Binance Trading Bot - Report

1. Project Overview

Purpose of the Project: The purpose of this project is to build a Binance Trading Bot that interacts with the Binance Futures Testnet API. The bot allows users to place different types of orders (market, limit, stop-limit, OCO, TWAP, grid) through a Flask web interface. The bot is designed for educational purposes, helping users automate the process of trading in a simulated environment.

The bot supports:

- Basic Orders: Market and Limit orders.
- Advanced Orders: Stop-Limit, One-Cancels-the-Other (OCO), Time-Weighted Average Price (TWAP), and Grid trading.
- Logging and Error Handling: Logs every request and response for transparency and debugging.

Key Technologies:

- **Python:** Programming language used.
- **Flask:** For the web interface and backend handling.
- **Binance API:** For interacting with the Binance Futures Testnet.
- **HTML/CSS:** For building the frontend.
- **ReportLab:** Used to generate the final report.pdf.

2. How the Bot Works

The Binance Trading Bot allows users to place various types of orders on the Binance Futures Testnet. The process flow is as follows:

1. **User Input:** The user interacts with the bot via a Flask web form that collects necessary order details (symbol, order type, price, quantity, etc.).
2. **Order Execution:** Once the user submits the form, the Flask app sends the details to the Binance Futures API. The bot then places the order and returns the result to the user.
3. **Order Types:**
   - **Market Order:** This type of order is executed immediately at the current market price.
   - **Limit Order:** The order will only execute when the market price matches the specified price.
   - **Stop-Limit Order:** When the stop price is reached, a limit order is triggered.
   - **OCO (One Cancels the Other):** Two orders are placed simultaneously — if one is filled, the other is automatically canceled.
   - **TWAP (Time-Weighted Average Price):** A large order is split into smaller parts and placed over a period of time.
   - **Grid Trading:** Multiple limit orders are placed at different price intervals.
4. **Logging:** Every interaction (successful order placement or errors) is logged in a bot.log file, providing a record of all actions and errors.
5. **Order Status:** After placing an order, the bot logs the details and provides feedback to the user through the web interface. The order can also be checked on the Binance Testnet account.

3. Code Breakdown

The code of the Binance Trading Bot is modular and structured into different components. Below are the key parts:

1. **app.py** (Main Flask App)

- Handles incoming user requests from the web form.
- Executes market, limit, stop-limit, and other order types.
- Logs each action and displays the results to the user.

2. **market_orders.py**

- Handles placing market orders via the Binance API.
- Validates inputs such as quantity and order type.

3. **limit_orders.py**

- Handles placing limit orders via the Binance API.
- Ensures the order price is correct and meets the minimum notional requirement.

4. **stop_limit.py**

- Implements stop-limit orders, where the order is placed only when the stop price is hit.

5. **oco.py**

- Implements OCO orders — placing two orders (take profit and stop loss), where if one is executed, the other is canceled.

6. **twap.py**

- Implements TWAP orders — splitting large orders into smaller parts and placing them over time.

7. **grid.py**

- Implements grid trading, placing multiple limit orders at different intervals within a price range.

8. **logger.py**

- Sets up logging to track every order placement and any errors that occur.

