Cetti, Paolo

Lucero Ruiz, Máximo

Quesada, Santiago

SaveApp: revolucionando el ahorro

Tesis para la obtención del título de grado de Ingeniería de Sistemas

Director: Carreño, Ignacio Luciano

UNIVERSIDAD CATÓLICA DE CÓRDOBA
JESUITAS

$aveApp

Informe de Proyecto Final

Cetti Paolo
2223989@ucc.edu.ar

Lucero Ruiz Máximo
2217924@ucc.edu.ar

Quesada Santiago
2217882@ucc.edu.ar

ÍNDICE

ABSTRACT

El presente informe documenta el desarrollo e implementación de SaveApp, una aplicación móvil inteligente orientada a mejorar el aprovechamiento de beneficios y promociones asociadas a tarjetas de crédito y débito emitidas por bancos en Argentina. El proyecto surge a partir de una problemática concreta: la fragmentación de la información sobre descuentos bancarios, su redacción compleja y la escasa visibilidad para el usuario final. Esta situación genera una pérdida sistemática de oportunidades de ahorro para millones de personas.

El objetivo principal es construir una solución automatizada, accesible y escalable que permita a los usuarios registrar sus tarjetas sin ingresar datos sensibles, visualizar descuentos disponibles, recibir notificaciones contextuales y contar con un asistente virtual basado en inteligencia artificial. Para lograrlo, se integraron diversas prácticas y tecnologías como web scraping, modelos de lenguaje natural (LLMs), geolocalización, notificaciones push, Firebase Authentication y una arquitectura basada en GraphQL y MongoDB.

La metodología combinó diseño centrado en el usuario, desarrollo iterativo y modularidad tecnológica. Se diseñaron interfaces intuitivas y se implementaron pruebas unitarias, de integración y de carga para validar la robustez del sistema. El resultado es una plataforma funcional que mejora la experiencia financiera cotidiana, empodera al usuario mediante información clara y accionable, y promueve tanto la inclusión digital como la transparencia.

SaveApp no solo soluciona un problema práctico, sino que plantea un nuevo estándar en el acceso a beneficios bancarios. Su arquitectura extensible y su enfoque centrado en el usuario permiten proyectarla a nivel regional, con potencial de integración con actores del ecosistema fintech y programas de fidelidad. Este trabajo representa una demostración concreta del poder de la tecnología aplicada al bienestar financiero de las personas.

This report documents the development and implementation of SaveApp, an intelligent mobile application designed to optimize the use of discounts and promotions linked to credit and debit cards issued by banks in Argentina. The project originates from a clear problem: the fragmentation of promotional information, its complex legal phrasing, and the low visibility of benefits for end-users. This context leads to a systematic loss of saving opportunities for millions of people.

The main objective is to build an automated, accessible, and scalable solution that allows users to register their cards without sensitive data, view available promotions, receive contextual notifications, and interact with a virtual assistant powered by artificial intelligence. To achieve this, technologies such as web scraping, large language models (LLMs), geolocation, push notifications, Firebase Authentication, and an architecture based on GraphQL and MongoDB were integrated.

The methodology combined user-centered design, iterative development, and modularity. Intuitive user interfaces were developed, and extensive unit, integration, and load testing was conducted to validate system robustness. The resulting platform improves the everyday

financial experience, empowers users through actionable and transparent information, and promotes both digital inclusion and financial awareness.

SaveApp not only solves a practical issue but also introduces a new standard in the accessibility of banking benefits. Its extensible architecture and user-first approach make it a candidate for regional expansion and integration with fintech partners and loyalty programs. This project demonstrates the real-world impact of applying technology to enhance personal financial well-being.

PRESENTACIÓN DEL TEMA

En la actualidad, el sistema financiero argentino ofrece una gran cantidad de promociones, beneficios y descuentos asociados al uso de tarjetas de crédito y débito emitidas por distintos bancos y entidades emisoras. Estos beneficios, si bien son atractivos y en muchos casos significativos, se encuentran fragmentados y distribuidos en múltiples sitios web, aplicaciones, correos electrónicos o incluso mensajes impresos, lo que dificulta su visualización en tiempo real por parte del consumidor. Esta dispersión de información genera una pérdida de oportunidades de ahorro y una desconexión entre el usuario y los beneficios que legítimamente le corresponden.

SaveApp nace como una respuesta innovadora a esta problemática: una aplicación móvil basada en inteligencia artificial que permite a los usuarios acceder de manera sencilla, centralizada y contextual a los descuentos disponibles con sus tarjetas, utilizando para ello tecnologías como Web Scraping, LLMs para el análisis semántico de los términos, geolocalización para identificar beneficios cercanos y un sistema de recomendaciones personalizadas.

A diferencia de otras soluciones presentes en el mercado, SaveApp no depende del ingreso manual de datos por parte de los usuarios o comunidades colaborativas, como ocurre con sitios como Descuentazo. Tampoco se restringe a sistemas cerrados de fidelización. En cambio, automatiza el relevamiento y análisis de información pública, integrando datos con una lógica inteligente que guía al usuario en su toma de decisiones cotidianas.

SaveApp no solo representa una herramienta de utilidad inmediata para la economía personal, sino también una propuesta tecnológica integral que pone en valor el uso combinado de técnicas modernas de software, inteligencia artificial y diseño de experiencia de usuario centrado en el consumidor.

GLOSARIO

- **Web Scraping (o Web Crawling):** Técnica de programación que permite la extracción automatizada de contenido estructurado o no estructurado desde páginas web.
- **Análisis semántico:** Proceso de interpretación automática del lenguaje natural utilizado en las promociones bancarias, con el fin de extraer condiciones clave como topes, días válidos, tarjetas compatibles y medios de pago. En SaveApp se realiza mediante modelos de lenguaje (LLMs).
- **LLM (Large Language Model):** Modelos de lenguaje de gran escala entrenados sobre grandes cantidades de texto para comprender y generar lenguaje humano.
- **PWA (Progressive Web App):** Aplicación web progresiva, diseñada para comportarse como una app nativa en dispositivos móviles.
- **Open Banking:** Marco regulatorio y técnico que permite a los usuarios compartir de manera segura sus datos financieros con terceros a través de interfaces abiertas (APIs), no disponible en Argentina.
- **ETL (Extract, Transform, Load):** Proceso de extracción, transformación y carga de datos desde diversas fuentes a sistemas de almacenamiento estructurado.
- **Crowdsourced:** Contenido generado por los propios usuarios de una plataforma, en lugar de ser cargado por una entidad central. Su calidad depende de la colaboración comunitaria.
- **Geofencing:** Técnica de geolocalización que permite definir zonas virtuales alrededor de puntos geográficos reales. Al ingresar o salir de esas zonas, la app puede activar notificaciones automáticas.
- **Asistente virtual:** Interfaz conversacional que permite a los usuarios interactuar en lenguaje natural para consultar promociones disponibles, recibir sugerencias personalizadas y hacer seguimiento de reintegros. Se alimenta de datos contextuales del usuario y del entorno.
- **Beneficio bancario:** Cualquier tipo de promoción ofrecida por una entidad financiera, como descuentos, reintegros o cuotas sin interés, aplicables al consumo con tarjetas emitidas por dicha entidad.
- **Comercio adherido:** Establecimiento que participa en una promoción específica y acepta beneficios bancarios. En SaveApp se identifican automáticamente a partir del análisis de términos y condiciones.
- **Condiciones de aplicación:** Conjunto de requisitos que deben cumplirse para acceder a un beneficio, como tipo de tarjeta, banco emisor, monto mínimo de compra, días válidos o canal de pago.

- **Normalización de datos:** Proceso técnico mediante el cual se transforman datos provenientes de distintas fuentes en un formato homogéneo.
- **Developer experience (DX o DevEx):** Percepción y satisfacción de los desarrolladores al usar herramientas, plataformas y procesos, buscando reducir fricciones y facilitar un desarrollo ágil y eficiente.
- **Low Code / Plataforma Low Code**: Entorno de desarrollo que permite construir aplicaciones con mínima escritura de código mediante componentes visuales, plantillas y lógica predefinida.
- **Privacy-by-design:** Enfoque que incorpora la privacidad y la protección de datos desde el inicio del diseño del sistema, aplicando principios como minimización de datos, finalidad específica, seguridad por defecto, retención limitada, transparencia, consentimiento y control del usuario, y seudonimización/anonimización cuando corresponde.
- **Agentes de IA:** Componentes (software) que perciben un contexto, toman decisiones y ejecutan acciones autónomas o semiautónomas para lograr un objetivo.
- **Schemas (GraphQL y tipado):** Estructuras tipadas que definen de forma explícita las entidades, sus campos y relaciones, además de las operaciones disponibles (queries y mutations).
- **Lock-in (Cloud Providers):** Situación en la que una arquitectura queda atada a servicios específicos de un proveedor cloud, dificultando migraciones o incrementando costos y riesgos.
- **Infrastructure as Code (IaC):** Práctica de describir y gestionar la infraestructura mediante archivos de configuración versionables, permitiendo reproducibilidad, auditoría y despliegues consistentes.
- **Points of Interest (POIs):** Identificadores geoespaciales estandarizados que representan un mismo comercio o lugar físico, aun cuando existan ligeras variaciones en nombre, dirección o coordenadas reportadas. Se utilizan para consolidar registros (deduplicación), evitar que pequeñas diferencias de lat/long generen múltiples puntos, y mantener un historial único por ubicación.

DIAGNÓSTICO DEL PROBLEMA

Contexto Actual

A medida que crece la oferta de servicios financieros y se diversifican los medios de pago, también lo hacen los programas de beneficios que ofrecen los bancos y entidades emisoras de tarjetas. Esta explosión de promociones tiene el potencial de mejorar la economía diaria de millones de personas, pero se encuentra con una barrera crítica: la visibilidad de la información.

En un país como Argentina, donde no existe un sistema de Open Banking que estandarice el acceso a los datos financieros, los usuarios están obligados a consultar múltiples fuentes de forma manual para conocer las promociones vigentes. Esto incluye visitar los sitios de cada banco y navegar por sus buscadores de promociones, cada uno con diferentes criterios de filtrado, formatos y condiciones de uso.

Problemas Identificados

- **Desconocimiento de promociones:** Diversas encuestas indican que muchos tarjetahabientes no recuerdan ni conocen los descuentos y recompensas disponibles al comprar. Por ejemplo, un sondeo de CreditCards.com halló que 23 % de los titulares no redimió sus recompensas durante el último año y 11 % no sabe cómo canjearlas [1]. Otro compendio de estadísticas muestra que 69 % de los usuarios con tarjetas de recompensas acumula puntos o millas sin usar y 14 % reconoce que le cuesta entender cómo aprovecharlos [2]. Además, un estudio de satisfacción de J.D. Power encontró que sólo 67 % de los encuestados comprende cómo acumular recompensas, 55 % sabe que sus puntos no expiran y apenas 30 % considera que siempre maximiza los beneficios de su tarjeta [3]. Asimismo, un reportaje citó que más de un tercio de los miembros de programas de lealtad no sabe cómo redimir las ofertas y unos cuatro de cada diez dejan expirar sus millas [4].

- **Falta de centralización:** La información sobre descuentos y promociones está dispersa en múltiples páginas web y contratos. Muchos emisores no promocionan sus beneficios en un único sitio accesible, lo que obliga a los usuarios a leer términos y condiciones extensos o a revisar varios canales [7]. En algunos casos, para canjear millas o descuentos hay que ingresar al portal específico del banco o llamar a líneas telefónicas, lo que evidencia la ausencia de un sistema centralizado y sencillo [8].

- **Falta de claridad:** Para obtener ciertos descuentos es necesario cumplir requisitos que no siempre quedan claros. Según CreditCards.com, 9 % de los encuestados encuentra demasiado confusos los programas de recompensas y otro 9 % no los usa por falta de tiempo [1]. CardRates detalla que entre quienes no canjearon sus puntos, 23 % los consideró de poco valor, 11 % no sabía cómo redimirlos y 9 % los calificó de

confusos [2]. El estudio de J.D. Power corrobora esta falta de comprensión general [3], mientras que un análisis periodístico indica que más de un tercio de los participantes en programas de fidelidad no sabe cómo canjear las ofertas y que unos cuatro de cada diez estadounidenses permiten que sus millas caduquen por confusión [4].

- **Interfaz inadecuada:** Las aplicaciones bancarias y plataformas de recompensas no priorizan la visualización clara de beneficios ni notifican en contexto. Los datos de J.D. Power muestran que sólo 30 % de los usuarios siente que siempre maximiza sus recompensas, lo que sugiere una experiencia de usuario deficiente [3]. CardRates apunta que 9 % de los encuestados considera confusos los programas, lo cual también puede estar relacionado con interfaces poco intuitivas [2], y CreditCards.com reporta un porcentaje similar de clientes que perciben complejidad en los programas [1].

- **Inexistencia de automatización:** No existen mecanismos públicos que integren y actualicen todas las promociones disponibles para el usuario final. En algunos países, los clientes deben consultar manualmente los portales de cada banco o llamar a las líneas de atención para conocer o canjear sus beneficios [8]. La falta de herramientas automatizadas y de educación financiera adecuada sobre los seguros y servicios que ya están incluidos en las tarjetas también contribuye a que estos pasen inadvertidos [6].

Oportunidades

Frente a esta problemática, surge una oportunidad única de ofrecer una solución tecnológica que simplifique y automatice todo el proceso desde la recolección de beneficios hasta su análisis personalizado. SaveApp se presenta como una herramienta innovadora que acerca la inteligencia al consumo cotidiano, donde una buena elección de pago puede traducirse en un ahorro real y significativo.

A su vez, representa una oportunidad para los comercios, que podrán atraer nuevos clientes y aumentar sus ventas al hacer visibles sus promociones, y para los bancos, que podrán potenciar el uso de sus tarjetas, mejorar la fidelización de sus usuarios y comprender mejor sus hábitos de consumo.

OBJETIVOS

Objetivo General

Desarrollar una aplicación móvil basada en inteligencia artificial y web scraping que permita a los usuarios optimizar el uso de sus tarjetas bancarias, accediendo a beneficios y descuentos de forma contextual y personalizada, integrando funciones de asesoramiento financiero, seguimiento de reintegros y visualización inteligente de promociones.

Objetivos Específicos

- Diseñar e implementar un sistema de registro y autenticación de usuarios que permita la carga de tarjetas sin requerir información sensible, garantizando la protección de los datos personales y cumpliendo con las buenas prácticas de seguridad informática.
- Desarrollar y desplegar crawlers web automáticos capaces de recopilar, actualizar y almacenar descuentos y beneficios desde los portales oficiales de bancos y billeteras virtuales, asegurando la integridad y consistencia de la información obtenida.
- Aplicar modelos de lenguaje (LLM) para el análisis automatizado de los términos y condiciones de las promociones, con el fin de identificar y clasificar las variables más relevantes, tales como tipo de beneficio, vigencia, medio de pago y restricciones.
- Integrar un sistema de geolocalización que permita detectar comercios cercanos con beneficios activos, optimizando la precisión en la ubicación y la pertinencia de las sugerencias al usuario.
- Implementar un sistema de notificaciones personalizadas en tiempo real, que informe a los usuarios sobre descuentos aplicables según su ubicación, historial de uso y preferencias, priorizando la oportunidad y relevancia de la información transmitida.
- Diseñar y desarrollar un módulo de seguimiento de reintegros, que posibilite el registro de compras con beneficios y la notificación automática al usuario cuando se cumpla el plazo de acreditación correspondiente.
- Desarrollar un asistente virtual basado en inteligencia artificial, capaz de interpretar consultas en lenguaje natural, visualizar las ofertas disponibles y ofrecer recomendaciones personalizadas sobre el uso óptimo de las tarjetas y beneficios asociados.
- Diseñar e implementar una arquitectura backend escalable y segura, que soporte la gestión eficiente de usuarios, beneficios y comercios, aplicando estándares actuales en materia de desarrollo seguro y despliegue en entornos cloud.
- Construir una base de datos optimizada para consultas geoespaciales y relaciones dinámicas, que permita un acceso ágil y eficiente a la información de descuentos, ubicaciones y reintegros registrados.
- Diseñar y desarrollar una interfaz de usuario intuitiva y responsiva para plataformas Android e iOS, validada mediante pruebas de usabilidad con un grupo piloto de usuarios, orientada a garantizar una experiencia fluida y satisfactoria.

MARCO TEÓRICO

Contexto General del Problema

El sistema financiero argentino, al igual que en muchos otros países, utiliza desde hace décadas herramientas de incentivo al consumo asociadas al uso de tarjetas de crédito y débito. Estas estrategias comerciales, impulsadas tanto por los bancos como por los comercios, se han vuelto un pilar en la dinámica del consumo minorista, no sólo como estímulo de ventas sino como mecanismo de fidelización del cliente.

El universo de beneficios disponibles es amplio y se diversifica constantemente: se ofrecen reintegros, descuentos, cuotas sin interés y promociones especiales que varían según el rubro, el banco emisor, el día de la semana, el canal de compra (online o presencial), y el medio de pago. Rubros como supermercados, gastronomía, vestimenta, electrónica, turismo, combustibles, videojuegos y farmacias concentran gran parte de estas campañas promocionales, aunque también se extienden a servicios, hogar, deportes, hoteles y más.

Estos beneficios, si bien están pensados como un valor agregado para el consumidor, presentan una complejidad intrínseca: su carácter volátil y condicionado. Las promociones tienen una validez temporal corta, se actualizan con frecuencia, y están sujetas a múltiples condiciones de aplicación. Este panorama ha generado un terreno fértil para el desarrollo de herramientas que permitan interpretar, clasificar y contextualizar estos beneficios con precisión.

Clasificación y características de las ofertas

Para entender el rubro, es necesario identificar los principales tipos de beneficios que ofrecen los bancos y entidades financieras:

- **Reintegro:** Devolución de una parte del monto total de la compra, acreditada en la cuenta del cliente luego de un plazo determinado. Ejemplo: “10% de reintegro en farmacias, tope $1.500, acreditación dentro de los 30 días hábiles.”
- **Descuento:** Reducción directa en el precio al momento de realizar la compra. Ejemplo: “20% de descuento pagando con Visa Banco Galicia los miércoles.”
- **Cuotas:** Posibilidad de pagar en varias cuotas sin interés (o con tasas promocionales), generalmente para compras superiores a cierto monto. Ejemplo: “6 cuotas sin interés en electrodomésticos pagando con Mastercard BBVA.”
- **Promociones:** Ofertas de tipo “x por y”, como 2x1 o 3x2, muchas veces limitadas a productos o marcas específicas. Ejemplo: “2x1 en entradas de cine con tarjeta Naranja los jueves.”

Estas ofertas pueden superponerse entre sí, tener múltiples restricciones y variar en su aplicabilidad según el medio de pago (débito, crédito, QR, etc.). Además, suelen tener vigencia limitada, lo que obliga al usuario a mantenerse constantemente actualizado.

Otra característica relevante es que los términos y condiciones de muchas ofertas siguen patrones similares: se especifica el tipo de tarjeta, banco emisor, días válidos, monto mínimo, tope máximo, locales adheridos, medios de pago aceptados, y plazos para el reintegro (si aplica). Sin embargo, esta información se encuentra redactada con lenguaje jurídico complejo, dificultando su comprensión.

Análisis de Campo

El análisis de campo se orientó a comprender la dinámica real de las promociones bancarias y de comercios para diseñar un sistema capaz de descubrir, normalizar y mantener vigente la información crítica para el usuario. Para ello se combinaron técnicas de desk research sobre fuentes públicas (portales de bancos, billeteras y retailers), exploraciones controladas de scraping para evaluar estructura y ritmo de cambio de las páginas, una revisión sistemática de Términos y Condiciones (TyC) para identificar patrones operativos y excepciones, y un mapeo exhaustivo de las fuentes donde cada entidad efectivamente publica y actualiza sus beneficios.

Esta investigación devolvió una serie de hallazgos clave:

1. **Alta volatilidad:** Las ofertas son marcadamente temporales y en la mayoría de los casos su vigencia efectiva ronda 1 mes. Esto exige políticas de refresco frecuentes (al menos mensuales).

2. **Cobertura por múltiples marcas y categorías:** En la mayoría de los casos, las promociones se vinculan a una única marca o comercio. Sin embargo, también se presentan ofertas que abarcan múltiples marcas y, en otros casos, beneficios aplicables a categorías generales (por ejemplo, “restaurantes” o “farmacias”). Estos escenarios anticipan la necesidad de definir los esquemas y las reglas para representar correctamente los rubros y las marcas.

3. **Información esencial oculta en TyC:** Los requisitos que habilitan o invalidan el beneficio (medios de pago, topes, exclusiones, días, canales, plazos de reintegro) suelen estar dispersos y redactados en lenguaje natural. Al tratarse de datos no estructurados, será necesario implementar técnicas de procesamiento de lenguaje natural para extraer e identificar los datos necesarios para el negocio.

4. **Localización de fuentes y datos desestructurados:** Identificar dónde publica cada banco implicó rastrear buscadores de promociones. La estructura es, en general, heterogénea lo que requirió estrategias de scraping adaptativas y un pipeline de normalización por fuente.

5. **Barreras geográficas y segmentación:** Las ofertas presentan alcance geográfico, principalmente por país y en algunos casos por provincia. Esto implica que el alcance máximo es a nivel país, debiendo dividir las ofertas por país en caso de expandirse fuera de Argentina, evitando así posibles errores por diferencias de moneda, normativa o disponibilidad.

6. **Catálogo propio de tarjetas y comercios adheridos:** Los TyC muchas veces enumeran solo exclusiones (“no aplica para…”) o declaran “todas las tarjetas del banco”, sin detallar producto por producto. Asimismo, cuando la oferta dice “para todos los locales de la marca”, no lista cada sucursal. Conclusión: es imprescindible mantener un catálogo propio de tarjetas (banco, producto, marca de servicio) y un catálogo de comercios adheridos.

7. **Patrones repetidos de TyC por banco:** Un mismo banco tiende a reutilizar plantillas (estructura y frases) entre promociones. Este comportamiento abre la posibilidad de incorporar una lógica de caché a nivel de plantilla o banco, lo que permitiría disminuir invocaciones innecesarias y, en consecuencia, optimizar costos y tiempos de procesamiento.

8. **Necesidad de deduplicación y trazabilidad:** Debido a re-publicaciones y cambios menores (título/fecha), el pipeline debe usar un identificador estable por fuente y campos de huella (hash) para hacer upserts idempotentes y evitar insertar duplicados a la base de datos.

Como cierre del análisis de campo, no se realizó una investigación de mercado formal en esta etapa, la propuesta nace de una necesidad observada por el equipo y fue validada de manera informal con conocidos, además de una encuesta exploratoria en el ámbito universitario que mostró alta aceptación. Las observaciones de uso indican que muchas personas dependen de la cartelería en los locales o de la información del personal para enterarse de beneficios, mientras que otras consultan manualmente billeteras virtuales o apps bancarias, un proceso percibido como tedioso. Estos hallazgos confirman la oportunidad de una solución que centralice información, haga explícitas las condiciones clave y reduzca fricción en el acceso a las promociones.

Soluciones similares en el mercado

Existen en el mercado argentino y global diversas plataformas que, desde distintos enfoques, buscan facilitar el acceso a descuentos, beneficios y promociones asociadas al consumo. Sin embargo, la mayoría de ellas presenta limitaciones importantes al momento de brindar una solución integral, contextualizada y personalizada como la que propone este proyecto.

A continuación se describen algunas de las alternativas existentes y sus principales características:

- **Descuentazo:** Es una plataforma argentina que reúne promociones bancarias en diversos rubros. Su contenido es de tipo crowdsourced, es decir, cargado por la comunidad de usuarios, lo cual implica una cobertura variable y sujeta a validación manual. Si bien permite filtrar por categoría o banco, no ofrece personalización automática según las tarjetas del usuario ni recomendaciones contextuales por ubicación. El diseño se basa en una exploración activa del usuario, sin inteligencia aplicada para la toma de decisiones.

- **MaxRewards:** Esta aplicación está orientada al mercado norteamericano y se enfoca en maximizar la obtención de puntos y recompensas de tarjetas de crédito. Se integra con los sistemas de fidelización (rewards programs) de emisores como American Express o Chase. Aunque incluye recomendaciones sobre qué tarjeta conviene usar en cada transacción, su foco no está en los descuentos bancarios ni en la lectura semántica de promociones específicas, sino en el rendimiento de acumulación de puntos. No es aplicable al mercado argentino, donde este tipo de programas tiene baja penetración.

- **MODO:** Aplicación argentina que actúa como billetera virtual y medio de pago unificado. Incluye una página de promociones propias en conjunto con los bancos adheridos, pero está limitada a aquellas entidades que tienen convenio con la app. No permite visualizar beneficios externos a su ecosistema, ni integrar todas las tarjetas que un usuario pueda tener. Además, las promociones se muestran de manera general, sin análisis personalizado de términos, compatibilidades o seguimiento de reintegros.

- **Ratoneando:** Plataforma centrada exclusivamente en comparar precios de productos en supermercados. No considera promociones bancarias ni beneficios personalizados según tarjetas. Su foco es el ahorro directo por precio de lista, sin analizar condiciones financieras asociadas al método de pago.

- **Tuki y Mosca:** Aplicaciones que trabajan con cupones de descuento, generalmente vinculados a marcas o categorías específicas. Están centradas en promociones propias o acuerdos con comercios puntuales, no en beneficios emitidos por bancos. Tampoco permiten integrar datos de tarjetas ni hacer recomendaciones personalizadas, y su modelo está basado en la generación de códigos o vouchers que deben presentarse al momento de la compra.

- **Clash:** Es una aplicación argentina relativamente nueva, que no se encontró al momento de la investigación inicial de este proyecto. A diferencia de otras soluciones más estructuradas, Clash se posiciona con un enfoque más cercano al de una red social de descuentos: son los propios comercios quienes suben sus promociones de forma directa a la plataforma.

En síntesis, estas alternativas cubren distintos nichos (desde medios de pago, cupones, precios, o programas de puntos) pero ninguna ofrece una solución transversal, automatizada y centrada en el usuario como lo plantea SaveApp.
La mayoría carece de:

- Integración entre múltiples emisores de tarjetas
- Lectura automatizada de términos y condiciones
- Geolocalización con notificaciones proactivas
- Visualización del impacto económico (ahorros y reintegros)
- Un asistente personalizado basado en IA.

Tecnologías disponibles

Este apartado presenta una exploración y análisis exhaustivo del ecosistema de herramientas tecnológicas candidatas para el desarrollo de SaveApp. La selección de cada componente no es una decisión trivial; es un acto estratégico que definirá la viabilidad del producto, su resiliencia ante cambios externos, su capacidad para escalar, la calidad de la experiencia del usuario final y, en última instancia, los costos operativos. La propia naturaleza volátil de las ofertas, que cambian con frecuencia y requieren adaptación inmediata, fue un factor determinante en la elección de tecnologías y en el diseño de la arquitectura del sistema. Cada tecnología se evalúa desde una perspectiva crítica y contextual, sopesando su aplicabilidad en el dinámico y particular escenario argentino, la profundidad de su curva de aprendizaje, la solidez de su comunidad de soporte y su mantenibilidad a largo plazo.

1. Mobile Backend

Comunicación entre Cliente y Servidor (APIs)

La comunicación entre la aplicación móvil (el cliente) y el servidor (el backend) es el torrente sanguíneo del sistema. La eficiencia, flexibilidad y robustez de esta comunicación, definida por la arquitectura de su Interfaz de Programación de Aplicaciones (API), impacta directamente en la velocidad de carga de la aplicación, el consumo de datos móviles del usuario y la capacidad de los desarrolladores para crear nuevas funcionalidades rápidamente.

REST (Representational State Transfer)

- **¿Qué es?** Es un estilo arquitectónico para construir servicios web, basado en la idea de tratar la información como "recursos" que se manipulan utilizando los verbos estándar del protocolo HTTP (GET, POST, PUT, DELETE).

- **Ventajas y Desventajas** Es un enfoque intuitivo, ampliamente conocido y fácil de implementar y depurar. Su principal desventaja son los problemas de "under-fetching" (necesitar hacer múltiples llamadas para obtener todos los datos) y "over-fetching" (obtener más datos de los necesarios), lo que genera latencia y consume el plan de datos del usuario de forma ineficiente.

- **¿Para qué la usaríamos?** Sería una opción viable y tradicional para construir nuestra API. Sin embargo, tendríamos que diseñar los endpoints cuidadosamente para mitigar sus ineficiencias, posiblemente creando endpoints específicos para cada vista de la aplicación.

GraphQL

- **¿Qué es?** Es un lenguaje de consulta para APIs que permite al cliente solicitar exactamente los datos que necesita, y nada más, en una sola petición. El poder de decidir la forma de la respuesta se traslada del servidor al cliente.

- **Ventajas y Desventajas** Su ventaja fundamental es que soluciona de raíz los problemas de under-fetching y over-fetching, resultando en una comunicación mucho más eficiente, tiempos de carga más rápidos y menor consumo de datos y batería para el usuario. La contrapartida es una mayor complejidad inicial en el backend.

- **¿Para qué la usaríamos?** Sería la arquitectura de API preferida para SaveApp. Nos permitiría que la aplicación móvil construyera consultas complejas para obtener todos los datos necesarios para una pantalla en una única llamada de red, mejorando drásticamente la experiencia del usuario final.

Autenticación y Gestión de Sesiones

La autenticación es el proceso de verificar la identidad de un usuario, mientras que la gestión de sesiones es el mecanismo para mantener esa identidad verificada a través de múltiples interacciones. Para SaveApp, este sistema debe ser una fortaleza segura que proteja los datos del usuario, pero a la vez una puerta de entrada casi invisible que minimice la fricción en el acceso.

JWT (JSON Web Tokens)

- **¿Qué es?** Es un estándar abierto para crear tokens de acceso que permiten una autenticación "stateless" (sin estado). El token es una cadena de texto firmada digitalmente que contiene información del usuario y una fecha de expiración.

- **Ventajas y Desventajas** La ventaja es que el servidor no necesita mantener un registro de las sesiones activas, lo que facilita la escalabilidad. El principal desafío es la revocación: un token robado es válido hasta que expira, lo que se mitiga usando tokens de corta duración junto con "refresh tokens".

