PistaGO

Sistema de gestión de reservas de pistas de tenis

Pedro Manuel Avilés Aguilera · 2º DAM · IES Portada Alta · Junio 2026

PROYECTO FINAL DE CICLO SUPERIOR

DESARROLLO DE APLICACIONES MULTIPLATAFORMA

A · EL CONTEXTO

Reservas gestionadas a mano. Caos.

Los clubes deportivos tradicionales gestionan sus pistas con métodos manuales: llamadas telefónicas, hojas de papel y mensajes de WhatsApp. El resultado es predecible: errores, conflictos y oportunidades perdidas. PistaGO nace para resolver este problema real con tecnología moderna.

🔀 Solapamientos

Dos socios reservando la misma franja horaria sin saberlo, generando conflictos y reclamaciones en el club.

👁️ Falta de visibilidad

No existe una forma fácil y centralizada de consultar qué pistas están libres en tiempo real.

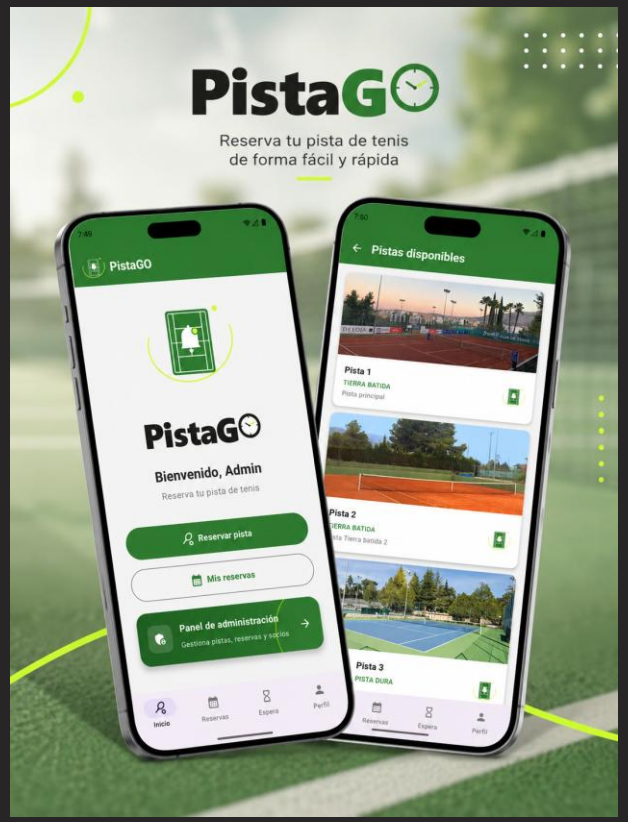
🗑️ Cancelaciones perdidas

Las pistas que quedan libres por cancelaciones no se reaprovechan, perdiendo capacidad de uso del club.

A · LA SOLUCIÓN

PistaGO: app + backend + admin

Un sistema completo, en producción real, accesible desde cualquier dispositivo Android. Diseñado con una arquitectura cliente-servidor limpia que separa responsabilidades y permite escalar cada componente de forma independiente. El resultado es una solución robusta, segura y con identidad visual propia.



App Android Nativa

- Kotlin + Jetpack Compose
- Material Design 3
- Identidad visual propia

Sistema de notificaciones

- Integrado con Firebase Cloud Messaging
- Información en tiempo real

Backend REST

- Spring Boot 3 + Kotlin
- JWT + Spring Security
- PostgreSQL 16

Infraestructura

- VPS Ubuntu (DigitalOcean)
- HTTPS con Let's Encrypt
- API en api.nacimiento.me

A · CASOS DE USO UML

Casos de uso · PistaGO

D · STACK TECNOLÓGICO

Tecnologías empleadas

El proyecto integra un stack tecnológico moderno y coherente, seleccionado para garantizar rendimiento, mantenibilidad y una experiencia de usuario fluida. Cada tecnología cumple un rol específico dentro de la arquitectura, desde la interfaz nativa en el cliente hasta la persistencia relacional en el servidor.

