TRADING BOT PROJECT

1. OVERVIEW

This project is a demo trading bot built for learning and interview purposes.
It simulates trading using real market data from the Binance public API.
The system consists of a Java Spring Boot backend and a React frontend built with Vite.

2.TECHNOLOGY STACK

BACKEND:
- Java 17
- Spring Boot
- Spring Web (REST API)
- Spring JDBC (JdbcTemplate)
- PostgreSQL
- Spring Scheduler
- WebClient (for HTTP calls to Binance)

FRONTEND:
- React
- Vite (fast development server and build tool)
- Fetch API

DATABASE:
- PostgreSQL

3.PROJECT STRUCTURE (BACKEND)

com.adrian.tradingbot
api
- Controllers that expose REST endpoints

bot
- BotService: main decision logic
- SignalService: generates BUY / SELL / HOLD signals
- TradeService: executes simulated trades
- TradingScheduler: runs the bot automatically every few seconds

marketdata
- MarketDataService: fetches candles from Binance API

portfolio
- PortfolioService: reads portfolio and holdings from database

- PortfolioSnapshot: represents current portfolio state

persistence
- TradeRepository: database access using JdbcTemplate

model
- Candle: market candle data
- Trade: executed trade entity
- Holding: current asset holding

config
- WebClient configuration

resources/db
- schema.sql (database schema)

DATABASE STRUCTURE

Tables:

portfolio
- id (int, primary key)
- cash (numeric)

holdings
- symbol (varchar)
- qty (numeric)
- avg_price (numeric)

trades
- id (bigserial, primary key)
- ts (timestamp)
- symbol (varchar)
- side (BUY / SELL)
- qty (numeric)
- price (numeric)
- fee (numeric)
- pnl (numeric)

4.HOW THE BOT WORKS

Step 1:
MarketDataService requests the last N candles from Binance.

Step 2:
SignalService calculates:
- Fast Moving Average
- Slow Moving Average

Step 3:
If fast MA crosses above slow MA -> BUY
If fast MA crosses below slow MA -> SELL
Otherwise -> HOLD

Step 4:
TradeService checks the current portfolio:
- If BUY and no holding exists -> buy with 10 percent of cash
- If SELL and holding exists -> sell entire position

Step 5:
Trade is saved in the database and portfolio is updated.

5.TRAINING (BACKTESTING)

Training means simulating the strategy on historical data.
How it works:
- Load historical candles (for example 200 candles)
- Start from candle index where MA calculation is possible
- For each candle:
- Calculate signal
- Simulate trade
- Track portfolio value

This allows testing if the strategy makes profit on past data.

6.FRONTEND (REACT + VITE)

The frontend is a simple dashboard.
Why Vite:
- Very fast startup
- Simple configuration
- Modern standard for React projects

Frontend features:
- Show bot status
- Show portfolio (cash and holdings)
- Show trade history
- Start / pause bot
- Run training (backtest)

The frontend calls backend REST endpoints using HTTP fetch.

7.HOW TO RUN THE PROJECT

BACKEND:

1. Start PostgreSQL
2. Create database (example: trading_bot)
3. Update application.properties with DB credentials
4. Run schema.sql
5. Run Spring Boot application

The backend runs on:
http://localhost:8080

FRONTEND:

1. Go to frontend folder
2. Run:
```
npm install
```
3. Run:
```
npm run dev
```

Frontend runs on:
http://localhost:5173

8.FUTURE IMPROVEMENTS

- Kafka for signal events
- Real exchange integration
- Risk management
- Multiple strategies
- Charts and indicators
- Docker deployment