- **¿Para qué la usaríamos?** Sería el mecanismo central para gestionar las sesiones de los usuarios. Tras el login, el cliente recibiría un JWT que incluiría en cada petición a la API para verificar su identidad de forma segura y eficiente.

Firebase Authentication

- **¿Qué es?** Es el servicio de autenticación de la plataforma BaaS (Backend as a Service) de Google. Ofrece un sistema de identidad completo y robusto que gestiona todo el ciclo de vida del usuario, desde el registro y el inicio de sesión hasta la recuperación de contraseñas y el almacenamiento seguro de credenciales.

- **Ventajas y Desventajas** La principal ventaja es su profunda integración con todo el ecosistema de Firebase (Firestore, Cloud Functions, Hosting), lo que simplifica el desarrollo. Su nivel gratuito es muy generoso, ideal para un MVP. La desventaja es que puede generar una fuerte dependencia del ecosistema de Google (vendor lock-in) y puede ser menos flexible que soluciones especializadas para flujos de autenticación empresariales muy complejos.

- **¿Para qué la usaríamos?** Sería la opción principal y más estratégica para el MVP de SaveApp. La usaríamos para manejar de forma rápida y segura todo el flujo de registro y login de usuarios, incluyendo el inicio de sesión con proveedores sociales como Google y Apple, acelerando el desarrollo y garantizando la seguridad desde el primer día.

Supabase Auth

- **¿Qué es?** Es el módulo de autenticación de Supabase, una plataforma que se posiciona como la alternativa de código abierto a Firebase. Está construido sobre PostgreSQL y se integra directamente con su sistema de seguridad a nivel de fila (Row Level Security), permitiendo un control de acceso a los datos muy granular.

- **Ventajas y Desventajas** Su gran ventaja es ser de código abierto, lo que elimina el "vendor lock-in" y permite la opción de auto-hospedar la solución para un control total. Su integración nativa con PostgreSQL es un plus si ya se usa esa base de datos. Como plataforma más joven que Firebase, algunas partes de su ecosistema podrían ser menos maduras.

- **¿Para qué la usaríamos?** La elegiríamos si la estrategia del proyecto fuera priorizar el uso de tecnología de código abierto para evitar la dependencia de un solo proveedor. Sería especialmente potente para SaveApp al combinarla con PostgreSQL como base de datos principal, unificando la lógica de autenticación y los permisos de acceso a los datos.

Auth0

- **¿Qué es?** Es una plataforma de identidad como servicio (IDaaS) de nivel empresarial, ahora propiedad de Okta. Es una solución altamente especializada y conocida por su extrema flexibilidad para manejar prácticamente cualquier escenario de autenticación y autorización.

- **Ventajas y Desventajas** Su ventaja es una flexibilidad inigualable, ideal para requisitos complejos como el inicio de sesión único (SSO) empresarial, autenticación para múltiples tipos de clientes (B2B, B2C) y la personalización de flujos con reglas y "Actions". La principal desventaja es su costo, que puede ser significativamente más alto que otras opciones, y su complejidad puede ser excesiva para un simple MVP.

- **¿Para qué la usaríamos?** Consideraríamos usar Auth0 si SaveApp tuviera desde el inicio requisitos de autenticación muy complejos o si el modelo de negocio incluyera una vertiente empresarial (B2B). En general, es una solución para una etapa de madurez del producto más que para el lanzamiento inicial.

Better Auth

- **¿Qué es?** Better Auth es un framework de autenticación de código abierto, enfocado en el ecosistema de TypeScript. A diferencia de los servicios de terceros, su filosofía es proporcionar las herramientas para que los desarrolladores "construyan su propia

autenticación" de forma fácil y segura directamente en su backend y base de datos, sin depender de un servicio externo.

- **Ventajas y Desventajas** La ventaja es el control total y la ausencia de dependencia de terceros, combinando la seguridad de una solución robusta con la flexibilidad de tener el código en tu propio sistema. Al ser una herramienta más nueva y un framework en lugar de un servicio, requiere que el equipo de desarrollo asuma la responsabilidad de la implementación y el hosting.

- **¿Para qué la usaríamos?** La elegiríamos si quisiéramos el máximo control sobre nuestro sistema de autenticación sin tener que construir toda la lógica de seguridad desde cero. Sería una opción para un equipo con experiencia técnica que valora la soberanía sobre los datos y la arquitectura, pero no quiere reinventar las complejidades de la gestión de sesiones y tokens.

Otros

Mongoose

- **¿Qué es?** Es una biblioteca de modelado de datos para MongoDB y Node.js, que proporciona una capa de abstracción sobre la base de datos. Permite definir esquemas con validaciones, middlewares y métodos personalizados para manejar los documentos de MongoDB de manera más controlada y estructurada.

- **Ventajas y Desventajas:** La ventaja principal es que impone una estructura clara sobre una base de datos NoSQL, simplificando validaciones y relaciones, y ofreciendo potentes herramientas de consulta. Como desventaja, puede limitar ciertas operaciones avanzadas de MongoDB y añadir una ligera sobrecarga de rendimiento frente a usar el driver nativo.

- **¿Para qué la usaríamos?** Para definir y gestionar los modelos de datos de SaveApp, como usuarios, tarjetas, ofertas y transacciones, asegurando consistencia en los datos y centralizando la lógica de negocio relacionada con la persistencia.

Typegoose

- **¿Qué es?** Es una librería que une Mongoose y TypeScript, permitiendo definir modelos de Mongoose utilizando clases y decoradores de TypeScript. De esta forma, se aprovecha el poder del tipado estático para crear modelos de datos de manera más intuitiva y segura, eliminando la necesidad de definir interfaces separadas para los esquemas.

- **Ventajas y Desventajas:** La principal ventaja es que reduce la redundancia de código y mantiene una única fuente de verdad para la estructura de los datos (la clase de TypeScript). Esto mejora la legibilidad y el mantenimiento, además de ofrecer un autocompletado más preciso en el IDE. Su principal desventaja es que introduce una capa de abstracción adicional sobre Mongoose, lo que puede generar una pequeña curva de aprendizaje y dependencia de otra librería.

- **¿Para qué la usaríamos?** Para definir los modelos de datos de SaveApp (usuarios, tarjetas, etc.) directamente con clases de TypeScript. Esto nos permitirá tener un código más limpio, seguro y fácil de entender, aprovechando al máximo las ventajas del tipado estático en la interacción con la base de datos MongoDB.

TypeGraphQL

- **¿Qué es?** Es un framework para construir APIs de GraphQL en Node.js utilizando TypeScript y decoradores. Su enfoque se centra en el "code-first", donde el esquema de GraphQL se genera automáticamente a partir de las clases y decoradores de TypeScript, en lugar de tener que escribirlo manualmente en el lenguaje de definición de esquemas (SDL).

- **Ventajas y Desventajas:** Su gran ventaja es que elimina la duplicación de tipos entre el código de TypeScript y el esquema de GraphQL, simplificando enormemente el desarrollo y asegurando que ambos estén siempre sincronizados. También se integra de manera nativa con otras librerías del ecosistema de TypeScript. Como desventaja, al generar el esquema automáticamente, puede ofrecer menos control granular que el enfoque "schema-first" y su ecosistema es más pequeño que el de soluciones como Apollo Server (aunque se integra con él).

- **¿Para qué la usaríamos?** Para construir la API de GraphQL de SaveApp. Nos permitiría definir los *resolvers*, *queries* y *mutations* usando clases de TypeScript, lo que agilizaría el desarrollo, reduciría errores y garantizaría que la API esté fuertemente tipada y bien documentada desde el propio código.

2. Mobile App

Arquitecturas Móviles

En este ámbito podemos distinguir tres enfoques principales para el desarrollo de aplicaciones: multiplataforma, nativo y basado en web (PWA).
Cada uno presenta ventajas y desventajas que impactan directamente en los costos, tiempos de desarrollo, capacidades técnicas y calidad final del producto.

Arquitecturas Basadas en Web

Progressive Web Apps (PWA)

- **¿Qué es?** Una PWA es una aplicación web que combina lo mejor de las páginas web y las aplicaciones móviles. Funciona directamente desde el navegador, pero puede instalarse en el dispositivo y trabajar sin conexión. Se desarrolla con tecnologías web estándar (HTML, CSS, JavaScript) y es independiente de las tiendas de aplicaciones.

- **Ventajas y Desventajas** Su mayor ventaja es que permite llegar a cualquier dispositivo con un único desarrollo web, eliminando la necesidad de publicar y mantener versiones separadas para iOS y Android. Reduce costos, acelera

el tiempo de salida al mercado y facilita las actualizaciones sin pasar por procesos de aprobación de tiendas. Sin embargo, sus capacidades están limitadas frente a aplicaciones nativas o multiplataforma, especialmente en iOS, donde algunas APIs y funciones (como push o acceso a hardware avanzado) tienen soporte reducido.

- **¿Para qué la usaríamos?** Una PWA sería una opción ideal para el lanzamiento rápido de SaveApp en etapas iniciales, permitiendo validar el producto con una base de usuarios amplia sin los costos del desarrollo nativo. Facilitaría iteraciones rápidas y despliegues inmediatos, manteniendo una experiencia consistente en móviles y escritorio.

Arquitecturas Móviles Multiplataforma

Estas tecnologías permiten lanzar un Producto Mínimo Viable (MVP) de manera eficiente, compartiendo la mayor parte del código para llegar al mercado de iOS y Android simultáneamente.

Flutter

- **¿Qué es?** Flutter es un kit de herramientas de UI de código abierto creado por Google. Permite construir aplicaciones para móvil, web y escritorio desde una única base de código en lenguaje Dart. Su característica principal es que utiliza su propio motor de renderizado (Skia) para dibujar cada píxel en la pantalla.

- **Ventajas y Desventajas** Su mayor ventaja es la consistencia visual absoluta entre plataformas y un ciclo de desarrollo extremadamente rápido gracias a su función de "Stateful Hot Reload". Su rendimiento es prácticamente nativo. La desventaja es que requiere aprender el lenguaje Dart y su ecosistema, que es menos extendido que el de JavaScript.

- **¿Para qué la usaríamos?** Flutter es un candidato ideal para desarrollar la app. Nos permitiría construir una aplicación con una interfaz de usuario atractiva y consistente para iOS y Android de forma rápida, reduciendo costos y tiempo de llegada al mercado.

React Native

- **¿Qué es?** React Native es un framework desarrollado por Meta (Facebook) que permite crear aplicaciones para iOS y Android utilizando JavaScript y la librería React. A diferencia de Flutter, actúa como un intermediario que utiliza los componentes de UI nativos de cada plataforma.

- **Ventajas y Desventajas** Su gran ventaja es la posibilidad de reutilizar el vasto ecosistema de librerías de JavaScript y React, muy abundantes en el mercado. La principal desventaja es que la comunicación entre JavaScript y

el código nativo (el puente) puede convertirse en un cuello de botella para el rendimiento en tareas intensivas.

- **¿Para qué la usaríamos?** Al igual que Flutter, es una opción viable y pragmática para el desarrollo de SaveApp, especialmente si el equipo ya tiene experiencia en el ecosistema de React y JavaScript.

Kotlin Multiplatform (KMP)

- **¿Qué es?** KMP es un enfoque híbrido que no busca compartir la interfaz de usuario, sino toda la lógica de negocio subyacente. El núcleo de la aplicación (llamadas a la API, acceso a base de datos, modelos) se escribe una vez en Kotlin y se compila como una librería para Android y iOS, mientras que la UI se construye de forma nativa en cada plataforma.

- **Ventajas y Desventajas:** Ofrece lo mejor de ambos mundos: la eficiencia de no duplicar la lógica de negocio y la calidad y rendimiento de una interfaz 100% nativa. Su principal desventaja es la complejidad, ya que requiere expertise en tres ecosistemas (Kotlin compartido, nativo iOS, nativo Android), lo que lo hace menos ideal para un lanzamiento rápido.

- **¿Para qué la usaríamos?** KMP no se perfila como una elección inicial para SaveApp debido a su complejidad. Sin embargo, representa una excelente opción de evolución a largo plazo, una vez que el producto esté maduro y busquemos maximizar la calidad y el rendimiento sin reescribir la lógica central.

Arquitecturas Móviles Nativas

Optar por una arquitectura nativa implica tomar la decisión estratégica de construir dos aplicaciones separadas e independientes, una para iOS y otra para Android, utilizando los lenguajes, herramientas y paradigmas de diseño específicos de cada plataforma. Esta vía, aunque considerablemente más costosa en términos de tiempo, recursos y complejidad de gestión, ofrece el mayor nivel posible de calidad, rendimiento e integración con el ecosistema. Es el camino que suelen tomar las aplicaciones maduras que, habiendo validado su modelo de negocio, buscan diferenciarse a través de una experiencia de usuario superior y una simbiosis perfecta con el dispositivo del usuario.

Swift y el Ecosistema iOS

- **¿Qué es?** Swift es el lenguaje de programación moderno, seguro y de alto rendimiento creado por Apple para desarrollar aplicaciones en todo su ecosistema (iOS, iPadOS, etc.). Para la interfaz, se utiliza el framework SwiftUI, que permite un desarrollo declarativo y moderno.

- **Ventajas y Desventajas** La ventaja es que ofrece el máximo rendimiento posible, una integración perfecta con el hardware y software del dispositivo (como Face ID, Apple Wallet) y una experiencia de usuario que se siente

completamente en casa en la plataforma. La principal desventaja es el alto costo y tiempo, ya que se debe desarrollar y mantener una base de código completamente separada solo para los dispositivos de Apple.

- **¿Para qué la usaríamos?** El desarrollo nativo con Swift sería el camino a seguir si SaveApp, en una etapa de madurez, decidiera que la máxima calidad y una diferenciación a través de una experiencia de usuario superior en iOS es una prioridad estratégica, asumiendo los costos más elevados que esto implica.

Kotlin y el Ecosistema Android

- **¿Qué es?** Kotlin es el lenguaje de programación moderno, conciso y seguro, designado por Google como el lenguaje oficial para el desarrollo de aplicaciones Android. Funciona sobre la Máquina Virtual de Java (JVM) y es 100% interoperable con Java. Para la construcción de interfaces, se utiliza Jetpack Compose, el moderno toolkit declarativo de Google que simplifica y acelera el desarrollo de UI, siendo el equivalente directo a SwiftUI de Apple.

- **Ventajas y Desventajas** La principal ventaja es el acceso al máximo rendimiento, fidelidad visual y la más profunda integración posible con las funcionalidades del sistema operativo Android (widgets, notificaciones avanzadas, servicios en segundo plano). La gran desventaja, además del alto costo de mantener un código separado, es la fragmentación del ecosistema Android: es necesario garantizar la compatibilidad en una inmensa variedad de dispositivos, tamaños de pantalla y versiones del sistema operativo, lo que incrementa el esfuerzo de pruebas.

- **¿Para qué la usaríamos?** Al igual que Swift para iOS, usaríamos Kotlin y Jetpack Compose si la estrategia a largo plazo de SaveApp fuera ofrecer la experiencia de usuario de más alta calidad posible en la plataforma Android. Dada la masiva cuota de mercado de Android en Argentina y Latinoamérica, esta podría ser una decisión estratégica clave para lograr el liderazgo del mercado, una vez que el modelo de negocio esté validado y se busque la diferenciación a través de la excelencia del producto.

Geolocalización y Notificaciones Contextuales

Uno de los pilares de SaveApp es entregar información pertinente en el momento y lugar exactos, convirtiendo los descuentos en oportunidades de ahorro proactivas.

Geofencing

- **¿Qué es?** El Geofencing es una técnica que permite a una aplicación registrar perímetros geográficos virtuales ("geofences") en el sistema operativo del teléfono. En lugar de que la app pregunte constantemente por la ubicación, el sistema

operativo le notifica de forma pasiva cuando el dispositivo entra o sale de una de estas áreas predefinidas.

- **Ventajas y Desventajas** Su ventaja radica en su alta eficiencia energética. El sistema operativo utiliza fuentes de baja energía (torres de celular, Wi-Fi) para monitorear la ubicación y solo activa el GPS brevemente si es necesario, preservando la batería. La desventaja es que requiere una gestión cuidadosa desde el backend para registrar las cercas correctas para cada usuario.

- **¿Para qué la usaríamos?** Esta tecnología es la piedra angular para las notificaciones contextuales. El backend enviaría a la app una lista de geofences alrededor de los comercios con promociones relevantes para el usuario. Al detectar la entrada a una de estas zonas, se activaría el flujo para enviar una notificación push.

Notificaciones Push

- **¿Qué es?** Son los servicios oficiales, seguros y optimizados que Google (FCM) y Apple (APNs) proveen para que un servidor pueda enviar mensajes a una aplicación en un dispositivo, incluso si la app no está en uso o la pantalla está apagada.

- **Ventajas y Desventajas** Son el estándar de la industria, altamente fiables, seguros y eficientes en el uso de batería y datos. No tienen desventajas técnicas significativas, pero su poder conlleva una gran responsabilidad en términos de experiencia de usuario para no generar "fatiga de notificaciones".

- **¿Para qué la usaríamos?** Son el vehículo final que entrega la alerta al usuario. Cuando nuestro sistema detecte un evento de geofencing y valide que una promoción es relevante, el servidor de SaveApp le ordenará a FCM o APNs que envíen el mensaje construido ("¡Estás en COTO! Hoy tenés 20% de reintegro con tu tarjeta Visa Galicia") al dispositivo del usuario.

APIs Geoespaciales

Para que la geolocalización y las notificaciones contextuales funcionen de forma confiable, necesitamos una base sólida y actualizada de puntos geográficos (POIs) de comercios. Si bien extraemos datos de comercios adheridos a partir de las propias ofertas, esa información suele venir incompleta y poco estandarizada (nombres ambiguos, direcciones parciales, falta de coordenadas). Por eso incorporamos un servicio externo especializado para enriquecer, normalizar y geocodificar estos datos. Investigamos dos alternativas principales: Google Maps Platform y el ecosistema de OpenStreetMap (OSM).

Google Maps Platform

Suite comercial con amplia cobertura, alta precisión y varias sub-APIs. Requiere clave, cumple SLA empresariales y tiene costos por uso. Esta plataforma posee una gran precisión y cobertura, ofrece mucha información y está bien documentada; sin embargo su costo suele ser elevado. Las herramientas de la plataforma que se investigaron son las siguientes:

Places API

- **¿Qué hace?** Búsqueda y detalle de lugares (Place Search + Place Details). Permite encontrar sucursales por nombre, categoría o proximidad; devuelve `place_id`, coordenadas, dirección, nombre, horario, teléfono, website, fotos y rating.
- **¿Para qué la usaríamos?** Se utilizaría para saber, en el caso de una oferta multimarca, a qué marca pertenece cada comercio adherido que tiene la oferta.

Geocoding API

- **¿Qué hace?** Convierte direcciones a coordenadas (geocoding) y coordenadas a direcciones legibles (reverse geocoding).
- **¿Para qué la usaríamos?** Para la normalización de direcciones de ofertas incompletas y para reverse geocoding para poder completar la ubicación con sus coordenadas en el caso de que estas no sean provistas por la oferta original.

OpenStreetMap (OSM)

Ecosistema abierto y colaborativo. Sin costos de licencia, pero con límites de uso en servicios públicos y variabilidad de cobertura según zona. Su cobertura y cantidad de información es menor en comparación con Google Maps, se investigaron las siguientes herramientas:

Nominatim (Geocoding)

- **¿Qué hace?** Geocodificación y reverse geocoding sobre datos OSM.
- **¿Para qué la usaríamos?** Para convertir direcciones a coordenadas y obtener `osm_id`/`osm_type`.

Overpass API (POIs)

- **¿Qué hace?** Consultas flexibles sobre el grafo OSM (puntos, vías y relaciones) por tags (ej., `shop=supermarket`, `brand=*`).
- **¿Para qué la usaríamos?** Para descubrir y validar sucursales por marca/categoría en zonas específicas, como fuente adicional de POIs.

3. Dashboard y Landing

Desarrollo Web

La presencia digital de SaveApp no se limita a la aplicación móvil, requiere de una sólida plataforma web que cumpla dos funciones críticas y distintas. Por un lado, una landing page, que actúa como la carta de presentación del producto, enfocada en el marketing, la adquisición de usuarios y la optimización para motores de búsqueda (SEO). Por otro lado, un panel administrativo (dashboard) interno y seguro, que es el centro de control operativo

para el equipo de SaveApp. Dado que los requisitos técnicos de estos dos productos son diametralmente opuestos (el primero demanda rendimiento extremo y visibilidad, mientras que el segundo exige funcionalidad, gestión de datos y seguridad), se ha investigado un amplio espectro de tecnologías de frontend, backend y servicios auxiliares para determinar las combinaciones más adecuadas.

Librerías de UI: La Base de la Interfaz

Una librería de UI proporciona los bloques de construcción fundamentales (componentes) para crear la interfaz con la que el usuario interactúa.

React

- **¿Qué es?** Es una librería de JavaScript, mantenida por Meta, para construir interfaces de usuario interactivas y dinámicas basadas en un sistema de componentes. React gestiona cómo se ve y se actualiza la UI en respuesta a los datos y las acciones del usuario.
- **Ventajas y Desventajas:** Su principal ventaja es su inmensa popularidad, lo que se traduce en una comunidad masiva, una cantidad ingente de librerías, tutoriales y una gran disponibilidad de desarrolladores. Es flexible y puede ser integrado en casi cualquier tipo de proyecto.
- **¿Para qué la usaríamos?** Ideal para el dashboard administrativo interno de SaveApp. Crearíamos una aplicación de cliente (Client-Side Rendering) que consume la API del backend para mostrar datos dinámicos, gestionar formularios complejos y ofrecer una experiencia de usuario rica e interactiva.

Shadcn

- **¿Qué es?** Es una colección de componentes de interfaz de usuario construidos sobre Radix UI y estilizados con Tailwind CSS, pensada para aplicaciones modernas en React y Next.js. No es un framework cerrado: el código de cada componente se copia en el proyecto, lo que permite una personalización total sin depender de una librería externa en tiempo de ejecución.
- **Ventajas y Desventajas:** La mayor ventaja es la calidad visual y la coherencia de diseño lista para usar, con soporte para accesibilidad y buenas prácticas. El hecho de incorporar el código al repositorio facilita modificar estilos, lógica o estructura sin limitaciones. Como desventaja, requiere mantener manualmente las actualizaciones de componentes si se quieren incorporar mejoras o correcciones del repositorio original.
- **¿Para qué la usaríamos?** Para acelerar la creación del dashboard administrativo, garantizando una UI consistente, moderna y accesible, con la flexibilidad de adaptar los componentes a las necesidades específicas del producto.

Meta-Frameworks: Estructura y Optimización

Los meta-frameworks se construyen sobre librerías como React para ofrecer soluciones completas que incluyen enrutamiento, renderizado en el servidor y optimizaciones de rendimiento "de fábrica".

Next.js (sobre React)

- **¿Qué es?** Es el meta-framework de producción para React. Extiende sus capacidades con funcionalidades cruciales como el renderizado del lado del servidor (SSR), la generación de sitios estáticos (SSG), optimización de imágenes y un sistema de enrutamiento basado en ficheros.
- **Ventajas y Desventajas:** Permite construir aplicaciones React completas y de alto rendimiento sin tener que configurar manualmente herramientas complejas. Su enfoque en SSR y SSG es excelente para el SEO y la velocidad de carga inicial.
- **¿Para qué la usaríamos?** Es la opción principal para construir la landing page pública de SaveApp. Usaríamos su modo de generación estática (SSG) para garantizar tiempos de carga casi instantáneos y una optimización perfecta para los motores de búsqueda (SEO).

Astro

- **¿Qué es?** Astro es un framework diseñado para construir sitios web rápidos y centrados en el contenido. Su innovación es la arquitectura de "islas", que renderiza HTML estático (cero JavaScript por defecto) y solo carga el JS de los componentes interactivos que lo necesitan.
- **Ventajas y Desventajas:** Velocidad extrema para sitios de contenido al minimizar el JavaScript. Es menos adecuado para aplicaciones web altamente dinámicas como un dashboard.
- **¿Para qué la usaríamos?** Sería una opción excepcional y altamente especializada para construir la landing page de SaveApp, garantizando una velocidad de carga de élite.

Herramientas de Desarrollo y Build

Estas herramientas son esenciales para el proceso de desarrollo, ya que compilan, empaquetan y sirven el código de la aplicación.

Vite

- **¿Qué es?** Es una herramienta de build y servidor de desarrollo moderno. Utiliza los módulos ES nativos del navegador para ofrecer un servidor de desarrollo extremadamente rápido y un proceso de empaquetado optimizado para producción.

- **Ventajas y Desventajas:** Su principal ventaja es la velocidad y la experiencia de desarrollo (DX) superiores. Es prácticamente instantáneo al iniciar y al aplicar cambios (Hot Module Replacement). Es agnóstico al framework, compatible con React, Vue, Svelte y otros.

- **¿Para qué la usaríamos?** Sería la herramienta elegida para crear y servir localmente nuestra aplicación de cliente de React para el dashboard. Su velocidad acelera drásticamente el ciclo de desarrollo.

Frameworks de Backend para la Web

El backend es el motor de la aplicación: procesa la lógica de negocio, interactúa con la base de datos y sirve los datos que consumen las interfaces de usuario. La elección de la tecnología de backend es crítica y depende de factores como los requisitos de rendimiento, la escala del proyecto y la experiencia del equipo de desarrollo.

Django

- **¿Qué es?** Un framework de alto nivel que sigue la filosofía de black box. Django proporciona una solución completa y robusta desde el primer momento, destacando por su potente **ORM** (Object-Relational Mapper) y, sobre todo, por su panel de administración autogenerado.

- **Ventajas y Desventajas:** Su principal ventaja es la velocidad de desarrollo para aplicaciones CRUD (Crear, Leer, Actualizar, Borrar). El panel de administración es un acelerador masivo. Por otro lado, su naturaleza monolítica y opinionada puede hacerlo menos flexible para microservicios o arquitecturas no convencionales.

- **¿Para qué lo usaríamos?** Es la opción ideal para generar de forma casi instantánea un dashboard administrativo interno y funcional para SaveApp. Permite gestionar todos los datos de la base de datos con un esfuerzo de desarrollo mínimo, ideal para una primera versión o para herramientas internas.

FastAPI

- **¿Qué es?** Un micro-framework moderno de Python, diseñado específicamente para construir APIs de alto rendimiento. Utiliza características modernas del lenguaje (como *type hints*) para ofrecer validación de datos, serialización y documentación de API interactiva automática (vía Swagger UI y ReDoc).

- **Ventajas y Desventajas:** Su rendimiento es excepcional, comparable al de Node.js o Go. La documentación automática ahorra muchísimo tiempo y mejora la comunicación entre equipos. Al ser un micro-framework, no incluye componentes como un ORM o un panel de admin, lo que da más flexibilidad pero requiere integrar librerías de terceros.

- **¿Para qué la usaríamos?** Es la elección perfecta para construir la API principal que será consumida por el frontend y la aplicación móvil. Su velocidad y eficiencia son clave para una experiencia de usuario fluida.

Node.js

- **¿Qué es?** Es un entorno de ejecución de JavaScript del lado del servidor, construido sobre el motor V8 de Chrome. Permite a los desarrolladores usar JavaScript para escribir el código del backend, como APIs, interactuar con bases de datos y gestionar la lógica de negocio.
- **Ventajas y Desventajas:** Su mayor ventaja es la posibilidad de usar JavaScript en todo el stack (full-stack), unificando el lenguaje del frontend y el backend. Su ecosistema, gestionado por NPM (Node Package Manager), es el más grande del mundo. Su modelo asíncrono lo hace muy eficiente para aplicaciones que manejan muchas conexiones simultáneas.
- **¿Para qué la usaríamos?** Es la base sobre la que construiríamos toda la API REST de SaveApp. Esta API sería la encargada de comunicarse con la base de datos, procesar los datos, autenticar a los usuarios y servir toda la información que necesita el dashboard administrativo para funcionar.

Go (Golang)

- **¿Qué es?** Go no es un framework, sino un lenguaje con una librería estándar muy potente que permite construir servidores web de alto rendimiento sin necesidad de dependencias externas. Su principal fortaleza es la concurrencia, manejada a través de *goroutines*.
- **Ventajas y Desventajas:** Produce un único archivo binario ejecutable sin dependencias, lo que simplifica enormemente el despliegue (ideal para contenedores como Docker). Ofrece un rendimiento excepcional y un consumo de memoria muy bajo. Su ecosistema de librerías, aunque creciente, no es tan vasto como el de Python o Node.js.
- **¿Para qué lo usaríamos?** Go sería la opción preferida para construir microservicios de muy alto rendimiento dentro de la arquitectura de SaveApp.

4. Bases de Datos

La naturaleza de los datos de SaveApp es inherentemente diversa. Coexisten datos relacionales y estructurados (usuarios, tarjetas), datos semi-estructurados y flexibles (las promociones con sus innumerables condiciones), datos geoespaciales, datos de búsqueda de texto y archivos binarios como imágenes. Confiar en una única tecnología para todas estas tareas sería subóptimo. Un enfoque de persistencia políglota, seleccionando la herramienta adecuada para cada trabajo, es la estrategia más robusta.

Bases de Datos Relacionales (SQL)

Ideales para datos estructurados donde la consistencia y la integridad son la máxima prioridad.

PostgreSQL

- **¿Qué es?** Es un sistema de gestión de bases de datos objeto-relacional de código abierto, reconocido por su robustez, cumplimiento de estándares y un conjunto de características avanzadas.

- **Ventajas y Desventajas** Su principal fortaleza son las transacciones ACID, que garantizan la integridad de los datos. Su extensión PostGIS lo convierte en una base de datos geoespacial de primer nivel, y su soporte para tipos de datos complejos como JSONB es excelente.

- **¿Para qué la usaríamos?** Sería la base de datos para los datos más críticos: perfiles de usuario, credenciales (hasheadas), tarjetas registradas y registros de auditoría. Además, con PostGIS, la usaríamos para almacenar la ubicación de los comercios y potenciar toda la funcionalidad de geolocalización.

