• RUNBOOK · SOFTWARE DE CERCANÍA

Plan técnico de implementación

Para tu gente · lo que se requiere, paso a paso

El detalle técnico, accionable, de cada una de las seis apps: fases, tareas concretas, dependencias y próximos pasos. Es el complemento operativo del documento de proyecto y del dossier.

Sembremos BANCO DE TIEMPO VECINAL

Del 85% desplegado a una red abierta y sostenible.

Objetivo y alcance

"Terminado" significa: que una persona ajena al círculo de demo pueda registrarse con su propio correo, entrar a su barrio, invitar a vecinos a un círculo, intercambiar favores con seguridad razonable, y que el proyecto esté bajo control de versiones y respaldado. No incluye —explícitamente fuera de alcance en esta etapa— app nativa de tiendas, notificaciones push en tiempo real ni un sistema de reputación completo; esos quedan como horizonte posterior al piloto.

Estado de partida: MVP funcional ~85%, desplegado punta a punta (frontend en `sembremos.pages.dev`, API en `sembremos-api.andrestaborga.workers.dev`, base D1 real). Auth, motor de reciprocidad, ciclo solicitar→aceptar→confirmar, chat y círculos ya operan.

Fases

Fase 0 — Blindaje del proyecto *(antes de tocar nada más)*

Poner una red de seguridad antes de seguir construyendo.

- ☐ `git init` en `Pasanaku\`, primer commit del estado actual, `.gitignore` para `.wrangler/`, `node_modules/` y `dist/`.
- ☐ Publicar el repo en GitHub (privado) como respaldo de historia.
- ☐ Exportar/respaldar la base D1 remota antes de cualquier migración (`wrangler d1 export`).

Fase 1 — Abrir la puerta (registro público real)

El cuello de botella número uno: hoy solo entran cuentas `@demo.bo`.

- ☐ Crear cuenta en Resend y verificar un dominio remitente (o usar el dominio de pruebas inicialmente).
- ☐ Cargar `RESEND_API_KEY` como secret del Worker (`wrangler secret put RESEND_API_KEY`).
- ☐ Verificar la rama de envío de enlace mágico en `worker/src/index.js` para emails no-demo.
- ☐ Probar el flujo completo de registro con un correo real externo.

Fase 2 — Encender la viralidad (invitaciones a círculos)

Hoy el botón "Invitar" solo muestra un toast (`app.js:542`).

- ☐ Backend: endpoint para generar un enlace de invitación con token por círculo (tabla nueva o reuso de `circles`).
- ☐ Backend: endpoint para canjear la invitación y unir al usuario al círculo.
- ☐ Frontend: reemplazar el toast por la generación y copia del enlace; vista de "unirse a círculo" al abrir el enlace.
- ☐ Probar invitación entre dos cuentas reales.

Fase 3 — Dar confianza (seguridad y bases de moderación)

- [ ] Rate-limiting en `/api/auth/request` (evitar abuso del envío de enlaces).
- [ ] Expiración y limpieza de tokens/sesiones vencidos.
- [ ] Reporte básico de un usuario/siembra (semilla de moderación, sin panel completo aún).
- [ ] Revisar CORS y orígenes permitidos para producción.

Fase 4 — Sanear el despliegue

Eliminar el riesgo de publicar versiones viejas.

- [ ] Decidir entre desplegar desde la raíz o automatizar la copia a `dist/` con un script (`npm run build:copy`).
- [ ] Documentar el comando único de deploy en el README.
- [ ] Confirmar separación fuente↔deploy acorde a tu convención.

Fase 5 — Piloto en un barrio real

- [ ] Elegir un barrio acotado (p. ej. uno de Sopocachi/Miraflores/Calacoto ya semillado).
- [ ] Reclutar 10-20 vecinos reales; sembrar contenido inicial real.
- [ ] Observar dos semanas: ¿se cierran favores?, ¿el medidor de capacidad se entiende?, ¿hay fricción en el registro?
- [ ] Iterar sobre lo que el piloto revele.

Dependencias y orden

```
Fase 0 (blindaje) ──► todo lo demás
Fase 1 (email)    ──► Fase 5 (piloto)        [sin registro real no hay piloto]
Fase 2 (círculos) ──► Fase 5 (piloto)        [la viralidad se prueba en el piloto]
Fase 3 (seguridad)──► Fase 5 (piloto)        [no abrir a desconocidos sin esto]
Fase 4 (deploy)   ──► independiente, hacer en cualquier momento tras Fase 0
```

La **Fase 0 bloquea todo** (no construir sin red de seguridad). Las fases 1, 2 y 3 son paralelizables entre sí pero **todas preceden al piloto**. La Fase 4 puede intercalarse cuando convenga.

Riesgos y mitigación

Riesgo	Impacto	Mitigación
Reiniciar la D1 remota por error (`db:init:remote` "reinicia todo")	Pérdida de datos reales	Respaldo antes de tocar (Fase 0); renombrar el script a algo explícito; nunca correrlo sin export previo
Abrir registro sin seguridad	Spam, abuso del enlace mágico	Hacer Fase 3 antes o junto a Fase 1; no anunciar públicamente hasta tener rate-limiting
Publicar versión vieja desde `dist/`	Bug en producción	Fase 4: un solo comando de deploy, sin copia manual

Riesgo	Impacto	Mitigación
Dependencia de una sola cuenta Cloudflare	Proyecto muere con su autor	Documentar accesos; evaluar titularidad institucional antes de escalar
Falta de confianza entre desconocidos	La red no despega fuera de conocidos	Mantener el piloto entre vecinos reales; introducir reputación tras el piloto

---

Cronograma

> No incluido en este nivel (Estándar). Para un cronograma con estimaciones por fase, vuelve a correr `/analizar-proyecto Pasanaku` en nivel **Exhaustivo** o dame las fechas/horas-semana de las que dispones y lo calculo.

