Ries [CITAR EL LIBRO]. 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 y mínimas necesarias para cumplir con las expectativas del usuario final y de los requisitos funcionales y no funcionales de la aplicación.

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

2.4. Contexto Tecnológico

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

2.4.1. Desarrollo de la aplicación 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 y APIs pre-

configuradas que reducen la complejidad de configuración del proyecto y aceleran el tiempo de desarrollo, permitiendo enfocar los esfuerzos en la implementación de funcionalidades de valor. Además *Expo* es un framework basado en *React* y en el caso de PANOT, *Typescript* lo que facilita el desarrollo modular, la reutilización de componentes y el tipado de los artefactos del proyecto debido a la naturaleza de estos lenguajes.

- Compatibilidad multiplataforma: Aunque el proyecto se centra inicialmente en iOS, *Expo* facilita la extensión futura a Android con mínimos cambios en el código base, garantizando una base sólida para el crecimiento del producto.
- Ecosistema Open Source: *Expo* cuenta con una comunidad activa y documentación muy completa, lo que facilita el aprendizaje y la resolución de problemas.
- Acceso a funcionalidades nativas: A través de módulos nativos y APIs expuestas por *Expo*, se mantiene acceso a capacidades del dispositivo como notificaciones push, almacenamiento local y sensores, sin requerir la complejidad del desarrollo nativo puro.

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

- Desarrollo nativo con *Swift* y *SwiftUI*: Esta opción habría proporcionado un rendimiento óptimo y acceso completo a todas las capacidades nativas de iOS. Sin embargo, fue descartada debido a su mayor complejidad de configuración, un tiempo de desarrollo más extenso y la limitación a una única plataforma, aspectos que no se alinean con los objetivos de eficiencia y escalabilidad del proyecto.
- Desarrollo multiplataforma con *Flutter*: También se consideró el uso de *Flutter*, un framework desarrollado por Google que permite crear aplicaciones móviles utilizando *Dart* y un motor de renderizado propio, compilando a código nativo. *Flutter* permite construir interfaces visuales consistentes en diferentes plataformas y evita la dependencia de componentes nativos del sistema operativo, similar a *Expo*. Sin embargo, esta alternativa también fue descartada principalmente debido a la poca familiaridad con el lenguaje *Dart*, lo que habría alargado inevitablemente el tiempo de desarrollo del proyecto.

2.4.2. Persistencia de datos y gestión local

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

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

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

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

- Simplicidad de integración: *Supabase* proporciona un cliente JavaScript/TypeScript que se integra naturalmente con React Native y Expo, facilitando la sincronización bidireccional entre el estado local y la base de datos.
- Escalabilidad: PostgreSQL es un motor de bases de datos relacional robusto y escalable, capaz de gestionar grandes volúmenes de datos y relaciones complejas entre entidades.
- Sincronización en tiempo real: *Supabase* ofrece capacidades de suscripción a cam-

bios en tiempo real, permitiendo que actualizaciones realizadas desde otros dispositivos se reflejen automáticamente en la aplicación.

- Seguridad integrada: Como se mencionó en la sección de privacidad, *Supabase* proporciona mecanismos de seguridad a nivel de fila y columna que garantizan el aislamiento de datos entre usuarios y encriptación de datos en todo su ciclo de vida.
- Coste de uso: el plan gratuito de *Supabase* ofrece llamas ilimitadas a su API, gestión de hasta 500.000 usuarios activos, y hasta 500MB de almacenamiento de datos, lo que supone una base sólida para hacer posible el desarrollo del proyecto.

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

2.4.3. Gestión de servicios externos mediante API Gateway

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

Las *Edge Functions* de Supabase son funciones serverless escritas en TypeScript y ejecutadas en un runtime basado en *Deno*, distribuidas globalmente en edge nodes cercanos a los usuarios. La arquitectura de gateway implementada por Supabase sigue el siguiente flujo de procesamiento:

1. Entrada de solicitud en el edge gateway: El gateway actúa como relay que enruta el tráfico, gestiona los headers de autenticación, valida tokens JWT y aplica reglas de enrutamiento y control de tráfico.
2. Aplicación de autenticación y políticas: El gateway (o la función) valida los JWTs de Supabase, aplica rate limiting y centraliza las comprobaciones de seguridad antes de ejecutar el código, garantizando que solo solicitudes autenticadas y autorizadas accedan a los servicios externos.

