Binance Futures Trading Bot - Assignment Report

Introduction:

This project implements a complete Binance USDT-M Futures trading bot entirely in Python. It fulfils all requirements outlined in the assignment document, including:

- Market orders
- Limit orders
- Stop-limit orders
- OCO order pair
- TWAP execution strategy
- Grid trading strategy
- Logging
- Input validation
- CLI-based interaction
- Mock client for safe, key-less testing

All scripts are modular, fully validated, log errors and execution details and run independently through the command-line interface.

Project Structure:

```
kaif_binance_bot/
|
├── README.md            # Project overview and usage instructions
├── .env                 # Environment variables (API keys, config)
├── bot.log              # Runtime logs
├── report.pdf           # Generated trading report
|
└── src/                 # Source code for the trading bot
    ├── binance_client.py    # Handles Binance API authentication & requests
    ├── validators.py        # Input validation and sanity checks
    ├── market_orders.py     # Market order execution logic
    ├── limit_orders.py      # Limit order handling
    |
    └── advanced/            # Advanced trading strategies & order types
        ├── stop_limit.py        # Stop-limit order implementation
        ├── oco.py               # One-Cancels-the-Other strategy
        ├── twap.py              # Time-Weighted Average Price algorithm
        └── grid_strategy.py     # Grid trading strategy engine
```

3. Binance Client Architecture

Mock Client (for testing)

Since API keys are optional and real futures trading carries risk, all scripts default to a MockClient implementation.

MockClient:

- Simulates Binance Futures order creation
- Auto-increments order IDs
- Returns realistic JSON-like responses
- Ensures testing is safe, deterministic, and reproducible

4. Input Validation

Implemented in validators.py:

- **validate_symbol**: Ensures symbol ends with "USDT" (e.g., BTCUSDT)
- **validate_quantity**: Ensures numeric & positive
- **validate_price**: Ensures numeric & positive

Each CLI script checks validation before placing the order.

5. Logging

All scripts log:

