ClearPath RD

clearpathrd.com

Infrastructure & Architecture Decision Record

Version 1.0 — February 2026 — Confidential

What this document is

The Infrastructure & Architecture Decision Record (IADR) is the authoritative reference for all infrastructure, hosting, and architectural decisions made before writing code. It explains what was chosen, what was rejected, and why. It is a living document — decisions that are revisited should be updated here, not just changed silently in code.

What this document covers

- System architecture overview — services, their roles, and how they connect
- Hosting platform evaluation and decision
- Repository structure
- Service stack — every third-party service with rationale
- Environment strategy — local, staging, production
- DNS and domain configuration
- Secrets management
- Complete environment variable register
- 12 Architecture Decision Records (ADRs) covering all major technical choices
- Pre-launch infrastructure checklist

1. System Architecture Overview

ClearPath RD is a two-service architecture: a Next.js frontend (server-side rendered, deployed to a hosting platform) and a standalone Fastify API (Node.js, deployed as a separate service). Both services share a single Supabase PostgreSQL database and use Supabase Auth for identity. No monorepo magic — two repositories, two deploy pipelines.

> *Guiding principle: keep services minimal and independently deployable. The frontend knows nothing about database schemas. The API is the single source of truth for all business logic. If the frontend breaks, the API still works. If the API is unreachable, the frontend degrades gracefully.*

1.1 Service Map

Five distinct service boundaries:

Service	Technology	Tier / Cost	Notes
Frontend	`Next.js 14 (App Router)`	Vercel Hobby → Pro	clearpathrd.com — user-facing app, certificate vault, public verify page
API	`Fastify + Prisma`	Render / Railway Starter	api.clearpathrd.com — all business logic, state machine, webhooks
Database	`PostgreSQL (Supabase)`	Supabase Free → Pro	Managed Postgres + Row Level Security + Storage
Auth	`Supabase Auth`	Included in Supabase	JWT issuance, email verification, password reset
Email	`Resend`	Free 3k/mo → paid	Transactional emails via React Email templates
Payments	`Stripe`	Pay-as-you-go	Checkout Sessions, webhooks, tax handling
PDF Storage	`Supabase Storage`	Included in Supabase	Certificate PDFs in private bucket, signed URLs
Scheduling	`node-cron (in API)`	No extra cost	Nightly jobs: expiry, C-NRPP alerts, email queue

1.2 Request Flow

A typical authenticated user request:

- Browser → Next.js frontend (clearpathrd.com)
- Next.js server component or client component → Fastify API (api.clearpathrd.com/api/v1/...)
- Fastify validates Supabase JWT, scopes query to user, executes via Prisma
- Prisma → Supabase PostgreSQL → response back up the chain

Certificate PDF download flow:

- Client requests GET /api/v1/certificates/:id/download from Fastify
- Fastify verifies ownership, retrieves pdf_storage_path from database
- Fastify calls Supabase Storage to generate a signed URL (valid 60 seconds)
- Fastify redirects client to the signed URL (HTTP 302)
- Client downloads PDF directly from Supabase Storage CDN

Stripe webhook flow:

- Stripe → POST /api/v1/webhooks/stripe on Fastify
- Fastify verifies Stripe-Signature header using STRIPE_WEBHOOK_SECRET
- Fastify processes event, updates KitOrder via Prisma, creates TestSession(s)
- Fastify enqueues order_confirm email via Resend SDK

2. Hosting Platform Decision

The frontend (Next.js) and the API (Fastify) are evaluated separately. Each has different requirements.

2.1 Frontend — Vercel

> *Decision: Vercel for the Next.js frontend. Rationale: Vercel built Next.js. The integration is first-class with zero configuration. For an App Router project, Vercel is the lowest-friction path to production.*

Evaluation summary:

Platform	Next.js support	Free tier	Notes
Vercel	★★★★★ Native	Generous (100GB bandwidth)	Built by Vercel. Zero-config App Router. Edge functions. ✔️ Selected.
Netlify	★★★★ Good	100GB bandwidth	Good Next.js support but not first-party. Some App Router quirks.
Render	★★★ Adequate	750 hrs/mo (sleeps)	No native Next.js edge support. Fine for API, not ideal for frontend.
Railway	★★★ Adequate	$5 credit/mo	Docker-based. More suited to API/backend than Next.js frontend.
Fly.io	★★ Manual	3 shared VMs	Requires Dockerfile. No Next.js-specific DX.

