**Instituto Puig Castellar**
**Ciclo formativo:** DAM
**Grado:** CFGS
Crédito de Síntesis / Proyecto intermodular


# 2Look

App móvil de asesoramiento estético e inteligencia artificial para barberías

**Autor/a/es:** Iván Martín Bodas
**Tutor:** Luis Elía Talón
**Curso:** 2025-2026
**Fecha de entrega:** 15/05/2026

## Resumen del proyecto

2LOOK nace de una observación cotidiana: casi todo el mundo ha salido alguna vez del barbero sin estar del todo contento, no porque el barbero lo haya hecho mal, sino porque la elección del corte fue a ciegas. La app resuelve exactamente eso: actúa como un asesor personal que analiza la forma del rostro mediante inteligencia artificial y recomienda los tres cortes que mejor le quedan al usuario antes de sentarse en la silla.

Técnicamente el proyecto integra tres capas. El frontend está construido con Ionic y Vue 3, lo que permite que funcione tanto en navegador como en Android. El backend corre sobre Spring Boot con una base de datos PostgreSQL alojada en Aiven Cloud. La inteligencia artificial se apoya en la API de Gemini, que analiza la foto del usuario, detecta la morfología facial y devuelve los tres cortes recomendados con imágenes de referencia generadas por fal.ai.

El flujo es directo: el usuario se registra o entra como invitado, ve los barberos disponibles en tiempo real, se hace una foto para obtener la recomendación de la IA, elige barbero, fecha y hora, y la cita queda guardada en la base de datos. También puede saltarse la IA y elegir el corte manualmente.

Para gestionar el trabajo se ha usado Kanban, ideal para un proyecto individual. El coste total es cero, aprovechando las opciones gratuitas de Gemini, Aiven, fal.ai e Ionic.

## Palabras clave

Aplicación móvil, Inteligencia artificial, Ionic + Vue 3, Spring Boot, Reserva online, Morfología facial, Base de datos en la nube, Kanban.

---

## Abstract

Project Summary

2LOOK was born from an everyday observation: almost everyone has left the barbershop at some point not entirely happy, not because the barber did a bad job, but because the choice of haircut was made blindly. The app solves exactly that: it acts as a personal advisor that analyzes the shape of the user's face using artificial intelligence and recommends the three haircuts that will suit them best before they even sit down in the chair.

Technically the project integrates three layers. The frontend is built with Ionic and Vue 3, allowing it to work on both browser and Android. The backend runs on Spring Boot with a PostgreSQL database hosted on Aiven Cloud. The artificial intelligence relies on the Gemini API, which analyzes the user's photo, detects facial morphology and returns the three recommended haircuts with reference images generated by fal.ai.

The flow is straightforward: the user registers or enters as a guest, sees the available barbers loaded in real time from the backend, takes a photo to get the AI recommendation, chooses a barber, date and time, and the appointment is saved in the database. They can also skip the AI and choose the haircut manually.

Kanban was used to manage the work, ideal for an individual project. The total cost is zero, taking advantage of the free tiers of Gemini, Aiven, fal.ai and Ionic.

## Keywords

Mobile application, Artificial intelligence, Ionic + Vue 3, Spring Boot, Online booking, Facial morphology, Cloud database, Kanban.

# Índice

# 1. Presentación del proyecto

## 1.1 Introducción

La idea de 2LOOK surgió de algo que le pasa a casi todo el mundo: te sientas en la barbería, el barbero te pregunta qué quieres y no tienes ni idea. O le enseñas una foto de algún famoso que has visto en el móvil, o le dices directamente "hazme lo que tú veas". El problema no es el barbero — el problema es que la mayoría de la gente elige un corte simplemente porque está de moda, sin saber si realmente le favorece según la forma de su cara o el tipo de pelo que tiene.

La tecnología había transformado por completo la forma en que se gestionan casi todos los aspectos de la vida cotidiana, pero cuando se trataba de algo tan personal como cambiar de imagen, la gente seguía yendo a ciegas. Hay aplicaciones para controlar lo que comes, cuánto duermes o cuántos pasos das, pero nada que te diga de forma sencilla y fiable qué corte te va a quedar bien antes de cortártelo. La mayoría de las personas ni siquiera sabe qué forma de cara tiene, y acaba pidiendo estilos que no le favorecen en absoluto.

Además, si se analizan las aplicaciones de barbería que ya existen, casi todas son simplemente herramientas de reserva online. Ninguna ayuda al usuario a decidir qué estilo le queda mejor antes de ir. Ese hueco fue el punto de partida de 2LOOK.

Desde el punto de vista técnico, este proyecto ha sido la oportunidad perfecta para aplicar todo lo aprendido durante el ciclo de DAM y llevarlo a algo real y funcional. Se ha construido una aplicación móvil completa de principio a fin: desde el diseño de la base de datos y la API REST con Spring Boot, hasta la interfaz móvil con Ionic y Vue 3, pasando por la integración de inteligencia artificial con Gemini para el análisis facial y fal.ai para la generación de imágenes de referencia. El reto más grande ha sido conseguir que todas estas tecnologías funcionen juntas de forma fluida, y el resultado es una herramienta que aporta valor real tanto al cliente como al profesional.

## 1.2 Contexto

El mundo de las barberías ha cambiado muchísimo en los últimos años. Antes íbamos a la peluquería del barrio, nos cortaban el pelo y ya está. Ahora las barberías son otra cosa: cuidan cada detalle, tienen su propia identidad, usan productos específicos y el cliente llega con expectativas mucho más altas. El cuidado personal masculino ha dejado de ser algo que se hacía por obligación para convertirse en algo que mucha gente disfruta y valora de verdad.

Pero hay algo que, con todo ese cambio, sigue igual: la forma de elegir el corte. Entras, te sientas, y o le enseñas una foto al barbero o le dices "tú mismo". No hay nada que te ayude a decidir antes de llegar ahí. Y eso es un problema más gordo de lo que parece, porque la forma de la cara importa muchísimo. Lo que le queda genial a alguien con cara ovalada puede quedar bastante mal en alguien con cara redonda o cuadrada. Los barberos buenos lo saben, pero el cliente de a pie no tiene ni idea, y al final la elección es casi siempre una apuesta.

Lo curioso es que hoy en día todo el mundo lleva en el bolsillo un móvil con una cámara decente y acceso a inteligencia artificial, y sin embargo nadie había aprovechado eso para ayudar a la gente a elegir su corte. Hay apps para reservar cita en la barbería, sí, pero ninguna que te diga antes "oye, con tu tipo de cara este corte te va a quedar mucho mejor que este otro".

Ahí es exactamente donde encaja 2LOOK. Un sector que está creciendo, un problema que nadie había resuelto, y la tecnología necesaria para solucionarlo ya disponible y al alcance de cualquiera.

## 1.3 Justificación

La razón principal por la que existe 2LOOK es sencilla: la gente no sale siempre contenta de la barbería, y muchas veces no es culpa del barbero sino de que el cliente eligió mal. Ese problema tiene solución tecnológica y nadie la había implementado de forma accesible y práctica.

El objetivo era construir algo que tuviera una lógica propia, que resolviera un problema real y que demostrara que es posible desarrollar una aplicación completa y funcional desde cero. No solo que funcione, sino que tenga sentido usarla.

Técnicamente, el proyecto justifica el uso de cada tecnología elegida. Ionic y Vue 3 porque se necesitaba una app móvil que funcione en Android e iOS sin tener que desarrollar en nativo. Spring Boot porque el backend necesitaba ser robusto, bien estructurado y fácil de escalar. Aiven porque la base de datos tenía que estar en la nube y accesible en todo momento. Gemini porque es la API de visión artificial más potente disponible de forma gratuita ahora mismo. Y fal.ai porque generar imágenes de referencia de los cortes añade un valor visual que hace la experiencia mucho más completa.

Al final, 2LOOK existe porque la tecnología tiene que servir para cosas del día a día. Ayudar a alguien a verse mejor, a ir al barbero con más confianza y a tomar una decisión informada sobre su imagen es algo pequeño pero concreto y real.

## 1.4 Objetivos

### 1.4.1 Objetivos Generales

- Desarrollar una aplicación móvil completa que combine el asesoramiento estético mediante inteligencia artificial con un sistema de gestión de reservas en barberías, ofreciendo al usuario una experiencia útil, moderna y fácil de usar de principio a fin.
- Conseguir que la tecnología resuelva un problema real y cotidiano: que cualquier persona pueda saber qué corte de pelo le favorece antes de ir al barbero, sin necesidad de tener conocimientos de estética ni de moda.
- Demostrar la capacidad de construir un sistema completo e integrado, uniendo frontend móvil, backend con API REST, base de datos en la nube e inteligencia artificial en un único producto funcional.

### 1.4.2 Objetivos Específicos

- Implementar un sistema de análisis facial usando la API de Gemini que detecte la forma de cara del usuario a partir de una foto y devuelva los tres cortes más recomendados de forma automática.
- Integrar fal.ai para generar imágenes de referencia de cada corte recomendado, dando al usuario una representación visual clara de cómo podría quedarle.
- Desarrollar un sistema de reservas completo donde el usuario pueda elegir barbero, servicio, fecha y hora, y la cita quede guardada en la base de datos en tiempo real.
- Construir un backend sólido con Spring Boot que gestione usuarios, barberos, servicios, reservas, favoritos, opiniones e historial de cortes a través de una API REST.
- Garantizar la persistencia y seguridad de los datos usando PostgreSQL en la nube con Aiven, asegurando que toda la información esté siempre disponible y protegida.
- Diseñar una interfaz limpia e intuitiva con Ionic y Vue 3, apoyándose en Figma para crear las pantallas antes del desarrollo, que permita al usuario navegar, reservar y usar el asesor de IA sin necesidad de ningún tipo de explicación previa.