Próximo paso recomendado

Empezar por la Fase 0 hoy mismo (`git init` + respaldo de la D1): es de bajo esfuerzo y desbloquea todo lo demás con seguridad. Inmediatamente después, la **Fase 1 (email real)** es la que más valor libera: convierte una demo cerrada en un producto abierto.

De Mano en Mano TRUEQUE ENTRE VECINOS

De un prototipo visual a una app real, con motor y comunidad.

Objetivo y alcance

"Terminado" significa: que una persona invitada y avalada pueda registrarse, entrar a su círculo en Sopocachi, publicar un objeto real con foto, proponer y cerrar un trueque 1:1 con estados persistentes, coordinar el encuentro por mensajería y que todo —identidad, objetos, avales, trueques— viva en un backend real y respaldado. Convertir, en una frase, **el prototipo navegable en una app funcional**.

Fuera de alcance en esta etapa: app nativa en tiendas (la PWA basta para el piloto), notificaciones push en tiempo real, sistema de reputación completo más allá de los avales, y las ferias presenciales convocadas (son v2 del concepto). Quedan como horizonte posterior al piloto.

Estado de partida: prototipo visual solo-frontend, ~95% como demo y ~10–15% como producto. Publicado en `de-mano-en-mano.pages.dev`. Toda la capa de presentación existe y es de alta fidelidad; **no existe nada de backend**. No hay git.

Fases

> Las fases técnicas (F0–F5) se mapean sobre las **4 fases económicas** del `Plan-Economico.html` así: F0 cae dentro de **F0 Validación** (~~$7k~~); ~~F1–F4 son el grueso de **F1 MVP**~~ ($35k); F5 abre la transición a **F2 Crecimiento** (~$75k).

F0 — Blindaje del proyecto *(antes de tocar nada)* · económica: F0 Validación

Poner una red de seguridad sobre lo que ya existe.

- ☐ `git init` en la carpeta; primer commit del estado actual del prototipo.
- ☐ `.gitignore` para `.wrangler/`, `node_modules/` y `dist/`.
- ☐ Publicar el repo en GitHub (privado) como respaldo de historia.
- ☐ Resolver la duplicación raíz↔`dist/`: definir la raíz como fuente única y `dist/` como salida de build (o desplegar desde la raíz).

F1 — El motor: backend mínimo · económica: F1 MVP

Construir la fuente de verdad que el prototipo no tiene.

- ☐ Scaffolding del Worker (Cloudflare) + base **D1**; `wrangler.jsonc` con binding.
- ☐ **Modelo de datos** (esquema D1): `usuarios`, `objetos`, `trueques`, `avales`, `puntos` (de encuentro), y tablas de apoyo (`circulos`/anillos, `mensajes`).
- ☐ Endpoints de lectura que reemplacen los datos quemados (`OBJ`, `POINTS`, `MINE`) por consultas reales.
- ☐ CORS y orígenes permitidos para el dominio de Pages.

F2 — Identidad: auth + invitaciones y avales · económica: F1 MVP

La puerta de entrada y, a la vez, el motor de crecimiento.

- [ ] Autenticación sin fricción (enlace mágico al correo, vía proveedor tipo Resend; o equivalente).
- [ ] Sistema de **invitaciones**: generar y canjear enlaces que unen a alguien a un círculo (hoy el onboarding "María González te invitó" es decorativo).
- [ ] Sistema de **avales**: registrar quién avala a quién; reflejarlo en el perfil y en la Vitrina ("te avala María").
- [ ] Anillos y "red de redes": visibilidad por 1°/2° anillo según la cadena de avales.

F3 — Publicar objeto real (cámara/fotos) · económica: F1 MVP

Dar vida a los objetos, reemplazando los *placeholders* ("Aquí abrirías cámara...").

- [ ] Captura/selección de foto desde el móvil; subida a **R2** (almacenamiento de objetos).
- [ ] Formulario real de publicación: título, categoría/chips, descripción, fotos.
- [ ] Inicio de la **Historia del objeto** persistente: primer nodo (origen) al publicar.

F4 — Proponer y cerrar trueque + estados + mensajería · económica: F1 MVP

Cerrar el ciclo del concepto con datos reales.

- [ ] Flujo **proponer** → **aceptar** → **confirmar** con estados persistentes por trueque.
- [ ] Al cerrar un trueque: avanzar la **Historia del objeto** (nueva mano) y actualizar stats.
- [ ] **Mensajería** por encuentro (coordinar lugar y hora); selección de punto de comunidad.
- [ ] Confirmación de encuentro que, con el uso, **valida puntos** ("✓ validado por la red").

F5 — Piloto en Sopocachi · económica: abre F2 Crecimiento

- [ ] Reclutar 30–60 vecinos reales de máxima confianza (familia y amigos), avalados.
- [ ] Sembrar objetos reales; coordinar los primeros trueques presenciales.
- [ ] Observar 2–4 semanas: ¿se cierran trueques?, ¿se entiende el aval?, ¿hay fricción al publicar?, ¿la doble coincidencia frena el intercambio?
- [ ] Iterar sobre lo que el piloto revele; recoger métricas para la solicitud de F2.

Dependencias y orden

```
F0 (blindaje) ──────────► todo lo demás
F1 (backend)  ──────────► F2, F3, F4   [sin modelo de datos no hay nada]
F2 (auth+avales) ───────► F5           [sin identidad real no hay piloto]
F3 (publicar) ──────────► F4           [hay que tener objetos para truecarlos]
F4 (trueque+chat) ──────► F5           [el ciclo completo se prueba en el piloto]
```

F0 bloquea todo (no construir sin red de seguridad). **F1 es el cimiento**: F2, F3 y F4 dependen de su modelo de datos. F5 (piloto) requiere que F2, F3 y F4 estén operativas. Dentro de F1 MVP económica, el orden técnico natural es F1 → F2 → F3 → F4.

