Becalia

SCHOLARSHIP INTELLIGENCE PLATFORM

Documentación Técnica de Plataforma

Arquitectura, motor de agente IA con orquestación MCP, modelo de datos, economía de tokens y diseño de seguridad de un SaaS de descubrimiento de becas multilingüe.

Agente IA · 3 MCP · Pipeline determinista · Coste predecible

DOCUMENTO	VERSIÓN / ESTADO	FECHA
Especificación de Arquitectura	v1.0 — MVP	Junio 2026
CLASIFICACIÓN	**FEATURE**	**AUDIENCIA**
Confidencial	001-becas-mvp	Ingeniería

Control del documento

Este documento describe la arquitectura, el diseño y los principios de ingeniería de la plataforma Becalia en su versión MVP. Es la fuente de verdad técnica derivada de la especificación dirigida por especificaciones (Spec-Kit) y del código fuente de producción.

Atributo	Valor
Nombre del producto	**Becalia** — Buscador de becas con agente IA
Tipo de sistema	Aplicación web SaaS multi-inquilino (B2C, con vertical B2B en exploración)
Dominio de producción	`becaliaco.com`
Repositorio	Monorepo (npm workspaces): `frontend/` + `backend/`
Metodología	Spec-Driven Development (Spec-Kit): `spec → plan → data-model → contracts → tasks`
Mercado objetivo	Estudiantes hispanohablantes; pagos en COP (Colombia)
Estado actual	MVP funcional; hardening de seguridad y migración a dominio de producción completados

CONVENCIONES

`FR-xxx` = Requisito funcional. `SC-xxx` = Criterio de éxito medible. `Principio I–V` = artículos de la constitución de ingeniería del proyecto. Los identificadores de modelo Anthropic se citan con su precio vigente por millón de tokens.

Índice de contenidos

01 Resumen ejecutivo

> Becalia es una plataforma SaaS que automatiza el descubrimiento de becas. Un estudiante sube su CV y completa su perfil; un **agente de IA ejecutado íntegramente en el backend** orquesta un pipeline determinista por etapas —parseo del CV, búsqueda web en tiempo real mediante tres fuentes MCP, extracción de páginas candidatas y ranking semántico— para devolver becas reales, rankeadas por afinidad y traducidas al idioma del estudiante.

1.1 El problema

La búsqueda de financiación académica está fragmentada en cientos de portales heterogéneos, a menudo en idiomas distintos al del candidato, con requisitos densos y fechas límite dispersas. El coste de oportunidad de la búsqueda manual es alto y sesga el acceso a la financiación hacia quienes disponen de tiempo, dominio idiomático y conocimiento del ecosistema. Becalia comprime ese trabajo de horas a **menos de 90 segundos**.

1.2 La propuesta de valor diferencial

El núcleo no es un buscador con filtros, sino un **agente que razona sobre el perfil completo del candidato** (formación, experiencia, idiomas, palabras clave del CV) y contrasta ese contexto contra resultados web frescos. Tres capacidades lo distinguen:

- **Matching semántico con explicación** — cada beca recibe un porcentaje de afinidad (0–100) calculado por un modelo capaz, no una coincidencia de palabras clave.
- **Ruptura de la barrera idiomática** — el sistema detecta el idioma del CV, busca en varios idiomas y traduce los requisitos al idioma del estudiante (Principio V).
- **Coste de inferencia predecible y auditable** — cada búsqueda mide y persiste su consumo real de tokens y coste por etapa; el modelo de negocio se apoya en esa contabilidad (Principio II).

≤90s	4	3	100%
BÚSQUEDA P90	ETAPAS DEL PIPELINE	FUENTES MCP	BÚSQUEDAS CON COSTE REGISTRADO

1.3 Características del MVP

Subsistema	Alcance
Cuentas y autenticación	Registro/login con email + contraseña, JWT con refresh rotatorio, reset de contraseña de un solo uso
Perfil y CV	Perfil académico estructurado + carga de CV en PDF con extracción de texto y parseo IA

Motor de becas	Agente con pipeline de 4 etapas sobre 3 fuentes de búsqueda/navegación
Control de tokens	Medición de coste por etapa, verificación de cupo previa al gasto, descuento de cupo/créditos
Monetización	Plan Free + Pro (suscripción) + packs de créditos, vía Mercado Pago (COP)
Experiencia premium	Landing 3D scroll-driven, estilo Liquid Glass, i18n ES/EN, accesibilidad AA
Asistente conversacional	Magic Chat servido a través del backend (clave nunca expuesta al cliente)