2.2 API — Render

> *Decision: Render for the Fastify API. Rationale: Render offers a persistent Node.js service (no cold start after idle) on a predictable pricing model. The free tier is adequate for MVP. Render auto-deploys from GitHub main branch on push, which is sufficient given no formal CI/CD pipeline for MVP.*

Key Render configuration for the Fastify API:

- Service type: Web Service (not Static Site)
- Environment: Node
- Build command: npm install && npx prisma generate
- Start command: node dist/server.js (after TypeScript build)
- Health check path: /health (returns HTTP 200 with {"status":"ok"})
- Auto-deploy: Yes, from main branch on GitHub
- Region: Oregon (US West) — closest to Canadian users on Render free tier. Upgrade to Ohio (US East) on paid for lower latency to Ontario/Quebec.

Why not Railway: Railway is an excellent alternative and should be reconsidered at scale. It has a cleaner developer experience and better observability tooling. For MVP, Render's free persistent service tier is more cost-effective.

3. Repository Structure

Two GitHub repositories. No monorepo. Each service is independently versioned and deployed.

3.1 clearpath-web (Next.js frontend)

Path	Contents
`clearpath-web/`	Root — Next.js 14 App Router project
`app/`	App Router pages and layouts
`app/(marketing)/`	Public marketing pages: home, pricing, about, how-it-works
`app/(auth)/`	Login, register, password reset, email verification
`app/(dashboard)/`	Authenticated app: homes, sessions, results, certificate vault
`app/verify/[id]/`	Public certificate verification page (unauthenticated)
`components/`	Shared React components
`components/ui/`	Primitive UI components (shadcn/ui or custom)
`lib/`	API client, utility functions, Supabase client
`lib/api.ts`	Typed fetch wrapper for Fastify API calls
`lib/supabase.ts`	Supabase browser + server clients
`public/`	Static assets: logo, OG images
`styles/`	Global CSS, Tailwind config
`emails/`	React Email templates (co-located for DX, sent via API)
`.env.local`	Local environment variables (gitignored)
`next.config.ts`	Next.js configuration

3.2 clearpath-api (Fastify API)

Path	Contents
`clearpath-api/`	Root — Fastify + Prisma + TypeScript
`src/`	All TypeScript source
`src/server.ts`	Fastify server entry point, plugin registration
`src/routes/`	Route handlers, one file per domain group
`src/routes/auth.ts`	Auth routes
`src/routes/homes.ts`	Home CRUD
`src/routes/orders.ts`	Kit order + Stripe checkout
`src/routes/sessions.ts`	Test session state machine
`src/routes/results.ts`	Result submission
`src/routes/certificates.ts`	Certificate retrieval + PDF download
`src/routes/contractors.ts`	Directory + lead events
`src/routes/webhooks.ts`	Stripe + Resend webhook handlers
`src/routes/admin.ts`	Admin-only endpoints
`src/plugins/`	Fastify plugins: auth, prisma, cors, rate-limit

`src/plugins/auth.ts`	Supabase JWT verification plugin
`src/services/`	Business logic: certificate generation, email, zone lookup
`src/services/certificate.ts`	PDF generation + Supabase Storage upload
`src/services/email.ts`	Resend integration + email scheduling
`src/services/zone.ts`	FSA → radon zone lookup
`src/services/stripe.ts`	Stripe checkout + webhook processing
`src/jobs/`	node-cron scheduled jobs
`src/jobs/expiry.ts`	Nightly session expiry + certificate expiry
`src/jobs/cnrpp.ts`	C-NRPP expiry alerts
`src/jobs/emailQueue.ts`	Scheduled email dispatch
`prisma/`	Prisma schema + migrations
`prisma/schema.prisma`	The schema (already specified)
`prisma/migrations/`	Migration history
`prisma/seed/`	Seed scripts: radon zone data, test contractors
`.env`	Environment variables (gitignored)

4. Service Stack Register

Every third-party service used in the MVP. All services have free tiers sufficient for launch. Upgrade triggers are noted.

