Based in Canada · Remote or relocation · Available now

Al Mokhtar Bekkour

I scale and de-risk *multi-tenant SaaS* — the *rewrites* and *migrations that can't go down*.

Seven years on Rails, with Go for the paths that take real load. The engineer brought in when a system has to change without breaking — incremental migrations, no big-bang cutover, no downtime a customer can see.

See the work Full résumé →

WHAT I'M AFTER	Senior / Staff — Backend or Full-stack Rails	Remote (Canada) or relocation	Small teams that ship and care about correctness

BY THE NUMBERS	7+ years — Senior Rails & Go	40% faster critical Postgres queries	1,000+ tenants on one platform	~60% less manual bookkeeping

OPEN TO	Full-time · Contract · Advisory

Experience

7+ YEARS · 5 ROLES

2021 — Now

C-CUBE Senior Software Developer

Cut critical Postgres query times 40% on the multi-tenant platform behind 1,000+ roofing contractors. Wired QuickBooks, Chargebee and Nylas together with idempotent, webhook-driven sync that survives partner outages. Owned the AWS side end to end: SQS/Sidekiq, CI/CD, the Prometheus/Grafana/Loki boards we actually watch.

Rails · PostgreSQL · AWS · Sidekiq

2020 — 2021

Rounded Senior Software Developer

Cut manual bookkeeping ~60% for 10,000+ freelancers with automated invoicing and tax workflows. Real-time FreshBooks and Stripe sync, where a rounding error is a support ticket.

Rails · Stripe · PostgreSQL

2019 — 2020

HiFive Group

Senior Software Developer

Built ProCRM from nothing — millions of records across call-center floors in Tangier and London. A write-heavy Postgres schema and the REST APIs feeding live cross-office KPI boards.

Rails · PostgreSQL · REST

2018 — 2019

AgriVillage

Software Developer

Agritourism marketplace — booking, Stripe payments, listings. Cut page times with query work and caching.

Rails · Stripe

2017 — 2018

Com en Scène

Software Developer

15+ client sites in PHP, Symfony and Drupal. Where I learned to ship on someone else's deadline.

PHP · Symfony · Drupal

APP ACADEMY — FULL-STACK WEB DEVELOPMENT · OFPPT — COMPUTER SCIENCE

Stack

BACKEND	Ruby on Rails · Go · Hotwire · GraphQL · REST · Sidekiq
SYSTEMS	Linux · Profiling · Memory & query performance · C
DATA	PostgreSQL · Redis
CLOUD & INFRA	AWS (EC2 · RDS · S3) · SQS · Docker · GitHub Actions
OBSERVABILITY	Prometheus · Grafana · Loki

HOW I WORK

The outages I've fixed were rarely clever. A missing index. A webhook with no idempotency key. A tenant boundary nobody tested. *I look for those first.*

Built to last, not to impress

PHILOSOPHY

Code is read far more than it is written. The system shipped today will be debugged at 3 a.m. in 2034 by someone who has never met you. I write for that person.

I choose mechanical reliability over novelty. Boring, observable and idempotent beats clever almost every time. A system that fails loudly and predictably is worth more than one that is elegant until the day it isn't. A new framework is a liability until it has earned its place — the cost of a system is not its first sprint, it is its next ten years of maintenance.

Legacy code is evidence, not an enemy. It is running the business. The job is not to be offended by it — it is to understand why it works, find the seams, and change it without anyone noticing. The strongest engineering decision is usually the smallest one that still solves the problem.

My bias: fewer moving parts, clear boundaries, and failure modes I can explain before they happen. That is what lets a team move fast for years instead of months.

Built to be understood, not just to run.

What broke, and what I changed

LESSONS FROM THE TRENCHES

A partner API went dark mid-sync.

Handlers weren't idempotent. On retry the same webhook applied twice — silent double-writes nobody caught until reconciliation.

FIX Idempotency keys and replay-safe handlers. A partner outage became a delay, not data corruption.

Critical queries slowed as tenants grew.

A query shape that was fine at ten tenants and quietly fatal at a thousand — wrong index, wrong access path under tenant filtering.

FIX Targeted indexing and query rework. 40% off the critical paths. The win was unglamorous, not architectural — most of them are.

One tenant could, in theory, see another.

Isolation was assumed at the app layer and never actually tested. The worst kind of bug: invisible until it is a breach.

FIX Tenant scoping enforced as a default and covered by tests, so isolation is structural — not a per-query discipline.

Selected work

SIDE PROJECTS

Constat ↗ B2B SAAS

Bug reports that don't make customers explain themselves. One link captures their screen, console and network; support replays it. Rails 8 + Hotwire dashboard, a Go service streaming uploads over WebSockets, a TypeScript recorder and a Chrome extension — four ways in, one backend, working end to end.

Rails 8 · Go · Hotwire · WebSockets

Railyard ↗ DISTRIBUTED SYSTEMS

A Rails control plane that drives a fleet of Go agents over gRPC: per-server routing, Docker/BuildKit deploys, live-streaming build logs and agent heartbeats. The split I keep coming back to — Rails for the product, Go for the load.

Rails · Go · gRPC · Docker

CASE STUDY

How to rewrite a SaaS without killing the business

The fear isn't the rewrite. It's losing revenue, data or customers during it. Here is how I avoid that.

THE CHAOS

An aging system where every change risked the whole platform. Tight coupling, no seams, a test suite nobody trusted. The instinct — “let's rewrite it from scratch” — is exactly what kills companies: a day-one rewrite freezes the business for months and usually ships a second system carrying the first one's bugs.

THE STRATEGY

Strangle, don't replace. Carve seams in the running system, route one slice at a time behind the live product, migrate incrementally with data intact. Rails stays the product surface; Go takes the paths that need the load. Every step is shippable and reversible. No big-bang cutover, no code freeze.

THE OUTCOME

The business kept shipping features while the architecture changed underneath it. No migration weekend, no rewrite death march, no customer-visible downtime. The new system arrived one safe step at a time — which is the only way it ever actually arrives.

A rewrite is a series of small, boring, reversible decisions. That's the whole trick.

CONTACT

Let's build *something*.

hey@almokhtar.dev

LinkedIn GitHub GitLab Himalayas

hey@almokhtar.dev · linkedin.com/in/almokhtarbekkour · github.com/almokhtarbr · gitlab.com/almokhtarbekkour

---

Senior Rails & Go Engineer — Multi-tenant SaaS at scale