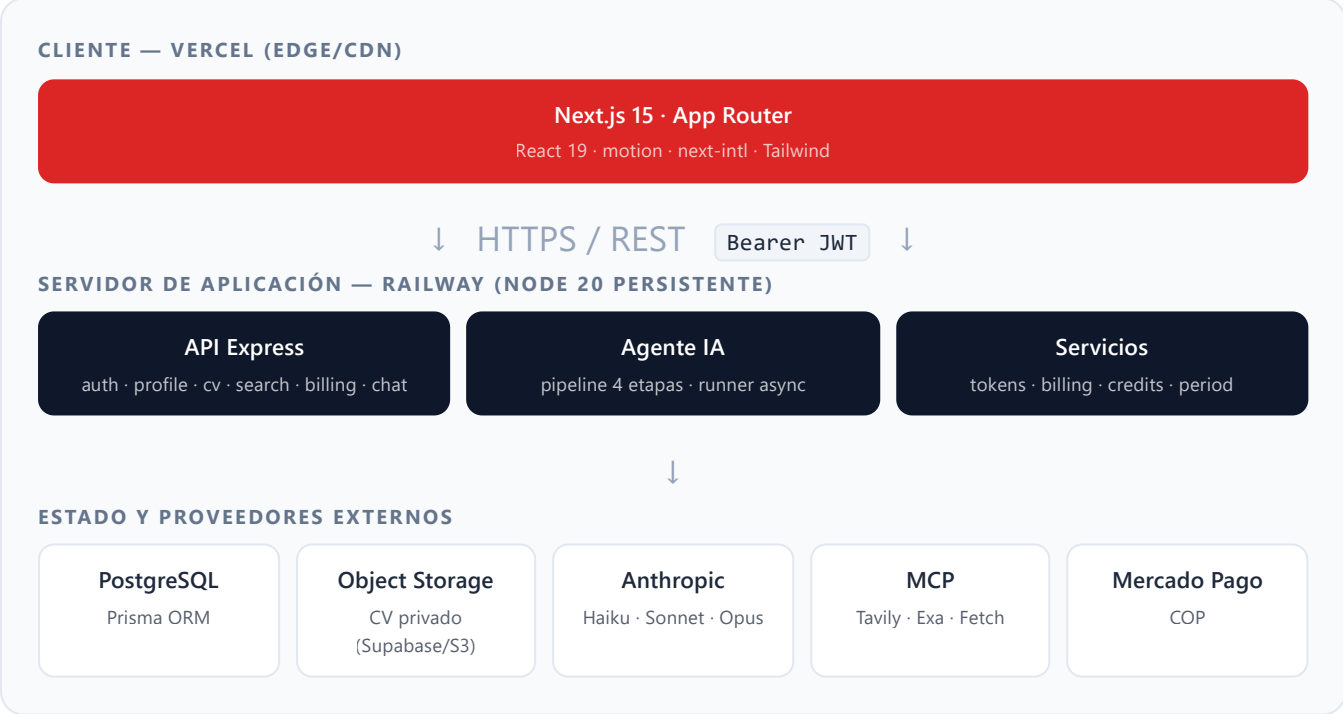
02 Arquitectura del sistema

Becalia adopta una arquitectura de **dos aplicaciones desplegables de forma independiente**, conectadas exclusivamente por una API REST. Esta separación no es estética: responde a una restricción dura del dominio y a dos principios de seguridad.

DECISIÓN DE ARQUITECTURA

El pipeline del agente ejecuta tareas LLM/MCP que pueden durar decenas de segundos. Las plataformas *serverless* de presentación (Vercel) penalizan los procesos largos y de cómputo sostenido. Por ello, todo el cómputo sensible y de larga duración vive en un servicio de backend persistente (Railway). El frontend es **únicamente presentación, i18n y animación**: nunca posee credenciales ni ejecuta lógica de negocio.

2.1 Topología



2.2 Responsabilidades por capa

Capa	Responsabilidad	No hace
Frontend (Next.js)	Renderizado, i18n, animación 3D, captura de formularios, polling de estado de búsqueda	No almacena claves, no llama a proveedores externos, no contiene lógica de cupo ni pagos
API / Routes	Validación con `zod`, autenticación, autorización por ownership, rate-limiting	No ejecuta el pipeline de forma síncrona en el request
Agente / Services	Orquestación del pipeline, medición de tokens, reglas de negocio de facturación	No accede a la red del cliente

Lib	Adaptadores: Prisma, LLM, clientes MCP, JWT, hashing, storage, email, logger	No contiene reglas de negocio

2.3 Ejecución asíncrona de la búsqueda

Una búsqueda no puede mantener un request HTTP abierto ~90 segundos sin fragilidad. El diseño desacopla la **aceptación** de la **ejecución**:

PATRÓN 202 + POLLING

```
POST /search   → valida cupo (middleware quota)
               → crea Search(status="running")
               → enqueueSearch(): setImmediate(executeSearch)   // segundo plano
               → 202 { searchId, status: "running" }            // responde de inmediato

GET  /search/:id  → el frontend hace polling mientras status === "running"
                  → running → complete | partial | failed
```

El `runner` ejecuta el pipeline fuera del ciclo del request, persiste el resultado en una transacción y descuenta cupo/crédito exactamente una vez. La transición a estado final es idempotente: solo finaliza una `Search` que siga en `running`. SSE/streaming queda como mejora futura fuera del MVP.

03 Stack tecnológico

TypeScript 5.x de extremo a extremo. Node.js 20 LTS en backend, React 19 / Next.js 15 en frontend.