MySQL / MariaDB

- **¿Qué es?** MySQL es la base de datos relacional de código abierto más popular del mundo, conocida por su fiabilidad y facilidad de uso. MariaDB es su "fork" (derivado) creado por la comunidad, que mantiene una alta compatibilidad y a menudo introduce características de vanguardia.

- **Ventajas y Desventajas** Su ventaja es su enorme popularidad, lo que se traduce en una vasta comunidad, documentación extensa y un gran soporte en casi todas las plataformas de hosting. Es reconocida por su excelente rendimiento en cargas de trabajo intensivas en lectura. Puede ser menos rica en funciones que PostgreSQL para consultas muy complejas.

- **¿Para qué la usaríamos?** Sería una alternativa directa y muy sólida a PostgreSQL para almacenar los datos estructurados del núcleo de la aplicación (usuarios, tarjetas, etc.). La elección entre MySQL y PostgreSQL a menudo se reduce a la familiaridad del equipo de desarrollo y a necesidades de características específicas.

Bases de Datos No Relacionales (NoSQL)

Diseñadas para la flexibilidad, la escalabilidad y para manejar datos que no se ajustan bien a un esquema de tablas y filas.

MongoDB

- **¿Qué es?** Es una base de datos NoSQL líder, orientada a documentos, que almacena los datos en un formato flexible similar a JSON llamado BSON.

- **Ventajas y Desventajas** Su ventaja clave es la flexibilidad de esquema. Permite almacenar datos cuya estructura es variable y difícil de predecir,

como las promociones. Esto simplifica enormemente el desarrollo y la evolución de la aplicación.

- **¿Para qué la usaríamos?** Sería la base de datos elegida para almacenar toda la información de las promociones. Cada promoción, con sus condiciones únicas y variables, sería un único documento, lo que nos permitiría adaptarnos a la diversidad de ofertas del mercado con gran agilidad.

Firebase Firestore

- **¿Qué es?** Es una base de datos de documentos NoSQL, flexible y escalable, ofrecida por Google, cuya característica más destacada es su capacidad de sincronización de datos en tiempo real con los clientes conectados.

- **Ventajas y Desventajas** Su gran ventaja es la capacidad de propagar cambios en la base de datos a las aplicaciones casi instantáneamente. Sus capacidades de consulta son más limitadas, pero su facilidad de integración y naturaleza en tiempo real son un gran plus.

- **¿Para qué la usaríamos?** La aprovecharíamos para guardar metadata y configuraciones de los usuarios, además de almacenar los UID de los usuarios de firebase para poder enviar las push notifications.

Redis

- **¿Qué es?** Redis es un almacén de estructuras de datos en memoria, extremadamente rápido, que se utiliza principalmente como base de datos, caché y agente de mensajes.

- **Ventajas y Desventajas** Su ventaja es su velocidad vertiginosa, ya que opera directamente sobre la RAM. Esto permite responder a consultas en milisegundos, reduciendo la carga sobre las bases de datos principales y mejorando la percepción de velocidad de la app.

- **¿Para qué la usaríamos?** Desempeñaría el rol crucial de una capa de caché. Almacenaríamos en Redis los resultados de consultas frecuentes o costosas (ej: "top 10 de promociones en un barrio específico") para servirlos de forma casi instantánea, mejorando la experiencia del usuario y la escalabilidad del sistema.

5. ETL y Chatbot

Extracción de Datos y Automatización (Web Scraping)

El corazón operativo de SaveApp es su capacidad para obtener y procesar información de un ecosistema digital no diseñado para la interoperabilidad. La estrategia de web scraping

debe combinar herramientas resilientes y escalables para emular la interacción humana de manera convincente.

BeautifulSoup

- **¿Qué es?** BeautifulSoup es una librería de Python diseñada para el análisis sintáctico (parsing) de documentos HTML y XML. Su función principal es transformar el código fuente estático de una página web en una estructura de datos organizada (un árbol de objetos), permitiendo a los desarrolladores navegar, buscar y extraer información específica de manera programática.

- **Ventajas y Desventajas** La ventaja fundamental de BeautifulSoup radica en su simplicidad y curva de aprendizaje casi plana, con una API legible, tolerante a HTML imperfecto y muy eficaz para navegar árboles DOM y localizar patrones repetitivos. Su única desventaja relevante es que no soporta XPath de forma nativa, trabaja con selectores CSS y búsquedas por atributos.

- **¿Para qué la usaríamos?** La usaríamos en los casos donde las promociones estén publicadas directamente en el HTML sin un backend estructurado ni endpoints. En ese escenario, BeautifulSoup permite extraer con rapidez y fiabilidad el contenido del DOM (listas, tablas, tarjetas de oferta y metadatos).

Scrapy

- **¿Qué es?** Scrapy no es una simple librería, sino un completo framework de crawling y scraping de alto nivel. Construido sobre una arquitectura asíncrona, está diseñado para gestionar múltiples peticiones de red de forma concurrente, lo que le confiere una velocidad y eficiencia muy superiores para la extracción masiva de datos.

- **Ventajas y Desventajas** Su principal ventaja es la robustez y escalabilidad. Permite orquestar todo el proceso de scraping, definir lógicas de extracción específicas por sitio ("spiders"), manejar errores de red, gestionar sesiones y procesar los datos extraídos en un flujo ordenado ("pipeline"). Su desventaja es una curva de aprendizaje más pronunciada que herramientas más simples, aunque la inversión se traduce en un sistema más mantenible.

- **¿Para qué la usaríamos?** Scrapy actuaría como el sistema nervioso central de la recolección de información. Definiríamos "spiders" para cada portal bancario (ej. GaliciaSpider, NaranjaXSpider), que navegarían de forma autónoma para recolectar las promociones, constituyendo la base del sistema de extracción de datos a gran escala.

Playwright (o Selenium)

- **¿Qué es?** Playwright es una moderna librería de automatización de navegadores que ofrece control programático sobre navegadores completos como Chromium, Firefox y WebKit. Su función es emular de manera fidedigna la interacción de un

usuario humano con un sitio web, pudiendo ejecutarse en modo "headless" (sin interfaz gráfica) para su despliegue en servidores.

- **Ventajas y Desventajas** La ventaja crítica de Playwright es su capacidad para resolver el problema del contenido dinámico, ya que procesa el JavaScript de las páginas. Su gran desventaja es que consume significativamente más recursos de CPU y memoria y es más lento que las peticiones directas, lo que se traduce en mayores costos de infraestructura.

- **¿Para qué la usaríamos?** Playwright es la pieza tecnológica clave para extraer datos de los portales bancarios modernos. Esta herramienta podría usarse cuando las ofertas se encuentran en HTML y el portal bancario genera ese contenido de manera dinámica, por otro lado, es muy útil frente a barreras anti-scrapers en donde se necesita sobrepasarlas mediante un navegador automatizado.

Agentes de IA para Web Scraping (Emergente)

- **¿Qué es?** Este enfoque representa un cambio de paradigma que utiliza un LLM, a menudo con capacidades de visión, para interpretar una página web de forma contextual. En lugar de buscar elementos por su código (selectores CSS o XPath), el agente revisa el dominio y entiende semánticamente dónde está la información relevante.

- **Ventajas y Desventajas** A largo plazo, su gran ventaja es la resiliencia: si un sitio web cambia su diseño, el agente de IA podría adaptarse sin necesidad de reescribir el código, reduciendo costos de mantenimiento. Sus desventajas actuales son el alto costo por cada análisis, la alta latencia y una fiabilidad que aún no es comparable a los métodos tradicionales.

- **¿Para qué la usaríamos?** Actualmente, no es una solución viable para el producto inicial debido a sus limitaciones y costos. Sin embargo, la consideramos una línea de investigación y desarrollo futuro para SaveApp, con el potencial de automatizar y robustecer el mantenimiento de los scrapers a largo plazo.

Procesamiento de Lenguaje Natural (NLP) y Modelos de Lenguaje

Las descripciones de las promociones están redactadas en lenguaje natural denso y ambiguo. Es crucial extraer y estructurar sus reglas de negocio con precisión para que el sistema sea funcional.

Expresiones Regulares

- **¿Qué es?** Las Expresiones Regulares (Regex) son un lenguaje formal de patrones para buscar y manipular texto.

- **Ventajas y Desventajas** Son computacionalmente muy baratas, rápidas y eficientes para extraer entidades predecibles y bien definidas (porcentajes, montos, fechas). Su limitación fundamental es su fragilidad y falta de comprensión contextual, no

pueden discernir el significado de un monto (si es un tope, un mínimo, etc.) ni manejar la variabilidad del lenguaje.

- **¿Para qué la usaríamos?** Servirían como una capa de pre-procesamiento rápida y eficiente. Las utilizaríamos para una primera pasada de extracción de datos estructurados y fáciles de identificar, antes de pasar el texto a modelos más complejos para su interpretación semántica.

Modelos de Lenguaje (LLMs)

- **¿Qué es?** Los LLMs han desarrollado una capacidad sin precedentes para comprender el contexto, la semántica y la intención detrás del lenguaje natural. La interacción se realiza a través de una API, enviando instrucciones precisas (prompts).

- **Ventajas y Desventajas** Su principal ventaja es su inmensa flexibilidad para manejar la variabilidad y la jerga del lenguaje de las promociones, permitiendo una extracción de datos estructurados muy precisa. Las desventajas son el costo asociado al uso de sus APIs (se paga por token procesado) y el riesgo de "alucinaciones" (datos incorrectos), lo que requiere una capa de validación posterior.

- **¿Para qué la usaríamos?** Los LLMs serían el cerebro encargado de interpretar las condiciones complejas de cada promoción. Les enviaríamos el texto de una oferta con un prompt detallado para que devuelvan un objeto JSON estructurado con todos sus atributos (tipo de beneficio, valor, tope, días, bancos, etc.), transformando el texto no estructurado en datos útiles para la aplicación.

Embeddings y Bases de Datos Vectoriales

- **¿Qué es?** Un modelo de "embeddings" convierte un fragmento de texto en un vector numérico que representa su significado semántico. Una base de datos vectorial es un sistema optimizado para almacenar estos vectores y realizar búsquedas de similitud ultrarrápidas, encontrando textos con significados parecidos en lugar de solo palabras clave.

- **Ventajas y Desventajas** La ventaja es que habilita funcionalidades de búsqueda y recomendación mucho más inteligentes y naturales para el usuario. La principal desventaja es que añade una capa de complejidad a la arquitectura de la aplicación y a los costos operativos.

- **¿Para qué la usaríamos?** Esta tecnología potenciaría funcionalidades de alto valor. Nos permitiría implementar una búsqueda donde un usuario pueda escribir "salidas de fin de semana" y encontrar promociones en cines y restaurantes, o potenciar un motor que recomiende ofertas similares a las que el usuario ha guardado, mejorando drásticamente el descubrimiento y la experiencia.

MCP Servers

- **¿Qué es?** Son servicios que actúan como intermediarios entre un modelo de lenguaje y sistemas externos, permitiendo que el modelo acceda a datos, herramientas o funciones específicas de manera controlada y estructurada. Operan mediante un protocolo estandarizado que define cómo el modelo solicita información o ejecuta acciones fuera de su propio contexto.

- **Ventajas y Desventajas:** Su principal ventaja es que amplían las capacidades del modelo sin comprometer la seguridad ni exponer directamente la infraestructura interna, ya que las interacciones se canalizan a través de endpoints controlados. Además, permiten reutilizar funciones y datos en diferentes flujos de trabajo de forma consistente. La desventaja es que requieren un diseño cuidadoso del protocolo y las funciones expuestas, así como una infraestructura adicional para mantener la disponibilidad y seguridad del servicio.

- **¿Para qué la usaríamos?** Los MCP Servers serían el puente entre los LLMs y los datos operativos de SaveApp. Los usaríamos para que el modelo consulte catálogos actualizados de bancos, tarjetas y comercios; valide umbrales o formatos; y ejecute funciones de negocio específicas sin exponer directamente la base de datos o la lógica interna del sistema.

Técnicas de Prompting

Esta sección presenta las técnicas de prompting que guiarán la interacción con modelos de lenguaje en SaveApp. Su correcta aplicación impacta directamente en la precisión y consistencia de la extracción y normalización de promociones, en el control de calidad y la reproducibilidad de resultados, así como en los costos y la latencia operativa del sistema.

Estructuración con Markdown

- **¿Qué es?** Organización del prompt en secciones nítidas (rol, objetivo, datos, reglas, formato) y delimitación explícita del texto a procesar.

- **Ventajas y desventajas.** Esta técnica reduce ambigüedades, evita la deriva de instrucciones y facilita la auditoría entre equipos, ya que todos leen y versionan la misma estructura. A cambio, exige disciplina para mantener convenciones coherentes y actualizar las reglas cuando cambian las políticas.

- **¿Para qué la usaríamos?** Para estructurar los prompts del ETL jerárquicamente.

One-shot y Few-shot

- **¿Qué es?** Inclusión de uno o pocos casos representativos para orientar formato, criterios y nivel de detalle sin necesidad de entrenar el modelo.

- **Ventajas y desventajas.** Alinea rápidamente las salidas con nuestro esquema y estilo, y evita el costo y complejidad de un fine-tuning inicial; sin embargo, puede

sesgar si los ejemplos no cubren la diversidad real del dominio y agrega consumo de contexto.

- **¿Para qué la usaríamos?** Para asegurar que las promociones salgan con nuestro JSON canónico, enseñar matices (como distinguir “descuento” vs. “reintegro”), calibrar el tono de textos cortos y guiar normalizaciones de entidades (bancos, tarjetas, categorías) de forma económica.

Razonamiento (Chain-of-Thought)

- **¿Qué es?** Inducir razonamiento interno paso a paso, limitando la salida a la respuesta final o a una justificación breve y controlada.

- **Ventajas y desventajas.** Mejora la exactitud en tareas con múltiples condiciones y fechas, reduce confusiones entre conceptos cercanos y eleva la coherencia global; por contra, tiende a aumentar latencia y costo, y requiere redactar instrucciones cuidadosas para evitar exponer trazas sensibles en la salida.

- **¿Para qué la usaríamos?** Para interpretar T&C complejas, validar coherencia de vigencias, resolver conflictos de enunciados, decidir si una promo aplica a una tarjeta concreta y determinar cuándo corresponde abstenerse o marcar campos como desconocidos.

LLM as a Judge

- **¿Qué es?** Uso de un LLM como evaluador que puntúa o compara salidas según una rúbrica de calidad definida (exactitud, consistencia con evidencia, formato).

- **Ventajas y desventajas.** Permite escalar control de calidad sin revisar todo a mano y habilita experimentos A/B entre prompts o variantes, pero puede heredar sesgos del modelo evaluador y exige rúbricas claras, umbrales y conjuntos de referencia; además añade un paso extra de cómputo.

- **¿Para qué la usaríamos?** Como compuerta de calidad antes de publicar promociones, para elegir la mejor extracción entre varias candidatas, detectar campos dudosos que requieren revisión humana y monitorear degradaciones cuando cambian los sitios.

Structured Output

- **¿Qué es?** Forzar que la salida cumpla un esquema estructurado (tipos, rangos, fechas ISO) y evitar texto libre.

- **Ventajas y desventajas.** Facilita el parseo y la validación automática en el backend, permite reintentos controlados si el esquema no valida y mejora la auditabilidad por

campo; la contracara es que esquemas demasiado rígidos pueden romper ante casos reales limítrofes y obligan a gestionar explícitamente datos faltantes o inciertos.

- **¿Para qué la usaríamos?** Para la ingesta directa de promociones al pipeline, el control de versiones de atributos.

Plantillas de system prompt dinámicas

- **¿Qué es?** Técnica para definir system prompts con variables que se inyectan dinámicamente en el pipeline o workflow, según el contexto o la tarea.
- **Ventajas y desventajas.** Permite personalizar las instrucciones en tiempo de ejecución, limitar el espacio de respuesta del modelo y reutilizar la misma plantilla para distintos escenarios. Sin embargo, requiere controlar la correcta inyección de variables y validar que el LLM interprete las restricciones según lo esperado.
- **¿Para qué la usaríamos?** Para ajustar el comportamiento del modelo en función de datos o condiciones específicas (restringir la selección de tarjetas, filtrar catálogos o acotar umbrales) sin necesidad de redefinir manualmente el prompt base.

Tool / Function Calling

- **¿Qué es?** Mecanismo que permite a un LLM invocar funciones predefinidas (p. ej., normalizar entidades, validar fechas, consultar catálogos) en lugar de responder con texto libre, integrando el modelo directamente con la lógica de negocio.
- **Ventajas y desventajas.** Ofrece control preciso sobre las respuestas, reduce la necesidad de interpretar o corregir salidas del modelo y facilita flujos conversacionales estructurados (validar → consultar → responder) sin riesgo de formatos inválidos. Como contrapartida, exige soporte en la plataforma del chatbot, un diseño claro de funciones y una gestión cuidadosa de errores y tiempos de respuesta.
- **¿Para qué la usaríamos?** En el chatbot, para que el modelo pueda, durante una conversación, consultar catálogos vivos de bancos, tarjetas o categorías, validar rangos y formatos, acceder a índices para deduplicación o registrar métricas y decisiones en tiempo real.

Selección semántica de contexto (RAG ligero)

- **¿Qué es?** Recuperar solo fragmentos relevantes por similitud semántica y usarlos como contexto, evitando enviar documentos completos.
- **Ventajas y desventajas.** Disminuye costo y latencia, reduce ruido y mejora precisión al focalizar la información, y facilita auditoría de procedencia; sin embargo,

depende de la calidad del recuperador y del índice, que requieren mantenimiento y umbrales bien ajustados.

- **¿Para qué la usaríamos?** Para que el chatbot pueda acceder a información de ofertas, términos y condiciones, políticas, etc.

Calibración de decodificación

- **¿Qué es?** Ajuste de temperatura, top-p, longitud máxima, penalizaciones y semillas para controlar variabilidad, costo y reproducibilidad.

- **Ventajas y desventajas.** Permite extracciones deterministas cuando la estabilidad es crítica y aporta variedad controlada en textos orientados a UX, además de optimizar el gasto por tarea, la desventaja es que requiere *tuning* por dominio y puede necesitar revisiones cuando cambian los modelos base.

- **¿Para qué la usaríamos?** Para mantener la consistencia en la extracción y normalización de datos, y asegurar que se cumplan los tiempos y costos incluso en picos de tráfico.

LLM Gateways

Esta sección reúne pasarelas multi-proveedor (proxies) que unifican el acceso a modelos (OpenAI, Anthropic, Google, Meta, etc.) detrás de un endpoint único, con observabilidad, ruteo/fallback, rate limiting, caché y control de costos. Reducen fricción para cambiar de modelo y estandarizan métricas y políticas.

Vercel AI Gateway

- **¿Qué es?** Pasarela AI/LLM que se interpone entre tu app y múltiples proveedores para unificar claves, modelos y políticas (cuotas, límites, cacheo, retrys) con observabilidad y analítica de tokens/latencias.

- **Ventajas y Desventajas:** la ventaja radica en un endpoint único multi-proveedor, ruteo/fallback, rate limits y caps por ruta/modelo, caché para prompts estables, panel de métricas integrado, DX sólida (middlewares, SDKs). Tiene como desventaja otro salto de red (compliance/latencia), menos control que un gateway self-hosted, y ligero lock-in al ecosistema Vercel para algunas integraciones avanzadas.

- **¿Para qué la usaríamos?** Centralizar claves/cuotas, aplicar políticas de costo/latencia, habilitar fallback entre proveedores y bajar costos con caché en prompts determinísticos (por ejemplo, validaciones o extractores estables).

OpenRouter

- **¿Qué es?** API unificada compatible con OpenAI para decenas/cientos de modelos. Permite cambiar de modelo vía model sin tocar SDKs, con ruteo/fallback, facturación centralizada y BYOK en varios casos.

- **Ventajas y Desventajas:** tiene un catálogo amplio, permite un cambio rápido de modelos, posee precios competitivos, endpoint simple y métricas razonables sin armar infraestructura propia. Su contra radica en la dependencia de un tercero para facturación y límites, variabilidad entre proveedores detrás del mismo endpoint y menos controles enterprise que opciones self-hosted.

- **¿Para qué la usaríamos?** Experimentación/A-B entre modelos sin reescribir código, fallback automático cuando un vendor cae o rate-limitea, y cobertura rápida de nuevos modelos/regiones.

Prompt Management Tools

Esta sección reúne herramientas para versionar, auditar y optimizar prompts, trazas y costos en flujos con LLMs. Su adopción mejora la reproducibilidad, el control de calidad y la velocidad de iteración del equipo.

PromptLayer

- **¿Qué es?** Plataforma para versionar prompts y registrar ejecuciones (prompt, respuesta, latencia, costo) con integración directa vía SDK o middleware.

- **Ventajas y Desventajas:** permite el versionado de prompts con histórico, comparaciones A/B simples, registro de runs y métricas operativas, integración rápida. Como desventaja tiene que el foco principal es en el logging/versionado (menos énfasis en experimentación avanzada y evaluaciones automáticas).

- **¿Para qué la usaríamos?** Para mantener un historial auditable de los prompts del Crawler/ETL y del chatbot, comparar variantes y revertir ante regresiones en extracción o clasificación.

Helicone

- **¿Qué es?** Capa de observabilidad y control de costos que actúa como proxy. Funciona tanto como LLM gateway ligero (proxy compatible con múltiples proveedores) como Prompt Management Tool orientada a tracing, etiquetado y sanitización de PII.

- **Ventajas y Desventajas:** ofrece métricas de uso y costo por ruta/usuario, dashboards en tiempo real, etiquetado de requests, sampling, redacción de PII, cache y rate limits básicos; se integra rápido y permite exportar datos para análisis. Como desventaja tiene que las optimizaciones avanzadas requieren configuración/tuning, y no es un router inteligente “full” (ruteo/fallback menos sofisticado que gateways dedicados).

- **¿Para qué la usaríamos?** Para monitorear costo/latencia de pipelines (extracción, validación, Judge), detectar picos, aplicar límites y activar caché en prompts costosos y estables; además, como proxy unificador sencillo cuando queremos centralizar logging y controles sobre varios proveedores.

Langfuse

- **¿Qué es?** Plataforma de trazabilidad (tracing) y evaluación para LLMs con suites de experimentación, datasets y scoring.
- **Ventajas y Desventajas:** permite realizar trazas detalladas paso a paso (incluye tool-calls), experimentos A/B, datasets de referencia, métricas y evaluaciones automáticas, gestión de prompts con versiones. Como contra posee una curva de adopción mayor (definir rúbricas/datasets) y requiere disciplina de etiquetado.
- **¿Para qué la usaríamos?** Como "centro de control" del pipeline semántico: comparar prompts, medir exactitud de campos clave (tope, vigencia, tarjetas), monitorear degradaciones cuando cambian los sitios y aprobar releases del ETL con evidencia.

6. Cloud

Plataformas de Despliegue Cloud

Esta sección detalla las plataformas donde se alojará y ejecutará toda la infraestructura de SaveApp. La elección impacta directamente en la velocidad de desarrollo, la escalabilidad y los costos operativos.

Railway

- **¿Qué es?** Railway es una moderna Plataforma como Servicio (PaaS) diseñada con un enfoque radical en la simplicidad. Permite desplegar una aplicación completa directamente desde un repositorio de código (ej. GitHub), aprovisionando y conectando automáticamente todos los servicios necesarios, como bases de datos, cachés y backends.
- **Ventajas y Desventajas:** Su principal ventaja es la extrema facilidad de uso y la velocidad de despliegue, lo que permite pasar del código a una aplicación funcional en minutos. La desventaja es que ofrece menos control granular sobre la infraestructura en comparación con un proveedor IaaS y es más costoso.
- **¿Para qué la usaríamos?** Sería un candidato ideal para desplegar el MVP de SaveApp. Nos permitiría lanzar el backend, la base de datos y la caché como servicios interconectados con un mínimo esfuerzo de configuración. Esto maximizaría la velocidad de entrega al permitir que el equipo se enfoque exclusivamente en el desarrollo de funcionalidades.

Vercel

- **¿Qué es?** Vercel es una Plataforma como Servicio (PaaS) altamente especializada, creada por el equipo detrás de Next.js. Está optimizada para el despliegue de frontends modernos y funciones serverless, con un enfoque principal en el rendimiento web y la experiencia del desarrollador.

- **Ventajas y Desventajas:** Su integración con Next.js es inmejorable, ofreciendo optimizaciones automáticas de rendimiento y una red de distribución de contenido (CDN) global que hace que los sitios web sean extremadamente rápidos. Su función de "Preview Deployments" para cada cambio de código facilita la colaboración. Su desventaja principal es su alto costo.

- **¿Para qué la usaríamos?** Sería la elección predilecta y especializada para desplegar la landing page y el dashboard administrativo. Alojaríamos el frontend en Vercel para garantizar la máxima velocidad de carga, mientras que los servicios de backend (API, scrapers, base de datos) se alojarían en otra plataforma como Railway o AWS.

Amazon Web Services (AWS)

- **¿Qué es?** AWS es el líder mundial en Infraestructura como Servicio (IaaS). Ofrece un catálogo inmenso de más de 200 servicios en la nube, que funcionan como bloques de construcción para crear cualquier tipo de arquitectura, desde servidores virtuales (EC2) y bases de datos gestionadas (RDS) hasta servicios de machine learning.

- **Ventajas y Desventajas:** Sus ventajas son la enorme variedad de servicios, su fiabilidad y escalabilidad probadas, y ser el estándar de la industria. La principal desventaja es su complejidad; la gestión de la infraestructura y el control de costos requieren un conocimiento técnico especializado (DevOps) y pueden ser abrumadores para un equipo pequeño.

- **¿Para qué la usaríamos?** Adoptaríamos AWS en una etapa de madurez de SaveApp, cuando necesitemos un control total sobre la infraestructura para optimizar costos y rendimiento a gran escala. Nos permitiría diseñar una arquitectura a medida, pero no sería la opción para el MVP debido a su alta sobrecarga operativa inicial.

Google Cloud Platform (GCP)

- **¿Qué es?** GCP es la plataforma de nube de Google y uno de los principales competidores de AWS. Ofrece un conjunto completo de servicios de IaaS y PaaS, y es particularmente reconocida por su excelencia en áreas como el análisis de datos, el machine learning (Vertex AI) y la orquestación de contenedores con Kubernetes (GKE).

- **Ventajas y Desventajas:** Su gran ventaja es su profunda integración con Firebase, lo que simplifica enormemente el desarrollo de la aplicación móvil y el backend si se elige ese ecosistema. Además, GCP ofrece una muy buena capa gratuita y créditos iniciales, permitiendo operar con costos muy bajos al principio. Como desventaja, aunque es un jugador masivo, su catálogo de servicios y su cuota de mercado global son menores que los de AWS.

- **¿Para qué la usaríamos?** Al igual que AWS, GCP sería una opción para una fase de crecimiento posterior. Podríamos elegirla específicamente si quisiéramos aprovechar sus potentes herramientas de IA para mejorar el procesamiento de las promociones. La decisión entre GCP y AWS dependería de las necesidades específicas del proyecto y la experiencia del equipo técnico.

Microsoft Azure

- **¿Qué es?** Microsoft Azure es la plataforma de computación en la nube de Microsoft, y uno de los tres gigantes del mercado junto con AWS y GCP. Ofrece un catálogo extremadamente amplio de servicios que cubren IaaS, PaaS y SaaS, y se destaca por su fuerte posicionamiento en el mercado empresarial y su integración nativa con el ecosistema de desarrollo y productividad de Microsoft.

- **Ventajas y Desventajas:** Su principal ventaja es la integración profunda con herramientas como Azure DevOps y Active Directory, lo que la convierte en una opción natural para equipos que ya operan dentro del ecosistema de Microsoft. Es particularmente fuerte en sus ofertas de Plataforma como Servicio (PaaS) y es el proveedor exclusivo de los servicios de OpenAI (Azure OpenAI Service), lo que le da una ventaja estratégica para aplicaciones de IA generativa. Como desventaja, sigue siendo de Microsoft.

- **¿Para qué la usaríamos?** Azure sería un contendiente principal para alojar toda la infraestructura de SaveApp. La elección sería especialmente estratégica si el equipo opta por Azure DevOps para la gestión del ciclo de vida del desarrollo, creando un flujo de trabajo altamente unificado. Además, podríamos utilizar el Azure OpenAI Service para acceder a los modelos de lenguaje de OpenAI (como GPT-4) de una manera segura y optimizada para empresas, potenciando nuestro sistema de extracción de datos de promociones.

MongoDB Atlas

- **¿Qué es?** MongoDB Atlas es la versión Database-as-a-Service (DBaaS) totalmente gestionada de MongoDB, disponible en AWS, GCP y Azure. Ofrece escalado automático, backups, alta disponibilidad y herramientas integradas (geoespacial, full-text, change streams, triggers y funciones) sin tener que administrar servidores.

- **Ventajas y Desventajas:** simplifica la operación (backups/monitoring/replicas), índice geoespacial nativo para consultas por cercanía, change streams para reaccionar a cambios en tiempo real (notificaciones, pipelines), multi-cloud (reduce lock-in al permitir despliegue en distintos proveedores), cifrado y controles de acceso empresariales, y capa gratuita para MVP. Como desventaja puede tener costo creciente a gran escala, y cierto lock-in funcional si se adoptan features propietarias (Atlas Search, Triggers/Functions). Además, posibles costos de egreso de red si se combina con servicios en otro proveedor.

- **¿Para qué la usaríamos?** En MongoDB Atlas almacenaríamos usuarios, tarjetas, promociones, comercios y eventos de uso; habilitaríamos consultas geoespaciales por proximidad mediante índices 2dsphere para resolver “beneficios cerca mío”;

aprovecharíamos change streams para disparar actualizaciones del feed y las recomendaciones cuando el Crawler ingrese o expire promociones; utilizaríamos triggers para normalización liviana o para enviar webhooks (por ejemplo, invalidar caché); opcionalmente implementaríamos Atlas Search para la búsqueda rápida de comercios y marcas; y, cuando sea necesario reducir latencia y aumentar resiliencia, desplegaríamos en configuración multi-región.

