Project Report: Binance CLI Bot

Author: Neeraj Patel

Date: September 9, 2025

Abstract

This report details the design, features, and technical challenges faced during the development of the Binance Futures CLI Trading Bot. The goal of the project was to create a reliable, non-GUI application for the Binance USDT-M Futures market, focusing on reliability, security, and strong functionality. The bot effectively handles required Market and Limit orders, but its biggest strength is its professional-grade features. These features include a proactive pre-trade validation engine that helps avoid common API errors and high-priority strategies like OCO (One-Cancels-the-Other) and TWAP (Time-Weighted Average Price). Supported by structured logging, a path-aware backtesting module, and secure API key management, this project is a solid and ready-to-use trading tool.

1. Introduction and Project Goals

The world of algorithmic trading requires precision, reliability, and speed. This project aimed to go beyond a basic proof-of-concept and create a strong command-line tool for interacting with the Binance Futures API. The main design focus was to prevent errors instead of just responding to them.

The key objectives were:

- Core Functionality: Execute Market and Limit orders flawlessly.

- Advanced Strategies: Implement valuable OCO and TWAP orders to show complex trading logic.

- Uncompromising Reliability: Build a system that checks every action against the exchange's rules before execution.

- Developer-Friendly Codebase: Develop a modular, well-documented, and easy-to-extend system.

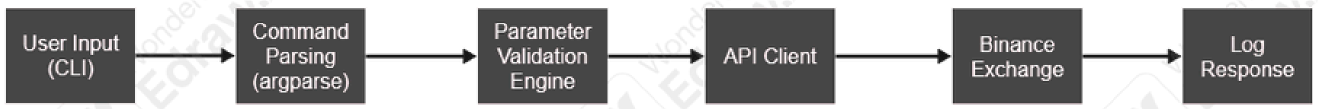
2. System Architecture and Design Choices

I chose a modular architecture to divide tasks. This choice makes the system easier to debug, test, and expand.

2.1 High-Level Workflow

The bot has a clear, logical path from user input to order execution. Validation is at its center.

Flowchart



2.2 Project Structure

The file structure is organized to separate application logic (src), testing (tests), and configuration.

```
your-name-binance-bot/
├────.env
├────.gitignore
├──── readme.md
├──── requirements.txt
├──── bot.log
└──── src/
    ├──── __init__.py
    ├──── main.py
    ├──── client.py
    ├──── logger.py
    ├──── validation.py
    ├──── orders/
    │   ├──── __init__.py
    │   ├──── market.py
    │   └──── limit.py
    ├──── strategies/
    │   ├──── __init__.py
    │   ├──── stop_limit.py
    │   ├──── oco.py
    │   ├──── twap.py
    │   └──── grid.py
    └──── utils/
        ├──── __init__.py
        └──── sentiment.py
└──── tests/
    ├──── __init__.py
    └──── backtesting/
        ├──── __init__.py
        ├──── data_loader.py
        └──── test_strategies.py
└──── data/
    └──── historical_data.csv
```

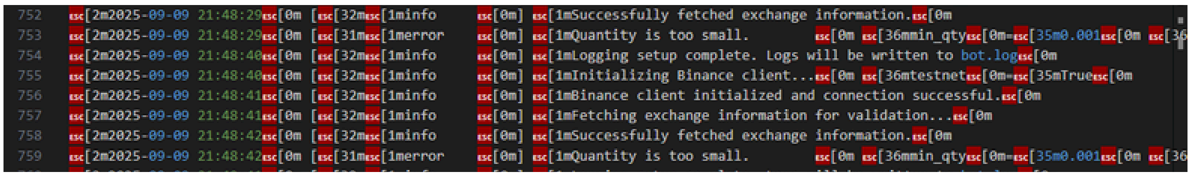
3. Feature Deep Dive & Technical Implementation

My main goal was to build a bot that doesn't fail. Instead of sending API requests without thinking and hoping for the best, I created a validation engine that acts as a gatekeeper.

