DOCUMENTATION

PYTHON DEVELOPMENT

WEEK-1

PROJECT NAME: Real-Time FinTech Fraud Detection System (SentinelStream)

Week 1 – Planning & Architecture

Organization -Zaalima Development Pvt. Ltd.

PITCHE ESHWAR-DOCUMENTATION

AKMAL-Created GitHub Repository. Created mockup admin dashboard.

SIVANANDANA-Designed the database architecture using a star schema.

1. Introduction

Week 1 focuses on **planning, system architecture, and foundational design** for the *SentinelStream* fraud detection platform. The objective of this phase is to establish a **scalable, production-grade architecture** capable of handling high-volume financial transactions with **low latency and high reliability**.

SentinelStream is designed for **large-scale Neo-banks**, where real-time decision-making is critical to prevent fraud while ensuring smooth transaction flow for legitimate users.

2. Use Case Overview (Production Scenario)

A Neo-bank processes **tens of thousands of credit card transactions per second** through external payment gateways. SentinelStream acts as an **intermediary fraud detection layer** between:

- External Payment Gateway
- Bank's Internal Ledger System

Core Responsibilities:

- Real-time transaction analysis
- Fraud risk scoring within **under 200 ms**
- Flagging suspicious transactions using:
  - Rule-based logic
  - Machine Learning predictions

3. High-Level System Architecture

SentinelStream follows a **microservices-based, API-first architecture**.

Architecture Components:

- **API Gateway**
  - Central entry point for all transaction requests
  - Handles routing, rate limiting, and authentication
- **Nano Services**
  - Individual services for:
    - Rule evaluation
    - ML scoring
    - Transaction logging
    - Notification handling

- **Supporting Components**
  - PostgreSQL (Immutable Ledger)
  - Redis (Rate limiting & session management)
  - Message queues for asynchronous processing

This architecture ensures:

- Horizontal scalability
- Fault isolation
- Low-latency processing

4. Feature Planning and Design (Week 1 Scope)

4.1 Basic Functions

Details

- User Transaction History API
- Current Balance Check
- Secure Transaction Logging

Implementation Focus

- RESTful API design
- Secure data storage practices
- Consistent API contracts using OpenAPI standards

4.2 Deep (Production) – Concurrency Handling

Details

- Use of **Idempotency Keys**
- Prevents:
  - Double charging
  - Duplicate transaction entries
- Handles retry scenarios due to API timeouts

Implementation Focus

- Middleware-based idempotency validation
- Logging and uniqueness checks for each request

4.3 Deep (Production) – Failure Recovery

Details

- Asynchronous Webhooks
- Non-blocking transaction status updates
- Notifications for:
  - Approved transactions
  - Declined transactions

Implementation Focus

- Celery / RabbitMQ for background processing
- Ensures main transaction thread remains fast and reliable

4.4 Deep (Production) – Rule Engine

Details

- Configurable rule-based fraud detection service
- Enables non-technical staff to define rules such as:
- *If transaction amount > RS 5000 AND location ≠ user home → Flag as RISK*

Implementation Focus

- Dedicated Python class structure
- Database-driven rule configuration
- Dynamic rule evaluation during runtime

5. Technology Stack (Finalized in Week 1)

Backend & Infrastructure

- **FastAPI** – High-performance API framework
- **PostgreSQL** – Immutable transaction ledger
- **Redis** – Rate limiting and session management

