Stack & Decisiones

Versión: 2025-10-26 · **Estado:** Aprobado · **Ámbito:** funcional (sin QA automatizada ni hardening avanzado)

1. Objetivo y Alcance

a. Definir el stack definitivo y las decisiones de diseño que previenen sorpresas en desarrollo, despliegue y operación. Esta versión es norma del proyecto.

2. Stack

a. Plataforma / Infra

i. Vercel — Functions Node (no Edge para webhooks)
   1. **Para qué:** Next.js con SSR/ISR y APIs; verificación de firmas requiere body crudo.
   2. **Por qué:** DX alta, latencia baja, encaje natural con Next y control total del request body

ii. Node.js LTS 22.x
   1. **Para qué:** engines.node: "22.x" y runtime del proyecto.
   2. **Por qué:** rendimiento, compatibilidad y ciclo de vida vigente.

iii. Previews de PR en Vercel
   1. **Para qué:** una URL de verificación por Pull Request.
   2. **Por qué:** facilita QA manual y demos sin despliegues a producción.

iv. Git + GitHub + Projects
   1. **Para qué:** repo, PRs, CODEOWNERS, tablero Kanban.
   2. **Por qué:** ritmo del equipo y trazabilidad.

v. ngrok (dev)
   1. **Para qué:** exponer API Routes locales para probar webhooks (Wompi, Postmark).
   2. **Por qué:** acelera integración sin desplegar.

b. Backend / Datos (Gestionado)

i. Supabase (Postgres + RLS + Auth + Storage + Scheduled/pg_cron + Backups)
   1. **Para qué:** Persistencia estándar con seguridad por fila; autenticación; archivos; jobs cerca de los datos.
   2. **Por qué:** minimiza horas de infraestructura sin cerrar salida (Postgres estándar).

ii. Supabase CLI (migraciones + tipos TypeScript)
   1. **Para qué:** administrar migraciones SQL y generar tipos TS del esquema (`supabase gen types typescript`).
   2. **Por qué:** tipado fuerte end-to-end y despliegues consistentes.

iii. Extensiones Postgres (baseline)
   1. **Para qué:** capacidades nativas necesarias sin librerías extra.
   2. **Por qué:** estandariza funcionalidades en todo el proyecto.

3. **Lista mínima:**
    a. `pgcrypto: gen_random_uuid()` para IDs.
    b. `citext` para emails/strings case-insensitive.
    c. `pg_cron` para jobs programados.

c. Frontend

i. Next.js 15 (App Router) + React 19 + TypeScript
    1. **Para qué:** SSR/ISR, Server Actions y DX moderna con tipado estático.
    2. **Por qué:** camino soportado y estable; el tipado reduce errores y acelera el desarrollo.

ii. Material UI (MUI) + Emotion
    1. **Para qué:** Set de componentes accesibles (AppBar, Drawer, Dialog, Menu, Table, Tabs, Snackbar) y layout (Box, Stack, Grid) con theming unificado (light/dark, palette, typography).
    2. **Por qué:** Simplicidad, menor complejidad y velocidad

iii. Material Icons (iconos)
    1. **Para qué:** Set oficial de íconos para UI (reemplaza lucide-react).
    2. **Por qué:** Consistencia visual con MUI y menor fricción.

iv. TanStack Query
    1. **Para qué:** fetching, caché y reintentos; estados de red.
    2. **Por qué:** menos plumbing y menos bugs de sincronización.

v. React Hook Form + Zod
    1. **Para qué:** formularios tipados y robustos.
    2. **Por qué:** validaciones consistentes en FE/BE.

vi. Luxon
    1. **Para qué:** fechas y zonas horarias.
    2. **Por qué:** persistimos en UTC y mostramos America/Bogota sin sorpresas.

vii. MUI X Date Pickers + Adapter Luxon
    1. **Para qué:** DatePicker/DateTimePicker/TimePicker para formularios y filtros.
    2. **Por qué:** Encaje nativo con MUI y con Luxon

viii. Big Calendar (RBC)
    1. **Para qué:** calendario liviano con vistas día/semana/mes y arrastre básico.
    2. **Por qué:** cubre planificación sin complejidad extra.
    3. **Alcance:** sin timeline multi-recurso; adapter `listEvents/onCreate/onMove/onResize`.

ix. Uppy
    1. **Para qué:** subidas con progreso y reintentos.
    2. **Por qué:** mejor UX con mínimo código.

x. Figma Make (prototipado funcional con IA)
    1. **Para qué:** Generar prototipos interactivos a partir de maquetas Figma y editar el código del prototipo en vivo para explorar flujos (calendario, inscripción, aulas) antes de construirlos en Next.js.