FRONTEND — VERCEL

- **Next.js 15** (App Router, RSC)
- **React 19**
- **motion** — scroll-scrubbing y animación (sucesor de framer-motion)
- **next-intl** — i18n ES/EN con routing por `[locale]`
- **Tailwind CSS** + estilo Liquid Glass
- **Playwright** — E2E del flujo crítico

BACKEND — RAILWAY

- **Express 4** sobre Node 20
- **@anthropic-ai/sdk** + **@google/generative-ai** (alternativa)
- **Prisma 6** → PostgreSQL
- **mercadopago** — pagos COP
- **argon2**, **jsonwebtoken**, **helmet**, **express-rate-limit**
- **zod** — validación de entrada y de entorno
- **pdf-parse**, **multer**, **pino** (logs), **@supabase/supabase-js**
- **Vitest** — unit, integración y contrato

3.1 Justificación de selecciones clave

Tecnología	Razón de la elección
argon2	Función de derivación de clave resistente a memoria; preferida sobre bcrypt para hashing de contraseñas modernas
Prisma	Tipado de extremo a extremo del esquema, migraciones versionadas y transacciones explícitas para la contabilidad de créditos
zod	Una sola fuente de validación: entrada de API, salida estructurada del LLM (`zod-to-json-schema`) y variables de entorno
motion	API declarativa de scroll (`useScroll` / `useTransform`) para el scroll-scrubbing del hero 3D con respeto a `prefers-reduced-motion`
pino	Logging estructurado JSON de alto rendimiento, apto para observabilidad en Railway

ABSTRACCIÓN DE PROVEEDOR LLM

El pipeline depende de una interfaz `LlmProvider` con tres implementaciones: `claudeLlm` (Anthropic, `usage` real), un proveedor Gemini equivalente (alternativa de bajo coste, mapeado por el mismo `ModelId` interno) y `mockLlm` (determinista, sin red, `usage = 0`). Esto permite construir y testear el pipeline extremo a extremo sin clave ni gasto, y conmutar de proveedor sin tocar la lógica de orquestación.

04 El motor de agente IA

El corazón de Becalia es un **agente con orquestación determinista por etapas**, no un agente autónomo en bucle libre. Esta decisión es deliberada: un pipeline fijo hace el coste de tokens *predecible* y cada etapa *testeable de forma aislada* (Principio II). El agente está implementado en `backend/src/services/agent/pipeline.ts` como una función pura que no toca la base de datos —la persistencia la realiza el `runner`.

4.1 El pipeline de cuatro etapas

1

PARSEO · CLAUDE-HAIKU-4-5 · TIMEOUT 10S

Extracción estructurada del CV

Recibe el texto del PDF (vía `pdf-parse`) + el perfil. Un modelo de bajo coste devuelve, con salida estructurada validada por `zod`: `{ area, level, languages[], keywords[], cvLanguage }`. La **detección de idioma del CV** ocurre aquí y dirige toda la traducción posterior.

2

BÚSQUEDA · 3 FUENTES EN PARALELO · TIMEOUT 20S/FUENTE · COSTE LLM 0

Descubrimiento de candidatos

Construye consultas bilingües a partir de keywords + perfil y consulta **Tavily** (búsqueda web amplia para IA), **Exa** (búsqueda semántica/neural para becas de nicho) y **DuckDuckGo** (scraper HTML gratuito) en paralelo. Deduplica por URL normalizada. Si una fuente falla o agota su timeout, se continúa con las demás y se registra en `failedSources` → la búsqueda se marca `partial`.

3

EXTRACCIÓN · PLAYWRIGHT/FETCH · TIMEOUT 18S · COSTE LLM 0

Lectura de páginas candidatas

Abre cada URL candidata con concurrencia limitada y extrae `{ title, requirements, deadline?, language, amount? }`, verificando que el enlace responde (`verified`). Si la etapa se cuelga o falla por completo, se degrada a `partial` con el uso medido hasta ese punto, sin colgar ni fallar toda la búsqueda.

4

RANKING · CLAUDE-SONNET-4-6 (ESCALABLE A OPUS-4-8) · TIMEOUT 25S

Ranking semántico y traducción

El modelo capaz recibe las becas crudas + el resumen del CV + el perfil + el idioma del estudiante, y produce `ScholarshipResult[]` con `matchPercent` (0–100), requisitos **traducidos al idioma del estudiante**, ordenados de mayor a menor afinidad. Su `usage` se mide y registra en `stageUsage.rank`.

4.2 Diagrama de flujo del contrato interno

```
POST /search (tras quota OK)
        │
        ▼
[1] cvParser (Haiku)        → { area, nivel, idiomas, keywords[], cvLanguage }
        │                      mide usage → stageUsage.parse
        ▼
[2] search (Tavily‖Exa‖DDG) → candidateUrls[] (deduplicadas)
        │                      fuente caída → continúa, marca partial · coste 0
        ▼
[3] extract (Fetch/PW)      → rawScholarships[] { title, url, reqs, deadline, lang, amount, verified }
        │                      timeout/fallo total → partial · coste 0
        ▼
[4] rank (Sonnet/Opus)      → ScholarshipResult[] { matchPercent, requirements(traducidos) }
        │                      mide usage → stageUsage.rank
        ▼
persistir Search(status, stageUsage, totalCostUsd) + ScholarshipResult[]
descontar cupo/créditos por coste real (una sola vez, transaccional)
```

4.3 Selección de modelo por etapa (coste/calidad mixto)

La asignación de modelos es la palanca principal de optimización de coste: un modelo barato para tareas mecánicas (parseo, extracción) y uno capaz solo donde el razonamiento añade valor (ranking).