Terraform (Infrastructure as Code)

- **¿Qué es?** Terraform es una herramienta de Infraestructura como Código (IaC) que permite definir la infraestructura de forma declarativa y reproducirla de manera consistente mediante los comandos plan y apply. Utiliza providers (GCP, AWS, Kubernetes, Vercel, etc.) y mantiene un estado (state) para identificar lo existente y determinar los cambios necesarios.

- **Ventajas y Desventajas:** En cuanto a ventajas, ofrece reproducibilidad, versionado y revisión por PR, modularidad mediante módulos reutilizables, soporte multi-cloud, detección de drift, etiquetado para control de costos y entornos aislados (dev/stg/prod). Por el lado de las desventajas, exige gestionar el *remote state* y los bloqueos, tiene una curva de aprendizaje considerable, arrastra particularidades según cada *provider* y requiere un manejo cuidadoso de secretos y permisos en CI.

- **¿Para qué la usaríamos?** La utilizaríamos para evitar configurar manualmente la infraestructura en la nube al realizar cambios hacia otra cuenta. Además, facilita el despliegue del flujo de trabajo para la obtención de ofertas, ya que el sistema de crawlers es dinámico y evoluciona con el tiempo.

Almacenamiento de Archivos

Para archivos no estructurados como imágenes, logos o documentos, se utilizan sistemas de almacenamiento de objetos, diseñados para la durabilidad, disponibilidad y escalabilidad masiva.

Amazon S3 (Simple Storage Service)

- **¿Qué es?** Es el servicio de almacenamiento de objetos de AWS y el estándar de facto de la industria. Permite almacenar y recuperar cualquier cantidad de datos desde cualquier lugar.

- **Ventajas y Desventajas** Es increíblemente duradero, escalable hasta el infinito y muy rentable para el almacenamiento. Se integra perfectamente con el vasto ecosistema de AWS. Como desventaja, la configuración de permisos puede ser compleja y los costos por transferencia de datos (egress) deben ser considerados.

- **¿Para qué la usaríamos?** Sería el repositorio principal para todos los archivos binarios de SaveApp. Aquí guardaríamos los logos de los comercios, las imágenes de las promociones y los avatares de los usuarios. En nuestra base de datos principal solo guardaríamos la URL que apunta al archivo en S3.

Google Cloud Storage

- **¿Qué es?** Es la solución de almacenamiento de objetos de Google Cloud Platform, un competidor directo de Amazon S3 que ofrece funcionalidades y niveles de servicio equivalentes.

- **Ventajas y Desventajas** Se integra perfectamente con el ecosistema de GCP y es conocido por su alto rendimiento y una estructura de precios a menudo considerada más simple. Su ecosistema de herramientas de terceros es amplio, aunque no tan vasto como el de S3.

- **¿Para qué la usaríamos?** Cumpliría exactamente el mismo rol que Amazon S3. La elección entre ambos dependería del proveedor de nube principal que elijamos para el resto de nuestra infraestructura (AWS o GCP), para maximizar la sinergia y minimizar la latencia y los costos de transferencia de datos.

Contenerización y Orquestación

Esta sección se enfoca en las tecnologías para empaquetar y gestionar nuestra aplicación como unidades estandarizadas, lo que garantiza la consistencia y facilita la escalabilidad.

Contenerización (Docker)

- **¿Qué es?** Docker es una plataforma que permite empaquetar una aplicación y todas sus dependencias en una unidad estandarizada y portable llamada "contenedor", garantizando que se ejecute de la misma manera en cualquier entorno.

- **Ventajas y Desventajas** Su principal ventaja es la consistencia y portabilidad del software, eliminando el problema de "en mi máquina funciona". Simplifica enormemente el despliegue y la creación de entornos de desarrollo locales idénticos a producción.

- **¿Para qué la usaríamos?** Usaríamos Docker para empaquetar cada uno de los microservicios de SaveApp (la API, los scrapers, el procesador de NLP). Esto nos permitiría gestionar, desplegar y escalar cada pieza del sistema de forma aislada y fiable.

Orquestación de Contenedores (Kubernetes)

- **¿Qué es?** Kubernetes (K8s) es un sistema de orquestación de código abierto que automatiza el despliegue, el escalado y la gestión de aplicaciones en contenedores a gran escala.

- **Ventajas y Desventajas** Es el estándar de facto para aplicaciones de alta disponibilidad, ofreciendo auto-reparación y escalado automático. Sin embargo, su curva de aprendizaje es muy pronunciada y su gestión añade una sobrecarga operativa significativa.

- **¿Para qué la usaríamos?** Su adopción sería una consideración estratégica para una fase de crecimiento, cuando la gestión manual de los contenedores se vuelva inviable.

7. CI/CD

La Integración y Entrega Continua (CI/CD) es una filosofía y un conjunto de herramientas que automatizan el ciclo de vida del desarrollo. Permiten compilar, probar y desplegar el código de forma automática, aumentando la velocidad y reduciendo los errores.

GitHub Actions

- **¿Qué es?** Es la plataforma de CI/CD nativa e integrada directamente en GitHub. Permite automatizar flujos de trabajo (workflows) en respuesta a eventos del repositorio, como un push a una rama o la creación de un pull request.

- **Ventajas y Desventajas** Su ventaja principal es la integración perfecta con el código fuente, sin necesidad de una herramienta externa. Ofrece un generoso nivel gratuito y un enorme marketplace de acciones pre-construidas. Su desventaja es que está intrínsecamente ligado a GitHub y que el contexto es propio del repositorio por lo que no permite pipelines complejos que integren distintos repositorios.

- **¿Para qué la usaríamos?** Si el código de SaveApp se aloja en GitHub, esta sería nuestra opción por defecto. La usaríamos para ejecutar pruebas automáticamente en cada pull request y para desplegar el backend y el frontend a sus respectivas plataformas cada vez que se fusione código a la rama principal.

GitLab CI/CD

- **¿Qué es?** Es la solución de CI/CD integrada en la plataforma GitLab. Es conocida por ser una herramienta "todo en uno" que abarca el ciclo de vida completo de DevOps, desde la gestión del código hasta el monitoreo.

- **Ventajas y Desventajas** La ventaja es tener una única plataforma unificada para todo el proceso, con una sintaxis de pipeline muy potente y configurable. Permite el uso de "runners" auto-hospedados para un mayor control. Su principal desventaja es que tiene más sentido si todo el proyecto ya reside en el ecosistema de GitLab, además de un contexto propio del repositorio al igual que GitHub Actions.

- **¿Para qué la usaríamos?** La elegiríamos si el equipo decidiera usar GitLab como plataforma de gestión de código. Nos permitiría definir pipelines complejos para construir, probar y desplegar todos los componentes de SaveApp desde un único lugar.

Azure DevOps

- **¿Qué es?** Azure DevOps es una suite de servicios integral de Microsoft que abarca todo el ciclo de vida del desarrollo de software. No es solo una herramienta de CI/CD, sino una plataforma unificada que incluye Azure Boards para la planificación ágil (similar a Jira), Azure Repos para el control de código fuente (Git), Azure Pipelines para la automatización de compilación y despliegue (CI/CD), y Azure Artifacts para la gestión de paquetes.

- **Ventajas y Desventajas:** Su principal ventaja es ser una solución "todo en uno" fuertemente integrada, lo que simplifica enormemente la cadena de herramientas al no tener que gestionar diferentes servicios para planificación, código y despliegue. Su interfaz de usuario es moderna e intuitiva, y su integración con el ecosistema de la nube de Azure es nativa y potente, aunque también soporta despliegues a otras nubes como AWS o GCP. Su principal desventaja es ser de Microsoft.

- **¿Para qué la usaríamos?** Utilizaríamos Azure Pipelines para crear flujos de CI/CD que compilen, prueben y desplieguen automáticamente cada componente de la aplicación en los diferentes entornos.

PROPUESTA DE SOLUCIÓN

Introducción general

La solución propuesta para SaveApp busca ofrecer una herramienta práctica que ayude a los usuarios a aprovechar al máximo los descuentos y beneficios de sus tarjetas de crédito y débito. La aplicación centraliza la información pública de promociones bancarias y comerciales, la organiza de forma clara y la combina con funciones de inteligencia artificial y geolocalización para brindar recomendaciones útiles en el momento de compra. Más que una app de recopilación de descuentos, SaveApp se orienta a simplificar la toma de decisiones cotidianas sobre qué tarjeta conviene usar en cada comercio, aportando comodidad y ahorro al usuario.

Alcance funcional

La solución contempla una amplia gama de funcionalidades diseñadas para cubrir todas las etapas del descubrimiento, comprensión y uso de beneficios bancarios. El sistema permite a los usuarios registrar sus tarjetas de forma segura, sin incluir información sensible, y acceder a beneficios personalizados en función de su perfil, ubicación y hábitos de compra. Se incluye un buscador avanzado con filtros detallados, un asistente inteligente basado en IA y un sistema de seguimiento de reintegros para asegurar que los descuentos prometidos sean realmente acreditados.

Además, el enfoque multilenguaje garantiza accesibilidad desde el inicio, contemplando la futura expansión regional del producto.

Historias de Usuario

- **Registro con Google:** Como usuario, quiero poder registrarme e iniciar sesión utilizando mi cuenta de Google, para simplificar el proceso de acceso y evitar tener que recordar contraseñas adicionales.
- **Favoritos de marcas:** Como usuario, quiero poder marcar mis marcas favoritas, visualizarlas en una sección dedicada y eliminarlas cuando desee, para acceder fácilmente a las ofertas que más me interesan.
- **Cambio de idioma:** Como usuario, quiero poder cambiar el idioma de la aplicación entre español e inglés, para utilizarla en mi idioma preferido.
- **Reporte de errores:** Como usuario, quiero poder reportar errores o problemas que encuentre en la aplicación, para contribuir a su mejora continua.
- **Donaciones a fundaciones:** Como usuario, quiero poder donar parte del dinero que ahorré mediante la app a una fundación benéfica, para colaborar con causas sociales de manera sencilla.

- **Guardar ofertas:** Como usuario, quiero poder guardar ofertas para revisarlas más tarde y visualizarlas en una sección de ofertas guardadas, para poder aprovecharlas en otro momento.

- **Reportar ofertas:** Como usuario, quiero poder reportar una oferta (por ejemplo, si está vencida o no corresponde), para mantener la información de la aplicación actualizada y confiable.

- **Mapa de comercios adheridos:** Como usuario, quiero poder visualizar los comercios adheridos en un mapa interactivo, para ubicar fácilmente dónde puedo aprovechar los descuentos.

- **Métodos de pago válidos:** Como usuario, quiero poder ver los métodos de pago válidos para cada oferta (crédito, débito, QR, etc.), para saber con qué medio puedo aplicarla.

- **Información destacada por tipo de oferta:** Como usuario, quiero que la app me muestre la información más relevante según el tipo de oferta (por ejemplo, cuotas sin interés, reintegros o descuentos directos), para entender rápidamente cómo aprovecharla.

- **Acceso al enlace original de la oferta:** Como usuario, quiero poder acceder mediante un botón directo al enlace original de la oferta en el sitio del banco o comercio, para consultar más detalles si lo necesito.

- **Preguntas predefinidas en el asistente virtual:** Como usuario, quiero poder seleccionar desde la pantalla del asistente virtual una lista de preguntas sugeridas o ejemplos, para facilitar la interacción sin tener que escribir manualmente.

- **Visualización de términos y condiciones originales:** Como usuario, quiero poder acceder al documento original de los términos y condiciones de cada oferta, para leer la información completa cuando lo considere necesario.

- **Visión general de métricas:** Como administrador, quiero visualizar en el dashboard un resumen con métricas clave (cantidad de usuarios, tarjetas totales, ofertas activas e históricas), para obtener una panorámica rápida del estado del sistema.

- **CRUD de entidades principales:** Como administrador, quiero crear, leer, actualizar y eliminar registros de entidades del sistema, para mantener el catálogo y los datos operativos consistentes desde una sola interfaz.

- **Filtrado de Vistas:** Como administrador, quiero poder filtrar y buscar la visualización de los registros en los listados de entidades, para adaptar la información a mis necesidades y facilitar la gestión de los datos.

- **Métricas por entidad:** Como administrador, quiero ver desgloses de métricas por banco y por marca (cantidad de tarjetas por banco, cantidad de ofertas por banco, cantidad de ofertas por marca), para identificar tendencias y concentraciones.

- **Métricas por usuario:** Como administrador, quiero consultar indicadores por usuario (cantidad de seguimientos por usuario, cantidad de tarjetas por usuario, cantidad de usuarios por banco), para detectar patrones de uso y segmentaciones.

- **Métricas por tarjeta/oferta:** Como administrador, quiero acceder a métricas específicas (cantidad de ofertas por tarjeta), para evaluar cobertura y relevancia de los medios de pago soportados.

Roles involucrados

- **Usuario**: Persona que se registra en la app, configura sus tarjetas y métodos de pago, consulta descuentos y recibe notificaciones. Es el actor principal del sistema.

- **Administrador de la Aplicación**: Encargado del mantenimiento técnico y supervisión del sistema. Asegura la actualización correcta de los datos extraídos, la seguridad y privacidad de la información y la operatividad general.

- **Agentes de IA**: Actores lógicos del sistema que se encargan del análisis semántico de las promociones y el procesamiento de las mismas.

- **Asistente Virtual o Chatbot**: Agente conversacional automatizado que asesora al usuario en tiempo real sobre qué tarjeta o método de pago utilizar, en base a sus hábitos y perfil.

Diseño del sistema

Arquitectura

SaveApp adopta una arquitectura cliente–servidor con multirepo para separar responsabilidades, versionar y desplegar cada pieza en forma independiente. La app móvil en Flutter interactúa exclusivamente con una API GraphQL servida por un backend en Node.js. Todas las lecturas y escrituras de la app pasan por esta API, que aplica reglas de negocio, autorización por resolver y composición de datos antes de persistir o consultar en la base de datos MongoDB.

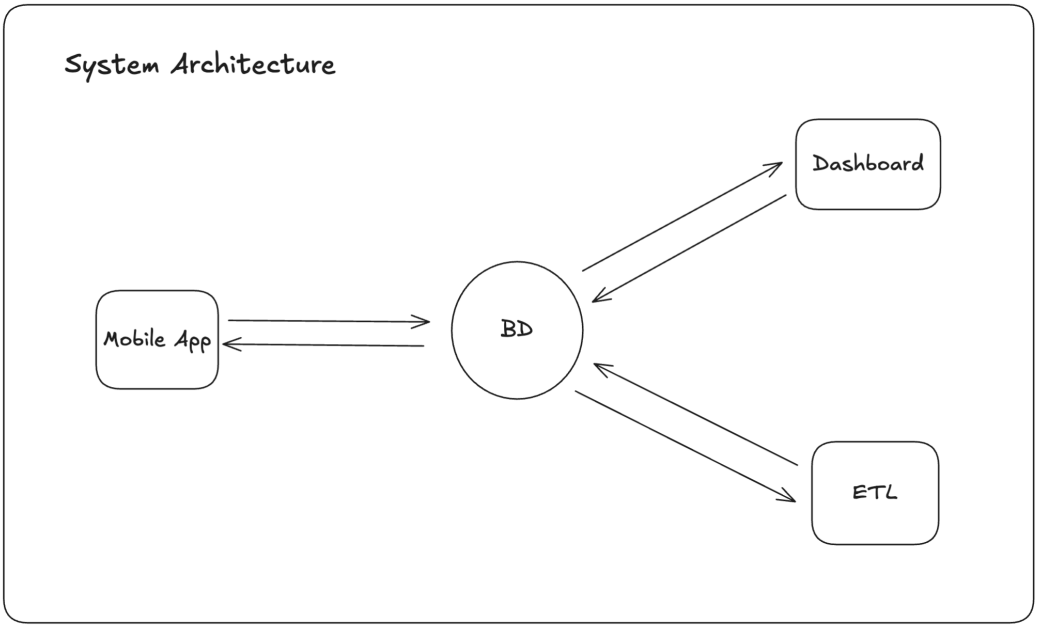
En paralelo, un pipeline ETL opera de forma asíncrona y desacoplada: los crawlers extraen información pública de bancos/billeteras, la etapa de transformación la normaliza y valida, y la capa de carga publica los registros ya curados en Mongo mediante upserts idempotentes. La app no se comunica jamás con el ETL, consume únicamente datos consolidados por la API asegurando tiempos de respuesta predecibles y una consistencia eventual del catálogo frente a las actualizaciones continuas.

Para operación y monitoreo, un dashboard interno consulta directamente la base (o vía API cuando aplica) para revisar estados del ETL, auditorías de cambios y salud del sistema. Este diseño separa claramente el tráfico online (app ↔ API ↔ base) de las actualizaciones

batch/near-real-time (ETL ↔ base), minimiza acoplamientos y facilita el escalado independiente de lectura, escritura y procesamiento. [1]

Vista general

- **App móvil:**
    - **Frontend (Flutter):** recibe la información desde el backend a traves de la API GraphQL/HTTPS con autenticación por token, además, cachea vistas y aplica paginación/infinite scroll para listas largas de ofertas.
    - **Backend API (Node.js + GraphQL):** orquesta la lógica de negocio, aplica autorización a nivel de resolver, compone agregados y expone schemas tipados.
- **Base de datos (MongoDB):** fuente de verdad para usuarios, tarjetas, ofertas, marcas y ubicaciones. Usa índices por vigencia/categoría y garantiza upserts idempotentes y mantiene versionado/auditoría de cambios relevantes.
- **Pipeline ETL (Extracción → Transformación → Carga):** recolecta, interpreta e inserta ofertas normalizadas sin interferir con el tráfico online; controla reintentos, deduplicación y trazabilidad por fuente/fecha.
- **Dashboard operativo:** interfaz para equipo interno que consulta estados del ETL, verifica integridad de datos, revisa alertas y ejecuta tareas administrativas con permisos restringidos.

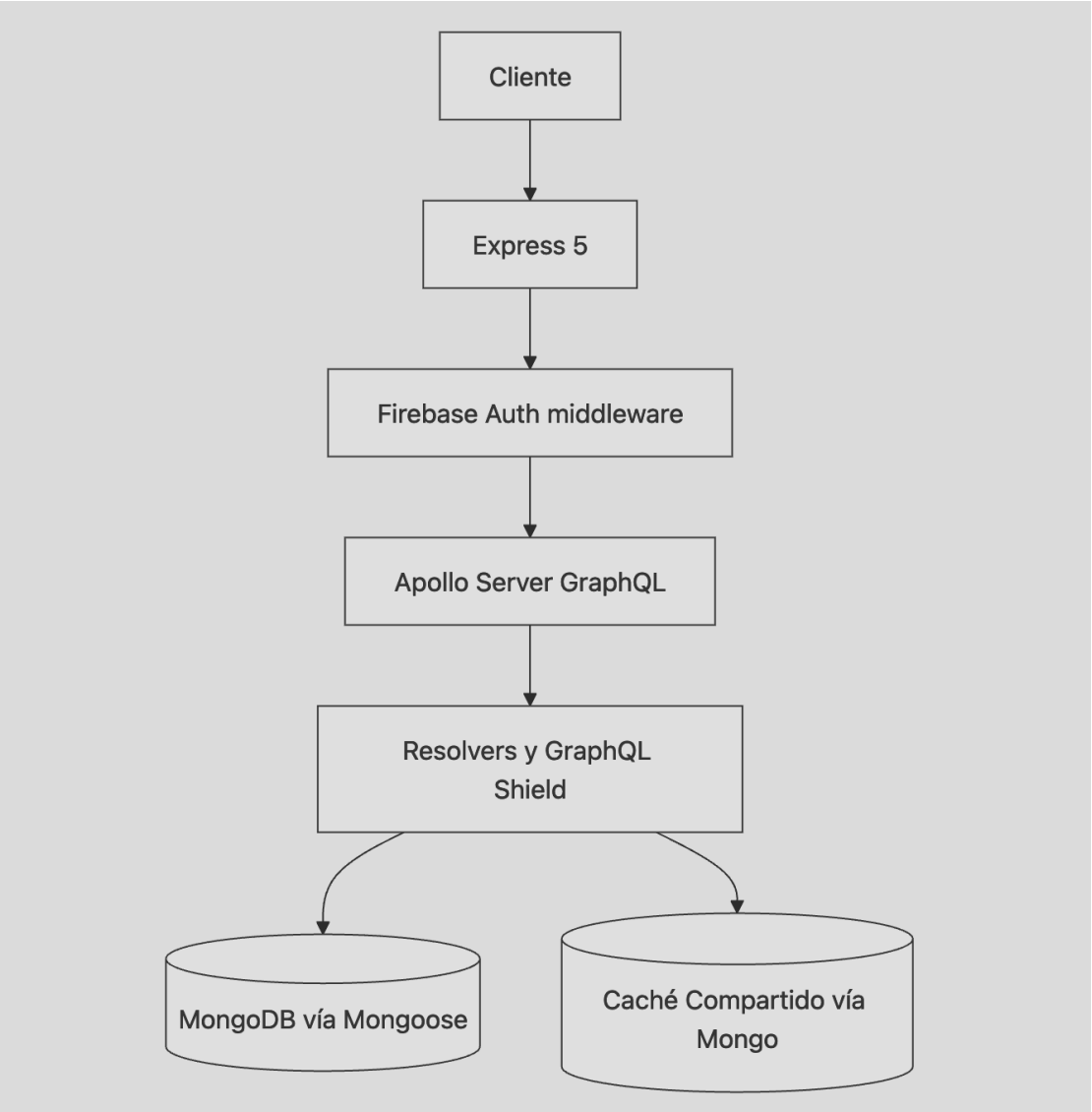


Multirepo y responsabilidades

SaveApp-Backend:

Lógica de negocio, autorización a nivel resolver, composición de agregados y validaciones. Expone schemas tipados y endpoints de administración restringidos. Es una API GraphQL construida con Express 5, Apollo Server y Type-GraphQL que implementa la lógica de negocio central de SaveApp.

El backend sigue una arquitectura en capas que procesa las peticiones a través del siguiente flujo:



El backend posee las siguientes responsabilidades por Capa:

- **Capa de Servidor HTTP:** Express 5 maneja las conexiones HTTP y CORS.
- **Capa de Middleware:** Incluye autenticación Firebase (firebaseAuth) y limitación de tasa (rateLimiter).

- **Capa GraphQL:** Apollo Server ejecuta queries y mutations, con Type-GraphQL generando el schema desde clases TypeScript. index.ts:5-9 GraphQL Shield aplica autorización a nivel resolver mediante reglas declarativas de permisos.
- **Capa de Resolvers:** Implementa la lógica de negocio y composición de agregados. index.ts:12 Utiliza modelos compartidos del paquete @saveapp-org/shared.
- **Capa de Acceso a Datos:** Mongoose ODM gestiona las operaciones con MongoDB.
- **Sistema de Caché:** Implementa una estrategia de dos niveles (L1 en memoria, L2 en MongoDB) para optimizar respuestas de lectura intensiva. cache.ts:2-3 El sistema se conecta a dos instancias MongoDB separadas: MONGO_URI para datos persistentes y CACHE_URI para caché.

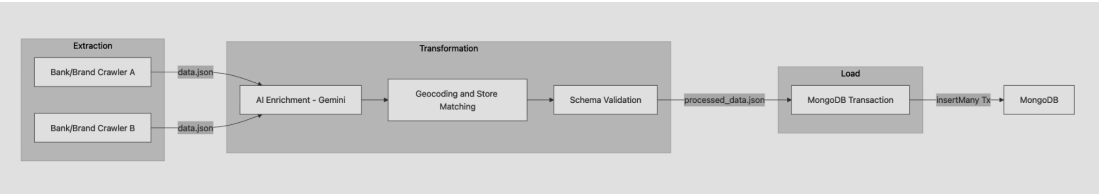
El sistema implementa autorización a nivel resolver mediante GraphQL Shield con el Principio de Menor Privilegio (todas las operaciones denegadas por defecto). Las validaciones se realizan mediante class-validator en los inputs de GraphQL.

El backend utiliza Firebase Admin SDK para autenticación y está diseñado para despliegue en contenedores con soporte para Google Cloud Run. El decorador @Cache del paquete compartido permite aplicar políticas de caché TTL a nivel de resolver.

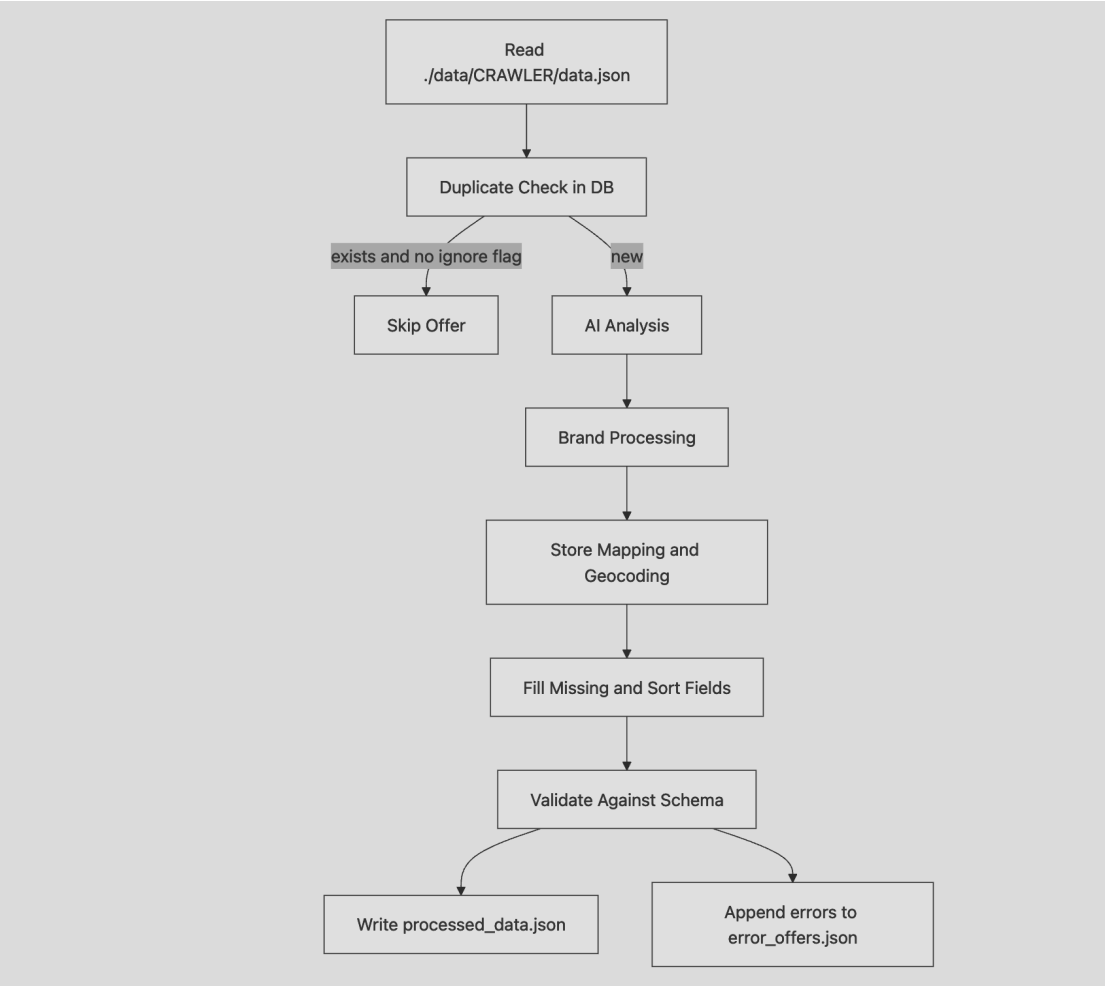
SaveApp-Crawlers

Este repositorio que contiene los distintos crawlers responsables de la extracción periódica desde fuentes públicas (bancos y billeteras). Cada crawler implementa su propia lógica de parsing y normalización de datos, generando registros con su respectivo **recordId**, utilizado como clave de referencia externa para detectar si un elemento ya existe en la base antes de intentar insertarlo. De esta manera, se evita la duplicación de información y se mantiene la integridad del catálogo. Además, en este repositorio se encuentra la capa de transformación y carga, que estandariza los datos extraídos y realiza su inserción/actualización en la base de datos.

Implementa un pipeline ETL con tres capas independientes que se comunican mediante archivos JSON



El procesamiento interno de la capa de transformación sigue estos pasos:

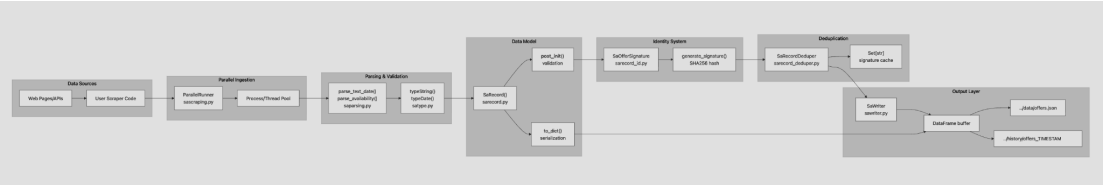


El sistema previene duplicados verificando la existencia de ofertas antes del procesamiento usando recordId como clave de referencia externa. Procesa ofertas en lotes configurables (por defecto 100) con caché y reintentos. El enriquecimiento con IA usa Google Gemini para categorización, descripción y extracción de campos estructurados. Las transacciones requieren MongoDB en modo replica set para garantizar atomicidad . Incluye wrappers listos para GCP Pub/Sub y Cloud Run Jobs.

SaveApp-SaCrawl

Librería compartida utilizada por todos los crawlers para estandarizar su ciclo de vida. Proporciona utilidades comunes como manejo de sesiones, parsers, normalización de campos (fechas, monedas, porcentajes), validación previa al upsert, logging estructurado y métricas. Además, centraliza la definición del uso de **recordid** dentro del flujo de carga para asegurar que las inserciones sean idempotentes.

El siguiente diagrama ilustra el flujo completo de procesamiento de datos en SaveApp-SaCrawl, desde las fuentes de datos hasta la persistencia final:



El proceso comienza con las Fuentes de Datos (Data Sources), donde el código del usuario interactúa con páginas web y APIs para extraer datos sin procesar. Estos datos pasan luego a la capa de Ingesta Paralela (Parallel Ingestion), donde ParallelRunner ejecuta el código de scraping de forma concurrente usando pools de procesos o hilos, soportando modos de salida como lista o generador.

