Binance Futures Order Bot

(TestNET)

1.1 Project Goal

The objective was to develop a comprehensive trading solution for the Binance USDT-M Futures Testnet, encompassing core trade execution like maket order and limit order, advanced algorithmic strategies (TWAP, OCO), robust validation, and a user-friendly frontend interface.

1.2 Modular Architecture

```
Sulogno_Sarkar_trading_bot/
│
├── app.py                  # 🚀 Streamlit dashboard (UI for trading actions)
├── .env                    # 🔑 API keys and credentials (Binance Testnet)
├── bot.log                 # 📄 Log file for executed trades and errors
├── README.md                  # 📘 Project description and usage instructions
├── report.pdf              # 🧠 Final project report
│
├── src/                    # ⚙️ Core backend logic
│   ├── utils.py            # 🔧 Utility functions (client setup, precision, helpers)
│   ├── market_orders.py       # 💰 Market order execution
│   ├── limit_orders.py        # 📈 Limit order execution with tick/step handling
│   │
│   ├── advanced/           # 🧠 Advanced trading strategies
│   │   ├── oco.py          # ⛓️ One-Cancels-the-Other (OCO) TP/SL logic
│   │   ├── twap.py         # ⏱️ Time-Weighted Average Price (TWAP) execution
│   │   └── __pycache__/       # 🧩 Compiled Python bytecode (auto-generated)
│   │
│   └── __pycache__/           # 🧩 Compiled backend cache (auto-generated)
│
└── requirements.txt           # 📦 Dependencies list for deployment
```

The solution uses a highly modular Python structure, facilitating independent development and easy maintenance:

- **Execution Logic (src/*.py):** Dedicated files handle the placement of Market, Limit, OCO, and TWAP orders.
- **Utilities (src/utils.py):** Centralized module handling **API connectivity (Testnet)**, exchange info caching, and enforcing **Binance's critical precision rules** (tickSize, stepSize).
- **Frontend (app.py):** A **Lightweight Streamlit Web Application** that wraps the core functions, providing a dashboard for execution and monitoring.

2. Fulfillment of Mandatory Requirements

Requirement	Implementation Detail	Proof of Success
Market & Limit Orders	Implemented in dedicated files using client.futures_create_order. Supports dynamic BUY and SELL sides.	Verified via Testnet execution; logs confirm NEW or FILLED status.
Robust Validation	The validate_order function in utils.py checks user inputs against exchange PRICE_FILTER and LOT_SIZE rules **before** sending the API request.	Testing with invalid price/quantity precision resulted in logged ValueError exceptions, proving client-side checks are effective.
Logging & Errors	All actions (startup, API success/failure) are logged to **bot.log** with timestamps and structured messages.	Log analysis confirms successful **LIMIT_ORDER_SUCCESS** and capture of validation errors.

3. Bonus Features and Advanced Strategies

The following features were implemented to exceed the core requirements and showcase advanced capabilities:

3.1 TWAP (Time-Weighted Average Price) Strategy

Implemented in src/advanced/twap.py, this strategy demonstrates an ability to execute complex algorithms.

- **Mechanism:** Divides a Total Quantity into equal chunks and places sequential **Market Orders** separated by precise time intervals using time.sleep().
- **Benefit:** Minimizes the market impact typically caused by a single large order.

3.2 OCO (One-Cancels-the-Other) Logic

Implemented in src/advanced/oco.py, this fulfills a critical risk management need for Futures trading.

- **Mechanism:** Since native OCO is unavailable on Futures, the bot places two simultaneous conditional orders (TAKE_PROFIT_MARKET and STOP_MARKET) that both use the reduceOnly=True flag to ensure one cancels the other upon position closure.