Service	Purpose	Free tier	Upgrade trigger	Key env vars
Supabase	Database + Auth + Storage	500MB DB, 1GB storage, 50k MAU	> 500MB DB or > 500k API reqs/mo	`SUPABASE_URL`, `SUPABASE_ANON_KEY`, `SUPABASE_SERVICE_ROLE_KEY`
Vercel	Next.js frontend hosting	100GB bandwidth, unlimited deploys	> 100GB bandwidth/mo	`NEXT_PUBLIC_API_URL`, `NEXT_PUBLIC_SUPABASE_URL`, `NEXT_PUBLIC_SUPABASE_ANON_KEY`
Render	Fastify API hosting	750 hrs/mo (persistent)	Need > 512MB RAM or < 50ms p95 latency	`DATABASE_URL`, `SUPABASE_SERVICE_ROLE_KEY`
Stripe	Payments + tax	2.9% + 30¢ per transaction (no monthly fee)	N/A — pay-as-you-go	`STRIPE_SECRET_KEY`, `STRIPE_WEBHOOK_SECRET`, `STRIPE_PRICE_ID_*`
Resend	Transactional email	3,000 emails/mo	> 3,000 emails/mo	`RESEND_API_KEY`, `RESEND_WEBHOOK_SECRET`
GitHub	Source control	Free for public + private repos	N/A	`N/A (SSH key in deploy config)`

5. Environment Strategy

Single Supabase project for MVP. Two deployment environments: local development and production. Staging will be added when the team grows beyond one developer or when production incidents require safe testing.

> *MVP rule: Never test against the production database with real user data. Use the Supabase Table Editor or a local Supabase instance (via Supabase CLI) for development. Seed scripts exist for both.*

5.1 Local Development

- Frontend: next dev on localhost:3000
- API: tsx watch src/server.ts on localhost:3001 (or ts-node-dev)
- Database: Option A — point directly at Supabase project with a dedicated dev schema. Option B — run supabase start for a fully local Supabase stack (Docker required). Option B is preferred once the team grows.
- Stripe: Use Stripe CLI (stripe listen --forward-to localhost:3001/api/v1/webhooks/stripe) to forward webhooks to local API
- Resend: Use Resend test mode — emails are accepted but not delivered

5.2 Production

- Frontend: Vercel production deployment, auto-deployed from main branch push to clearpath-web
- API: Render Web Service, auto-deployed from main branch push to clearpath-api
- Database: Supabase production project (single project for MVP)
- All environment variables set in Vercel / Render dashboards — never committed to Git

5.3 Database Migration Strategy

Migrations are managed by Prisma Migrate. The process for deploying a schema change to production:

- 1. Make schema change locally in prisma/schema.prisma
- 2. Run npx prisma migrate dev --name <description> to generate migration file
- 3. Commit migration file to Git alongside the code change that requires it
- 4. On Render deploy, the build step runs: npx prisma migrate deploy (non-interactive, applies pending migrations)
- 5. Never run prisma migrate dev in production — only prisma migrate deploy

6. DNS & Domain Configuration

clearpathrd.com is registered. All DNS changes are made at the registrar (or Cloudflare if DNS is proxied there).

Host	Type	Value	Purpose
clearpathrd.com	CNAME	cname.vercel-dns.com	Root domain → Vercel (Next.js frontend)
www	CNAME	cname.vercel-dns.com	www redirect → Vercel
api	CNAME	<render-service>.onrender.com	api.clearpathrd.com → Render (Fastify API)
verify	CNAME	cname.vercel-dns.com	verify.clearpathrd.com → Vercel (certificate verify page, same app)

SSL/TLS: Both Vercel and Render provision Let’s Encrypt certificates automatically on custom domain attachment. No manual certificate management required.

CORS: The Fastify API allows requests from https://clearpathrd.com and https://www.clearpathrd.com only. The ALLOWED_ORIGINS environment variable controls this. Localhost origins are allowed in development mode.