# 2. Estrategia y planificación

## 2.1 Estrategia de desarrollo y viabilidad

Desde el principio quedó claro que el mayor riesgo del proyecto era intentar construirlo todo a la vez. Una app móvil, un backend, una base de datos en la nube y una integración con inteligencia artificial son cuatro elementos que por separado ya tienen su complejidad, así que la estrategia fue construir de dentro hacia fuera: primero que el servidor funcionara bien y guardara datos correctamente, luego que la app pudiera comunicarse con él, y por último añadir la capa de IA una vez que todo lo demás era estable.

En cuanto a la viabilidad, el proyecto es viable en los tres aspectos que importan:

Técnica. Todas las tecnologías elegidas están ampliamente documentadas y tienen comunidades activas. Gemini es una de las APIs de visión artificial más potentes disponibles ahora mismo. Spring Boot es un estándar en el desarrollo de backends Java. Ionic con Vue 3 es una combinación sólida para apps multiplataforma. No se ha inventado nada nuevo, se han combinado herramientas que funcionan.

Temporal. El proyecto se organizó en fases para que cada parte estuviera terminada y probada antes de pasar a la siguiente. Esto evitó que los problemas se acumularan y permitió ajustar prioridades cuando algo se complicaba más de lo esperado, como fue el caso de la integración entre la IA y el resto del sistema.

Económica. El coste total del proyecto es cero. Gemini tiene tier gratuito con 1.500 peticiones diarias (con un límite de 15 por minuto), más que suficiente para desarrollo y demos. Aiven ofrece una capa gratuita para PostgreSQL. fal.ai proporciona créditos gratuitos al registro. Ionic y Vue 3 son open source. No fue necesario invertir nada para construir una aplicación que en un contexto real tendría un coste de infraestructura mensual muy asumible.

## 2.2 Metodología de trabajo

Para el desarrollo de 2LOOK no se siguió un plan rígido de principio a fin. Se optó por una metodología ágil basada en Kanban, una decisión que resultó clave para la buena gestión del proyecto.

Kanban funciona con un tablero visual donde las tareas pasan por tres estados: pendiente, en progreso y terminado. Para implementarlo se utilizó Trello, una herramienta online que permite crear y gestionar este tipo de tableros de forma visual e intuitiva. Cada funcionalidad del proyecto se representaba como una tarjeta dentro del tablero, con su descripción y estado actualizado en todo momento. Esto permitía tener siempre una visión clara de qué estaba hecho, qué estaba en desarrollo y qué quedaba por abordar.

En lugar de tener una lista interminable de cosas por hacer, se definían tareas concretas y pequeñas que podían completarse en una sesión de trabajo. Eso mantuvo la motivación alta y generó una sensación real de avance constante a lo largo de todo el desarrollo.

La clave fue dividir el proyecto en módulos funcionales e ir construyendo y probando cada uno antes de pasar al siguiente. Primero el backend y la base de datos, luego la conexión con el frontend, después el sistema de reservas, y por último la integración con la IA. Cada módulo se probaba a fondo antes de continuar, lo que permitió detectar errores de forma temprana, cuando todavía eran fáciles de corregir.

Otro aspecto importante fue la priorización. No todas las funcionalidades tienen el mismo peso, y con tiempo limitado hay que ser honesto sobre qué es imprescindible y qué es un extra. La IA, el sistema de reservas y la conexión con el backend eran innegociables. Otras funcionalidades como los pagos reales o las notificaciones push quedaron identificadas como mejoras futuras sin que eso comprometiera el resultado final.

En definitiva, Kanban con Trello permitió mantener el orden sin perder la flexibilidad, que es exactamente lo que se necesita cuando se desarrolla en solitario y las circunstancias cambian sobre la marcha.

## 2.3 Planificación

La planificación del proyecto se dividió en cinco fases bien diferenciadas, cada una con un objetivo claro y unos entregables concretos antes de pasar a la siguiente.

**Fase 1 — Análisis y diseño del sistema**

Antes de escribir una sola línea de código se definió qué tenía que hacer la aplicación y cómo tenía que estar estructurada. Se identificaron los casos de uso principales, se diseñó el esquema de la base de datos con sus siete tablas y se eligieron las tecnologías que mejor encajaban con los requisitos del proyecto. También se esbozaron las pantallas principales para tener claro el flujo de navegación antes de empezar a construirlo.

**Fase 2 — Backend y base de datos**

Con el diseño claro, el primer paso fue montar el servidor. Se configuró Spring Boot, se conectó con la base de datos PostgreSQL en Aiven Cloud y se implementaron todos los endpoints REST necesarios: personas, barberos, servicios, reservas, favoritos, opiniones e historial. También se añadió la configuración de seguridad y CORS para que el frontend pudiera comunicarse con el backend sin problemas. Esta fase fue la más importante porque todo lo demás dependía de que el servidor funcionara bien.

**Fase 3 — Frontend móvil**

Con el backend estable se comenzó a construir la interfaz con Ionic y Vue 3. Se desarrollaron todas las pantallas de la aplicación: login, registro, home, búsqueda, perfil de barbero, reservas, historial, citas y perfil de usuario. En esta fase también se conectó el frontend con el backend real, sustituyendo los datos hardcodeados por llamadas a la API.

**Fase 4 — Integración de la IA**

La fase más compleja del proyecto. Se implementó la pantalla de cámara con acceso real al hardware del dispositivo, se integró la API de Gemini para el análisis facial y se añadió fal.ai para la generación de imágenes de referencia de cada corte recomendado. Se trabajó especialmente en la gestión de errores y en el sistema de rotación automática entre claves de API de Gemini para garantizar que la funcionalidad estuviera siempre disponible dentro de los límites del tier gratuito.

**Fase 5 — Pruebas y ajustes finales**

La última fase se dedicó a probar el sistema completo de principio a fin. Se verificó que el flujo de reserva funcionara correctamente, que los datos se guardaran bien en la base de datos, que la IA respondiera de forma coherente a distintos tipos de fotos y que la navegación entre pantallas fuera fluida y sin errores. También se realizaron ajustes de diseño y se refinaron detalles de usabilidad detectados durante las pruebas.

# 3. Análisis

## 3.1 Casos de uso

A continuación se describen las situaciones principales en las que un usuario interactúa con la aplicación, cubriendo los flujos más importantes desde que abre la app hasta que completa una acción.

**Registro e inicio de sesión**

El usuario abre la aplicación y puede crear una cuenta nueva introduciendo su nombre, email y contraseña, o bien entrar directamente como invitado sin necesidad de registrarse. Si ya tiene cuenta, accede con su email y contraseña. Los datos se validan en el frontend antes de enviarse al backend, que comprueba que el email no esté ya registrado y devuelve la sesión del usuario si todo es correcto.

**Asesoramiento estético con IA**

El usuario accede a la pestaña de cámara, se hace una foto o sube una imagen desde la galería, y el sistema la envía a la API de Gemini. En pocos segundos la aplicación detecta la forma de cara del usuario, muestra una descripción de sus características y presenta los tres cortes que mejor le van a quedar, cada uno con su nombre, una explicación de por qué le favorece y una imagen de referencia generada por fal.ai. Desde esta pantalla el usuario puede reservar directamente con el corte recomendado por la IA.

**Búsqueda y consulta de barberos**

El usuario puede explorar los barberos disponibles desde la pantalla principal o usar el buscador para encontrar uno concreto. Al entrar en el perfil de un barbero puede ver su

especialidad, valoración, galería de cortes realizados y la evolución de sus puntuaciones en los últimos meses. Si le interesa, puede marcarlo como favorito para tenerlo siempre a mano.

**Reserva de cita**

El usuario selecciona un barbero, elige el servicio que quiere realizarse ya sea de forma manual desde el listado o a través de la recomendación de la IA, escoge una fecha en el calendario y selecciona un horario disponible. La cita se guarda en la base de datos y aparece en la sección de citas del usuario con todos los detalles. Desde ahí puede gestionarla, cancelarla o proceder al pago.

**Gestión de citas y pago**

Desde la pestaña de citas el usuario puede ver su reserva activa con el nombre del barbero, el servicio, la fecha y la hora. Si aún no ha pagado, puede hacerlo seleccionando su método de pago preferido entre wallet, Google Pay o tarjeta de crédito. Una vez completado el pago puede activar un recordatorio para que la aplicación le notifique antes de la cita.

**Historial y valoraciones**

Después de cada servicio, el usuario puede consultar su historial completo de cortes con la fecha, el barbero y la hora de cada visita. También puede dejar una reseña valorando el servicio con estrellas y un comentario, que quedará visible para otros usuarios en el perfil del barbero.

**Gestión del perfil**

El usuario puede editar su información personal, cambiar su foto de perfil, consultar sus favoritos, gestionar sus métodos de pago y cerrar sesión. Todas las opciones están accesibles desde la pestaña de perfil de forma clara y organizada.