Etapa	Modelo	Entrada $/1M	Salida $/1M	Por qué
1 · Parseo CV	`claude-haiku-4-5`	1.00	5.00	Tarea de extracción estructurada; no requiere razonamiento profundo
4 · Ranking	`claude-sonnet-4-6`	3.00	15.00	Matching semántico, explicación y traducción de calidad
4 · Ranking (escalado)	`claude-opus-4-8`	5.00	25.00	Opción de máxima calidad configurable

Etapas 2 y 3 no consumen LLM: su coste de inferencia es 0. El gasto de una búsqueda se concentra en las etapas 1 y 4.

4.4 Tolerancia a fallos y presupuesto de latencia

Cada etapa está acotada por un timeout dedicado; la suma respeta el objetivo de ~90s p90 (SC-002). La filosofía es de **degradación elegante**: ante el fallo de una fuente o etapa no crítica, el pipeline continúa con resultados parciales en vez de abortar.

Etapa	Timeout	Comportamiento ante fallo
1 · cvParser	10 s	Fallo → la búsqueda se marca `failed` (no hay perfil parseable)
2 · search	20 s / fuente	Fuente caída → continúa; todas caídas → 0 candidatos

3 · extract	18 s	Timeout/fallo → `partial` con uso medido hasta aquí
4 · rank	25 s	Timeout → la búsqueda se marca `failed`

REGLA DE NO FACTURACIÓN DE FALLOS

Una búsqueda sin resultados útiles **no se cobra** (no descuenta cupo). Una búsqueda fallida registra únicamente el coste real medido —siempre menor—, nunca una búsqueda completa. El descuento de cupo/crédito ocurre una sola vez, transaccionalmente, al cerrar la `Search`.

05 Economía de tokens

> **PRINCIPIO II — NO NEGOCIABLE**
>
> El coste de tokens debe ser predecible, medido y auditable. El `tokens.service` es la pieza crítica para el cobro y se trata con máximo cuidado y tests dedicados: es la **fuente de verdad para la facturación**.

5.1 Captura del consumo

Por cada llamada a un modelo se captura el `usage` completo, incluyendo los tokens de caché —que tienen tarifa distinta—. El consumo de cada etapa se acumula con `UsageAccumulator` y se persiste en `Search.stageUsage` (JSON) junto con `Search.totalCostUsd` (`Decimal(10,6)`).

ESTRUCTURA DE USO POR ETAPA

```
StageUsage = {
  model,                                  // claude-haiku-4-5 | sonnet-4-6 | opus-4-8
  inputTokens, outputTokens,
  cacheCreationTokens, cacheReadTokens,   // tarifa de caché distinta
  costUsd                                 // redondeado a 6 decimales
}
```

5.2 Fórmula de coste

La tarifa de caché de Anthropic se modela explícitamente: la escritura de caché cuesta ×1.25 el precio de entrada y la lectura ×0.10. El redondeo es determinista a 6 decimales para evitar *drift* de coma flotante.

```
costUsd = inputTokens        / 1e6 · price.in
        + outputTokens       / 1e6 · price.out
        + cacheCreationTokens / 1e6 · price.in · 1.25
        + cacheReadTokens    / 1e6 · price.in · 0.10

round6(n) = Math.round((n + Number.EPSILON) · 1e6) / 1e6
```

5.3 Verificación previa al gasto

El middleware `quota` se ejecuta **antes** de crear la búsqueda y antes de gastar un solo token (FR-032, SC-005). La regla es pura y testeable:

```
hasQuota(user, creditBalance) =
     user.searchesUsedThisPeriod < quotaForPlan(user.plan)
  OR creditBalance > 0

// si no se cumple → HTTP 402 QUOTA_EXCEEDED + CTA upgrade/comprar
```

El cupo del periodo se resetea de forma perezosa (`period.service`) antes de evaluar, y el saldo de créditos se calcula como `SUM(delta)` sobre el libro mayor. Existe además un techo de seguridad de **200 000 tokens por búsqueda** contra una búsqueda anómala.

5.4 Garantías de la contabilidad

- **Sin doble cobro** — el descuento ocurre una sola vez al cerrar la búsqueda, protegido por un guard de estado idempotente (`searchId`).
- **Sin saldo negativo** — el consumo es atómico dentro de una transacción Prisma, resistente a concurrencia.
- **Orden de descuento** — primero se agota el cupo del plan; superado, se descuenta de los créditos (`delta = -1` , `reason = search_consumption`).
- **100% auditable** — el desglose por etapa queda guardado, soportando tanto el cobro como la transparencia hacia el usuario (SC-004).

06 Modelo de datos

Modelo relacional sobre PostgreSQL gestionado vía Prisma. Toda entidad propiedad del usuario lleva `userId` y se consulta filtrando siempre por el dueño (Principio III — ownership). Tiempos en UTC.