Riesgos y mitigación

Riesgo	Impacto	Mitigación
Construir backend desde cero (no es un MVP casi listo)	Mucho trabajo antes de ver valor nuevo	Empezar por el modelo de datos (F1) y reemplazar datos quemados gradualmente; mantener el prototipo como referencia viva de UX
La confianza no está resuelta (avales/moderación inexistentes)	Red frágil al abrirse a desconocidos	Mantener el piloto entre conocidos avalados; moderación mínima desde F2; reputación tras el piloto
Doble coincidencia del trueque	Pocos intercambios se concretan	Tablón de pedidos abierto (ya en el diseño); medir y, si hace falta, explorar trueques en cadena o "fichas de favor" sin romper el espíritu gratuito
Seguridad de encuentros presenciales	Riesgo real, delegado a lo comunitario	Default fuerte de puntos de comunidad + acompañamiento + validación por uso; nunca prometer escrow que no existe
Arranque en frío (~100 usuarios densos/zona)	La red no despega	Anillos cerrados de familia; lugares "sugeridos / aún sin estrenar"; densidad antes que alcance
Sin git / duplicación raíz↔dist	Pérdida de trabajo, deploy de versión vieja	F0 lo resuelve antes que nada
Dependencia de una sola cuenta Cloudflare	El proyecto muere con su autor	Documentar accesos; evaluar titularidad institucional antes de escalar

---

Cronograma relativo

Alineado con el plan económico (30 meses, 4 fases). En términos relativos del trabajo técnico:

- **F0** — días (bajo esfuerzo, desbloquea todo).
- **F1–F4 (el MVP real)** — el grueso del esfuerzo; encaja en la ventana **Mes 3–9 / F1 MVP** del plan económico (1 dev full-stack + diseño a medio tiempo, según el presupuesto).
- **F5 (piloto)** — semanas de observación; abre la solicitud de **F2 Crecimiento**.

> Para un cronograma con fechas y horas-semana concretas, indícame tu disponibilidad y lo calculo.

Próximo paso recomendado

Hoy: F0 (`git init` + respaldo + resolver la duplicación raíz↔ `dist/`). Es de bajo esfuerzo y protege todo lo construido. **Inmediatamente después: F1** —el modelo de datos en D1—, porque es el cimiento sin el cual ninguna otra pieza del concepto puede dejar de ser ficción quemada. Con el prototipo como guía de UX, la construcción tiene un mapa claro y un riesgo de diseño bajo.

Libracus TU BIBLIOTECA, COMPARTIDA

Del 80-85% desplegado a un producto blindado, abierto y sostenible.

Objetivo y alcance

"Terminado" significa: que el proyecto esté a salvo (bajo control de versiones con respaldo remoto), que la catalogación funcione en cualquier dispositivo (no solo Chrome/Android), que el descubrimiento crezca más allá del enlace por código, que la documentación refleje la realidad, que exista una red mínima de pruebas/CI y que haya una vía de ingreso activable (freemium de escaneos). Queda **explícitamente fuera de alcance** en esta etapa: publicación en tiendas nativas (App Store / Play), notificaciones push en tiempo real y un sistema de reputación/moderación completo. Eso es horizonte posterior.

Estado de partida: MVP funcional ~80–85%, desplegado punta a punta (frontend en `libracus.pages.dev`, API en `libracus-api.andrestaborga.workers.dev`, base D1 real con datos en producción). Escaneo de lomos, ISBN, alta manual, CRUD de biblioteca/estantes/temas/portadas, auth completa (Google + correo, verificación, reset, `claimDevice`) y red social con persistencia D1 real **ya operan**.

Fases

Fase 0 — BLINDAJE *(antes de tocar nada más — máxima urgencia)*

El repositorio tiene historia pero **no tiene remoto**, y toda la capa de auth/grupos/clubes/sugerencias/bibliotecas-guardadas + ~14 pantallas + `migrations/` + el respaldo SQL están **sin commitear** (62 entradas sucias verificadas: 26 modificadas, 35 sin trackear, 1 borrada). El código de esa capa **solo vive en este disco**. Esto es lo primero.

- ☐ Revisar el `.gitignore` para no subir secrets ni `node_modules/` / `dist/` / `.wrangler/`; confirmar que `api/backup-libracus-2026-06-11.sql` se respalda aparte (es dato real, no debería ir al repo de código en claro).
- ☐ `git add -A` y commit del estado completo (toda la capa social y las pantallas hoy untracked).
- ☐ **Crear un remoto privado** (GitHub `gh repo create libracus --private`) y `git push`. Este es el paso que elimina el riesgo de pérdida total.
- ☐ Guardar una copia del respaldo D1 (`api/backup-libracus-2026-06-11.sql`) fuera del disco (almacenamiento separado / OneDrive ya lo cubre parcialmente, verificar).
- ☐ Exportar de nuevo la base D1 remota como respaldo fresco antes de cualquier migración futura (`wrangler d1 export`).

Fase 1 — Sanear documentación y verdad del repo

Cerrar la brecha entre lo que el README dice y lo que el código hace.

- ☐ Actualizar `README.md`: quitar la mención a **R2** (la realidad es KV), corregir el estado (no "Fase 1 sin hacer": el MVP está ~85%), y listar los endpoints reales.
- ☐ Documentar el comando único de deploy (frontend y API) y los secrets requeridos (`GOOGLE_BOOKS_API_KEY`, OAuth, mailer).
- ☐ Dejar un `CHANGELOG` o nota de arquitectura que refleje la capa social conectada.

