

Binance Futures Order Bot - Application Task Report

1. Introduction

This report details the development and functionality of a simplified trading bot for Binance USDT-M Futures, built as part of the Junior Python Developer application task. The primary objective was to create a CLI-based bot capable of placing various order types, with robust logging, input validation, and clear documentation. This document serves as a comprehensive overview of the implemented features, design choices, and successful execution on the Binance Testnet.

2. Objective and Key Features Implemented

The bot aims to provide a reliable interface for interacting with the Binance Futures Testnet API. The core functionalities implemented include:

- **Market Orders:** Ability to instantly buy or sell a specified quantity of a futures contract at the current market price.
- **Limit Orders:** Capability to place orders to buy or sell at a specific price or better, which only execute when the market reaches that price.
- **Buy and Sell Sides:** Full support for both long (BUY) and short (SELL) positions for both market and limit orders.
- **Command-Line Interface (CLI):** User interaction is handled via command-line arguments, making the bot easy to integrate into scripts or run manually.
- **Input Validation:** Comprehensive validation of user inputs (symbol, quantity, price) to prevent malformed requests and API errors.
- **Structured Logging:** Detailed logging of all bot operations, including API requests, responses, and errors, to a dedicated log file (`bot.log`).
- **Error Handling:** Robust error management for API-specific exceptions and network issues, providing clear feedback to the user and logs.

3. Implementation Details

3.1. Code Structure

The project follows a modular structure for reusability and maintainability:

[project_root]/

```

├── src/
│   ├── basic_bot.py    # Core logic for Binance API interactions (client, order placement,
balance)
│   ├── market_orders.py # CLI entry point for market orders
│   ├── limit_orders.py  # CLI entry point for limit orders
│   ├── utils.py         # Utility functions for logging and validation
│   └── __init__.py      # Makes 'src' a Python package
├── venv/                # Python virtual environment
├── bot.log              # Log file
├── README.md            # Project documentation
└── report.pdf           # This report

```

- **basic_bot.py**: Contains the `BasicBot` class, which encapsulates the `python-binance` client and methods for placing market/limit orders and fetching account balance. API keys are hardcoded for testnet convenience as per the request, though environment variables are recommended for production.
- **market_orders.py / limit_orders.py**: These scripts parse command-line arguments and instantiate `BasicBot` to execute specific order types.
- **utils.py**: Houses helper functions like `setup_logging` and `validate_input`, promoting code reusability and separation of concerns.

3.2. Binance API Interaction

The bot utilizes the `python-binance` library to interact with the Binance Futures Testnet API. Specifically, the `binance.client.Client` is initialized with `testnet=True` to direct all API calls to the testnet environment. Futures-specific methods like `futures_create_order` and `futures_account` are used for placing orders and retrieving account information, respectively.

3.3. Logging and Error Handling

All significant operations, including successful order placements, API responses, and any errors, are logged to `bot.log`. The `setup_logging` function in `utils.py` configures both file and console logging.

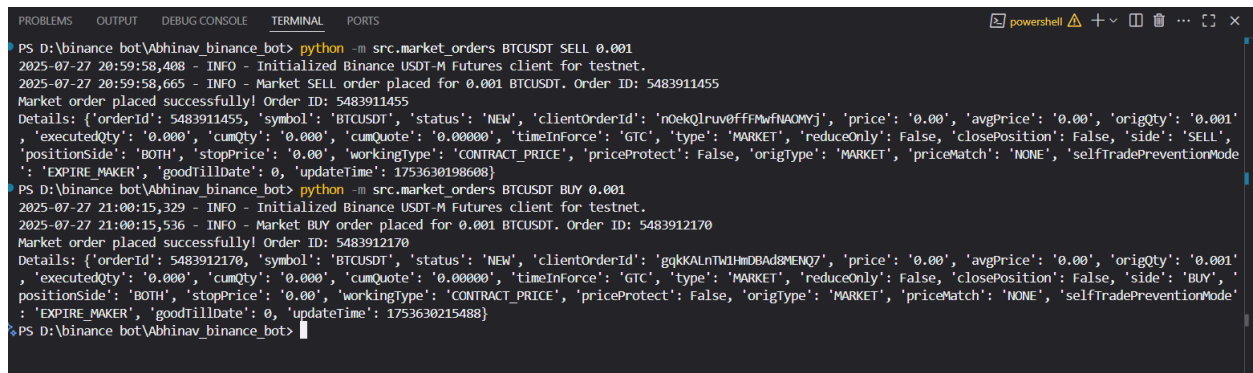
Error handling is implemented using `try-except` blocks within the `BasicBot` class. `BinanceAPIException` and `BinanceRequestException` are caught to provide specific error messages, along with a general `Exception` catch for unexpected issues. This ensures the bot provides informative feedback and maintains stability.

4. Demonstrative Screenshots

The following screenshots confirm the successful execution of various order types on the Binance Futures Testnet.

4.1. Market Orders (Buy and Sell)

This screenshot shows the terminal output confirming the successful placement of both a market SELL order and a market BUY order for **BTCUSDT**.