## 3.2 Requisitos funcionales

Los requisitos funcionales definen qué debe ser capaz de hacer la aplicación. Se han organizado por módulos para facilitar su lectura.

**Autenticación y usuarios**

- La aplicación debe permitir el registro de nuevos usuarios con nombre, email y contraseña.
- El sistema debe validar que el email no esté ya registrado antes de crear la cuenta.
- El usuario debe poder iniciar sesión con email y contraseña y recibir su sesión correctamente.
- Debe existir la opción de entrar como invitado sin necesidad de crear una cuenta.
- La sesión debe persistir mientras el usuario no cierre la aplicación.
- El usuario debe poder cerrar sesión desde su perfil.

**Asesoramiento con IA**

- La aplicación debe acceder a la cámara del dispositivo para capturar una foto del usuario.
- La foto debe enviarse a la API de Gemini para analizar la morfología facial.
- El sistema debe devolver la forma de cara detectada junto con una descripción.
- Deben mostrarse los tres cortes más recomendados, cada uno con su nombre y justificación.

- Para cada corte recomendado debe generarse una imagen de referencia mediante fal.ai.
- Si una clave de API de Gemini supera su cuota o devuelve error, el sistema debe rotar automáticamente a la siguiente de las tres claves configuradas.

**Barberos y servicios**

- La pantalla principal debe cargar los barberos disponibles en tiempo real desde el backend.
- Debe poder consultarse el perfil completo de cada barbero con su especialidad, valoración y galería de cortes.
- La aplicación debe mostrar el listado de servicios disponibles cargado desde la base de datos.
- El usuario debe poder marcar barberos como favoritos y consultarlos desde su perfil.

**Reservas**

- El usuario debe poder elegir un barbero, un servicio, una fecha y una hora para su cita.
- El servicio puede seleccionarse manualmente o venir recomendado por la IA.
- La reserva debe guardarse en la base de datos con estado inicial pendiente.
- El usuario debe poder ver su cita activa con todos los detalles desde la pestaña de citas.
- Debe ser posible cancelar una reserva activa desde la misma pantalla.

**Pagos**

- La aplicación debe ofrecer al menos tres métodos de pago: wallet, Google Pay y tarjeta de crédito.
- Una vez completado el pago la reserva debe marcarse como pagada.

- El usuario debe poder activar un recordatorio de la cita tras confirmar el pago.

**Historial y opiniones**

- El usuario debe poder consultar su historial completo de cortes con fecha, barbero y hora.
- Debe poder dejar una valoración con estrellas y comentario para cada servicio recibido.
- Las valoraciones deben quedar guardadas en la base de datos y ser visibles en el perfil del barbero.

**Perfil de usuario**

- El usuario debe poder editar su nombre, email y foto de perfil.
- Debe poder consultar y gestionar sus favoritos desde el perfil.
- La aplicación debe permitir cerrar sesión de forma limpia eliminando los datos de sesión.

## 3.3 Requisitos no funcionales

Los requisitos no funcionales definen cómo debe comportarse el sistema, independientemente de lo que hace.

**Rendimiento**

- La aplicación debe cargar la pantalla principal y los barberos en menos de 3 segundos con conexión normal.

- El análisis de la IA debe completarse y mostrar resultados en un tiempo razonable, no superando los 10 segundos en condiciones normales de red. Las imágenes de referencia generadas por fal.ai se cargan de forma asíncrona en segundo plano sin bloquear la interfaz.
- Las imágenes generadas por fal.ai deben cargarse de forma asíncrona sin bloquear la interfaz.

**Usabilidad**

- La navegación debe ser intuitiva y no requerir ningún tipo de explicación previa para completar una reserva.
- El diseño debe ser coherente en todas las pantallas, manteniendo la misma paleta de colores, tipografía y estilo de componentes.
- La aplicación debe funcionar correctamente en dispositivos Android e iOS con pantallas de tamaño estándar.

**Disponibilidad**

- El backend debe estar disponible siempre que el servidor esté arrancado.
- La base de datos en Aiven Cloud garantiza disponibilidad continua dentro de los términos del tier gratuito.
- La aplicación debe degradarse de forma elegante cuando el backend no esté disponible, mostrando datos locales en lugar de errores en blanco.
- Para evitar que el servidor de Render entre en modo inactivo por falta de uso, se ha configurado UptimeRobot para realizar una petición cada 5 minutos al endpoint de barberos, garantizando así la disponibilidad continua del backend.

**Seguridad**

- Las contraseñas se almacenan en la base de datos tal como las introduce el usuario, siendo una mejora futura implementar cifrado con BCrypt.
- La API REST no expone endpoints sensibles sin control, gracias a la configuración de Spring Security.
- Las claves de API de Gemini y fal.ai no se exponen en repositorios públicos en un entorno de producción.

**Mantenibilidad**

- El código del frontend está organizado por páginas, componentes, store y router, facilitando la localización y modificación de cualquier parte.
- El backend sigue una arquitectura por capas: modelo, repositorio, servicio y controlador, lo que permite modificar la lógica de negocio sin afectar al resto del sistema.
- Todos los endpoints del backend siguen convenciones REST estándar.

**Escalabilidad**

- La arquitectura cliente-servidor permite escalar el backend de forma independiente al frontend.
- La base de datos está diseñada para soportar múltiples usuarios, barberos y reservas sin cambios estructurales.
- El sistema de rotación automática de claves de Gemini permite adaptarse a cambios en las cuotas sin modificar el código.

## 3.4 Análisis de alternativas tecnológicas

Antes de decidir el stack tecnológico final se valoraron distintas alternativas para cada capa del sistema.

**Diseño de interfaz**

La alternativa a Figma era usar Adobe XD o directamente diseñar en el propio código. Adobe XD requiere suscripción y Sketch solo está disponible en macOS. Figma en cambio es gratuito, funciona en el navegador sin instalación y permite prototipar pantallas interactivas de forma rápida. Se eligió Figma para diseñar todas las pantallas antes de empezar con Ionic y Vue 3, lo que permitió validar el flujo de navegación y la estética visual sin escribir una sola línea de código.

**Frontend**

La principal alternativa a Ionic con Vue 3 era React Native. React Native ofrece un rendimiento más cercano al nativo y una comunidad muy grande, pero requiere más configuración inicial y el desarrollo es más complejo para un proyecto individual con tiempo limitado. Flutter también se consideró, pero implica aprender Dart, un lenguaje que no forma parte del currículo de DAM. Se eligió Ionic con Vue 3 porque permite reutilizar conocimientos de desarrollo web, tiene una curva de aprendizaje más suave y genera una app Android funcional con el mismo código base.

**Backend**

La alternativa más directa a Spring Boot era Node.js con Express. Node.js es más ligero y rápido de configurar para proyectos pequeños, pero Spring Boot ofrece una estructura más sólida, mejor integración con bases de datos relacionales a través de JPA y es el estándar en entornos empresariales Java, que es precisamente lo que se trabaja en DAM. También se valoró Django con Python, descartado por la misma razón que Flutter: implicaba aprender un lenguaje fuera del currículo principal.

**Base de datos**

Se valoró usar MySQL en local frente a PostgreSQL en Aiven Cloud. MySQL en local es más simple de configurar inicialmente, pero limita el acceso a la base de datos a una sola máquina. PostgreSQL en Aiven resuelve ese problema al estar en la nube, es accesible desde cualquier entorno y el tier gratuito es más que suficiente para el volumen de datos del proyecto. Además PostgreSQL tiene mejor soporte para tipos de datos avanzados y es más robusto en general.

**Inteligencia artificial**

Las principales alternativas a Gemini eran OpenAI GPT-4 Vision y AWS Rekognition. GPT-4 Vision no tiene tier gratuito real para proyectos de desarrollo, lo que lo descartó desde el principio por razones económicas. AWS Rekognition es potente para detección facial pero está orientado a reconocimiento de identidad, no a análisis estético, y su integración es más compleja. Gemini ofrece capacidades de visión artificial de primer nivel, tiene tier gratuito con cuota diaria suficiente para desarrollo y demos, y su API es sencilla de integrar directamente desde el frontend.

**Generación de imágenes**

La alternativa a fal.ai era usar directamente la API de DALL-E de OpenAI o la de Midjourney. DALL-E genera imágenes de buena calidad pero tiene un coste por imagen y no ofrece una opción gratuita real. Midjourney directamente no expone una API pública. fal.ai en cambio da créditos gratuitos al registrarse, su API es muy sencilla de usar y permite acceder a varios modelos de Black Forest Labs, entre ellos FLUX.1 Kontext, que es el que finalmente se usa en la app porque permite editar la propia foto del usuario aplicando solo el cambio de peinado en lugar de generar una imagen desde cero. Para el propósito del proyecto, que es enseñar al usuario cómo le quedaría el corte recomendado sobre su propia cara, fal.ai cumple de maravilla.

# 4. Diseño

## 4.1 Arquitectura del sistema

2LOOK sigue una arquitectura cliente-servidor clásica dividida en tres capas bien diferenciadas que se comunican entre sí de forma ordenada.