6.1 Diagrama de relaciones

```
User 1──1 Profile
User 1──1 CVDocument
User 1──* Search 1──* ScholarshipResult
User 1──* SavedScholarship *──1 ScholarshipResult   (favoritos)
User 1──1 Subscription
User 1──* CreditLedgerEntry
User 1──* PasswordResetToken
ProcessedWebhook   (idempotencia Mercado Pago — sin FK)
B2bLead            (leads del formulario institucional — sin FK)
```

6.2 Entidades principales

Entidad	Propósito	Campos notables
User	Cuenta del estudiante	`email` (único), `passwordHash` (argon2), `refreshTokenHash`, `plan`, `searchesUsedThisPeriod`, `periodResetAt`, `locale`
Profile	Contexto académico (1:1)	`academicLevel` (enum), `fieldOfStudy`, `country` (ISO), `languages[]`
CVDocument	CV en PDF (1:1)	`storageKey` (bucket privado), `mimeType`, `detectedLanguage`, `parsedSummary` (JSON)
Search	Una ejecución del pipeline	`status` (enum), `detectedLanguage`, `stageUsage` (JSON), `totalCostUsd` (Decimal 10,6)
ScholarshipResult	Beca encontrada	`title`, `sourceUrl`, `matchPercent`, `requirements`, `deadline`, `language`, `amount`, `verified`
SavedScholarship	Favoritos del usuario	único por `(userId, scholarshipId)`
Subscription	Estado de suscripción (1:1)	`mpPreapprovalId`, `status` (enum), `currentPeriodEnd`
CreditLedgerEntry	Libro mayor de créditos	`delta` (±), `reason` (enum), `mpPaymentId` — saldo = `SUM(delta)`
ProcessedWebhook	Idempotencia de pagos	`mpEventId` (único)

PasswordResetToken	Reset de contraseña	`tokenHash` (único), `expiresAt`, `usedAt` (un solo uso)

6.3 Máquinas de estado

SEARCH.STATUS

```
running
  ├─→ complete   (todo OK)
  ├─→ partial    (≥1 fuente/etapa falló,
  │               pero hubo resultados)
  └─→ failed     (cobra solo coste real)
```

SUBSCRIPTION.STATUS

```
none
  └─→ active      ⇒ User.plan = pro
        ├─→ past_due
        └─→ canceled
  (transiciones por webhook MP,
   idempotentes)
```

6.4 Diseño de tokens de autenticación

- **Access token (JWT)** — corto (≈15 min), *stateless*, firmado con `JWT_SECRET`.
- **Refresh token** — rotatorio; se almacena **hasheado** en `User.refreshTokenHash`. Al renovar se rota (invalida el anterior); el logout limpia el hash.
- **Reset token** — el enlace lleva el token en claro (viaja solo por email); en DB se guarda únicamente el hash. Inválido si `usedAt != null` o expirado; al usarse, invalida los anteriores del mismo usuario.

07 Contratos de API

API REST en `/api/v1`, JSON. Autenticación por `Authorization: Bearer <JWT>` salvo rutas públicas. Errores con forma `{ error: { code, message } }`. Toda entrada se valida con `zod`. RL = rate-limit por usuario + IP.

7.1 Superficie de endpoints

Método · Ruta	Auth	Descripción
`POST /auth/register`	Público	Crea cuenta (plan Free por defecto) → 201 `{ token, user }`
`POST /auth/login`	Público	Credenciales → 200 / 401 sin revelar existencia del email
`POST /auth/reset/request`	Público	Genera token de reset y envía email → 200 siempre
`POST /auth/reset/confirm`	Público	Restablece la contraseña con token de un solo uso
`POST /auth/logout`	Auth	Invalida el refresh → 204
`GET · PUT /profile`	Auth	Lee/actualiza el perfil académico
`POST /cv`	Auth · multipart	Sube PDF (≤10 MB); valida tipo/tamaño → 201 / 415 / 413
`GET /cv`	Auth	Metadata del CV actual (sin binario)
`POST /search`	Auth	Verifica cupo, encola pipeline → 202 `{ searchId }` / 402 / 409
`GET /search/:id`	Auth · ownership	Estado + resultados (objeto de polling)
`GET /search`	Auth	Historial paginado de búsquedas
`GET /billing/summary`	Auth	Plan, consumo, saldo, estado de suscripción
`POST /billing/subscribe`	Auth	Inicia suscripción Pro → `{ initPoint }`
`POST /billing/credits`	Auth	Inicia compra de pack de créditos → `{ initPoint }`
`POST /billing/webhook`	Firma MP	Procesa eventos de pago de forma idempotente
`POST /chat`	Auth	Reenvía a Magic Chat con la clave del backend → `{ reply }`
`POST /favorites` · `/contact`	Auth / Público	Favoritos del usuario y leads B2B

7.2 Endpoint crítico — `POST /search`

```
Precondición: Profile completo + CV. Middleware quota ANTES de ejecutar.
202 → { searchId, status: "running" }            // encola pipeline async
402 → { error: { code: "QUOTA_EXCEEDED" } }       // no gasta tokens
409 → perfil/CV incompletos
```

Los textos de UI los resuelve el frontend con `next-intl`; la API devuelve datos. El campo `requirements` de cada beca ya viene traducido al idioma del estudiante desde la etapa de ranking. CORS está restringido al dominio del frontend.

