Product Requirements & Specification Document

Project Name

FinSight - Real-Time Financial Analytics Engine

Overview

FinSight is a high-performance backend system for processing, analyzing, and visualizing large-scale financial transactions. It leverages Java Streams, multithreading, and Spring Boot for analytics, PostgreSQL for persistent storage, JWT-secured APIs, advanced error handling, and a Dockerized microservices architecture. The project is open-source and research-oriented.

---

1. Objectives

Objective	Description
Real-Time Analytics	Process and analyze financial transactions with minimal latency
Scalable Architecture	Support high throughput and horizontal scaling
Secure APIs	Ensure robust authentication and authorization using JWT
Fault Tolerance	Provide advanced error handling and resilience
Open-Source & Research	Facilitate extensibility and community contributions

---

2. Core Features

Feature	Description
Transaction Ingestion	Accept and validate high-volume transaction streams via RESTful API
Real-Time Processing	Analyze transactions using Java Streams and multithreading
Data Storage	Persist transactions and analytics results in PostgreSQL
Analytics API	Expose endpoints for querying analytics and visualizations
Security	JWT-based authentication and role-based authorization
Error Handling	Centralized, structured error reporting and logging
Containerization	Deploy as Dockerized microservices

---

3. Functional Requirements

3.1 Transaction Ingestion

- Accept JSON-formatted transaction data via REST API

- Validate schema and business rules
- Support batch and streaming ingestion

3.2 Real-Time Analytics

- Compute metrics (e.g., volume, frequency, anomalies) in real-time
- Use Java Streams and multithreading for parallel processing
- Support custom analytics modules (plug-in architecture)

3.3 Data Storage

- Store raw transactions and analytics results in PostgreSQL
- Ensure ACID compliance and efficient indexing

3.4 Analytics API

- Provide endpoints for:
  - Aggregated metrics (e.g., totals, averages)
  - Time-series data
  - Anomaly reports
- Support filtering, sorting, and pagination

3.5 Security

- JWT-based authentication for all endpoints
- Role-based access control (admin, analyst, viewer)
- Secure sensitive data in transit and at rest

3.6 Error Handling

- Centralized exception handling (Spring Boot @ControllerAdvice)
- Structured error responses (HTTP status, error code, message)
- Logging with correlation IDs

3.7 Containerization & Deployment

- Each service packaged as a Docker container
- Docker Compose for local orchestration
- Environment-based configuration

---

4. Non-Functional Requirements

Category	Specification
Performance	<100ms average API response time under typical load
Scalability	Horizontal scaling via stateless microservices
Reliability	99.9% uptime, graceful degradation on failure
Security	OWASP Top 10 compliance, encrypted secrets
Maintainability	Modular codebase, open-source documentation
Observability	Centralized logging, metrics, and health checks

5. System Architecture

```
[Client]
   |
[API Gateway (Spring Boot, JWT)]
   |
[Microservices: Ingestion, Analytics, Reporting]
   |
[PostgreSQL Database]
   |
[Monitoring & Logging]
```

- All services are Dockerized and communicate via REST/gRPC.
- API Gateway handles authentication and routing.

6. Technology Stack

Component	Technology
Language	Java 17+
Framework	Spring Boot
Database	PostgreSQL 14+
Build Tool	Maven
Containerization	Docker
Security	JWT, Spring Security
Orchestration	Docker Compose
Logging/Monitoring	Open-source stack (e.g., ELK, Prometheus)

7. API Specifications (Sample)

Transaction Ingestion

```
POST /api/v1/transactions
Authorization: Bearer <JWT>
Content-Type: application/json

{
  "transactionId": "string",
  "timestamp": "ISO8601",
  "amount": "decimal",
  "currency": "string",
  "accountId": "string"
}
```

Analytics Query

```
GET /api/v1/analytics/summary?from=2024-01-01&to=2024-01-31
Authorization: Bearer <JWT>
```

---

8. Open-Source & Research Considerations

- Codebase licensed under Apache 2.0
- Public documentation and contribution guidelines
- Modular design for research extensions

---

9. Milestones

Milestone	Target Date
MVP Architecture	Month 1
Core Analytics Engine	Month 2
API & Security	Month 3
Dockerization & CI/CD	Month 4
Open-Source Release	Month 5

---

10. Risks & Mitigations

Risk	Mitigation
High data volume	Use efficient streaming and batching
Security vulnerabilities	Regular audits, dependency updates
Performance bottlenecks	Profiling, horizontal scaling

---

11. Acceptance Criteria

- All core features implemented and tested
- Meets performance and security benchmarks
- Fully containerized and documented
- Open-source ready with contribution guidelines

End of Document