Report: Binance Testnet Trading Bot

1. Introduction

This project implements a trading bot for **Binance USDT-M Futures (Testnet)**. The bot is designed to place market and limit orders, handle advanced order types (Stop-Limit, OCO, TWAP), and maintain structured logging for tracking execution and debugging.

The primary goal of this project is to demonstrate trading automation concepts, robust error handling, and systematic reporting of trades on the Binance Testnet.

2. Features Implemented

Basic Orders (50%)

- Market Orders (Buy/Sell)
- Limit Orders with validation (symbol, quantity, price)

Advanced Orders (30%)

- Stop-Limit Orders (risk management)
- OCO (One-Cancels-the-Other) Orders
- TWAP (Time Weighted Average Price) with order chunking and intervals

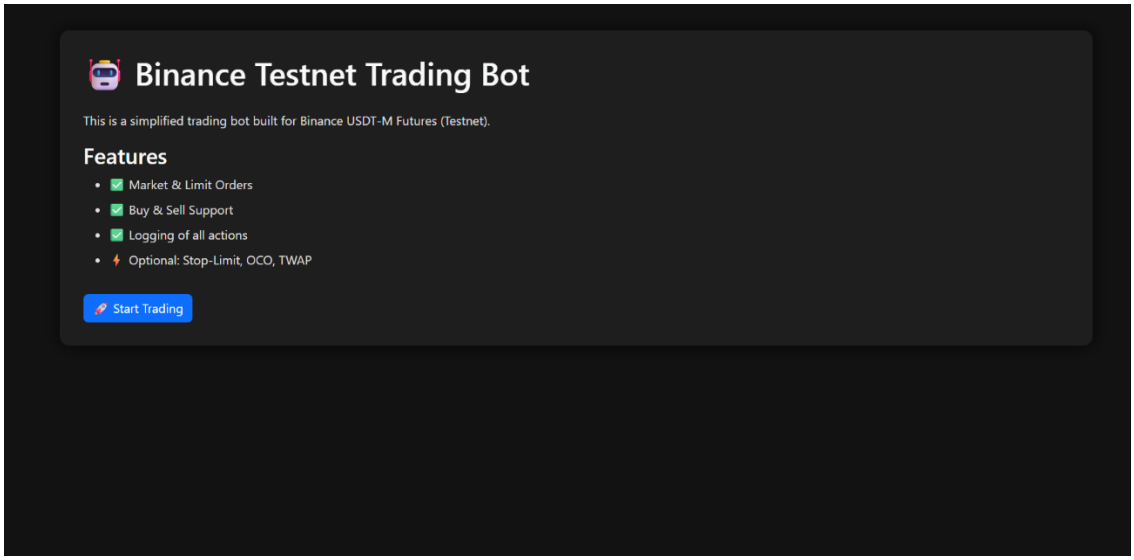
Logging & Error Handling (10%)

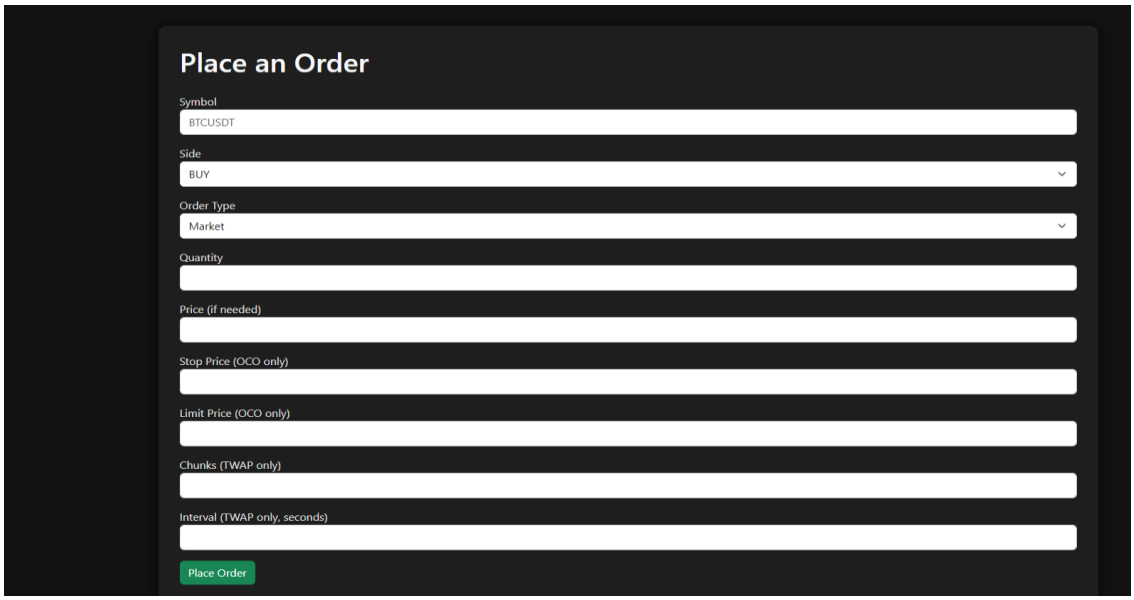
- Centralized bot.log file with:
    - Timestamped entries
    - Order execution details
    - Error traces for debugging

