

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, 6 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, 6 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},  
}
```

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



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

[Cita opcional para el proyecto]

— [Autor de la cita]

Agradecimientos

[Escribir aquí los agradecimientos del proyecto]

Resumen

mas a delante...

Palabras clave:

Abstract

[Write here the project summary in English]

Keywords:

Índice general

1	Introducción	1
1.1	Motivación	1
1.2	Descripción y Alcance del Proyecto	1
1.3	Objetivos	1
1.4	Estructura de la memoria	1
2	Estado de la Técnica y Fundamentos Tecnológicos	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	8
2.4	Tecnologías para el Desarrollo de Aplicaciones para iOS	14
2.5	Contexto tecnológico	17
3	Desarrollo del Proyecto	18
3.1	Metodología y Entorno de Desarrollo	18
3.2	Especificación de Requisitos de Software	18
3.3	Diseño de la Arquitectura del Sistema	18
3.4	Fases de la Implementación	19
3.5	Despliegue y Lanzamiento de PANOT	19

4 Verificación y Resultados	20
5 Conclusiones y Trabajo Futuro	21
5.1 Conclusiones Generales	21
5.2 Aplicación de Conocimientos Adquiridos en el Grado	21
5.3 Líneas de Trabajo Futuro	21

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 Fundamentos 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 analógica¹. 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. [INTRODUCIR CITACIÓN PAPER] muestran cómo un modelo computacional puede aprender representaciones relacionales estructuradas y realizar generalización de cero disparos² 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

¹Transmisión de conocimientos de una situación a otra

²Escenario 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. 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 IA —donde funcionalidades que antes requerían desarrollo especializado ahora están disponibles mediante APIs u otros servicios— 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 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 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.

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 la fase de diseño arquitectónico.

El principio fundamental que guía la eficiencia por diseño en ingeniería de software fue enunciado por Donald Knuth en su obra *The Art of Computer Programming*: "La optimización prematura es la causa de todos los males" [INCLUIR CITACIÓN]. Este principio establece que, aunque la eficiencia es un objetivo importante, debe lograrse mediante un enfoque equilibrado que priorice primero la funcionalidad y la claridad del código, para posteriormente optimizar de manera iterativa aquellos componentes que realmente requieren mejoras de rendimiento.

En la práctica, las empresas de software modernas implementan el principio de eficiencia por diseño mediante diversas estrategias arquitectónicas. Por ejemplo, la adop-

ción 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 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 sanitarios [AÑADIR CITA AL TRABAJO DE MIGUEL], ya que permite enviar datos sensibles sobre pacientes cifrados 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 con modelos de lenguaje 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 sistemas que utilizan modelos de lenguaje, es necesario considerar dos dimensiones principales: el coste computacional y el tiempo de respuesta.

Una estrategia arquitectónica fundamental para optimizar la eficiencia en estos sistemas 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 pilares 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 Minstral 3B de Mistral, suponen un coste computacional menor, pero no ofrecen las mismas capacidades que los prime-

ros.

[insertar tabla de costes 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 ingenierado 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 gobernador 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 auto-gestionar estas medidas de seguridad a través de dos mecanismos: Por un lado, todos los datos se encriptan en reposo mediante AES-256 y en tránsito mediante 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. Tecnologías para el Desarrollo de Aplicaciones para iOS

2.4.1. Desarrollo Nativo

El desarrollo nativo para iOS consiste en crear aplicaciones utilizando específicamente las herramientas, lenguajes y frameworks proporcionados por Apple para la plataforma. Este enfoque permite aprovechar al máximo las capacidades del sistema operativo iOS y de los dispositivos de Apple, garantizando un rendimiento óptimo y acceso completo a todas las funcionalidades del sistema.

El lenguaje de programación predominante en el desarrollo nativo para iOS es *Swift*, introducido por Apple en 2014 como sucesor de *Objective-C*. *Swift* es un lenguaje moderno, seguro y de alto rendimiento que combina características de programación orientada a objetos y funcional, diseñado específicamente para ser más intuitivo y menos propenso a errores que su predecesor. Aunque *Objective-C* se sigue manteniendo en proyectos legados o de alto riesgo de migración.

La herramienta principal para el desarrollo nativo es *Xcode*, el entorno de desarrollo integrado (IDE) oficial de Apple. Xcode proporciona un editor de código, compilador, depurador, simulador de iOS, herramientas de gestión de interfaces y un sistema completo de gestión de proyectos. Además, incluye *Interface Builder* para diseñar interfaces de usuario de forma visual e *Instruments* para análisis de rendimiento y detección de fugas de memoria.

Para la construcción de interfaces de usuario, Apple proporciona dos frameworks principales: *UIKit*, el framework tradicional basado en programación imperativa y eventos, y *SwiftUI*, introducido en 2019, que utiliza un paradigma declarativo y permite crear interfaces de manera más moderna y eficiente. SwiftUI facilita el desarrollo de interfaces adaptativas y reactivas, aunque UIKit sigue siendo ampliamente utilizado, especialmente en proyectos existentes o cuando se requiere mayor control sobre el comportamiento de la interfaz. El desarrollo nativo también permite el uso directo de *Cocoa Touch*, la capa de frameworks de iOS que incluye servicios fundamentales como *Core Data* para persistencia de datos, *Core ML* para modelos de lenguaje, *Core Location* para servicios de ubicación, y acceso completo a las APIs del sistema operativo.

2.4.2. Frameworks Multiplataforma

Además del desarrollo nativo, existen varios frameworks multiplataforma que permiten desarrollar aplicaciones para iOS junto con otras plataformas (principalmente Android) desde una base de código compartida.