Cliente Android

- Kotlin · Jetpack Compose
- Material Design 3
- Arquitectura MVVM + Clean
- Hilt · Retrofit · Coil
- DataStore

Backend & Datos

- Spring Boot 3 (Kotlin)
- Spring Security · JWT
- JPA / Hibernate
- Bean Validation
- PostgreSQL 16 · 7 tablas · 3FN

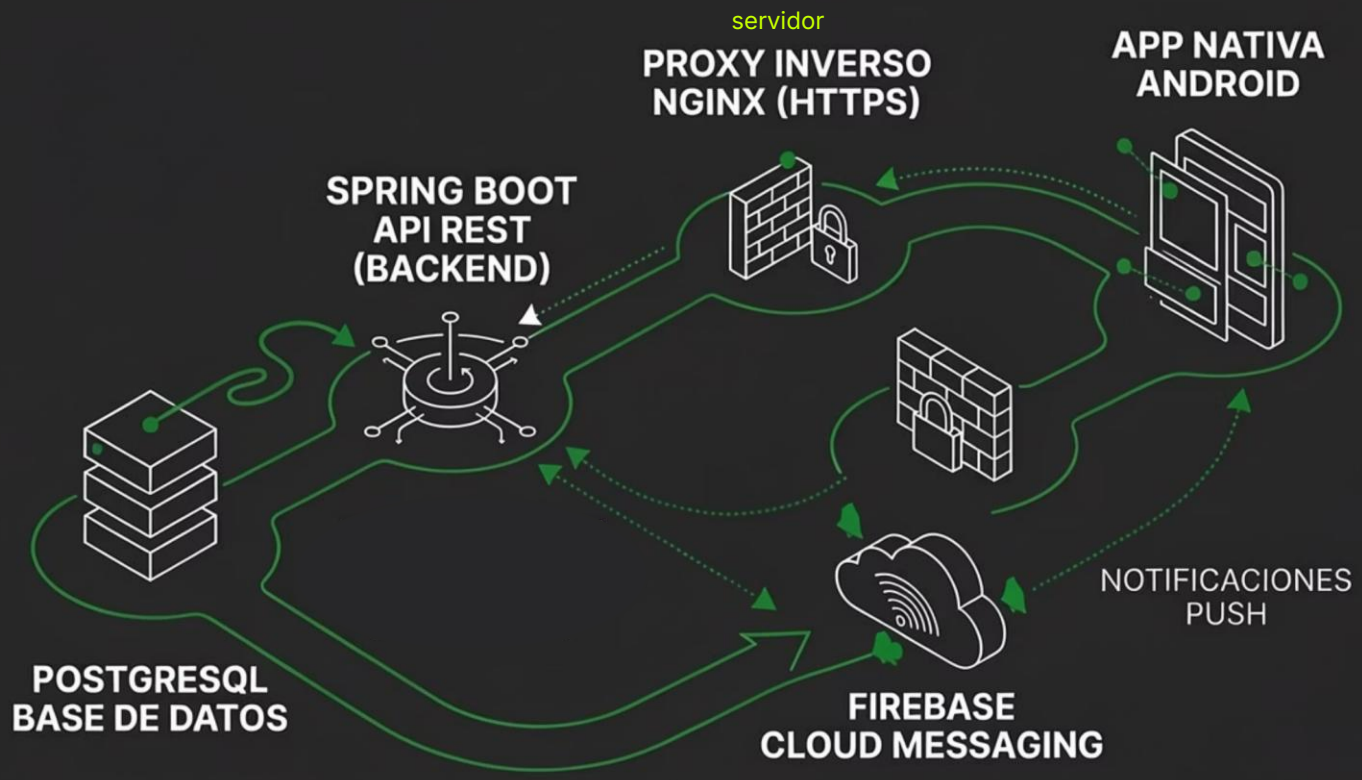
Infraestructura

- VPS Ubuntu (DigitalOcean)
- Nginx · HTTPS Let's Encrypt
- Firebase Cloud Messaging