7. Secrets Management

> *Rule: No secrets in Git. Ever. .env and .env.local are gitignored. Secrets are set directly in the Vercel and Render dashboards. The only exception is NEXT_PUBLIC_* variables which are intentionally public (Supabase anon key, API base URL).*

7.1 Secret Classification

Classification	Example	Rule
Public	`NEXT_PUBLIC_API_URL`	Safe to commit. Set as environment variable in Vercel. Visible in browser bundle. Never put secrets here.
Server-only	`STRIPE_SECRET_KEY`	Set in Render / Vercel server-side env vars. Never in NEXT_PUBLIC_*. Never logged.
Webhook secrets	`STRIPE_WEBHOOK_SECRET`	Never committed. Set in platform env vars. Rotated if ever exposed. Used to verify inbound webhook signatures.
Database URL	`DATABASE_URL`	Contains credentials. Set in Render env vars only. The Prisma connection string includes the Supabase connection pooler URL.
Service role key	`SUPABASE_SERVICE_ROLE_KEY`	Bypasses Row Level Security. API only. Never in the frontend. Treat as a root password.

8. Environment Variable Register

Complete list of all environment variables required across both services. Set these before running either service locally or deploying.

8.1 clearpath-api (.env)

Variable	Service	Description / Where to get it
DATABASE_URL	Supabase	PostgreSQL connection string from Supabase project settings → Database → Connection string. Use the connection pooler (port 6543) for production.
SUPABASE_URL	Supabase	Project URL from Supabase project settings e.g. https://xyzxyz.supabase.co
SUPABASE_ANON_KEY	Supabase	Public anon key. Used to initialise Supabase client in API for Auth operations.
SUPABASE_SERVICE_ROLE_KEY	Supabase	Service role key. Bypasses RLS. Used for admin operations and Storage signed URL generation.
SUPABASE_STORAGE_BUCKET	Supabase	Name of the Storage bucket for certificate PDFs. e.g. certificates
STRIPE_SECRET_KEY	Stripe	API → Developers → API keys → Secret key. Starts with sk_live_ in production.
STRIPE_WEBHOOK_SECRET	Stripe	Webhooks → Add endpoint → Signing secret. Starts with whsec_. One per endpoint.
STRIPE_PRICE_ID_STANDARD_LONG	Stripe	Price ID for standard_long product from Stripe dashboard. Starts with price_.
STRIPE_PRICE_ID_REAL_ESTATE_SHORT	Stripe	Price ID for real_estate_short product.
STRIPE_PRICE_ID_TWIN_PACK	Stripe	Price ID for twin_pack product.
RESEND_API_KEY	Resend	API Keys → Create API key. Starts with re_.
RESEND_WEBHOOK_SECRET	Resend	Webhooks → signing secret for delivery status events.
RESEND_FROM_EMAIL	Resend	Verified sender address e.g. hello@clearpathrd.com
APP_BASE_URL	Config	https://clearpathrd.com in production. http://localhost:3000 locally.
API_BASE_URL	Config	https://api.clearpathrd.com in production. http://localhost:3001 locally.
ALLOWED_ORIGINS	Config	Comma-separated list of allowed CORS origins. e.g. https://clearpathrd.com,https://www.clearpathrd.com
NODE_ENV	Config	development | production. Set automatically by Render. Set manually locally.
PORT	Config	Port for Fastify to listen on. Render sets this automatically. Default 3001 locally.
CERTIFICATE_VERIFY_BASE_URL	Config	https://clearpathrd.com/verify in production. Used when constructing verification_url on certificates.
LOG_LEVEL	Config	info in production. debug locally. Fastify pino logger level.

8.2 clearpath-web (.env.local)

Variable	Service	Description / Where to get it
NEXT_PUBLIC_API_URL	Config	Base URL of the Fastify API. https://api.clearpathrd.com in production. http://localhost:3001 locally.
NEXT_PUBLIC_SUPABASE_URL	Supabase	Same as SUPABASE_URL in API. Public — safe in browser bundle.
NEXT_PUBLIC_SUPABASE_ANON_KEY	Supabase	Same as SUPABASE_ANON_KEY in API. Public — safe in browser bundle.
NEXT_PUBLIC_STRIPE_PUBLISHABLE_KEY	Stripe	Publishable key from Stripe dashboard. Starts with pk_live_ in production. Used to initialise Stripe.js in the browser.
NEXT_PUBLIC_SITE_URL	Config	https://clearpathrd.com in production. Used for OG metadata, canonical URLs.