El **frontend** es la capa con la que interactúa el usuario. Está construido con Ionic y Vue 3 y corre en el dispositivo del usuario (Android, iOS o navegador web). Se encarga de mostrar la interfaz, gestionar la navegación entre pantallas y comunicarse con el backend a través de peticiones HTTP. También integra directamente las APIs externas de Gemini y fal.ai para el análisis facial y la generación de imágenes, ya que estas llamadas no requieren pasar por el servidor propio.

El **backend** es la capa intermedia. Está construido con Spring Boot y expone una API REST con endpoints para gestionar todas las entidades del sistema: personas, barberos, servicios, reservas, favoritos, opiniones e historial. Se encarga de recibir las peticiones del frontend, aplicar la lógica de negocio y comunicarse con la base de datos. Corre en local durante el desarrollo y en producción se despliega en Render mediante un contenedor Docker.

La **base de datos** es la capa de persistencia. Es una base de datos PostgreSQL alojada en Aiven Cloud, accesible desde cualquier entorno con las credenciales correctas. Spring Boot se conecta a ella a través de JPA e Hibernate, que gestionan el mapeo entre las entidades Java y las tablas de la base de datos.

El flujo de una operación típica sería el siguiente: el usuario realiza una acción en la app, el frontend construye una petición HTTP y la envía al backend, el backend procesa la petición, consulta o modifica la base de datos según corresponda, y devuelve la respuesta al frontend, que actualiza la interfaz con los datos recibidos.

## 4.2 Modelo de datos

La persistencia de la aplicación 2Look se gestiona mediante una base de datos PostgreSQL alojada en la nube (Aiven Cloud). El modelo se compone de siete entidades JPA relacionadas que garantizan la integridad de la información y soportan toda la lógica de negocio orientada al cliente.

A continuación, se detallan las tablas finales según la implementación en el backend:

**persona**: Es la tabla central de usuarios. Almacena nombre, email, password (en texto plano por agilidad en el desarrollo) y el rol. La fecha de registro se genera automáticamente mediante la anotación @PrePersist.

**barbero**: Extiende la información de los profesionales. Mantiene una relación @OneToOne con la tabla persona a través de persona_id. Contiene la especialidad, la foto_url para la interfaz, la valoracion y una biografia.

**servicio**: Define el catálogo disponible con nombre, descripcion, precio (BigDecimal) y la duracion_min estimada para cada corte o tratamiento.

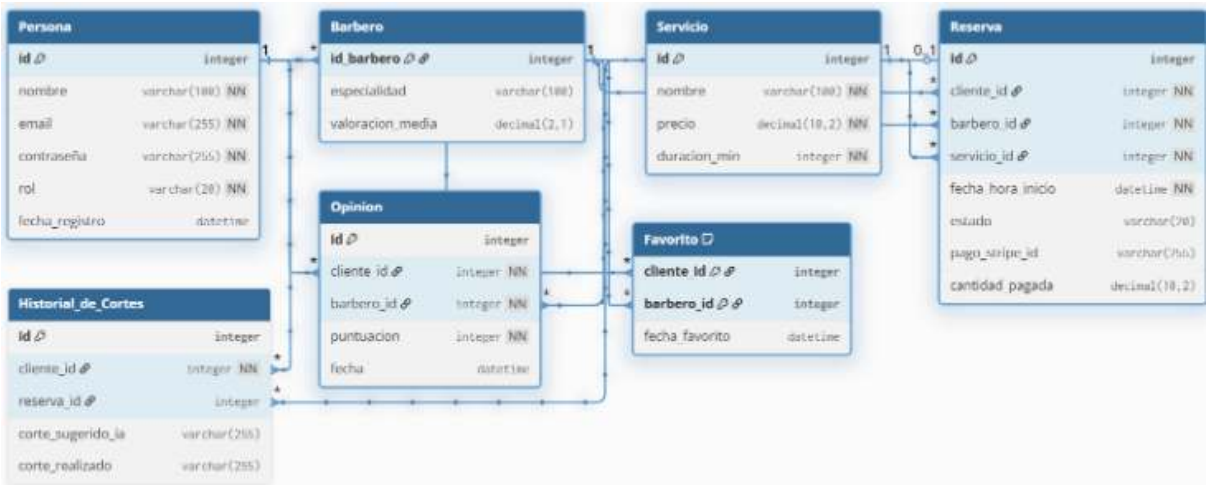
**reserva**: Entidad crítica que vincula al cliente, al barbero y al servicio. Registra la fecha_cita, el estado de la misma y los datos de la pasarela de pagos: pago_stripe_id y cantidad_pagada.

**opinion**: Permite a los usuarios valorar a los profesionales. Cada registro vincula a un cliente con un barbero, almacenando una puntuacion numérica, un comentario y la fecha de la reseña.

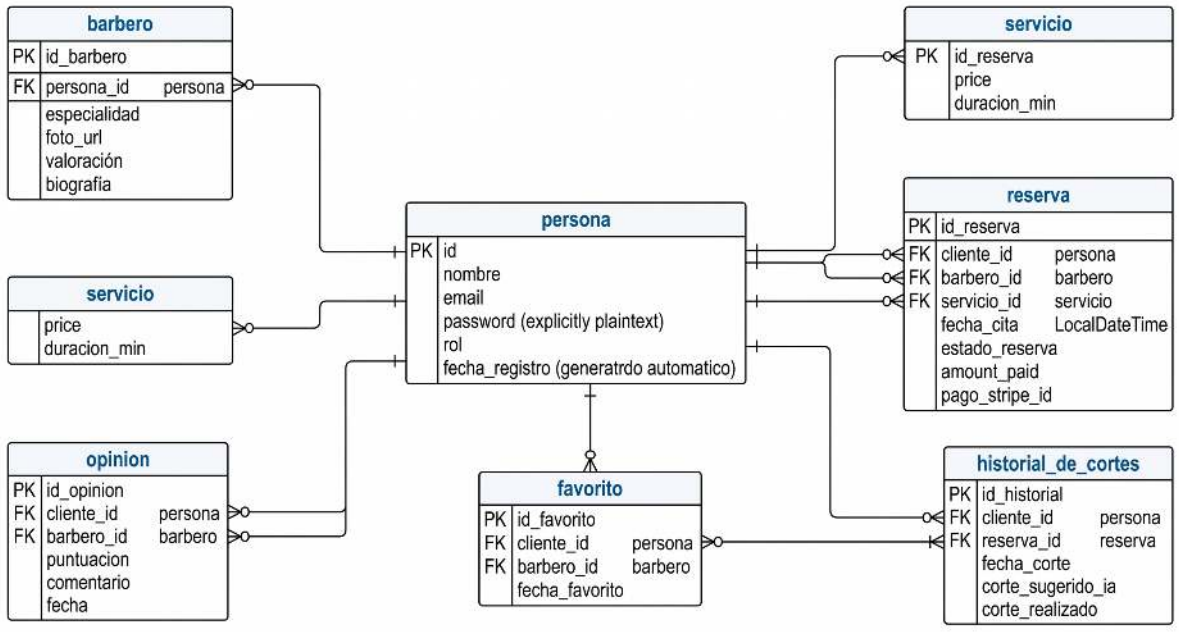
**favorito**: Gestiona la relación de preferencia entre clientes y barberos, permitiendo que el usuario acceda rápidamente a sus profesionales de confianza.

**historial_de_cortes**: Almacena el registro histórico de servicios realizados. Es una pieza clave para la funcionalidad de IA, ya que guarda el corte_sugerido_ia comparándolo con el corte_realizado.

Este diagrama representa la estructura lógica planteada al inicio del proyecto.




Modelo de datos final implementado en el sistema



## 4.3 Diseño de interfaz

Las pantallas de la aplicación se diseñaron previamente en Figma, lo que permitió definir la estructura visual, la paleta de colores y el flujo de navegación antes de comenzar el desarrollo. Partir de un prototipo fiel en Figma facilitó tomar decisiones de diseño sin tener que modificar código, y sirvió como referencia durante toda la fase de implementación del frontend.

El diseño de 2LOOK sigue una estética oscura y moderna, coherente con el sector de las barberías premium. La paleta de colores se basa en azules nocturnos y negros profundos con acentos en azul eléctrico, lo que da a la app una personalidad visual clara y diferenciada.

La navegación principal se organiza en cinco pestañas accesibles desde la barra inferior: Home, Citas, Cámara, Historial y Perfil. Esta estructura permite al usuario acceder a cualquier sección principal en un solo toque sin perderse en menús anidados.

