Project Title

CLI-Based Binance USDT-M Futures Order Bot

Problem Statement

Cryptocurrency futures trading requires fast, accurate, and reliable order execution. Manual order placement through exchange interfaces is often inefficient, error-prone, and unsuitable for advanced trading strategies such as time-sliced execution or risk-managed exits. These limitations become more significant when handling large orders or volatile market conditions, where execution delays and human errors can lead to financial losses.

This project focuses on the design and implementation of a **Command-Line Interface (CLI) based automated trading bot** for **Binance USDT-M Futures**. The system supports multiple order types, including both basic and advanced strategies, while ensuring robust input validation, structured logging, and reproducible execution. All operations are performed on the Binance Futures Testnet to ensure safety and correctness.

Objectives

The main objectives of this project are:

- To design and implement **market and limit order execution** using the Binance Futures API
- To support **advanced trading strategies** such as OCO (One-Cancels-the-Other) and TWAP (Time-Weighted Average Price)
- To ensure **robust input validation** to prevent invalid or unsafe trades
- To implement **structured logging** for traceability, debugging, and audit purposes
- To provide a **reproducible and modular CLI-based trading system** suitable for further extension

---

System Architecture

The system follows a layered and modular architecture to ensure clarity, reliability, and extensibility.

Architecture Flow

CLI Input

↓

Validation Layer

↓

Order Logic (Market / Limit / OCO / TWAP)

↓

Binance Futures Testnet API

↓

Logging System (bot.log)

Architecture Explanation

- **CLI Input Layer**: Accepts user commands such as symbol, order type, quantity, price, and strategy parameters.
- **Validation Layer**: Verifies correctness of inputs (symbol format, order side, quantity, price ranges).
- **Order Logic Layer**: Executes the appropriate trading logic based on the selected order type or strategy.
- **Binance Futures Testnet API**: Handles communication with the exchange using authenticated API calls.
- **Logging System**: Records all events, including successful executions and errors, in a structured log file.

Technology Stack

Component	Technology
Programming Language	Python 3
Exchange	Binance USDT-M Futures
API Library	python-binance
Logging	Python logging module
Execution Environment	Binance Futures Testnet
Interface	Command Line Interface (CLI)

Implementation Details

6.1 Market Orders

Market orders are executed immediately at the best available price in the market.

- The user provides the trading symbol, order side, and quantity through the CLI.
- The system validates inputs and submits a **MARKET** order to the Binance Futures Testnet.
- Execution details are logged in bot.log.

Evidence Included:

- Terminal execution:
- python src/market_orders.py BTCUSDT BUY 0.01
- Log entry confirming successful market order placement.

6.2 Limit Orders

Limit orders allow execution at a user-defined price.

- The user specifies symbol, side, quantity, and price.
- The order is placed using timeInForce="GTC" (Good-Till-Cancelled).
- The order remains open until the specified price condition is met.

Evidence Included:

- Terminal command execution
- Log entry confirming limit order placement

6.3 OCO Orders (Advanced)

Binance **USDT-M Futures does not support native OCO orders**.
Therefore, OCO behavior is **correctly simulated** using two independent orders:

- A **take-profit LIMIT order**
- A **stop-loss STOP_MARKET order**

Both orders are placed simultaneously to manage risk effectively.

Evidence Included:

- Terminal command execution
- Log entries confirming:
    - Take-profit order placement
    - Stop-loss order placement

6.4 TWAP Strategy (Advanced – High Value)

The TWAP strategy is designed to reduce market impact when executing large orders.

- The total order quantity is divided into equal smaller parts.
- Each part is executed as a market order at fixed time intervals.
- This approach avoids sudden price movements caused by large single orders.

Evidence Included:

- Terminal output showing multiple executions
- bot.log entries such as:
    - TWAP order 1/5
    - TWAP order 2/5
    - …
    - TWAP execution completed

Logging & Validation

Robust validation and logging are integral to the system.

- **Input validation** prevents invalid symbols, incorrect order sides, and unsafe quantities.
- **Structured logging** ensures full traceability of all actions.
- Logs include:
    - Timestamps
    - Severity levels (INFO, ERROR)
    - Execution and error details

The bot.log file serves as a reliable audit trail for all trading activity.

Results & Observations

- All core and advanced orders were successfully executed on the Binance Futures Testnet.
- Market and limit orders behaved as expected.
- OCO simulation correctly placed both take-profit and stop-loss orders.
- TWAP strategy executed orders at fixed intervals without failure.
- Logging accurately captured all events and validations.

Conclusion

The project successfully achieves its objectives by implementing a fully functional CLI-based trading bot for Binance USDT-M Futures. Both core and advanced order types were implemented with proper validation and logging. The system is modular, extensible, and reproducible, making it suitable for further enhancements and real-world adaptations.

Future Enhancements

Possible future improvements include:

- Implementation of **grid trading strategies**
- Advanced **risk management rules**
- Development of a **web-based or GUI interface**
- Deployment in **live trading environments** with additional safeguards