Fran Sammauro

Backend Developer · Systems & Low-Level Programming

Buenos Aires, Argentina · fs.sammauro@gmail.com · sammaurolabs.com · github.com/FranSammauro

SUMMARY

Backend-focused developer and 5th-year secondary school student with a strong interest in systems programming, cybersecurity, and distributed systems. Builds projects in Rust, C, Python, and TypeScript, with a focus on APIs, networking, Linux, and software architecture

TECHNICAL SKILLS

Languages: Rust, C, Python, TypeScript / JavaScript, Go, SQL

Backend: Axum, Tokio, Node.js, REST APIs, PostgreSQL, Prisma

Systems: Linux, Networking, Reverse Proxies, Distributed Systems, Job Queues

Tools: Git, Docker, GDB, QEMU, NASM, GCC, Make, Neovim

PROJECT EXPERIENCE

RAPTOR — Reverse Proxy / API Gateway

Aug 2026 – Rust

- **Routing:** Implemented longest-prefix-match routing with unit tests for deterministic upstream selection.
- **Proxying:** Built the request-forwarding layer using Hyper and Hyper-Util, including upstream URI rewriting, request-body forwarding, propagation of X-Request-Id, and response handling.
- **Observability:** Added structured request logging with method, path, upstream target, HTTP status, and request latency, providing the foundation for debugging and monitoring proxied traffic.
- **Configuration:** Designed a typed configuration system around raptor.yaml, defining routes, upstream targets, and logging settings with validation and structured error handling using thiserror.
- **Architecture:** Built the service around Rust, Axum, Hyper, and Tokio, separating configuration, routing, proxying, and application startup into independent modules.

DISTRIBUTED JOB QUEUE

Aug 2026 – Rust · In Development

- **Architecture:** Designing a distributed job-processing system inspired by systems such as Celery, BullMQ, and Sidekiq, focused on reliable asynchronous execution of background workloads
- **Job API:** Defined an HTTP-based interface for submitting typed jobs with structured payloads, establishing the foundation for producers to enqueue work independently from the workers that execute it.
- **Distributed Processing:** Designing the system around independent producers, workers, and a central queue, allowing jobs to be processed asynchronously and workers to scale independently from the API layer.
- **Reliability:** Planning mechanisms for job state tracking, retries, failure handling, and persistent job metadata so that temporary worker or infrastructure failures do not silently lose work.
- **Engineering Focus:** Using the project to explore distributed-systems concepts including message delivery, worker coordination, concurrency, backpressure, fault tolerance, and horizontal scalability.

Distributed Saga Orchestrator

May 2026 – Go

- **Orchestration:** Implemented the Saga pattern from scratch to coordinate multi-step distributed workflows, executing local transactions sequentially and triggering compensating actions when downstream operations fail.
- **Compensation:** Built automatic reverse-order compensation, allowing previously completed steps to be undone when a later step fails and preventing distributed workflows from remaining partially completed.
- **Persistence:** Designed a PostgreSQL-backed persistence layer using database/sql and raw SQL, storing saga instances and step execution history for durable state tracking and recovery.
- **Crash Recovery:** Implemented a journal-before-execute strategy that persists saga state transitions before executing each step, allowing interrupted workflows to resume without blindly re-running completed operations.
- **Architecture:** Structured the system into independent saga definitions, orchestration logic, transaction steps, and persistence layers, keeping the core orchestration engine decoupled from individual business operations.
- **Infrastructure:** Containerized PostgreSQL with Docker Compose and automated database migrations, providing a reproducible environment for running and testing the orchestrator.

EDUCATION

I.V.D Secondary School — In progress (5th Year).

Started 2023 · Continue.