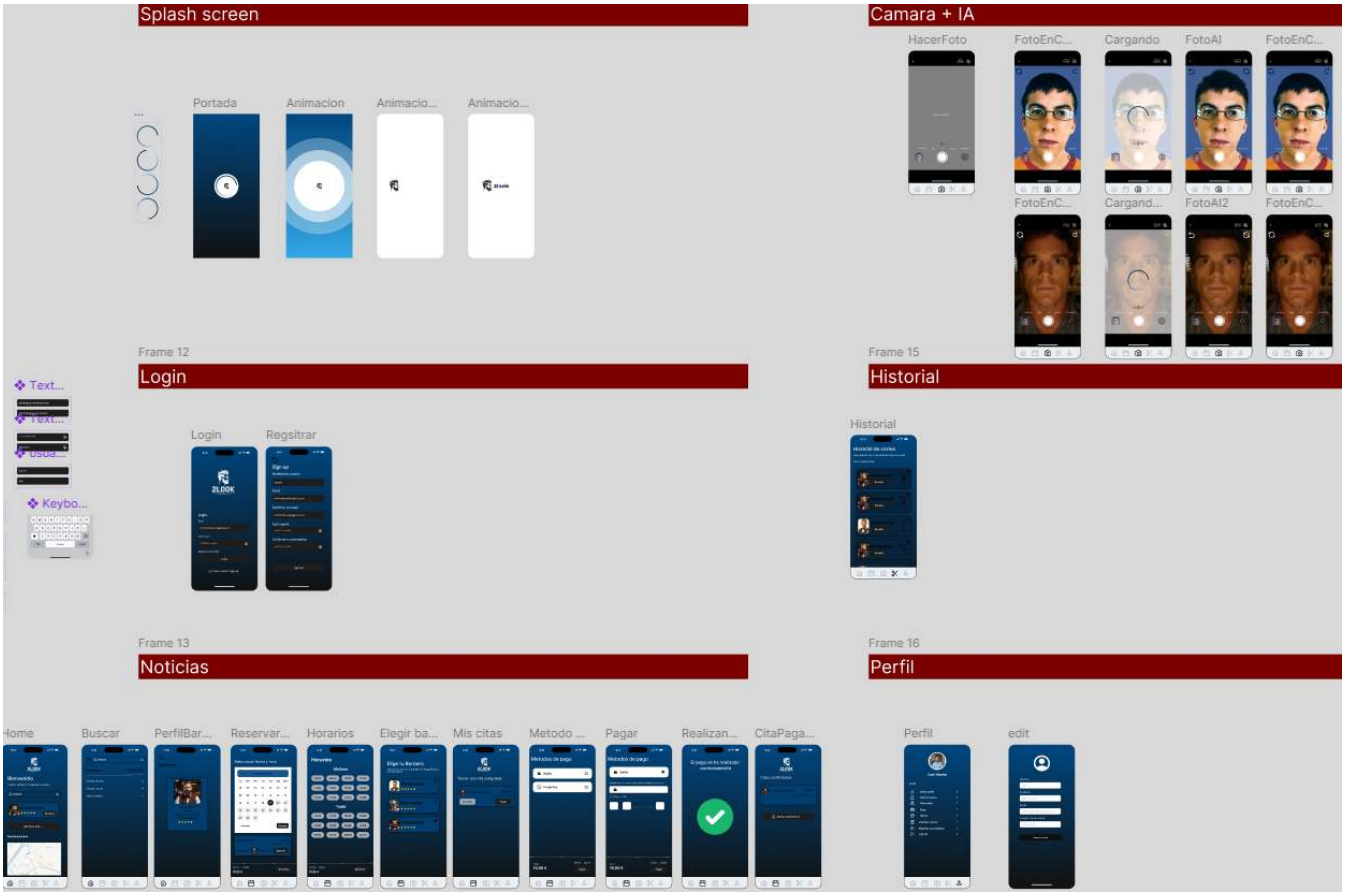
La pantalla de inicio muestra un saludo, una barra de búsqueda y un scroll horizontal con las tarjetas de los barberos disponibles. Más abajo hay un botón de acceso rápido a la reserva y un mapa interactivo con la ubicación de la barbería.

La pantalla de cámara tiene tres fases visuales bien diferenciadas: la vista de cámara con el óvalo guía para centrar la cara, una pantalla de carga con la foto capturada en el fondo desenfocado mientras la IA analiza, y la pantalla de resultados con la forma detectada, la descripción y las tres tarjetas de cortes recomendados con sus imágenes.

El flujo de reserva está dividido en pantallas independientes para no sobrecargar al usuario con demasiadas decisiones a la vez: elegir barbero, elegir corte, seleccionar fecha en el calendario, elegir horario y método de pago. Cada pantalla tiene una barra fija en la parte inferior con el resumen del precio y el botón de continuar.

La tipografía combina Rajdhani para títulos, que da un carácter urbano y moderno, con DM Sans para el texto corriente, que garantiza legibilidad en tamaños pequeños.

En general el diseño prioriza la claridad y la velocidad: el usuario puede completar una reserva en menos de un minuto desde la pantalla principal, y el asesor de IA da su respuesta en pocos segundos sin interrumpir el flujo de la aplicación.



# 5. Desarrollo

## 5.1 Estructura del proyecto

El proyecto está dividido en dos partes bien diferenciadas que conviven en el mismo repositorio de GitHub: el frontend y el backend.

La carpeta frontend contiene toda la aplicación móvil desarrollada con Ionic y Vue 3. Dentro de src se organiza en páginas (pages), que son las 17 pantallas de la app; componentes reutilizables (components); el store de estado global (store), donde viven user.ts con la sesión del usuario y api.ts con todos los servicios de conexión al backend; y el router (router/index.ts) que define toda la navegación. La carpeta public/assets contiene las imágenes estáticas como fotos de barberos y el logo. Capacitor añade la carpeta android con el proyecto nativo que se compila para generar el APK.

La carpeta backend contiene el servidor Spring Boot. Sigue una arquitectura por capas estándar: model con las entidades JPA que mapean las tablas de la base de datos, repository con las interfaces de acceso a datos, service con la lógica de negocio y controller/rest con los controladores REST que exponen los endpoints. También hay una clase SecurityConfig que configura Spring Security y los CORS, y los archivos application.yaml y application-dev.yaml con la configuración del servidor y la conexión a Aiven.

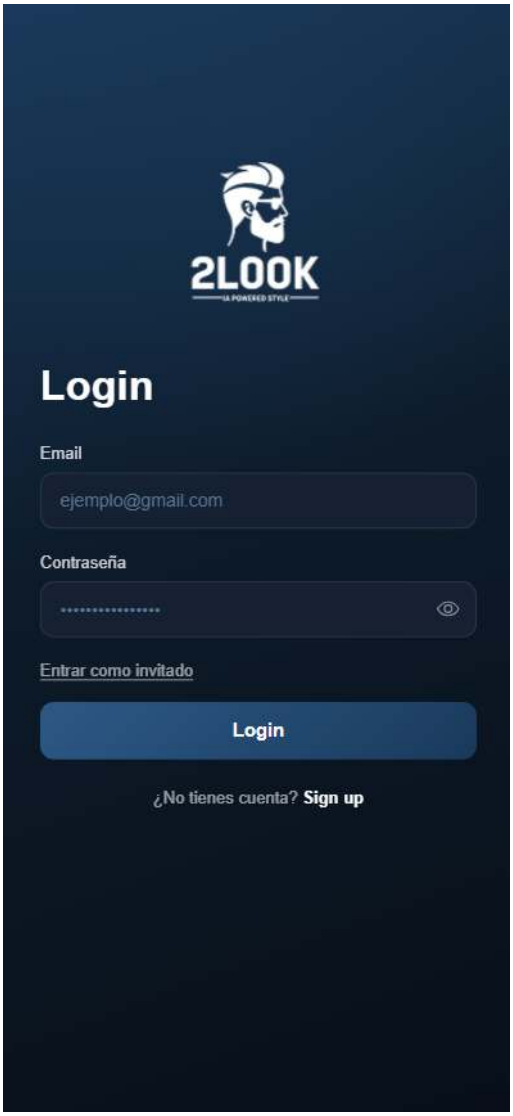
El despliegue se gestiona mediante variables de entorno: en Vercel se configuran VITE_API_URL con la URL del backend, VITE_GEMINI_API_KEY (tres claves independientes de tres cuentas distintas), VITE_FAL_API_KEY y VITE_STRIPE_PUBLISHABLE_KEY. En Render se configura la contraseña de la base de datos Aiven y la clave secreta de Stripe. El backend incluye un Dockerfile basado en Java 21 que permite su despliegue en cualquier plataforma de contenedores. Durante el desarrollo se utilizó DBeaver como herramienta visual para conectarse a la base de datos de Aiven, ejecutar consultas SQL y verificar que los datos se guardaban correctamente en cada fase del proyecto.