D · ARQUITECTURA

Cliente-servidor con notificaciones push

La arquitectura sigue un patrón cliente-servidor REST puro, con una capa de notificaciones push que permite al backend comunicarse proactivamente con los usuarios. Nginx actúa como proxy inverso y termina la conexión HTTPS, mientras que Firebase Cloud Messaging gestiona la entrega de avisos en tiempo real. El resultado es un sistema desacoplado, seguro y escalable.



El flujo de notificaciones es clave: cuando un usuario cancela una reserva, el backend identifica automáticamente al primero de la lista de espera y le envía un push a través de Firebase, sin intervención manual.

D · ARQUITECTURA

Arquitectura Interna – PistaGO

Arquitectura basada en MVVM + Clean Architecture. Separación por capas para lograr una app mantenible, escalable y testable.

APP ANDROID

PRESENTATION (UI Layer)

- Jetpack Compose
- ViewModel

Patrón MVVM
La UI observa el ViewModel
Acciones del usuario -> ViewModel.
El ViewModel obtiene datos del dominio

DOMAIN (Bussines Layer)

- Modelos de negocio
- Interfaces de Los respoitorios

Reglas de negocio
Contiene la lógica y entidades de la aplicación. No depende de ninguna tecnología

DATA (Data Layer)

- Implementa repositorios
- Fuentes de DATOS

Acceso a datos
Obtiene y envía datos
A la API.
Traduce DTOs a modelos
De dominio.

API REST (Retrofit)

BACKEND – SPRING BOOT

CONTROLLERS	SERVICES	REPOSITORIES	BASE DE DATOS
Expone endpoints REST	Contiene la Lógica de negocio	Acceso a datos (Spring Data JPA)	Almacenamiento persistente

LISTA ESPERA SCHEDULER

Rotación automática de la cola cada 30 segundos
(@Scheduled(fixedRate = 30000))
Asegura el avance automático de la lista de espera cuando hay disponibilidad.

INYECCIÓN DE DEPENDENCIAS Y COMPONENTES COMPARTIDOS

HILT (DI)

Gestión de dependencias con Hilt.

Módulos:

- AppModule: prove Retrofit, OkHttp, Gson, TokenDataStore(JWT)
- RepositoryModule: enlaza interfaces con Implementaciones (@Binds)

OTROS COMPONENTES

- PistaGoApp (@HiltAndroidApp) : Inicializa el grafo de dependencias
- MainActivity: punto de entrada de la Interfaz.
- TokenDataStore: almacenamiento Seguro del token JWT

NOTIFICACIONES PUSH

PistaGoFirebaseService
(FirebaseMessagingService)

Recepción de notificaciones push mediante Firebase Cloud Messaging (FCM)

BENEFICIOS

- Separación de responsabilidades
- Código mantenible y escalabe
- Fácil de probar
- Cambios aislados por capa
- Nuevas funcionalidades sin reescribir.

