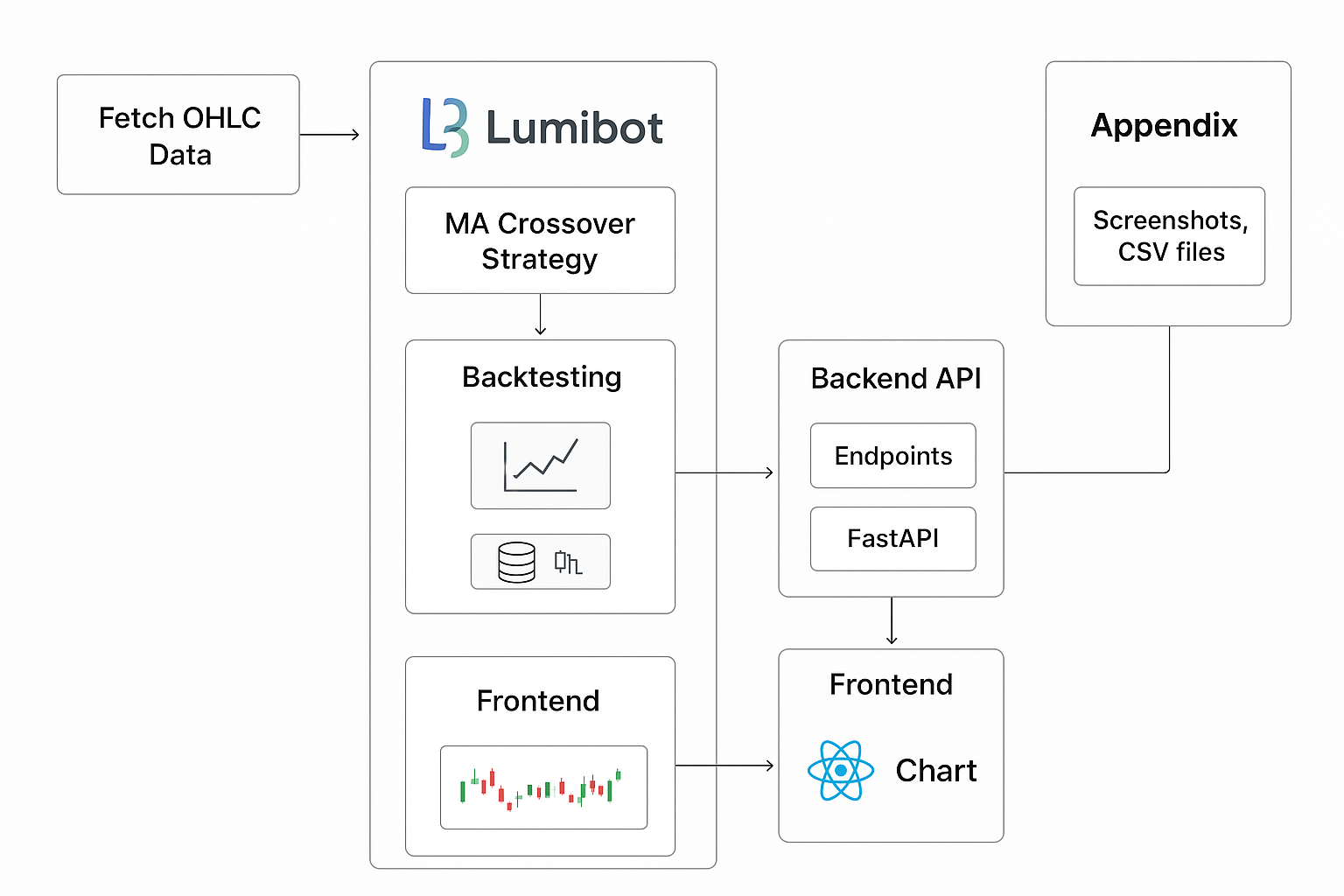
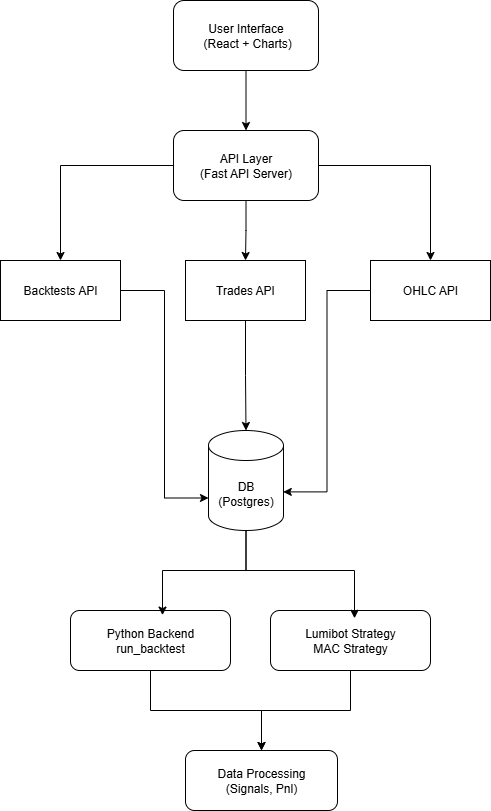
**MACrossover Trading Strategy Assignment**