08 Modelo de negocio y facturación

Modelo *freemium* con suscripción mensual y créditos consumibles, operado en pesos colombianos (COP) mediante Mercado Pago.

8.1 Planes y packs (configurables, sin hardcode disperso)

Producto	Identificador	Cupo / Créditos	Precio (COP)
Plan Free	`free`	3 búsquedas / mes	—
Plan Pro	`pro_monthly`	50 búsquedas / mes	$39.900 / mes
Pack de créditos	`pack_10`	10 búsquedas	$19.900
Pack de créditos	`pack_30`	30 búsquedas	$49.900

Salvaguarda transversal: techo de 200 000 tokens por búsqueda.

8.2 Flujos de pago

- **Suscripción Pro** — vía Mercado Pago *Preapproval*; al confirmarse, `Subscription.status = active` ⇒ `User.plan = pro` y el cupo mensual se amplía.
- **Créditos** — pago único vía Checkout; al confirmarse se asienta un `CreditLedgerEntry` con `delta = +créditos` y `reason = purchase`.

8.3 Idempotencia de webhooks

> **PROCESAMIENTO DE EVENTOS A PRUEBA DE REINTENTOS**
>
> Mercado Pago reintenta los webhooks. Antes de procesar un evento se verifica que su `data.id` no exista en `ProcessedWebhook` (clave única). La firma se valida con `x-signature` + `x-request-id`. Esto garantiza que un reintento nunca produce un doble asiento de créditos ni un cambio de plan duplicado (FR-043).

```
POST /billing/webhook
  1. validar firma (x-signature, x-request-id)        → 400 si inválida
  2. ¿mpEventId ya en ProcessedWebhook? → sí: 200 no-op (idempotente)
  3. procesar: payment → CreditLedger | subscription_preapproval → Subscription
  4. registrar mpEventId · responder 200
```

Diseño defensivo: si las claves de pago no están configuradas, los endpoints de billing devuelven `503 BILLING_NOT_CONFIGURED` y el SDK de Mercado Pago se instancia de forma perezosa, nunca en el arranque.

09 Seguridad y privacidad

> **PRINCIPIO III — SEGURIDAD Y PRIVACIDAD POR DEFECTO**
>
> Todas las claves de proveedor (LLM, MCP, Magic Chat, Mercado Pago) viven **exclusivamente en el backend**. El frontend nunca las recibe (FR-007, SC-008: cero claves en el bundle, verificable por inspección del build).

9.1 Defensa en profundidad

Control	Implementación
Contraseñas	Hashing con **argon2**; nunca en texto plano ni expuestas
Sesiones	JWT corto firmado + refresh token rotatorio almacenado hasheado
Autorización	**Ownership por** `userId` en cada consulta a datos del usuario
Validación de entrada	`zod` en el 100% de los cuerpos, parámetros y subida de archivos
Rate-limiting	`express-rate-limit` por usuario + IP en rutas sensibles; `trust proxy` para IP real tras Railway/Vercel
Cabeceras HTTP	**helmet**; `x-powered-by` deshabilitado
CORS	Restringido al origen del frontend (`FRONTEND_ORIGIN`)
Límite de payload	`express.json({ limit: '1mb' })` ; CV ≤ 10 MB con validación de tipo
Validación de entorno	`zod` sobre `process.env` al boot; en producción se exige la presencia de claves núcleo
Saneamiento de datos	Limpieza de bytes de control/NUL del texto scrapeado antes de persistir (Postgres 22021)

9.2 Tratamiento de datos personales (PII)

- El CV es dato personal: se almacena en un **bucket privado**, referenciado por `storageKey`, asociado únicamente a su dueño.
- El binario del CV nunca se devuelve por API; solo metadata.
- El borrado en cascada (`onDelete: Cascade`) garantiza que eliminar un usuario elimina su perfil, CV, búsquedas y créditos.
- El detalle legal/regulatorio completo (retención, RGPD) se aborda antes de producción a gran escala; existen páginas de privacidad y términos en el frontend.

9.3 Validación de fuentes y resultados

- Solo se persisten resultados con `sourceUrl` absoluta `http(s)` válida; un resultado sin URL utilizable se descarta.
- Las becas con `deadline` vencido se filtran antes de persistir.
- El enlace se marca `verified` únicamente si responde.

10 Frontend y experiencia

> **PRINCIPIO IV — EXPERIENCIA PREMIUM ACCESIBLE**
>
> Inmersión visual de alta gama que se degrada con gracia: ninguna animación bloquea la funcionalidad ni rompe la accesibilidad.

10.1 Hero 3D scroll-driven

La landing presenta una Tierra en vídeo controlada por el scroll (*scroll-scrubbing*): el avance del vídeo se sincroniza con el progreso del scroll mediante `useScroll` / `useTransform` de **motion**. El activo original (`earthp.mov`, ~151 MB) se transcodifica con ffmpeg a MP4 (H.264) + WebM a ~1080p con keyframe por frame, con objetivo de **5–10 MB**.