D · BASE DE DATOS

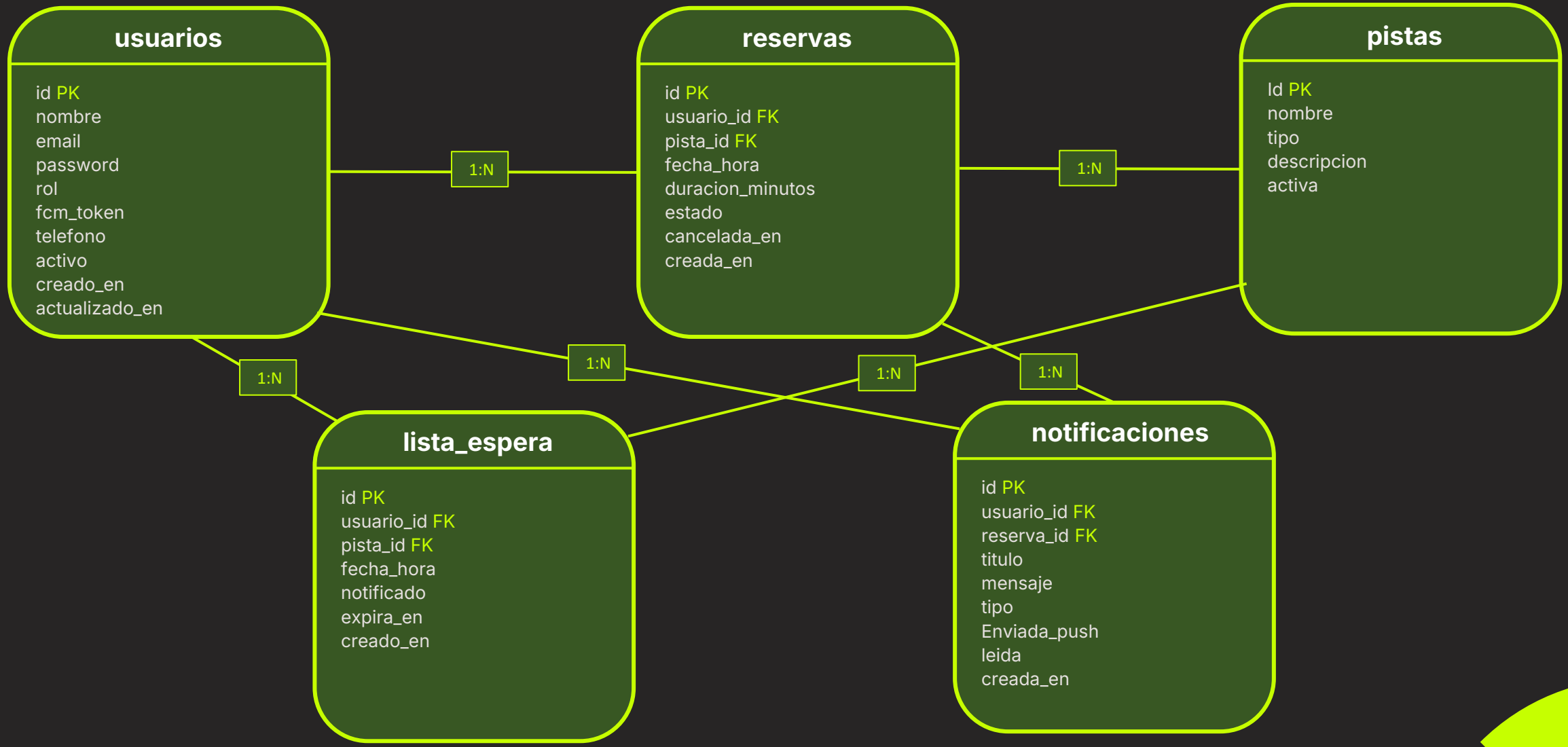
7 tablas modeladas, 5 activas en producción

El modelo de datos está formado por siete tablas realacionadas entre sí, de las cuales cinco se encuentra en producción. Se han incluido índices compuestos optimizados para las consultas de disponibilidad, que son las operaciones más frecuentes del sistema. A continuación se detalla cada tabla y su propósito dentro del dominio del club.

usuarios	Socios y administradores. Token FCM.
pistas	Catálogo de pistas.
reservas	Reservas por usuario y sus 3 estados
lista_espera	Cola FIFO por pista y franja horaria.
notificaciones	Historial de avisos push enviados.
Bloqueos_horario	Bloque de horario para mantenimientos y/o terneos.
configuracion_sistema	Parámetros globales del club: horarios, límites de reserva, políticas de cancelación.

