MARVIN

Python Backend & API Portfolio

Selected work in Python, backend reliability, APIs, automation and practical AI tooling.

jacson981112@gmail.com

A quick introduction

I have spent more than 12 years programming, and the work I enjoy most is usually the part that needs some digging: an API that behaves differently than expected, a background job that fails halfway through, or a piece of existing code that needs to become easier to trust. I prefer small, clear changes over unnecessary rewrites, and I like leaving enough notes that the next person can understand what changed.

WHAT THIS PDF SHOWS

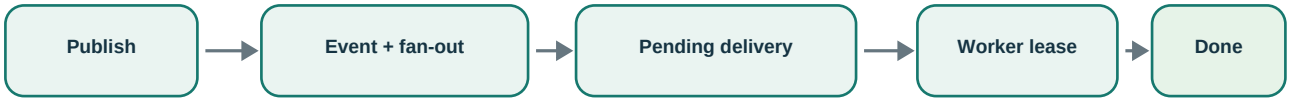
Three portfolio projects that reflect how I think about backend work: durable state and retries, tool-driven workflows with human review, and document retrieval with traceable evidence. The AI-oriented projects are recent portfolio prototypes, not client deployments.

Core stack	Python, FastAPI, REST APIs, SQLite/PostgreSQL, SQL, Git, Docker, Linux
Typical work	API integration, backend fixes, automation, data processing, debugging, reliability
Working style	Review the existing system first, reproduce the problem, make focused changes, test the result, and document the handover.

RelayVault

Durable event delivery without hiding the failure paths

RelayVault is a small SQLite-backed webhook outbox. I like this project because it deals with the unglamorous parts that often cause real backend problems: duplicate requests, competing workers, stale ownership, retries, and cleanup after something dies at the wrong time.



Retries, reconciliation and dead-letter handling stay explicit in the state machine.

What I focused on

- **Idempotent publishing:** replaying the same event does not quietly create duplicate work, and conflicting replays are rejected instead of being ignored.
- **Transactional fan-out:** the event and its subscriber deliveries are created together, so there is no half-published state to repair later.
- **Worker leases:** a worker gets an owner ID, a random token and an expiry. A stale worker cannot come back later and overwrite a newer result.
- **Recovery:** retry backoff, dead-letter handling, expired-lease reconciliation, retention and an audit command are part of the design rather than afterthoughts.

Runtime	Python 3.11+ standard library, SQLite
Tests	16/16 unit and concurrency tests passed in a fresh local run
Stress check	200 events, 4 publishers, 6 workers -> 200 completed, 0 audit errors
Scope	A delivery coordinator/outbox. It does not pretend to be a complete HTTP webhook platform.