Una vez ingresados los datos, entran en la fase de Parsing y Validación (Parsing & Validation). Las funciones de saparsing.py como parse_text_date() y parse_availability() procesan datos textuales, mientras que las funciones de satype.py como typeString(), typeDate(), typeInt() y typeFloat() validan y convierten tipos.

El corazón del sistema es el Modelo de Datos (Data Model), representado por SaRecord, el dataclass central que representa ofertas scrapeadas. Su método __post_init__() ejecuta validación automática de campos, y to_dict() serializa el objeto para persistencia.

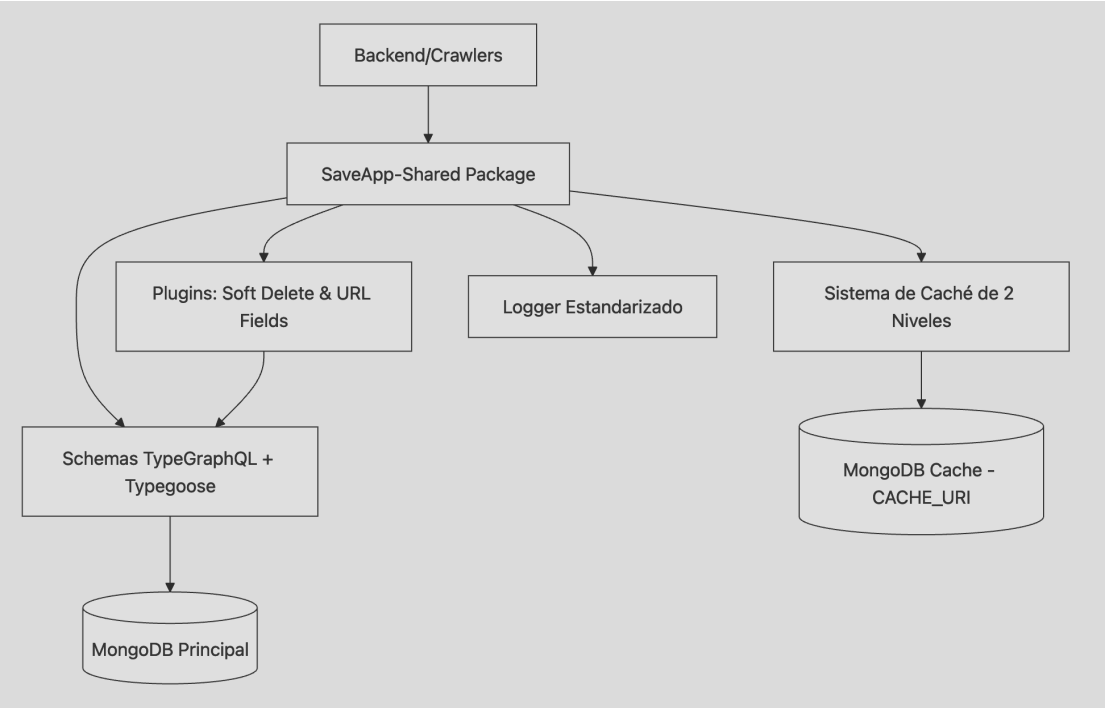
Para garantizar la unicidad de los registros, el Sistema de Identidad (Identity System) utiliza SaOfferSignature para generar hashes SHA256 desde campos seleccionados del registro. Este sistema siempre incluye campos por defecto: bank, domain, country y offerUrl.

La Deduplicación (Deduplication) es manejada por SaRecordDeduper, que mantiene un caché en memoria de firmas ya vistas. Su método dedup() retorna una tupla (is_duplicate: bool, signature: str) para cada registro procesado.

Finalmente, la Capa de Salida (Output Layer) utiliza SaWriter, un context manager que bufferiza registros en un DataFrame. Al salir del contexto, escribe automáticamente a dos ubicaciones: archivo actual (../data/) y archivo histórico con timestamp (../history/), soportando formatos JSON, CSV y XLSX.

SaveApp-Shared (Node.js package)

SaveApp-Shared es un paquete Node.js interno que centraliza contratos compartidos, modelos de datos, plugins y utilidades para los servicios de SaveApp (Backend y Crawlers).



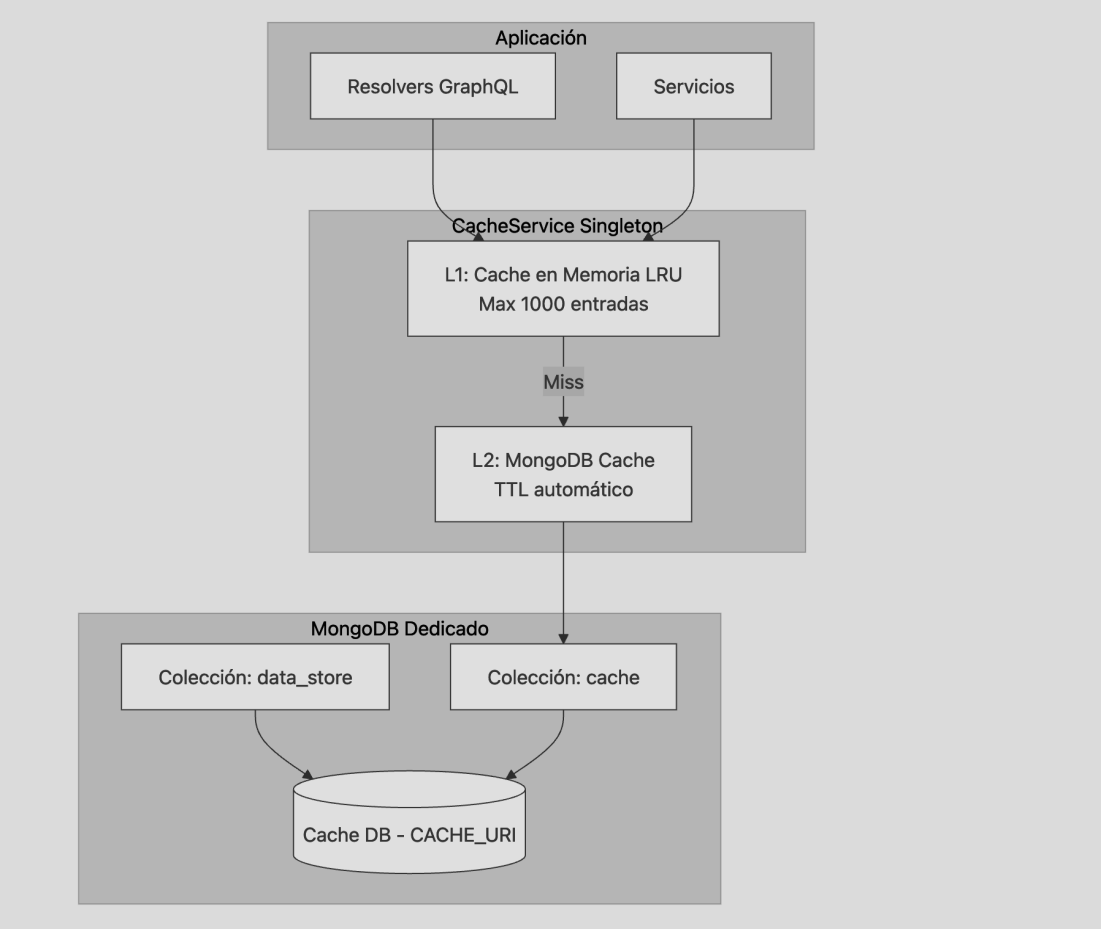
El núcleo del paquete consiste en 9 schemas principales que actúan como contratos compartidos entre todos los servicios. Estos schemas incluyen entidades fundamentales como User para cuentas de usuario con integración Firebase, Offer para ofertas promocionales de tarjetas de crédito y entidades de soporte como Brand, Store, Card, Bank y Category, además de Comment y Tracking para las interacciones de usuario.

La arquitectura de schemas utiliza un patrón de decoradores duales que combina TypeGraphQL para definiciones de tipos GraphQL y Typegoose para estructuras de colecciones MongoDB. Esta dualidad permite que los mismos schemas sirvan tanto para APIs GraphQL como para operaciones de base de datos, eliminando la necesidad de mantener definiciones separadas. Los modelos Mongoose resultantes se exportan completamente configurados y listos para usar en cualquier servicio consumidor.

El paquete implementa un plugin sofisticado de soft delete que sobreescribe las operaciones básicas de Mongoose como delete, find y count con lógica personalizada de eliminación suave. Este plugin añade automáticamente tres campos a los schemas: deleted (booleano), deletedAt (fecha opcional) y deletedBy (referencia opcional al usuario que eliminó el registro). El plugin también extiende los modelos con métodos de instancia como delete() y restore(), métodos estáticos como deleteById(), restoreById() y deleteManyByIds(), y query helpers como withDeleted() y onlyDeleted() que permiten controlar la visibilidad de registros eliminados en las consultas. Este sistema está aplicado a 8 de los 9 schemas principales, siendo Tracking la única excepción.

Cuenta con un sistema de caché de dos niveles con una capa L1 en memoria y una capa L2 persistente en MongoDB. La capa L1 utiliza una estrategia LRU (Least Recently Used) con un máximo de 1000 entradas, mientras que la capa L2

proporciona persistencia con TTL automático mediante un índice expiresAt en MongoDB.



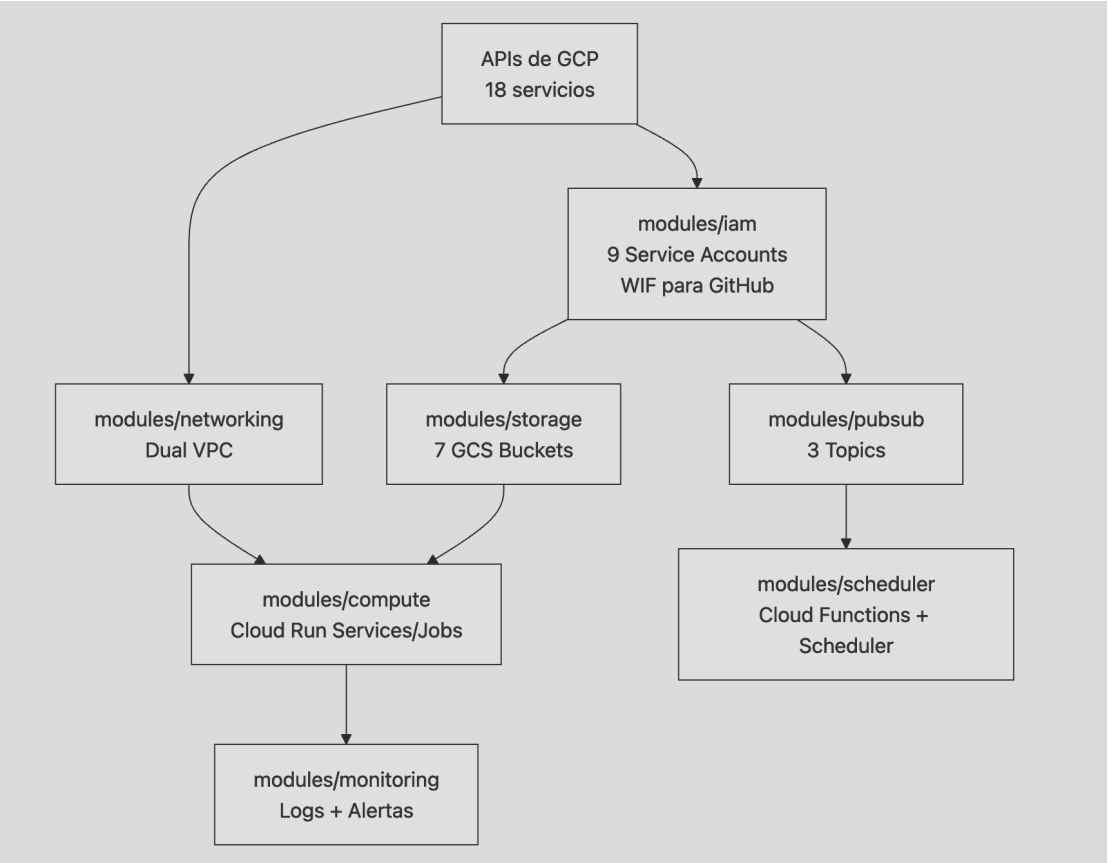
El sistema incluye un decorador @Cache que permite aplicar caché automático a nivel de método con deduplicación de requests concurrentes, cache.ts:377-434 y una funcionalidad de DataStore persistente para almacenar datos que deben persistir indefinidamente sin TTL. Una característica importante es que el sistema utiliza una conexión MongoDB completamente separada mediante la variable de entorno CACHE_URI, lo que permite aislar fallos y escalar el sistema de caché independientemente de la base de datos principal. mongo.ts:4-28

Además, el paquete proporciona un sistema de logging estandarizado basado en chalk que ofrece funciones con colores distintivos para diferentes niveles de log: logInfo, logWarn, logError, logSuccess, logDebug y logStatus, facilitando el debugging visual en todos los servicios. Las exportaciones están organizadas mediante un mapa modular que permite importaciones específicas desde diferentes puntos de entrada como @saveapp-org/shared/schemas para schemas, @saveapp-org/shared/models para modelos, @saveapp-org/shared/cache para el sistema de caché, y @saveapp-org/shared/logger para utilidades de logging. package.json:29-69

SaveApp-Infrastructure

SaveApp Infrastructure es un sistema de infraestructura como código en Google Cloud Platform que gestiona dos sistemas separados mediante una arquitectura de VPC dual. El sistema principal incluye servicios de aplicación (saveapp-backend, saveapp-dashboard-backend, saveapp-dashboard-frontend) desplegados en Cloud Run, mientras que el sistema ETL opera en una VPC completamente aislada para procesar datos.

El pipeline ETL sigue una arquitectura event-driven que se ejecuta en tres etapas coordinadas mediante Pub/Sub topics. Cloud Scheduler activa la función pipeline-orchestrator diariamente a las 2 AM (zona horaria México), que ejecuta los cuatro crawler jobs en paralelo. Cada crawler extrae datos y los escribe al bucket saveapp-crawler-processing, archivando copias en saveapp-crawler-history. Al completar, publican mensajes "SUCCESS" al topic crawler-completed, disparando automáticamente la función transformation-trigger que ejecuta el job de transformación con APIs de Google Maps y Gemini AI. Finalmente, el job de carga persiste los datos procesados en MongoDB mediante inserciones por lotes de 100 registros.



La infraestructura está construida con módulos Terraform que siguen un orden de dependencias estricto. El módulo base habilita 18 APIs de GCP, seguido por los módulos de networking que crean las VPCs dual con sus conectores. El módulo IAM provisiona 9 service accounts con permisos de mínimo privilegio y configura Workload Identity Federation para autenticación sin claves desde GitHub Actions.

Los módulos de storage crean 7 buckets con políticas de lifecycle: el bucket de procesamiento tiene auto-eliminación a los 7 días, mientras que el de historial implementa transiciones automáticas entre clases de almacenamiento para optimizar costos. El módulo de monitoring configura log sinks y políticas de alertas que notifican por email cuando ocurren fallos en crawlers, transformación o mensajes llegan a dead letter queues.

Los operadores pueden ejecutar el pipeline manualmente usando el script run-etl-pipeline.sh, que obtiene la URI de la función orquestadora de los outputs de Terraform y realiza una petición POST autenticada al endpoint /orchestrate_pipeline. La infraestructura soporta múltiples ambientes (dev/prod) mediante backends de estado Terraform separados en environments/dev/backend.tf y environments/prod/backend.tf.

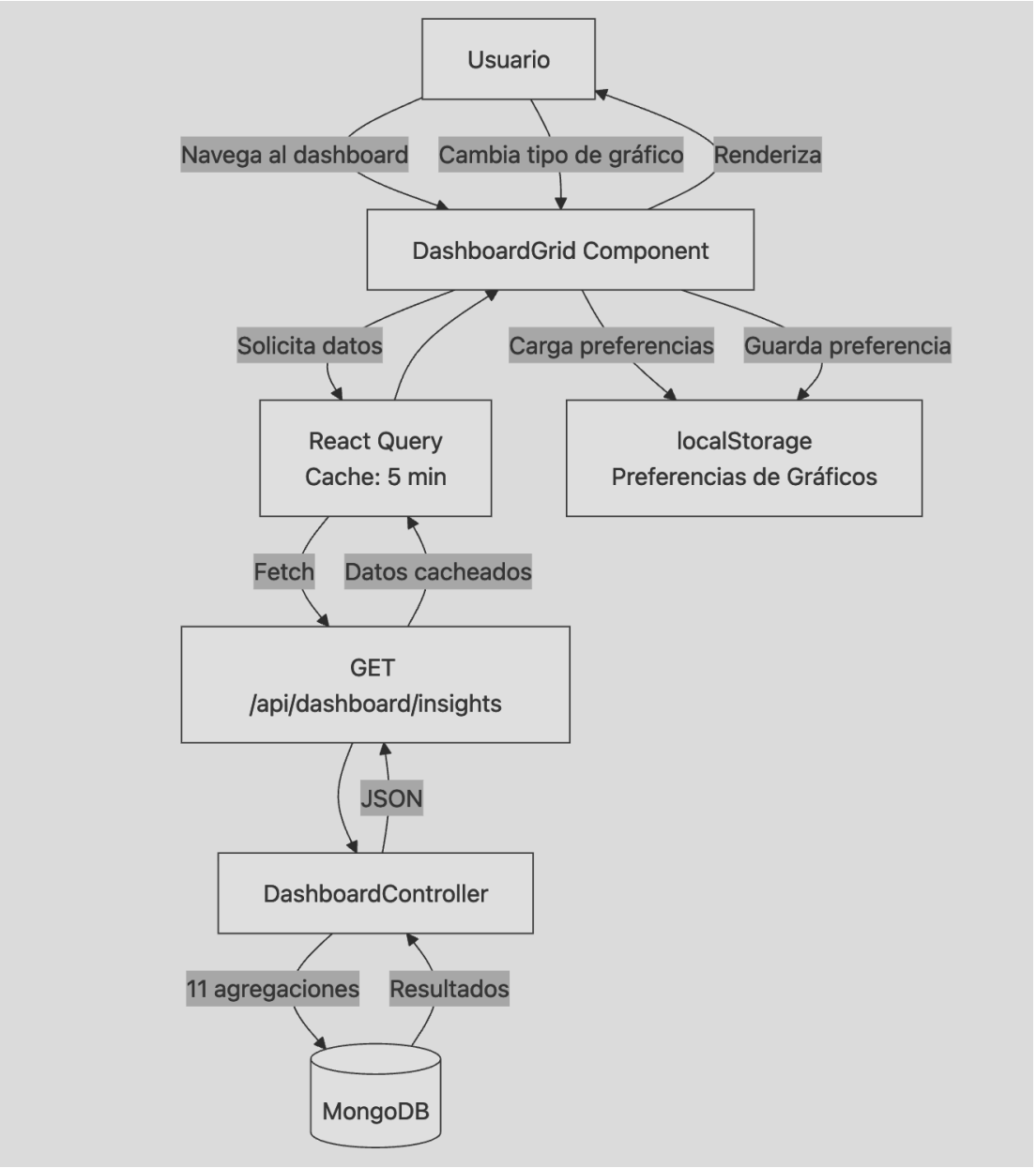
SaveApp-Dashboard

El dashboard de SaveApp muestra métricas sobre usuarios, bancos, tarjetas y ofertas, con visualizaciones interactivas configurables por el usuario. El sistema tiene tres componentes: el frontend DashboardGrid que renderiza la cuadrícula, el backend con endpoint /api/dashboard/insights que ejecuta agregaciones MongoDB, y un sistema de preferencias en localStorage.

Métricas disponibles:

- **Numéricas:** Total de Usuarios, Total de Tarjetas, Ofertas Activas, Total de Ofertas, Promedio de Ofertas Rastreadas.
- **Distribuciones:** Tarjetas por Banco, Ofertas por Banco/Marca/Tarjeta, Usuarios por Banco, Promedio de Tarjetas por Usuario.

Además, los usuarios pueden alternar entre tipos de gráficos (barras, líneas, pastel) para las distribuciones..



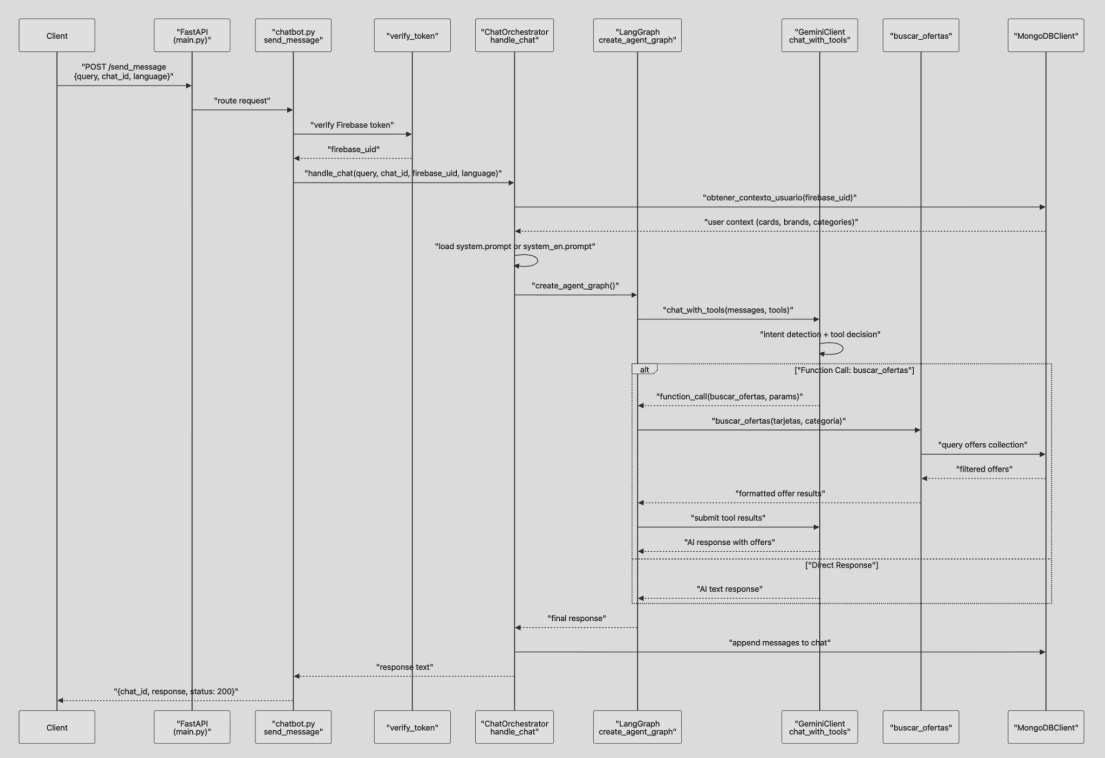
SaveApp-Chatbot

SaveApp-Chatbot es un backend desarrollado en Python 3.11 con FastAPI que funciona como asistente conversacional para la plataforma SaveApp. El sistema integra tres tecnologías principales: Google Gemini para procesamiento de lenguaje natural, MongoDB para persistencia de datos, y Firebase Authentication para gestión de usuarios.

La arquitectura del sistema se centra en un orquestador llamado ChatOrchestrator que coordina todas las operaciones. Este orquestador gestiona la comunicación entre el cliente de MongoDB (que almacena información de usuarios, tarjetas, ofertas, marcas y categorías), el cliente de Gemini (que procesa las consultas en

lenguaje natural), y un sistema de prompts que define el comportamiento del asistente tanto en español como en inglés.

El flujo de procesamiento comienza cuando un usuario envía un mensaje a través de los endpoints POST /new_chat o POST /send_message, ambos protegidos por autenticación Firebase mediante tokens Bearer. Una vez verificado el token, el sistema carga el prompt del sistema según el idioma configurado y construye un contexto personalizado consultando MongoDB para obtener las tarjetas registradas del usuario, sus marcas favoritas y categorías de interés. Este contexto se inyecta al inicio del historial de mensajes para que el modelo de IA tenga acceso a la información del usuario en cada interacción.



El sistema utiliza la capacidad de function calling de Google Gemini para ejecutar búsquedas de ofertas de manera inteligente. El modelo decide cuándo invocar la función buscar_ofertas basándose en la intención del usuario, interpretando frases como "salir a comer" como búsquedas en la categoría de restaurantes. El asistente está configurado para responder de forma clara y concisa en español, nunca inventar ofertas, y manejar conversaciones casuales sin ejecutar búsquedas innecesarias.

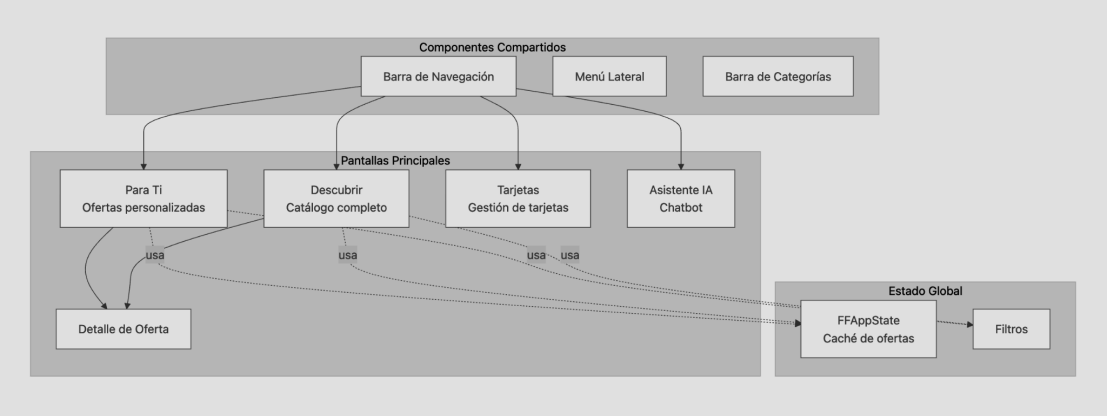
Un aspecto técnico importante es el sistema de normalización de categorías implementado en CATEGORY_MAP, que mapea variaciones de texto del usuario (como "supermercados", "supermercado", "mercado") a nombres canónicos en la base de datos ("Supermarkets"). Esta normalización incluye eliminación de diacríticos y conversión a minúsculas para garantizar coincidencias precisas.

SaveApp-Landing

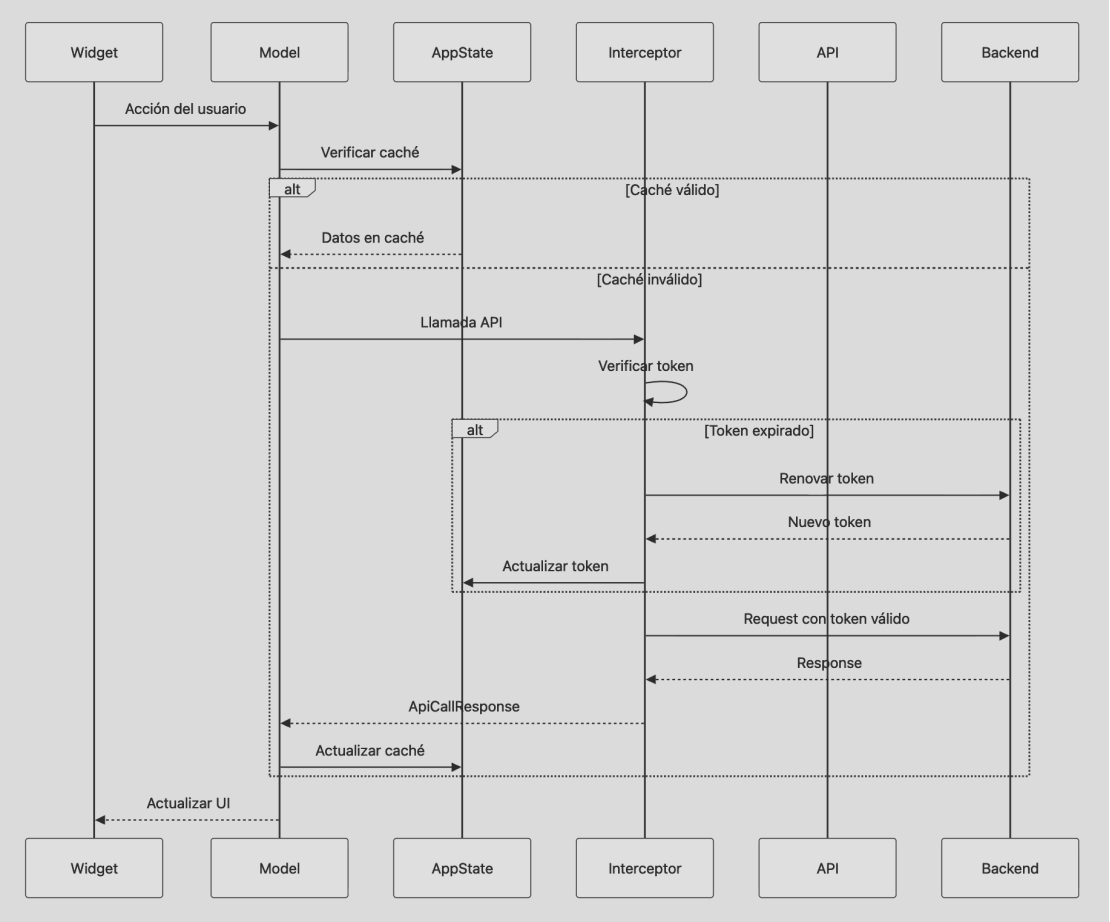
Es un sitio web estático publicado en saveapp.com.ar construido con Next.js 15 que sirve como página de aterrizaje para recopilar registros de acceso anticipado antes del lanzamiento de la beta privada de SaveApp. Se encuentra desplegado mediante GitHub Pages y Cloudflare como DNS/proxy para poder medir su tráfico de red.

SaveApp-Flutterflow

SaveApp es una aplicación Flutter que utiliza una arquitectura de tres capas con integración backend mediante GraphQL y REST APIs. El sistema se organiza en tres grupos de API principales: autenticación (SaveAppAPIAuthGroup), operaciones de datos GraphQL (SaveAppAPIGraphqlGroup), y chatbot con IA (SaveAppAPIChatbotGroup).