D · DIAGRAMA E-R

Diagrama Entidad-Relación · PistaG



IMP · DECISIONES TÉCNICAS

Lista de espera con notificaciones push

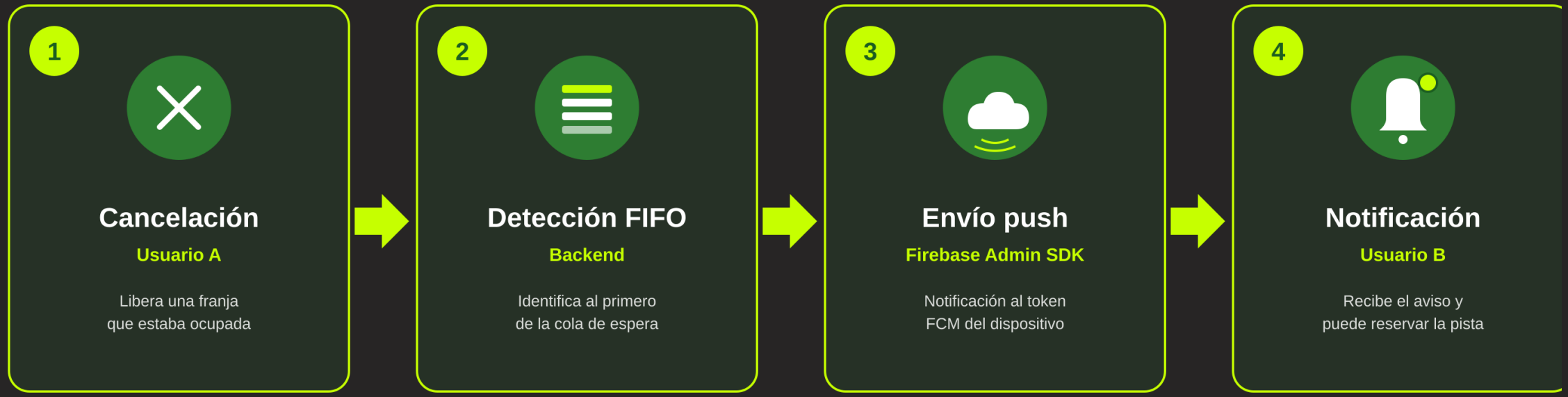
Uno de los retos más interesantes del proyecto fue automatizar la reasignación de pistas cuando un usuario cancela, sin que el siguiente en la lista tenga que estar pendiente manualmente. La solución combina una cola FIFO en backend con notificaciones push automáticas vía Firebase.

El Reto

Si una franja está ocupada, ¿cómo reasigno automáticamente cuando el usuario cancela, sin que el siguiente tenga que estar pendiente?

La Solución

1. El usuario se apunta a una cola FIFO en backend
2. Posición calculada dinámicamente por orden de llegada
3. Cuando alguien cancela, el backend identifica al primero
4. Envía push vía Firebase Admin SDK al token FCM
5. Se registra en historial para no notificar dos veces



Registrado en historial · No se notifica dos veces al mismo usuario

Este flujo elimina la necesidad de supervisión manual y garantiza que las pistas vacías se ocupen de forma inmediata y ordenada.

IMP · REGLAS Y ESTADÍSTICAS

Reglas de negocio y estadísticas a medida

Además de la lista de espera, el proyecto incorpora otras decisiones técnicas que reflejan un enfoque pragmático: controlar el acceso equitativo a las pistas y optimizar el peso de la aplicación sin sacrificar funcionalidad.

Una reserva activa por usuario

Para que las pistas se repartan entre todos los socios, un usuario solo puede tener **una reserva confirmada activa** a la vez. Los administradores quedan exentos de esta restricción.

Caso 1 · Una reserva activa

Usuario con reserva activa intenta crear otra → **error 409**.

Caso 2 · Permitir cuando ya expiró

Su reserva anterior ya ha pasado → se permite crear la nueva.

Caso 3 · Excepción user admin

Si el rol es **ADMINISTRADOR** → se permite siempre.