2. **Por qué:** Reduce el tiempo de ida y vuelta entre diseño y desarrollo; permite validar interacción y estados rápidamente con la estructura del diseño original preservada.

d. Integraciones de negocio

i. Wompi (Hosted Checkout)
   1. **Para qué:** cobros con PCI/3DS resuelto y webhook de confirmación.
   2. **Por qué:** menor tiempo/riesgo en pagos para Colombia.

ii. Postmark + React Email
   1. **Para qué:** emails de verificación, reset y confirmaciones con plantillas en React.
   2. **Por qué:** entregabilidad alta y mantenimiento simple.

e. Transversales (Frameworks en Plataforma)

i. Identity & RBAC/RLS
   1. **Para qué:** autenticación Supabase y políticas default deny por tabla; scopes por rol.
   2. **Por qué:** acceso mínimo necesario sin backend proxy complejo.

ii. Documents & Storage
   1. **Para qué:** buckets público/privado, URLs firmadas y limpieza de huérfanos.
   2. **Por qué:** control de acceso simple y efectivo.

f. Backoffice interno

i. Appsmith (sólo staff)
   1. **Para qué:** panel interno (Usuarios/Roles, Feature Flags, Outbox failed + retry, Jobs re-run, consultas Audit).
   2. **Por qué:** acelerar operación sin construir UI admin a mano.
   3. **Checklist de vistas mínimas:**
      a. Usuarios/Roles
      b. Feature Flags
      c. Outbox Failed (retry)
      d. Jobs (re-run)
      e. Auditoría (filtros)

g. Utilidades de calidad de vida y CI

i. ESLint + Prettier + lint-staged (con simple-git-hooks o Husky)
   1. **Para qué:** formato y lint automáticos antes de cada commit.
   2. **Por qué:** PRs limpios y menos roturas por estilo o tipos.

ii. Validación de `process.env` con Zod
   1. **Para qué:** fallar temprano si faltan claves (WOMPI, POSTMARK, SUPABASE, etc.).
   2. **Por qué:** evita fallos en runtime y despliegues inválidos.

iii. pnpm (vía Corepack)
   1. **Para qué:** gestor de paquetes rápido con lockfile eficiente.
   2. **Por qué:** instalaciones veloces y CI predecible.

iv. CI mínimo con GitHub Actions + Dependabot

1. **Para qué:** correr typecheck/lint/build en PRs y mantener dependencias críticas.
2. **Por qué:** integridad básica sin añadir complejidad.
3. **Ajustes:** Dependabot sólo seguridad, mensual; workflows simples (un job por PR).

v. Trae (editor de código con IA)

1. **Para qué:** Editor/IDE con asistencia de IA (chat/contexto del repo, generación/edición/explicación de código) para acelerar tareas repetitivas y refactors controlados, manteniendo nuestro estándar (ESLint/Prettier/TypeScript).
2. **Por qué:** Integra agentes/modelos modernos y flujo tipo "copilot" con DX rápida, sin atarnos a un vendor del runtime. Mejora la productividad en FE/BE y reduce tiempo de exploración.

3. Decisiones de diseño que previenen "gotchas" (normativas)

a. Fechas, horas y zonas horarias

i. Norma

1. Persistir y procesar en UTC en backend y BD.
2. Renderizar en UI en America/Bogota (salvo preferencia del usuario).
3. Instantes → `timestamptz`. Fechas puras (sin hora) → `date`.
4. Intercambio en ISO-8601 (`YYYY-MM-DDTHH:mm:ss.sssZ`).
5. Rangos de tiempo con semántica [inicio, fin) (fin exclusivo).

ii. Motivo

1. Evita drift y solapes.

iii. Aplicación mínima

1. Columnas `created_at/updated_at timestamptz DEFAULT now()` (UTC).
2. Conversión al renderizar, no al persistir.
3. Para filtros de un día local: calcular rango UTC de [YYYY-MM-DDT00:00-05:00, YYYY-MM-DDT24:00-05:00) antes de consultar.

iv. DoD

1. Un evento creado a las 23:30 Bogotá se re-muestra con esa misma hora local.

b. Dinero y montos

i. Norma

1. Representar montos en unidades mínimas (enteros).
2. Prohibido float/double para dinero.
3. Cálculos en enteros; redondeo final por regla contable (half-up).
4. Formateo local (COP) solo en UI.

