Fraud Lens — System Design Documentation

Sujay S

May 14, 2025

Contents

1 Introduction

Fraud Lens is a containerised web application that predicts the likelihood of credit-card fraud, explains each prediction using Google Gemini 2.0 Flash, and provides live system and model monitoring dashboards. The system is composed of loosely-coupled micro-services deployed with Docker Compose. Users access the React front-end at `localhost:3002`; the UI connects to a FastAPI back-end which, in turn, orchestrates model inference, OTP-based registration, email notifications, and metrics collection.

Goals

- Accurate and low-latency fraud prediction via an MLflow model server.
- Clear explanations for regulatory compliance and user trust.
- Full MLOps observability (accuracy drift, service health, resource usage).
- Secure account management with OTP e-mail verification.
- Horizontal extensibility by keeping each concern in an isolated container.

2 Architecture Overview

2.1 Component Diagram

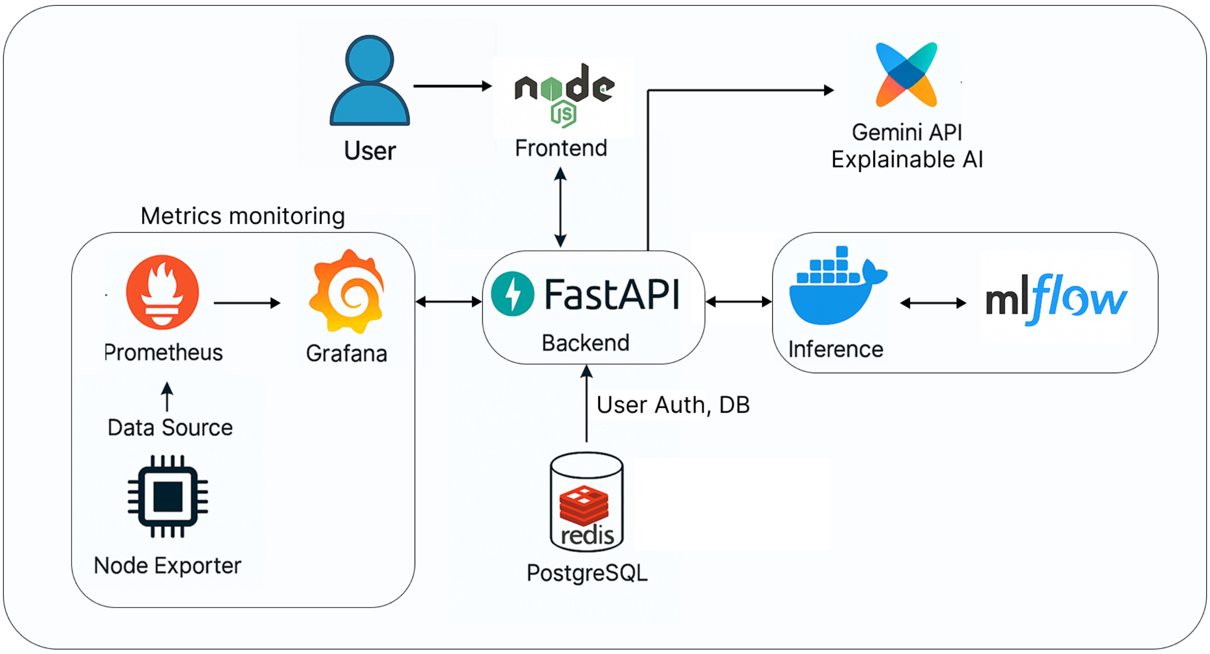

Figure 1: App framework

2.2 Component Descriptions

React Frontend Single-page application served by Nginx in the `frontend` container. Routes: `/login`, `/register`, `/predict`, `/explain`, `/metrics`. Embeds Grafana panels via iframes.

FastAPI Backend Core business logic and REST API. Validates input, interacts with model server, Redis, Postgres, and exposes Prometheus metrics with `prometheus_fastapi_instrumentator`.

MLflow Model Service Lightweight REST endpoint at `/invocations` exposing a LightGBM classifier. Container name: `fraud-inference` (port 8080).