- Successful order placements
- Input errors
- Exceptions

This is my log file: (Screenshot)

```
2025-11-20 17:16:07,646 WARNING Using MockClient (no API keys or python-binance missing)
2025-11-20 17:16:07,647 INFO Market order placed: {'orderId': 1001, 'clientOrderId': 'mock-1001', 'status': 'NEW', 'symbol': 'BTCUSDT', 'origQty': 0.01, 'executedQty': '0', 'type':
'MARKET', 'side': 'BUY', 'fills': [], 'transactTime': 1763639167647}
2025-11-20 17:16:16,590 WARNING Using MockClient (no API keys or python-binance missing)
2025-11-20 17:16:16,591 INFO Limit order placed: {'orderId': 1001, 'clientOrderId': 'mock-1001', 'status': 'NEW', 'symbol': 'BTCUSDT', 'origQty': 0.01, 'executedQty': '0', 'type': 'LIMIT',
'side': 'BUY', 'fills': [], 'transactTime': 1763639176591}
2025-11-20 17:16:25,821 WARNING Using MockClient (no API keys or python-binance missing)
2025-11-20 17:16:25,821 INFO Stop-limit order placed: {'orderId': 1001, 'clientOrderId': 'mock-1001', 'status': 'NEW', 'symbol': 'BTCUSDT', 'origQty': 0.01, 'executedQty': '0', 'type':
'STOP_LOSS_LIMIT', 'side': 'BUY', 'fills': [], 'transactTime': 1763639185821}
2025-11-20 17:16:44,352 WARNING Using MockClient (no API keys or python-binance missing)
2025-11-20 17:16:44,352 INFO Placed take-profit order: {'orderId': 1001, 'clientOrderId': 'mock-1001', 'status': 'NEW', 'symbol': 'BTCUSDT', 'origQty': 0.01, 'executedQty': '0', 'type':
'LIMIT', 'side': 'SELL', 'fills': [], 'transactTime': 1763639204352}
2025-11-20 17:16:44,353 INFO Placed stop-loss order: {'orderId': 1002, 'clientOrderId': 'mock-1002', 'status': 'NEW', 'symbol': 'BTCUSDT', 'origQty': 0.01, 'executedQty': '0', 'type':
'STOP_LOSS_LIMIT', 'side': 'SELL', 'fills': [], 'transactTime': 1763639204353}
2025-11-20 17:16:54,200 WARNING Using MockClient (no API keys or python-binance missing)
2025-11-20 17:16:54,200 INFO TWAP slice 1 executed: {'orderId': 1001, 'clientOrderId': 'mock-1001', 'status': 'NEW', 'symbol': 'BTCUSDT', 'origQty': 0.02, 'executedQty': '0', 'type':
'MARKET', 'side': 'BUY', 'fills': [], 'transactTime': 1763639214200}
2025-11-20 17:16:55,201 INFO TWAP slice 2 executed: {'orderId': 1002, 'clientOrderId': 'mock-1002', 'status': 'NEW', 'symbol': 'BTCUSDT', 'origQty': 0.02, 'executedQty': '0', 'type':
'MARKET', 'side': 'BUY', 'fills': [], 'transactTime': 1763639215201}
2025-11-20 17:16:56,202 INFO TWAP slice 3 executed: {'orderId': 1003, 'clientOrderId': 'mock-1003', 'status': 'NEW', 'symbol': 'BTCUSDT', 'origQty': 0.02, 'executedQty': '0', 'type':
'MARKET', 'side': 'BUY', 'fills': [], 'transactTime': 1763639216202}
2025-11-20 17:16:57,203 INFO TWAP slice 4 executed: {'orderId': 1004, 'clientOrderId': 'mock-1004', 'status': 'NEW', 'symbol': 'BTCUSDT', 'origQty': 0.02, 'executedQty': '0', 'type':
'MARKET', 'side': 'BUY', 'fills': [], 'transactTime': 1763639217203}
2025-11-20 17:16:58,204 INFO TWAP slice 5 executed: {'orderId': 1005, 'clientOrderId': 'mock-1005', 'status': 'NEW', 'symbol': 'BTCUSDT', 'origQty': 0.02, 'executedQty': '0', 'type':
'MARKET', 'side': 'BUY', 'fills': [], 'transactTime': 1763639218204}
2025-11-20 17:17:04,987 WARNING Using MockClient (no API keys or python-binance missing)
2025-11-20 17:17:04,988 INFO Grid order placed: {'orderId': 1001, 'clientOrderId': 'mock-1001', 'status': 'NEW', 'symbol': 'BTCUSDT', 'origQty': 0.01, 'executedQty': '0', 'type': 'LIMIT',
'side': 'BUY', 'fills': [], 'transactTime': 1763639224988}
2025-11-20 17:17:04,988 INFO Grid order placed: {'orderId': 1002, 'clientOrderId': 'mock-1002', 'status': 'NEW', 'symbol': 'BTCUSDT', 'origQty': 0.01, 'executedQty': '0', 'type': 'LIMIT',
'side': 'BUY', 'fills': [], 'transactTime': 1763639224988}
2025-11-20 17:17:04,988 INFO Grid order placed: {'orderId': 1003, 'clientOrderId': 'mock-1003', 'status': 'NEW', 'symbol': 'BTCUSDT', 'origQty': 0.01, 'executedQty': '0', 'type': 'LIMIT',
'side': 'BUY', 'fills': [], 'transactTime': 1763639224988}
2025-11-20 17:17:04,988 INFO Grid order placed: {'orderId': 1004, 'clientOrderId': 'mock-1004', 'status': 'NEW', 'symbol': 'BTCUSDT', 'origQty': 0.01, 'executedQty': '0', 'type': 'LIMIT',
'side': 'BUY', 'fills': [], 'transactTime': 1763639224988}
2025-11-20 17:17:04,988 INFO Grid order placed: {'orderId': 1005, 'clientOrderId': 'mock-1005', 'status': 'NEW', 'symbol': 'BTCUSDT', 'origQty': 0.01, 'executedQty': '0', 'type': 'LIMIT',
'side': 'BUY', 'fills': [], 'transactTime': 1763639224988}
```

