

Pawsi Desarrollo y seguridad de una aplicación en producción

Diego Puzio · Caso de estudio · Octubre de 2026

Desarrollé Pawsi para publicar mascotas en adopción, perdidas y avistadas, con una versión web y una aplicación Android construida con Capacitor. Al revisar su seguridad, preparé un entorno local de recuperación y corregí dos problemas del sistema de mensajes: la confianza en las solicitudes de notificación y los permisos excesivos de actualización.

Mi trabajo abarcó desarrollo, despliegue, revisión del backend, pruebas y documentación. Este caso presenta las correcciones y la evidencia de su funcionamiento. El alcance es acotado: no constituye una auditoría exhaustiva ni una certificación de seguridad.

Del prototipo al producto

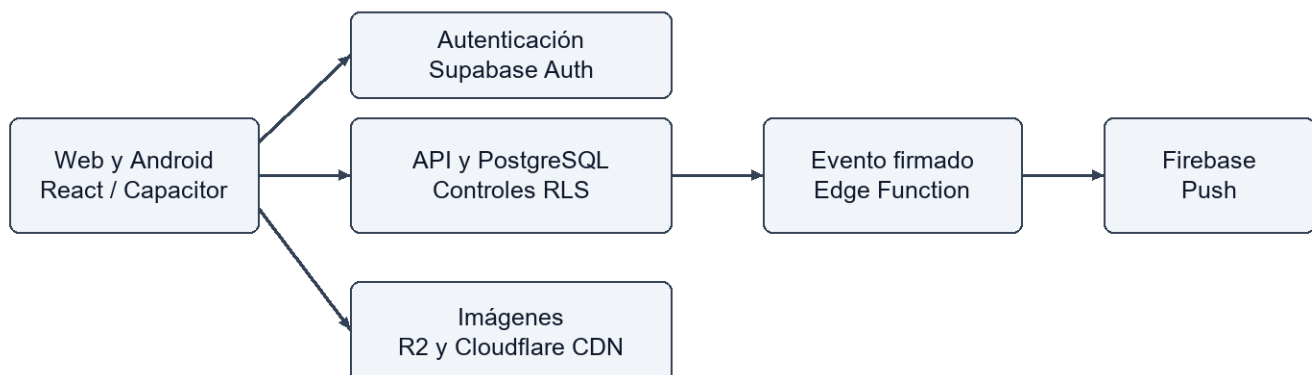
Pawsi se lanzó como webapp en septiembre de 2025 y ganó la Hackathon del Milagro en Salta Capital, Argentina, con su MVP web. Continué su desarrollo hasta publicar la versión Android con Capacitor en Google Play en octubre de 2025. El proyecto tuvo difusión en medios locales y participé en más de 30 entrevistas para presentarlo. La aplicación sigue funcionando, aunque su actividad disminuyó después del pico inicial de usuarios.

La experiencia incluyó construir el producto, publicarlo, atender su operación y explicar su utilidad a públicos no técnicos. La revisión de seguridad surgió de esa responsabilidad de mantenimiento.

Arquitectura y decisiones

La interfaz utiliza React y TypeScript. Capacitor permite distribuir el cliente en Android. Supabase aporta autenticación, PostgreSQL, políticas de acceso por fila y Edge Functions. Las imágenes se sirven mediante almacenamiento de objetos R2 y el CDN de Cloudflare. Firebase Cloud Messaging entrega las notificaciones push.

El almacenamiento de imágenes y la base de datos tienen ciclos de recuperación distintos: restaurar los registros no recupera automáticamente los archivos. La política de caché del dominio de medios mantiene los recursos durante un año tanto en Cloudflare como en el navegador. Esa decisión exige considerar la persistencia de copias al reemplazar o retirar imágenes; purgar el CDN no borra la caché de los navegadores.



Esquema simplificado de los servicios del caso.

Un laboratorio antes de modificar producción

Preparé Supabase local sobre Docker y WSL. El proyecto agrupa varios contenedores, entre ellos PostgreSQL, autenticación, API, almacenamiento y panel de administración. El frontend se configuró con la URL y las credenciales del entorno local. Las integraciones externas quedaron deshabilitadas o sustituidas para evitar envíos y operaciones contra producción durante las pruebas.

El primer arranque mostró que las migraciones disponibles no reconstruían por sí solas una base vacía: una migración intentaba alterar una tabla que aún no existía. Para el ensayo utilicé una restauración selectiva del respaldo PostgreSQL y contrasté los conteos de tablas y registros. También comprobé que las pruebas posteriores no dejaran datos sintéticos en la base restaurada.

El ensayo validó una recuperación parcial de estructura y datos. No incluyó los archivos de almacenamiento, las sesiones activas ni toda la configuración de los proveedores. Por eso no lo presento como una recuperación integral ni como una medición de RTO o RPO.