# 1. Project Overview

This project demonstrates a Moving Average Crossover (MAC) trading strategy implemented using Python, Lumibot, and PostgreSQL, with visualization in React using Lightweight Charts.  
The goal is to:  
1. Backtest a trading strategy using historical OHLC data.  
2. Implement stop-loss and take-profit mechanisms.  
3. Store results in a structured PostgreSQL database for easy comparison.  
4. Display results and trades on an interactive frontend chart.

1. **Architecture:**

**  
  
  
  
  
  
  
  
Explanation:**

* **Frontend (React + Lightweight Charts)**: Displays OHLC and trade markers.
* **API Layer (FastAPI):** Serves backtest data, trades, and OHLC to frontend.
* **Database (PostgreSQL):** Stores all historical data, backtest results, trades.
* **Python Backend & Lumibot Strategy:** Executes backtests, generates signals, calculates performance metrics.
* **Data Processing:** Computes cumulative returns, Sharpe ratio, drawdowns.

# 3. Strategy Details

## 3.1 MAC Strategy Logic

The Simple Moving Average (SMA) Crossover strategy involves:  
- Short-term SMA: 20-day moving average  
- Long-term SMA: 50-day moving average  
  
**Trading signals:**

|  |  |
| --- | --- |
| **Condition** | **Action** |
| Short SMA crosses above Long SMA | BUY |
| Short SMA crosses below Long SMA | SELL |
| Position loss ≥ 1% | STOP-LOSS SELL |
| Position profit ≥ 50% | TAKE-PROFIT SELL |

## 3.2 Python Implementation

- Libraries Used: Pandas, NumPy, yfinance, SQLAlchemy, Lumibot.  
- Backtesting Logic:  
 - Fetch historical OHLC data using YahooDataBacktesting.  
 - Generate signals based on SMA crossover.  
 - Execute trades with stop-loss and take-profit.  
 - Track portfolio value, cumulative returns, and performance metrics.  
  
  
- Performance Metrics Calculated:  
 - Total Returns (%)  
 - Maximum Drawdown (%)  
 - Sharpe Ratio  
 - Number of Trades

# 4. Database Schema

## 4.1 Tables

“backtests” – Summary of each backtest  
“trades” – Individual trades executed by the strategy  
“ohlc” – Historical price data used for backtesting  
 **Schema Diagram:**  
**backtests:**  
  
| id | name | symbol | start\_date | end\_date | short\_window | long\_window | stop\_loss\_pct | take\_profit\_pct | initial\_capital | total\_return\_pct | max\_drawdown\_pct | sharpe | n\_trades | created\_at |  
  
