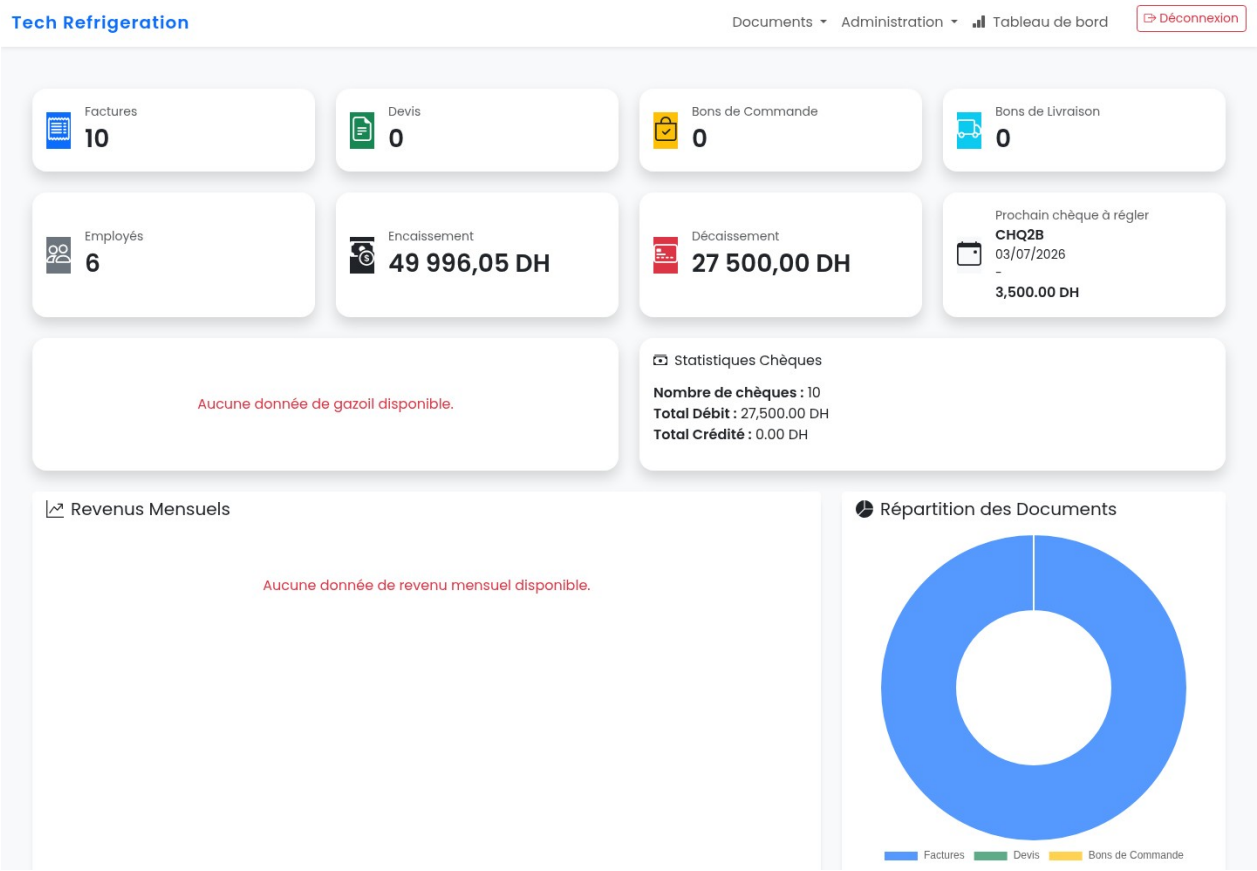
Centralizing field-service, commercial and financial operations in one Laravel platform

A real SME operations platform connecting customers, quotations, delivery documents, invoices, stock, interventions, workforce tasks and operating expenses.

STATUS	TEAM	DEMO	SOURCE
Real client delivery · Demo hardening	**Solo delivery**	**Protected demo planned**	**Private repository**



Primary visual evidence

ANASS BENAMEUR · FULL-STACK PRODUCT ENGINEER

The product, the problem and the proof

A refrigeration-services company was running customers, quotations, invoices, stock and technician work across paper, memory and disconnected files. I was brought in for four weeks, solo, to fix that. What shipped is a Laravel platform where a quotation, the invoice it becomes, the stock movement it triggers and the technician who executed the job are the same connected record, not four separate ones someone has to reconcile by hand.

4 weeks	Solo	18+	Desktop + mobile
ORIGINAL DELIVERY WINDOW	END-TO-END OWNERSHIP	CAPTURED OPERATIONAL MODULES	RESPONSIVE ACCESS

Context

Fractal Tech brought me in to build the operating system for a real refrigeration-services SME between July and August 2025: a company that sells, quotes, delivers and invoices refrigeration equipment and interventions, and had been doing all of it without a shared system. Four weeks, one developer, one client who needed something that actually worked on day one — every data-modelling decision had to be right the first time, because there wasn't slack in the schedule to redo the relational core halfway through.

My role and ownership

- Requirements analysis and workflow mapping
- Merise/UML relational data modelling
- Laravel backend and Blade interface
- Authentication and role-oriented access
- Deployment, documentation and user handoff

Personal contribution

Requirements, architecture, data model, backend, frontend, deployment, documentation and handoff.

Why the existing workflow needed a system

None of this was a technology failure so much as the predictable result of a business that grew its operations faster than its record-keeping, which is most SMEs at this exact stage. Quotations were typed fresh each time because nothing linked back to a customer's history. Invoices existed with no reliable trace to the job or the stock they consumed. Technicians' fieldwork wasn't visible to the office, and the office's paperwork wasn't visible to technicians. A stock count meant someone walking to the shelf, because nothing tracked what a completed job had actually consumed.

Before and after

BEFORE

- Scattered customer and supplier records
- Repeated manual preparation of business documents
- Weak traceability between commercial and field activity
- Disconnected stock, cash, fuel and repair records
- No unified workspace for administration and technicians

AFTER

- Centralized master data and operational records
- Structured quotation-to-invoice workspaces
- Searchable history across documents and interventions
- Inventory, cash and operating-cost modules in one platform
- Responsive access for office and field use

Meaningful result

One system replaced disconnected customer, document and operational records.

How the system works

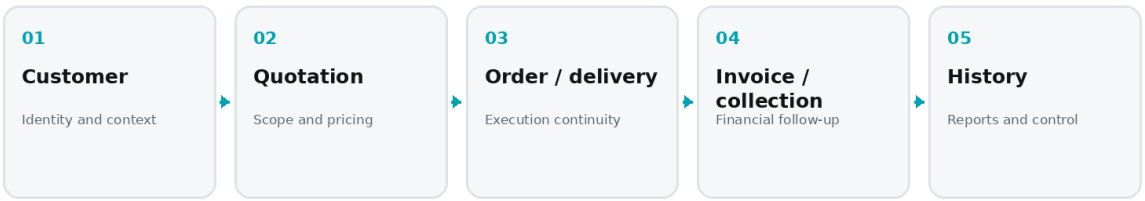
Everything hangs off five connected states rather than five independent modules that happen to share a login page: a customer record establishes identity and history; a quotation captures scope and price against that customer; an order/delivery step preserves execution continuity once the quotation is accepted; an invoice closes the loop financially and stays permanently linked back to the job that generated it; and a history layer makes every one of those states searchable across customers, documents and interventions.

Operational workflow

01	02	03	04	05
Customer	**Quotation**	**Order / delivery**	**Invoice / collection**	**History**
Identity and context	Scope and pricing	Execution continuity	Financial follow-up	Reports and control

SYSTEM ARCHITECTURE

Integrated Field Service & Operations Platform



Evidence-led architecture diagram · generated from the verified project workflow

Stock movements attach to the same relational core: an intervention that consumes a part writes a stock-ledger entry, not a manual adjustment someone has to remember to make later. Technician task assignments and operating expenses follow the same pattern. The result is that a manager can open one customer and see their full quotation-to-invoice history, the stock consumed against it, and which technician handled it, in one place, without cross-referencing five different documents. The sequence below is exactly what ships in the delivered system:

Architecture and decisions

Laravel 10 and PHP 8 over anything more exotic — a four-week SME engagement needs a framework where the distance from data model to shipped feature is short, not a chance to experiment with something unproven. MySQL for transactional integrity on financial documents. Blade instead of a separate SPA: it kept delivery cohesive, ran comfortably on the ordinary office hardware the client actually has, and still supported distinct views per role without the overhead of a second deployment target under a four-week clock.

Critical engineering decisions

DECISION 01

Connected records over isolated CRUD

Customers, documents, stock movements, interventions and financial events share relational context, reducing duplicate entry and preserving operational history.

DECISION 02

Server-rendered operational interface

Blade kept delivery cohesive and pragmatic for ordinary business hardware while supporting role-specific views.

DECISION 03

Documents as workflows

Quotations, orders, delivery notes, invoices and collections were treated as business states rather than disconnected files.

DECISION 04

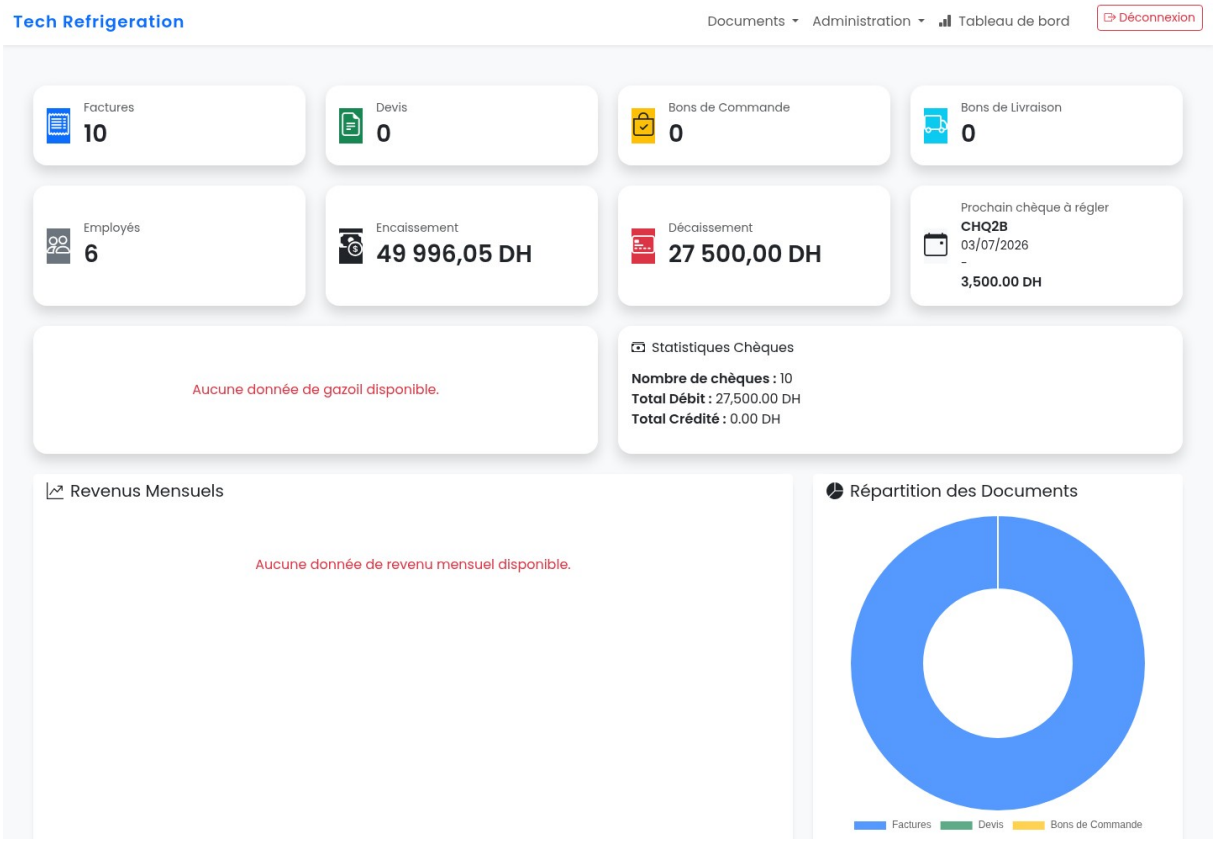
Safe proof environment

The portfolio environment uses synthetic data and an isolated database, proving breadth without exposing client information.