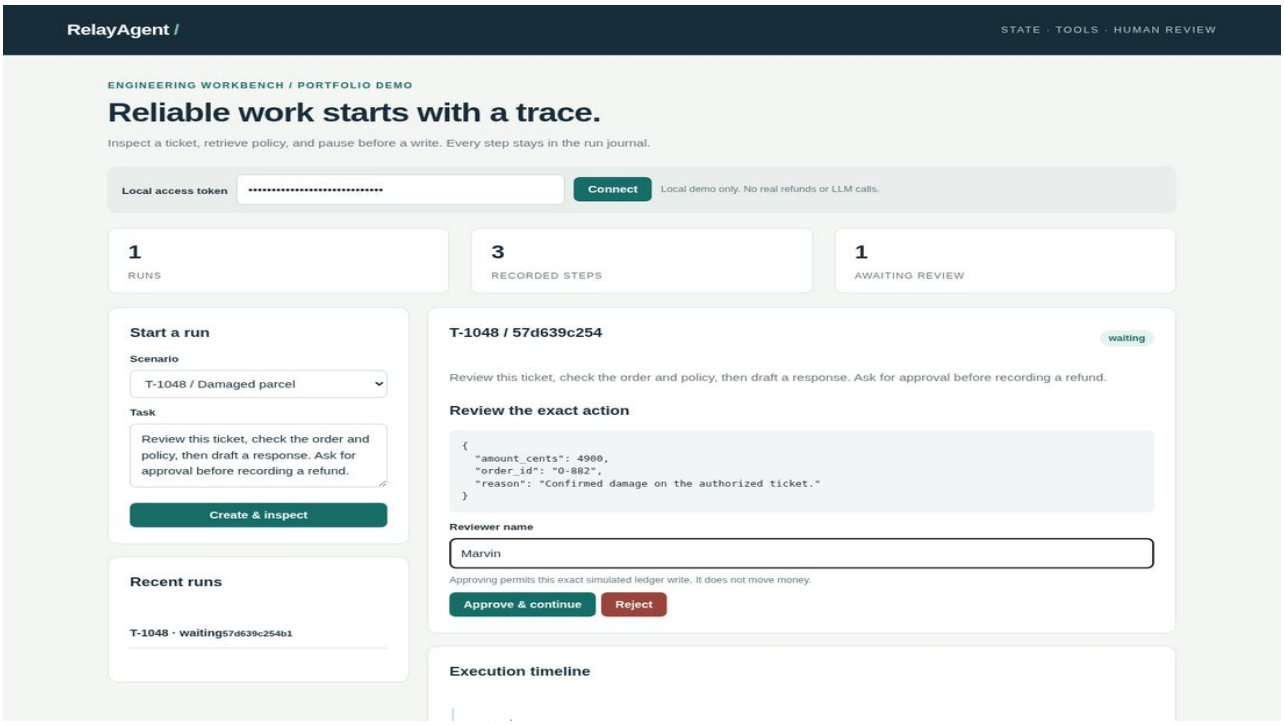
Why this matters for client work

A lot of backend bugs are really state-management bugs. This project is a good example of how I approach them: make ownership explicit, keep transitions inspectable, and write tests for the awkward timing cases instead of testing only the happy path.

RelayAgent

A stateful workflow that pauses before sensitive actions

RelayAgent is a recent portfolio prototype built around a simple support workflow. It looks up a ticket, checks order and policy data, records each tool decision, and pauses for human approval before a simulated refund is written to a local ledger. The interesting part for me was not the chat interface - it was making the run resumable and making approval apply to the exact action being reviewed.



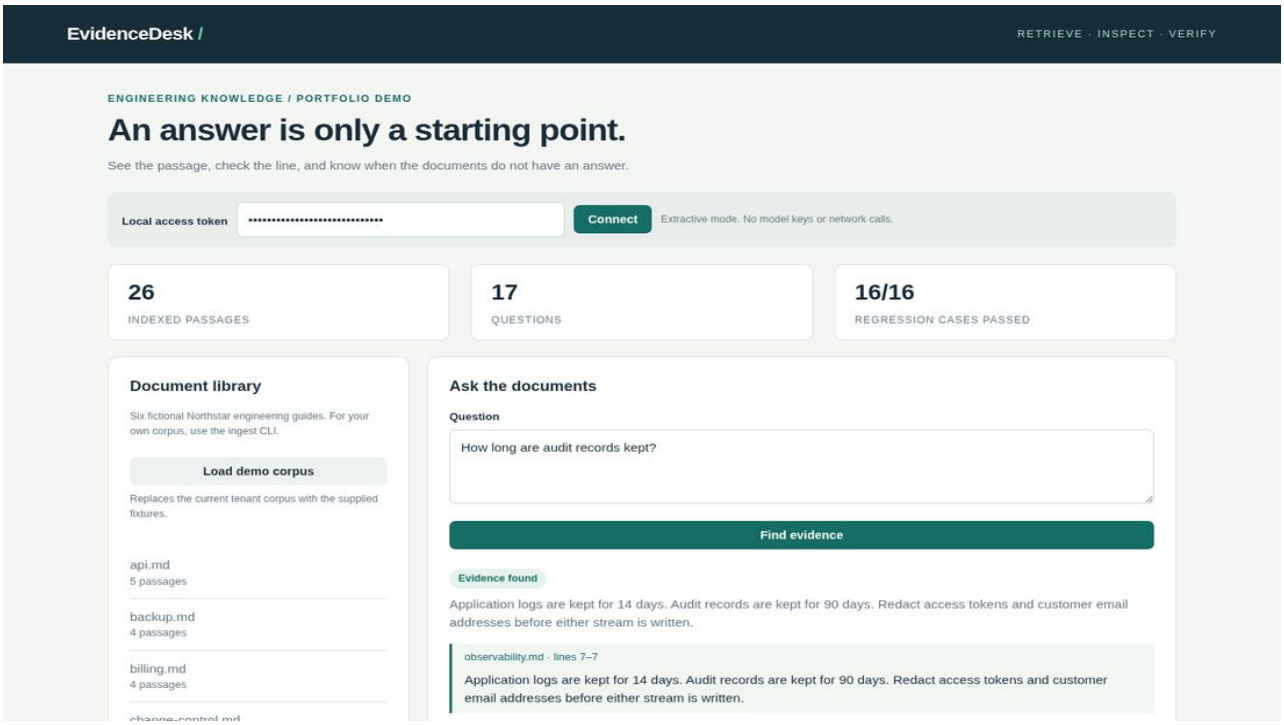
- **Persistent run journal:** decisions and tool results are stored so a run can be inspected and resumed.
- **Lease-based execution:** a worker that loses its lease can finish computing, but it cannot commit stale output.
- **Content-bound approval:** changing the tool or arguments after review invalidates the approval.
- **Practical testing:** the package includes tests for lease expiry, competing workers, crash replay, policy checks and local API behavior.