**trades:**  
| id | backtest\_id | timestamp | side | price | size | pnl | cumulative\_return\_pct |  
  
**ohlc:**  
| id | symbol | timestamp | open | high | low | close | volume |

## 4.2 Design Motivation

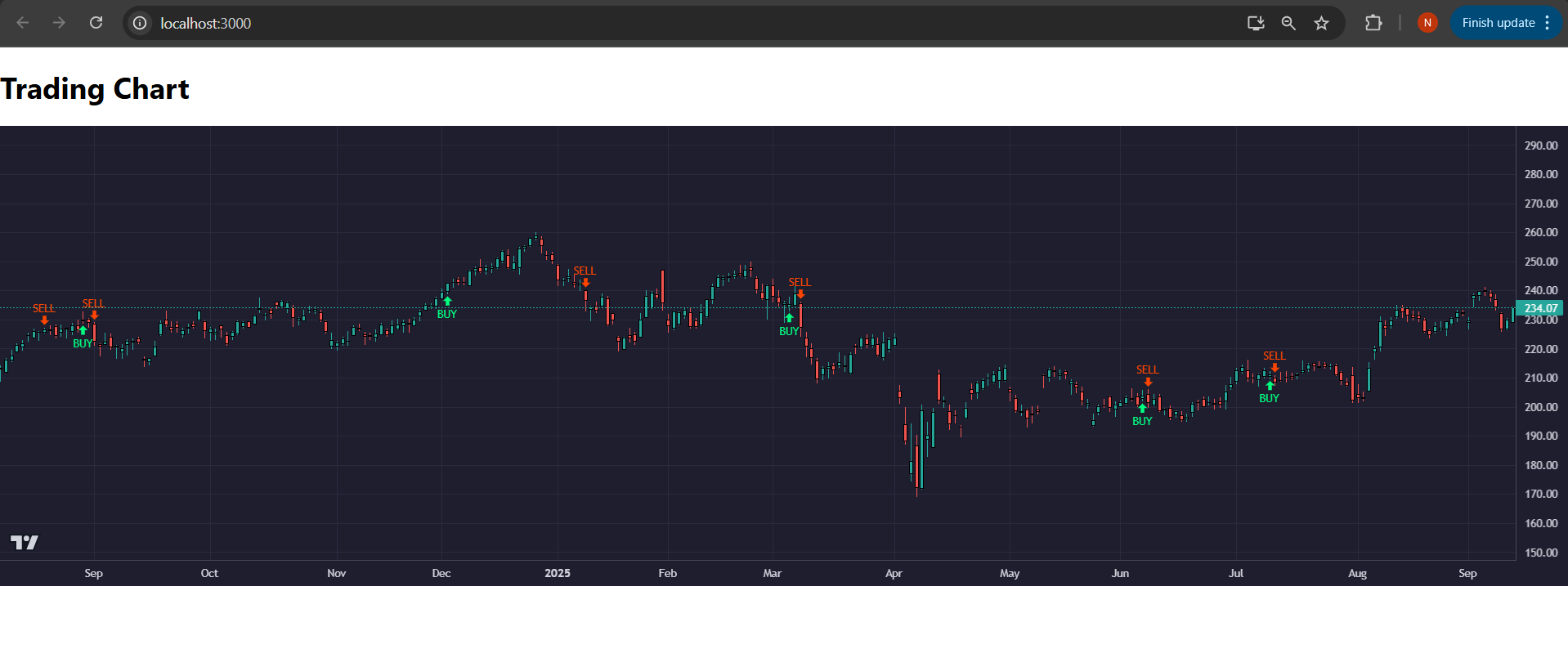
- Normalization: Avoids duplication by separating trades, OHLC, and backtests.  
- Flexibility: Can store multiple backtests and symbols.  
- Ease of Comparison: Performance metrics stored in backtests allow quick comparison across strategies and symbols.