3. Ejecución en el edge runtime: La función se ejecuta en un nodo de Edge Runtime distribuido regionalmente más cercano al usuario, minimizando la latencia de procesamiento.
4. Integraciones y acceso a datos: Las funciones comúnmente invocan APIs de Supabase (Auth, Postgres, Storage) o APIs de terceros, utilizando estrategias de conexión optimizadas para entornos edge/serverless.
5. Observabilidad y logs: Las invocaciones emiten logs y métricas que pueden explorarse en el dashboard de Supabase o integrarse con sistemas de monitorización downstream como Sentry.
6. Respuesta a través del gateway: El gateway reenvía la respuesta al cliente y registra los metadatos de la solicitud para análisis y auditoría posterior.

Esta arquitectura de API Gateway proporciona múltiples beneficios que justifican su adopción:

- Seguridad centralizada: Todas las claves de API y credenciales de servicios externos se mantienen en el servidor, nunca expuestas al cliente móvil. El gateway valida automáticamente la autenticación antes de procesar cualquier solicitud, reduciendo significativamente la superficie de ataque.
- Baja latencia: La distribución global de las funciones garantiza que las solicitudes se procesen en el nodo más cercano geográficamente al usuario, reduciendo el tiempo de respuesta total.
- Escalabilidad automática: Las Edge Functions se escalan automáticamente según la demanda, eliminando la necesidad de gestionar infraestructura de servidores y optimizando los costes operativos mediante un modelo de pago por uso.
- Simplificación del desarrollo: La integración nativa con las APIs de Supabase permite que las funciones accedan directamente a la base de datos, autenticación y almacenamiento sin configuración adicional, reduciendo la complejidad del código y la gestión de credenciales.
- Rate limiting y control de tráfico: El gateway proporciona mecanismos integrados para limitar la frecuencia de solicitudes, protegiendo tanto los servicios externos como la infraestructura propia de sobrecargas.

[incluir brevemente alternativas consideradas]

2.4.4. Capa de inteligencia artificial

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

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

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

[TABLA COMPARATIVA DE LOS TRES MODELOS DE OPENAI]

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

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

[TABLA COMPARATIVA DE AMBOS MODELOS]

2.4.5. Pasarela de pagos

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

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

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

2.4.6. Métricas y análisis de uso

Una parte crítica en el desarrollo de productos digitales, es tener un claro entendimiento de cómo los usuarios interactúan con el producto. Para ello, es esencial que los desarrolladores tengan documentadas métricas de éxito y uso de funcionalidades, con

el objetivo de poder tomar decisiones informadas basadas en datos reales, teniendo en cuenta qué aspectos de la aplicación están aportando valor al usuario y cuales no.

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

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

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

Como alternativas se consideraron *Mixpanel*, que ofrece herramientas avanzadas para el análisis de productos y retención de usuarios con un SDK robusto para React Native, y *Amplitude*, una plataforma líder en análisis de productos que proporciona capacidades avanzadas de análisis de cohortes y embudos [AÑADIR EXPLICACION DE AMBAS??]. Sin embargo, ambas alternativas presentan modelos de precios más restrictivos en sus planes gratuitos comparado con *PostHog*, y requieren una configuración más compleja para su integración con Expo. También se evaluó *Google Analytics*, una solución gratuita y ampliamente utilizada, pero su enfoque está más orientado al análisis de tráfico web que al análisis de producto específico para aplicaciones móviles.

3. Desarrollo del Proyecto

3.1. Metodología y Entorno de Desarrollo

[Escribir aquí la metodología y entorno de desarrollo del proyecto]

3.1.1. Gestión Ágil del Proyecto con GitHub Projects (Git Flow + Kanban)

[Escribir aquí la gestión ágil del proyecto con GitHub Projects (Git Flow + Kanban)]

3.1.2. Estrategia de Ramificación y Control de Versiones

[Escribir aqui la estrategia de ramificacion y control de versiones]

3.1.3. Estructura de Repositorios en la Organización

[Escribir aqui la estructura de repositorios en la organizacion]

3.2. Especificación de Requisitos de Software

[Escribir aqui la especificacion de requisitos de software]

3.3. Diseño de la Arquitectura del Sistema

[Escribir aqui el diseño de la arquitectura del sistema]

3.4. Fases de la Implementación

[Escribir aqui las fases de la implementacion]

3.5. Despliegue y Lanzamiento de PANOT

[Escribir aqui el despliegue y lanzamiento de PANOT]

4. Verificación y Resultados

[Escribir aquí los resultados y evaluación del proyecto]

5. Conclusiones y Trabajo Futuro

5.1. Conclusiones Generales

[Escribir aquí las conclusiones generales]

5.2. Aplicación de Conocimientos Adquiridos en el Grado

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

5.3. Líneas de Trabajo Futuro

[Escribir aquí las líneas de trabajo futuro]

Índice de términos