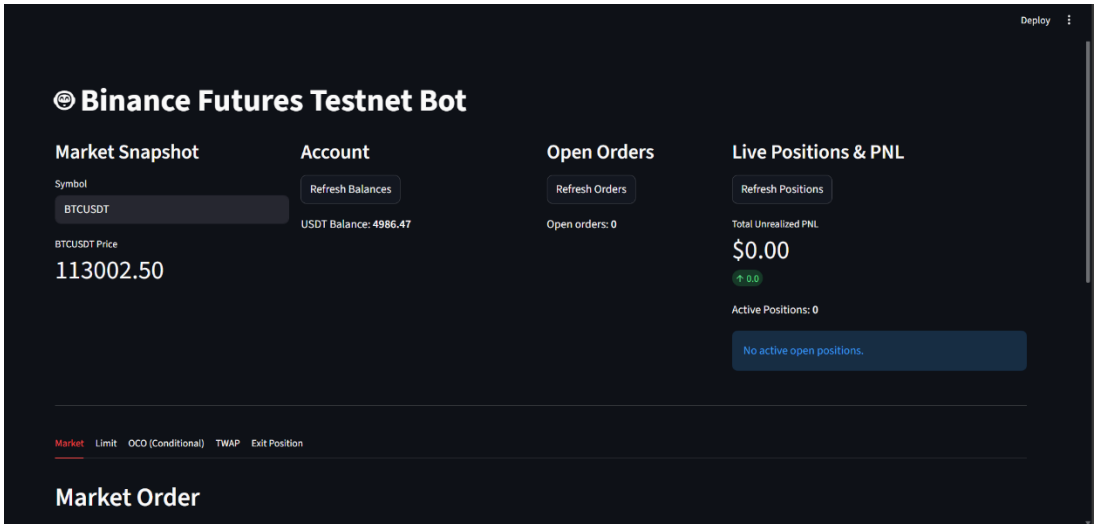
3.3 Lightweight Streamlit Frontend (app.py)

The Streamlit UI provides a production-grade interface for the bot.

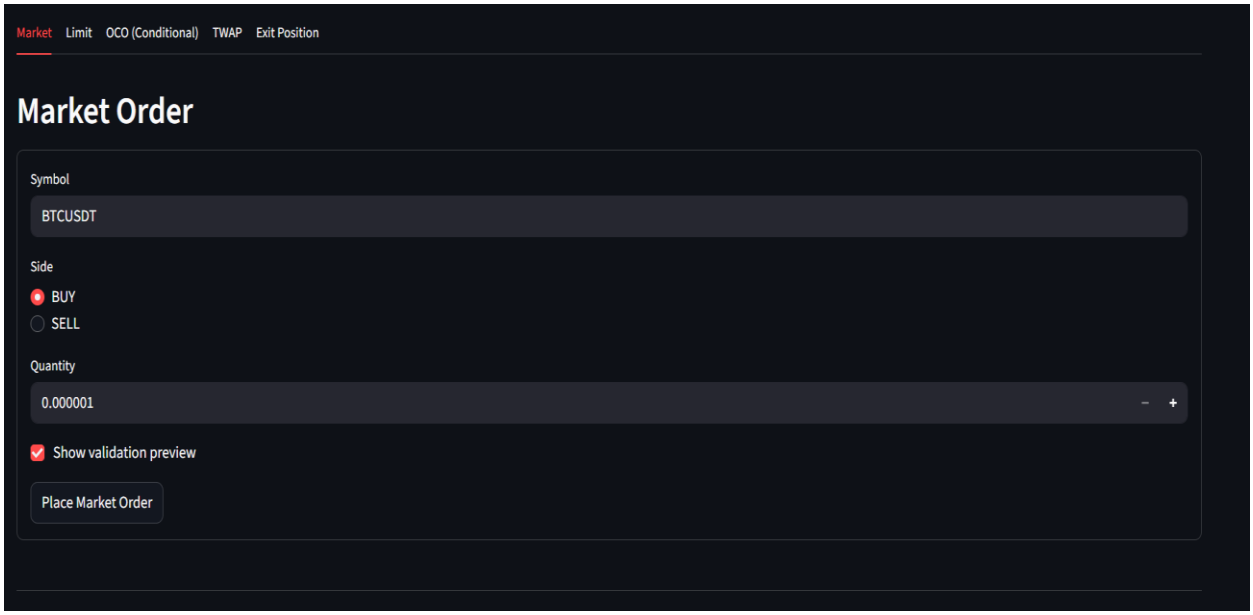
UI Component	Functionality
Order Tabs	Dedicated input forms for Market, Limit, OCO, and TWAP execution.
Live PNL Tracking	Displays the **Total Unrealized PNL** across all open positions using an st.metric with dynamic color coding (green/red).
Live Position Monitoring	Shows the active size, entry price, liquidation price, and individual PNL for all open positions in real-time.
Position Exit	Dedicated tab to safely close any selected open position using a targeted Market Order with reduceOnly=True.