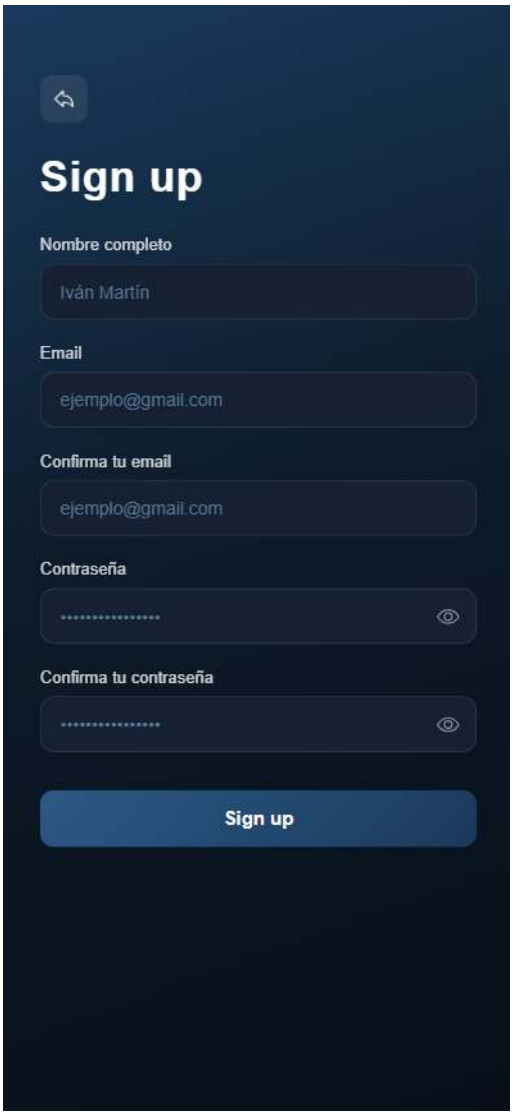
## 5.2 Implementación de funcionalidades

En este apartado se describe la implementación de las principales funcionalidades de la aplicación. La interfaz resultante sigue un diseño oscuro y minimalista, pensado para ofrecer una experiencia de usuario fluida tanto en Android como en iOS

**Autenticación**

El sistema de acceso se implementó mediante endpoints específicos en el backend para la gestión de sesiones y el registro de nuevos usuarios. Una vez que el usuario se identifica correctamente, el servidor devuelve su perfil con el rol de cliente. Para mejorar la experiencia de uso, los datos de sesión se almacenan de forma local en el dispositivo, permitiendo que la aplicación reconozca al usuario sin necesidad de reintroducir sus credenciales en cada apertura.



**Conexión frontend-backend**

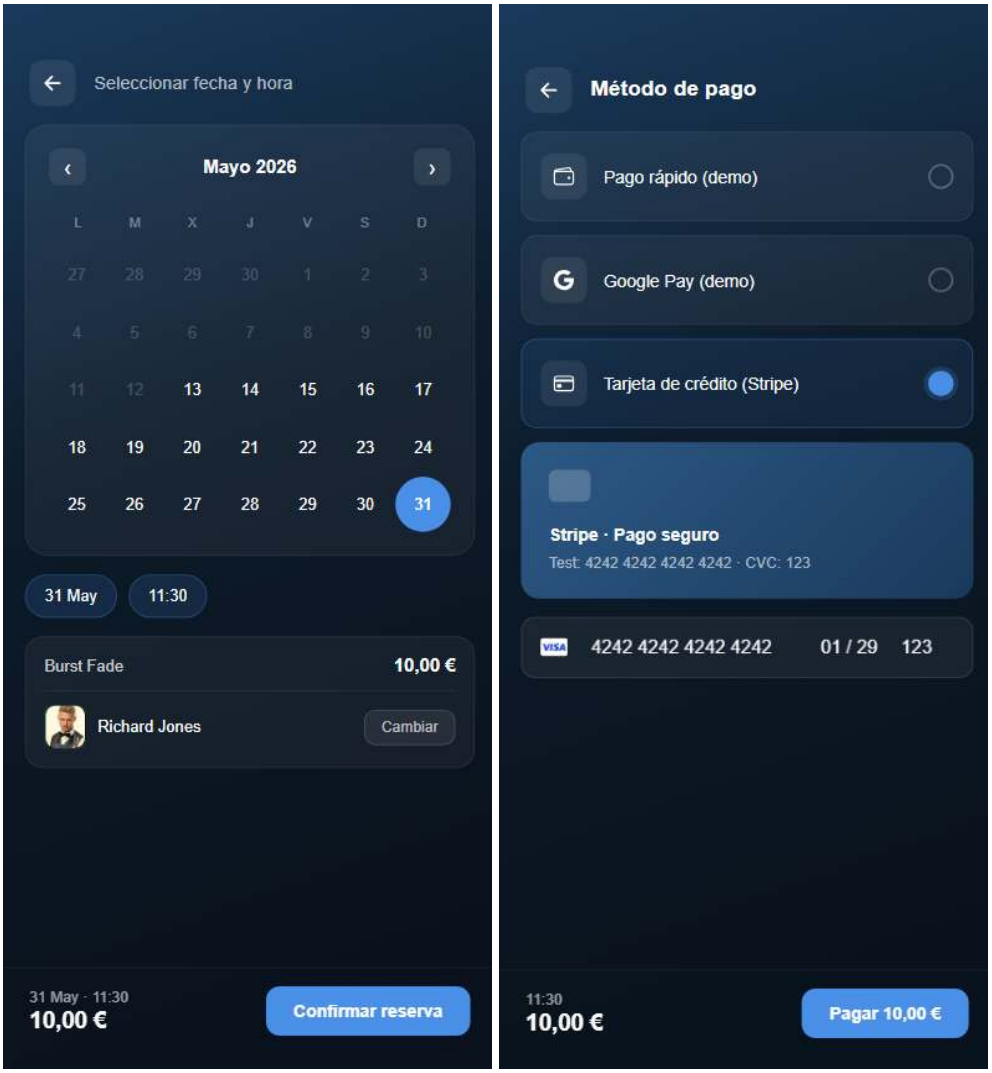
Toda la comunicación entre la aplicación y el servidor se centralizó en un único archivo de servicios api.ts. Cada entidad del sistema dispone de sus propios métodos de acceso: barberos, reservas, servicios, opiniones, entre otros. Todas las pantallas utilizan este servicio en lugar de realizar peticiones directas al servidor, lo que hace el código más limpio y fácil de mantener.

api.ts

```
export const BASE_URL = import.meta.env.VITE_API_URL || 'http://192.168.1.132:8080'

async function request<T>(method: string, path: string, body?: any): Promise<T> {
  const res = await fetch(`${BASE_URL}${path}`, {
    method,
    headers: { 'Content-Type': 'application/json' },
    body: body ? JSON.stringify(body) : undefined
  })
  if (!res.ok) {
    const err = await res.json().catch(() => ({}))
    throw new Error(err?.message || `Error ${res.status}`)
  }
  return res.json()
}

export const personaApi = {
  getAll: () => request<any[]>('GET', '/api/personas'),
  getById: (id: number) => request<any>('GET', `/api/personas/${id}`),
  create: (data: any) => request<any>('POST', '/api/personas', data),
  update: (id: number, data: any) => request<any>('PUT', `/api/personas/${id}`, data),
  delete: (id: number) => request<void>('DELETE', `/api/personas/${id}`)
}

export const barberoApi = {
  getAll: () => request<any[]>('GET', '/api/barberos'),
  getById: (id: number) => request<any>('GET', `/api/barberos/${id}`),
  create: (data: any) => request<any>('POST', '/api/barberos', data),
  update: (id: number, data: any) => request<any>('PUT', `/api/barberos/${id}`, data),
  delete: (id: number) => request<void>('DELETE', `/api/barberos/${id}`)
}
```

**Sistema de reservas**

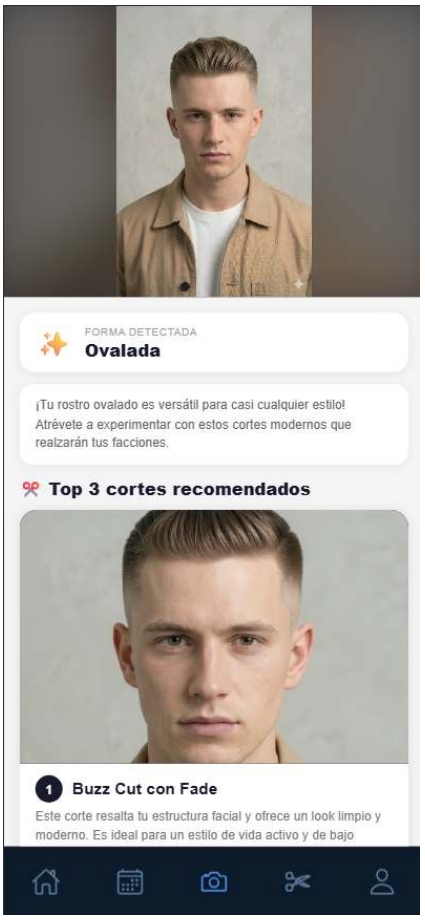
El flujo de reserva se dividió en varias pantallas independientes para no sobrecargar al usuario con demasiadas decisiones a la vez. En primer lugar se selecciona el barbero, después el corte de forma manual o mediante la recomendación de la IA, a continuación la fecha en un calendario interactivo y finalmente el horario. Al confirmar, la cita se guarda en la base de datos. En caso de haber accedido como invitado, la reserva se almacena temporalmente en el dispositivo sin pasar por el servidor.



**Asesor de imagen con IA**

La pantalla de cámara tiene tres fases: captura, análisis y resultado. Al hacer la foto se comprime a un máximo de 800px de ancho para reducir el tamaño y no agotar los tokens de la IA, y luego se envía en base64 a la API de Gemini junto con un prompt estructurado que le pide un JSON con la forma de cara detectada y tres cortes recomendados. El sistema rota automáticamente entre tres claves de API de Gemini de cuentas distintas y prueba varios modelos en orden (gemini-2.5-flash, gemini-2.5-flash-lite, gemini-flash-latest y gemini-2.0-flash entre otros): si una clave o modelo devuelve error 429, 400 o 404 pasa al siguiente sin que el usuario lo note. Una vez recibida la respuesta se parsea el JSON y se muestra al instante la forma de cara y los nombres de los cortes. En paralelo, sin bloquear la interfaz, se mandan tres peticiones a fal.ai usando el modelo FLUX.1 Kontext, que aplica el cambio de peinado sobre la propia foto del usuario manteniendo la cara, expresión, iluminación y fondo originales. Las imágenes van apareciendo en pantalla a medida que se generan.