- *React Native*, desarrollado por Meta, permite crear aplicaciones móviles utilizando JavaScript y React. El framework utiliza un puente nativo que comunica el código JavaScript con componentes nativos de cada plataforma, permitiendo acceso a APIs nativas mientras se comparte la mayor parte de la lógica de negocio.
- *Expo*, construido sobre React Native, proporciona un conjunto de herramientas y servicios que simplifican el desarrollo, incluyendo un runtime unificado, APIs listas para usar y un sistema de compilación en la nube. Expo reduce significativamente la complejidad de configuración del proyecto y facilita el despliegue, aunque con algunas limitaciones en el acceso a funcionalidades nativas avanzadas.
- *Flutter*, desarrollado por Google, utiliza el lenguaje *Dart* y un motor de renderizado propio que compila a código nativo. Flutter construye la interfaz de usuario desde cero en cada plataforma, evitando la necesidad de componentes nativos del sistema operativo y proporcionando mayor consistencia visual entre plataformas.

Otras alternativas multiplataforma incluyen *Xamarin* (ahora *.NET MAUI*) que utiliza C# y .NET, *Ionic* que combina tecnologías web (HTML, CSS, JavaScript) con capacidades nativas mediante *Cordova*, y *Unity* para aplicaciones que requieren capacidades gráficas avanzadas, esencialmente, videojuegos.

La elección entre desarrollo nativo y multiplataforma depende de factores como requisitos de rendimiento, necesidad de acceso a funcionalidades nativas avanzadas, tiempo de desarrollo, mantenimiento a largo plazo o recursos del equipo.

2.4.3. Proceso de Desarrollo y Distribución

El ciclo de vida de una aplicación iOS desde el desarrollo hasta su distribución sigue un proceso estructurado:

1. *Desarrollo*: Durante esta fase, el código se compila y ejecuta en simuladores iOS o dispositivos físicos mediante perfiles de desarrollo.
2. *Pruebas internas*: Para pruebas internas, las aplicaciones, una vez compiladas, es posible su distribución mediante *TestFlight*, plataforma que permite a desarrolladores invitar hasta 10.000 beta testers externos sin necesidad de certificados adicionales.
3. *Archive*: El proceso de *Archive* genera un paquete de distribución de la aplicación (.ipa) optimizado y firmado con los certificados de distribución³. Esta versión archivada puede subirse a *App Store Connect*, portal web de Apple para gestión de aplicaciones, donde se configura información de la aplicación, capturas de pantalla, descripciones y metadatos requeridos para la publicación.
4. *App Review*: Una vez hecha la petición de subida, este proceso de revisión de Apple verifica que el producto cumple con las directrices de la App Store, incluyendo seguridad, privacidad, calidad técnica y contenido. Una vez aprobada, la aplicación está disponible para distribución pública o privada según la configuración establecida.
5. *Distribución*: La distribución puede realizarse mediante tres canales principales:

³Los certificados de distribución son credenciales digitales emitidas por Apple que permiten identificar y autenticar al desarrollador, garantizando que la aplicación proviene de una fuente confiable y verificada. Son esenciales para firmar digitalmente las aplicaciones antes de su distribución en la App Store.

- App Store: Para usuarios finales a través de la tienda oficial de Apple.
- Distribución empresarial (*Enterprise*): Permite a las organizaciones internas distribuir aplicaciones privadas a sus empleados, fuera de la App Store.
- Distribución ad-hoc: Permite instalar la aplicación en un número limitado de dispositivos específicos, identificados mediante perfiles de aprovisionamiento ⁴.

2.5. Contexto tecnológico

2.5.1. Arquitectura de PANOT

Para materializar estos valores, el sistema se ha estructurado en dos componentes arquitectónicos fundamentales que operan de manera complementaria:

- PANOT (cliente): Constituye la capa de contacto entre el usuario y el sistema, proporcionando una interfaz intuitiva y adaptativa que facilita la captura de interacciones, la visualización de sus relaciones, y la presentación de recomendaciones contextuales entre otras características. El diseño de esta aplicación se centra en maximizar la conexión emocional con el usuario y en proporcionar una experiencia orgánica y sin fricción.
- PANOT OS (servidor): Representa la capa de inteligencia o el cerebro del sistema, encargada de procesar, analizar y aprender continuamente de las interacciones capturadas por la aplicación móvil. PANOT OS es el responsable de construir y mantener las representaciones relacionales estructuradas, de inferir patrones y preferencias del usuario, de generar recomendaciones contextuales y de evolucionar dinámicamente el conocimiento sobre las relaciones.

En este marco, PANOT se encarga de crear la conexión emocional y facilitar la interacción inmediata, mientras que PANOT OS proporciona la personalización adaptativa y la comprensión contextual que permiten que esa conexión se profundice con el tiempo. Ambos componentes trabajan de manera sinérgica para materializar los valores diferenciadores de conexión emocional y personalización adaptativa, garantizando que el sistema no solo responda a las necesidades actuales del usuario, sino que evolucione continuamente para acompañar la dinámica natural de sus relaciones.

⁴Los perfiles de aprovisionamiento son archivos que vinculan a un desarrollador y su aplicación con una cuenta de desarrollador, dispositivos y servicios autorizados

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 aquí la estrategia de ramificación y control de versiones]

3.1.3. Estructura de Repositorios en la Organización

[Escribir aquí la estructura de repositorios en la organización]

3.2. Especificación de Requisitos de Software

[Escribir aquí la especificación de requisitos de software]

3.3. Diseño de la Arquitectura del Sistema

[Escribir aquí 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