Validation	52 automated tests passed in the prepared portfolio package
Demo mode	Runs locally with fixture data and no paid model calls
Important limit	The refund tool is simulated. No real money movement or customer messages were performed.

EvidenceDesk

Document retrieval where the source stays visible

EvidenceDesk started from a simple preference: if a system answers a question from documents, I want to see the passage it used. The prototype indexes a small document set, retrieves likely evidence, returns source-line citations, and rejects answers whose quoted evidence cannot be traced back to the source.



- **Inspectable retrieval:** BM25 is the baseline, with an optional hybrid path that combines lexical and embedding rankings.
- **Source checks:** quoted evidence and source identifiers are validated instead of trusting a generated citation string.
- **Tenant-scoped data:** queries are separated by tenant at the data layer, while the documentation is clear that this is not a complete identity/authorization system.
- **Evaluation:** answerable and unsupported questions are tested separately so abstaining can be the correct result.

Validation	50 automated tests passed
Regression set	16/16 small synthetic extractive cases passed
Important limit	The regression set is small and synthetic. It is not a claim of production LLM accuracy.

How I work with client code

For freelance work, I try to make the first step boring in a good way: understand the problem before changing anything. That keeps a small bug fix from turning into an accidental rewrite and makes it easier to agree on what "done" means.

1. Reproduce	I ask for the smallest example that shows the issue, plus the error message, expected result and setup notes.
2. Trace	I follow the data and state through the relevant code path before deciding where the fix belongs.
3. Change	I keep the change focused. If I see a separate problem or a larger redesign opportunity, I call it out instead of quietly expanding the scope.
4. Check	I run the existing tests where available and add a focused regression check when it makes sense.
5. Handover	I send the updated code with a short explanation of what changed, how to run it, and anything the buyer should still know.

GOOD FIT FOR MY BANIGIG SERVICE

Small Python fixes, FastAPI/backend tasks, REST API integrations, automation scripts, CSV/JSON processing, database-related fixes and debugging existing code. If a task is outside my experience or too large for the selected package, I would rather say that before starting.

A note on portfolio scope

RelayVault is a standalone Python project. RelayAgent and EvidenceDesk are recent AI-assisted portfolio prototypes created to exercise backend reliability, APIs, testing and review workflows. They are shown here as engineering samples, not as claims of customer deployment or production scale.

If you have a Python, backend, API or automation problem, send me the smallest useful description of it. I will tell you what I can realistically take on before we start.