Technology stack

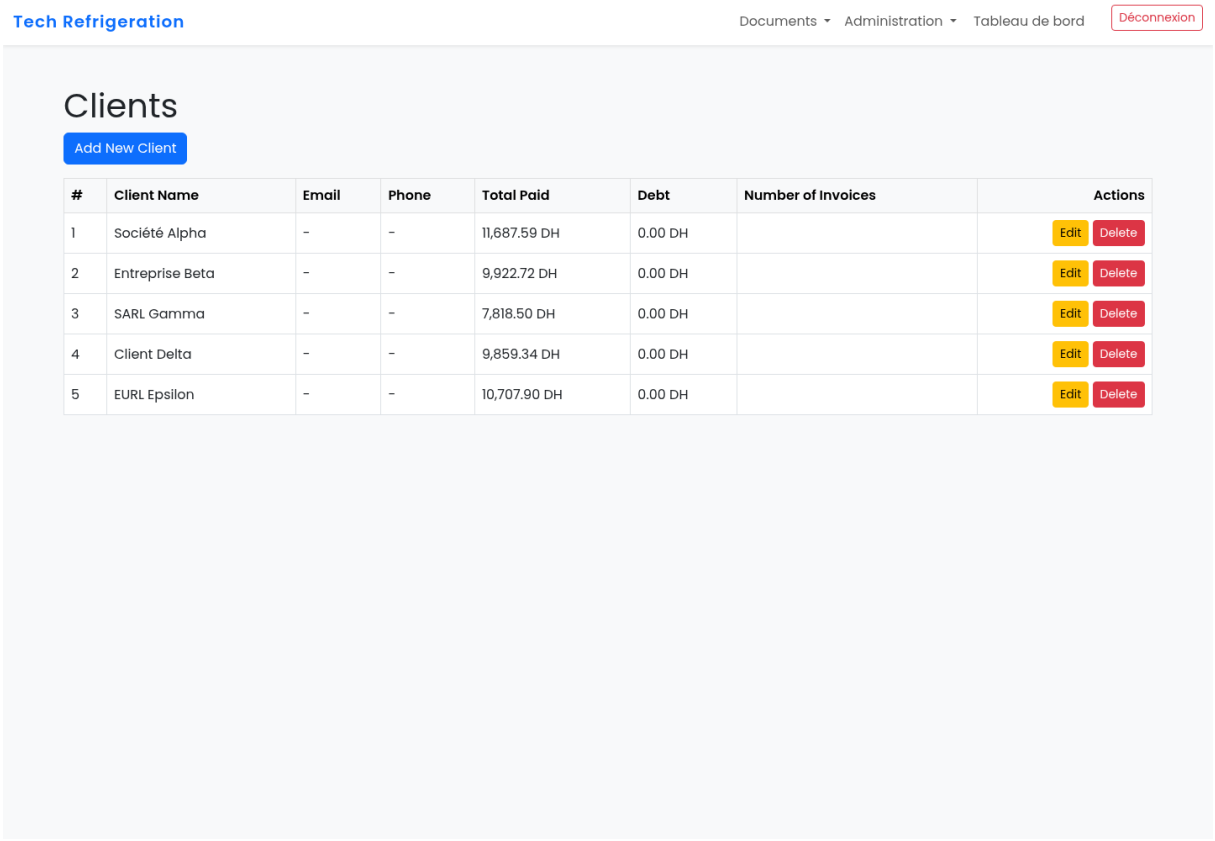
Laravel 10	PHP 8	MySQL	Blade	MVC	Role-based access

Evidence 1 of 4



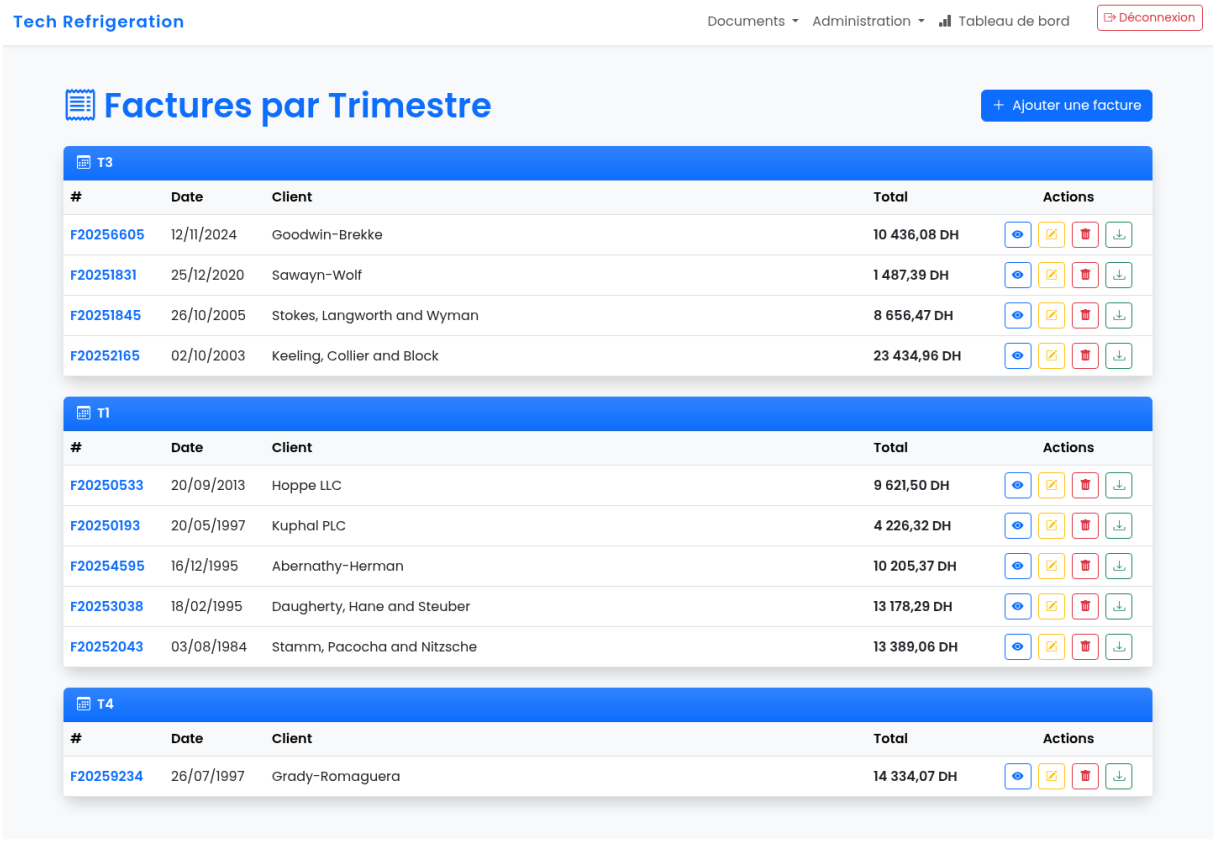
Operational cockpit combining document, cash and activity indicators.

Evidence 2 of 4



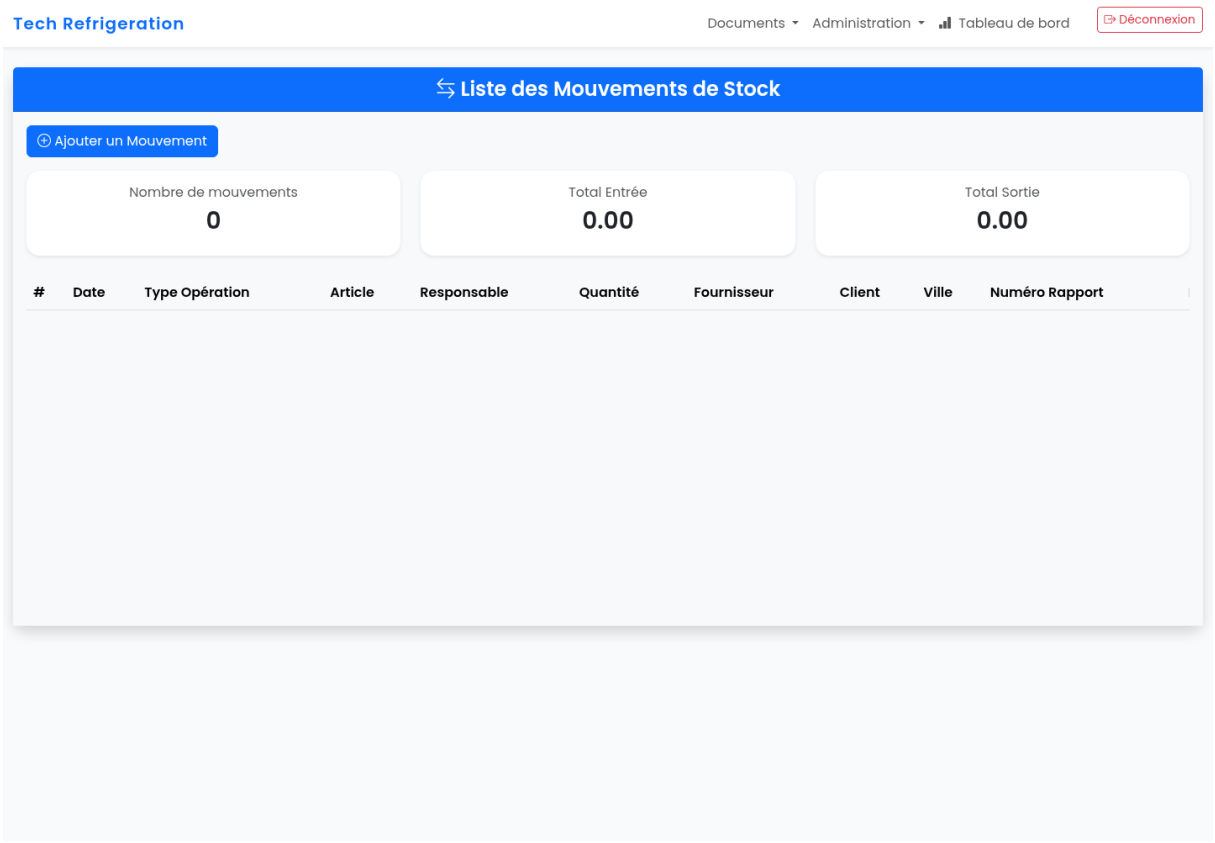
Central customer directory feeding commercial and intervention workflows.

Evidence 3 of 4



Invoice workspace preserving financial document history.

Evidence 4 of 4



Inventory movement ledger for inbound, outbound and adjustment events.

Value, proof and honest limitations

Operational impact

The client went from five disconnected record types to one searchable operational history, with management able to see documents, stock and field activity in the same place for the first time. That is the actual deliverable — not a UI, a reconciliation problem that stopped existing. The expanded source shows 33 migrations, 32 controllers and 28 models across 120 Blade views, which is the shape of a real relational business system, not a demo CRUD app with a login screen bolted on.