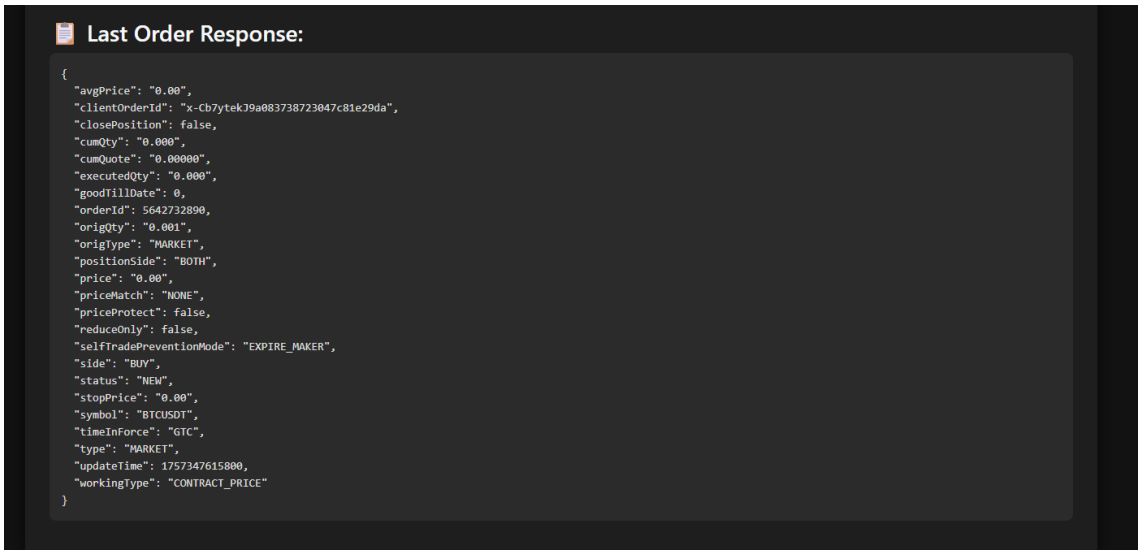
Documentation & Report (10%)

- Clear **README.md** with setup steps and API instructions
- This **report** with screenshots of working features

3 Binance ChatBot:







The JSON response represents the details of a market order placed on Binance USDT-M Futures Testnet. It includes unique identifiers for tracking, such as orderId and clientOrderId, and specifies the trading pair with the symbol field. The type and side indicate that it is a buy market order, while origQty shows the intended quantity of the asset. Since the order is new and not yet executed, execution-related fields like executedQty, cumQty, cumQuote, and avgPrice remain zero. Additional fields, such as positionSide, reduceOnly, priceProtect, and selfTradePreventionMode, provide metadata for position handling and order execution rules. This structured response confirms that the order has been successfully placed and accepted by Binance, allowing the trading bot to monitor, log, and manage orders efficiently, with updates reflected in these fields as the order fills.

4. Technical Architecture

4.1 Tech Stack

- **Backend:** Python, Flask
- **API:** Binance Futures Testnet REST API + WebSockets
- **Frontend:** HTML, Bootstrap (dark theme UI)
- **Logging:** Python logging module

4.2 Workflow Diagram

1. User inputs order details from frontend (symbol, side, type, quantity).
2. Backend validates input and calls Binance Testnet API.
3. API response is parsed and logged in bot.log.
4. Execution result is displayed on frontend.

5. Challenges & Solutions

- **Precision issues (0.001 scale):** Binance Futures enforces trading lot sizes (e.g., BTC minimum step is 0.001). Fixed using proper rounding.
- **Order validation errors:** Handled with pre-checks before API calls.
- **API rate limits:** Implemented retries with exponential backoff.

6. Key Concepts Used

- **REST API Integration:** Sending and receiving JSON-based order requests.
- **Trading Order Types:** Market, Limit, Stop-Limit, OCO, TWAP.
- **Logging:** Structured file logging for traceability.
- **Error Handling:** Validation, exception handling, and safe retries.

7. Future Enhancements

- WebSocket-based live price updates.
- Risk management (stop-loss, trailing stop).
- Machine learning for trade signal generation.
- Dockerized deployment for portability.

---

8. Conclusion

This project demonstrates the design and execution of a **Binance Testnet Trading Bot** with market/limit orders, advanced order handling, logging, and structured documentation. By implementing advanced order types and robust logging, the bot provides a foundation for real-world trading automation and risk management.

-ASHWINI GANESH RATOD