Fase 2 — Fallback de código de barras en iOS

Hoy `BarcodeScreen` depende de `BarcodeDetector`, presente solo en Chrome/Android.

- ☐ Evaluar una librería JS de lectura de ISBN que funcione en iOS Safari (p. ej. ZXing/ `@zxing/browser`) como respaldo cuando `BarcodeDetector` no exista.
- ☐ Mantener `BarcodeDetector` como vía rápida donde esté disponible; degradar con elegancia al lector JS.
- ☐ Probar el flujo ISBN → metadatos en un iPhone real.

Fase 3 — Descubrimiento por cercanía (geo)

Hoy el descubrimiento es por código/enlace (`/u/:code`); falta la dimensión geográfica.

- ☐ Decidir el modelo de privacidad: opt-in explícito, zona aproximada (no ubicación exacta), respeto al `share_code` existente.
- ☐ Backend: campo de zona/geohash en la cuenta; endpoint de "bibliotecas cerca" filtrado por opt-in.
- ☐ Frontend: pantalla de descubrimiento por cercanía que reutilice `SharedLibraryScreen`.
- ☐ Probar con cuentas reales en zonas distintas.

Fase 4 — Pruebas y CI mínimos

Reducir el riesgo de romper sin darse cuenta.

- ☐ Tests unitarios de las piezas críticas y silenciosas: parser de `spines.ts`, `matchCandidate` (umbral 0.34, volúmenes romanos/ES/EN), validaciones de `wishes.ts`.
- ☐ Un workflow de CI (GitHub Actions) que corra typecheck + tests en cada push.
- ☐ Smoke test del endpoint `/scan/spines` con una imagen fija de referencia.

Fase 5 — Evaluar migración de KV a R2 (condicional)

Solo si las portadas crecen.

- ☐ Medir el peso real de las portadas en KV (límite práctico ~24 MB/valor).
- ☐ Si se acerca al límite o crece el catálogo, planificar migración de `COVERS` (KV) a R2 con compatibilidad de `/covers/:key`.
- ☐ No hacerlo antes de tiempo: KV en plan gratuito hoy alcanza.

Fase 6 — Freemium de escaneos (activar la vía de ingreso)

El único costo variable es Workers AI por escaneo: cobrar exactamente por eso.

- ☐ Definir la cuota gratuita (p. ej. N escaneos/mes — **valor a calibrar contra el costo real medido**, hoy hipótesis).
- ☐ Backend: contador de escaneos por cuenta y mes; gate en `/scan/spines` al superar la cuota.
- ☐ Frontend: pantalla de plan, aviso de cuota, flujo de upgrade.
- ☐ Explorar **afiliación a librerías** sobre la mecánica de deseos como ingreso secundario.

Dependencias y orden

```
Fase 0 (blindaje)        ──► todo lo demás        [no construir sin red de seguridad]
Fase 1 (docs)            ──► independiente, hacer junto a F0
Fase 2 (barcode iOS)     ──► independiente
Fase 3 (geo)             ──► tras F0; se prueba con cuentas reales
Fase 4 (tests/CI)        ──► tras F0; protege todo lo que venga después
Fase 5 (R2)              ──► condicional, solo si KV se llena
Fase 6 (freemium)        ──► requiere medir el costo real de IA primero
```

La **Fase 0 bloquea todo**: es media tarde de trabajo y elimina el riesgo de pérdida total. Las fases 1–4 son paralelizables entre sí. La 5 es condicional. La 6 depende de tener una medición real del costo de escaneo antes de fijar la cuota.

Riesgos y mitigación

Riesgo	Impacto	Mitigación
Git sin remoto + capa entera sin commitear	Pérdida TOTAL del código más valioso si falla el disco	**Fase 0 hoy mismo**: commit + remoto privado + push. No avanzar en nada más antes de esto
Subir secrets o el respaldo SQL al repo	Fuga de credenciales / datos reales	Revisar `.gitignore` antes del primer push; respaldo SQL fuera del repo de código en claro
Calidad del modelo de visión en lomos difíciles	Catalogación incompleta, frustración	Ya mitigado (3 reintentos, parser tolerante, matching 0.34); sumar tests con imágenes reales (F4)
Cuota de Google Books / Open Library sin contrato	Cortes en el enriquecido	Concurrencia limitada a 3 ya implementada; vigilar cuotas; cachear resultados
KV se llena con portadas	Fallos al guardar portada	Medir peso (F5); migrar a R2 solo si hace falta
Fijar cuota freemium sin medir costo de IA	Modelo que pierde o ahuyenta	**Medir antes de cobrar**: la cuota es hipótesis hasta tener el costo real por escaneo
Barcode solo en Chrome/Android	Usuarios iOS sin respaldo ISBN	Fallback JS (ZXing) en F2

Cronograma relativo

> Secuencia relativa, no fechas. La **Fase 0 es de horas**, no de días, y desbloquea todo. Las fases 1–4 pueden intercalarse en bloques cortos. La 5 es reactiva (solo si KV se llena). La 6 requiere primero una fase de medición del costo real de Workers AI por escaneo —ese dato hoy no existe y no debe inventarse—. Para un cronograma con estimaciones por fase, indica las horas-semana de las que dispones y se calcula.

Próximo paso recomendado

Hacer la Fase 0 ahora mismo. Abre una terminal en `Libracus\`, revisa el `.gitignore`, `git add -A`, commitea todo (especialmente la capa de auth/social y las pantallas hoy untracked), crea un repo privado en GitHub y empuja. Es el paso de mayor retorno y menor esfuerzo de todo el plan: convierte un proyecto que **solo existe en un disco** en uno respaldado. Todo lo demás puede esperar; esto no.

PaProducir PRODUCCIÓN MUSICAL DEL GRUPO

Del 90 % usable al disco compartido, blindado y abierto a la banda.