Verified evidence

- Real SME delivery recorded in the CV
- 18+ captured operational modules
- Desktop and mobile responsive captures
- Expanded source inventory includes 33 migrations, 32 controllers, 28 models and 120 Blade views
- Original handoff included technical documentation and a user guide

Current limitations

- The later expanded source requires a dedicated hardening release before public deployment.
- Automated tests are insufficient for document transitions, permissions and stock reconciliation.
- Several screenshots use sparse synthetic data and prove interface scope more than production volume.
- The public case study distinguishes the original delivery from later experimental expansion.

Current status

STATUS	TEAM	DEMO	SOURCE
Real client delivery · Demo hardening	**Solo delivery**	**Protected demo planned**	**Private repository**

WHY THIS PROJECT MATTERS

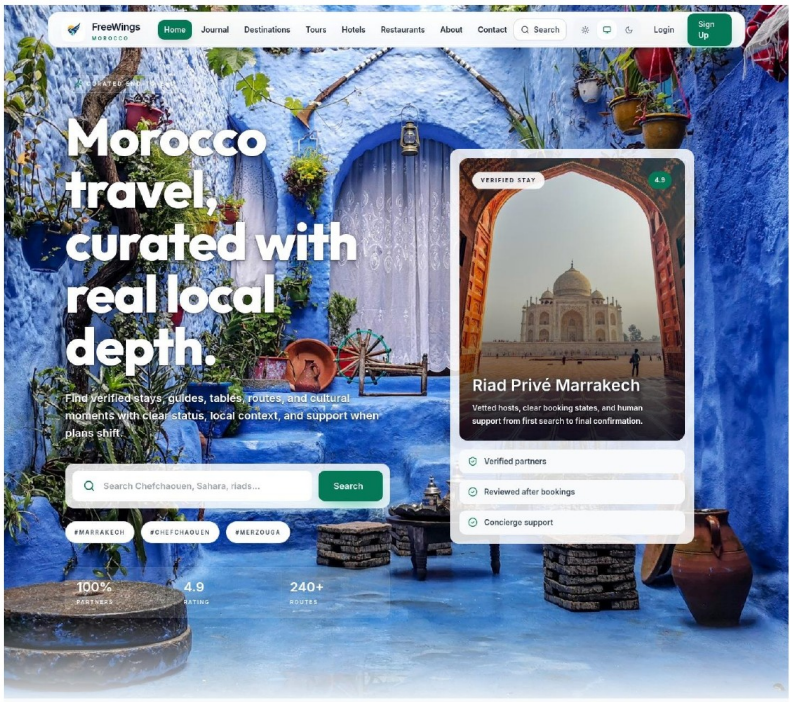
One system replaced disconnected customer, document and operational records.

This case study is designed to be inspectable: strong claims are connected to screenshots, source evidence or explicitly stated limitations.

Engineering a unified Moroccan tourism platform from discovery to reservation and payment

A decoupled Laravel and React product connecting tourism discovery, recommendations, bookings and CMI payment workflows.

STATUS	TEAM	DEMO	SOURCE
Two-person academic product · Functional demonstration	Two-person project	Functional demo deployment planned	Private project source



Primary visual evidence

ANASS BENAMEUR · FULL-STACK PRODUCT ENGINEER

The product, the problem and the proof

FreeWings is a two-person final-cycle product exploring a unified Moroccan tourism experience. It brings destinations, hotels, tours, restaurants, activities and editorial content into one discovery journey, then connects search and recommendation to a polymorphic booking flow and a real integration with CMI — Morocco's interbank card payment gateway — implemented with HMAC-SHA512 signing, idempotent webhook handling and transactional reconciliation, running in certification/preproduction mode.

2 people	**94 pages**	**6 domains**	**CMI**
DOCUMENTED PROJECT TEAM	TECHNICAL REPORT	TOURISM CONTENT TYPES	CERTIFICATION-MODE INTEGRATION

Context

Travellers researching Morocco bounce between a hotel site, a tour operator, a restaurant listing and a blog, then often book nowhere in particular because nothing connects the pieces into one journey. FreeWings, built for a licence-level final-cycle project at SUPMTI with my partner Oumaima Bakkali, tests whether a single product — for travellers, tourism professionals and administrators — can hold that whole journey without collapsing the real differences between what a hotel, a restaurant and a multi-day tour actually need to sell.

My role and ownership

- Shared product and technical design
- Laravel REST API and domain modelling
- React/TypeScript product interface
- Search, recommendation and booking workflows
- Payment, security and observability design

Personal contribution

Shared architecture and implementation across API, SPA, booking, payment, security and observability.

Why the existing workflow needed a system

A hotel, a restaurant and a guided tour are not the same kind of thing to sell — different pricing logic, different availability rules, different cancellation terms — but a traveller wants to compare and book all three without noticing the difference underneath. On top of that, a payment gateway callback can arrive twice, arrive late, or arrive with a tampered amount, and a booking system that isn't built for that will either double-charge someone or lose track of whether they paid at all.

Before and after

BEFORE

- Fragmented tourism discovery across unrelated services
- No shared search and recommendation experience
- Different reservation flows for each service type
- Payment callbacks vulnerable to duplicate processing without idempotency
- Limited cross-layer visibility into failures

AFTER

- One catalogue spanning destinations, hotels, tours, restaurants and activities
- Unified search and recommendation experience
- Polymorphic booking lifecycle shared across service types
- HMAC-SHA512 validation, amount checks and idempotent reconciliation design
- Request IDs, HTTP telemetry, Web Vitals and client error reporting

Meaningful result

One coherent product experience across discovery, reservation and payment states.

How the system works

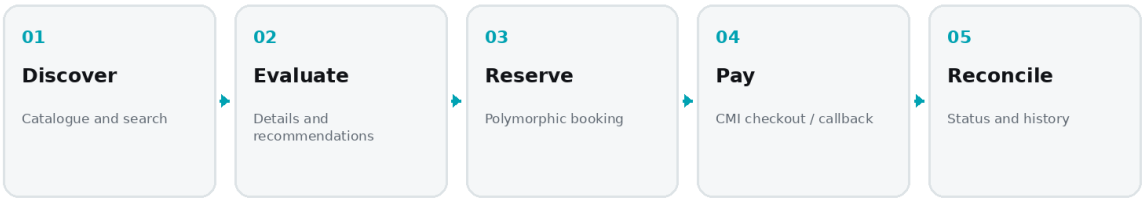
Two independent applications talk over REST. Laravel handles routing, FormRequest validation, Policy-based authorization, services and API resources; React handles the interface with TanStack Query for server state and Zod for typed validation. A booking is polymorphic — one bookings table serves hotels, tours and restaurants alike via bookable_type/bookable_id — so adding a fourth service type doesn't mean building a fourth booking system from scratch.

Operational workflow

01	02	03	04	05
Discover	**Evaluate**	**Reserve**	**Pay**	**Reconcile**
Catalogue and search	Details and recommendations	Polymorphic booking	CMI checkout / callback	Status and history

SYSTEM ARCHITECTURE

FreeWings — Intelligent Tourism Platform



Evidence-led architecture diagram · generated from the verified project workflow

Payment runs in three phases: the backend signs a payload with HMAC-SHA512 and hands the traveller off to CMI's hosted checkout, so card data never touches FreeWings' own servers; CMI posts a webhook back, verified against the signature, checked for amount match, and processed through a firstOrCreate idempotency guard so a duplicate callback is silently absorbed instead of double-processed; and the traveller's return page polls booking status every five seconds until the reconciliation lands. A parallel observability layer — request IDs, HTTP telemetry, Web Vitals, client-side error reporting via sendBeacon — exists so a failure anywhere in that chain is diagnosable, not a mystery.

Architecture and decisions

Two independent applications communicate through REST. Laravel organizes routes, middleware, FormRequests, controllers, policies, services, models and API resources. React uses TypeScript, TanStack Query, Zod and feature-oriented components. The recommendation score is computed directly in SQL — rating times 20, plus review count times 4, plus active hotels, restaurants, tours and activities weighted separately — ordered with orderByRaw so ranking stays fast without pulling every destination into PHP to sort it.

Critical engineering decisions

DECISION 01

Decoupled API and SPA

The split supports a richer customer experience and reusable contracts, while requiring deliberate authentication and deployment coordination.

DECISION 02

Polymorphic booking model

Hotels, tours and activities share reservation lifecycle logic without forcing every service into an identical data model.

DECISION 03

Idempotent payment reconciliation

CMI callbacks may repeat, so signature checks, amount validation and controlled reconciliation protect reservation and payment state.

DECISION 04

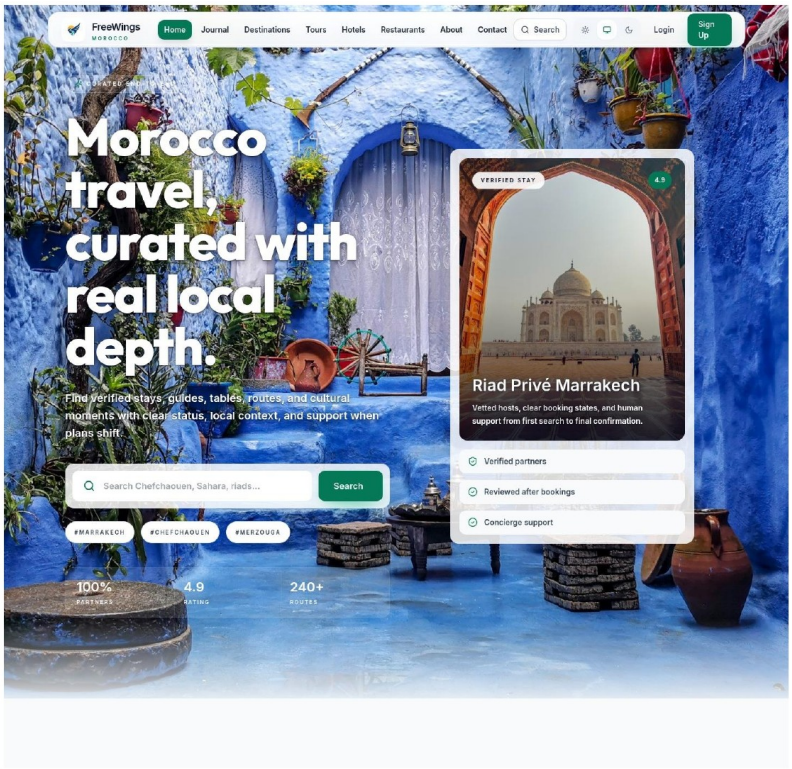
Client and server instrumentation

Request IDs, HTTP telemetry, Web Vitals and sendBeacon error reporting connect user experience with backend diagnostics.