# 5. Backtesting Example

Symbols Tested: AAPL, MSFT, TSLA  
Initial Capital: $10,000  
Short Window: 10 days  
Long Window: 40 days  
Stop Loss: 1%  
Take Profit: 50%  
  
  
  
**Sample Results Table:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Symbol | Total Return (%) | Max Drawdown (%) | Sharpe Ratio | Number of Trades |
| AAPL | 12.5 | -8.3 | 1.12 | 15 |
| MSFT | 18.9 | -10.1 | 1.34 | 20 |
| TSLA | 25.2 | -15.4 | 1.22 | 18 |

# 6. Frontend Visualization

Technology: React + Lightweight Charts  
Features:  
- Candlestick chart of OHLC data  
- Buy/Sell markers on chart  
- Tooltip displaying OHLC for hovered candlestick  
- Responsive resizing  
**Screenshot of UI:**  




# 7. Project Structure

project/  
│  
├─ backend/  
│ ├─ backtest.py  
│ ├─ db.py  
│ ├─ api.py  
│ ├─ lumibot\_strategy.py  
│ ├─ run\_strategy.py  
│ └─ schema.sql  
│  
├─ frontend/  
│ ├─ App.tsx  
│ └─ components/  
│ └─ Chart.tsx  
│  
├─ docker-compose.yml  
├─ .env  
└─ README.docx