9. Architecture Decision Records

Twelve ADRs documenting the major technical decisions made before writing code. Each follows the standard format: Context → Decision → Rationale → Alternatives → Consequences.

ADR-001 — Separate frontend and API services, not a Next.js monolith Status: Accepted	
Context	Next.js supports API routes natively. We could build the entire application as a single Next.js project with API routes handling backend logic.
Decision	Two separate services: Next.js (Vercel) for the frontend and Fastify (Render) for the API. No Next.js API routes for business logic.
Rationale	Business logic in Next.js API routes is harder to test in isolation, harder to reuse, and tightly coupled to the React framework. Fastify with Prisma is a better home for state machine logic, webhook handling, and scheduled jobs. The API can be developed and tested independently. If we later add a mobile app or partner integrations, the API is already decoupled.
Alternatives considered	Next.js full-stack monolith: simpler infrastructure, one deploy. Rejected because business logic complexity (state machines, webhooks, scheduled jobs) warrants a dedicated API.
Consequences	Increased infrastructure complexity (two deploys, two sets of env vars, CORS configuration). Accepted as a worthwhile trade-off for long-term maintainability.

ADR-002 — PostgreSQL via Supabase, not a standalone managed database Status: Accepted	
Context	We need PostgreSQL. Options include Supabase, Neon, PlanetScale (MySQL), Railway PostgreSQL, Render PostgreSQL, or a raw AWS RDS instance.
Decision	Supabase for PostgreSQL. Single project for MVP.
Rationale	Supabase provides PostgreSQL plus Auth (JWT issuance, email verification) plus Storage (for certificate PDFs) in one platform. This eliminates the need for a separate auth service and a separate file storage service. The free tier is generous for MVP. Row Level Security is a valuable safety net even though the API enforces ownership checks in application code.
Alternatives considered	Neon: serverless PostgreSQL with branching. Better for staging environments. Considered for a future staging setup. Neon + Clerk (auth) + Cloudflare R2 (storage) is a viable alternative stack. Railway PostgreSQL: simpler but no Auth or Storage bundled.
Consequences	Supabase vendor lock-in on Auth and Storage. Acceptable for MVP. Migration path exists: Supabase Auth can be replaced with any OIDC provider; Supabase Storage is S3-compatible.

ADR-003 — Prisma ORM over raw SQL or Drizzle Status: Accepted	
Context	Database access layer choices: raw SQL (pg), Drizzle ORM, Prisma ORM, Kysely.

Decision	Prisma ORM with Prisma Migrate for schema management.
Rationale	Prisma provides a generated type-safe client, a migration system, and a readable schema format. The schema.prisma file serves as executable documentation. Prisma Migrate provides a safe, version-controlled migration history. The generated client eliminates a class of runtime SQL errors. Drizzle is lighter-weight and closer to SQL, but Prisma's migration tooling and schema-as-documentation properties win for this project.
Alternatives considered	Drizzle: lighter, closer to SQL, excellent TypeScript inference. Considered seriously. Rejected for MVP because Prisma's migration system and schema file are more approachable for an early-stage product where the schema is still evolving. Drizzle is the right choice for a v2 rewrite.
Consequences	Prisma client generation adds a build step. The Prisma client does not handle connection pooling well at scale — Supabase's connection pooler (pgBouncer, port 6543) must be used in production via DATABASE_URL.

ADR-004 — Supabase Auth over Clerk, Auth.js, or custom JWT

Status: Accepted

Context	User authentication is a cross-cutting concern. Options: Supabase Auth, Clerk, Auth.js (NextAuth), custom Fastify JWT with bcrypt.
Decision	Supabase Auth. The Fastify API verifies Supabase JWTs on every protected request using the JWKS endpoint.
Rationale	Supabase Auth is already included in the Supabase stack. It handles email verification, password reset, and JWT issuance with no additional service or cost. The Fastify API verifies the JWT signature using Supabase's public JWKS endpoint — no session database required. The user.id in the JWT sub claim maps directly to the users table PK.
Alternatives considered	Clerk: polished DX, great React hooks, but adds cost and another vendor. Auth.js: good for Next.js monoliths but awkward when the API is a separate Fastify service. Custom JWT: full control but responsible for security-critical cryptography.
Consequences	Tightly coupled to Supabase. If Supabase is replaced, Auth must be replaced simultaneously. Accepted as a justified trade-off at MVP scale.