- **Fallback obligatorio** — en móvil o con `prefers-reduced-motion` se sirve un póster estático; el vídeo pesado no bloquea la interacción (SC-006: 100% de degradación).
- **60 fps** objetivo en desktop de gama media, animando solo `transform` / `opacity` sin layout shift.

10.2 Sistema de diseño — Liquid Glass

Token	Valor	Uso
Navy profundo	`#0F172A` / `#1E293B`	Superficies base, fondos
Rojo acción	`#DC2626`	Acento, CTAs
Fondo claro	`#F8FAFC`	Lienzo de contenido
Tipografía	Inter	Toda la interfaz
Movimiento	400–600 ms	Transiciones fluidas con respeto a reduced-motion

10.3 Internacionalización y accesibilidad

- **i18n** — `next-intl`, routing por `[locale]`, ES/EN, detección automática del idioma del navegador y selector manual (FR-053).
- **Ruptura idiomática end-to-end** — el idioma del CV se detecta en el backend (etapa 1) y los requisitos de cada beca llegan ya traducidos al idioma del estudiante.
- **Accesibilidad AA** — contraste 4.5:1, navegación por teclado, focus visible (FR-054).

10.4 Mapa de pantallas

Landing (hero 3D) · Login / Registro / Reset · Onboarding (perfil + CV) · Dashboard de becas · Detalle de beca · Cuenta (plan / créditos / consumo) · Pricing · B2B · Privacidad · Términos. Asistente flotante (Magic Chat) disponible en la app.

11 Calidad y tolerancia a fallos

11.1 Estrategia de testing

El backend mantiene una suite Vitest centrada en las piezas de mayor riesgo de negocio. La medición de tokens y la facturación reciben atención dedicada.

Suite	Cubre
`tokens.test.ts`	Cálculo de coste, redondeo determinista, tarifa de caché
`quota.test.ts`	Lógica pura de cupo (plan + créditos) antes del gasto
`consumption.test.ts`	Reglas de facturación: qué búsqueda es *billable*
`pipeline.test.ts`	Orquestación de 4 etapas con provider mock (sin red ni coste)
`runner.test.ts`	Persistencia idempotente, filtros de resultados, transición de estado
`billing.test.ts` · `webhook.test.ts`	Flujos de pago e idempotencia de webhooks
`period.test.ts` · `auth.test.ts`	Reset perezoso de periodo y autenticación
Playwright (frontend)	E2E del flujo crítico + accesibilidad de la landing

11.2 Principios de robustez aplicados

- **Pipeline puro** — la orquestación no toca la DB; la persistencia y el cobro viven en el runner, aislando la lógica testeable del efecto secundario.
- **Inyección de dependencias** — `PipelineDeps` y `RunnerDeps` permiten sustituir LLM, fuentes, storage y timeouts en tests.
- **Timeouts en cada borde** — descarga del CV, parseo del PDF y cada etapa LLM/MCP están acotados; nada deja una búsqueda colgada en `running` para siempre.
- **Degradación elegante** — fallo parcial → `partial`; fallo total → `failed` con coste real, nunca cobro de búsqueda completa.

11.3 Criterios de éxito medibles

ID	Criterio
SC-001	Registro → perfil + CV → primera lista de becas en < 5 minutos
SC-002	Resultados (o vacío útil) en < ~90 s en el percentil 90
SC-003	≥ 80% de becas con enlace que responde y deadline vigente
SC-004	Coste de tokens registrado en el 100% de las búsquedas

SC-005	0 búsquedas ejecutadas sobre cupo agotado
SC-006	Landing fluida ≈60 fps; degradación a póster en el 100% de casos reduced-motion/móvil
SC-008	0 claves de proveedor en el bundle del frontend

12 Infraestructura y despliegue

12.1 Topología de despliegue

Componente	Plataforma	Notas
Frontend	Vercel	CDN/Edge, build de Next.js; `vercel.json`
Backend	Railway	Node 20 persistente; `Dockerfile` + `railway.json`
Base de datos	PostgreSQL gestionado	Railway PG o Supabase; migraciones Prisma versionadas
Almacenamiento de CV	Supabase Storage / S3	Bucket privado; `service_role` solo en backend
Dominio	`becaliaco.com`	Migrado a dominio de producción

12.2 Build y arranque del backend

```
build : prisma generate && tsc -p tsconfig.json
start : node dist/index.js
dev   : tsx watch src/index.ts
```

`prisma generate` se incluye en el build para el despliegue en Railway vía `npm ci` ; Prisma reside en dependencias de producción.

12.3 Observabilidad y salud

- **Logging** — `pino` / `pino-http`, estructurado en JSON; los fallos de pipeline registran `searchId` y stack.
- **Healthcheck** — `GET /health` sin rate-limit para sondas de infraestructura.
- **Webhook** — el endpoint de Mercado Pago se monta antes del rate-limiter para no descartar reintentos legítimos.

12.4 Modos de configuración degradada

El sistema arranca y opera de forma útil aun sin todas las integraciones, lo que facilita desarrollo y demos:

Sin clave de…	Comportamiento
LLM (Anthropic/Gemini)	Pipeline usa `mockLlm` determinista, coste 0
Mercado Pago	Endpoints de billing → `503 BILLING_NOT_CONFIGURED`
Email	El enlace de reset se registra en logs en lugar de enviarse
Magic Chat	El asistente responde en modo demo (mock)

13 Principios de ingeniería

El proyecto se gobierna por una constitución de cinco artículos. Cada decisión de arquitectura se valida contra ellos (Constitution Check del plan: PASA, sin violaciones).