Technology stack

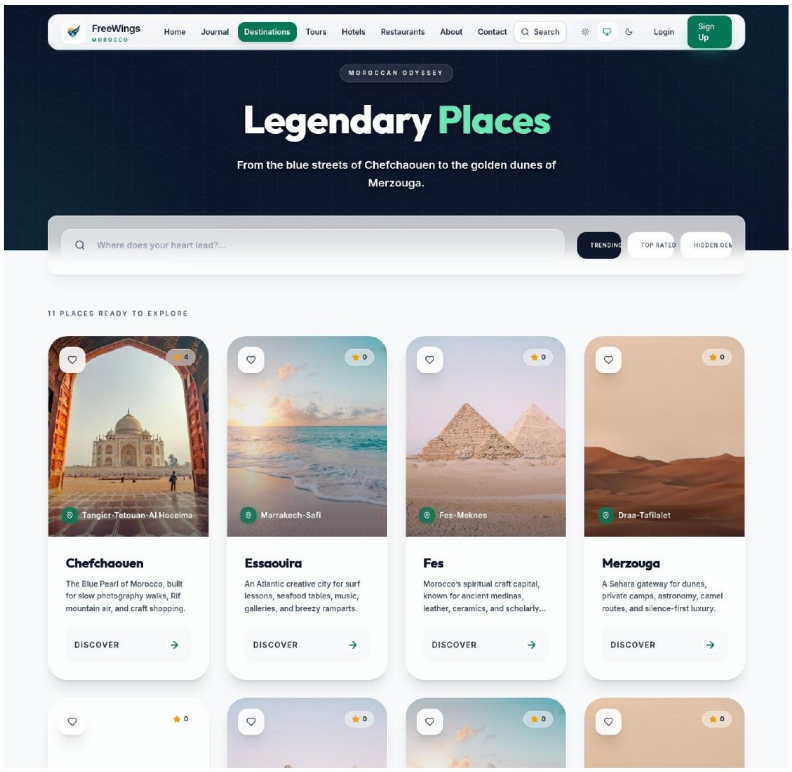
Laravel 10	PHP 8.2	React 18	TypeScript	Sanctum	CMI 3D Secure

Evidence 1 of 4



Homepage combining discovery, positioning and recommendations.

Evidence 2 of 4

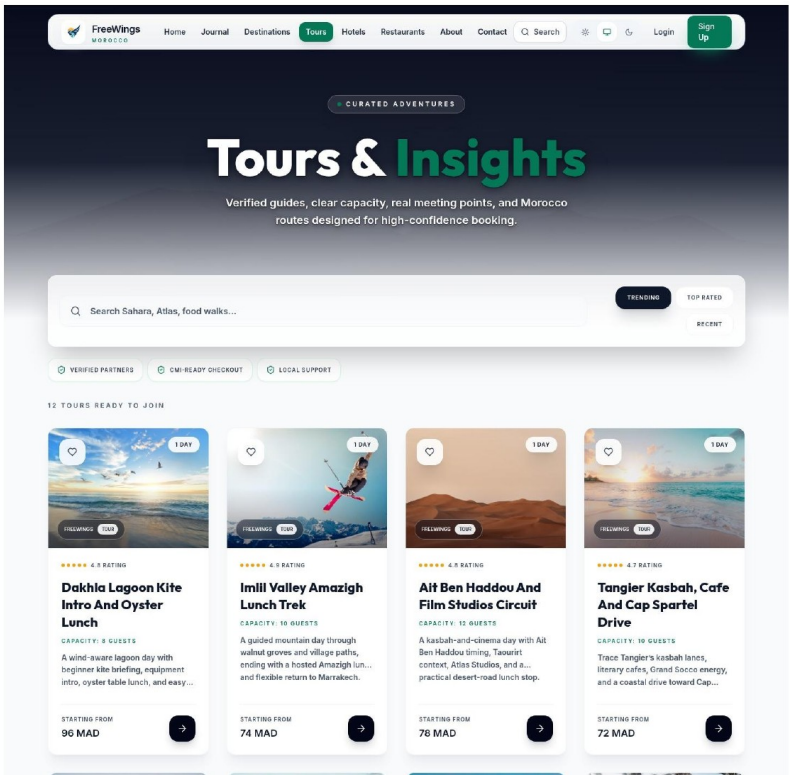


Destination catalogue designed for comparison and exploration.

Evidence 3 of 4

Destination detail connecting context, content and conversion actions.

Evidence 4 of 4



Tour catalogue presenting bookable experiences.

Value, proof and honest limitations

Operational impact

FreeWings demonstrates the harder thing: a coherent multi-entity marketplace experience, not a single CRUD module wearing a nice UI. The supplied 94-page report and interface captures show a complete discovery journey, detailed service pages, authentication, unified search, both payment-return states, and editorial content — the full shape of a real product, not a slice of one.

Verified evidence

- 94-page final-cycle report with architecture and UML
- Real UI captures for catalogues, detail pages, search and authentication
- Payment-return screens for pending and failed states
- Documented CMI signature, idempotency and reconciliation design
- Health, request telemetry and Web Vitals architecture

Current limitations

- Developed by two contributors and never presented as a solo build.
- CMI was implemented in certification/preproduction mode, not proven live commerce.
- Bearer-token storage in localStorage should be hardened because it increases XSS exposure.
- No independent production load test or security audit is claimed.

Current status

STATUS	TEAM	DEMO	SOURCE
Two-person academic product · Functional demonstration	**Two-person project**	**Functional demo deployment planned**	**Private project source**

WHY THIS PROJECT MATTERS

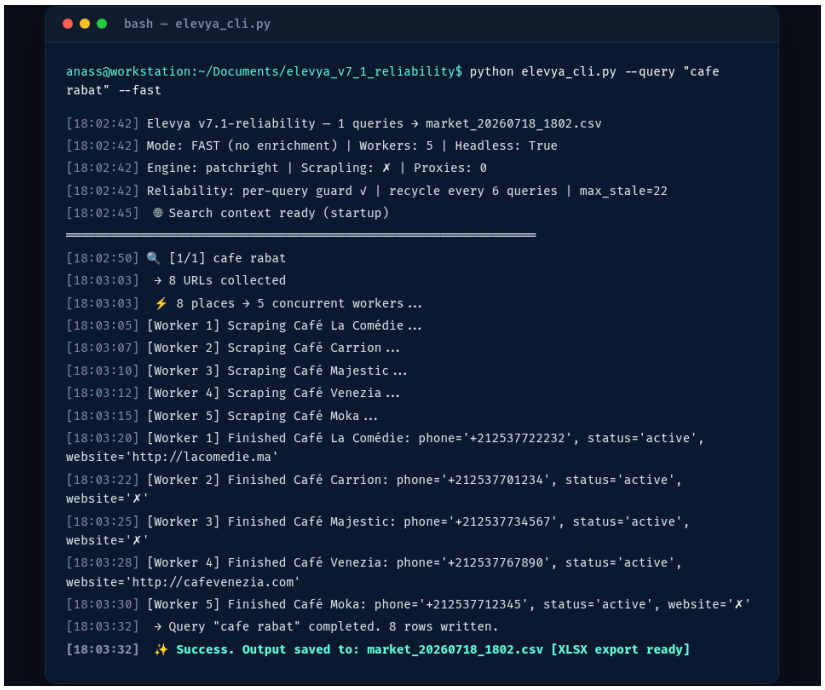
One coherent product experience across discovery, reservation and payment states.

This case study is designed to be inspectable: strong claims are connected to screenshots, source evidence or explicitly stated limitations.

Automating local-market research into enriched and prioritized commercial intelligence

A modular Python acquisition and enrichment system that converts public business information into normalized, deduplicated and ranked prospect data.

STATUS	TEAM	DEMO	SOURCE
Working technical demonstration · Quality hardening	**Solo engineering project**	**Controlled demo planned**	**Private repository**



Primary visual evidence

ANASS BENAMEUR · FULL-STACK PRODUCT ENGINEER

The product, the problem and the proof

Manual local-market research is repetitive in a way that doesn't reward care: search, open each listing, check the website, copy fields into a spreadsheet, notice duplicates two hundred rows later. I built a modular Python engine that does that acquisition and enrichment work end to end — collecting public local-business listings, enriching each one from its own website, deduplicating, and scoring the result so a reviewer starts with the businesses worth calling, not an alphabetical dump.

297	38	95.6%	98.0%
UNIQUE RECORDS IN SUPPLIED DEMO	NORMALIZED FIELDS	PHONE COVERAGE IN DEMO	COORDINATE COVERAGE IN DEMO

Context

Built for prospect-research workflows: a team needs to know which local businesses are active, whether they have a real web presence, and which ones are worth prioritizing, with a GUI for a one-off research pass and a CLI for repeatable, automatable jobs. It is the acquisition-mechanics counterpart to the other data platforms in this portfolio: where the ATS Intelligence Engine solves normalization across ten APIs and Signal Intelligence solves evidence lineage across many source families, this project is about the raw mechanics of collecting and enriching public listing data reliably at volume.

My role and ownership

- System architecture and module separation
- Browser acquisition and concurrent orchestration
- Sitemap/BFS website enrichment crawler
- Normalization, deduplication and scoring logic
- GUI, CLI, checkpointing and multi-format exports

Personal contribution

Architecture, acquisition, enrichment, scoring, reliability, GUI, CLI and exports.

Why the existing workflow needed a system

Listing pages expose data in inconsistent, semi-structured form. Company websites hide contact information in different places — footer, contact page, a link buried three clicks deep in a menu. Repeated searches produce duplicate entries that quietly inflate a dataset. And a raw list of business names tells you nothing about which ones are actually reachable or worth a call, which is the only question that matters commercially once the research is done.

Before and after

BEFORE

- Repeated manual searches and profile opening
- Inconsistent contact, location and reputation fields
- No systematic enrichment from company websites
- Duplicate records across related queries
- No transparent way to prioritize prospects

AFTER

- Concurrent acquisition with isolated worker contexts
- Normalized 38-field business records
- Bounded website crawl using sitemaps, JSON-LD and contact-page prioritization
- Deterministic deduplication and streaming export
- Separate data, contact and outreach-priority scores

Meaningful result

297 unique records transformed into a structured research dataset in the supplied run.

How the system works

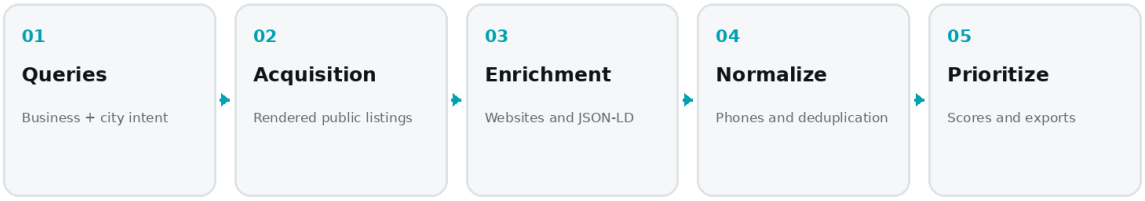
Acquisition runs on Playwright/Patchright for dynamic listing pages, orchestrated asynchronously with isolated browser contexts so multiple queries run concurrently without interfering with each other. A separate aiohttp crawler handles each business's own website, faster than a full browser when static HTML is enough, falling back to fuller rendering when it isn't.

Operational workflow

01	02	03	04	05
Queries	**Acquisition**	**Enrichment**	**Normalize**	**Prioritize**
Business + city intent	Rendered public listings	Websites and JSON-LD	Phones and deduplication	Scores and exports

SYSTEM ARCHITECTURE

Local Market Intelligence & Prospect Discovery Engine



Evidence-led architecture diagram · generated from the verified project workflow

Every record normalizes into a 38-field schema, gets deduplicated against everything already collected in the run, and is scored into a tier based on review count, rating and phone/website presence, so the output a reviewer opens is a prioritized shortlist, not a flat alphabetical table. Records stream to CSV as they're collected, with JSON and styled XLSX export layered on top, and a selector-health monitor compares field-fill rates run over run so a listing site changing its markup shows up as a measurable drop instead of silently corrupting the dataset. The operational sequence:

Architecture and decisions

The runtime combines Playwright or Patchright for dynamic listing pages with asynchronous Python orchestration across isolated browser contexts, and a separate lightweight aiohttp crawler for company websites — because most business websites don't need a full browser just to read a footer for an email address, and skipping that overhead matters at volume.

Critical engineering decisions

DECISION 01

Separate acquisition from intelligence

Extraction gathers evidence while scoring and classification live in a dedicated layer, making business logic inspectable.

DECISION 02

Bound website enrichment

Sitemap discovery and contact relevance improve coverage, while page caps and early stopping prevent wasteful full-site crawling.