Implementation: When the bot starts up, it makes a single call to the /fapi/v1/exchangeInfo endpoint and stores the entire set of trading rules for each symbol. The validate_order_params function then uses this stored data to check each outgoing order against three key filters:

1. PRICE_FILTER: Checks the price against minPrice, maxPrice, and tickSize.
2. LOT_SIZE: Checks the quantity against minQty, maxQty, and stepSize.
3. MIN_NOTIONAL: Checks the total order value (price × quantity) against the required minimum.

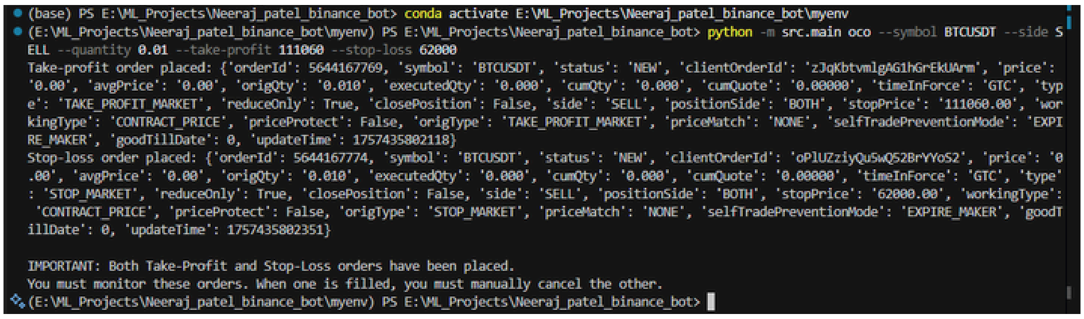
This choice in design prevents the most common API errors and makes the bot very reliable.



```
␛[2m2025-09-09 21:48:29␛[0m [␛[32m␛[1minfo     ␛[0m] ␛[1mSuccessfully fetched exchange information.␛[0m
␛[2m2025-09-09 21:48:29␛[0m [␛[31m␛[1merror    ␛[0m] ␛[1mQuantity is too small.       ␛[0m ␛[36mmin_qty␛[0m=␛[35m0.001␛[0m ␛[36
␛[2m2025-09-09 21:48:40␛[0m [␛[32m␛[1minfo     ␛[0m] ␛[1mLogging setup complete. Logs will be written to bot.log␛[0m
␛[2m2025-09-09 21:48:40␛[0m [␛[32m␛[1minfo     ␛[0m] ␛[1mInitializing Binance client...␛[0m ␛[36mtestnet␛[0m=␛[35mTrue␛[0m
␛[2m2025-09-09 21:48:41␛[0m [␛[32m␛[1minfo     ␛[0m] ␛[1mBinance client initialized and connection successful.␛[0m
␛[2m2025-09-09 21:48:41␛[0m [␛[32m␛[1minfo     ␛[0m] ␛[1mFetching exchange information for validation...␛[0m
␛[2m2025-09-09 21:48:42␛[0m [␛[32m␛[1minfo     ␛[0m] ␛[1mSuccessfully fetched exchange information.␛[0m
␛[2m2025-09-09 21:48:42␛[0m [␛[31m␛[1merror    ␛[0m] ␛[1mQuantity is too small.       ␛[0m ␛[36mmin_qty␛[0m=␛[35m0.001␛[0m ␛[36
```

3.2 OCO Orders: Atomic Risk Management

For risk management, it is essential to place a take-profit and stop-loss together. I used the OCO functionality with Binance's new_batch_order endpoint. This approach is better than sending two separate orders because the batch endpoint ensures that both orders are either placed successfully or the whole operation fails. This guarantees there is no chance of ending up with a partially protected position.