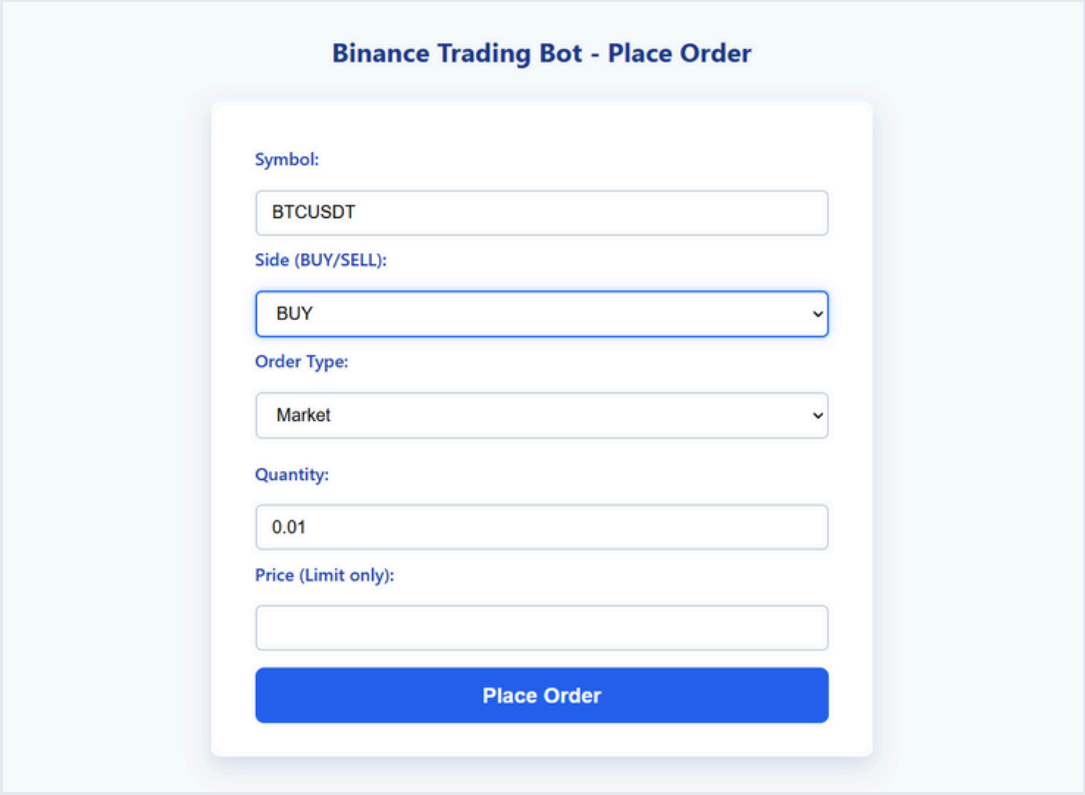
9. **config.py**

- Contains sensitive information like API keys and configuration data.

4. Screenshots

Below are some of the screenshots that you can add to this report:

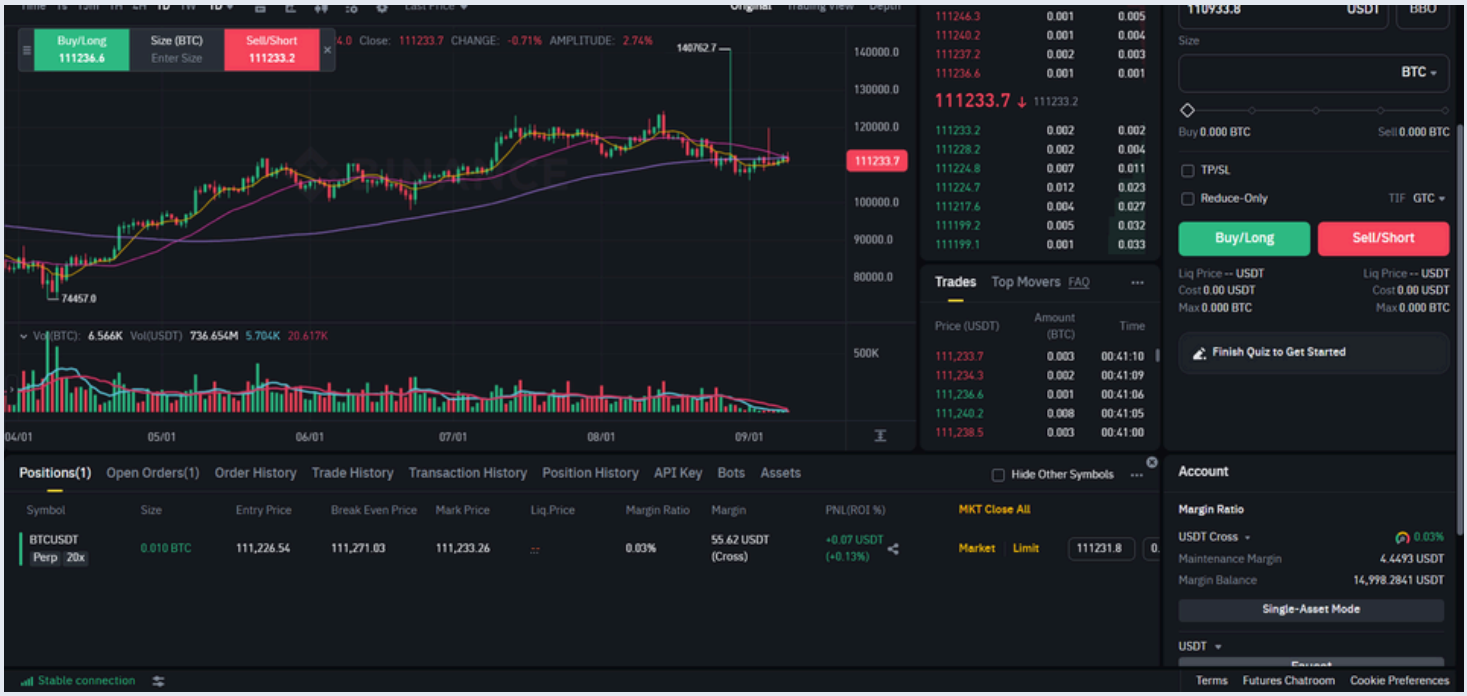
Screenshot 1: Flask UI after placing a Market Order(confirming the success message)