Objetivo y alcance

"Terminado" significa: que toda la banda —no un solo correo— pueda entrar con su Google y trabajar sobre el mismo disco compartido; que la app se instale bien en Android; que dos personas editando a la vez no se pierdan trabajo; y que el proyecto esté bajo control de versiones, respaldado y con migraciones que no se puedan olvidar a medias.

Fuera de alcance en esta etapa (horizonte posterior, no condición de "terminado"): la apertura a múltiples bandas (F4, opcional) y la subida de audio propio a R2 (hoy `Env.AUDIO` está declarado pero no se usa; las demos se enlazan desde Drive/YouTube). Se nombran como posibilidades, no como compromisos.

Estado de partida: PWA funcional ~90 %, desplegada punta a punta (frontend en `paproducir.pages.dev`, Pages Functions desde `functions/`, base D1 `paproducir-db` real). Multi-álbum, Agenda, AudioCompare A/B, export PDF, arrastrar-y-soltar, snapshots con restauración y merge por canción ya operan.

Fases

Fase 0 — git + respaldo *(antes de tocar nada más)*

Poner una red de seguridad antes de seguir. Hoy **no hay git**.

- ☐ `git init` en `PaProducir\`, primer commit del estado actual.
- ☐ `.gitignore` para `node_modules/`, `dist/`, `.wrangler/` y secretos locales.
- ☐ Publicar el repo en GitHub (privado) como respaldo de historia.
- ☐ Exportar/respaldar la base D1 remota antes de cualquier migración (`wrangler d1 export paproducir-db --remote`).

Fase 1 — Iconos PNG (instalación en Android)

Hoy `public/` solo tiene `favicon.svg` y el manifest usa SVG; falta el set PNG.

- ☐ Generar iconos PNG a partir del favicon de ecualizador (al menos `192×192` y `512×512`, más una variante `maskable`).
- ☐ Añadir los PNG al manifest de `vite-plugin-pwa` junto al SVG.
- ☐ Verificar la instalación real en un Android (icono correcto en el lanzador, splash).

Fase 2 — Sumar integrantes (BAND_EMAILS + test users de Google)

Hoy `BAND_EMAILS` solo tiene `andrestaborga@gmail.com`: el disco "compartido" lo usa una persona.

- ☐ Recolectar los correos de Google de los integrantes de la banda.
- ☐ Añadirlos a `BAND_EMAILS` en `wrangler.jsonc` (separados por coma) y volver a desplegar.
- ☐ En Google Cloud Console → Google Auth Platform → Audience, agregarlos como *test users* (o publicar la app de OAuth si serán muchos).
- ☐ Probar el login y la sincronización con al menos un segundo correo real.

Fase 3 — Merge campo a campo

Hoy dos ediciones de la **misma** canción pierden la más antigua (el merge es por canción, no por campo).

- ☐ Definir la granularidad del merge en `lib/merge.ts` (campos de `Song`: etapa, tareas, letra, acordes, archivos, notas).
- ☐ Implementar la fusión por campo con marca de tiempo por campo, manteniendo el flujo optimista actual (`X-Base-Version` + 409 + reintento).
- ☐ Probar el caso de conflicto real: dos clientes editan campos distintos de la misma canción a la vez; ninguno pierde su cambio.
- ☐ Actualizar el README (que aún dice "el último que guarda gana") para reflejar el comportamiento real.

Fase 4 — Multi-banda *(opcional)*

Solo si alguna vez se quiere abrir a otras bandas. El frontend ya es multi-álbum; el backend está clavado a `ws_id = 'band'`.

- ☐ Parametrizar `ws_id` por grupo en `album.ts`, `versions.ts` y `restore.ts` (hoy literal `'band'`).
- ☐ Modelo de membresías: qué correo pertenece a qué banda (tabla nueva o ampliación de la allowlist).
- ☐ Ajustar el frontend para seleccionar/identificar la banda activa.
- ☐ Migrar el dato existente a un `ws_id` nombrado sin perder el disco actual.

Fase 5 — Script de migraciones D1

Hoy `schema.sql` y `schema-versions.sql` son archivos sueltos; olvidar el segundo rompe `album` / `versions` / `restore`.

- ☐ Reunir ambos esquemas en un único script (o carpeta `migrations/` ordenada) que se aplique de una vez.
- ☐ Documentar el comando único de migración (local y `--remote`) en el README.
- ☐ Probar la aplicación del esquema completo sobre una base limpia y confirmar que snapshots y restore funcionan.

Dependencias y orden

```
Fase 0 (git + respaldo) ──► todo lo demás
Fase 1 (iconos PNG)     ──► independiente, hacer cuando convenga tras F0
Fase 2 (integrantes)    ──► el paso que vuelve "compartido" el disco; alto valor, bajo esfu
Fase 3 (merge campo)    ──► gana relevancia DESPUÉS de F2 (sin más gente, el conflicto casi
Fase 4 (multi-banda)    ──► OPCIONAL; solo si se decide abrir a otras bandas
Fase 5 (migraciones)    ──► independiente; hacer junto a F0 si se va a tocar la base
```

La **Fase 0 bloquea todo** (no seguir sin red de seguridad). La **Fase 2** es la de mayor valor por esfuerzo: es lo que convierte la herramienta de "personal" a "de la banda". La **Fase 3** solo importa de verdad una vez que hay varias personas editando (es decir, tras la Fase 2). Las fases 1 y 5 son independientes y pueden intercalarse. La **Fase 4 es opcional** y queda como horizonte.