DECISION 03

Stream results and preserve progress

Incremental writes and checkpoints reduce the cost of interruptions and make long research runs safer.

DECISION 04

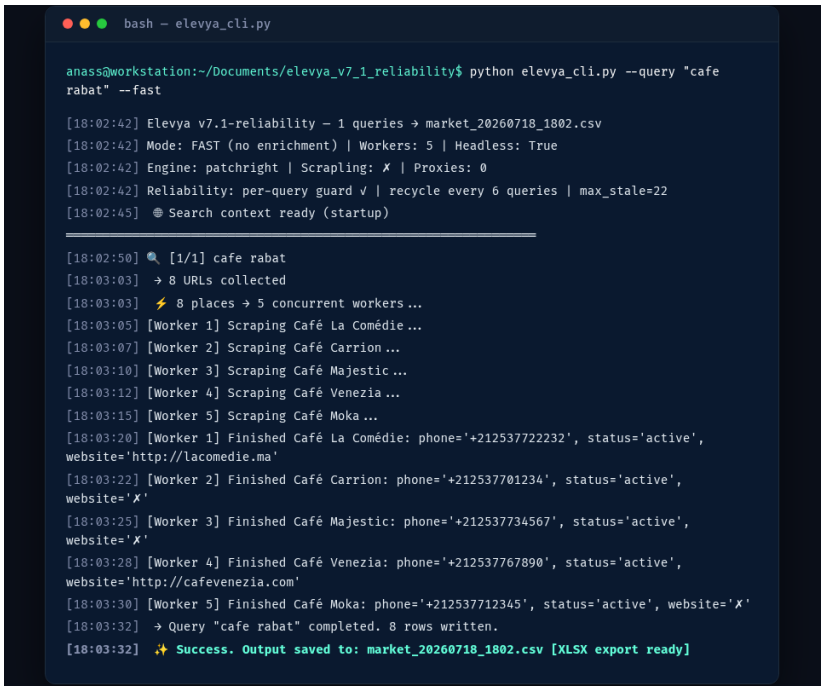
Expose GUI and CLI interfaces

The GUI supports operators; the CLI supports repeatable jobs, automation and technical inspection.

Technology stack

Python	**asyncio**	**Playwright / Patchright**	**aiohttp**	**Data scoring**	**CSV / JSON / XLSX**

Evidence 1 of 2



CLI run showing concurrent workers, successful extraction and export.

Evidence 2 of 2

SUPPLIED DEMONSTRATION DATASET · SANITIZED VIEW

NAME	CITY	PHONE	RATING	REVIEWS	WEBSITE URL	MARKET TIER	OUTREACH PRIORITY
La Bodega de Casablanca	Casablanca	available	4.1	1774	—	GOLDMINE	9
Moods Café-Restaurant	Casablanca	available	4.8	10810	available	ELITE	6
Pomo Dolce	Casablanca	available	4.6	1098	available	ELITE	9
Restaurant Dar El Kaid	Casablanca	available	4.3	1066	available	ELITE	7
Holy Brunch	Casablanca	available	4.6	1419	available	ELITE	7
le 7ème Ciel Casablanca	Casablanca	available	4.3	480	available	ELITE	8
La Brasserie Bavaroise	Casablanca	available	4.4	814	—	GOLDMINE	9
Bliss Rooftop	Casablanca	available	4.4	1152	available	ELITE	7

Sanitized preview of the supplied 297-record dataset.

Value, proof and honest limitations

Operational impact

The supplied Casablanca run turned five search queries into 297 unique, normalized business records automatically: 95.6% with a working phone number, 98.0% with coordinates, 49.2% with a website, 14.1% with an email — the kind of coverage that turns a research task measured in days into one measured in minutes.

Verified evidence

- CLI capture shows five concurrent workers and structured export
- Supplied dataset contains 297 rows and 38 fields
- Source separates GUI, CLI, engine, crawler and intelligence modules
- CSV, JSON and styled XLSX exports
- Selector-health monitoring and resume/checkpoint support

Current limitations

- 137 supplied rows contain an incorrect category value caused by a selector fallback.
- 289 descriptions match the address field, indicating an extraction-quality defect.
- 188 secondary-phone values duplicate the primary phone; some social links are tracking URLs.
- WhatsApp availability is estimated, not individually verified.
- A stronger populated GUI capture is still needed.

Current status

STATUS	TEAM	DEMO	SOURCE
Working technical demonstration · Quality hardening	**Solo engineering project**	**Controlled demo planned**	**Private repository**

WHY THIS PROJECT MATTERS

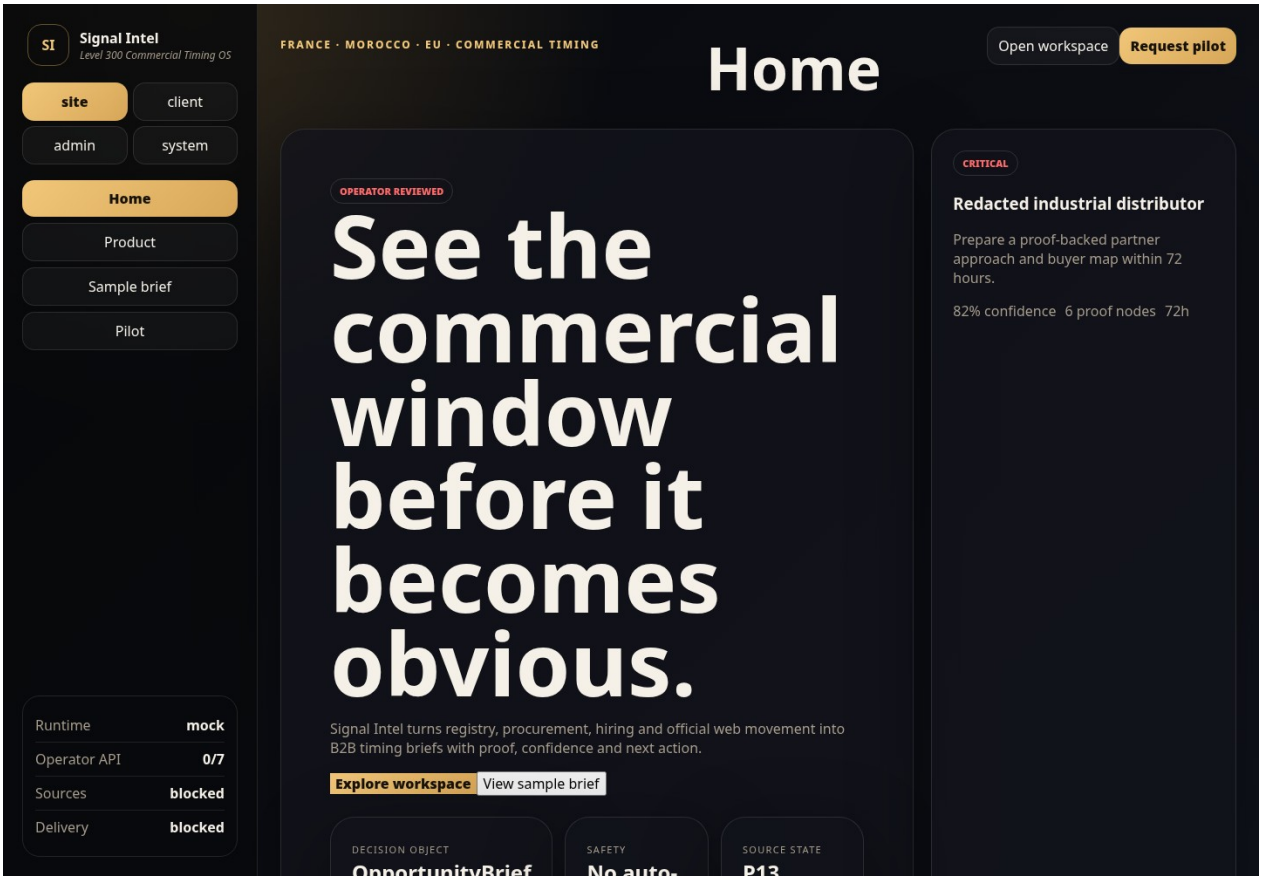
297 unique records transformed into a structured research dataset in the supplied run.

This case study is designed to be inspectable: strong claims are connected to screenshots, source evidence or explicitly stated limitations.

Transforming fragmented market signals into evidence-backed commercial dossiers

A multi-source intelligence architecture connecting acquisition, evidence lineage, qualification, human review gates and client delivery.

STATUS	TEAM	DEMO	SOURCE
Advanced platform · Controlled validation	**Founder-led platform build**	**Protected validation environment planned**	**Private repository**



Primary visual evidence

ANASS BENAMEUR · FULL-STACK PRODUCT ENGINEER

The product, the problem and the proof

A commercial opportunity is almost never one clean data point — it's a registry filing, a hiring pattern, a procurement notice and a timing signal that only mean something together. Elevya Signal Intelligence is the platform I designed to collect those fragments across source families, keep the evidence lineage of where each piece came from, run them through pattern detection and qualification, and stop before a human reviewer decides whether the resulting brief is good enough to send.

18	37	40+	Human gate
COMPONENTS IN ARCHITECTURE AUDIT	OFFLINE TESTS PASSED IN SMOKE REPORT	ORDERED MIGRATIONS	REVIEW BEFORE DELIVERY

Context

Multiple source families — registries, job boards, procurement portals, company sites — a canonical data layer, pattern detection, watchlists, briefing generation, production-readiness tooling, and a client-facing workspace. The captured product interface runs in explicit mock mode; live sources and delivery are currently blocked by design until the readiness gates pass. Its acquisition layer for job-market signals reuses the same adapter architecture as the standalone ATS Intelligence Engine in this portfolio, extended here with registry, procurement and hiring-velocity sources feeding one canonical model.

My role and ownership

- Platform and source-ecosystem architecture
- Canonical schemas and migration ordering
- Quality gates and production-readiness harnesses
- Evidence, pattern and briefing workflow design
- Client workspace and opportunity-brief interface

Personal contribution

Architecture, data model, evidence workflow, gates, readiness tooling and product interface.

Why the existing workflow needed a system

Without lineage, an automated alert is just noise with a timestamp — there is no way to tell whether it's corroborated by a second source, whether the underlying data is stale, or whether it's safe to put in front of a client. Independent source scripts with inconsistent state, signals with no shared identity, and no clear separation between observation, interpretation and delivery meant client output depended on manual assembly and production readiness was nearly impossible to verify honestly.

Before and after

BEFORE

- Independent source scripts with inconsistent state
- Signals without shared identity or evidence lineage
- No clear separation between observation, interpretation and delivery
- Client output dependent on manual assembly
- Production readiness difficult to verify

AFTER

- Source manifest and ordered migrations
- Canonical entities, events and lineage records
- Pattern, validation, watchlist and briefing layers
- Operator review checklist before delivery
- Preflight, smoke, audit and soak commands for readiness evidence

Meaningful result

A traceable path from multi-source evidence to a human-reviewed opportunity brief.

How the system works

Raw signals arrive from source adapters into a canonical schema, each one carrying its evidence trail forward rather than collapsing into an opaque number. Pattern detection looks for corroboration across sources — a hiring spike plus a registry change plus a procurement notice is a categorically different signal than any one of those alone, and the system is built to treat it that way.