ADR-005 — Stripe for payments, not Paddle or a manual invoicing flow Status: Accepted	
Context	The platform sells physical kits at fixed prices ($54.99, $89.99, $99.99 CAD + tax). Payment options: Stripe, Paddle, Square, manual invoicing (email + e-transfer).
Decision	Stripe Checkout Sessions with webhook-driven order fulfillment.
Rationale	Stripe is the industry standard for developer-first payments. Stripe Checkout handles the payment UI, PCI compliance, and 3D Secure automatically. Stripe Tax can calculate Canadian provincial tax rates (HST/GST/PST) server-side, eliminating a manual tax table. Webhooks provide reliable, retryable payment event notifications.
Alternatives considered	Paddle: handles tax remittance automatically (merchant of record model). Attractive for avoiding Canadian GST/HST registration complexity, but Paddle's checkout UX is less polished. Revisit at scale. Manual invoicing: zero integration work but terrible UX and no automation.
Consequences	Stripe does not act as merchant of record — ClearPath RD is responsible for GST/HST registration and remittance once revenue crosses the $30,000 threshold. Stripe Tax assists with calculation but does not handle remittance.

ADR-006 — Resend for transactional email over SendGrid or SES Status: Accepted	
Context	Eight email types need to be sent reliably: order confirmation, activation, four check-in emails, results prompt, and certificate ready. Options: Resend, SendGrid, Amazon SES, Postmark, Mailgun.
Decision	Resend with React Email for template rendering.
Rationale	Resend has a developer-first API, an excellent free tier (3,000 emails/month), and first-class React Email integration. React Email templates live in the repository and can be previewed locally, version-controlled, and type-checked. Resend's webhook delivers delivery status events (delivered, bounced) which are required for the email audit log.
Alternatives considered	SendGrid: market leader, more complex API, worse DX. SES: cheapest at scale ($0.10/1,000 emails) but requires AWS account, sandbox approval, and a bounce/complaint handling system. Postmark: excellent deliverability, no free tier. Mailgun: legacy feel, documentation quality has declined.
Consequences	If email volume exceeds 3,000/month before revenue supports Resend's paid tier, migrate to SES. The Resend SDK call is isolated in src/services/email.ts — migration is a single-file change.

ADR-007 — node-cron for scheduled jobs, not a job queue Status: Accepted	
Context	Three scheduled jobs are needed: nightly session/certificate expiry, C-NRPP certification expiry alerts, and the email scheduling queue. Options: node-cron (in-process), BullMQ + Redis, Inngest, Trigger.dev, pg-boss.
Decision	node-cron running within the Fastify API process. No external job queue for MVP.
Rationale	At MVP scale (< 1,000 sessions), a job queue adds operational complexity that is not justified. node-cron runs scheduled functions on a cron schedule within the same

	Node.js process as the API. Jobs are idempotent — running twice has the same effect as running once. If the API restarts during a job, the job simply runs again at the next scheduled time. No Redis, no separate worker process, no dashboard.
Alternatives considered	BullMQ + Redis: production-grade job queue with retry, priority, and delayed jobs. The right choice when job reliability and throughput matter. Revisit when: jobs run longer than 30 seconds, jobs fail non-trivially, or volume requires a dedicated worker. Inngest / Trigger.dev: managed job queues with excellent DX. Evaluate at first sign of job reliability problems.
Consequences	Jobs run in the same process as the API. If the API is restarted, in-flight jobs are interrupted. Acceptable for nightly batch jobs. Not acceptable for critical path operations (e.g., sending a single important email) — those use direct Resend SDK calls, not the queue.