Corrección del envío de notificaciones

El flujo anterior permitía que el cliente solicitara el envío de una notificación. El cambio trasladó la decisión al backend: una inserción válida en la tabla de mensajes genera el evento que habilita el envío. El destinatario se obtiene de ese evento, sin confiar en una identidad arbitraria enviada al endpoint.

El flujo actual es el siguiente:

1. El cliente autenticado guarda un mensaje sujeto a los controles de la base de datos.
2. Un trigger de inserción crea el aviso interno y registra un evento privado.
3. La base firma la solicitud con HMAC y una marca temporal. El secreto se conserva en Vault y en la configuración privada de la función.
4. La Edge Function valida la firma y la vigencia. Una operación atómica permite reclamar el evento una sola vez.
5. Si corresponde enviar, la función llama a Firebase y registra el resultado.

La función antigua permanece como punto de compatibilidad sin realizar envíos. Esto permitió conservar los clientes publicados sin exigir una actualización de la aplicación Android. Los cambios de estado de lectura no generan nuevos avisos.

Agregué límites de frecuencia, vencimiento de eventos y controles de duplicación. Activar el envío no recorre mensajes históricos. Los eventos que exceden los límites se omiten; no forman una cola que pueda liberar notificaciones acumuladas más tarde.

En el laboratorio se ejecutaron 23 comprobaciones con Auth, PostgreSQL, pg_net y Edge Runtime locales, usando un proveedor push simulado. Se incluyeron firmas inválidas o vencidas, solicitudes repetidas, concurrencia, límites de frecuencia y fallos de configuración o proveedor.

Durante el despliegue apareció una respuesta 503 porque el secreto de la Edge Function tenía un nombre distinto al esperado por el código. Corregí la configuración y comprobé la recepción en Android. Una llamada sin firma recibió 401. Estas verificaciones complementan las pruebas locales; no equivalen a repetir todos los escenarios en producción.

Elegí un único intento de envío por evento, sin reintentos automáticos. Esto reduce el riesgo de duplicación, pero admite pérdida del push si el proceso falla después de reclamar el evento. La notificación interna y el push son mecanismos distintos.

Corrección de permisos de mensajes

La política RLS identificaba al destinatario como la persona autorizada a actualizar un mensaje. Sin embargo, el permiso SQL de UPDATE alcanzaba todas sus columnas. La intención era permitir marcar como leído, pero un destinatario podía también cambiar el asunto u otros campos.

Reproduce el cambio de asunto con cuentas sintéticas en el laboratorio. Después retiré el permiso general de actualización y concedí únicamente UPDATE sobre read_at al rol de usuarios autenticados. Conservé RLS para decidir qué filas puede modificar cada usuario.

La combinación tiene dos responsabilidades: RLS restringe los mensajes accesibles y los privilegios por columna restringen qué se puede cambiar dentro de ellos. El frontend ya actualizaba solamente read_at, de modo que la corrección no requirió modificarlo.

Las 15 comprobaciones locales incluyeron la reproducción inicial, la lectura legítima, el bloqueo de las otras nueve columnas, los intentos del remitente y de un tercero, el acceso anónimo y la ausencia de nuevos eventos al marcar como leído. En producción verifiqué los permisos efectivos y el funcionamiento de la lectura entre dos cuentas. No intenté alterar mensajes reales para demostrar el abuso.

Resultados y alcance

Recuperación local. Restauración selectiva y comparación de conteos. No demuestra recuperación de archivos o de todas las integraciones.

Notificaciones. 23 comprobaciones locales, recepción Android y rechazo sin firma en producción. El proveedor se simuló en el laboratorio; no se certifica todo el sistema.

Permisos de mensajes. 15 comprobaciones locales, privilegios efectivos y lectura legítima en producción. La corrección se limita a la actualización de mensajes.

La revisión me permitió convertir reglas implícitas del producto en controles del backend y pruebas verificables. También dejó una distinción importante para la recuperación: disponer de un respaldo y demostrar qué permite restaurar son tareas diferentes.

El material público se limita a este alcance. Los registros operativos, los respaldos, las credenciales y la información de usuarios permanecen fuera del caso de estudio.

Prensa y contacto

- InSalta — Una app salteña que reúne a mascotas perdidas con sus dueños
- Qué Pasa Salta — Pawsi y las mascotas perdidas en Salta
- Agenda Salta — La app creada en Salta que ayuda a encontrar mascotas perdidas
- Diego Puzio en LinkedIn

Las notas ofrecen contexto público del producto; no verifican los resultados técnicos de esta revisión. El código operativo y los respaldos se mantienen privados.