Operational workflow

01	02	03	04	05
Sources	**Evidence**	**Qualification**	**Dossier**	**Gate**
Registry, hiring, procurement	Canonical events and lineage	Patterns and confidence	Opportunity brief	Human review and delivery

SYSTEM ARCHITECTURE

Elevya Signal Intelligence Platform

Evidence-led architecture diagram · generated from the verified project workflow

Anything that clears qualification lands in a client workspace as a brief with confidence, supporting evidence and a range, sitting in an explicit review state until an operator signs off — the system is built to refuse to auto-send, on purpose. A control CLI, ordered PostgreSQL migrations, scheduler support and a readiness-audit harness sit underneath: one recorded run passed the quality gate, 37 offline tests and a migration dry-run, then correctly failed at the live-connectivity stage because DATABASE_URL and live source credentials weren't present — the harness refusing to mark an incomplete environment production-ready, exactly as it should.

Architecture and decisions

The production bundle includes a control CLI, ordered PostgreSQL migrations, scheduler support, architecture audit tooling and a readiness-audit harness. The job-market acquisition layer inside this platform shares its adapter architecture directly with the standalone ATS Intelligence Engine case study in this portfolio — built once, proven independently, and extended here with the registry, procurement and hiring-velocity layers that turn a normalized job feed into a corroborated commercial signal. That relationship is stated here on purpose.

Critical engineering decisions

DECISION 01

Preserve evidence lineage

Signals are commercially credible only when the source, timing and transformation path remain inspectable.

DECISION 02

Separate detection from delivery

Pattern and qualification layers can produce a candidate brief, but external delivery stays review-gated.

DECISION 03

Test production readiness explicitly

Preflight, smoke, migration, benchmark and soak commands create evidence instead of relying on a deployment label.

DECISION 04

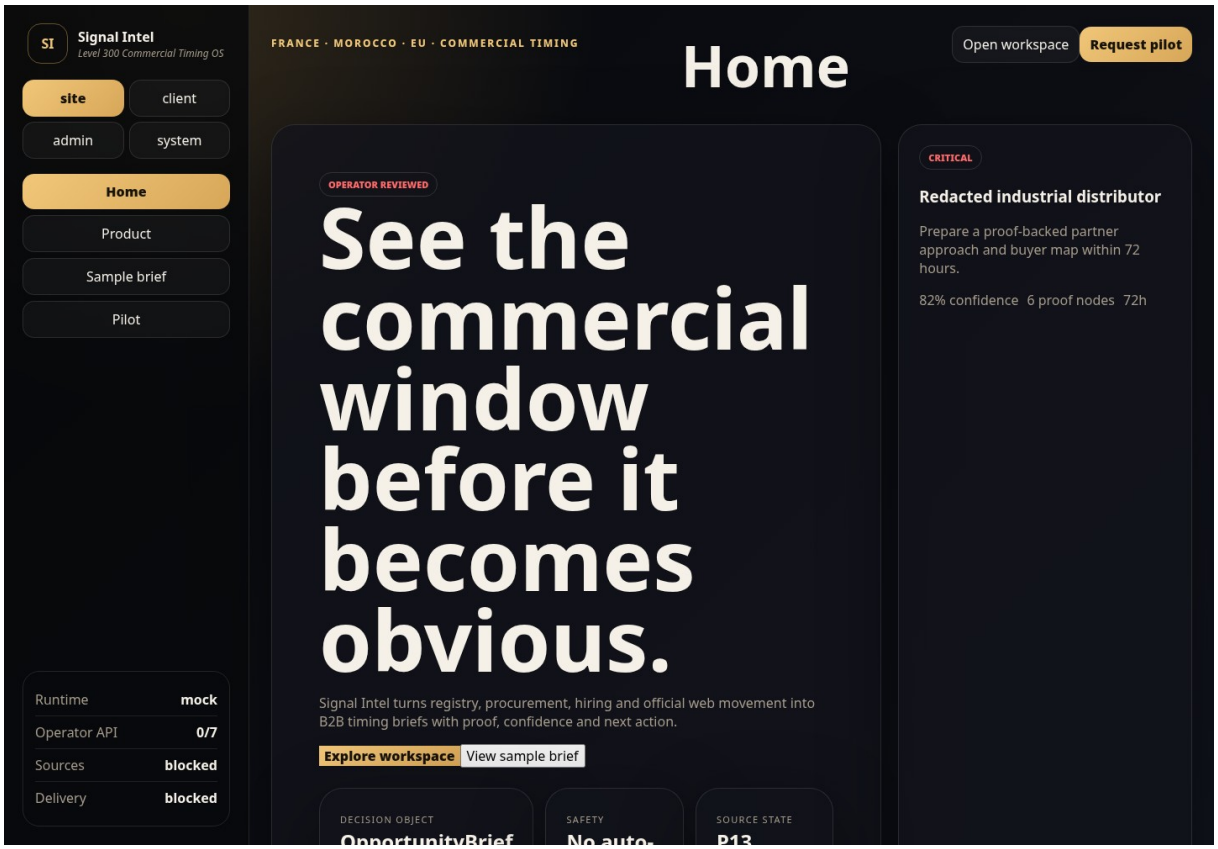
Expose limitations in the UI

The captured build visibly reports mock runtime, blocked sources and blocked delivery rather than pretending to be live.

Technology stack

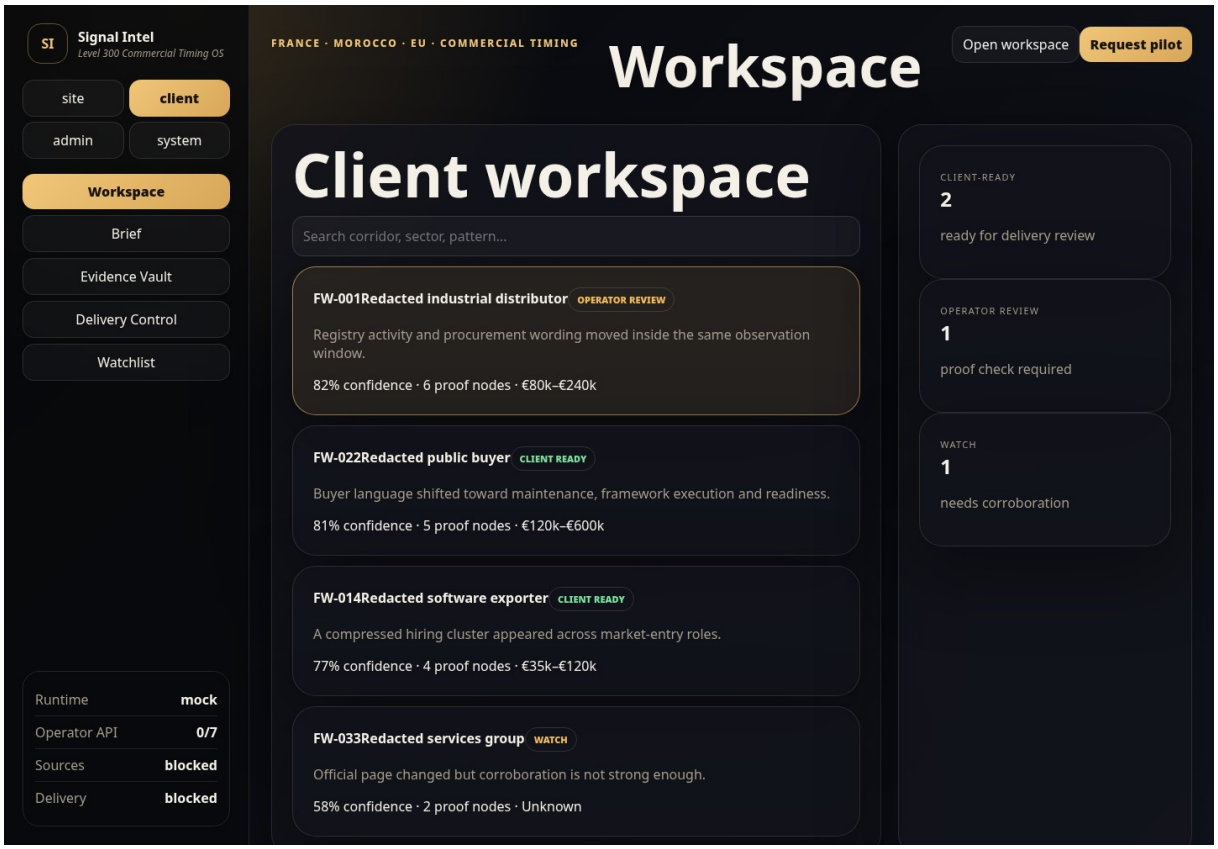
Python	PostgreSQL	Source adapters	Pattern engine	Evidence lineage	React product UI

Evidence 1 of 4



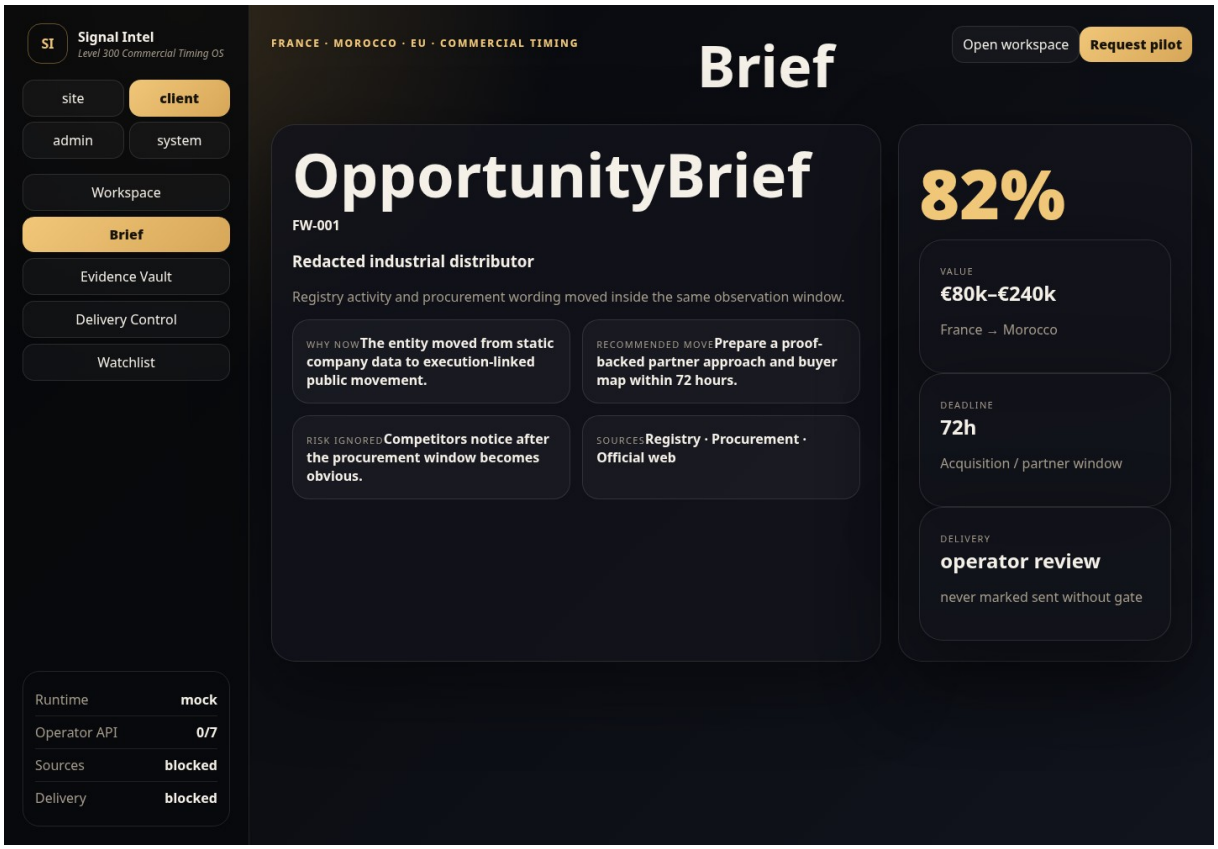
Product shell positioning the platform around commercial timing.