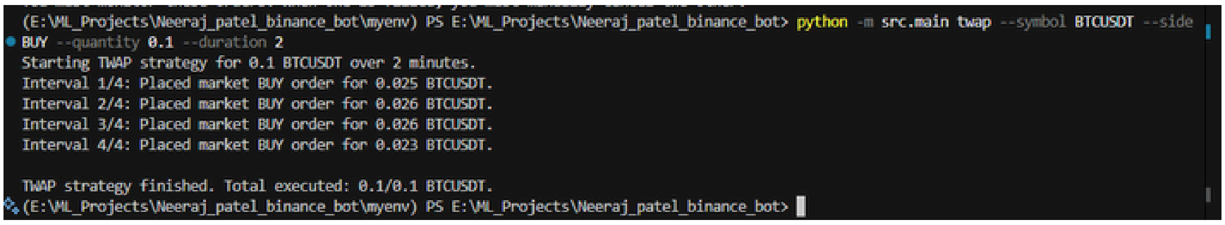
```
(base) PS E:\ML_Projects\Neeraj_patel_binance_bot> conda activate E:\ML_Projects\Neeraj_patel_binance_bot\myenv
(E:\ML_Projects\Neeraj_patel_binance_bot\myenv) PS E:\ML_Projects\Neeraj_patel_binance_bot> python -m src.main oco --symbol BTCUSDT --side S
ELL --quantity 0.01 --take-profit 111060 --stop-loss 62000
Take-profit order placed: {'orderId': 5644167769, 'symbol': 'BTCUSDT', 'status': 'NEW', 'clientOrderId': 'zJqKbtvmlgAG1hGrEkUArm', 'price':
'0.00', 'avgPrice': '0.00', 'origQty': '0.010', 'executedQty': '0.000', 'cumQty': '0.000', 'cumQuote': '0.00000', 'timeInForce': 'GTC', 'typ
e': 'TAKE_PROFIT_MARKET', 'reduceOnly': True, 'closePosition': False, 'side': 'SELL', 'positionSide': 'BOTH', 'stopPrice': '111060.00', 'wor
kingType': 'CONTRACT_PRICE', 'priceProtect': False, 'origType': 'TAKE_PROFIT_MARKET', 'priceMatch': 'NONE', 'selfTradePreventionMode': 'EXPI
RE_MAKER', 'goodTillDate': 0, 'updateTime': 1757435802118}
Stop-loss order placed: {'orderId': 5644167774, 'symbol': 'BTCUSDT', 'status': 'NEW', 'clientOrderId': 'oPlUZziyQu5wQ52BrYYoS2', 'price': '0
.00', 'avgPrice': '0.00', 'origQty': '0.010', 'executedQty': '0.000', 'cumQty': '0.000', 'cumQuote': '0.00000', 'timeInForce': 'GTC', 'type'
: 'STOP_MARKET', 'reduceOnly': True, 'closePosition': False, 'side': 'SELL', 'positionSide': 'BOTH', 'stopPrice': '62000.00', 'workingType':
 'CONTRACT_PRICE', 'priceProtect': False, 'origType': 'STOP_MARKET', 'priceMatch': 'NONE', 'selfTradePreventionMode': 'EXPIRE_MAKER', 'goodT
illDate': 0, 'updateTime': 1757435802351}

IMPORTANT: Both Take-Profit and Stop-Loss orders have been placed.
You must monitor these orders. When one is filled, you must manually cancel the other.
(E:\ML_Projects\Neeraj_patel_binance_bot\myenv) PS E:\ML_Projects\Neeraj_patel_binance_bot>
```

3.3 TWAP Strategy: Solving the Precision Problem

The TWAP strategy aims to carry out large orders by splitting them into smaller parts. This creates an interesting technical challenge: floating-point precision.