Supporting Libraries

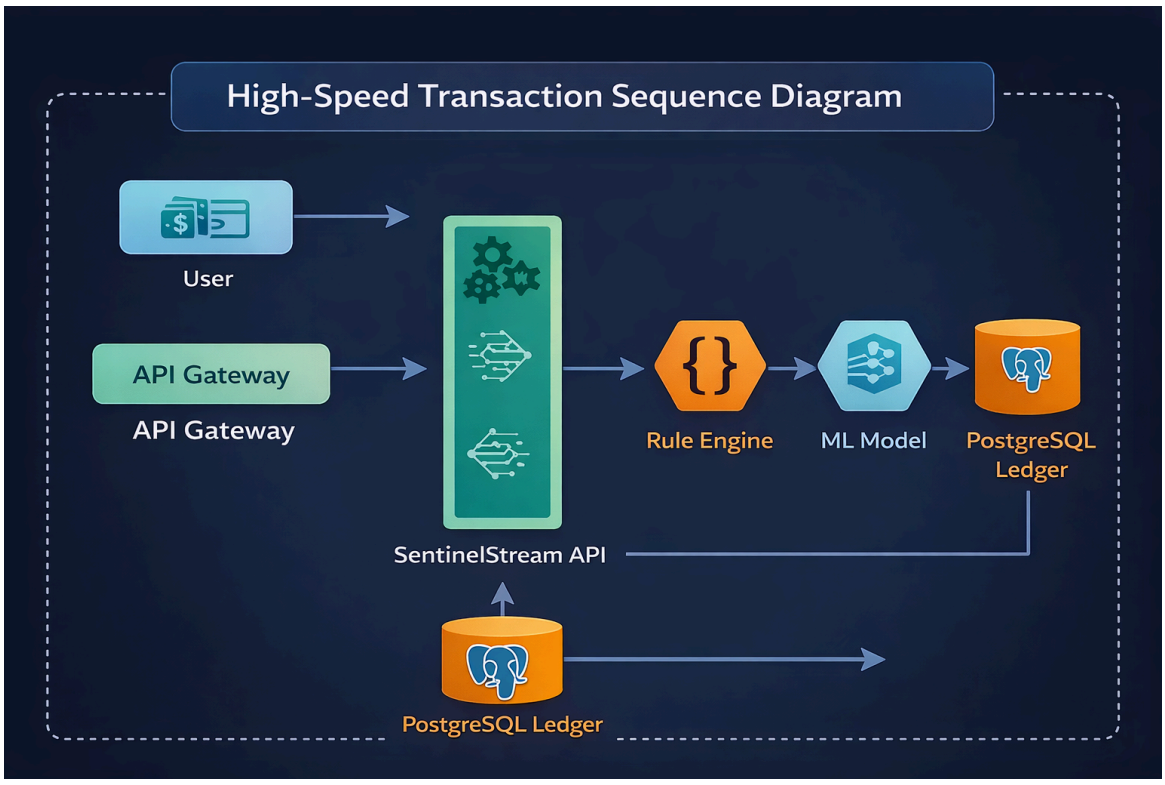
- **FastAPI-Limiter** – Burst traffic control
- **Pydantic** – Strict input data validation
- **Swagger / OpenAPI** – API contract definition

6. Diagrams Identified (Planning Phase)

During Week 1, required diagrams were identified for later implementation:

- **High-Speed Transaction Sequence Diagram**
  Flow:
- User → API Gateway → SentinelStream API →
- Rule Engine → ML Model → PostgreSQL Ledger

These diagrams will be implemented in later weeks.



7. Week 1 Execution Summary

Goal & Focus

Planning and Architecture Design

Key Tasks Completed

- Defined comprehensive **Software Requirement Specification (SRS)**
- Designed relational **Database Schema**
  - Star Schema for analytics efficiency
- Initialized **GitHub repository**
- Planned **CI/CD pipeline** using GitHub Actions
- Created **Admin Dashboard wireframes** (mock-ups)

Review & Deliverables

- Verified **3NF database normalization**
- Confirmed **API contract stability** using Swagger/OpenAPI

8. Learning Outcomes

By the end of Week 1:

- A clear production-ready architecture was established
- Core fraud detection strategies were finalized
- Scalability, reliability, and performance were addressed early
- The project moved from concept to structured system design

9. Conclusion

Week 1 successfully laid the **architectural and design foundation** for the SentinelStream fraud detection engine. With clearly defined APIs, database schemas, rule logic, and ML integration plans, the project is now ready to move into **implementation and development phases** in the upcoming weeks.

THANK YOU