Evidence 2 of 4



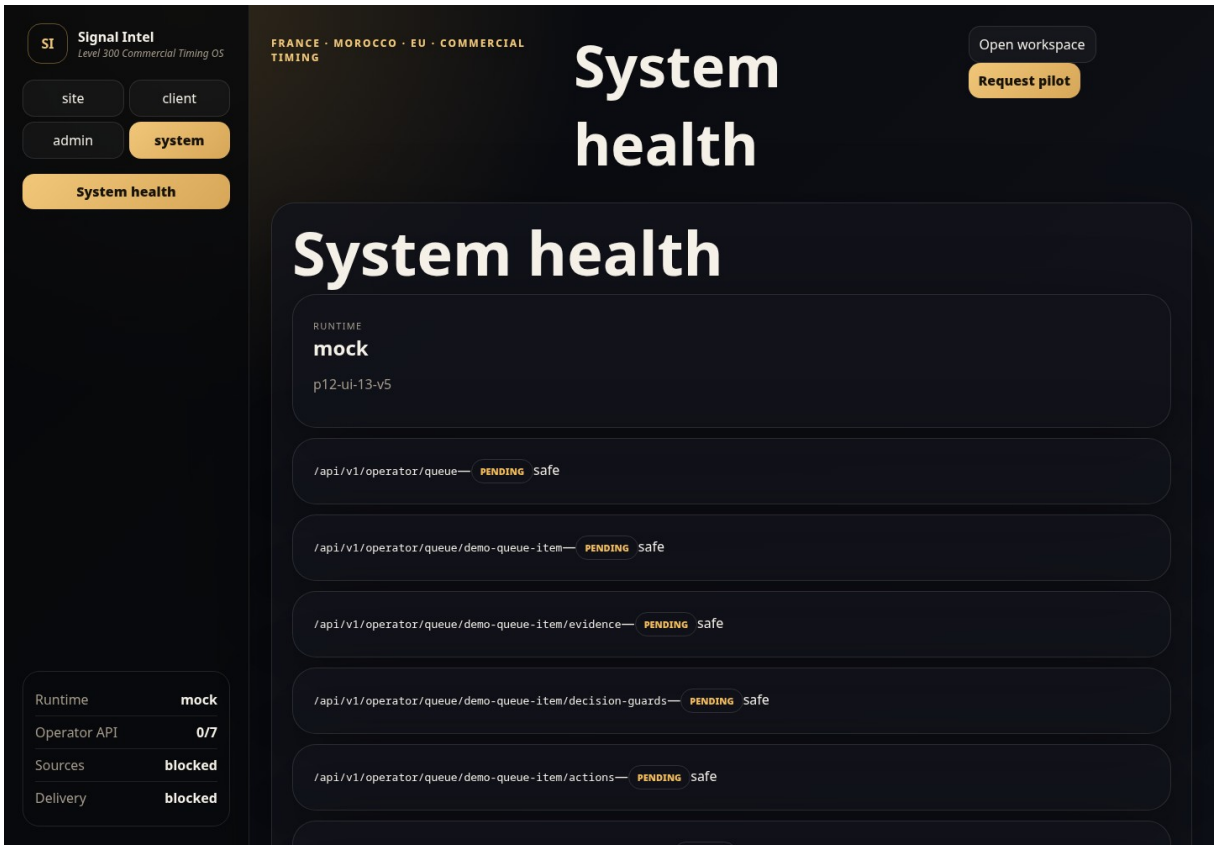
Client workspace listing opportunities and review states.

Evidence 3 of 4



Synthetic opportunity brief with confidence, range and operator-review state.

Evidence 4 of 4



System-health view explicitly showing mock mode and blocked dependencies.

Value, proof and honest limitations

Operational impact

The platform establishes a coherent path from fragmented evidence to a structured opportunity brief. This is the clearest demonstration in the portfolio of designing for trust under uncertainty: a system that would rather show "blocked, missing credentials" than fabricate a green status.

Verified evidence

- Architecture audit covers 18 system components
- Quality gate passed compile, AST, manifest and contract checks
- Offline smoke step passed 37 tests
- Ordered migration dry-run completed successfully
- UI captures show workspace, brief, system health and responsive shell

Current limitations

- The captured UI is a product shell running in mock mode; sources and delivery were blocked.
- A recorded preflight failed because database connectivity and live source credentials were absent.
- Architecture audit results show uneven maturity across modules.
- Several components need consistent caching, raw-artifact archives, metrics or local replay tests.
- Do not describe the platform as stable production until live smoke, benchmark and soak reports pass.

Current status

STATUS	TEAM	DEMO	SOURCE
Advanced platform · Controlled validation	**Founder-led platform build**	**Protected validation environment planned**	**Private repository**

WHY THIS PROJECT MATTERS

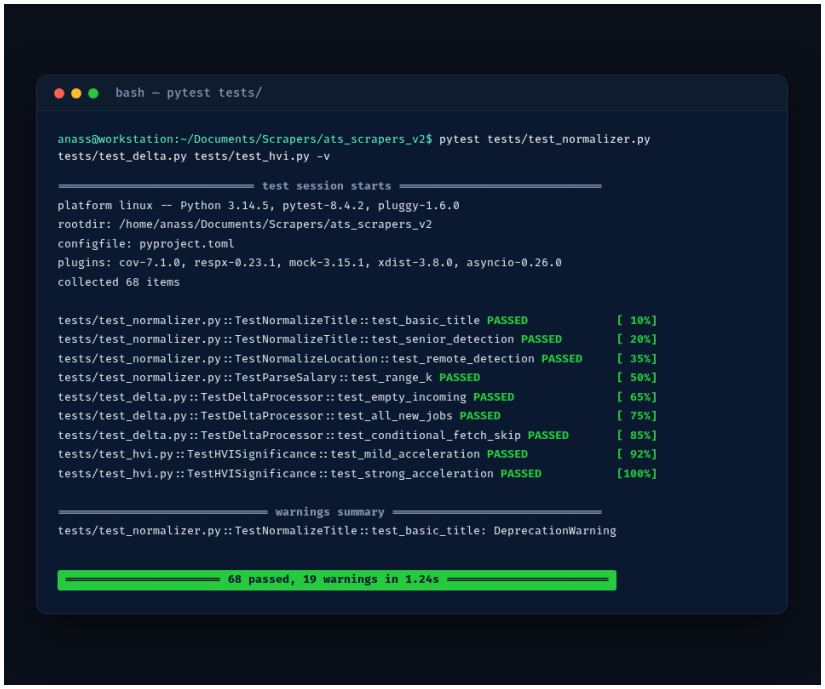
A traceable path from multi-source evidence to a human-reviewed opportunity brief.

This case study is designed to be inspectable: strong claims are connected to screenshots, source evidence or explicitly stated limitations.

Building a multi-source hiring intelligence engine with canonical normalization and change tracking

A Python acquisition layer that converts heterogeneous applicant-tracking systems into dependable, comparable job-market signals.

STATUS	TEAM	DEMO	SOURCE
Beta · Source reviewed · Selected tests passing	Solo engineering project	Fixture-based technical demo planned	Private repository



Primary visual evidence

ANASS BENAMEUR · FULL-STACK PRODUCT ENGINEER

The product, the problem and the proof

Every applicant-tracking system represents the same idea — a job posting — differently: different APIs, different HTML, different pagination, different field names for the same concept. This engine puts ten of them behind a shared adapter interface, converts everything into one canonical job model, and computes the delta between snapshots rather than re-ingesting the same postings as new data every run.

10	24	68	1.24 s
ADAPTER IMPLEMENTATIONS	TEST MODULES	TESTS PASSED IN CAPTURED RUN	CAPTURED SELECTED-SUITE RUNTIME

Context

Structured as a proper Python package — Python 3.11+, database schemas, orchestration scripts, observability hooks — with dedicated modules for rate limiting, proxy health, normalization, revision history and hiring-velocity scoring, aimed at recurring monitoring rather than a one-off export. It is the acquisition core that also powers the job-market side of the larger Elevya Signal Intelligence platform elsewhere in this portfolio, built first here as a standalone, reusable package.

My role and ownership

- Adapter and registry architecture
- Normalization and delta processing
- Rate limiting, retries and proxy-health controls
- PostgreSQL schemas and revision history
- Automated tests for core logic and sources

Personal contribution

Adapters, registry, canonical model, state comparison, revisions, network controls and tests.

Why the existing workflow needed a system

A title with a requisition code baked in, a location string that mixes city and remote-work signals in no consistent order, a salary range written as free text, a contract type spelled four different ways across four platforms — none of that is usable for comparison until it's normalized. And without snapshot-to-snapshot comparison, "new data" is really just the same postings re-ingested on every run.

Before and after

BEFORE	AFTER
• One-off source-specific scripts	• Adapter registry with source-specific clients
• Different field names and types per ATS	• Canonical title, location, contract, salary and function fields
• Repeated processing when boards have not changed	• Conditional fetch and rate-limit controls where supported
• No durable record of added, changed or removed jobs	• Delta and revision layers for lifecycle tracking
• No company-level hiring momentum signal	• High-Velocity Indicator for acceleration classification

Meaningful result

Ten ATS integrations converge on one canonical job and change model.

How the system works

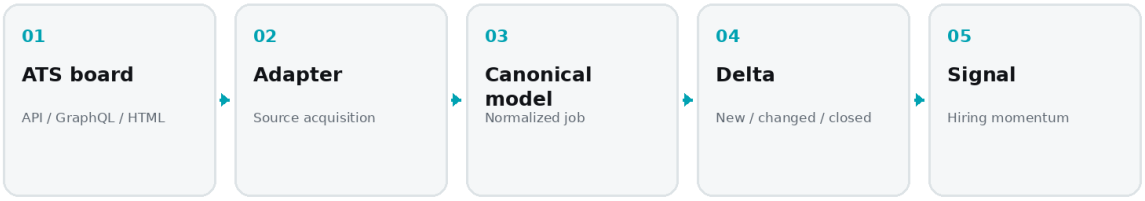
Ten source adapters — Greenhouse, Lever, Ashby, Workable, Personio, BambooHR, Recruitee, SmartRecruiters, Teamtailor and France Travail — sit behind one registry and shared orchestration, so adding an eleventh platform means writing one adapter, not touching the pipeline that runs all ten. Every record, regardless of source, lands in the same canonical schema.

Operational workflow

01	02	03	04	05
ATS board	**Adapter**	**Canonical model**	**Delta**	**Signal**
API / GraphQL / HTML	Source acquisition	Normalized job	New / changed / closed	Hiring momentum

SYSTEM ARCHITECTURE

ATS Intelligence Engine



Evidence-led architecture diagram · generated from the verified project workflow

A delta module compares each new snapshot against the last known state per source, so only genuine changes generate a revision record instead of re-writing everything every run. A hiring-velocity module turns the resulting change history into a company-level acceleration signal — the piece that makes ten normalized feeds worth more than ten spreadsheets, because it answers "who's hiring faster than last month," not just "who's hiring." The pipeline in sequence:

Architecture and decisions

The package targets Python 3.11+ and uses asyncio, httpx, psycopg and Selectolax. The adapter interface is strict enough that the orchestrator never needs to know which platform it's talking to, at the cost of upfront design time — a cost that pays back every time a new ATS gets added, since the tenth adapter took a fraction of the effort the first one did.

Critical engineering decisions

DECISION 01

Thin adapters, shared policy

Platform-specific parsing stays isolated while retries, rate limits, metrics and database behavior improve once for the whole engine.