Riesgos y mitigación

Riesgo	Impacto	Mitigación
Tocar la base D1 sin respaldo	Pérdida del disco real	Respaldo antes de tocar (Fase 0); `wrangler d1 export` previo a toda migración
`PUT` corrupto sobre el blob único `id='band'`	Sobrescribe todo el disco	Validación mínima ya presente + snapshots cada 5 min + restore; reforzar en F3
Olvidar `schema-versions.sql` al migrar	Rompe snapshots y restore	Fase 5: script único de migraciones; documentar el comando
Caducidad del OAuth de Google / correo sin habilitar	Nadie entra (portón único)	Mantener vivo el proyecto de OAuth; checklist de *test users* en Fase 2; documentar el acceso
Edición simultánea de la misma canción	Se pierde la edición más antigua	Mitigado hoy por snapshots; resuelto de raíz en Fase 3 (merge campo a campo)
Sin git	Un cambio desafortunado no tiene vuelta atrás	Fase 0: `git init` + GitHub privado

---

Cronograma

> No incluido en este nivel. Para un cronograma con estimaciones por fase, indica las horas-semana de las que dispones y se calcula. Como referencia relativa de esfuerzo (no de fechas): **F0** y **F2** son de pocas horas; **F1** y **F5**, de medio día cada una; **F3** es la más sustancial; **F4** es un proyecto aparte y opcional.

Próximo paso recomendado

Empezar hoy por la Fase 0 (`git init` + respaldo de la D1): bajo esfuerzo, desbloquea todo con seguridad. Inmediatamente después, la **Fase 2 (sumar a la banda)** es la que más valor libera: convierte un cuaderno personal en el cuaderno compartido que la app promete ser.

PaCocinarnos RECETARIO DE LA CASA

Del ~90% publicado a un recetario blindado, con cara y comunidad cuidada.

Objetivo y alcance

"Terminado" significa: que el proyecto esté bajo control de versiones y respaldado (que el deploy deje de ser la única copia), que más recetas tengan foto real, que las versiones que aporta la comunidad pasen por una moderación mínima, que el login con Google quede verificado de punta a punta, y que se haya tomado una decisión clara sobre el bilingüismo de las versiones de usuario. No incluye —explícitamente fuera de alcance en esta etapa— app nativa de tiendas, notificaciones push, ni un sistema de reputación de la comunidad; esos quedan como horizonte posterior.

Estado de partida: PWA funcional ~90%, desplegada en `pacocinarnos.pages.dev`, con 80 entradas de contenido (76 recetas + 4 básicos) todas en estado `ready`, glosario de 22 términos, capa social sobre Supabase (proyecto `wjsvhfubqigjpyeexzdf`) con RLS sólido. Sin marcadores `[TODO]` pendientes en el contenido.

Fases

Fase 0 — Blindaje del proyecto *(antes de tocar nada más)*

Poner una red de seguridad antes de seguir: hoy el deploy es la única copia.

- ☐ `git init` en `PaCocinarnos\`, primer commit del estado actual.
- ☐ `.gitignore` para `.wrangler/`, `node_modules/` y cualquier artefacto local.
- ☐ Publicar el repo en GitHub (privado) como respaldo de historia.
- ☐ Exportar un respaldo del esquema y los datos de Supabase (`recipe_versions`, `favorites`, `profiles`) antes de cualquier migración futura.

Fase 1 — Darle cara (más fotos reales)

Hoy solo 5 de 80 entradas tienen foto; el resto usa emoji.

- ☐ Priorizar las recetas más buscadas / emblemáticas para fotografiar primero.
- ☐ Añadir las imágenes a `img/` (o `img/refs/`) y referenciarlas en `js/data.js`.
- ☐ Mantener un peso razonable por imagen (la app es cache-first; cuidar el tamaño del caché del `sw.js`).
- ☐ Subir la versión del *service worker* (`sw.js`) al cambiar los assets cacheados.

Fase 2 — Cuidar la comunidad (moderación mínima)

Hoy las versiones publicadas (`recipe_versions`) aparecen sin revisión.

- ☐ Definir el modelo de moderación más simple que funcione: por ejemplo un campo de estado (`pending` / `approved`) en `recipe_versions` o una marca de "reportado".
- ☐ Ajustar las políticas RLS y la vista `versions_with_author` para que la comunidad muestre solo lo aprobado (o todo menos lo reportado, según se decida).

- [ ] Frontend (`js/app.js`): vista mínima de revisión para el custodio, o al menos un botón de "reportar" en cada versión.
- [ ] Probar el flujo con una versión de prueba.

Fase 3 — Verificar la puerta (Google OAuth)

La configuración de OAuth vive en el panel de Supabase y no es verificable desde el código.

- [ ] Confirmar en el panel de Supabase que el proveedor Google está bien configurado (client ID, redirect URLs de producción).
- [ ] Probar el login con Google de punta a punta con una cuenta real externa.
- [ ] Verificar que el trigger `handle_new_user` crea el perfil correctamente al entrar por Google.
- [ ] Confirmar la migración de favoritos de `localStorage` a la nube tras el primer login.

Fase 4 — Pulir lo bilingüe de la comunidad

Las versiones de usuario se guardan en un solo idioma, a diferencia del contenido oficial.

- [ ] Decidir la política: aceptar versiones monolingües marcándolas con su idioma, o pedir ambos idiomas en el editor.
- [ ] Reflejar la decisión en el editor (`js/app.js`) y en la forma de almacenar el `jsonb` de `recipe_versions`.
- [ ] Mostrar en la comunidad el idioma de cada versión para no confundir al lector.

Fase 5 — Difusión en la comunidad real

- [ ] Compartir la app en espacios de la comunidad boliviana en DC (grupos, asociaciones, parroquias).
- [ ] Invitar a aportar las primeras versiones reales para que la comunidad no esté vacía.
- [ ] Observar qué recetas se buscan más y qué sustituciones generan dudas; iterar el contenido sobre eso.