ii. Motivo

1. Evita errores de redondeo y discrepancias con la pasarela.

iii. Aplicación mínima

1. Campo `amount_in_minor int8 NOT NULL`.
2. Utilitarios `minorToMoney/moneyToMinor`.

iv. DoD

1. El total cobrado coincide exactamente con el mostrado (ni +1 ni −1 unidad mínima).

c. Webhooks e idempotencia (cuerpo crudo)

i. Norma

1. Verificar firma usando raw body.
2. Idempotencia estricta:
    a. Webhooks: `event_id` único (`UNIQUE`).
    b. Endpoints cliente críticos (p. ej., crear orden): header `Idempotency-Key` (`UNIQUE`).
3. En duplicados: responder 2xx con el mismo resultado previo, sin efectos nuevos.

ii. Motivo

1. Previene reprocesos y efectos duplicados.

iii. Aplicación mínima

1. Índices únicos por `event_id/idempotency_key`.
2. Helper `withIdempotency(key, handler)` reutilizable.

iv. DoD

1. Tres POST seguidos con la misma `Idempotency-Key` devuelven el mismo payload y no crean registros extra.

d. Control de acceso: Auth, RBAC y RLS

i. Norma

1. RLS “default deny” en tablas de negocio.
2. Autorización por roles/scopes en servidor.
3. No exponer llaves con privilegios al cliente.

ii. Motivo

1. Mínimo privilegio y separación clara cliente/servidor.

iii. Aplicación mínima

1. Políticas RLS por entidad (read own/read role/write role).
2. Guardas `can(scope)` en backend y protección de rutas en UI.

iv. DoD

1. Un usuario sin permisos no accede a datos aun con llamadas directas al API.

e. Feature Flags (toggles) — conjunto reducido y operativo

i. Norma

1. Flags booleanos con `key` en `kebab-case`.
2. Cambios auditados (quién/cuándo).
3. Lectura con helper `isEnabled(key)` (cache corto).

4. Sin persistir flags por usuario.

ii. Conjunto permitido (reducido)
1. `checkout-enabled` (ON por defecto): al estar OFF, UI desactiva compra/inscripción y el servidor rechaza nuevas órdenes con 503 “Inscripciones cerradas temporalmente”.
2. `email-sending-enabled` (ON por defecto): al estar OFF, `sendEmail()` no envía y devuelve “aceptado sin envío”; se registra advertencia para seguimiento.
3. `admin-readonly-mode` (OFF por defecto): al estar ON, bloquea mutaciones en rutas de administración (solo lectura).

iii. Motivo
1. Permite apagar/encender piezas sensibles sin redeploy, con trazabilidad.

iv. Aplicación mínima
1. Tabla `feature_flag(key text primary key, enabled bool, updated_at timestamptz, updated_by uuid)` + RLS.
2. Pantalla en Appsmith para alternar y auditar.

v. DoD
1. Cambiar un flag refleja el comportamiento en UI y servidor en la siguiente petición sin despliegue nuevo.

f. Contrato de errores API

i. Norma
1. Respuesta de error:

```
{ "code": "STRING_SNAKE", "message": "humano", "details": {},
"correlation_id": "uuid" }
```

2. Códigos canónicos mínimos:
   a. `UNAUTHORIZED`
   b. `FORBIDDEN, NOT_FOUND`
   c. `VALIDATION_FAILED`
   d. `IDEMPOTENCY_CONFLICT`
   e. `DUPLICATE_EVENT`
   f. `RATE_LIMITED`
   g. `WEBHOOK_INVALID_SIGNATURE`
   h. `INTERNAL_ERROR`

ii. Motivo
1. Depuración consistente y UX clara.

iii. Aplicación mínima
1. Middleware que asigne `correlation_id` y capture errores.

iv. DoD
1. Todos los errores de API incluyen `code` reconocido y `correlation_id`.

g. Emails (identidad y entrega)

i. Norma
   1. Email de usuario lowercase y único (índice sobre `lower(email)` o `citext`).
   2. Flujos de verificación y reset de contraseña activos.
   3. Configurar SPF/DKIM/DMARC antes de salida a producción.
   4. No incluir PII sensible en enlaces.

ii. Motivo
   1. Evita duplicados y mejora entregabilidad y seguridad.