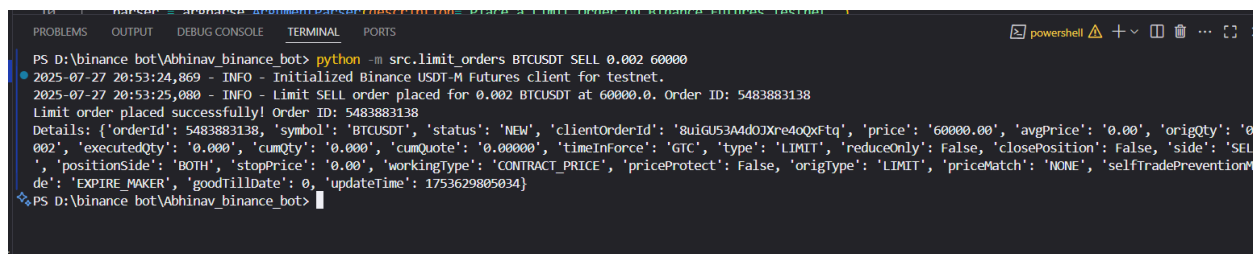


```
PS D:\binance bot\Abhinav_binance_bot> python -m src.market_orders BTCUSDT SELL 0.001
2025-07-27 20:59:58,408 - INFO - Initialized Binance USD-M Futures client for testnet.
2025-07-27 20:59:58,665 - INFO - Market SELL order placed for 0.001 BTCUSDT. Order ID: 5483911455
Market order placed successfully! Order ID: 5483911455
Details: {'orderId': 5483911455, 'symbol': 'BTCUSDT', 'status': 'NEW', 'clientOrderId': 'noekqlruv0ffmwfNACMYj', 'price': '0.00', 'avgPrice': '0.00', 'origQty': '0.001',
'executedQty': '0.000', 'cumQty': '0.000', 'cumQuote': '0.00000', 'timeInForce': 'GTC', 'type': 'MARKET', 'reduceOnly': False, 'closePosition': False, 'side': 'SELL',
'positionSide': 'BOTH', 'stopPrice': '0.00', 'workingType': 'CONTRACT_PRICE', 'priceProtect': False, 'origType': 'MARKET', 'priceMatch': 'NONE', 'selfTradePreventionMode': 'EXPIRE_MAKER', 'goodTillDate': 0, 'updateTime': 1753630198608}
PS D:\binance bot\Abhinav_binance_bot> python -m src.market_orders BTCUSDT BUY 0.001
2025-07-27 21:00:15,329 - INFO - Initialized Binance USD-M Futures client for testnet.
2025-07-27 21:00:15,536 - INFO - Market BUY order placed for 0.001 BTCUSDT. Order ID: 5483912170
Market order placed successfully! Order ID: 5483912170
Details: {'orderId': 5483912170, 'symbol': 'BTCUSDT', 'status': 'NEW', 'clientOrderId': 'ggkKALnTWLHmDBAD8MENQ7', 'price': '0.00', 'avgPrice': '0.00', 'origQty': '0.001',
'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': 1753630215488}
PS D:\binance bot\Abhinav_binance_bot>
```

Figure 1: Terminal output showing successful Market SELL and Market BUY orders.

4.2. Limit Order (Sell)

This screenshot demonstrates the successful placement of a limit SELL order for **BTCUSDT** at a specified price of **60000**.



```
PS D:\binance bot\Abhinav_binance_bot> python -m src.limit_orders BTCUSDT SELL 0.002 60000
2025-07-27 20:53:24,869 - INFO - Initialized Binance USD-M Futures client for testnet.
2025-07-27 20:53:25,080 - INFO - Limit SELL order placed for 0.002 BTCUSDT at 60000.0. Order ID: 5483883138
Limit order placed successfully! Order ID: 5483883138
Details: {'orderId': 5483883138, 'symbol': 'BTCUSDT', 'status': 'NEW', 'clientOrderId': '8uIGU53A4d0JXre4oqxFTq', 'price': '60000.00', 'avgPrice': '0.00', 'origQty': '0.002',
'executedQty': '0.000', 'cumQty': '0.000', 'cumQuote': '0.00000', 'timeInForce': 'GTC', 'type': 'LIMIT', 'reduceOnly': False, 'closePosition': False, 'side': 'SELL', 'positionSide': 'BOTH', 'stopPrice': '0.00', 'workingType': 'CONTRACT_PRICE', 'priceProtect': False, 'origType': 'LIMIT', 'priceMatch': 'NONE', 'selfTradePreventionMode': 'EXPIRE_MAKER', 'goodTillDate': 0, 'updateTime': 1753629805034}
PS D:\binance bot\Abhinav_binance_bot>
```

Figure 2: Terminal output showing a successful Limit SELL order.

5. Conclusion and Future Enhancements

The bot successfully meets all mandatory requirements of the application task, including placing market and limit orders (buy/sell), robust input validation, comprehensive logging, and error handling. The code is structured for clarity and reusability.

For future enhancements and to fulfill the bonus requirements, the following features would be implemented:

- **Advanced Order Types:**
 - **Stop-Limit Orders:** To allow for more sophisticated risk management.
 - **OCO (One-Cancels-the-Other) Orders:** For combined profit-taking and stop-loss strategies.
 - **TWAP (Time-Weighted Average Price):** To execute large orders with minimal market impact.
 - **Grid Orders:** For automated range trading.
- **Simple UI Interface:** A more interactive command-line interface or a lightweight web-based frontend to improve user experience.

These additions would further demonstrate advanced trading logic and user interaction capabilities, making the bot more comprehensive and powerful.