El sistema implementa un interceptor de tokens (TokenRefresher) que gestiona automáticamente la renovación de tokens JWT antes de cada llamada a las APIs protegidas. Este interceptor se adjunta tanto al grupo GraphQL como al de chatbot, garantizando que las sesiones de usuario permanezcan activas sin intervención manual.



El flujo de datos sigue un patrón de caché-first donde FFAppState mantiene estructuras de caché específicas (ForYouCache, DiscoverCache, myCardsCache, BanksCache) que minimizan las llamadas redundantes al backend. Cada llamada API incluye métodos de extracción tipados que parsean campos específicos del JSON de respuesta usando JSONPath.

La arquitectura GraphQL utiliza un formato de request body consistente con campos query y variables, permitiendo consultas parametrizadas y paginación. Las operaciones incluyen gestión de usuarios, tarjetas, ofertas, bancos, marcas, categorías, favoritos y seguimientos, todas accesibles mediante instancias estáticas de clases de llamada.

Pantallas principales [1]

- **Vista inicial (Initial Preview)**: Vista inicial que se muestra al iniciar la aplicación por primera vez que explica las funcionalidades de la aplicación y para qué se usan los permisos solicitados al usuario.

- **Inicio / Registro / Login**: Interfaces simples y limpias con validaciones de campos esenciales. Incluyen soporte para recuperación de contraseña y configuración inicial.
- **“Para ti” (For you)**: Vista personalizada que muestra promociones relevantes según ubicación actual, tarjetas cargadas y preferencias anteriores.
- **“Explorar” (Discover)**: Vista que permite buscar todas las promociones activas. Incluye filtros por banco, tipo de tarjeta, ubicación, día de la semana, tipo de beneficio (descuento, reintegro, cuotas o promoción), rubro comercial, monto mínimo y medio de pago.
- **“Mis tarjetas” (My Cards)**: Representación visual estilo billetera donde el usuario puede ver, agregar, editar o eliminar sus tarjetas. Se muestran con sus respectivos logos de banco y marca.
- **“Ofertas guardadas” (Saved Offers)**: Vista donde el usuario puede consultar las ofertas que ha marcado previamente para revisar o utilizar más adelante. Estas ofertas se organizan cronológicamente o por rubro, y permiten realizar acciones como eliminar, compartir o ver en detalle.
- **“Oferta” (detalle de promoción)**: Pantalla que presenta toda la información estructurada de una oferta específica, procesada previamente por IA. Incluye: tope de descuento o reintegro, días de la semana donde aplica, período de validez, tarjetas compatibles, medio y canal de aplicación (presencial o web), tiempo estimado de reintegro y listado de comercios adheridos. También puede mostrar alertas importantes extraídas del análisis semántico del texto original.
- **“Marcas favoritas”**: Vista donde el usuario puede seleccionar sus marcas preferidas de una lista completa o buscador. Las marcas favoritas se utilizan para priorizar recomendaciones, enviar alertas y personalizar los resultados en “Para ti” o “Explorar”.
- **“Asistente”**: Chatbot embebido que responde en lenguaje natural. El usuario puede consultar por promociones, reintegros, consejos financieros, y recibir respuestas personalizadas.

Pantallas secundarias [1]

- **Menú lateral (Drawer):** Vista lateral accesible desde cualquier pantalla principal. Permite al usuario gestionar su cuenta y acceder a funciones complementarias de la aplicación. Incluye opciones como: cambio de idioma (Español/Inglés), visualización de favoritos, ofertas guardadas y seguimientos activos. También integra botones para reportar errores, realizar donaciones a fundaciones asociadas y alternar entre modo claro y oscuro.
- **Perfil:** Pantalla que muestra la información del usuario autenticado (nombre, correo electrónico y foto o inicial). Desde esta vista, el usuario puede cerrar sesión o actualizar su información personal.

- **Favoritos:** Vista dedicada a las marcas favoritas del usuario. Permite visualizar los comercios o entidades previamente marcados como favoritos, eliminarlos o acceder rápidamente a sus promociones vigentes. Esta sección ayuda a personalizar los resultados de las pantallas “Para ti” y “Explorar”, priorizando los beneficios más relevantes.

- **Ofertas guardadas:** Pantalla donde se muestran las ofertas que el usuario decidió conservar para utilizarlas más adelante. Cada oferta incluye información clave como el comercio, tipo de beneficio, días de aplicación y entidad bancaria. Desde aquí se pueden eliminar las ofertas, compartirlas o acceder al detalle completo.

- **Seguimientos:** Vista que centraliza el monitoreo de los reintegros activos, mostrando para cada transacción la fecha de inicio, la fecha estimada de finalización y el estado del beneficio. Permite eliminar registros manualmente o recibir alertas automáticas cuando un reintegro está próximo a acreditarse.

Implementación

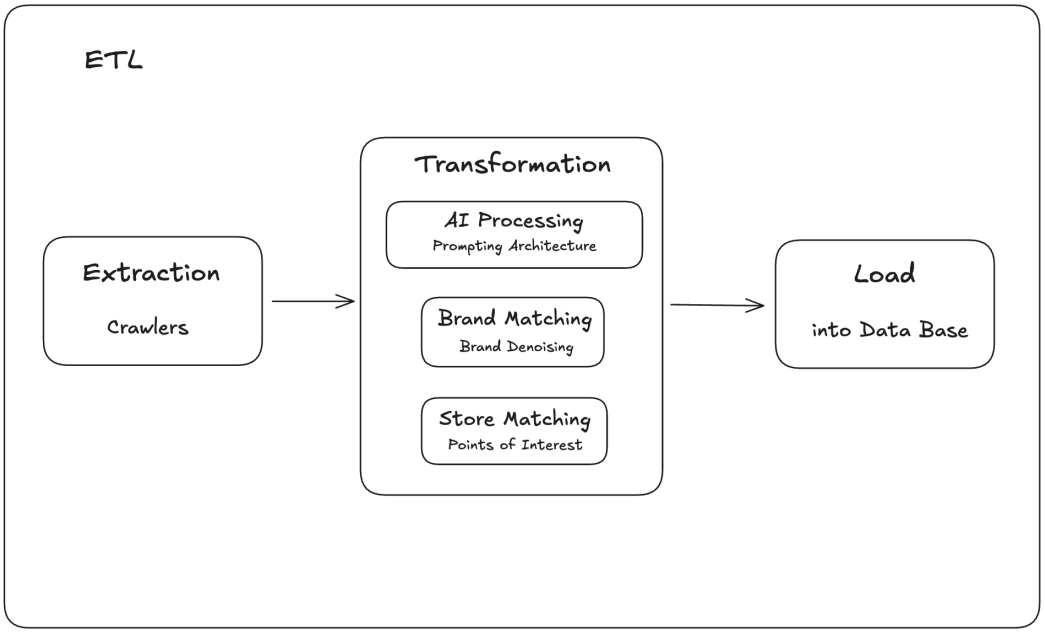
Módulos del sistema

Sobre la arquitectura de la app, la solución se organiza en módulos funcionales que encapsulan responsabilidades y datos, y se comunican mediante GraphQL (queries, mutations y, cuando corresponde, subscriptions), con interfaces claras entre ellos.

- **Autenticación:** Gestiona alta, login y sesiones seguras (Google/Email), renovación y expiración de tokens, recuperación de credenciales y políticas por rol. Además, la API valida el contexto del usuario para proteger datos y operaciones.

- **Tarjetas:** Permite al usuario registrar sus tarjetas de crédito y débito especificando banco emisor, tipo y marca, sin ingresar datos confidenciales como número o código de seguridad. Esta información se utiliza para filtrar y recomendar beneficios compatibles en base a las promociones disponibles.

- **ETL:** El proceso ETL se organiza en tres etapas encadenadas. Extracción obtiene datos brutos desde las fuentes públicas mediante crawlers. Transformación constituye el núcleo inteligente: sobre los datos extraídos aplica submódulos específicos para unificar entidades (Entity Matching / Brand Judging), normalizar comercios y ubicaciones (POIs) y procesar los Términos y Condiciones con una Prompting Architecture que interpreta y estructura las reglas de cada oferta. Finalmente, Carga persiste el resultado ya curado en la base de datos mediante operaciones idempotentes, listas para ser consultadas por los servicios de la aplicación.



- **Extracción (Web Crawlers):** Está compuesta principalmente por webcrawlers, scripts escritos en Python que utilizan librerías como Requests y BeautifulSoup para navegar sitios de bancos y billeteras de manera automatizada, detectando estructuras conocidas y extrayendo la información relevante para luego almacenarla.

- **Transformación:** Es el componente más importante del sistema, correspondiente a la fase de transformación del flujo ETL, donde se realiza el procesamiento inteligente de los datos extraídos por los crawlers. Este módulo utiliza modelos de lenguaje (LLMs) a través de APIs externas (como Gemini) para interpretar y estructurar la información de las ofertas. Sus principales funciones incluyen:

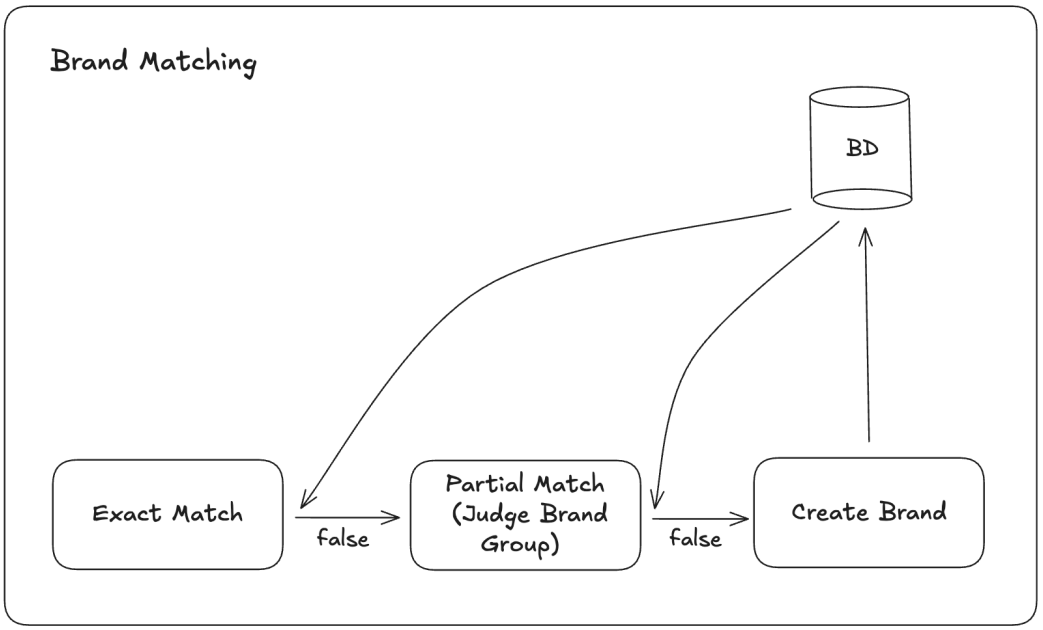
    - Análisis semántico de títulos, descripciones y términos y condiciones de las promociones.
    - Normalización de la información en un formato estándar, independiente de la fuente original.
    - Clasificación automática por tipo de beneficio (descuento, reintegro o cuotas).
    - Validación y limpieza de datos, eliminando duplicados o entradas inválidas.
    - Enriquecimiento contextual, agregando campos derivados como bancos, marcas, categorías y métodos de pago válidos.
    - Generación de objetos normalizados, listos para ser cargados en la base de datos.

    Dentro de Transformación se resuelve la vinculación de entidades clave detectadas en las ofertas con nuestro catálogo: **marcas** y **comercios**. Primero se determina la marca correcta y, a partir de ello, se resuelven los

locales donde aplica la promoción. El objetivo es garantizar IDs consistentes, evitar duplicados y permitir consultas y notificaciones confiables.

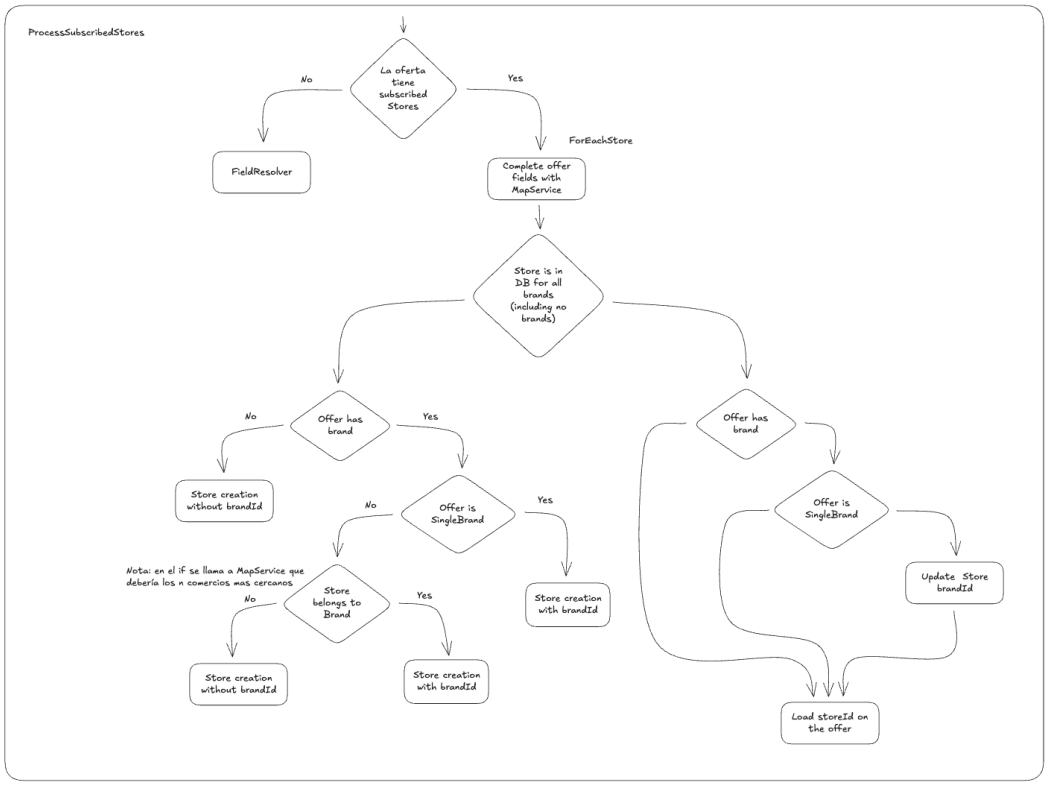
Brand Matching

Vincula la marca detectada en la oferta con el catálogo. Primero intenta una coincidencia exacta sobre el nombre normalizado y, si la encuentra, devuelve el `brand_id` existente. Si falla, aplica coincidencia parcial evaluando variantes y alias para decidir si pertenece a un grupo de marca ya registrado; si corresponde, retorna ese `brand_id`. Cuando ninguna de las dos coincide, crea la marca y devuelve el nuevo `brand_id`. Todas las escrituras se realizan como upserts idempotentes para mantener consistencia y trazabilidad.



Store Matching

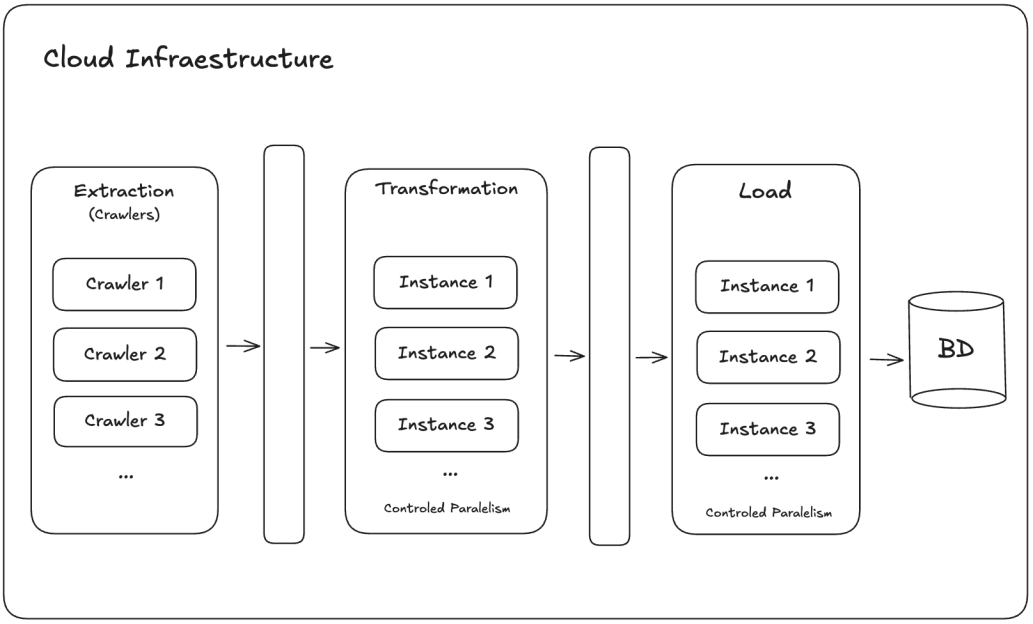
Complementa al paso anterior resolviendo los locales donde aplica la promoción. Si los TyC indican que rige para todos los locales de la marca, se expande contra la lista de comercios adheridos existente. Si enumeran sucursales específicas, se normalizan nombre y dirección, se busca el comercio y, de no existir, se crea y se devuelve su `store_id`. Ante ambigüedad en la fuente, se asocia provisionalmente por marca + localidad y se deriva a una cola de revisión.



Este módulo constituye el núcleo de inteligencia del sistema, asegurando la calidad y coherencia de la información antes de ser almacenada.

- **Carga:** Recibe los datos transformados y realiza operaciones upsert en la base de datos MongoDB, garantizando consistencia e integridad. Indexa campos críticos (vigencia, ubicación, categoría), vincula las ofertas con bancos, marcas y tarjetas. Permite que el backend exponga información curada y lista para consulta a través de GraphQL.

Todo el sistema corre en la nube con capacidad de escalado horizontal. La Extracción se paraleliza por *crawler*, la Transformación se despliega como servicio independiente que concentra los submódulos mencionados, y la Carga opera con paralelismo controlado hacia la base de datos para sostener throughput sin comprometer consistencia.



- **Asistente AI:** Chatbot embebido basado en modelos de lenguaje natural (LLMs), diseñado para brindar asesoramiento financiero personalizado. Puede responder preguntas como “¿Qué promoción tengo cerca hoy?” o “¿Cuándo me deberían haber reintegrado la compra de Carrefour?”, accediendo al contexto del usuario y a los datos procesados por la aplicación.

- **Notificaciones push:** Envía alertas contextuales al usuario sobre nuevas promociones, reintegros pendientes o beneficios cercanos. Utiliza los datos de geolocalización y preferencias personales para priorizar las notificaciones más relevantes.

Tecnologías utilizadas y justificación

Cada tecnología fue seleccionada como resultado de un proceso iterativo de evaluación, pruebas de concepto y validación con respecto a la problemática planteada. A diferencia del marco teórico, donde se consideran alternativas en abstracto, en esta sección se justifica por qué se descartaron opciones y qué aprendizajes surgieron durante el desarrollo. Esta sección también expone las decisiones tomadas frente a imprevistos técnicos y limitaciones propias del ecosistema argentino.

- **Flutter (Frontend mobile)**

  El proceso de definición del frontend comenzó con la idea de una Progressive Web App desarrollada en React, usando PWABuilder para facilitar su instalación. Sin embargo, las pruebas en iOS revelaron múltiples problemas: ausencia de notificaciones push, restricciones en el acceso a la geolocalización en segundo plano, baja performance y falta de compatibilidad con gestos y comportamiento nativo. Estas limitaciones afectaban directamente a funcionalidades críticas del sistema como GeoFencing o alertas contextuales, forzando un replanteo de la arquitectura.

Se compararon Flutter y React Native como soluciones multiplataforma reales. Si bien React Native facilitaba el onboarding del equipo por su base en JavaScript, se identificaron problemas con compatibilidad en iOS y necesidad de usar módulos nativos adicionales. En contraposición, Flutter ofrecía mejor performance gráfica, una comunidad creciente y control total sobre el diseño visual, lo que lo volvía más apropiado para una interfaz centrada en tarjetas, beneficios y recomendaciones.

No obstante, al no tener experiencia previa en Flutter, el equipo optó por usar FlutterFlow, una plataforma Low Code que permite generar código limpio, mantener extensibilidad y exportar a entornos productivos sin sacrificar personalización. Aun así, usar una plataforma externa trae compromisos; por un lado, personalización más acotada que en Flutter puro, por el otro, dependencia de un tercero para el ciclo de desarrollo. En la práctica, el equipo sorteó la primera limitación incorporando funciones personalizadas en Flutter nativo dentro del proyecto (mediante custom code, widgets y acciones), manteniendo la flexibilidad donde hacía falta. Respecto a la dependencia, FlutterFlow permite descargar el código del proyecto, lo cual resulta clave para continuar el desarrollo en Flutter puro cuando se necesite, reduciendo el lock-in y asegurando sustentabilidad técnica a largo plazo.

Un hallazgo relevante, que no había surgido con claridad en la investigación inicial, fue que el geofencing no tiene soporte pleno en los frameworks multiplataforma (Flutter/React Native) porque es capacidad propia del sistema operativo y se gestiona únicamente con lenguaje nativo (Swift en iOS y Kotlin en Android). Se exploraron alternativas como cron jobs en background, pedir permisos de ubicación “siempre” como si fuera una app de navegación o integrar librerías nativas comerciales. Ninguna resultó viable ya que los cron jobs son poco confiables por las políticas de suspensión, la ubicación constante degrada la experiencia (batería/privacidad) y suele ser rechazada por las guías de plataforma, y las librerías nativas evaluadas implicaban costos de licencia elevados, riesgos de bloqueos del SO en actualizaciones y baja mantenibilidad. En consecuencia, el enfoque con FlutterFlow permitió entregar un MVP funcional y escalable en menos tiempo, con el plan explícito de migrar a implementaciones nativas para geofencing si la evolución del producto lo exige.

- **Node.js + Apollo Server + GraphQL (Backend API)**

  El backend se prototipó inicialmente en REST, pero la necesidad de suministrar a cada pantalla solo la información estrictamente relevante motivó la adopción de GraphQL. Esta decisión introdujo dos ventajas clave, en primer lugar la exposición del servicio mediante un mono-endpoint (`/graphql`) que simplifica el orquestado de llamadas en el cliente móvil y reduce el acoplamiento, en segundo lugar la selección exacta de campos a nivel de consulta (field-level selection) permite al frontend definir la “forma” de la respuesta y evitar tanto el overfetching como el underfetching. El resultado son payloads más pequeños y menos viajes de red, con impacto directo en latencia, consumo de datos y batería, especialmente beneficioso en dispositivos de menor potencia o en contextos de conectividad limitada. Esto fue determinante en vistas personalizadas como la de “For You”, donde el volumen y la variabilidad de datos dependen del perfil del usuario.

La curva de aprendizaje de GraphQL y sus resolvers supuso un desafío inicial. No obstante, Apollo Server facilitó la modularización del esquema, la integración de middlewares (autenticación y logging) y el manejo de errores tipados. El desarrollo comenzó en JavaScript, pero la necesidad de tipado estático para mantener la coherencia entre el schema, los resolvers y los modelos de datos llevó a migrar a TypeScript (una versión transpilada de JavaScript con tipado fuerte). Esta decisión también se vio impulsada por la mejora de la Developer Experience: gracias a TypeScript y su integración con el IDE, el desarrollo se enriquece con autocompletado inteligente y detección temprana de errores de tipo, lo que reduce fallos de integración antes de la ejecución. El resultado es un ciclo de desarrollo más rápido y predecible, con menos errores por desalineaciones de schema, y una base de código más mantenible y extensible.

Node.js se seleccionó por su naturaleza asincrónica y su eficiencia en el manejo de múltiples conexiones concurrentes, junto con su alta compatibilidad con el resto del stack JavaScript. Esto permitió mantener una base tecnológica unificada entre frontend y backend, simplificando el desarrollo, las pruebas y la evolución del producto.

- **MongoDB (Base de datos NoSQL)**

  Durante las primeras iteraciones del diseño de datos se contempló el uso de bases relacionales como PostgreSQL, pero la naturaleza semiestructurada de las promociones (campos opcionales, condiciones variables, rangos, fechas, tarjetas y rubros) impulsó la adopción de un modelo NoSQL. Tras evaluar alternativas NoSQL se seleccionó MongoDB por su flexibilidad de esquema, el soporte natural para documentos anidados y su capacidad geoespacial nativa (índices 2dsphere y consultas por proximidad), aspectos críticos para el motor de cercanía. Esta elección redujo la complejidad del modelado inicial y habilitó respuestas ágiles en filtros por ubicación y vigencia.

  En cuanto al despliegue, se optó por MongoDB Atlas, plataforma de datos totalmente administrada y multinube que integra base de datos, búsqueda y analítica. La condición multinube evita el lock-in con un proveedor específico, a diferencia de alternativas como DynamoDB, acopladas a AWS, y permite ejecutar en el proveedor que resulte más conveniente en cada etapa. En este contexto, el servicio fue levantado en GCP para aprovechar créditos disponibles, manteniendo la portabilidad futura entre nubes. Operativamente, Atlas simplifica tareas de administración (backups, escalado, monitoreo y aplicación de parches), favoreciendo el foco en las capas de negocio y en la evolución del producto.

- **Python (Web scraping)**

  Ante la ausencia de estándares de Open Banking en Argentina, primero se evaluó el uso de APIs abiertas y, al no cubrir los casos de uso, se decidió implementar scraping sobre sitios públicos. Durante la fase de investigación se exploraron Puppeteer, Selenium y Playwright para automatizar la navegación en páginas dinámicas; Playwright se utilizó únicamente como prueba de concepto dentro de una

etapa de la Transformation Layer y, dado que esa capa no prevaleció en la arquitectura final, no pasó a formar parte del stack. La implementación vigente priorizó un enfoque liviano con Python, Requests y BeautifulSoup como parser para HTML, lo que redujo la complejidad operativa y el costo de mantenimiento. En sitios con estructuras más complejas se aplicaron estrategias acordes al stack —como la detección de endpoints internos, el manejo de tiempos de respuesta y la consolidación de contenido renderizado cuando estuvo disponible— evitando automatizar la navegación del sitio. Python se consolidó por su ecosistema maduro, su comunidad y la disponibilidad de librerías estables para tareas de crawling y parsing.

Los principales obstáculos provinieron de barreras anti-bot y de la reputación de IP. Numerosos sitios o CDNs bloquearon rangos de datacenters y aplicaron limitaciones de tasa o detección de patrones de tráfico anómalos que se manifestaron como respuestas 429 o 403 intermitentes, además de huellas de navegador y de transporte (headers, timing, TLS) que delataron automatización. Se identificó que una solución efectiva para estos casos era la rotación de proxies.

Gran parte de estas dificultades se resolvieron mediante la creación de sacrawl, una librería interna que estandarizó el ciclo de vida de cada crawler y concentró un gestor de sesión, control de ritmo y cuotas, perfiles de fingerprint rotables, utilidades de parsing y normalización de fechas, monedas y porcentajes, validación previa al upsert, observabilidad con logging estructurado, métricas y trazas por dominio, e idempotencia basada en hashes de contenido y claves naturales.

- **Modelos de Lenguaje (LLM - GPT)**

  Desde el inicio se contempló el uso de modelos de lenguaje (LLMs), ya que el procesamiento confiable de ofertas con terminología legal, formatos heterogéneos y textos extensos difícilmente podía resolverse solo con reglas. Aun así, las primeras versiones buscaron reducir costo y complejidad mediante expresiones regulares, reglas manuales y clasificadores básicos; el enfoque resultó insuficiente por la ambigüedad del lenguaje y la falta de estandarización entre bancos.

  La adopción de un pipeline con LLMs vía APIs externas permitió identificar entidades clave (topes, vigencias, tarjetas admitidas, condiciones particulares) con mayor precisión y tolerancia a variaciones de formato. No obstante, su uso introdujo desafíos propios: alucinaciones (respuestas plausibles pero falsas) y no determinismo (variabilidad entre ejecuciones con la misma entrada). Para mitigarlos, se aplicaron buenas prácticas de prompt engineering: instrucciones más estrictas y orientadas a extracción, ejemplos guiados (few-shot), restricciones de salida (JSON validado contra esquemas), control de temperatura/top-p, verificación post-proceso con validadores y reglas, y reintentos con chequeos de coherencia.

  Se trabajó también en la división de responsabilidades de prompts, separando plantillas por tarea (extracción de campos, normalización, clasificación, validación) para simplificar pruebas y mejorar mantenibilidad.

En términos de costo/beneficio, se estableció la necesidad de equilibrar la mejora de prompts con la elección del modelo: subir de modelo incrementa precisión pero también costo; mejorar el prompt puede abaratar, aunque con retornos decrecientes. Por ello, se priorizó medir sobre conjuntos de prueba etiquetados, comparar variantes (A/B) y seleccionar el punto de operación que maximiza exactitud por token gastado.

Para sostener el ciclo de vida de prompts se evaluaron herramientas de prompt management (versionado, historial de cambios, etiquetado por entorno, experimentación/A-B, métricas de uso, rollbacks y trazabilidad de "prompt → salida → coste/latencia"). Se adoptó PromptLayer para centralizar versionado y telemetría de prompts, asociar ejecuciones a variantes específicas, monitorear consumo por tarea y facilitar la reproducibilidad de resultados, lo que mejoró la gobernanza y la capacidad de iteración del pipeline.

Pese a las mejoras, la dependencia de servicios externos y el costo por token continúan bajo seguimiento, con métricas de precisión, frescura y gasto que orientan futuras optimizaciones del pipeline y eventuales cambios de proveedor o estrategia.

- **Firebase Authentication**

  La primera implementación del sistema de login fue desarrollado por nosotros utilizando la autenticación de BetterAuth, basado en JWT, para el control de expiración y validación de roles. Sin embargo, problemas como la falta de documentación de BetterAuth hacía que mantener un sistema seguro, actualizado y libre de vulnerabilidades demandara una inversión considerable que no se justificaba frente a las soluciones existentes.