Dependencias y orden

```
Fase 0 (blindaje)   ──► todo lo demás
Fase 1 (fotos)      ──► independiente, hacer en cualquier momento tras Fase 0
Fase 2 (moderación) ──► Fase 5 (difusión)   [no abrir comunidad sin red de seguridad]
Fase 3 (OAuth)      ──► Fase 5 (difusión)   [el login debe funcionar antes de invitar gente
Fase 4 (bilingüe)   ──► conviene antes de la Fase 5, pero no la bloquea
```

La **Fase 0 bloquea todo** (no construir sin red de seguridad). Las fases 2 y 3 preceden a la difusión: no conviene abrir la comunidad a más gente sin moderación mínima ni sin login verificado. Las fases 1 y 4 son paralelizables y pueden intercalarse cuando convenga.

Riesgos y mitigación

Riesgo	Impacto	Mitigación
El deploy es la única copia (sin git)	Pérdida irrecuperable del proyecto	Fase 0 primero: git + repo privado + respaldo de Supabase
Comunidad sin moderación	Contenido inapropiado o spam visible	Fase 2 antes de difundir; al menos un botón de reportar
OAuth mal configurado (no verificable en código)	Usuarios no pueden entrar con Google	Fase 3: probar de punta a punta con cuenta real
Versiones en un solo idioma	Confusión en una app bilingüe	Fase 4: marcar idioma de cada versión como mínimo
Caché del service worker pesado al sumar fotos	App lenta o caché inflado	Cuidar peso de imágenes; subir versión del `sw.js`
Dependencia de una cuenta personal (Cloudflare + Supabase)	El proyecto muere con su autor	Documentar accesos; evaluar custodia institucional antes de escalar

---

Cronograma

> Estimaciones relativas, sin fechas inventadas. La **Fase 0** es de muy bajo esfuerzo y va primero. Las fases 1 a 4 son de esfuerzo medio y en buena parte paralelizables. La **Fase 5** es continua (difusión y observación). Si quieres un cronograma con horas-semana concretas, indícame de cuánto tiempo dispones por semana y lo calculo.

Próximo paso recomendado

Empezar por la Fase 0 hoy mismo (`git init` + repo privado + respaldo de Supabase): es de bajo esfuerzo y elimina el riesgo más serio del proyecto —que toda la obra dependa de una sola copia en el deploy—. Inmediatamente después, la **Fase 1 (más fotos reales)** es la que más mejora la experiencia visible con menos complejidad técnica.

Discasus TU COLECCIÓN DE MÚSICA, COMPARTIDA

Del 85-90% desplegado a un producto blindado, completo y sostenible.

Objetivo y alcance

"Terminado" significa: que el proyecto esté a salvo (bajo control de versiones con respaldo remoto), que los datos que ya existen en el modelo (conservación Goldmine, enlace de escucha, RPM y color de vinilo) tengan **interfaz** para mostrarse y capturarse, que el código deje de arrastrar la nomenclatura de Libracus, que los errores de texto y los assets heredados queden corregidos, que se decida la cuestión de los IDs commiteados y la eventual migración a R2, y que haya una vía de ingreso activable (freemium de escaneos). Queda **explícitamente fuera de alcance** en esta etapa: publicación en tiendas nativas (App Store / Play), notificaciones push en tiempo real y un sistema de reputación/moderación completo.

Estado de partida: MVP funcional ~85–90%, desplegado punta a punta (frontend en `discasus.pages.dev`, API en `discasus-api.andrestaborga.workers.dev`, base D1 real, `DISCOGS_TOKEN` operativo en producción). Escaneo de portada, código de barras, alta manual, CRUD de colección/estantes/temas, auth completa (Google + correo, verificación, reset) y red social con persistencia D1 real (grupos, clubes de escucha, deseos, sugerencias) **ya operan**.

Fases

Fase 0 — BLINDAJE *(antes de tocar nada más — máxima urgencia)*

El proyecto **no está bajo control de versiones**: no existe repositorio git (`git status` → *"not a git repository"*). **Todo el código vive en un solo disco, sin historia ni remoto.** Si el disco falla, se pierde el producto completo. Esto es lo primero. *(En un blindaje aparte ya está en curso; se documenta igual por su criticidad.)*

- ☐ `git init` en `Discasus\`.
- ☐ Crear un `.gitignore` que excluya `node_modules/`, `dist/`, `.wrangler/` y cualquier archivo de secrets local; confirmar que no se suben tokens ni respaldos de datos en claro.
- ☐ Decidir sobre `database_id` y el `id` de KV ya escritos en `api/wrangler.jsonc`: no son credenciales, pero contradicen la nota de privacidad del proyecto. Mantenerlos o externalizarlos antes del primer push público.
- ☐ `git add -A` y commit del estado completo.
- ☐ **Crear un remoto privado** (GitHub `gh repo create discasus --private`) y `git push`. Este es el paso que elimina el riesgo de pérdida total.
- ☐ Exportar la base D1 remota como respaldo fresco (`wrangler d1 export discasus`) y guardarlo fuera del disco.

Fase 1 — UI de conservación Goldmine y enlace de escucha *(la brecha funcional más clara)*

Los campos ya existen en el schema, en los tipos y en el cliente de API, pero **no tienen interfaz**: no se muestran ni se capturan.

- [ ] **Pantalla de detalle del disco:** mostrar `media_condition` y `sleeve_condition` (Goldmine), `rpm`, `vinyl_color`, `disc_count` y un botón al `listen_url` (Spotify/YouTube/Bandcamp) cuando exista.
- [ ] **Pantalla de edición:** selector de grados Goldmine (M, NM, VG+, VG, G…) para disco y funda por separado; campo de `listen_url`; campos de RPM y color de vinilo (visibles solo cuando el formato es `vinyl`).
- [ ] Mostrar la **tracklist** (ya se descarga de Discogs) en el detalle, si aún no se renderiza.
- [ ] Verificar que el `share_status` (`private | showcase | exchange | gift`) sea editable desde la UI.