ADR-008 — PDF generation in-process using pdf-lib or Puppeteer, stored in Supabase Storage

Status: Accepted

Context	The certificate PDF must be generated server-side and stored for download. Options: pdf-lib (in-process JavaScript), Puppeteer (headless Chrome, HTML to PDF), external service (DocRaptor, WeasyPrint).
Decision	pdf-lib for in-process PDF generation. PDFs stored in a private Supabase Storage bucket. Download via signed URL.
Rationale	pdf-lib is a pure JavaScript library with no external dependencies — no headless browser, no external service, no additional cost. Certificate PDFs have a fixed, well-defined layout (see Certificate Generation Spec). pdf-lib gives precise control over typography and layout. Puppeteer would be simpler to style (CSS) but adds ~300MB to the API Docker image and significant memory overhead. Certificates do not need complex CSS rendering.
Alternatives considered	Puppeteer: HTML/CSS → PDF. Easier to maintain visually but expensive in terms of process memory (> 512MB RAM impact). Would require upgrading the Render service tier. External service (DocRaptor): clean API but $15+/mo fixed cost and an external dependency. Rejected for MVP.
Consequences	pdf-lib requires precise coordinate-based layout. Changes to the certificate design require code changes, not CSS changes. This is acceptable given the certificate layout is locked in the Certificate Generation Spec.

ADR-009 — TypeScript throughout, no JavaScript files in the API Status: Accepted	
Context	The Fastify API could be written in plain JavaScript or TypeScript.
Decision	TypeScript everywhere. tsconfig.json targets Node 18+. Build step: tsc or tsup. No ts-node in production.
Rationale	The Prisma generated client is typed. The Fastify ecosystem has excellent TypeScript support. Type safety on database entities, API request/response shapes, and enum values eliminates a large class of runtime errors. The upfront cost of TypeScript configuration is paid once; the benefit compounds over the entire development lifetime.
Alternatives considered	JavaScript: faster to start, no build step. Rejected because the schema is complex (11 entities, 16 enums) and runtime type errors against the database would be difficult to diagnose.
Consequences	A build step is required before running the API (tsc or tsup). Render's build command handles this. Locally, tsx watch src/server.ts provides hot-reload without a manual build step.

ADR-010 — Row Level Security disabled at the table level, enforced in application code Status: Accepted	
Context	Supabase supports PostgreSQL Row Level Security (RLS) which can enforce data ownership at the database level. The API could rely on RLS as a safety net, or enforce ownership purely in application code.
Decision	RLS is kept disabled on all tables for MVP. Ownership is enforced in Fastify route handlers by scoping all queries to the authenticated user's ID.
Rationale	The API uses the SUPABASE_SERVICE_ROLE_KEY which bypasses RLS regardless. RLS only provides protection if client applications access the database directly (e.g., Supabase JS client from the browser). Since all data access goes through the Fastify API, RLS adds no meaningful security benefit in this architecture. Maintaining RLS policies in parallel with application-layer ownership checks introduces maintenance overhead and a risk of drift.
Alternatives considered	Enable RLS as defence-in-depth: if a bug in application code returns the wrong user's data, RLS would catch it. Valid argument. Revisit if the frontend ever uses the Supabase JS client to access the database directly (e.g., realtime subscriptions). If that pattern is adopted, enable RLS.
Consequences	All data ownership enforcement is in application code. A bug in a Fastify route handler could expose data across users. Mitigated by: standard code review, ownership-checking utility functions used consistently across routes, and integration tests that verify cross-user access is rejected.

ADR-011 — Fastify over Express for the API framework Status: Accepted	
Context	Node.js API framework choice: Express, Fastify, Hono, Koa.
Decision	Fastify.

Rationale	Fastify is faster than Express (benchmark: ~2× throughput on simple routes), has first-class TypeScript support, a schema-first validation system (JSON Schema / Typebox), and a plugin architecture that maps well to the modular route structure. Fastify's built-in request validation reduces boilerplate. The fastify-jwt and @fastify/cors plugins are well-maintained.
Alternatives considered	Express: familiar, massive ecosystem, but no built-in TypeScript support or schema validation. Hono: extremely fast, excellent TypeScript, edge-compatible, but smaller plugin ecosystem. Worth evaluating for v2. Koa: middleware-based, less popular for new projects.
Consequences	The team must learn Fastify's plugin system if unfamiliar. Fastify's schema validation (JSON Schema) is more verbose than Zod but avoids a runtime dependency. Fastify's error handling model is different from Express — errors must be returned via reply.send(), not throw.