  Firebase Authentication fue elegido por su integración nativa con Flutter, su confiabilidad, su escalabilidad automática y su interfaz de administración intuitiva. Además, su esquema de precios permitía implementar todas las funcionalidades necesarias sin costo.

  Inicialmente se implementó el inicio de sesión con correo y contraseña, pero luego se reemplazó por autenticación mediante Google SSO. Esta decisión resolvió de forma elegante la validación de identidad, evitando que el sistema tuviera que manejar contraseñas o flujos de recuperación de credenciales.

  El uso de Firebase permitió también sincronizar sesiones en segundo plano, activar notificaciones personalizadas y simplificar la gestión de usuarios desde un único panel.

- **Geolocalización**

  La geolocalización habilita recomendaciones por cercanía y requiere un encuadre preciso de permisos y configuración en ambos sistemas operativos. En términos generales, es necesario explicar claramente el propósito (por ejemplo, mostrar descuentos cercanos), solicitar el nivel de acceso adecuado (solo al usar la app, en segundo plano o siempre) y ajustar la aplicación para que reciba actualizaciones de

ubicación conforme a las políticas de cada plataforma. El objetivo es pedir el mínimo permiso necesario, manteniendo transparencia y un equilibrio entre utilidad y consumo de batería.

La solución principal prevista fue el geofencing por su eficiencia: el sistema operativo activa la app al entrar o salir de regiones predefinidas alrededor de comercios. Dado que existe un límite de regiones por aplicación, se planteó un enfoque jerárquico: un cerco amplio (macro-zona) despierta la app y, al detectar movimiento, se rotan dinámicamente los cercos “hijos” más relevantes cercanos al usuario.

Sin embargo, esta implementación quedó restringida por el framework multiplataforma utilizado. En consecuencia, se evaluaron alternativas como la ubicación en segundo plano (sujeta a throttling como Doze/Low Power y a políticas que posponen ejecuciones) y la ubicación continua tipo navegación mediante Foreground Service o actualizaciones constantes (más intrusiva y con mayor impacto en batería). Dado que estas alternativas no ofrecen una experiencia de uso adecuada, se decidió posponer la funcionalidad para una etapa siguiente, con integración nativa (Swift/Kotlin) que permita un geofencing robusto y conforme a las pautas de cada plataforma.

- **Notificaciones push**

  Las notificaciones se integran con FCM (Android) y APNs (iOS) para enviar alertas contextuales (cercanía a comercios adheridos, vigencias por vencer, reintegros pendientes). Para habilitar las notificaciones se configuraron las cuentas de desarrolladores para cada plataforma (Android y iOS) y se vincularon con el proyecto de Firebase. En ambos casos se solicita el permiso al usuario para habilitar las notificaciones en su dispositvo.

  Para favorecer la comprensión y la aceptación, se desarrollaron pantallas de pre-permiso que explican de forma clara por qué se solicitan notificaciones (beneficios cercanos y recordatorios útiles) y cuándo se utilizarán. Esta práctica mejora la tasa de consentimiento y alinea la experiencia con las políticas de cada plataforma, manteniendo la transparencia como principio central.

- **Google Cloud Platform + Terraform**

  Durante la planificación de infraestructura se evaluaron proveedores como AWS y Azure. GCP ofreció la mejor sinergia con Firebase, una capa gratuita generosa y una consola unificada, habilitando un despliegue inicial sin costos y con opciones de escalabilidad horizontal. A diferencia de otras capas gratuitas, Google además otorga créditos promocionales que pueden utilizarse en prácticamente cualquier servicio de la plataforma e incluso en servicios de terceros alojados en su infraestructura, lo que facilita experimentar y crecer sin incurrir en gasto inmediato. La integración con Cloud Run, Firestore y Artifact Registry simplificó el despliegue de contenedores y la gestión de datos, con buena estabilidad, rendimiento y documentación.

Para reforzar portabilidad y gobernanza se incorporó Terraform como Infrastructure as Code. Esta decisión permite cambiar de cuenta o proyecto de GCP de forma controlada (por ejemplo, al desvincular el proyecto de cuentas personales o de la universidad), minimizar el lock-in al describir la infraestructura en código declarativo y versionado, estandarizar entornos (desarrollo, staging, producción) mediante módulos y workspaces, y auditar o revertir cambios con planes reproducibles y state gestionado. En conjunto, GCP aporta un ecosistema conveniente para el arranque y la operación, mientras que Terraform garantiza reproducibilidad y flexibilidad para futuras migraciones o reconfiguraciones administrativas.

- **Docker**

  Si bien no fue posible dockerizar la totalidad del sistema, dado que el frontend es una aplicación móvil distribuida a través de tiendas y por lo tanto no es posible ejecutarlo en contenedores, sí se dockerizaron todos los componentes restantes (backend, crawlers y módulo ETL).

  Cada uno se encapsuló en su propio contenedor, lo que permitió desarrollar y probar los servicios de forma aislada, reducir errores derivados de diferencias de entorno y facilitar la integración continua mediante pipelines automatizados.

  Una de las principales ventajas fue la posibilidad de generar imágenes preconstruidas, libres de claves o secretos, y publicarlas directamente en un container registry. Esto agilizó las pruebas en la nube y estableció una base sólida para una futura orquestación con Kubernetes u otra solución escalable.

Planificación de pruebas

El backend de SaveApp expone su funcionalidad mediante resolvers GraphQL que encapsulan reglas de negocio, autorizaciones y composición de datos. Dado que la app móvil y módulos internos dependen de respuestas consistentes y normalizadas, este plan prioriza pruebas unitarias a nivel de resolver para capturar rápidamente regresiones lógicas, contratos de datos y manejo uniforme de errores. La estrategia se apoya en *mocks* de modelos y utilidades para aislar la lógica del resolver, y complementa con pruebas “Pop” que validan la normalización (por ejemplo, convertir `bankId` en un objeto `bank`). Aunque no sustituye pruebas de integración, este enfoque ofrece alto *feedback* a bajo costo y guía la calidad desde el núcleo de negocio.

Alcance y componentes bajo prueba

Este plan cubre las operaciones críticas de recuperación, mutación y población de entidades principales (usuarios, marcas, categorías, bancos, tarjetas, comercios y seguimientos de ofertas). Se incluye además la familia de resolvers “Pop”, cuyo rol es entregar estructuras listas para consumo de UI sin *joins* posteriores. Así, se valida tanto el “qué” (reglas) como el “cómo” (formato de salida).

Resolvers cubiertos

- **User:** `UserResolver`, `PopUserResolver`, `populateUser`
- **Brand:** `BrandResolver`
- **Category:** `CategoryResolver`
- **Bank:** `BankResolver`
- **Card:** `CardResolver`, `PopCardResolver`, `populateCard`
- **Store:** `StoreResolver`
- **Tracking:** `TrackingResolver`, `PopTrackingResolver`, `populateTracking`

Tipos de pruebas: Unitarias con *mocks* de `@saveapp-org/shared/models`, utilidades y decoradores de caché.

Entorno de pruebas

Las pruebas se ejecutan con **Jest**, elegido por su rapidez, *watch mode* y ecosistema maduro. El aislamiento se logra *moqueando* modelos Mongoose/Typegoose, utilidades comunes (fechas, `toPlainObject`) y decoradores de caché, de modo que cada *spec* evalúa exclusivamente la lógica del resolver. Los datos de prueba se gestionan con *fixtures*/factories por entidad para mejorar legibilidad y evitar *coupling* entre casos. La autenticación se simula con `context.user`, permitiendo ejercitar rutas de autorización sin depender de un proveedor real.

- **Framework:** Jest
- **Aislamiento:** *Mocks* para Model de Mongoose/Typegoose, utilidades (`toPlainObject`, fechas) y decoradores de caché.
- **Datos:** Objetos *mock* por entidad (`User`, `Brand`, `Card`, `Offer`, `Tracking`, `Store`).
- **Autenticación:** Contexto GraphQL simulado mediante `context.user`.

Supuestos y dependencias

Se asume que los modelos respetan contratos básicos (devuelven documento, `null` o error) y que `toPlainObject` elimina acoplamientos a Mongoose, dejando POJOs aptos para serialización. Los decoradores de caché se tratan como una optimización que no cambia la semántica; por ello se valida su integración superficial. También se explicitan fallos frecuentes (por ejemplo, *cast* de `ObjectId`) para asegurar que burbujean como errores manejables por la capa superior.

- Los modelos devuelven documentos o `null`/errores siguiendo el contrato simulado.
- `toPlainObject` transforma documentos a objetos planos donde aplica.
- Los decoradores de caché no alteran la lógica funcional (se valida integración superficial).
- Los IDs pueden fallar por formato (“Cast to ObjectId failed”) y deben propagarse.

Riesgos conocidos

El uso intensivo de *mocks* puede esconder incompatibilidades con Mongoose real, índices o *hooks* no simulados. Además, los *paths* de error genéricos podrían diferir de producción, y los escenarios de concurrencia en unit tests son necesariamente sintéticos. Para mitigar, se sugiere complementar con pruebas de integración (por ejemplo, Mongo en memoria) y *contract tests* GraphQL que verifiquen el *schema* y los códigos de error end-to-end.

- Dependencia alta de *mocks* puede ocultar problemas de integración real.
- *Paths* de error genéricos pueden diferir del comportamiento de producción de Mongoose.
- Escenarios de concurrencia sintéticos (limitados a la lógica del resolver).

Criterios de aceptación

La aceptación se apoya en tres pilares: (1) *paths* felices devuelven exactamente lo esperado; (2) errores frecuentes (no autenticado, no encontrado, BD, IDs inválidos) se manejan de forma uniforme y predecible; (3) los resolvers "Pop" entregan estructuras normalizadas, reduciendo la carga de transformación en clientes y minimizando ambigüedades en UI.

- Cada **query/mutation** devuelve el resultado esperado en *paths* felices.
- Cada **query/mutation** gestiona adecuadamente: no autenticado, recurso no encontrado, errores de base de datos, IDs vacíos/ inválidos.
- Los resolvers "Pop" devuelven estructuras pobladas y normalizadas (p. ej., `bank` en vez de `bankId`).

Cobertura por módulo y casos de prueba

Esta sección detalla, por módulo, el propósito, los comportamientos esperados y los casos de prueba cubiertos, con conectores y justificaciones de diseño para facilitar trazabilidad con requisitos y criterios de aceptación.

- ### User

  Propósito y encuadre:
  El módulo de usuario concentra autorización contextual, gestión del perfil y la orquestación de relaciones con tarjetas, marcas, ofertas guardadas y *trackings*. Las pruebas buscan garantizar: (i) lectura segura del usuario autenticado, (ii) consistencia de mutaciones (agregar/quitar dependencias) y (iii) resiliencia ante contextos inválidos y errores de base.

  Lectura y población:
  `me` retorna el usuario autenticado; si falta `context.user`, corresponde UNAUTHENTICATED; si el documento no existe, NOT_FOUND('User'). La utilidad `populateUser` entrega *payload* coherente para la UI: debe poblar `cards`, `favourites`, `trackings` y `savedOffers` en un usuario y en arreglos; cualquier error de *populate* se propaga.

Mutaciones y reglas de negocio.

- **Tarjetas:** `addCard` exige autenticación, existencia de `user` y `card`, y evita duplicados (ALREADY_ADDED('Card')). `removeCard` retorna éxito o NOT_FOUND('User') si no se actualizó el documento.
- **Marcas favoritas:** `addFavourite` valida existencia (NOT_FOUND('Brand')); `removeFavourite` requiere autenticación.
- **Ofertas guardadas:** `addSavedOffer` retorna NOT_FOUND('Offer') o ALREADY_ADDED('Offer'); `removeSavedOffer` exige autenticación.
- **Trackings:** `addTracking` delega en `TrackingResolver.createTracking` y luego puebla; `removeTracking` informa NOT_FOUND('Tracking for this offer') si no existe.
- **Perfil:** `updateMe` soporta actualizaciones parciales; retorna `null` si el usuario no se encuentra.

Robustez y bordes:
IDs vacíos son rechazados explícitamente; `context` nulo/indefinido dispara fallas claras; errores de conexión/BD **burbujean**.

Casos cubiertos (síntesis):
- `populateUser` (único/array/errores)
- `me` (OK/UNAUTHENTICATED/NOT_FOUND/errores BD)
- Mutaciones (`add/removeCard`, `add/removeFavourite`, `add/removeSavedOffer`, `add/removeTracking`, `updateMe`)
- Edges (contexto nulo, IDs vacíos, errores BD transversales).

- ## PopUser

Propósito y encuadre:
`PopUserResolver` entrega el mismo usuario con relaciones pobladas, eliminando *joins* en el cliente.

Comportamiento esperado:
`mePop` retorna, si hay autenticación, un usuario con `cards`, `favourites`, `trackings` y `savedOffers` poblados. Sin contexto: UNAUTHENTICATED. Si no existe: NOT_FOUND('User'). Errores de *populate*: se propagan.

Casos cubiertos (síntesis):
- `mePop` (autenticado con relaciones pobladas / UNAUTHENTICATED / NOT_FOUND / errores de *populate*).

- ## Brand

Propósito y encuadre:
El resolver de marca atiende (i) búsqueda por ID y (ii) un campo derivado sensible al contexto de usuario: `isFavouritedByUser`.

Búsqueda por ID:
`getBrandById` retorna la marca o NOT_FOUND('Brand'). *Casts* inválidos, IDs vacíos/extremos, errores BD, concurrencia y *timeouts* se validan para mantener determinismo y propagación correcta.

Campo derivado `isFavouritedByUser`:
Con usuario autenticado, retorna `true` si la marca está en `user.favourites`; de lo contrario, `false`. Sin autenticación o sin `context.user`: `null`. Se contemplan usuario inexistente (NOT_FOUND('User')), arrays malformados (tratar como no favorito), errores de BD y excepciones de `equals`, todo sin colapsar el flujo.

Casos cubiertos (síntesis):
- `getBrandById` (OK/NOT_FOUND/cast inválido/errores/IDs extremos/concurrencia/timeouts)
- `isFavouritedByUser` (true/false/null, NOT_FOUND('User'), arrays malformados, errores BD, `equals` lanza, brand sin `_id`).

- ## Category

Propósito y encuadre:
Las categorías estructuran el catálogo; se exige predictibilidad ante vacíos y errores.

Listados y búsqueda:
`getAllCategories` devuelve la lista o `[]`; errores BD se propagan. `getCategoryById` retorna la categoría o NOT_FOUND('Category'); *cast* inválido y errores BD están contemplados.

Casos cubiertos (síntesis):
- `getAllCategories` (OK/vacío/errores BD)
- `getCategoryById` (OK/NOT_FOUND/cast inválido/errores BD).

- ## Bank

Propósito y encuadre:
Se validan (i) operaciones de lectura (listado y por ID) y (ii) la integración del decorador de caché como optimización no-semántica.

Caché como optimización:
El decorador debe estar aplicado, invocar al método original y permitir que los errores burbujeen; no cambia contratos.

Listados y detalle:
`getAllBanks` devuelve lista o `[]`; `getBankById` retorna documento o **NOT_FOUND('Bank')**. *Casts* inválidos y errores BD se verifican, incluyendo la propagación en `BankModel.findById` directo.

Casos cubiertos (síntesis):

- Decorador de caché (aplicado/llama al original/propaga errores)
- `getAllBanks` (OK/vacío/errores BD)
- `getBankById` (OK/NOT_FOUND/cast inválido/errores BD incl. llamada directa).

- **Card**

Propósito y encuadre:
Las tarjetas conectan usuarios con bancos y beneficios; se prueba la normalización estructural y la variedad de consultas, además de la exposición Pop para la UI.

Población y normalización (`populateCard`):
Una card poblada convierte `bankId` → `bank` y elimina `bankId`. Debe funcionar en documento único y en arreglos; un arreglo vacío conserva `[]`. Errores de *populate* se propagan.

Consultas y Pop resolvers:
`CardResolver`: `getAllCards` (lista/vacío/errores), `getCardById` (OK/`NOT_FOUND('Card')`/*cast* inválido/errores BD), `getCardsByBank(bankId)` (filtra por banco; tolera `null/undefined` → `[]`; bancos inexistentes y volumen alto quedan en recomendaciones).
`PopCardResolver`: `getAllPopCards` y `getPopCardById` verifican población correcta y errores propagados; `myCards(context)` exige autenticación, retorna tarjetas pobladas, y contempla `UNAUTHENTICATED`, `NOT_FOUND('User')`, errores BD y contexto nulo/indefinido.

Casos cubiertos (síntesis):
- `populateCard` (bankId→bank, arrays, `[]`, errores)
- `getAllCards`
- `getCardById`
- `getCardsByBank` (incluye `null/undefined` → `[]`) · `getAllPopCards` · `getPopCardById`
- `myCards` (autenticado/UNAUTHENTICATED/NOT_FOUND('User')/errores BD/contexto nulo).

- **Store**

Propósito y encuadre:
Comercios presentan datos opcionales y gran variabilidad; se evalúan entradas inválidas y escenarios operativos adversos (red, memoria, autenticación a Mongo).

Búsqueda por ID con diversidad de entradas:
`getStoreById` retorna el documento o NOT_FOUND('Store'). Se validan IDs vacíos (`''`), `null/undefined`, *casts* inválidos, IDs extremadamente largos o con caracteres especiales, e invocaciones sucesivas con distintos IDs, preservando el contrato.

Errores, concurrencia y consistencia:
Errores BD, *timeouts* de red, conexión/autenticación a Mongo, presión de memoria y errores personalizados se propagan; múltiples instancias del resolver y concurrencia mantienen determinismo.

Datos mínimos, completos y opcionales:
Se prueba que documentos mínimos/completos y campos opcionales nulos/indefinidos/`''` no rompan el flujo, entregando *payload* utilizable por la UI.

Casos cubiertos (síntesis):
`getStoreById` (OK/NOT_FOUND/cast inválido/IDs vacíos-extremos/falsy/`undefined` desde `findById`/invocaciones sucesivas/múltiples instancias) · Errores (BD, red/timeout, conexión/auth Mongo, memoria, personalizados) · Datos (mínimo/completo/opcionales nulos) · Garantías (propagación, firma consistente, determinismo bajo concurrencia).

- ## Tracking

Propósito y encuadre:
El módulo de *tracking* habilita seguir ofertas en el tiempo, calcular vigencia y entregar datos normalizados para la UI. Las pruebas cubren: (i) normalización y población (`populateTracking` y `populateOffer`), (ii) reglas de negocio en `TrackingResolver` (creación, eliminación y determinación de vigencia) y (iii) entrega a cliente en `PopTrackingResolver` (colecciones pobladas y consistentes con activos/inactivos).

Población y normalización (`populateTracking` / `populateOffer`):
Se valida el mapeo de claves foráneas: si `offerId` llega poblado, se mueve a `offer` y se elimina `offerId`. Aplica a un tracking y a arreglos (entrada `[]` conserva `[]`). Se comprueba que `TrackingModel.populate` reciba `[{ path: 'offerId' }]` y que luego `populateOffer` sea invocado por cada elemento. Cualquier error en la población de `offerId` o del propio `populateOffer` se propaga sin silenciarse.

Lógica de negocio del resolver (`TrackingResolver`):
`createTracking(offerId)` recupera la oferta, obtiene `refundTerm` y calcula `startDate`/`endDate` con `currentDate()` y `getEndDate(refundTerm)`. Se cubre el path feliz y las precondiciones: oferta inexistente ⇒ `NOT_FOUND('Offer')`; refund term faltante/nulo/0 ⇒ `NOT_FOUND('Refund term')`. Además, errores operativos (p. ej., *timeouts* en la selección) deben disparar la excepción. En concurrencia, múltiples invocaciones producen resultados consistentes (también se verifica el conteo de `findById`).
`deleteTracking(id)` expone un contrato booleano: `true` si elimina; `false` si no existe o el ID es vacío; errores de base (incluida presión de memoria simulada) burbujean.

`isActive(tracking)` delega en `isActiveDate(endDate)`; debe funcionar con fechas futuras/pasadas y con `endDate` nulo sin lanzar excepciones.

Entrega a la UI (`PopTrackingResolver`):
`myTrackedOffers(context)` retorna trackings poblados con `offer` cuando hay autenticación. Sin usuario en contexto: UNAUTHENTICATED. Si el usuario no existe: NOT_FOUND('User'). Usuario sin trackings: `[]`. Se contabilizan invocaciones a `populateOffer` para asegurar una por tracking. Errores en `findById → populate → select` se propagan; contextos malformados (objeto sin `user` o `context undefined`) deben fallar explícitamente.
`isActive` en `PopTrackingResolver` mantiene idéntico comportamiento al de `TrackingResolver`.

Casos cubiertos (síntesis):

- `populateTracking`/`populateOffer`: mapeo `offerId→offer`, eliminación de `offerId`, arrays (no vacío y `[]`), `TrackingModel.populate({ path: 'offerId' })`, `populateOffer` 1×elemento.
- `createTracking`: éxito con `refundTerm` → `startDate/endDate`; `NOT_FOUND('Offer')`; `NOT_FOUND('Refund term')` ante ausencia/`null`/`0`; *timeout* en selección lanza; concurrencia consistente.
- `deleteTracking`: `true` si elimina; `false` si no existe o ID vacío; errores BD (incl. "heap out of memory").
- `isActive`: usa `isActiveDate(endDate)`; maneja `endDate` nulo; escenarios de fecha futura y pasada.
- `myTrackedOffers`: autenticado OK (trackings poblados); UNAUTHENTICATED; NOT_FOUND('User'); sin trackings → `[]`; errores propagan; conteo de `populateOffer` proporcional al número de trackings.

Matriz de cobertura frente a requisitos esperados

Se asegura trazabilidad con requisitos operativos: autenticación, búsqueda por ID, listados, relaciones pobladas, manejo de errores, concurrencia y campos derivados. Esta matriz permite detectar huecos y priorizar refuerzos donde el riesgo es mayor (por ejemplo, errores de red/auth de Mongo o *timeouts*).

- **Autenticación:** `User`, `PopUser`, `PopCard`, `PopTracking`.
- **Búsqueda por ID:** `Brand`, `Category`, `Bank`, `Card`, `Store`.
- **Listados:** `getAll...` en `Bank`, `Card`, `Category`, `PopCard` (poblado).
- **Relaciones pobladas:** `populateUser`, `populateCard`, `populateTracking`, resolvers **Pop***.
- **Errores:** `NOT_FOUND`, `UNAUTHENTICATED`, BD, *cast* `ObjectId`, *timeouts*, memoria, red/auth Mongo.
- **Concurrencia:** `Brand`, `Store`, `Tracking`.

- **Derivados/normalización:** `isActive`, `bankId → bank`, `offerId → offer`.

Criterios de salida

El plan define salida clara para evitar *scope creep*: todas las unitarias vigentes deben pasar, no deben quedar casos críticos sin cubrir en los resolvers actuales y los mensajes de error deben ser consistentes con `ERRORS`. Esto habilita *merge* seguro y reduce el costo de mantenimiento.

- Todas las pruebas unitarias existentes pasan.
- No hay casos faltantes críticos respecto a los resolvers actuales.
- Errores y mensajes consistentes con `ERRORS`.

Recomendaciones de pruebas adicionales

Como cierre, se proponen extensiones para robustecer áreas de riesgo: duplicados y *parity* de errores en `User`, escenarios de rendimiento (listas grandes en `Brand` y `Card`), validación real de *cache* en `Bank`, bordes temporales en `Tracking` y *contract tests* GraphQL para asegurar esquema y permisos end-to-end.

- **User:** `addFavourite`/`addSavedOffer` con `UNAUTHENTICATED` y duplicados; paridad de errores BD en todas las *mutations*.
- **Brand:** rendimiento con *favourites* masivo.
- **Bank:** validar *cache* hit/miss real si se habilita caché efectiva.
- **Card:** `getCardsByBank` con banco inexistente y alto volumen.
- **Tracking:** fechas con zonas horarias y bordes de mes.
- **Integración:** *Contract tests* `schema.graphql` y permisos (p.ej., `permissions.ts`).

Prácticas operativas (sugeridas para el informe)

- **Gestión de datos de prueba:** *Factories* por entidad, *fixtures* mínimas y reutilizables, y *helpers* para `context.user`.
- **Métricas y umbrales:** *Coverage* objetivo (p.ej., 85% líneas/ramas en resolvers), reporte en CI y *gating* en PR.
- **Nomenclatura y estructura:** `resolverName.spec.ts` con secciones *happy path*, *errors* y *edges*; *mocks* en `__mocks__`.
- **CI/CD:** ejecución paralela, *watch* local, y *pre-commit hook* para suites rápidas.
- **Mitigación de riesgos:** una *smoke* de integración con Mongo en memoria y *contract tests* periódicos para alinear errores y *schema*.

Pruebas de carga y estrés

Para las pruebas de carga y estrés del backend de SaveApp se utilizó el Runner de Postman ejecutando *collections* de GraphQL contra el endpoint `/graphql`. Las colecciones se organizaron por flujo (lecturas, escrituras y resolvers "Pop"), con ambientes para `baseUrl`/token y archivos de datos (CSV/JSON) que parametrizan `operationName`, `query` y `variables`. Se emplearon pre-request scripts (generación/renovación de credenciales y `correlation-id`) y test scripts (registro de latencias y errores). La concurrencia se emuló lanzando múltiples Runners en paralelo con ramp-up y *think time* variable.

Alcance

- **Queries:** `me`, `mePop`, `getAllBanks`, `getBankById`, `getBrandById`, `getAllCategories`, `getCategoryById`, `getAllCards`, `getCardById`, `getAllPopCards`, `getPopCardById`, `getStoreById`, `myTrackedOffers`.
- **Mutations:** `addCard`, `removeCard`, `addFavourite`, `removeFavourite`, `addSavedOffer`, `removeSavedOffer`, `addTracking`, `deleteTracking`, `updateMe`.

Metodología

- **Diseño de colecciones** por tipo de flujo (lectura pesada, escritura/actualización, "Pop").
- **Datasets** con usuarios, IDs y combinaciones de filtros/variables para generar **variabilidad** de payloads.
- **Concurrencia** por paralelización de Runners con arranque escalonado (ramp-up), *think time* aleatorio y tamaños de payload distintos.
- **Validaciones** en *test scripts* para registrar tiempos, respuestas GraphQL y códigos de error del resolver.

Escenarios ejercitados

- **Lectura concurrente:** ráfagas y mesetas sobre `me`/`mePop`, `getAll*` y búsquedas por ID; los resolvers "Pop" fuerzan población y payloads mayores.
- **Escrituras concurrentes:** oleadas de `add`/`removeCard`, `add`/`removeFavourite`, `add`/`removeSavedOffer`, `add`/`deleteTracking`, `updateMe` para observar contención, validaciones e idempotencia lógica.
- **Mix realista de tráfico:** patrón **read-heavy** con **mutations** intercaladas; contraste **bursty** (picos cortos) vs **steady** (carga sostenida) para evaluar degradación y recuperación.

Conclusiones

Se definió utilizar el Runner de Postman porque permite correr carga real sobre el mismo contrato GraphQL que usa la app, con bajo costo de orquestación y reproducibilidad: reutilizamos colecciones del equipo, variamos datos sin infra adicional y golpeamos el stack completo (middlewares, resolvers, DB, caché). Es ideal para ciclos rápidos de “cambiar → medir → decidir” y para comparar perfiles de carga entre ramas/configuraciones.

Esta configuración nos permitió caracterizar el perfil de carga de cada resolver, anticipar el comportamiento en producción y preparar estrategias de escalado vertical si el límite es CPU/memoria del proceso u horizontal si el cuello está en concurrencia o latencia externa (DB/red).

En conjunto, la estrategia de verificación consolidada refleja el recorrido del equipo, las decisiones ante problemas reales y elecciones tecnológicas alineadas con una visión de producto escalable y sostenible. El resultado es un backend robusto y personalizable para el mercado argentino, con base sólida para su extensión a toda la región latinoamericana.

BENEFICIOS POST-IMPLEMENTACIÓN

La implementación de SaveApp representa una transformación significativa en la manera en que los usuarios acceden, comprenden y utilizan los beneficios asociados a sus tarjetas bancarias. A través de una solución integral, automatizada y centrada en el usuario, los beneficios esperados tras la puesta en funcionamiento del sistema abarcan tanto el plano individual del consumidor como el ecosistema financiero en su conjunto. A continuación, se detallan los principales beneficios post-implementación:

Empoderamiento del usuario y educación financiera

SaveApp no solo permite ahorrar dinero, sino también adquirir mayor conciencia sobre el uso inteligente de los medios de pago. Al visualizar el impacto real de sus decisiones (descuentos utilizados, reintegros recibidos, ahorro acumulado), el usuario se convierte en un actor activo y estratégico de su economía personal. Esto contribuye a una mejora en la salud financiera, al fomentar decisiones informadas y conscientes.

Acceso unificado a la información

Una de las principales problemáticas antes de SaveApp era la dispersión y dificultad de acceso a las promociones bancarias. Gracias a su sistema de scraping y análisis semántico, SaveApp centraliza la información de múltiples entidades en un solo lugar, eliminando la necesidad de recorrer distintos sitios o interpretar condiciones confusas. A diferencia de los modelos basados en crowdsourcing, que suelen sufrir cobertura incompleta, sesgos de participación, latencias (demoras en la carga/actualización), ruido y duplicados, además de riesgos de spam/manipulación y mayores costos de moderación y verificación, el enfoque automatizado proporciona actualizaciones sistemáticas, trazabilidad de origen (cada dato se vincula a su fuente oficial), normalización canónica de formatos y criterios homogéneos de interpretación. El resultado es una experiencia más confiable, fluida y efectiva; el feedback de usuarios puede usarse como señal complementaria para detectar anomalías, pero la fuente de verdad permanece en los datos verificados y capturados automáticamente desde los sitios oficiales.