DECISION 02

Normalize before downstream use

A canonical job record converts inconsistent titles, locations, contracts and salaries into fields that support analytics.

DECISION 03

Track revisions, not only current rows

Delta processing distinguishes new, changed and closed jobs and preserves evidence for hiring momentum.

DECISION 04

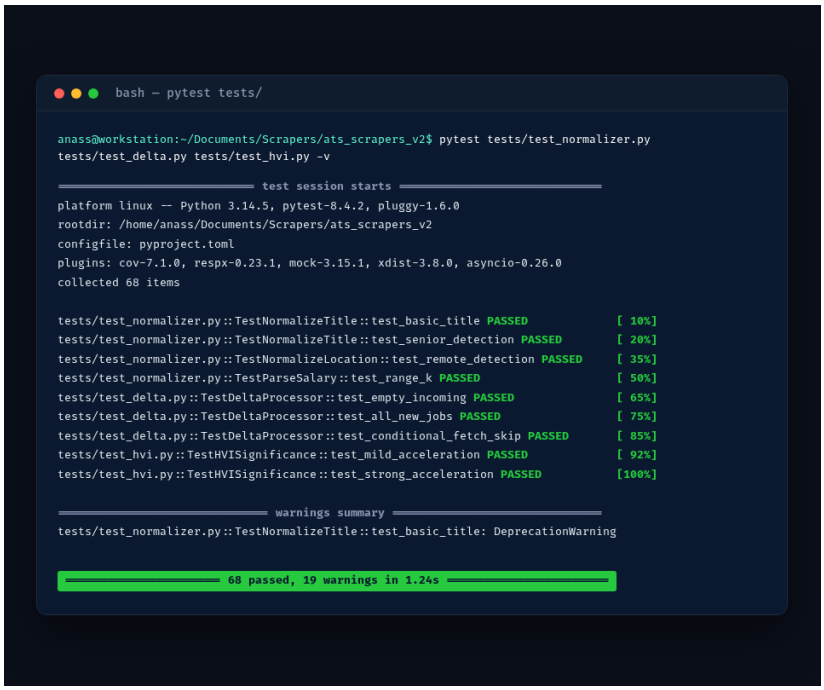
Test contracts and edge cases

The repository includes tests for infrastructure and adapters; the supplied capture shows 68 passing tests with 19 warnings.

Technology stack

Python 3.11+	**asyncio**	**httpx**	**PostgreSQL**	**Adapters**	**Delta / revisions**

Evidence 1 of 2

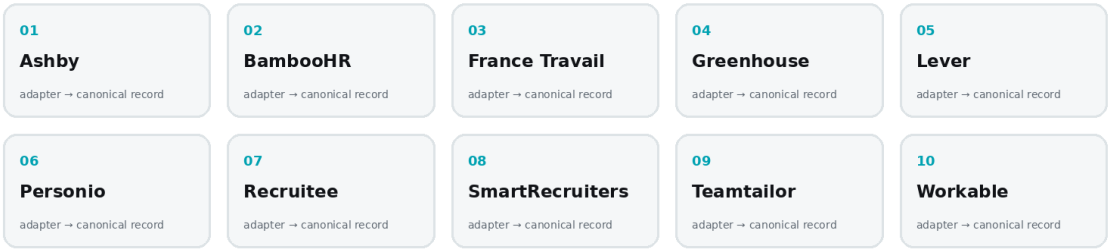


Selected automated suite: 68 tests passed with 19 warnings in 1.24 seconds.

Evidence 2 of 2

ADAPTER COVERAGE · SOURCE-REVIEWED

Ten source adapters behind one canonical job model



The public case study claims only adapters identified in the supplied repository.

Source-reviewed adapter coverage behind the canonical model.

Value, proof and honest limitations

Operational impact

Ten adapter source files, 24 test modules covering infrastructure and platform clients, and PostgreSQL schemas for core records, revisions and cache with optimized indexes — this is the acquisition layer that made the larger Signal Intelligence platform possible to build in the time it took, because the hardest normalization problem was already solved and tested here first.

Verified evidence

- 10 adapter source files identified
- 24 test modules covering infrastructure and platform clients
- Captured selected run: 68 passed, 19 warnings in 1.24 seconds
- PostgreSQL schemas for core records, revisions, cache and optimized indexes
- CLI scripts for sweeps, exports, HVI backfill and classifier training

Current limitations

- No live multi-platform sweep was captured for this portfolio package.
- The screenshot proves selected tests, not every production integration or credential path.
- An optimization percentage in source documentation lacks a benchmark artifact and is not repeated here.
- Warnings remain in the captured suite and should be removed or explicitly accepted.
- A public demo should use fixtures or cached responses.

Current status

STATUS	TEAM	DEMO	SOURCE
Beta · Source reviewed · Selected tests passing	**Solo engineering project**	**Fixture-based technical demo planned**	**Private repository**

WHY THIS PROJECT MATTERS

Ten ATS integrations converge on one canonical job and change model.

This case study is designed to be inspectable: strong claims are connected to screenshots, source evidence or explicitly stated limitations.

Centralizing institutional stock movements, threshold alerts and reporting

A Laravel inventory portal built from administrative requirements to centralize products, movements, low-stock alerts and formal reporting.

STATUS	TEAM	DEMO	SOURCE
Academic delivery · Visual evidence pending	Solo internship project	Demo pending source recovery	Source not currently published

EVIDENCE STATUS

Documented project · interface captures still required

VERIFIED

CV delivery record

Stakeholder analysis, Laravel implementation, alerts and exports are documented.

DOCUMENTED

Workflow and data model

Products, entries, exits, thresholds and reporting form the stated scope.

PENDING

Visual proof

Dashboard, movement history, alert and export screenshots are not supplied.

NEXT GATE

Source recovery

Run the portal with synthetic data, capture evidence and validate the final case study.

No interface has been fabricated. The portfolio discloses the evidence gap explicitly.

Evidence status map - no interface has been fabricated.

ANASS BENAMEUR · FULL-STACK PRODUCT ENGINEER

The product, the problem and the proof

During a final internship at Faculté Chariaa in Fès, I built a stock-management portal directly from requirements gathered with administrative stakeholders — the kind of engagement where continuity and traceability matter more than visual polish, and where the requirements-gathering itself is half the engineering work.

Feb–May 2025 INTERNSHIP PERIOD	**Full cycle** ANALYSIS TO DOCUMENTATION	**Thresholds** CONFIGURABLE STOCK ALERTS	**2 formats** EXCEL AND PDF REPORTING

Context

A public university environment, where a stock system has to survive staff turnover and produce reports that satisfy an administrative audit trail, not just look good in a demo. Between February and May 2025, I worked directly with administrative stakeholders to define the requirements before writing a line of code.

My role and ownership

- Stakeholder requirements analysis
- Merise data modelling
- Laravel backend and Blade interface
- Functional testing
- Documentation and handoff preparation

Personal contribution

Requirements, modelling, implementation, testing and documentation.

Why the existing workflow needed a system

Manual stock checking meant nobody could answer "how much do we actually have" without walking to the shelf. Low-stock situations were caught late because nothing was watching thresholds. And there was no single, reconstructable history of what entered and left inventory — exactly the kind of gap an administrative audit finds.

Before and after

BEFORE	AFTER
• Manual or fragmented product records	• Centralized product and stock records
• Stock levels checked reactively	• Entry and exit movement history
• Movement history difficult to reconstruct	• Automatic threshold-alert logic
• Reports assembled separately	• Excel and PDF exports
• Limited administrative visibility	• Documented institutional workflow

Meaningful result

Centralized movement history, threshold alerts and formal inventory reporting.

How the system works

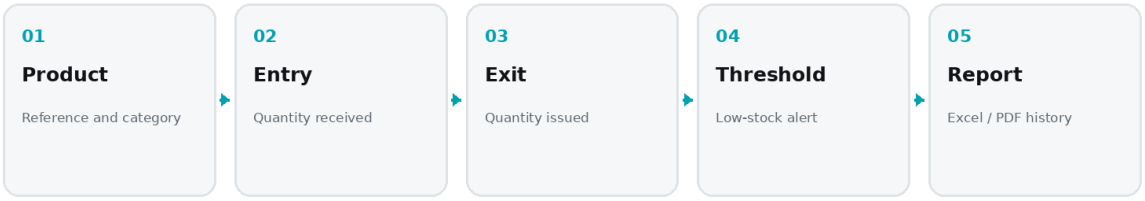
Products and movements share one relational core, modelled with Merise before a line of code was written and reviewed with the stakeholder group so the data model was agreed on, not just assumed. Configurable thresholds trigger alerts when stock drops below a set level per product, rather than relying on someone remembering to check manually.

Operational workflow

01	02	03	04	05
Product	**Entry**	**Exit**	**Threshold**	**Report**
Reference and category	Quantity received	Quantity issued	Low-stock alert	Excel / PDF history

SYSTEM ARCHITECTURE

Institutional Stock Management Portal



Evidence-led architecture diagram · generated from the verified project workflow

Every movement — entry, exit, adjustment — writes to a history that Excel and PDF reports generate from directly, so a printed report reflects the same data an administrator would see on screen, not a separately maintained summary that can quietly drift out of sync with reality. The lifecycle, step by step:

Architecture and decisions

The documented stack is Laravel, MySQL, Blade and MVC in a straightforward structure — the standard, unglamorous choice, deliberate for an institution that needs long-term maintainability over novelty. Requirements and data relationships were modelled with Merise before implementation began.

Critical engineering decisions

DECISION 01

Model movement history explicitly

Inventory must remain explainable through entries and exits rather than only a mutable quantity field.

DECISION 02

Thresholds as an operational control

Administrators need early warning instead of discovering shortages during a request.

DECISION 03

Formal export formats

Excel supports analysis while PDF supports stable administrative circulation.

DECISION 04

Document the complete lifecycle

Institutional adoption depends on requirements, testing and documentation as much as on code.

Technology stack

Laravel	PHP	MySQL	Blade	Merise	Excel / PDF

Evidence 1 of 1

EVIDENCE STATUS

Documented project · interface captures still required

VERIFIED	DOCUMENTED	PENDING	NEXT GATE
CV delivery record	**Workflow and data model**	**Visual proof**	**Source recovery**
Stakeholder analysis, Laravel implementation, alerts and exports are documented.	Products, entries, exits, thresholds and reporting form the stated scope.	Dashboard, movement history, alert and export screenshots are not supplied.	Run the portal with synthetic data, capture evidence and validate the final case study.

No interface has been fabricated. The portfolio discloses the evidence gap explicitly.

Evidence map explaining what is verified and what remains pending.

Value, proof and honest limitations

Operational impact

The delivered system gave the institution centralized product records, a reconstructable movement history, automatic low-stock alerts, and reports generated straight from the operational data instead of assembled separately — a complete requirements-to-documentation cycle over a February to May 2025 internship.

Verified evidence

- CV documents the internship, stack and complete delivery cycle
- Threshold alerts, movement history and Excel/PDF exports are consistently documented
- Requirements analysis and Merise modelling were part of the project
- Visual proof remains pending and is clearly disclosed

Current limitations

- No project screenshots were included in the supplied screenshot archive.
- No source archive or live runtime was supplied for independent verification.
- The project should not be presented as a final visual showcase until dashboard, movement, alert and export captures are added.
- No numerical impact is claimed without institutional measurement.

Current status

STATUS	TEAM	DEMO	SOURCE
Academic delivery · Visual evidence pending	**Solo internship project**	**Demo pending source recovery**	**Source not currently published**

WHY THIS PROJECT MATTERS

Centralized movement history, threshold alerts and formal inventory reporting.

This case study is designed to be inspectable: strong claims are connected to screenshots, source evidence or explicitly stated limitations.