ADR-012 — Single GitHub repository per service, manual deploys for MVP Status: Accepted	
Context	Code organisation options: monorepo (Turborepo, Nx) with both services, or two separate repositories. Deployment: GitHub Actions CI/CD pipeline, or platform-native auto-deploy on push.
Decision	Two separate GitHub repositories. No CI/CD pipeline. Render and Vercel auto-deploy from main branch on push.
Rationale	A monorepo would require Turborepo or Nx configuration, shared package management, and additional build complexity. For two services that share no code (the API and frontend do not share TypeScript types or utilities at launch), a monorepo adds overhead without benefit. Platform-native auto-deploy (Render and Vercel watch the main branch) provides adequate deployment automation without a GitHub Actions pipeline. The absence of automated tests at MVP launch means a CI pipeline would only add latency to deploys, not safety.
Alternatives considered	Monorepo: enables shared TypeScript types between API and frontend (e.g., shared API response types). Revisit when: API response shapes need to be consumed by the frontend with type safety, or when a mobile app is added. GitHub Actions: required when: automated tests are added, or when the deploy process requires non-trivial steps beyond build + deploy.
Consequences	No shared types between API and frontend at launch. API response shapes are documented in the OpenAPI spec and must be manually kept in sync with the frontend's API client. This is technical debt that should be addressed in v2 with a shared types package or OpenAPI code generation.

10. Pre-Launch Infrastructure Checklist

Complete this checklist before writing the first production-bound line of code and again before launching publicly.

Supabase

- Create Supabase project and record project URL and anon key
- Create service role key and store securely (not in Git)
- Run prisma migrate deploy against the Supabase database to initialise schema
- Run seed script for radon_zone_map (prisma/seed/radon_zones.ts)
- Run seed script for initial contractor listings
- Create certificates Storage bucket — set to private (not public)
- Confirm email verification is enabled in Supabase Auth settings
- Set Supabase Auth redirect URL to https://clearpathrd.com/auth/callback
- Set Supabase Auth email template sender to hello@clearpathrd.com (after DNS verification)

Stripe

- Create Stripe account and complete business verification
- Create three Products and Prices in Stripe dashboard (standard_long, real_estate_short, twin_pack)
- Enable Stripe Tax for Canada — set nexus for applicable provinces
- Register webhook endpoint: https://api.clearpathrd.com/api/v1/webhooks/stripe
- Select events: payment_intent.succeeded, payment_intent.payment_failed, charge.refunded
- Copy webhook signing secret (whsec_...) to Render environment variables
- Test webhook flow with Stripe CLI before go-live

Resend

- Create Resend account
- Add and verify clearpathrd.com domain in Resend (requires DNS TXT and DKIM records)
- Create API key with send permissions
- Register webhook endpoint: https://api.clearpathrd.com/api/v1/webhooks/resend
- Select events: email.delivered, email.bounced, email.complained
- Copy webhook signing secret to Render environment variables
- Send a test email to confirm domain verification and DKIM signing

Vercel

- Connect clearpath-web GitHub repository to Vercel
- Add all NEXT_PUBLIC_* environment variables in Vercel project settings
- Add custom domain: clearpathrd.com and www.clearpathrd.com
- Confirm SSL certificate is provisioned
- Test production build locally with next build before first Vercel deploy

Render

- Connect clearpath-api GitHub repository to Render
- Set service type to Web Service, runtime Node
- Set build command: npm install && npx prisma generate && npm run build
- Set start command: node dist/server.js
- Add health check path: /health
- Add all environment variables from Section 8.1 to Render service
- Add custom domain: api.clearpathrd.com
- Confirm SSL certificate is provisioned
- Deploy and confirm /health returns HTTP 200

Final verification

- End-to-end test: register account → verify email → add home → purchase kit → confirm order email received
- Webhook test: trigger Stripe payment_intent.succeeded in test mode, confirm TestSession is created
- Certificate test: submit a result, confirm certificate is generated and PDF is accessible
- Public verify test: open https://clearpathrd.com/verify/[certificate-uuid] without being logged in
- Confirm CORS is rejecting requests from non-allowlisted origins
- Confirm SUPABASE_SERVICE_ROLE_KEY is not present in any frontend bundle

— End of Infrastructure & Architecture Decision Record —
ClearPath RD | clearpathrd.com | Version 1.0 | February 2026