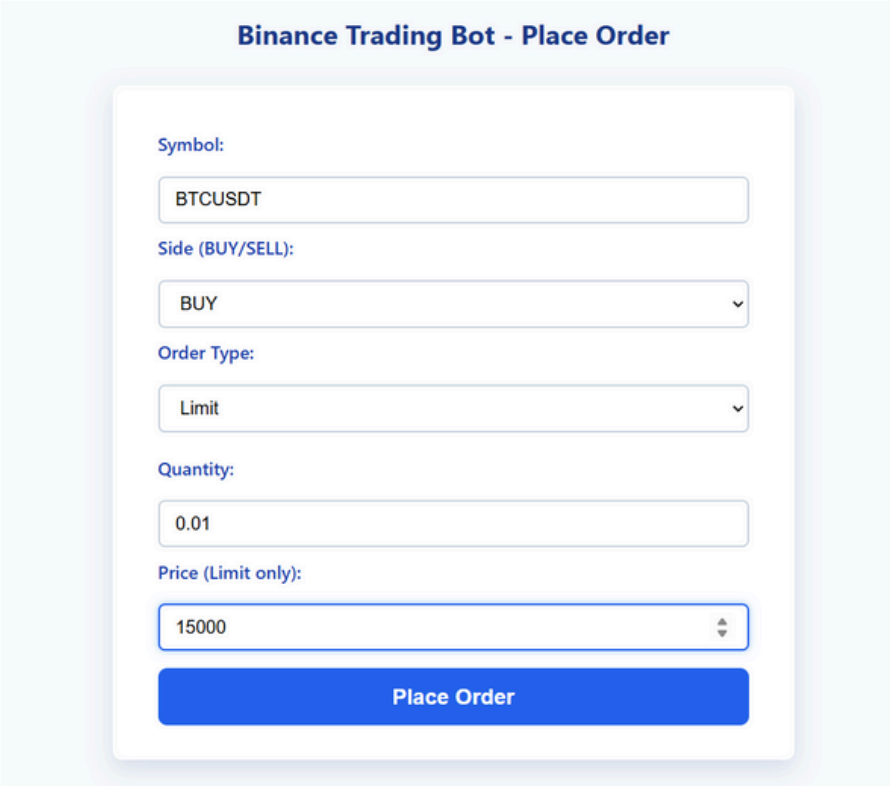
Order placed successfully!

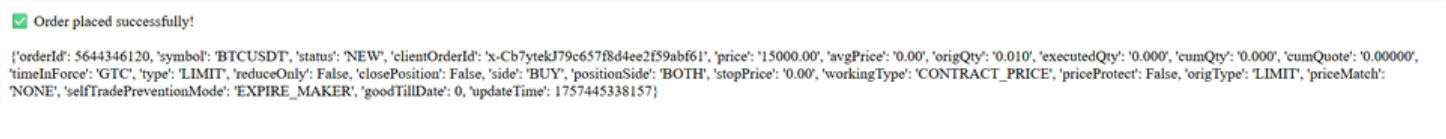
{'orderId': 5644339197, 'symbol': 'BTCUSDT', 'status': 'NEW', 'clientOrderId': 'x-Cb7ytekJd941342238022aa1ef3071', 'price': '0.00', 'avgPrice': '0.00', 'origQty': '0.010', 'executedQty': '0.000', 'cumQty': '0.000', 'cumQuote': '0.00000', 'timeInForce': 'GTC', 'type': 'MARKET', 'reduceOnly': False, 'closePosition': False, 'side': 'BUY', 'positionSide': 'BOTH', 'stopPrice': '0.00', 'workingType': 'CONTRACT_PRICE', 'priceProtect': False, 'origType': 'MARKET', 'priceMatch': 'NONE', 'selfTradePreventionMode': 'EXPIRE_MAKER', 'goodTillDate': 0, 'updateTime': 1757444874229}

Screenshot 2: Binance Testnet Dashboard showing the order status in the Binance account.



Screenshot 3: Flask UI after placing a Limit Order.





5. Challenges and Solutions

Challenge 1: Handling API Errors

During the development, several API errors occurred due to invalid API keys or insufficient permissions.

- **Solution:** Improved error handling with detailed logging to ensure better understanding of the issue. Used the proper testnet API keys and verified the permissions.

Challenge 2: Handling Timeouts

During the development, several API errors occurred due to invalid API keys or insufficient permissions.

- **Solution:** Improved error handling with detailed logging to ensure better understanding of the issue. Used the proper testnet API keys and verified the permissions.

Challenge 3: Binance API Limits

Binance's API has strict limits on the notional value of orders, meaning the order quantity and price need to be large enough to meet the minimum value (100 USDT).

- **Solution:** Added logic to check the notional value before placing the order and return an appropriate message to the user if the order size is too small.