Aumento en la utilización de promociones existentes

Muchas promociones bancarias no se aprovechan simplemente porque los usuarios no las conocen. Al ofrecer notificaciones contextuales en el momento y lugar adecuado, la app incrementa la tasa de utilización de beneficios ya disponibles. Esto representa un mayor valor real para el usuario, sin necesidad de cambiar hábitos de consumo ni contratar servicios adicionales. [1]

Automatización de tareas repetitivas y tediosas

Tareas como revisar manualmente sitios de bancos, comparar descuentos, calcular topes de reintegro o validar condiciones específicas son automatizadas por la app mediante IA.

Esto ahorra tiempo, reduce errores de interpretación y libera al usuario de una carga cognitiva innecesaria.

Mejora en la experiencia de usuario frente a apps tradicionales

Frente a billeteras o apps bancarias que muchas veces no priorizan la experiencia de usuario, SaveApp se presenta como una herramienta moderna, intuitiva y amigable. Desde su navegación fluida hasta el uso de IA conversacional, el diseño centrado en el usuario genera mayor engagement, satisfacción y fidelidad.

Transparencia y trazabilidad de beneficios

Con el módulo de seguimiento de reintegros y el panel de ahorro acumulado, SaveApp otorga al usuario la capacidad de auditar y verificar el cumplimiento de las promociones. Esta transparencia refuerza la confianza en las entidades emisoras y en la plataforma misma.

Escalabilidad regional y replicabilidad del modelo

El diseño desacoplado, el scraping adaptable y el análisis semántico genérico permiten que SaveApp pueda ser extendido fácilmente a otros países de la región. Esto multiplica su impacto potencial y posiciona al sistema como una solución escalable más allá del caso argentino.

Valor estratégico para bancos y comercios

Indirectamente, SaveApp también representa una oportunidad para los bancos emisores y los comercios adheridos: al aumentar la visibilidad y uso de promociones, se incrementa el volumen de transacciones, se fideliza a los clientes y se optimiza el retorno de las campañas promocionales. En el futuro, la app podría incluso integrarse con estos actores para potenciar alianzas estratégicas.

Base para funcionalidades futuras

La arquitectura modular de SaveApp, con captura automática de beneficios, análisis semántico y API GraphQL de mono-endpoint, permite que la app evolucione de “ahorro para usuarios” a una herramienta importante para bancos y comercios, complementando sus canales existentes sin cambiar sus procesos internos.

- **Medición de efectividad de promociones:** SaveApp cierra el loop entre impresión → interés → visita → uso declarado/reintegro esperado, generando KPIs como tasa de visualización y de uso por banco, tarjeta, rubro, local, zona y franja horaria. Permite atribución de campañas, detección de canibalización/saturación, análisis de cohortes y experimentación A/B con reglas simples (como vigencia, tope, días). Todo bajo privacy-by-design, sin datos sensibles, con seudonimización y consentimiento.

- **Recomendaciones y promociones personalizadas:** El motor de ranking combina afinidad (historial y preferencias), contexto (geolocalización y horario) y probabilidad de uso, para mostrar solo lo relevante y proponer bonificaciones dinámicas (como por ejemplo “martes sin tope en tu zona”) que maximizan el lift por segmento.
- **Espacio de promoción para locales:** A largo plazo, SaveApp puede ofrecer un portal para comercios y bancos que permita carga de promos, promociones destacadas etiquetadas, segmentación por radio/horarios y presupuesto controlado, con límites de frecuencia y reportes en tiempo real.

En conjunto, esto posiciona a SaveApp como plataforma viva que entrega valor medible, los bancos incrementan el uso de sus tarjetas, los comercios ganan tráfico incremental y los usuarios reciben ahorros pertinentes sin fricción.

Inclusión digital y democratización del ahorro

SaveApp permite que cualquier persona con un smartphone, incluso sin conocimientos técnicos o financieros avanzados, pueda acceder a información clara, personalizada y accionable sobre sus beneficios. Esto democratiza el acceso al ahorro y promueve la inclusión financiera, cerrando brechas de conocimiento y aprovechamiento entre distintos segmentos de la población.

IMPACTOS

La implementación de SaveApp genera un impacto integral en múltiples dimensiones del ecosistema digital, financiero y social. A través de su enfoque centrado en la automatización, la personalización y el empoderamiento del usuario, se esperan repercusiones concretas a nivel económico, social, ambiental y tecnológico.

Impacto Económico

Ahorro directo para los usuarios

Uno de los beneficios más tangibles de SaveApp es el ahorro monetario real que genera para sus usuarios. Al centralizar, clasificar y notificar promociones que de otro modo pasarían desapercibidas, la aplicación permite que cada persona optimice su poder adquisitivo.

Mayor efectividad en campañas bancarias y comerciales

Para bancos emisores de tarjetas y comercios adheridos, SaveApp incrementa el retorno sobre la inversión de sus campañas de beneficios. Al facilitar el acceso y comprensión de las condiciones, la tasa de uso de los descuentos mejora considerablemente. Esto se traduce en mayor volumen de transacciones, fidelización del cliente y un uso más inteligente del presupuesto promocional.

Estímulo a la economía digital

La aplicación fomenta el uso de medios de pago electrónicos y herramientas digitales. Este hábito beneficia la formalización de la economía, reduce el uso de efectivo y potencia la trazabilidad de las transacciones, un aspecto fundamental para la evolución del sistema financiero nacional.

Reducción de costos operativos

Al automatizar la obtención, análisis y presentación de beneficios, se eliminan tareas que tradicionalmente requerían intervención humana o inversiones en call centers, newsletters y materiales gráficos. Esto genera eficiencia no solo para los usuarios, sino también para bancos, fintechs y comercios.

Impacto Social

Inclusión financiera y digital

SaveApp promueve la democratización del acceso a la información financiera, facilitando que personas con baja alfabetización digital o bancaria comprendan y aprovechen beneficios disponibles. Al traducir el lenguaje técnico en información simple, personalizada y contextual, reduce barreras de entrada al sistema financiero formal y fomenta mayor participación económica.

Reducción de la asimetría de información

La oferta de beneficios suele presentarse dispersa y con términos complejos. SaveApp funciona como mediador tecnológico entre el ciudadano y las instituciones: integra fuentes, normaliza condiciones y simplifica términos y restricciones, brindando transparencia y mejorando la capacidad del usuario para comparar opciones y reclamar cuando corresponde.

Dimensión solidaria y comunitaria

En articulación con Fundación Effetá, la app incorpora un módulo de donaciones voluntarias para canalizar parte de los ahorros hacia programas educativos y sociales. El ahorro individual se convierte así en impacto colectivo, fortaleciendo el entramado comunitario y el fin social del proyecto.

SaveApp integra docencia, extensión e investigación alrededor de un problema social concreto y prioriza el impacto social dentro de las áreas de RSU de la UCC [1].

Impacto Medioambiental

Disminución del uso de materiales impresos

Tradicionalmente, muchos beneficios y promociones eran comunicados a través de folletos, cartelería o tickets impresos. La digitalización total de esta información mediante SaveApp reduce la necesidad de imprimir materiales, lo que tiene un impacto directo en la reducción del consumo de papel y tinta.

Optimización de desplazamientos

Al permitirle al usuario planificar sus compras con antelación y según ubicación, la app reduce desplazamientos innecesarios, evitando viajes duplicados o recorridos ineficientes. Esto contribuye indirectamente a la reducción de la huella de carbono asociada al transporte urbano.

Promoción del consumo digital responsable

SaveApp hace visibles promociones antes dispersas y potencia su aprovechamiento. Dado que la mayoría de estos beneficios se activan con medios de pago digitales o tarjetas, la app desalienta el uso de efectivo y elimina tickets y vouchers impresos. El resultado es una economía más desmaterializada que, desde la tecnología, contribuye a la sostenibilidad ambiental al reducir materiales, traslados y residuos asociados a soportes físicos.

CONCLUSIONES

La construcción de SaveApp representa mucho más que el desarrollo de una simple aplicación móvil: se trata de un proceso integral que articula necesidades reales del entorno financiero argentino, tecnologías de vanguardia, metodologías de diseño centradas en el usuario y una visión estratégica de impacto. A través del recorrido de este Proyecto Integrador, se logró transformar una problemática cotidiana —la dispersión, complejidad y desaprovechamiento de promociones bancarias— en una oportunidad concreta de mejora a nivel individual, social, económico y tecnológico.

Durante las distintas etapas del proyecto, desde la conceptualización inicial hasta la validación funcional de los módulos desarrollados, se confirmaron múltiples hipótesis clave:

- Que existe una gran cantidad de beneficios disponibles que no son aprovechados por desconocimiento o falta de acceso claro.[1]
- Que es posible automatizar la recolección, interpretación y contextualización de esta información mediante scraping, inteligencia artificial y geolocalización.
- Que una interfaz bien diseñada, acompañada de funcionalidades inteligentes, puede cambiar radicalmente la forma en que las personas se relacionan con sus finanzas cotidianas.
- Que el desarrollo de tecnología de impacto no requiere necesariamente grandes estructuras corporativas, sino visión, metodología y compromiso.

El sistema desarrollado demuestra la capacidad de generar valor real: mejora la toma de decisiones del consumidor, reduce la fricción para aprovechar promociones, democratiza el acceso a la información financiera y posiciona al usuario como protagonista activo de su ahorro.

Desde el punto de vista técnico, el proyecto implicó enfrentar y superar desafíos relevantes, como la falta de estándares abiertos en Argentina (Open Banking), la necesidad de extraer informacion de textos legales ambiguos, o las limitaciones impuestas por plataformas móviles como iOS en cuanto a PWA. Cada decisión tecnológica tomada fue evaluada, justificada y alineada con los objetivos funcionales del sistema.

Desde el punto de vista metodológico, se aplicaron criterios de arquitectura modular, desarrollo iterativo y enfoque en la escalabilidad, lo que habilita a SaveApp no solo a cumplir su propósito actual, sino a evolucionar hacia nuevas funcionalidades y mercados.

Como resultado, se construyó una plataforma robusta, extensible y con una propuesta de valor diferenciadora: inteligencia financiera accesible para todos. *Una herramienta que, sin depender de bancos, sin almacenar datos sensibles y sin exigir grandes conocimientos técnicos, permite al usuario común tomar decisiones informadas y beneficiarse de algo que ya le pertenece: sus descuentos.*

ANEXOS

A continuación se incluyen materiales relevantes para la comprensión, validación y reproducción del proyecto. Estos anexos complementan y respaldan los resultados presentados en el cuerpo principal del informe, y ofrecen una base sólida para la continuidad y evolución futura del proyecto SaveApp.

Bibliografía y Fuentes Consultadas

- Documentación oficial de tecnologías utilizadas:
    - Flutter - https://docs.flutter.dev
    - FlutterFlow - https://docs.flutterflow.io
    - GraphQL - https://graphql.org/learn
    - MongoDB - https://www.mongodb.com/docs
    - Gemini API (Google AI) - https://ai.google.dev/gemini-api/docs
    - PromptLayer - https://docs.promptlayer.com
    - Firebase - https://firebase.google.com/docs

- Sitios oficiales de bancos argentinos relevados para el análisis de beneficios:
    - Ualá - https://www.uala.com.ar/
    - Burbank - https://www.brubank.com/
    - NaranjaX - https://www.naranjax.com/
    - Macro - https://www.macro.com.ar/
    - Santander - https://www.santander.com.ar/personas
    - Galicia - https://www.galicia.ar/personas

- Fuentes consultadas sobre las problemáticas mencionadas:

1. CreditCards.com & YouGov. (2023). Survey on unredeemed rewards. https://www.creditcards.com/statistics/unused-credit-card-rewards-poll}

2. CardRates. (2025). Statistics on unused credit card rewards. https://www.cardrates.com/news/credit-card-reward-redemption-statistics

3. J.D. Power. (2025). U.S. Credit Card Satisfaction Study. The Financial Brand. https://thefinancialbrand.com/news/payments-trends/card-satisfaction-muted-by-surcharges-rewards-confusion-and-debt-levels-192110

4. Clarion Ledger. (2018). Credit card rewards programs confuse Americans; many leaving money on the table. https://www.clarionledger.com/story/news/2018/04/20/credit-card-rewards-programs-confuse-americans-many-leaving-money-table/530623002

5. El Economista. (2023). Conozca los beneficios de su tarjeta de crédito. https://www.eleconomista.com.mx/finanzaspersonales/Cual-es-la-tarjeta-de-credito-que-mas-te-conviene-20230619-0087.html

6. Comisión Nacional para la Protección y Defensa de los Usuarios de Servicios Financieros (CONDUSEF). (2023). Proteja su Dinero – Beneficios de la tarjeta. https://revista.condusef.gob.mx/credito/2023/05/beneficios-al-pagar-con-tu-tdc

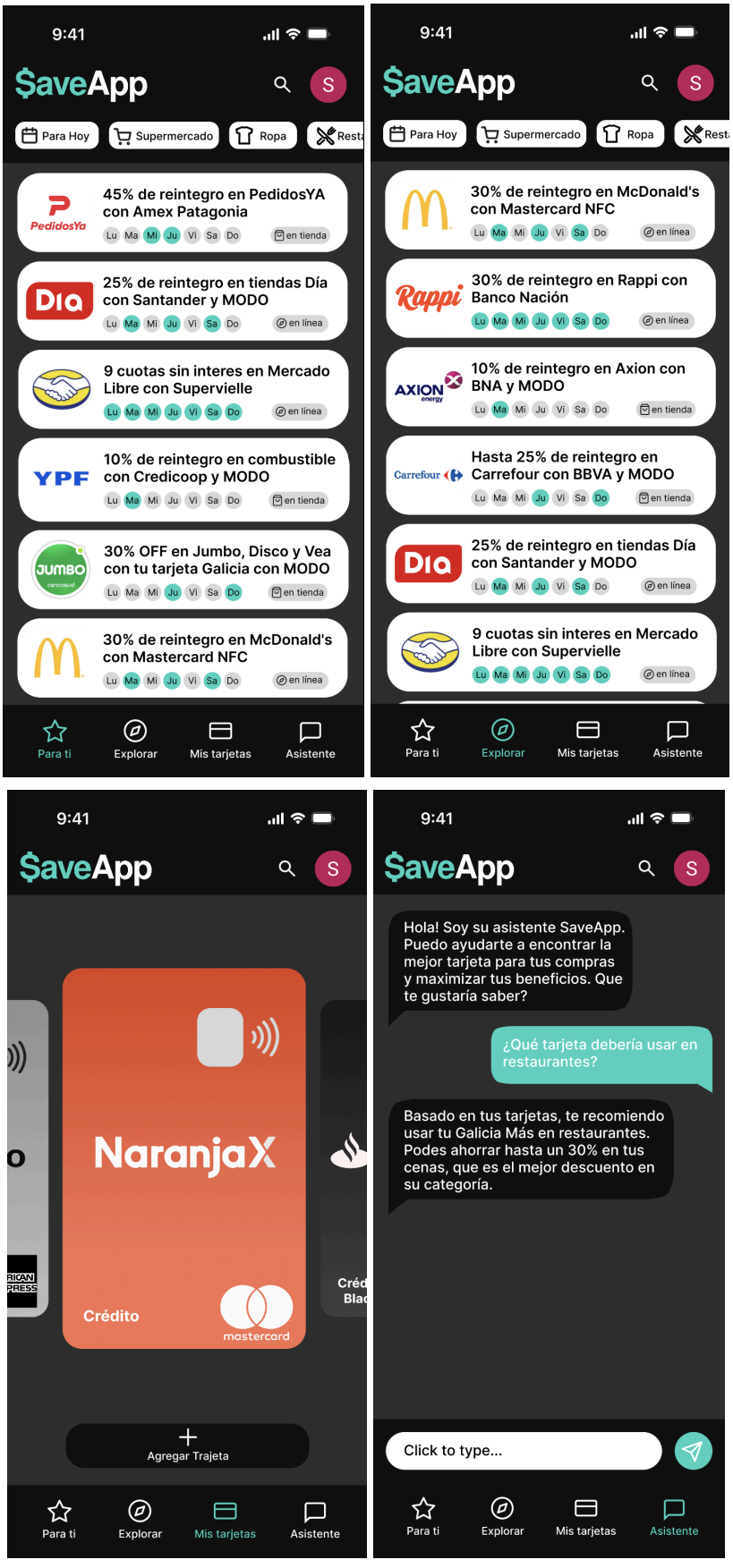
7. Consumer Reports. (2022). Beneficios ocultos de las tarjetas de crédito. https://www.consumerreports.org/es/dinero/tarjetas-de-credito/beneficios-de-tarjetas-que-no-conozcas-a1208189105

8. La FM. (2023). Beneficios de las tarjetas de crédito que los usuarios desconocen. https://www.lafm.com.co/economia/tarjeta-de-credito-los-beneficios-que-muchos-no-conocen

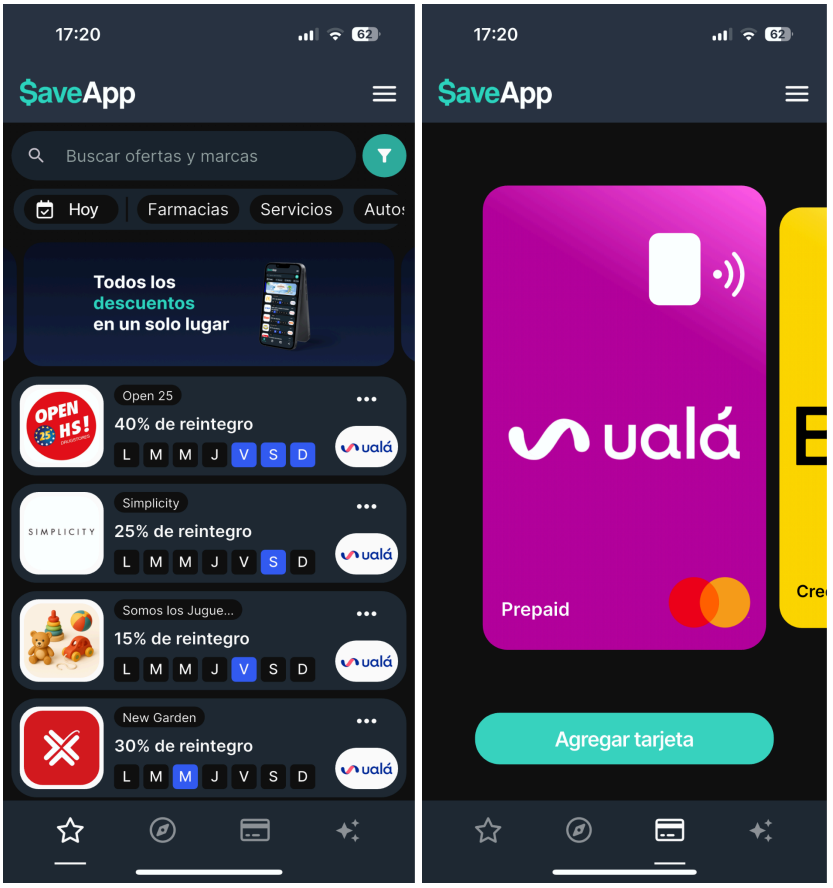
9. Chócale. (2025). Errores comunes al usar la tarjeta de crédito. https://chocale.cl/2025/03/las-trampas-y-errores-al-usar-mal-las-tarjetas-de-credito

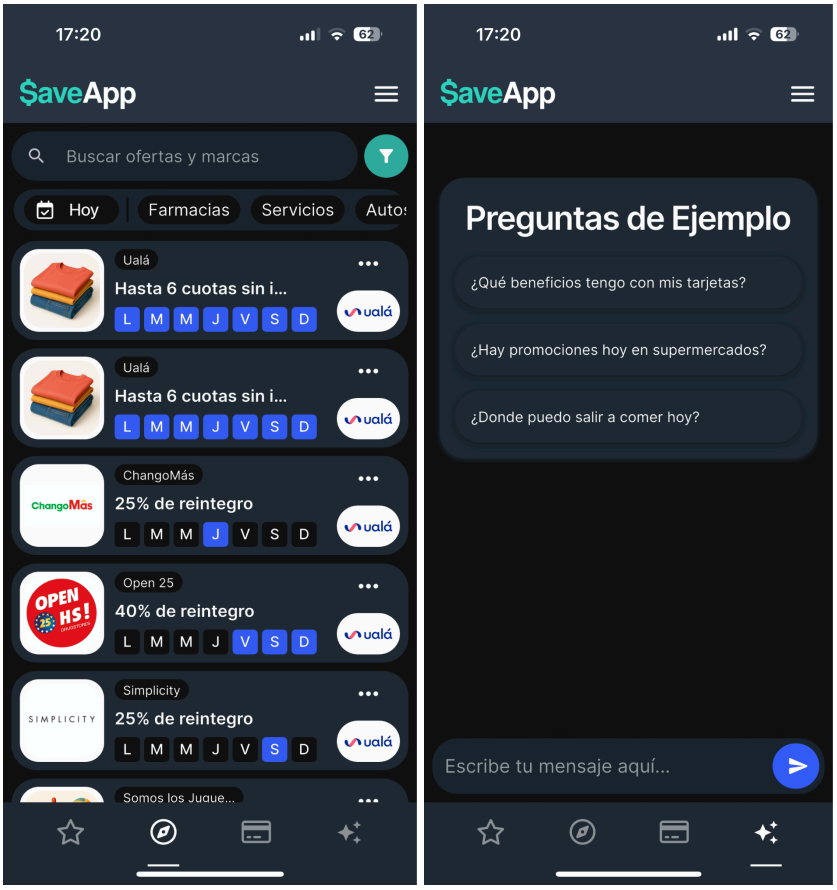
10. Semana. (2023). Los beneficios de la tarjeta de crédito que pocos utilizan. https://www.semana.com/finanzas/consumo-inteligente/articulo/los-10-beneficios-en-su-tarjeta-de-credito-que-poco-o-nunca-utiliza/202304

Mockups iniciales (Figma)

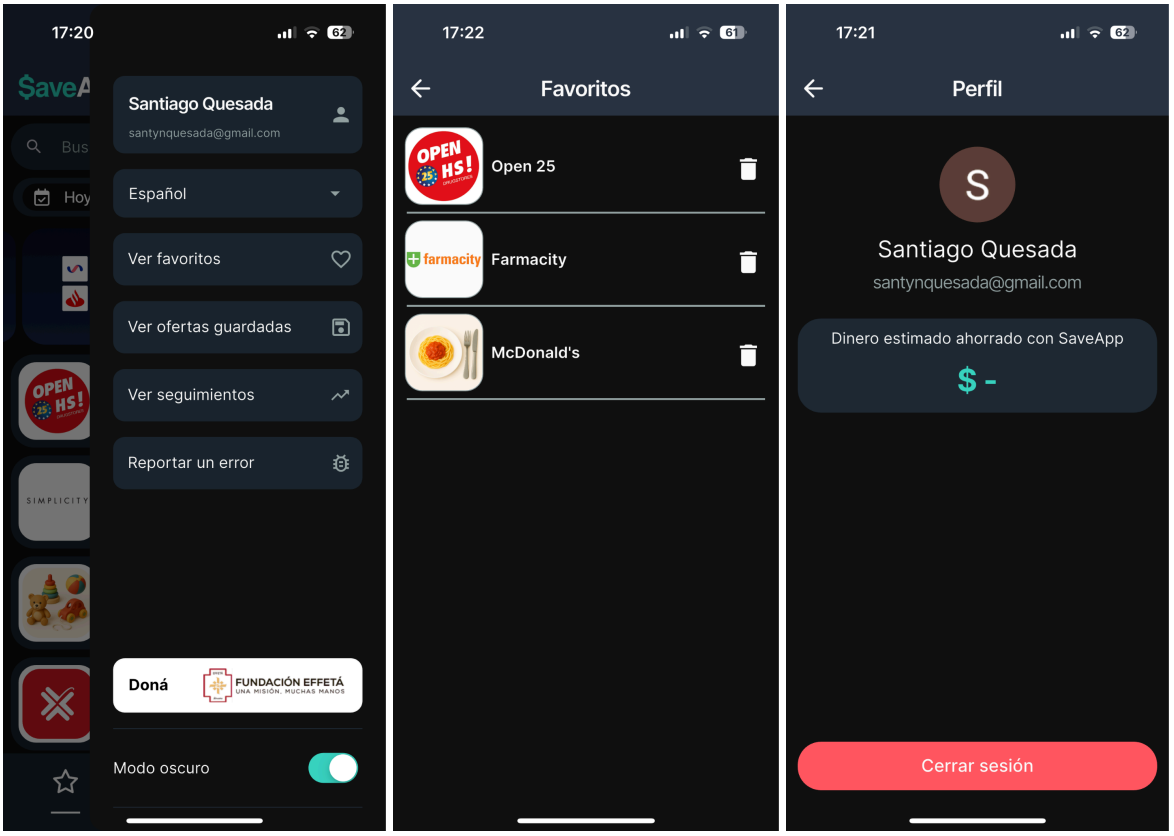


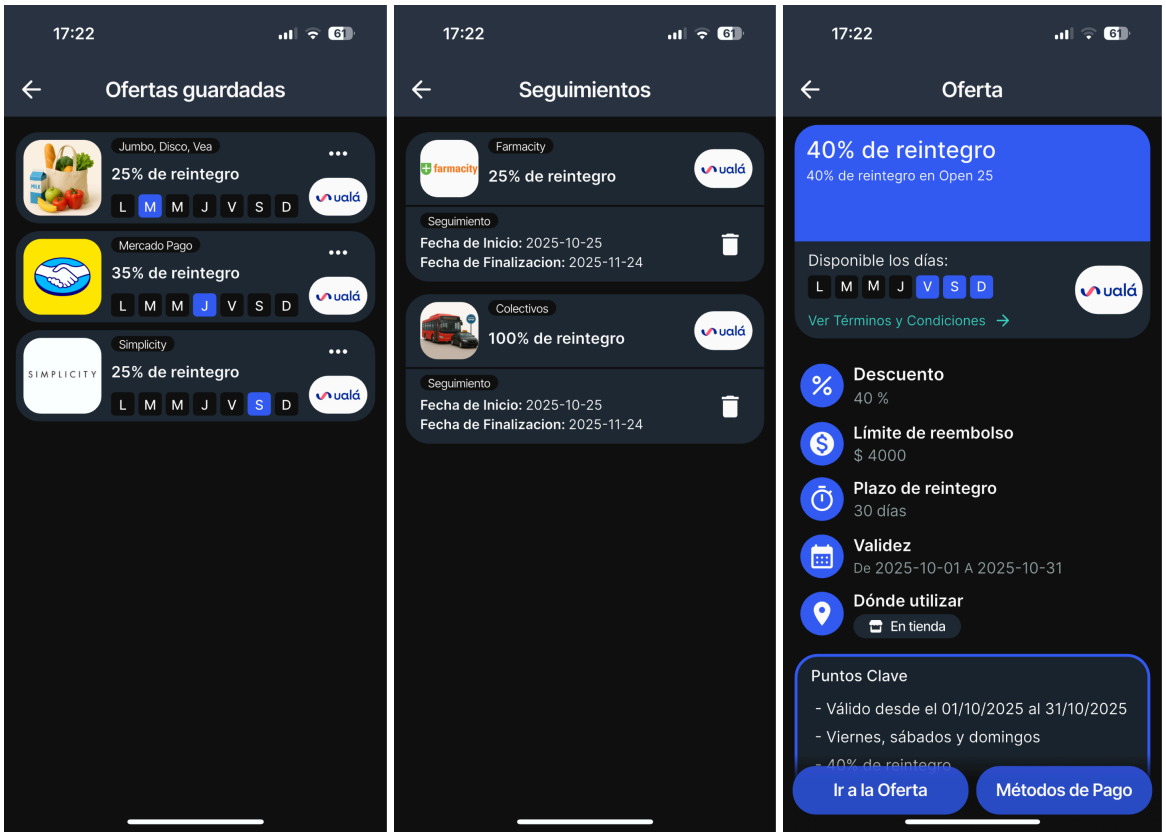
Pantallas Principales





Pantallas Secundarias





Enfoque RSU

SaveApp es el trabajo final de grado de tres estudiantes de Ingeniería en Sistemas de la Universidad Católica de Córdoba (UCC). Se trata de una aplicación móvil con enfoque fintech que integra web-scraping, geolocalización, inteligencia artificial y un chat asistente para reunir, simplificar y notificar en tiempo real los descuentos y promociones de comercios de uso cotidiano como supermercados, farmacias y transporte público.

El proyecto nace como respuesta a una asimetría de información que afecta especialmente a los sectores con menor alfabetización financiera: las promociones están dispersas en múltiples canales, redactadas con términos complejos y rara vez disponibles en el punto de venta. Esta dificultad reduce la posibilidad de ahorrar y genera una pérdida directa de recursos. SaveApp se propone resolver este problema al procesar la información, automatizar su procesamiento y comunicársela al usuario justo en el momento de pagar.

SaveApp se desarrolló en colaboración con la Fundación Effetá, organización social dedicada a la promoción social y educativa. La integración de un módulo de donaciones permite que los usuarios destinen parte de sus ahorros a fortalecer los programas de la Fundación, ampliando su alcance comunitario. De este modo, la dimensión solidaria se convierte en un eje central del proyecto, al transformar el beneficio individual del ahorro en un recurso sostenible para acciones de impacto social.

El proyecto fue validado por aproximadamente 50 personas que participaron en las encuestas iniciales, confirmaron la utilidad de la aplicación y expresaron su interés en donar parte de sus ahorros. Asimismo, se mantuvo contacto con negocios de Córdoba que manifestaron su interés en visibilizar sus ofertas dentro de la app, fortaleciendo así el vínculo entre la comunidad académica, el sector comercial y la sociedad.

Esta iniciativa se alinea con el paradigma de Responsabilidad Social Universitaria promovido por la universidad al integrar docencia, extensión e investigación en el abordaje de un problema social concreto. SaveApp impacta en las cinco áreas definidas por la Secretaría de Proyección y RSU de la UCC, priorizando especialmente el impacto social, aporta un beneficio tangible a la comunidad al permitir un ahorro económico directo, visibiliza las ofertas de pequeños comercios y canaliza donaciones hacia la Fundación Effetá.

El enfoque RSU se integra transversalmente en los objetivos y resultados del proyecto, convirtiendo la innovación tecnológica en un servicio social útil, medible, escalable y con alto potencial de replicabilidad, en línea con los criterios para su acreditación como Trabajo Final con Enfoque RSU.