IMP · PRUEBAS

30 pruebas unitarias, 100% de éxito

Las pruebas unitarias han sido escritas desde el primer día del proyecto, siguiendo una disciplina de desarrollo guiado por tests. El resultado es un conjunto de 30 pruebas que cubren la lógica de negocio crítica del sistema, con una tasa de éxito del 100% y un tiempo total de ejecución de apenas 2 segundos.

30

Pruebas totales

Cubriendo toda la lógica de negocio crítica

29

Lógica de negocio

Tests específicos de servicios y reglas

100%

Tasa de éxito

Todas las pruebas pasan correctamente

1,998

Segundos en total

Tiempo de ejecución completo del suite

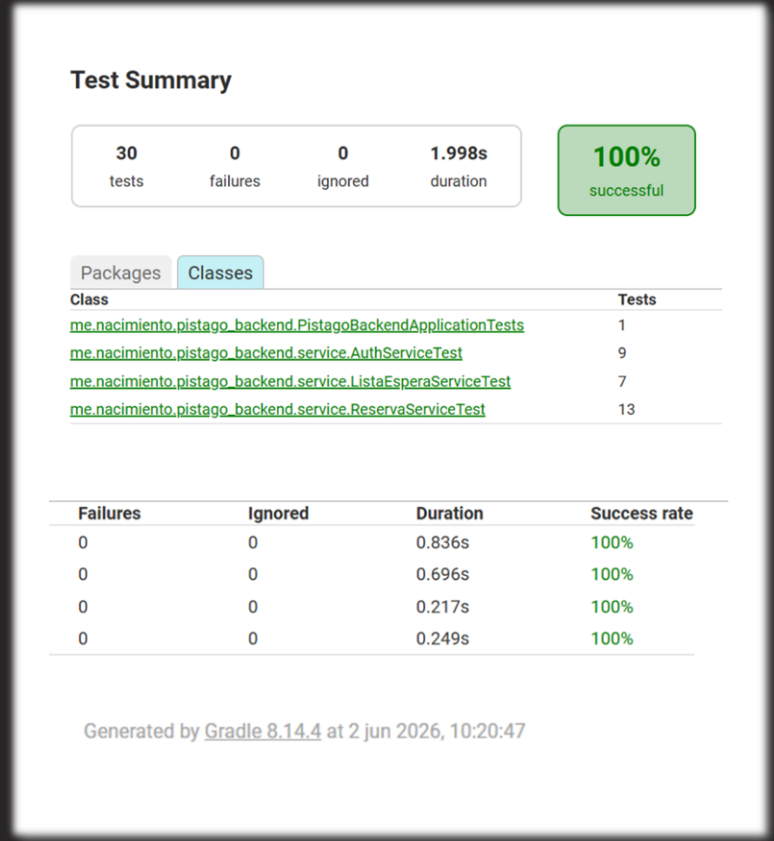
Desglose por servicio

- **ReservaServiceTest** · 13 pruebas
- **AuthServiceTest** · 9 pruebas
- **ListaEsperaServiceTest** · 7 pruebas
- **Spring contextLoads()** · 1 prueba

Cobertura funcional

Incluyen:

- Autentificación
- Reservas
- Cancelaciones
- Gestión de listas de espera



Test Summary

30 tests	0 failures	0 ignored	1.998s duration	100% successful

Packages | Classes

Class	Tests
me.nacimiento.pistago_backend.PistagoBackendApplicationTests	1
me.nacimiento.pistago_backend.service.AuthServiceTest	9
me.nacimiento.pistago_backend.service.ListaEsperaServiceTest	7
me.nacimiento.pistago_backend.service.ReservaServiceTest	13

Failures	Ignored	Duration	Success rate
0	0	0.836s	100%
0	0	0.696s	100%
0	0	0.217s	100%
0	0	0.249s	100%

