■ PRD: FinTech Data Infrastructure Dashboard

1. Overview

The FinTech Data Infrastructure Dashboard is a full-stack platform designed to ingest, store, analyze, and visualize financial market data. It enables users to explore assets, view historical prices, and monitor key indicators (e.g., SMA, EMA, RSI, volatility) in an interactive dashboard. The project demonstrates data engineering, backend systems design, and AI/analytics integration.

This is a portfolio-grade project for showcasing backend engineering, SQL optimization, system design, and full-stack development skills.

---

2. Objectives
- Build an end-to-end financial data pipeline that mimics production-grade infrastructure.
- Provide real-time insights on assets through analytics and indicators.
- Develop a scalable, testable backend API for asset and price data.
- Deliver a clean and interactive dashboard to visualize data.
- Showcase expertise in Python, SQL, FastAPI, React, and system design.

---

3. Users & Use Cases

Primary Users
- Portfolio analysts / students / developers wanting to explore assets and indicators.
- Recruiters/hiring managers reviewing technical portfolio projects.

Use Cases
1. Search for an asset (e.g., AAPL, BTC).
2. View historical OHLCV data stored in PostgreSQL.
3. Analyze computed indicators (SMA, EMA, RSI, returns, volatility).
4. Visualize trends with charts and tables.
5. (Stretch) Receive signals/alerts when thresholds are met (e.g., RSI < 30).

---

4. Features

Core Features
1. Data Ingestion
- Scheduled fetch of OHLCV data from market APIs.
- Deduplication & error handling.
- Logging of ingestion jobs.

2. Database Layer
- Tables: assets, prices, indicators, jobs, alerts.
- Constraints & indexes for optimized queries.

3. Analytics Layer
- Calculate SMA, EMA, RSI, daily returns, volatility.

- Store results in indicators table.

4. Backend API (FastAPI)
- GET /assets → list/search assets.
- GET /prices → historical OHLCV.
- GET /indicators → calculated metrics.
- GET /health, GET /metrics.

5. Frontend Dashboard (React + D3/Recharts)
- Home page: asset search & overview.
- Asset detail: chart of prices & indicators.
- Portfolio view: multi-asset performance comparison.

Stretch Features
- Alerts when thresholds are hit (e.g., RSI < 30).
- Prometheus metrics + Grafana dashboards.
- Authentication (JWT) for private API endpoints.

---

5. Technical Requirements

Backend
- Python 3.11+
- FastAPI for API services
- SQLAlchemy + Alembic for ORM + migrations
- PostgreSQL 15+ for storage
- APScheduler / Celery for scheduling jobs
- pytest for unit/integration testing

Frontend
- React + Vite
- Recharts/D3.js for visualization
- Material UI for layout

Infrastructure
- Docker Compose for local dev
- GitHub Actions for CI/CD
- Render/Fly.io (optional deploy)
- Vercel/Netlify for frontend deploy

---

6. Success Metrics
- ■ Data ingestion job completes successfully with no duplicates.
- ■ Indicators computed and stored for at least 2+ assets.
- ■ API responds within <200ms for common queries (indexed).
- ■ React dashboard displays charts from backend data.
- ■ Test coverage >70% for ingestion + API.
- ■ Project fully documented in README.

---

7. Risks & Mitigations
- API rate limits → Use caching & backoff logic.
- Large datasets → Add indexing & pagination.
- Deployment complexity → Use Docker Compose for consistent local runs.
- Resume readability → Keep documentation concise & include architecture diagram.

---

8. Deliverables
- Full backend code (Python + FastAPI).
- PostgreSQL schema + migrations.
- Data ingestion scripts with scheduler.
- React dashboard with charts.
- Tests + CI/CD pipeline.
- README with setup, architecture, and usage.