Fase 2 — Limpieza de nomenclatura heredada de Libracus

El frontend arrastra nombres de libros que mienten sobre lo que hacen.

- [ ] Renombrar `type BookMetadata` → `ReleaseMetadata` (ya existe como alias; consolidar).
- [ ] Renombrar `lookupByIsbn` → `lookupByBarcode` (recibe un barcode EAN/UPC, no un ISBN).
- [ ] Renombrar `scanSpines` → `scanCover` (escanea portadas, no lomos).
- [ ] Renombrar componente `BookCover` → `ReleaseCover` y pantallas `BookDetailScreen` / `BookEditScreen` → `ReleaseDetailScreen` / `ReleaseEditScreen`.
- [ ] Hacerlo con búsqueda-y-reemplazo cuidadosa y un typecheck completo después de cada renombre.

Fase 3 — Corregir el texto "los portadas" y revisar assets heredados

Pulido visible de poca dificultad, alto impacto de percepción.

- [ ] Corregir en `ScanScreen` el texto *"Acércate a los portadas"* → *"Acércate a la portada"* (o "a las portadas").
- [ ] Revisar los `group-logos` heredados de Libracus en `dist` / `public`: confirmar cuáles se usan, reemplazar o eliminar los que no correspondan a la identidad de Discasus.
- [ ] Barrido rápido de otros textos que mencionen "libro", "lomo" o "ISBN" en la UI.

Fase 4 — Evaluar migración de KV a R2 (condicional)

Solo si las portadas crecen.

- [ ] Medir el peso real de las portadas en KV (límite práctico ~24 MB/valor).
- [ ] Si se acerca al límite o crece el catálogo, planificar migración de `COVERS` (KV) a R2 manteniendo compatibilidad de las rutas de portada.
- [ ] No hacerlo antes de tiempo: KV en plan gratuito hoy alcanza.

Fase 5 — Freemium de escaneos (activar la vía de ingreso)

El único costo variable es Workers AI por escaneo: cobrar exactamente por eso.

- [ ] **Medir primero** el costo real de Workers AI por escaneo de portada (no existe ese dato hoy; sin él, cualquier cuota es inventada).
- [ ] Backend: contador de escaneos por cuenta y mes; gate en `/scan/cover` al superar la cuota.
- [ ] Frontend: pantalla de plan, aviso de cuota, flujo de upgrade.
- [ ] Definir la cuota gratuita (p. ej. N escaneos/mes — **valor a calibrar contra el costo real medido**, hoy hipótesis).
- [ ] Explorar ingresos secundarios afines al nicho coleccionista (sin comprometer la experiencia).

Dependencias y orden

```
Fase 0 (blindaje)            ──► todo lo demás        [no construir sin red de seguridad]
Fase 1 (UI Goldmine/listen)  ──► tras F0; mayor valor funcional
Fase 2 (nomenclatura)        ──► tras F0; independiente de F1
Fase 3 (texto/assets)        ──► tras F0; trivial, paralelizable
Fase 4 (R2)                  ──► condicional, solo si KV se llena
Fase 5 (freemium)            ──► requiere medir el costo real de IA primero
```

La **Fase 0 bloquea todo**: es media tarde de trabajo y elimina el riesgo de pérdida total. Las fases 1–3 son paralelizables entre sí. La 4 es condicional. La 5 depende de tener una medición real del costo de escaneo antes de fijar la cuota.

Riesgos y mitigación

Riesgo	Impacto	Mitigación
Sin git / sin remoto	Pérdida TOTAL del código si falla el disco	**Fase 0 hoy mismo**: `git init` + commit + remoto privado + push. No avanzar en nada más antes de esto
`database_id` / `id` de KV commiteados	Contradicen la nota de privacidad (no son credenciales)	Decidir en F0: mantener o externalizar antes del primer push
Dependencia de la API de Discogs	Cortes o cobertura insuficiente del enriquecido	Token + rate limit + User-Agent ya implementados; MusicBrainz como respaldo; vigilar términos de uso
Rate limit de MusicBrainz (~1 req/s)	Respaldo lento	Respetar el límite; cachear resultados
Calidad del modelo de visión en portadas gráficas	Catalogación incompleta, frustración	Parser tolerante + matching 0.34 + respaldo por código de barras y alta manual
KV se llena con portadas (~24 MB/valor)	Fallos al guardar portada	Medir (F4); migrar a R2 solo si hace falta
Fijar cuota freemium sin medir costo de IA	Modelo que pierde o ahuyenta	**Medir antes de cobrar**: la cuota es hipótesis hasta tener el costo real por escaneo

---

Cronograma relativo

> Secuencia relativa, no fechas. La **Fase 0 es de horas**, no de días, y desbloquea todo. Las fases 1–3 pueden intercalarse en bloques cortos (la 1 es la de mayor valor funcional; la 3 es trivial). La 4 es reactiva (solo si KV se llena). La 5 requiere primero una fase de medición del costo real de Workers AI por escaneo —ese dato hoy no existe y no debe inventarse—. Para un cronograma con estimaciones por fase, indica las horas-semana de las que dispones y se calcula.

Próximo paso recomendado

Hacer la Fase 0 ahora mismo. Abre una terminal en `Discasus\`, `git init`, crea un `.gitignore` (excluyendo `node_modules/`, `dist/`, `.wrangler/` y secrets), decide qué hacer con los IDs de `wrangler.jsonc`, commitea todo, crea un repo privado en GitHub y empuja. Es el paso de mayor retorno y menor esfuerzo de todo el plan: convierte un proyecto que **solo existe en un disco, sin siquiera repositorio**, en uno respaldado. Todo lo demás puede esperar; esto no.