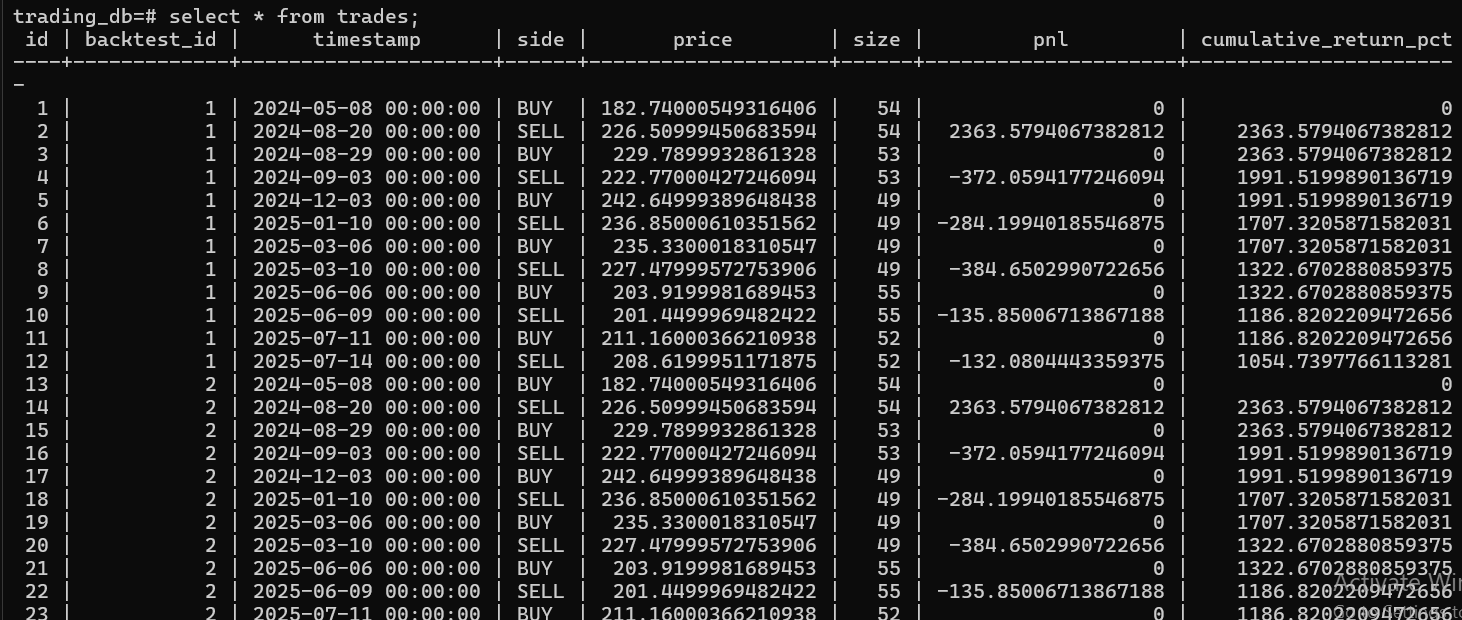
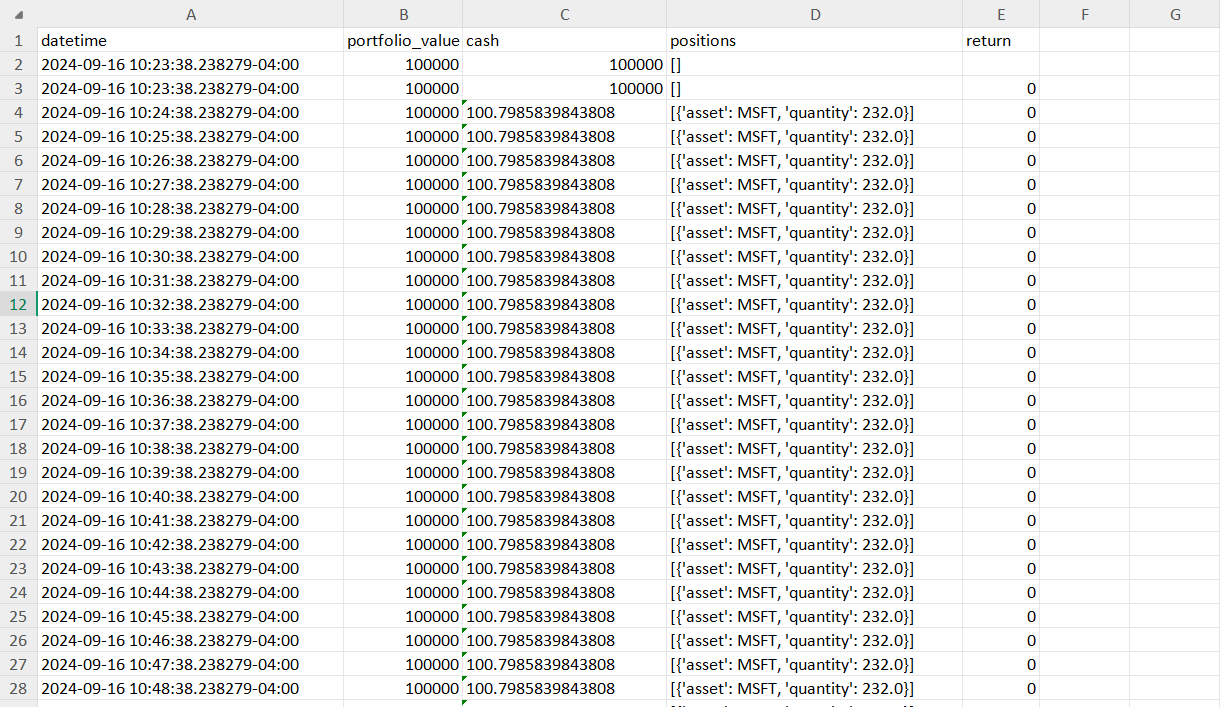
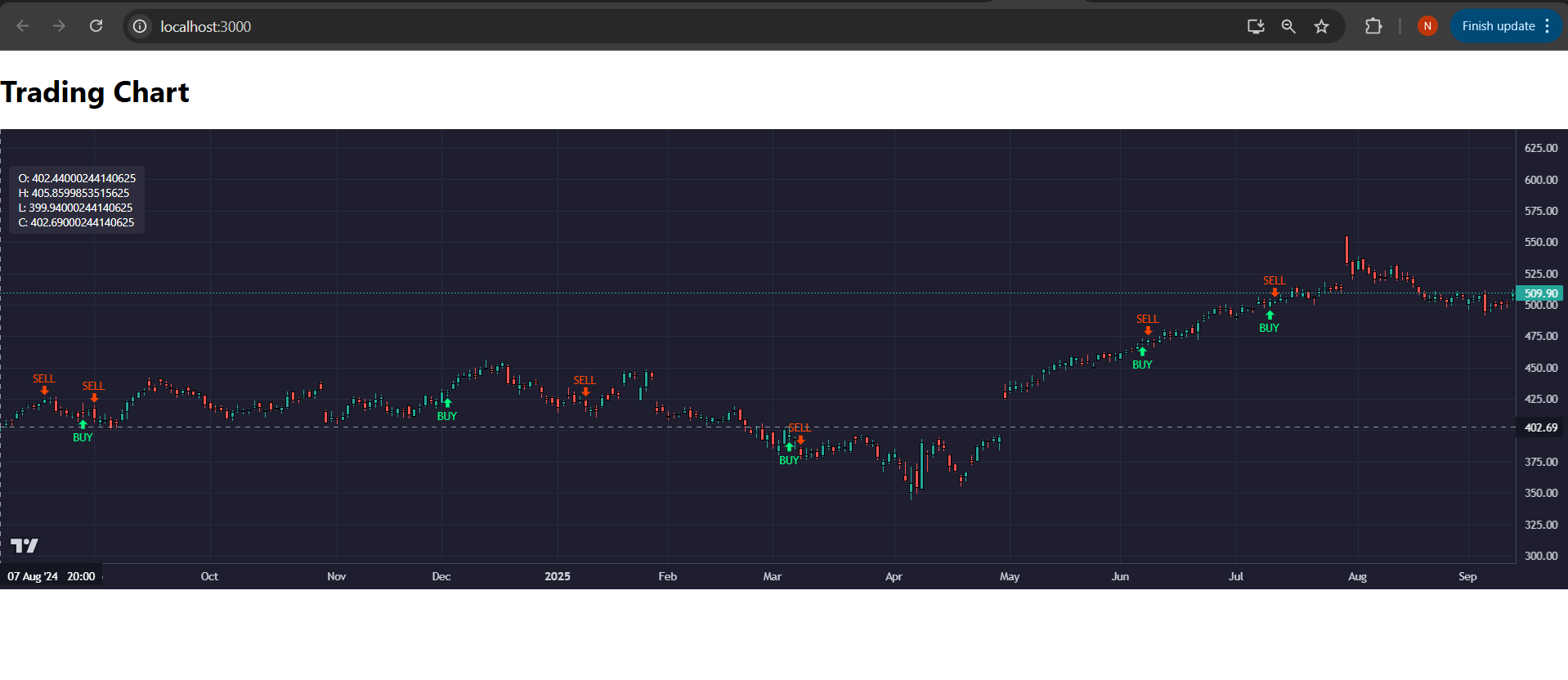
# 8. How to Run

1. Start PostgreSQL using Docker:  
***docker-compose up -d***  
  
2. Install Python dependencies:  
***pip install -r requirements.txt***  
  
3. Run Backtests:  
***python backend/run\_strategy.py***  
4. Start FastAPI Backend:  
***uvicorn backend/api:app --reload***  
  
5. Start Frontend:  
***npm install  
npm start***

# 9. Observations and Insights

- Adding stop-loss reduced risk but slightly lowered overall returns.  
- Take-profit strategy allows booking profits in trending markets.  
- Strategy performs differently across symbols due to volatility differences.  
- Storing data in PostgreSQL enables multi-symbol analysis and historical record-keeping.

1. Additional Screenshots:  
     
   Trades Table Screenshot:

  
  
OHLC CSV Screenshot:  
  
  
Portfolio Chart:  


# 11. References

- Lumibot Documentation: [https://lumibot.lumiwealth.com/](lumibot.lumiwealth.com)  
- Lightweight Charts: [https://github.com/tradingview/lightweight-charts](tradingview/lightweight-charts)  
- SQLAlchemy ORM: [https://www.sqlalchemy.org/](http://www.sqlalchemy.org)