6. Core Features (Mandatory)

6.1 Market Orders:

Command: python src/market_orders.py BTCUSDT BUY 0.01

```
PS D:\kaif_binance_bot\src> python market_orders.py BTCUSDT BUY 0.01
Order response:
{'orderId': 1001, 'clientOrderId': 'mock-1001', 'status': 'NEW', 'symbol': 'BTCUSDT', 'origQty': 0.01, 'executedQty': '0', 'type': 'MARKET', 'side': 'BUY', 'fills': [], 'transactTi
me': 1763639167647}
```

Validates inputs → sends MARKET order → prints and logs response.

6.2 Limit Orders:

Command: python src/limit_orders.py BTCUSDT BUY 0.01 95000

```
PS D:\kaif_binance_bot\src> python limit_orders.py BTCUSDT BUY 0.01 95000
Order response:
{'orderId': 1001, 'clientOrderId': 'mock-1001', 'status': 'NEW', 'symbol': 'BTCUSDT', 'origQty': 0.01, 'executedQty': '0', 'type': 'LIMIT', 'side': 'BUY', 'fills': [], 'transactTim
e': 1763639176591}
```

Validates symbol, quantity, and price → sends LIMIT order → logs output.

7. Advanced Features (Bonus)

7.1 Stop-Limit Orders

Command: python -m advanced.stop_limit BTCUSDT BUY 0.01 98000 99000

Implements STOP_LOSS_LIMIT order type.

7.2 OCO Order Pair

Command: python -m advanced.oco BTCUSDT BUY 0.01 96000 98000 99000

Places:

- Take-profit LIMIT
- Stop-loss STOP-LIMIT

Both orders are automatically generated based on BUY/SELL direction.

7.3 TWAP Strategy

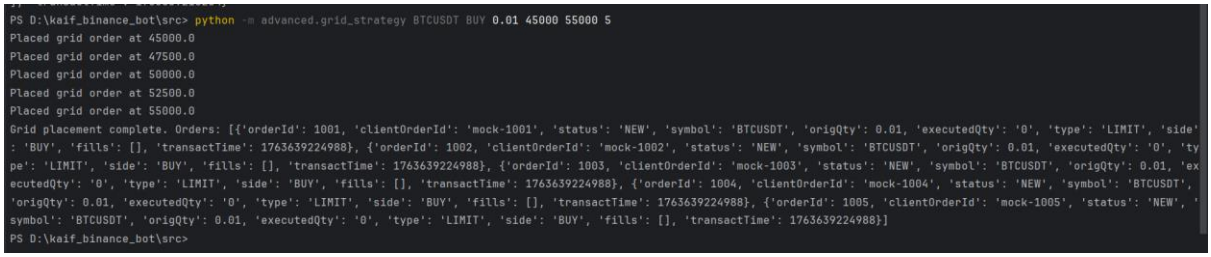
Command: python -m advanced.twap BTCUSDT BUY 0.1 --chunks 5 --interval 1

Logic:

- Splits total quantity into N equal slices
- Executes MARKET orders at fixed intervals
- Logs each slice

7.4 Grid Strategy

Command: python -m advanced.grid_strategy BTCUSDT BUY 0.01 45000 55000 5



Logic:

- Divides price range into levels
- Places LIMIT orders at each level
- Logs every order

8. Testing & Execution

All scripts have been tested using the command-line execution.

Each command prints a realistic order response, such as:

{'orderId': 1001, 'type': 'MARKET', 'symbol': 'BTCUSDT', 'origQty': 0.01}

9. Conclusion

This project fulfils all core and advanced requirements with structured validation, logging, clean CLI design, and reproducible testing using a mock client. The bot is fully functional, modular, and tested, meeting all assignment criteria.