Redis In-memory store for OTP codes and verification flags.

Postgres Relational database for persistent user accounts.

Prometheus Scrapes the backend and Node Exporter; stores time-series metrics.

Node Exporter Exposes host-level CPU/RAM/disk/network metrics.

Grafana Visualises Prometheus data; specific panels are embedded in the Metrics page.

3 High-Level Design

3.1 Design Choices and Rationale

- **Micro-services & Docker**: each concern is isolated, enabling independent scaling and CI pipelines.
- **FastAPI**: async support, automatic OpenAPI generation, and first-class Pydantic validation keep the backend type-safe.
- **MLflow Model Server**: decouples training from serving; any future model version can be hot-swapped by updating the registry tag.
- **Prometheus + Grafana**: de-facto standard for MLOps observability with minimal overhead.
- **Redis for OTP**: single-purpose cache with key expiry, perfect for short-lived verifications.
- **Postgres for Auth**: ACID semantics protect user data; SQL makes analytics easy.
- **Gemini 2.0 Flash**: provides near-real-time, cost-effective text generation suitable for per-prediction explanations.

3.2 Request / Response Flow

1. User logs in via JWT-based authentication.[1]
2. Front-end sends a `POST /predict` request; the backend forwards features to the MLflow service and returns the probability.
3. Feedback is optionally sent to `/feedback`, updating Prometheus counters (*TP, FP, TN, FN*).
4. The Explain page calls `/explain/prompt` to auto-fill the prompt, then `POST /explain`; backend calls Gemini and returns the explanation.
5. All REST calls and Node Exporter metrics are scraped by Prometheus; Grafana queries and embeds the panels.

[1] Password hashes stored with bcrypt.

4 Low-Level Design

4.1 API Endpoint Specification

Method / Path	Request	Response / Notes
POST /auth/send-otp	`{email}`	200 {code_sent}. Rate-limited; stores OTP in Redis.
POST /auth/verify-otp	`{email, otp}`	200 {verified}. Marks e-mail as verified.
POST /auth/register	See RegisterRequest	201 {registered}. Triggers welcome e-mail.
POST /auth/login	{email, password}	200 JWT token.
GET /clients/me	JWT hdr	200 user profile.
POST /predict	InputForm (JSON)	{fraud_probability, prediction}. Updates `latest` store. Prometheus counters inc. (total, success/fail).
GET /explain/latest	—	Last input + prediction. 404 if none.
GET /explain/prompt	—	Auto-generated prompt combining features + model output. 404 if none.
POST /explain	{prompt} or empty	{explanation}. Calls Gemini. Prometheus counters inc.
POST /feedback	{prediction, correct}	200 {status: ok}. Updates TP/FP/TN/FN counters.

Table 1: REST API reference

4.2 Data Models

User Table (Postgres)

- `id` (PK, serial)
- `email` (varchar, unique)
- `hashed_pw`
- `name`, `age`, `gender`, `country`
- `created_at` (timestamp with timezone, default=now)

Prometheus Metrics Key counters exported by the backend:

- `model_predict_`: total, success, failure, TP, FP, TN, FN *(label: client)*
- `model_explain_`: total, success, failure *(label: client)*

Input Features — abbreviated listing:

```
{
  "amt": 123.45,
  "lat": 37.77,
  "long": -122.41,
  "merch_lat": 37.81,
```

```
  "merch_long": -122.48,
  "tx_hour": 13,
  ...
}
```

5 Deployment & Operations

- Docker Compose file exposes the ports listed in Fig. **??** and mounts volumes for Postgres data and MLflow models.
- CI pipeline: build & test on push, publish container images to GHCR, auto-deploy via `docker compose pull & up -d` on the server.
- Alerts: Prometheus rules for high CPU (`> 90%` for 5 m), low memory (`< 500 MiB`), and model F1 drop (`< 0.85`).

6 Conclusion

The documented architecture demonstrates how a modern MLOps stack can be composed from open-source components to deliver trustworthy fraud detection with rich observability. Future work includes: automated dataset drift checks, canary deployments for new model versions, and RBAC for the metrics dashboard.