I MVP PRIMERO (YAGNI)

Solo los subsistemas del MVP. Sin panel admin, multi-planes, alertas ni postulación automática. Estructura mínima de dos aplicaciones.

II COSTE PREDECIBLE (NO NEGOCIABLE)

Pipeline por etapas (no bucle libre); modelo barato para parseo, capaz solo para ranking; cupo verificado antes de gastar; `usage` persistido por búsqueda.

III SEGURIDAD POR DEFECTO

Claves solo en backend; argon2; JWT con expiración; `zod` en toda entrada; rate-limit; ownership por `userId`; CV en bucket privado.

IV EXPERIENCIA PREMIUM ACCESIBLE

motion + Liquid Glass; fallback póster en móvil/reduced-motion; contraste AA; teclado + focus; sin layout shift.

V ROMPE LA BARRERA DEL IDIOMA

Detección del idioma del CV en la etapa 1; traducción de requisitos en la etapa de ranking; interfaz `next-intl` ES/EN extensible. La accesibilidad lingüística es una capacidad de producto de primera clase, no un añadido.

14 Roadmap y fuera de alcance

14.1 Explícitamente fuera del MVP

El alcance se acota deliberadamente (Principio I). Las siguientes capacidades se difieren a especificaciones posteriores:

- Panel de administración
- Múltiples niveles de planes
- Alertas automáticas de nuevas becas
- Postulación automática a becas
- Equipos / organizaciones y analítica avanzada

14.2 Evolución técnica prevista

Área	Mejora futura
Transporte de estado	SSE/streaming en lugar de polling para el estado de búsqueda
Extracción	Playwright MCP completo (hoy la base usa `fetch`)
Sesiones	Tabla `Session` para multi-dispositivo
Calidad de ranking	Escalado configurable a Opus 4.8 y *adaptive thinking* por `effort`
Negocio	Vertical B2B para instituciones (lead capture ya presente)

ESTADO DEL PROYECTO

MVP funcional con hardening de seguridad post-auditoría completado y migración a dominio de producción (`becaliaco.com`) con Mercado Pago en modo producción.

A Apéndices

A.1 Variables de entorno del backend

Validadas con `zod` al arranque. En producción, las marcadas como requeridas deben estar presentes o el proceso aborta.

Variable	Propósito	Prod.
`DATABASE_URL`	Cadena de conexión PostgreSQL	Requerida
`JWT_SECRET`	Firma de tokens (≥16 chars en prod)	Requerida
`PUBLIC_BACKEND_URL`	URL pública del backend (notification_url de MP)	Requerida
`FRONTEND_ORIGIN`	Origen permitido para CORS	Requerida
`ANTHROPIC_API_KEY` / `GOOGLE_AI_API_KEY`	Proveedor LLM (al menos uno; si faltan, mock)	Opcional*
`TAVILY_API_KEY` / `EXA_API_KEY`	Fuentes de búsqueda MCP	Requeridas
`PLAYWRIGHT_MCP_URL`	MCP de navegación (mejora; base usa fetch)	Opcional
`MP_ACCESS_TOKEN` / `MP_WEBHOOK_SECRET`	Mercado Pago (pagos + firma webhook)	Opcional
`EMAIL_API_KEY` / `EMAIL_FROM`	Envío de emails de reset	Opcional
`MAGICCHAT_*`	Asistente conversacional	Opcional
`SUPABASE_*` / `CV_BUCKET_*`	Almacenamiento privado de CV	Opcional

* La ausencia degrada la función a su modo mock/no-op, nunca filtra una clave al cliente.

A.2 Tabla de precios de modelos (USD / 1M tokens)

Modelo	Identificador	Entrada	Salida	Caché escritura	Caché lectura
Haiku 4.5	`claude-haiku-4-5`	$1.00	$5.00	×1.25 in	×0.10 in
Sonnet 4.6	`claude-sonnet-4-6`	$3.00	$15.00	×1.25 in	×0.10 in
Opus 4.8	`claude-opus-4-8`	$5.00	$25.00	×1.25 in	×0.10 in

A.3 Glosario

Término	Definición
MCP	Model Context Protocol — protocolo de conexión de herramientas/fuentes a un agente. Becalia usa Tavily, Exa y Playwright/Fetch.

Pipeline determinista	Secuencia fija de etapas (vs. agente autónomo en bucle), que hace el coste predecible y cada etapa testeable.
Scroll-scrubbing	Técnica que vincula el avance de un vídeo al progreso del scroll del usuario.
Liquid Glass	Lenguaje visual de glassmorphism premium (superficies translúcidas, profundidad, desenfoque).
Ownership	Patrón de autorización: toda consulta a datos de usuario filtra por su `userId`.
Idempotencia	Propiedad por la que repetir una operación no cambia el resultado (clave en webhooks y cierre de búsqueda).
Spec-Kit	Metodología spec-driven: `spec → plan → data-model → contracts → tasks`.