Numatix-Quant Developer Assignment

Multi-Timeframe Strategy Execution & Trade Matching

Deadline: 30th December, 11:59 PM

Objective

The goal of this assignment is to evaluate your ability to:

- Design a **rule-based quantitative trading strategy**
- Implement it in a **clean, modular, class-based Python architecture**
- Ensure **identical behavior** between backtesting and live trading
- Demonstrate discipline in execution, logging, and validation

Strategy profitability or complexity is NOT a selection criterion or required for this assignment

We are evaluating **engineering rigor, correctness, and execution parity**, not alpha.

Core Requirements

1. Strategy Design (Multi-Timeframe)

- Design a **multi-timeframe trading strategy**, for example:
  - 15-minute timeframe for entries
  - 1-hour timeframe for confirmation / filter
- Clearly define:
  - Entry rules
  - Exit rules
  - Position sizing logic
  - Trade direction (long / short)

The strategy must be **deterministic and rule-based**.

---

2. Class-Based Architecture (Mandatory)

You must implement the strategy using **Python classes**.

Important constraint:

> The **same strategy class** must be used for both:

- Backtesting
- Live trading

No duplicate logic.

No “similar but separate” implementations.

This is a **hard requirement**.

---

3. Backtesting Implementation

- Use **backtesting.py** to run historical simulations
- Log **every trade** with:
    - Timestamp
    - Symbol
    - Direction (BUY / SELL)
    - Entry price
    - Exit price
- Save backtest trades to a CSV file:

backtest_trades.csv

---

4. Live Trading System (Binance Testnet)

- Implement a live trading system using **Binance Testnet REST API**
- Use a **modular, class-based structure**
- Execution logic must:
    - Use the same strategy class as backtesting
    - Follow the same signal generation and execution flow
- No hardcoded signals or shortcuts

Live trades must be saved to:

live_trades.csv

---

5. Trade Matching & Validation (Critical)

- Compare backtest trades vs live trades
- Tally:
    - Number of trades
    - Direction
    - Approximate timing
- Trades executed on Binance Testnet must **closely match** those generated during backtesting

Small differences due to:

- Latency
- Candle close timing
  are acceptable, but logic mismatches are not.

---

6. Logging & Observability

You must include:

- Clear logging of:
    - Signal generation
    - Order placement
    - Order execution
- Logs should make it easy to trace:

Market Data → Signal → Order → Fill

---

7. Documentation & Summary

Submit a short summary (1–2 pages or README) covering:

- Strategy logic (high-level)
- Architecture overview
- How parity between backtest and live execution was ensured
- Observations from trade matching

---

8. Interview Walkthrough (Mandatory)

You will be required to:

- Walk through your code
- Explain:
    - Strategy logic
    - Class design
    - Execution flow
    - Trade matching approach

Inability to explain your own code will result in rejection.

What We Are Evaluating

Strong Signals

- Single source of truth for strategy logic
- Clean separation of:
    - Data
    - Strategy
    - Execution
- Discipline in logging and validation
- Awareness of live vs backtest differences

Not Evaluated

- Strategy returns
- Sharpe ratio
- Alpha complexity
- Overfitting

Submission Requirements

Your submission must include:

- Python source code
- backtest_trades.csv
- live_trades.csv
- README / summary document

Ensure the repository is **clean, well-documented, and reproducible**.

Important Notes

- You may use external libraries and tools
- Use of LLM’s is prohibited.
- You **must** understand and explain your implementation