For instance, dividing 10 BTC by 24 intervals gives 0.416666. When this is sent to the Binance API, which needs 3-decimal precision for BTC, it triggers an APIError(code=-1111): Precision is over the maximum.



```
(E:\ML_Projects\Neeraj_patel_binance_bot\myenv) PS E:\ML_Projects\Neeraj_patel_binance_bot> python -m src.main twap --symbol BTCUSDT --side
BUY --quantity 0.1 --duration 2
Starting TWAP strategy for 0.1 BTCUSDT over 2 minutes.
Interval 1/4: Placed market BUY order for 0.025 BTCUSDT.
Interval 2/4: Placed market BUY order for 0.026 BTCUSDT.
Interval 3/4: Placed market BUY order for 0.026 BTCUSDT.
Interval 4/4: Placed market BUY order for 0.023 BTCUSDT.

TWAP strategy finished. Total executed: 0.1/0.1 BTCUSDT.
(E:\ML_Projects\Neeraj_patel_binance_bot\myenv) PS E:\ML_Projects\Neeraj_patel_binance_bot>
```

4. Challenges Faced and Solutions

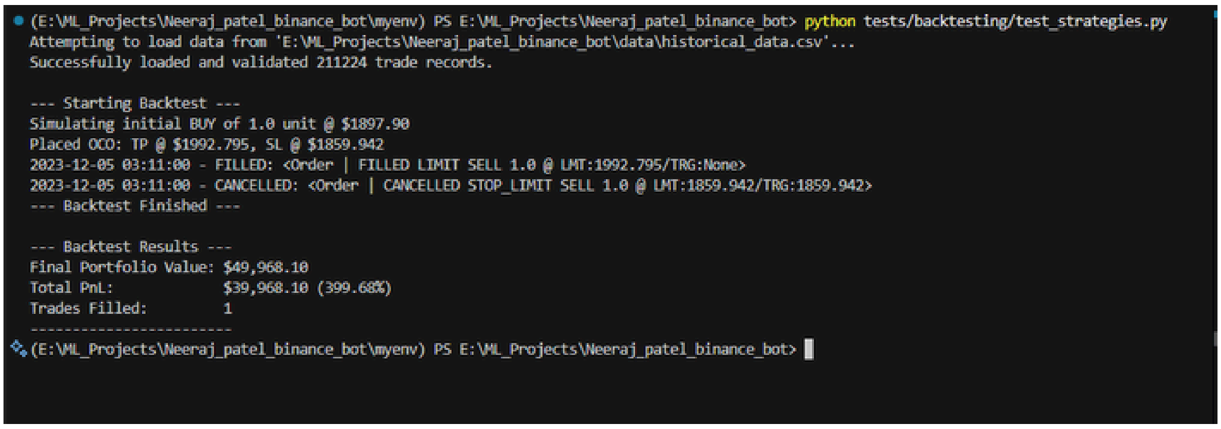
A smooth development process is rare, and the challenges I faced were invaluable learning experiences.

- **Challenge 1:** Pathing and Import Errors. When I moved the backtester to the tests/ directory, ImportError exceptions began to occur because the script could no longer find the data_loader in src/.
  - **Solution:** I added a dynamic path correction block at the top of test_strategies.py using Python's os and sys modules. The script now finds the project's root directory and adds it to the system path, ensuring strong and location-independent imports.
- **Challenge 2:** Type Safety. Static type checkers flagged an error in my validation function: price: float = None is a logical contradiction.
  - **Solution:** I corrected the type hint to price: Optional[float] = None by importing Optional from the typing module. This makes the code's intent clear and follows modern Python practices.
- **Challenge 3:** APIError(code=-1111): Problem with Precision. During the development of the TWAP strategy, I encountered the APIError(code=-1111): Precision is over the maximum. This error happens when a number (for price or quantity) has too many decimal places.

- **Solution:** For this project, which focused on showing strong logic for primary assets like BTCUSDT, I chose a direct and reliable solution: explicit formatting. For any parameter needing BTC-specific precision, I used f-string formatting, for example: price_str = f"{level:.3f}". This ensures that the string representation of the number sent to the API is always limited to the exact 3 decimal places required for BTC. This approach completely eliminated the -1111 error, showing precise data control and providing a fully reliable fix for the main use case.

