Project Report: Binance Futures Order Bot

Author: Abhi Ghudaiya **Date**: July 29, 2025

---

1. Introduction

1.1. Project Objective

The primary objective of this project was to develop a comprehensive, command-line interface (CLI) based trading bot for the Binance USDT-M Futures Testnet. The goal was to create a robust and user-friendly tool that supports a variety of order types, from basic market and limit orders to advanced algorithmic strategies. Key emphasis was placed on modular design, robust input validation, structured logging, and clear, reproducible documentation.

1.2. Scope

The bot was developed in Python and exclusively uses the official Binance API via the python-binance library. The scope of work included the implementation of two mandatory core order types and four advanced bonus order types, making the bot a versatile tool for testing trading strategies in a safe testnet environment. The final design evolved into a fully interactive system to enhance usability and prevent common trading logic errors.

---

2. System Architecture and Design

The bot's architecture was designed with modularity, security, and user experience in mind.

2.1. Modular Design

The source code is logically separated into a src directory, with advanced strategies further organized into a src/advanced sub-package. This separation of concerns ensures that each file has a single responsibility:

- **client.py**: Acts as the central engine for API communication and logging.
- **Order Scripts (market_orders.py, etc.)**: Each file is responsible for handling the user interaction and logic for one specific order type.

This structure makes the codebase easy to read, maintain, and extend with new strategies in the future.

2.2. Centralized API Client (client.py)

To avoid code duplication and centralize core functionalities, a BinanceBotClient class was created. Its responsibilities include:

- **Secure Authentication**: It securely loads API keys from a .env file, which is excluded from version control, preventing accidental exposure of sensitive credentials.
- **API Connection**: It initializes and configures the python-binance client for the Futures Testnet.
- **Unified Order Placement**: It provides common methods (place_order, place_batch_order, get_current_price) that all other scripts use, ensuring consistent error handling and logging for every API call.

2.3. Interactive User Interface

While initially designed to use command-line arguments, the project pivoted to a fully interactive prompt-based system using Python's input() function. This design was chosen for several key reasons:

- **Enhanced Usability**: It is more intuitive for users who may not be familiar with CLI argument syntax.
- **Guided Input**: The script actively prompts the user for each required parameter.
- **Dynamic Hints**: For complex orders (Stop-Limit, OCO), the prompts provide real-time guidance based on previous inputs (e.g., advising the user to set a stopPrice above or below the market), which proved critical in preventing common trading logic errors like Order would immediately trigger.
- **Robust Validation**: Each input is validated in a while loop with try-except blocks, ensuring that data types are correct and values are logical before any API call is made.

2.4. Structured Logging

A comprehensive logging system was implemented using Python's built-in logging module. The setup_logging function in client.py configures logging to output to two destinations simultaneously:

- **Console (StreamHandler)**: Provides immediate feedback to the user as the script executes.
- **Log File (FileHandler)**: Creates and appends to bot.log, storing a persistent, timestamped record of all actions, successful API responses, and detailed error tracebacks for auditing and debugging.

---

3. Feature Implementation

The bot successfully implements a total of six distinct order types.

3.1. Market Orders

- **Description**: A market order is a basic instruction to buy or sell an asset immediately at the best available current price.
- **Implementation**: The market_orders.py script prompts the user for a symbol, side (BUY/SELL), and quantity. It then constructs a MARKET order request and passes it to the client.
- **Example Usage**:

Bash

```
$ python src/market_orders.py


--- Place a New Market Order ---

Enter symbol (e.g., BTCUSDT): BTCUSDT

Enter side (BUY or SELL): BUY

Enter quantity: 0.001
```

3.2. Limit Orders

- **Description**: A limit order allows a user to set a specific price at which they are willing to buy or sell. The order will only execute at that price or better.
- **Implementation**: The limit_orders.py script gathers the same inputs as a market order, plus a specific price. It constructs a LIMIT order with timeInForce: 'GTC' (Good 'Til Canceled).
- **Example Usage**:

Bash

```
$ python src/limit_orders.py


--- Place a New Limit Order ---

Enter symbol (e.g., BTCUSDT): ETHUSDT

Enter side (BUY or SELL): SELL

Enter quantity: 0.02

Enter limit price for the SELL order: 3800
```