Generated by Gradle 8.14.4 at 2 jun 2026, 10:20:47

start .\build\reports\tests\test\index.html

PRO · DESPLIEGUE

Pipeline de despliegue · PistaGO

Backend → Producción

1. **Desarrollo**
   IntelliJ IDEA
   Kotlin + Gradle
2. **Build**
   `./gradlew bootJar`
   JAR ejecutable
3. **Despliegue**
   `scp → VPS`
   `systemctl restart`
4. **Endpoint público**
   `https://api.nacimiento.me`
   HTTPS · Let's Encrypt

VPS DigitalOcean · Ubuntu 24.04

- **systemd** — arranque + reinicio
- **Spring Boot** — JAR en puerto 9090
- **PostgreSQL 16** — nativo, no Docker
- **Nginx** — proxy inverso + SSL
- **Let's Encrypt** — cert HTTPS auto

Cliente Android → Distribución

1. **Desarrollo**
   Android Studio
   Kotlin + Compose
2. **Build**
   APK firmado
   keystore propio
3. **Publicación**
   `nacimiento.me/pistago`
   descarga directa + QR
4. **Instalación**
   APK en dispositivo
   socio del club

Repositorio único en github.com/pedroaviless/PistaGO · monorepo con backend y app Android

Documentación interactiva de la API en api.nacimiento.me/swagger-ui.html

PRO · DESPLIEGUE

En producción, accesible públicamente

PistaGO no es un prototipo local: está desplegado en un VPS real con dominio propio, HTTPS válido y APK firmada disponible para descarga. El proyecto integra además un portfolio público con documentación técnica y el código fuente completo en GitHub.

VPS Ubuntu · DigitalOcean

GitHub Student Pack · Nginx como proxy inverso · HTTPS con Let's Encrypt renovable automáticamente.

Código abierto

Repositorio público en **github.com/pedroaviless** con historial de commits siguiendo Conventional Commits.

Portfolio del proyecto

Documentación completa en **nacimiento.me/proyectos/pistago** con capturas, decisiones técnicas y roadmap.

APK firmada descargable

Disponible en **nacimiento.me/pistago** con código QR para instalación directa en Android.

Usuario de prueba:

usuarioprueba@pistago.com · pistago2026

Lo aprendido

- Arquitectura de un sistema completo en producción
- DevOps básico: VPS, HTTPS, dominios propios
- Disciplina técnica: tests, commits, documentación
- Integración de todos los módulos del ciclo formativo

Trabajo futuro

- Caché local offline con Room, guardar datos estructurados en SQL
- Recordatorios push previos al Partido
- Bloqueos por franja horaria y configuración dinámica desde BD
- Automatización de la transición a reserva EXPIRADA con scheduler periódico,
- Integración y despliegue continuos (CI/CD) con GitHub Actions: ejecutar la suite de test en cada push y publicar el JAR al VPS automáticamente al fusionar a main.

Probar la app: **nacimiento.me/pistago** · Portfolio: **nacimiento.me** · Código: **github.com/pedroaviless**

DIF · PROBLEMAS Y DECISIONES

Retos técnicos y decisiones clave

Seguridad y autenticación

Problema:

Proteger datos y operaciones sensibles

Decisión:

JWT + Spring Security + control por roles

Resultado:

Acceso seguro para usuarios y administradores

Evitar reservas conflictivas

Problema:

Dos usuarios podían intentar reservar la misma pista

Decisión:

Validaciones de disponibilidad en el backend

Resultado:

Integridad garantizada de las reservas

Rotación automática de la cola

Problema:

Si el usuario notificado no responde, la cola se bloquearía

Decisión:

Scheduler cada 30 s con expiración de 3 min.

Resultado:

La cola siempre avanza

Notificaciones en tiempo real

Problema:

Firebase puede fallar

Decisión:

FcmService captura excepciones y devuelve Boolean

Resultado:

Cancelaciones siempre completadas

DEMO EN VIVO

Vamos a verlo funcionar

GRACIAS

¿Preguntas?

Probar la app
nacimiento.me/pistago

Portfolio
nacimiento.me

Código
github.com/pedroaviless