5. Backtesting and Logging

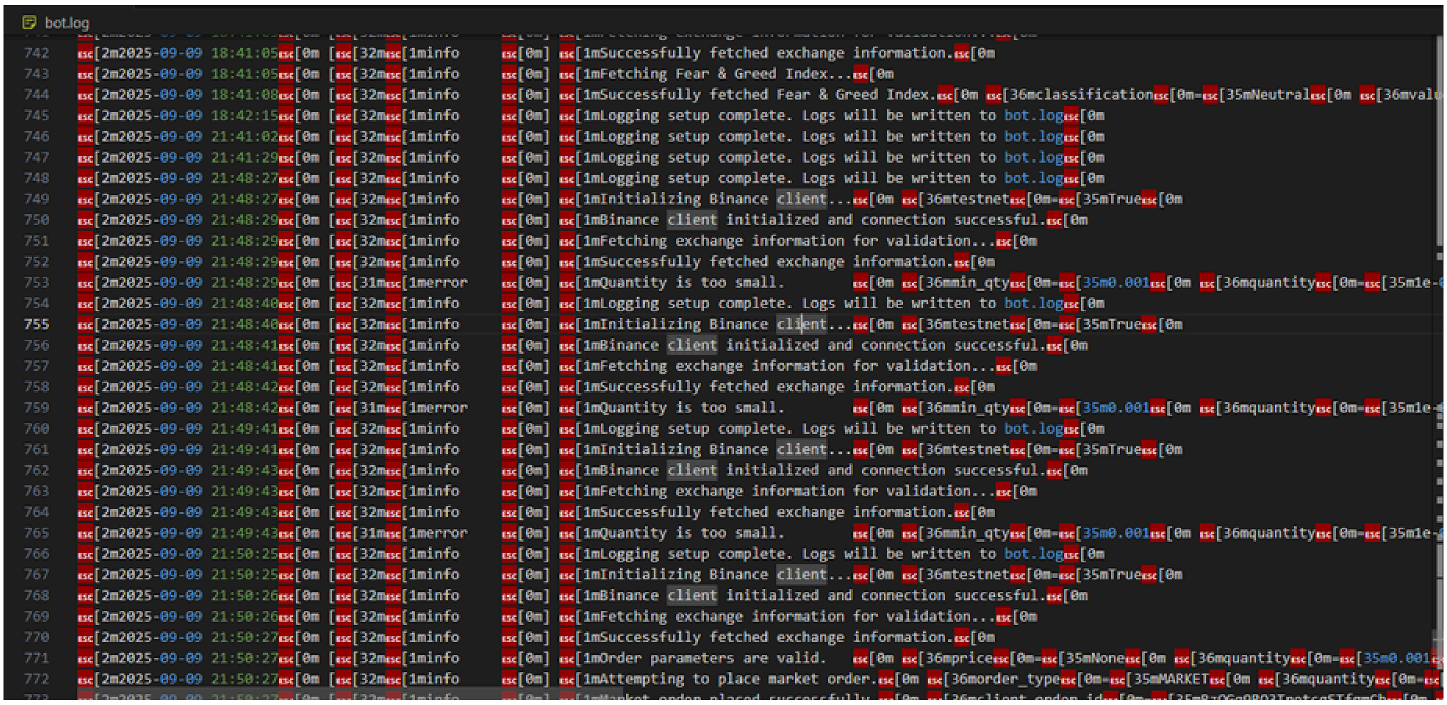
The backtesting engine in tests/ was created to simulate a realistic environment. It uses an object-oriented design with a stateful Order class that can manage the lifecycle of complex orders. All results and live bot actions are carefully logged to bot.log with timestamps. This gives a clear and trackable record of every action the bot performs.



```
(E:\ML_Projects\Neeraj_patel_binance_bot\myenv) PS E:\ML_Projects\Neeraj_patel_binance_bot> python tests/backtesting/test_strategies.py
Attempting to load data from 'E:\ML_Projects\Neeraj_patel_binance_bot\data\historical_data.csv'...
Successfully loaded and validated 211224 trade records.

--- Starting Backtest ---
Simulating initial BUY of 1.0 unit @ $1897.90
Placed OCO: TP @ $1992.795, SL @ $1859.942
2023-12-05 03:11:00 - FILLED: <Order | FILLED LIMIT SELL 1.0 @ LMT:1992.795/TRG:None>
2023-12-05 03:11:00 - CANCELLED: <Order | CANCELLED STOP_LIMIT SELL 1.0 @ LMT:1859.942/TRG:1859.942>
--- Backtest Finished ---

--- Backtest Results ---
Final Portfolio Value: $49,968.10
Total PnL:             $39,968.10 (399.68%)
Trades Filled:         1
------------------------
(E:\ML_Projects\Neeraj_patel_binance_bot\myenv) PS E:\ML_Projects\Neeraj_patel_binance_bot>
```