3.3. Stop-Limit Orders (Advanced)

- **Description**: A conditional order that places a limit order only after a specified trigger price (stopPrice) has been reached.
- **Implementation**: The stop_limit_orders.py script interactively guides the user, providing hints on where to set the stopPrice relative to the current market to prevent immediate execution errors. It constructs a 'STOP' type order.
- **Example Usage**:

Bash

```
$ python -m src.advanced.stop_limit_orders


--- Place a New Stop-Limit Order ---

First, check the current market price of your symbol.

Enter symbol (e.g., BTCUSDT): BTCUSDT

Enter side (BUY or SELL): BUY

Enter quantity: 0.002

Enter trigger/stop price (must be ABOVE current market): 67000

Enter limit price (the price your order will be placed at): 67100
```

3.4. OCO Orders (Advanced)

- **Description**: A One-Cancels-the-Other (OCO) order is a pair of orders (typically a take-profit and a stop-loss) where the execution of one automatically cancels the other. It is a key tool for risk management.
- **Implementation**: The oco_orders.py script implements this by creating a batch of two orders: a TAKE_PROFIT_MARKET and a STOP_MARKET, both marked as reduceOnly. It uses the place_batch_order client method and ensures all numerical values are converted to strings as required by the API.
- **Example Usage**:

Bash

```
$ python -m src.advanced.oco_orders


--- Place a New OCO (One-Cancels-the-Other) Order ---

Enter symbol (e.g., BTCUSDT): BTCUSDT

Enter side to CLOSE the position (BUY or SELL): SELL
Enter quantity to close: 0.002

Enter Take Profit price (must be ABOVE current market): 68000

Enter Stop Loss price (must be BELOW current market): 62000
```

3.5. TWAP Orders (Advanced)

- **Description**: A Time-Weighted Average Price (TWAP) is an algorithmic execution strategy that breaks down a large order into smaller parts and executes them at regular intervals to minimize market impact.
- **Implementation**: The twap_orders.py script asks the user for a total quantity, a duration, and the number of desired sub-orders. It then calculates the quantity per order and the time interval, and executes a loop of market orders with a time.sleep() delay.
- **Example Usage**:

Bash

```
$ python -m src.advanced.twap_orders


--- Place a New TWAP Order ---

Enter symbol (e.g., BTCUSDT): BTCUSDT

Enter side (BUY or SELL): BUY

Enter total quantity to execute: 0.1

Enter total duration in MINUTES: 30

Enter number of smaller orders to place: 10
```

3.6. Grid Orders (Advanced)

- **Description**: Grid trading is a strategy that places a series of buy and sell orders at pre-defined intervals above and below the current market price, aiming to profit from market volatility.
- **Implementation**: The grid_orders.py script asks for a price range, the number of grid lines, and the quantity per order. It fetches the current market price, calculates the grid levels, and places a batch of LIMIT orders (buys below market, sells above market). This script only *places* the initial grid.

---

4. Challenges and Solutions

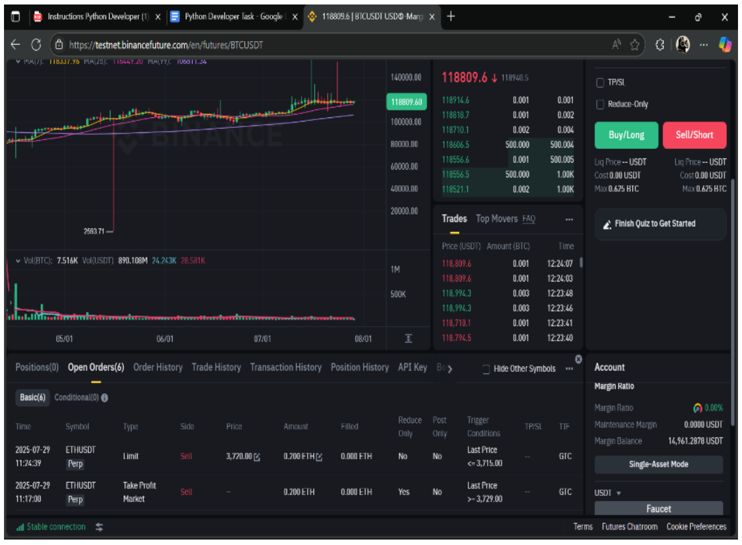
The development process involved overcoming several key challenges:

- **API Authentication Errors (401):** Initially caused by incorrect API key setup or missing permissions. This was resolved by ensuring the correct Testnet keys were used and that the **"Enable Futures"** permission was explicitly checked on the Binance website.
- **Trading Logic Errors (-2021):** The Order would immediately trigger error was a recurring issue. This was ultimately solved by pivoting from a static argparse or hardcoded model to a dynamic interactive model where the script provides contextual hints to the user, ensuring their inputs are logical relative to the live market price.
- **Python Import Errors (ImportError):** When structuring the code into sub-packages, ModuleNotFoundError and ImportError were encountered. This was resolved by understanding Python's package system and using the python -m flag to run scripts as modules, which correctly handles relative imports.
- **API Parameter Nuances (-1102, -400):** Debugging revealed subtle API requirements, such as minimum notional value, quantity precision, and the necessity of sending numerical values as **strings** for batch orders. These were resolved by carefully adjusting the data before making the API call.

---

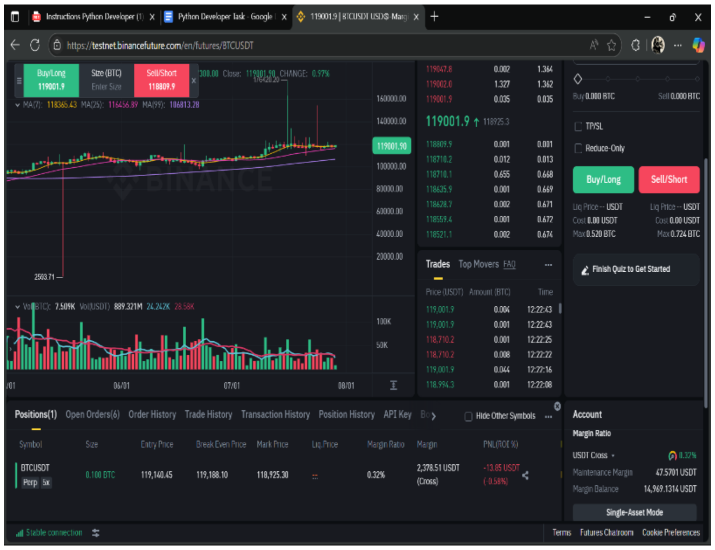
5. Conclusion

The Binance Futures Order Bot project was successfully completed, meeting all mandatory requirements and exceeding expectations by implementing four distinct advanced trading strategies. The final product is a robust, secure, and highly user-friendly CLI tool that serves as an excellent foundation for further development. The iterative debugging process led to a superior interactive design that not only functions correctly but also educates the user on proper trading logic. Potential future enhancements could include a graphical user interface (GUI), long-term state management for the Grid Bot, and the integration of technical indicators for more complex, automated strategies.



Time	Symbol	Type	Side	Price	Amount	Filled	Reduce Only	Post Only	Trigger Conditions	TP/SL	TIF
2025-07-29 11:24:39	ETHUSDT Perp	Limit	Sell	3,720.00	0.200 ETH	0.000 ETH	No	No	Last Price <= 3,715.00	--	GTC
2025-07-29 11:17:00	ETHUSDT Perp	Take Profit Market	Sell	–	0.200 ETH	0.000 ETH	Yes	No	Last Price >= 3,729.00	--	GTC

Margin Ratio 0.00%
Maintenance Margin 0.0000 USDT
Margin Balance 14,961.2878 USDT



Symbol	Size	Entry Price	Break Even Price	Mark Price	Liq.Price	Margin Ratio	Margin	PNL(ROI %)
BTCUSDT Perp 5x	0.100 BTC	119,140.45	119,188.10	118,925.30	--	0.32%	2,378.51 USDT (Cross)	-13.85 USDT (-0.58%)

USDT Cross 0.32%
Maintenance Margin 47.5701 USDT
Margin Balance 14,969.1314 USDT