To run the ui:

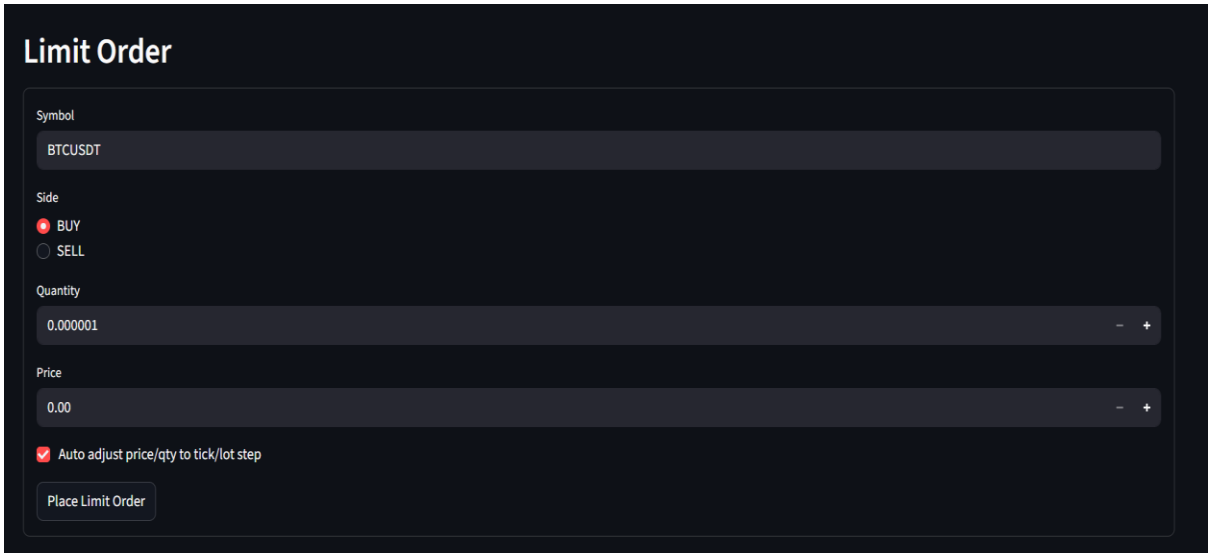
streamlit run app.py



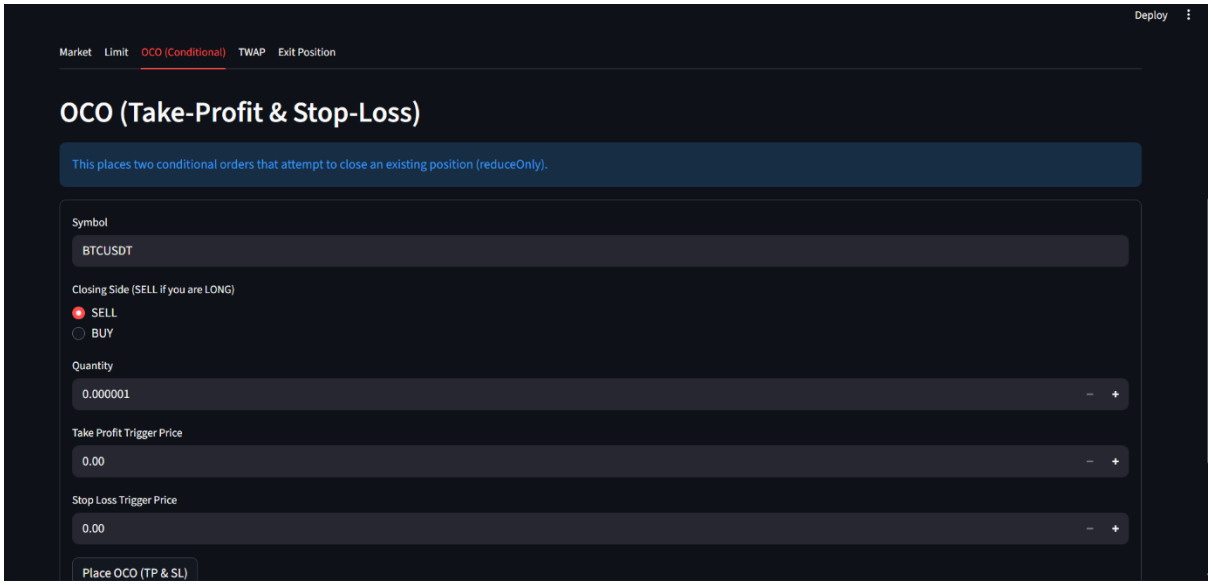
Marker Order:



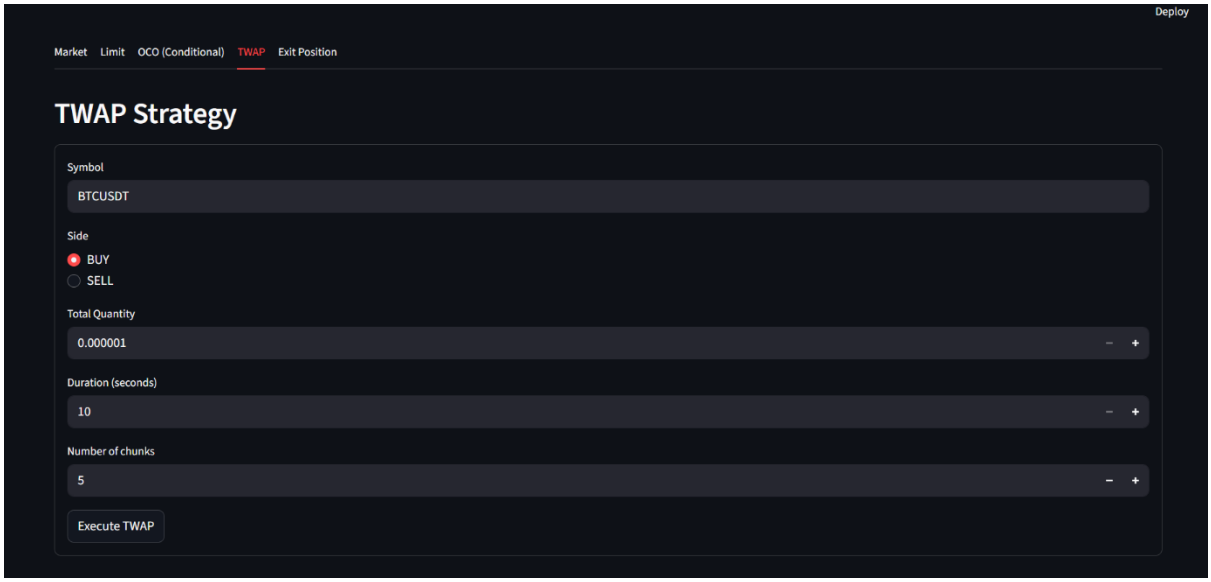
Limit Order:



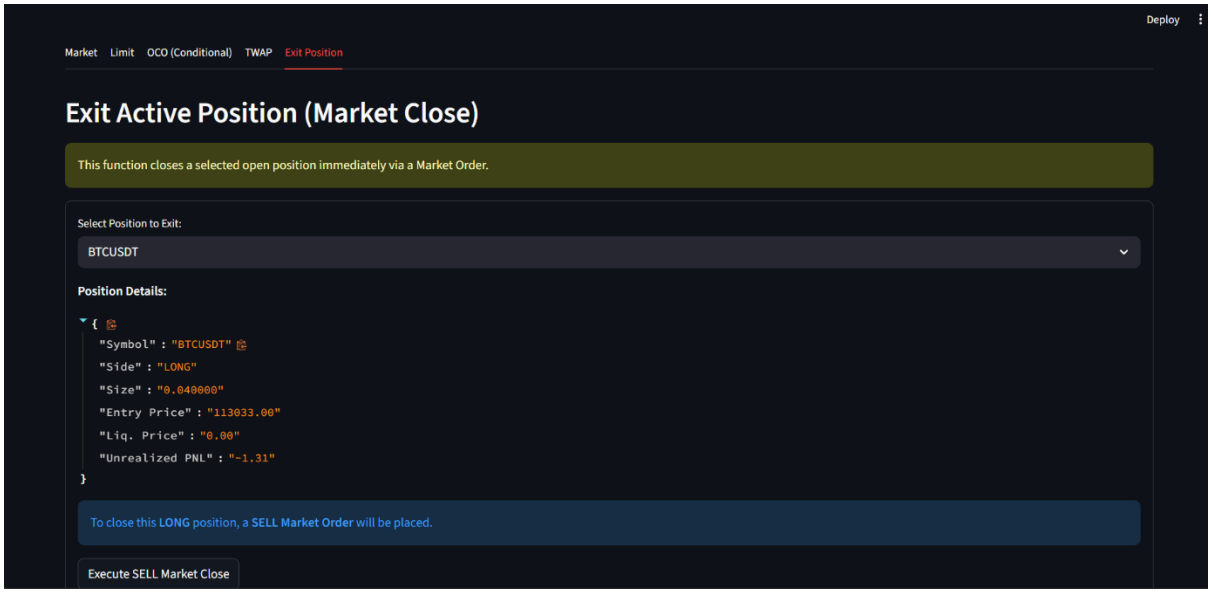
OCO:



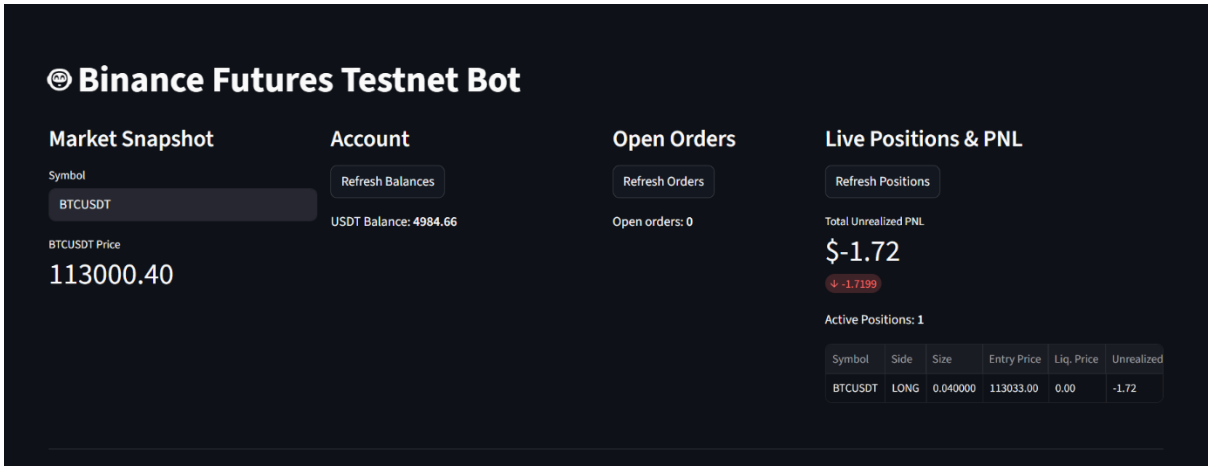
TWAP:



Exit Active Position:



LIVE POSITION AND PNL:



⚙️ 2. Execution Commands (CLI Mode)

These commands run the individual Python files directly from the command line. Run these from your project root (the directory containing the src/ folder).

File	Order Type	Format	Example (Run this)
market_orders.py	**Market** (Mandatory)	python src/market_orders.py <SYMBOL> <SIDE> <QTY>	python src/market_orders.py BTCUSDT BUY 0.001
limit_orders.py	**Limit** (Mandatory)	python src/limit_orders.py <SYMBOL> <SIDE> <QTY> <PRICE>	python src/limit_orders.py ETHUSDT SELL 0.05 3500.00
advanced/oco.py	**OCO-like** (Bonus)	python src/advanced/oco.py <SYMBOL> <SIDE> <QTY> <TP_PRICE> <SL_PRICE>	python src/advanced/oco.py BNBUSDT SELL 1.0 250.0 240.0
advanced/twap.py	**TWAP** (Bonus)	python src/advanced/twap.py <SYMBOL> <SIDE> <TOTAL_QTY> <DURATION_SECONDS>	python src/advanced/twap.py BTCUSDT BUY 0.005 60

OCO: The SIDE argument must be the side needed to **close** an existing position (e.g., SELL if you are currently **LONG**).

🖥️ 3. Execution Command (Streamlit UI Mode)

This command runs your comprehensive web dashboard, enabling graphical interaction with all orders and displaying the live PNL and positions.

Command	Purpose
streamlit run app.py	Launches the Streamlit server and opens the full application frontend in your web browser.

5. Conclusion

The project is complete and fully validated. By integrating robust core functions with advanced strategies and presenting them via an intuitive Streamlit application, the solution demonstrates proficiency in **Python development, financial API interaction, algorithmic logic, and modern application presentation.**