iii. Aplicación mínima
   1. Validación y normalización de email en registro/cambio.
   2. Checklist DNS en pre-lanzamiento.

iv. DoD
   1. No se crean cuentas duplicadas por mayúsculas/minúsculas y la verificación funciona end-to-end.

h. Subidas de archivos

i. Norma
   1. Upload mediante signed URL (PUT directo) con validación previa de MIME/tamaño.
   2. Buckets diferenciados: público (solo lectura) y privado (acceso firmado).
   3. Sanitizar nombre de archivo y rechazar extensiones no permitidas.

ii. Motivo
   1. Control de acceso y simplicidad operativa.

iii. Aplicación mínima
   1. Endpoint `POST /storage/signed-upload` que valida y emite URL temporal.
   2. UI con barra de progreso y manejo de error básico.

iv. DoD
   1. Un archivo prohibido nunca recibe URL firmada; los permitidos suben y se sirven según permisos.

i. Scheduler (cron) y tareas diferidas

i. Norma
   1. Jobs idempotentes y re-ejecutables con seguridad.
   2. Registrar ejecuciones en `job_run` (estado, intentos, timestamps).
   3. Reintentos con backoff; fallos visibles y recuperables desde backoffice.
   4. Las funciones programadas no realizan llamadas a servicios externos desde SQL; cualquier integración externa se delega a endpoints de aplicación.

ii. Motivo
   1. Previene loops y estados inconsistentes.

iii. Aplicación mínima
   1. Tablas `job/job_run`.

2. Botón “re-intentar” en Appsmith.

iv. DoD

1. Re-ejecutar un job no crea efectos duplicados.

j. Outbox (eventos de dominio)

i. Norma

1. Despacho con FOR UPDATE SKIP LOCKED en lotes pequeños y advisory lock global.
2. Reintentos con backoff y dead-letter tras N intentos.
3. Orden estable por created_at.
4. Consumidores idempotentes (UPSERT por clave natural/event_id).

ii. Motivo

1. Entrega confiable y consistente sin dependencias externas.

iii. Aplicación mínima

1. Tabla outbox_event y vista “failed + retry” en Appsmith.

iv. DoD

1. Procesar el mismo evento dos veces no duplica efectos.

k. Rate limiting mínimo

i. Norma

1. Limitar signup y reset password por clave compuesta (p. ej., IP+email) con umbrales sugeridos de 5/min y 20/día.
2. En caso de límite, responder 429 con `Retry-After` si aplica.

ii. Motivo

1. Evita abuso de endpoints sensibles.

iii. Aplicación mínima

1. Tabla/contador atómico `rate_limit(key, window, count, updated_at)`.
2. Middleware previo al handler.

iv. DoD

1. Exceder el umbral devuelve 429 de forma consistente.

l. Frontend: SSR vs. solo navegador

i. Norma

1. Librerías que dependen del DOM se cargan con `dynamic(..., { ssr: false })`.
2. Acceso a `window/document` únicamente en componentes client o efectos.
3. Solo variables `NEXT_PUBLIC_*` pueden usarse en el cliente.
4. El calendario RBC debe cargarse con dynamic(import(...), { ssr: false }).

ii. Motivo

1. Evita errores de SSR y fugas de configuración.

iii. Aplicación mínima

1. `"use client"` donde corresponda.
2. Helper `env` diferenciado cliente/servidor.

iv. DoD

1. La app renderiza sin errores de SSR y sin exponer secretos.

m. Migraciones y seeds

i. Norma
   1. Todo cambio de esquema en migraciones versionadas.
   2. Prohibido modificar esquema manualmente en entornos compartidos.
   3. Seeds mínimas reproducibles.

ii. Motivo
   1. Evita drift entre entornos.

iii. Aplicación mínima
   1. Pipeline local reproducible para levantar BD desde cero.

iv. DoD
   1. Clonar el repo y ejecutar el script de setup deja el sistema operativo con datos base.

n. Borrado y consistencia referencial

i. Norma
   1. Soft delete con `deleted_at` para entidades críticas.
   2. `UNIQUE` parciales (`WHERE deleted_at IS NULL`) donde aplique.
   3. Cascadas solo cuando sean seguras; si no, validar bloqueos en app.

ii. Motivo
   1. Evita pérdidas irreversibles y mantiene unicidades.

iii. Aplicación mínima
   1. Triggers o validaciones que impidan duplicar al restaurar.

iv. DoD
   1. Restaurar un registro no rompe unicidades ni relaciones.