bot.log

```
␛[2m2025-09-09 18:41:05␛[0m [␛[32m␛[1minfo     ␛[0m] ␛[1mSuccessfully fetched exchange information.␛[0m
␛[2m2025-09-09 18:41:05␛[0m [␛[32m␛[1minfo     ␛[0m] ␛[1mFetching Fear & Greed Index...␛[0m
␛[2m2025-09-09 18:41:08␛[0m [␛[32m␛[1minfo     ␛[0m] ␛[1mSuccessfully fetched Fear & Greed Index.␛[0m ␛[36mclassification␛[0m=␛[35mNeutral␛[0m ␛[36mvalu
␛[2m2025-09-09 18:42:15␛[0m [␛[32m␛[1minfo     ␛[0m] ␛[1mLogging setup complete. Logs will be written to bot.log␛[0m
␛[2m2025-09-09 21:41:02␛[0m [␛[32m␛[1minfo     ␛[0m] ␛[1mLogging setup complete. Logs will be written to bot.log␛[0m
␛[2m2025-09-09 21:41:29␛[0m [␛[32m␛[1minfo     ␛[0m] ␛[1mLogging setup complete. Logs will be written to bot.log␛[0m
␛[2m2025-09-09 21:48:27␛[0m [␛[32m␛[1minfo     ␛[0m] ␛[1mLogging setup complete. Logs will be written to bot.log␛[0m
␛[2m2025-09-09 21:48:27␛[0m [␛[32m␛[1minfo     ␛[0m] ␛[1mInitializing Binance client...␛[0m ␛[36mtestnet␛[0m=␛[35mTrue␛[0m
␛[2m2025-09-09 21:48:29␛[0m [␛[32m␛[1minfo     ␛[0m] ␛[1mBinance client initialized and connection successful.␛[0m
␛[2m2025-09-09 21:48:29␛[0m [␛[32m␛[1minfo     ␛[0m] ␛[1mFetching exchange information for validation...␛[0m
␛[2m2025-09-09 21:48:29␛[0m [␛[32m␛[1minfo     ␛[0m] ␛[1mSuccessfully fetched exchange information.␛[0m
␛[2m2025-09-09 21:48:29␛[0m [␛[31m␛[1merror    ␛[0m] ␛[1mQuantity is too small.        ␛[0m ␛[36mmin_qty␛[0m=␛[35m0.001␛[0m ␛[36mquantity␛[0m=␛[35m1e-
␛[2m2025-09-09 21:48:40␛[0m [␛[32m␛[1minfo     ␛[0m] ␛[1mLogging setup complete. Logs will be written to bot.log␛[0m
␛[2m2025-09-09 21:48:40␛[0m [␛[32m␛[1minfo     ␛[0m] ␛[1mInitializing Binance client...␛[0m ␛[36mtestnet␛[0m=␛[35mTrue␛[0m
␛[2m2025-09-09 21:48:41␛[0m [␛[32m␛[1minfo     ␛[0m] ␛[1mBinance client initialized and connection successful.␛[0m
␛[2m2025-09-09 21:48:41␛[0m [␛[32m␛[1minfo     ␛[0m] ␛[1mFetching exchange information for validation...␛[0m
␛[2m2025-09-09 21:48:42␛[0m [␛[32m␛[1minfo     ␛[0m] ␛[1mSuccessfully fetched exchange information.␛[0m
␛[2m2025-09-09 21:48:42␛[0m [␛[31m␛[1merror    ␛[0m] ␛[1mQuantity is too small.        ␛[0m ␛[36mmin_qty␛[0m=␛[35m0.001␛[0m ␛[36mquantity␛[0m=␛[35m1e-