Si las tres claves agotan su cuota o todos los modelos devuelven error, el sistema activa automáticamente un modo demo que muestra una cara ovalada con tres cortes de ejemplo predefinidos, evitando que el usuario vea un error en blanco.

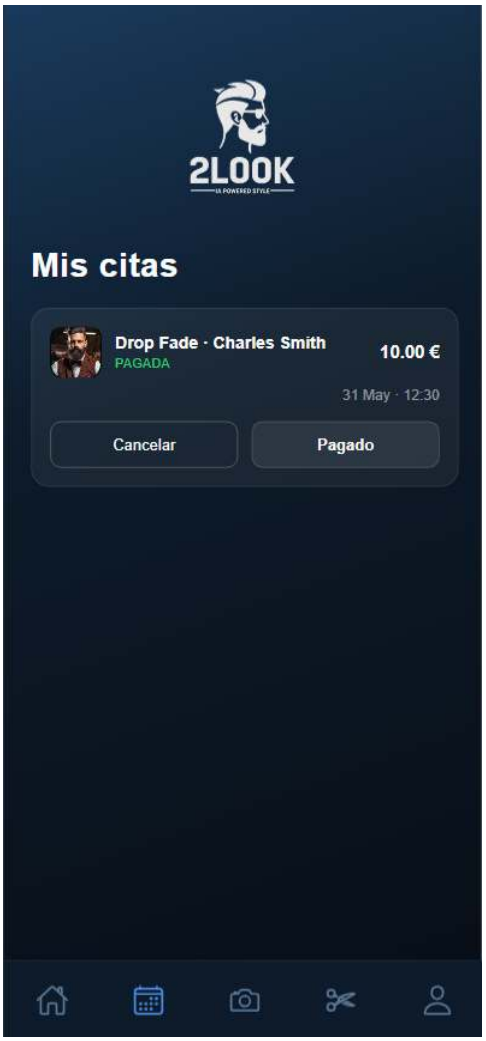


**Pagos con Stripe**

Los pagos se integraron mediante la librería oficial de Stripe para JavaScript. Al seleccionar el método de pago por tarjeta, la aplicación inicializa un formulario seguro de Stripe Elements que gestiona los datos de la tarjeta sin que estos pasen en ningún momento por el propio servidor. Una vez confirmado el pago, Stripe devuelve un identificador de transacción que queda asociado a la reserva en la base de datos. La integración se realizó en modo test durante todo el desarrollo, lo que permitió probar el flujo completo con tarjetas de prueba sin necesidad de datos bancarios reales.

**Gestión de citas**

La pantalla de citas carga las reservas activas del usuario directamente desde la base de datos, mostrando únicamente las que se encuentran en estado pendiente o confirmado. Cada cita muestra el nombre del barbero, el servicio contratado, la fecha y el precio. Es posible cancelar cualquier cita desde esta misma pantalla, lo que actualiza su estado en la base de datos de forma automática. En caso de haberse accedido como invitado sin registrarse, la aplicación muestra igualmente la última cita realizada, aunque esta no queda guardada de forma permanente.



## 5.3 Pruebas

Se realizaron pruebas funcionales sobre los flujos principales de la aplicación en un entorno de producción real, con el backend desplegado en Render y el frontend en Vercel, accesibles desde cualquier dispositivo con conexión a internet.

El login y el registro se probaron con distintos casos: datos correctos, email ya registrado, contraseña incorrecta y acceso como invitado. También se comprobó que la sesión se mantiene al cerrar y volver a abrir el navegador, algo que al principio fallaba y requirió ajustes en el almacenamiento local.

El flujo de reserva se recorrió de principio a fin varias veces: elegir barbero, seleccionar servicio, escoger fecha y hora, confirmar y ver la cita en la pestaña correspondiente. También se verificó que cancelar una cita actualiza correctamente su estado en la base de datos.

La cámara y el asesor de IA se probaron con fotos en distintas condiciones de luz. El mayor problema durante las pruebas fue la gestión de los límites de la API de Gemini: al hacer varios intentos seguidos se agotaba la cuota por minuto, lo que llevó a implementar la rotación entre tres claves distintas y el modo demo como red de seguridad.

Durante el desarrollo se subió por accidente el archivo .env al repositorio público de GitHub, dejando expuestas las claves de Gemini y fal.ai. GitHub detectó la fuga automáticamente, se regeneraron todas las claves comprometidas en Google AI Studio y fal.ai, y se configuró el .gitignore para que el archivo no volviera a subirse nunca más. El incidente sirvió para entender la importancia de proteger los secretos en proyectos con repositorios públicos.

El pago con Stripe se probó en modo test con la tarjeta 4242 4242 4242 4242. Se verificó que al completar el pago la reserva queda marcada como pagada y que el importe aparece correctamente en el panel de Stripe.

Las pruebas en móvil se hicieron tanto en iPhone con Safari como en Android con Chrome.En iOS apareció un problema con el botón de la cámara que quedaba cortado por la barra del navegador, que se solucionó ajustando el CSS con variables de safe area.

# 6. Conclusiones

## 6.1 Conclusiones generales

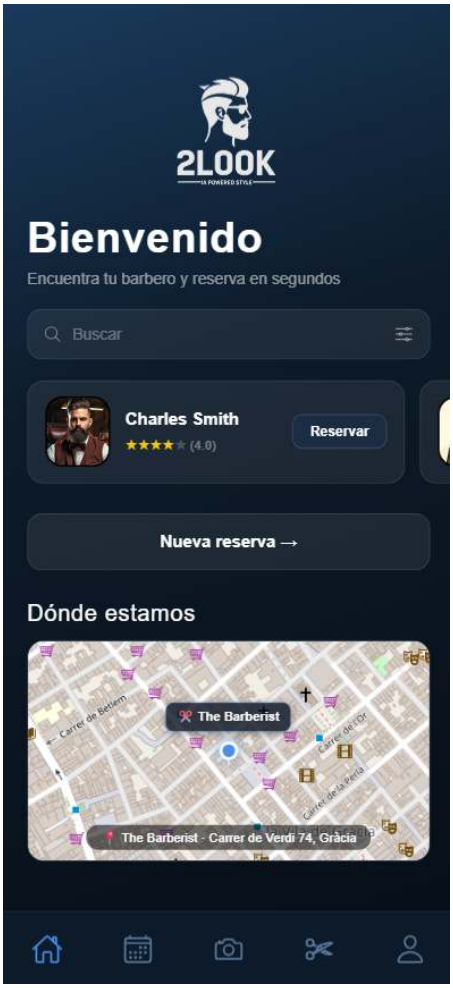
El proyecto ha llegado a donde se pretendía llegar. La aplicación funciona, está desplegada y resuelve el problema que motivó su desarrollo: que alguien pueda saber qué corte le queda bien antes de ir al barbero, sin complicaciones.

El frontend está en https://2look-app.vercel.app (Vercel) y el backend en https://twolookapp.onrender.com (Render), con la base de datos en Aiven Cloud. Para evitar que Render duerma el servidor por inactividad, UptimeRobot lo pingea automáticamente cada 5 minutos. Cualquiera puede entrar desde el móvil, registrarse y usar la app sin instalar nada.

Además de la aplicación principal, se ha desarrollado una landing web producto en HTML, CSS y JavaScript plano, alojada en https://2look-web.vercel.app, que sirve como página de presentación de 2Look orientada al usuario final. Desde ella se puede acceder directamente a la aplicación o descargar la versión APK para Android, disponible en las releases del repositorio de GitHub.

Más allá de que funcione técnicamente, el resultado tiene sentido como producto real: tiene una estética cuidada, una experiencia de usuario fluida y conecta todas las piezas, desde la IA hasta el pago, de forma que el usuario no nota la complejidad que hay detrás.


2LOOK
Cómo funciona
Características
Tecnología
Probar ahora
✦ Powered by AI
Tu próximo corte, decidido por inteligencia artificial
2Look analiza la forma de tu cara y te muestra cómo te quedarían los tres cortes que más te favorecen — sobre tu propia foto, no sobre un modelo.
Probar online
Descargar APK
3
análisis gratis
0€
sin registro
10s
resultado IA
✦ IA
Centra tu cara aquí


2LOOK
Bienvenido
Encuentra tu barbero y reserva en segundos
Buscar
Charles Smith
★★★★☆ (4.0)
Reservar
Nueva reserva →
Dónde estamos
The Barberist
The Barberist · Carrer de Verdi 74, Gràcia


FORMA DETECTADA
Ovalada
¡Tu rostro ovalado es versátil para casi cualquier estilo! Atrévete a experimentar con estos cortes modernos que realzarán tus facciones.
Top 3 cortes recomendados
1
Buzz Cut con Fade
Este corte resalta tu estructura facial y ofrece un look limpio y moderno. Es ideal para un estilo de vida activo y de bajo

## 6.2 Consecución de objetivos

Prácticamente todos los objetivos planteados al inicio se han cumplido. La IA analiza la cara, sugiere cortes y genera imágenes de referencia sobre la propia foto del usuario. El sistema de reservas funciona de principio a fin. El backend gestiona todas las entidades correctamente. Los pagos van con Stripe.