o. Paginación y ordenación

i. Norma
   1. Preferir keyset pagination en listados grandes; `OFFSET` para listados pequeños.
   2. Orden estable con columna secundaria (`id`) para empates.

ii. Motivo
   1. Evita “saltos” con inserciones concurrentes.

iii. Aplicación mínima
   1. Endpoints aceptan `cursor` y devuelven `nextCursor`.

iv. DoD
   1. Insertar elementos nuevos no altera la secuencia ya mostrada.

p. Secrets y configuración

i. Norma
   1. Variables de entorno del servidor no se exponen al cliente (sin `NEXT_PUBLIC_`).
   2. Prohibido loguear secretos.
   3. Validación de configuración al arranque; si falta, el proceso falla con error claro.

ii. Motivo
    1. Minimiza superficie de riesgo y errores de despliegue.
iii. Aplicación mínima
    1. Módulo `/config/env.ts` que valida y exporta `env`.
iv. DoD
    1. Falta un secreto → el servicio se niega a iniciar con mensaje explícito.

q. Plan de compatibilidad de runtime

i. Norma
    1. Si una librería crítica de UI bloquea el avance por incompatibilidad, se aplica fallback de versión de React/Next documentado.
    2. Activación cuando el bloqueo supera 1 día y es confirmado.
    3. Reversión al objetivo cuando esté disponible el fix del ecosistema.
ii. Motivo
    1. Evita frenos por dependencias externas.
iii. Aplicación mínima
    1. Mantener tabla “lib → versión probada” en el README técnico.
iv. DoD
    1. Se puede alternar de forma controlada sin tocar el dominio.

r. Backoffice operativo

i. Norma
    1. El backoffice (flags, usuarios/roles, outbox fallidos, jobs) se gestiona en Appsmith.
    2. No se duplica funcionalidad administrativa en el frontend público.
ii. Motivo
    1. Reduce superficie y acelera operación interna.
iii. Aplicación mínima
    1. Páginas: Flags (toggle + auditoría), Usuarios/Roles, Outbox Failed (retry), Jobs (re-run).
iv. DoD
    1. Operaciones internas comunes se resuelven sin cambios de código.

s. Seguridad básica de endpoints

i. Norma
    1. Rutas internas (cron/ops) protegidas con secreto en header.
    2. CORS restrictivo: solo orígenes autorizados.
    3. Métodos HTTP: sin efectos en `GET`; usar `POST/PUT/DELETE` para mutaciones.
ii. Motivo
    1. Evita exposición accidental y abuso trivial.
iii. Aplicación mínima

1. Middleware de autenticación interna; lista de orígenes permitidos.

iv. DoD

1. Una llamada sin secreto válido responde 401/403.

t. Naming y consistencia

i. Norma

1. IDs como UUID v4; claves de negocio con `text` + `UNIQUE`.
2. `snake_case` en BD; `camelCase` en JSON/TS.
3. Campos estándar: `id`, `created_at`, `updated_at` y opcional `deleted_at`.

ii. Motivo

1. Homogeneidad entre capas y legibilidad.

iii. Aplicación mínima

1. Mapeo consistente entre BD y DTOs.

iv. DoD

1. Sin nombres mixtos ni inconsistentes a través de la API.

u. Auditoría de cambios (append-only)

i. Norma

1. Registrar acciones sensibles en `audit_log` (actor, acción, entidad, diff, timestamp, ip).
2. Triggers en tablas críticas para altas/bajas/cambios relevantes.
3. Filtros por actor/entidad/fecha disponibles para consulta interna.

ii. Motivo

1. Trazabilidad y responsabilidad operativa.

iii. Aplicación mínima

1. Tabla `audit_log` + triggers; vista de consulta en Appsmith.

iv. DoD

1. Cambiar un flag o un rol deja huella consultable (quién, cuándo, qué).

v. Approval Engine (acciones sensibles)

i. Norma

1. Operaciones sensibles (p. ej. cambios de rol, reembolsos) requieren solicitud de aprobación previa.
2. Estados: `pending`, `approved`, `rejected`, `cancelled`.
3. Registro del historial de decisiones por solicitud.

ii. Motivo

1. Control de riesgo y gobierno operativo.

iii. Aplicación mínima

1. Tablas `approval_request/step/action` y endpoints de approve/reject.
2. Bandeja en Appsmith para pendientes y trazas.

iv. DoD

1. Sin aprobación, la acción sensible no se ejecuta; al aprobar, queda auditado.