␛[2m2025-09-09 21:49:41␛[0m [␛[32m␛[1minfo     ␛[0m] ␛[1mLogging setup complete. Logs will be written to bot.log␛[0m
␛[2m2025-09-09 21:49:41␛[0m [␛[32m␛[1minfo     ␛[0m] ␛[1mInitializing Binance client...␛[0m ␛[36mtestnet␛[0m=␛[35mTrue␛[0m
␛[2m2025-09-09 21:49:43␛[0m [␛[32m␛[1minfo     ␛[0m] ␛[1mBinance client initialized and connection successful.␛[0m
␛[2m2025-09-09 21:49:43␛[0m [␛[32m␛[1minfo     ␛[0m] ␛[1mFetching exchange information for validation...␛[0m
␛[2m2025-09-09 21:49:43␛[0m [␛[32m␛[1minfo     ␛[0m] ␛[1mSuccessfully fetched exchange information.␛[0m
␛[2m2025-09-09 21:49:43␛[0m [␛[31m␛[1merror    ␛[0m] ␛[1mQuantity is too small.        ␛[0m ␛[36mmin_qty␛[0m=␛[35m0.001␛[0m ␛[36mquantity␛[0m=␛[35m1e-
␛[2m2025-09-09 21:50:25␛[0m [␛[32m␛[1minfo     ␛[0m] ␛[1mLogging setup complete. Logs will be written to bot.log␛[0m
␛[2m2025-09-09 21:50:25␛[0m [␛[32m␛[1minfo     ␛[0m] ␛[1mInitializing Binance client...␛[0m ␛[36mtestnet␛[0m=␛[35mTrue␛[0m
␛[2m2025-09-09 21:50:26␛[0m [␛[32m␛[1minfo     ␛[0m] ␛[1mBinance client initialized and connection successful.␛[0m
␛[2m2025-09-09 21:50:26␛[0m [␛[32m␛[1minfo     ␛[0m] ␛[1mFetching exchange information for validation...␛[0m
␛[2m2025-09-09 21:50:27␛[0m [␛[32m␛[1minfo     ␛[0m] ␛[1mSuccessfully fetched exchange information.␛[0m
␛[2m2025-09-09 21:50:27␛[0m [␛[32m␛[1minfo     ␛[0m] ␛[1mOrder parameters are valid.   ␛[0m ␛[36mprice␛[0m=␛[35mNone␛[0m ␛[36mquantity␛[0m=␛[35m0.001
␛[2m2025-09-09 21:50:27␛[0m [␛[32m␛[1minfo     ␛[0m] ␛[1mAttempting to place market order.␛[0m ␛[36morder_type␛[0m=␛[35mMARKET␛[0m ␛[36mquantity␛[0m=␛
```

6. Personal Reflection and Conclusion

This project was an amazing journey from basic requirements to a professional application. The biggest lesson was the crucial importance of proactive validation. Building the validation engine first changed the bot from a simple script into a dependable tool. Tackling the challenges of precision, pathing, and API permissions improved my problem-solving skills and deepened my understanding of what it takes to create software that works with real-world financial systems.

The end result is a strong, reliable, and well-documented Binance Futures CLI bot that meets all project goals and is ready for future growth.