Donde más trabajo hubo que ajustar fue en la gestión de la cuota de Gemini. El plan inicial era un sistema sencillo con una sola clave, pero al final se implementó tanto la rotación entre múltiples modelos (gemini-2.5-flash, gemini-2.5-flash-lite, gemini-2.0-flash entre otros) como entre tres claves de API de cuentas distintas, lo que resultó más robusto y resistente a los límites del tier gratuito.

Lo único que no quedó completamente resuelto son las notificaciones push. Funcionan en Safari de iOS pero no en Chrome, porque Apple obliga a todos los navegadores de iPhone a usar su motor y tiene restricciones propias. Es una limitación de la plataforma, no del código.

## 6.3 Valoración de la metodología y planificación

Kanban con Trello funcionó bien para este tipo de proyecto. Tener las tareas divididas en pequeñas piezas concretas ayudó a mantener el ritmo y a no perderse cuando algo se complicaba más de lo previsto.

Lo que más se subestimó al principio fue todo lo relacionado con la IA. No el análisis en sí, sino la gestión de los límites de la API gratuita de Gemini. Cada intento consume cuota, y cuando se desarrolla y se prueba seguido, el límite diario se agota rápido. Eso obligó a buscar soluciones que no estaban en el plan inicial.

Mirando atrás, habría sido útil dedicar más tiempo a probar en dispositivos reales antes del despliegue. Varios problemas que aparecieron en iPhone, como el botón de la cámara

cortado o el comportamiento del scroll, no se detectaron hasta probar en el móvil con la versión desplegada.

## 6.4 Visión de futuro

Tras el desarrollo , se han identificado varios puntos donde la aplicación podría evolucionar para ser un producto comercial completo:

**Seguridad y Cifrado**: La mejora más urgente es la implementación de BCrypt para cifrar las contraseñas en la base de datos. Actualmente, por agilidad en las pruebas, se almacenan en texto plano, pero para una fase de producción real es innegociable el uso de hashes de seguridad.

**Panel para Barberos**: La aplicación actual está 100% enfocada al cliente. Una evolución lógica sería crear una interfaz o una segunda app para que los barberos puedan gestionar su propia agenda, ver sus ingresos y actualizar sus fotos o biografía sin intervención del administrador.

**Optimización de la IA**: Actualmente usamos una rotación de tres API Keys de Gemini para evitar límites de cuota. Una mejora sería implementar un sistema de caché de recomendaciones o pasar a un modelo de pago por uso para garantizar que la app nunca se quede "colgada" si tiene muchos usuarios simultáneos.

**Sistema de Notificaciones**: Implementar notificaciones push (mediante Firebase) para recordar al usuario su cita una hora antes o avisarle si el barbero ha tenido que cancelar por una emergencia.

**Fidelización**: Añadir un sistema de "puntos" o "monedero" donde, tras cada reserva pagada con Stripe, el usuario acumule un pequeño porcentaje para descontar en su siguiente corte.

# 7. Glosario

**API REST**: Interfaz de programación de aplicaciones basada en el protocolo HTTP que permite la comunicación entre sistemas mediante peticiones y respuestas en formato JSON.

**API Key**: Clave de autenticación que identifica a una aplicación al hacer peticiones a una API externa, permitiendo controlar el acceso y los límites de uso.

**Ionic**: Framework de código abierto para el desarrollo de aplicaciones móviles híbridas usando tecnologías web estándar como HTML, CSS y JavaScript.

**Vue 3**: Framework progresivo de JavaScript para la construcción de interfaces de usuario reactivas.

**Spring Boot**: Framework de Java que facilita la creación de aplicaciones backend con configuración mínima y arquitectura por capas.

**PostgreSQL**: Sistema de gestión de bases de datos relacionales de código abierto, robusto y con soporte para tipos de datos avanzados.

**JPA/Hibernate**: Tecnologías que gestionan el mapeo entre objetos Java y tablas de base de datos relacionales.

**Gemini**: API de inteligencia artificial de Google con capacidades de visión artificial para analizar imágenes y generar respuestas en lenguaje natural.

**fal.ai**: Plataforma de inteligencia artificial para la generación de imágenes mediante modelos de difusión.

**Stripe**: Plataforma de procesamiento de pagos online que permite integrar métodos de pago en aplicaciones web y móviles.

**Aiven**: Servicio en la nube para alojar bases de datos gestionadas con alta disponibilidad.

**Vercel**: Plataforma de despliegue optimizada para aplicaciones frontend con integración continua desde GitHub.

**Render**: Plataforma de despliegue para aplicaciones backend con soporte para contenedores Docker.

**Kanban**: Metodología ágil de gestión del trabajo basada en un tablero visual con columnas que representan el estado de cada tarea.

**Trello**: Herramienta online de gestión de proyectos basada en tableros Kanban.

**CORS**: Mecanismo de seguridad del navegador que controla las peticiones entre dominios distintos.

**Base64**: Codificación que permite enviar imágenes como cadenas de texto en peticiones HTTP.

**Fallback**: Mecanismo de respaldo que se activa automáticamente cuando el sistema principal no está disponible.

**Docker**: Plataforma que permite empaquetar y ejecutar aplicaciones en entornos aislados mediante contenedores.

**UptimeRobot**: Servicio de monitorización que realiza peticiones periódicas a un servidor para mantenerlo activo.

# 8. Bibliografía


Ionic. (s.f.). Navegación con pestañas en Vue.
https://ionicframework.com/docs/vue/navigation

Ionic. (s.f.). Primera app con cámara y galería.
https://ionicframework.com/docs/vue/your-first-ap

Capacitor. (s.f.). API de cámara — getPhoto, permisos.
https://capacitorjs.com/docs/apis/camera

Capacitor. (s.f.). Variables CSS safe-area. https://capacitorjs.com/docs/apis/system-bars

Pinia. (s.f.). Definición de stores — state, actions. https://pinia.vuejs.org/core-concepts/

Vue Router. (s.f.). Navegación programática.
https://router.vuejs.org/guide/essentials/navigation.html

Spring Boot. (s.f.). Propiedades de configuración.
https://docs.spring.io/spring-boot/appendix/application-properties/index.html

Spring Data JPA. (s.f.). Documentación oficial.
https://docs.spring.io/spring-data/jpa/reference/index.html

Spring Security. (s.f.). Configuración CORS.
https://docs.spring.io/spring-security/reference/servlet/integrations/cors.html

Spring MVC. (s.f.). CORS con WebMvcConfigurer.
https://docs.spring.io/spring-framework/reference/web/webmvc-cors.html

Google AI. (s.f.). Análisis de imágenes con Gemini.
https://ai.google.dev/gemini-api/docs/image-understanding

fal.ai. (s.f.). FLUX.1 Kontext dev — edición de imagen.
https://fal.ai/models/fal-ai/flux-kontext/dev/api

fal.ai. (s.f.). Quickstart JavaScript. https://docs.fal.ai/model-apis/quickstart

Stripe. (s.f.). Payment Element — formulario de tarjeta. https://docs.stripe.com/payments/payment-element

Stripe. (s.f.). Payment Intents API. https://docs.stripe.com/payments/payment-intents

Stripe. (s.f.). Tarjetas de prueba. https://docs.stripe.com/testing

Aiven. (s.f.). Primeros pasos con PostgreSQL. https://aiven.io/docs/products/postgresql/get-started

Vercel. (s.f.). Configuración del proyecto. https://vercel.com/docs/project-configuration

Render. (s.f.). Despliegue con Docker. https://render.com/docs/docker

Docker. (s.f.). Buenas prácticas para Spring Boot. https://www.docker.com/blog/9-tips-for-containerizing-your-spring-boot-code/

UptimeRobot. (s.f.). API y monitores HTTP. https://uptimerobot.com/api/

# 9. Anexos

El código fuente completo del proyecto está disponible en el repositorio de GitHub: https://github.com/IvanMartinBodas/2lookApp

La aplicación desplegada en producción está accesible en: https://2look-app.vercel.app

Las pruebas de pago se realizaron con la tarjeta de prueba de Stripe: número 4242 4242 4242 4242, fecha de expiración 12/26 y CVC 123.

### Nota sobre el uso de inteligencia artificial

Durante el desarrollo de este proyecto se ha utilizado inteligencia artificial Claude como herramienta de apoyo en dos ámbitos distintos.

En lo que respecta a la programación, la IA se utilizó para resolver problemas concretos de maquetación y CSS, así como para depurar errores puntuales en partes del código que resultaban especialmente complejas o estaba encallado. En ningún caso se delegó el diseño de la arquitectura ni las decisiones técnicas del proyecto: la estructura del sistema, la elección de tecnologías y la lógica de funcionamiento fueron elaboradas íntegramente por el autor.

En cuanto a la redacción de esta memoria, el proceso seguido fue el siguiente: el autor redactó cada apartado con sus propias palabras y criterio, y posteriormente utilizó la IA para mejorar la expresión y coherencia del texto. El resultado fue revisado y validado en su totalidad por el autor antes de incorporarlo al documento, asumiendo el contenido como propio en todo momento.

El uso de estas herramientas ha sido consciente, supervisado y limitado a tareas de soporte, sin sustituir en ningún caso el aprendizaje ni el trabajo del autor.

## Licencia