Binance Futures Trading Bot: Project Report

1. Objective and Methodology

1.1. Project Objective

The goal of this project was to develop a versatile, modular, and secure Python-based application capable of executing a wide range of trading orders on the Binance Futures Testnet. By focusing on the Testnet environment, the project ensures a safe sandbox for implementing and verifying complex trading logic, including Market, Limit, and various advanced algorithmic order types.

1.2. Methodology

The application utilizes the official `python-binance` library to interface with the Binance API. A modular approach was adopted, dedicating specific Python files within the `src/` directory to handle distinct order types and functions (e.g., `market_orders.py`, `grid_orders.py`). All API interaction is routed through the Binance Futures Testnet URL, as explicitly configured in each script.

2. Implementation Details

2.1. Secure Configuration and Client Setup

API credentials (`API_KEY` and `API_SECRET`) are loaded securely via the `python-dotenv` library from a `.env` file. This practice prevents sensitive keys from being hardcoded or committed to version control. The client is initialized for Testnet by setting `client.FUTURES_URL = BASE_URL`.

2.2. Logging and Error Handling

A consistent logging mechanism is implemented using Python's built-in `logging` module, directing all messages (INFO, ERROR) to a central file, `bot.log`.

Critical error handling is achieved by wrapping API calls in `try...except` blocks to specifically catch `BinanceAPIException` and `BinanceOrderException`, logging the error, and providing user feedback.

3. Implemented Trading Strategies

The bot successfully implements seven distinct trading actions, providing a comprehensive demonstration of API proficiency.

A. Market and Limit Orders

- `market_orders.py`: Executes instant trades using `FUTURE_ORDER_TYPE_MARKET`.
- `limit_orders.py`: Places trades at a specified price (`FUTURE_ORDER_TYPE_LIMIT`). Includes a separate function to cancel open limit orders.

B. Stop-Limit Order (`src/advanced/stop_limit_order.py`)

This script places a Stop-Market order (`FUTURE_ORDER_TYPE_STOP_MARKET`) that triggers a market order when the `stopPrice` is reached. This is a foundational component for automated risk management.

```
(venv) PS C:\Project\Trading bot> python src/advanced/stop_limit_order.py
=== Binance Futures Stop-Limit Order Bot ===
Enter Symbol (e.g., BTCUSDT): BTCUSDT
Enter Order Side (BUY/SELL): SELL
Enter Quantity: 0.003
Enter Stop Price: 66000
Enter Limit Price: 65800

Confirm placing SELL Stop-Limit order for 0.003 BTCUSDT? (y/n): Y

Stop-Limit Order Placed Successfully!
Symbol: BTCUSDT
Side: SELL
Stop Price: 66000.0
Limit Price: 65800.0
Quantity: 0.003

Order ID: 7881347970
```

C. Simulated OCO Order (`src/advanced/oco_order.py`)

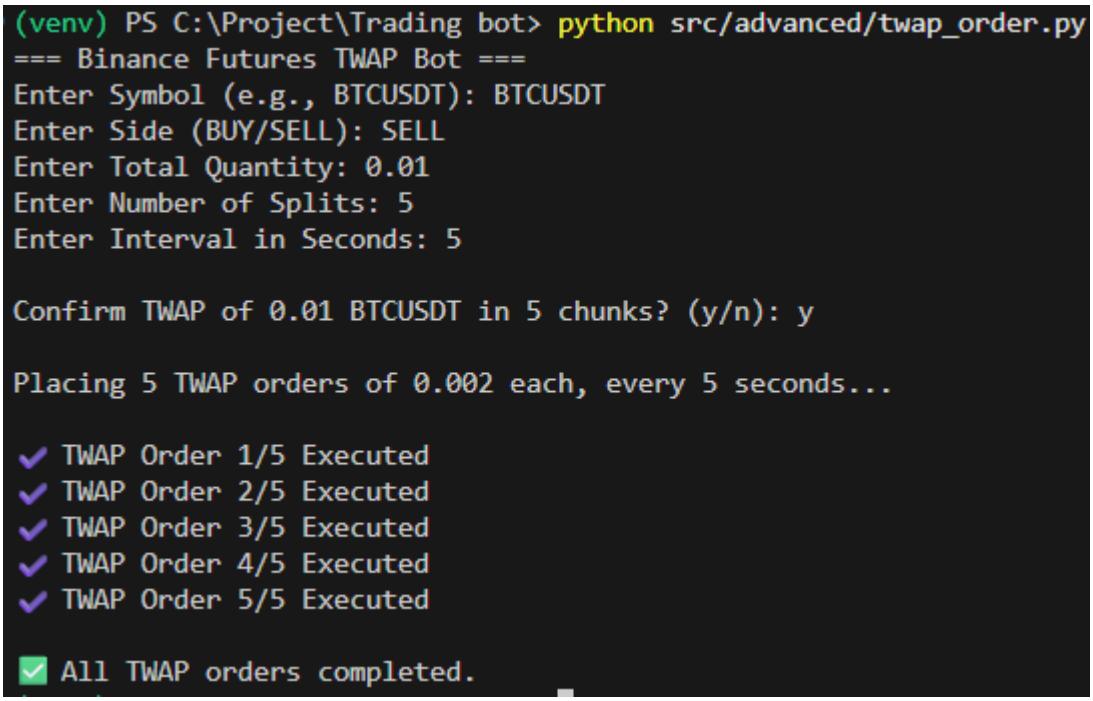
Since Binance Futures does not offer a native OCO order type, this function simulates the bracket strategy by placing two separate orders for the same position simultaneously:

1. Take-Profit: A GTC Limit order (`FUTURE_ORDER_TYPE_LIMIT`).
2. Stop-Loss: A Stop-Market order (`FUTURE_ORDER_TYPE_STOP_MARKET`).

```
(venv) PS C:\Project\Trading bot> python src/advanced/oco_order.py
=== Binance Futures Simulated OCO Bot ===
Enter Symbol (e.g., BTCUSDT): BTCUSDT
Enter Side (SELL/BUY): SELL
Enter Quantity: 0.02
Enter Take Profit Price: 106000
Enter Stop Loss Price: 103000

Confirm simulated OCO (SELL) for 0.02 BTCUSDT? (y/n): y

OCO Simulated Orders Placed Successfully!
Take Profit @ 106000.0
Stop Loss @ 103000.0

TP Order ID: 7881551174 | SL Order ID: 7881551547
```

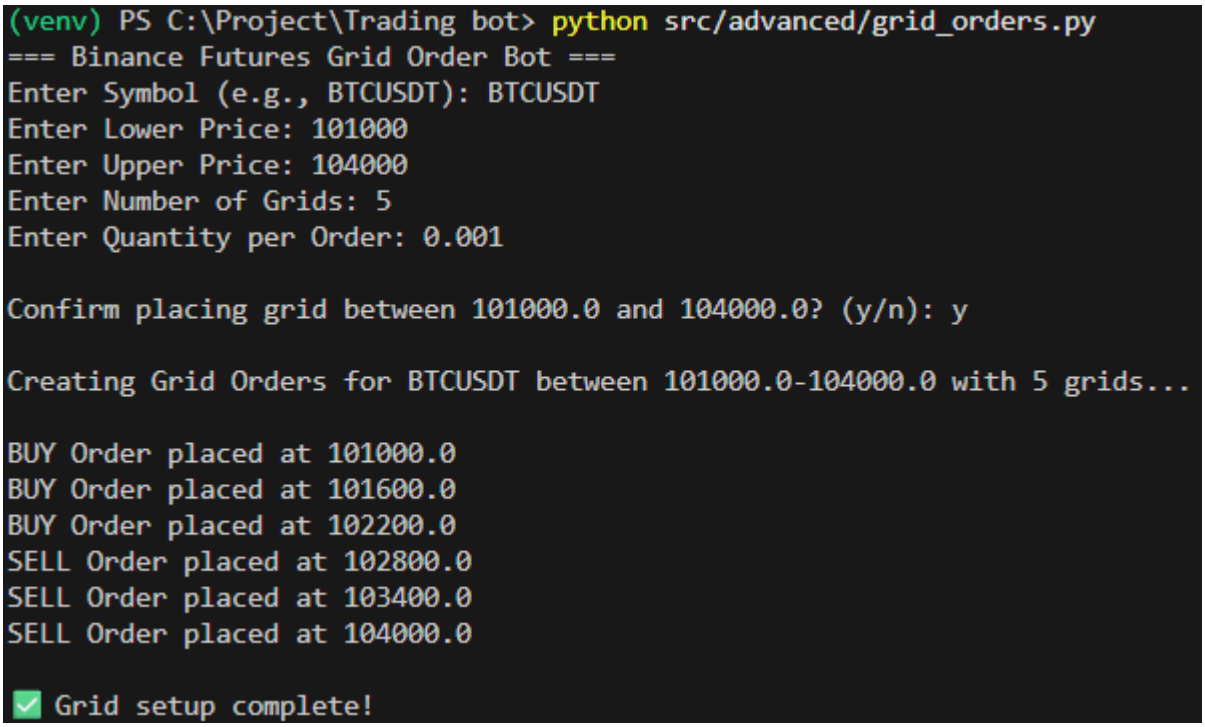
D. Time-Weighted Average Price (TWAP) (`src/advanced/twap_order.py`)

The TWAP function divides the total required quantity into equal `chunks` and executes them sequentially using Market orders, pausing for a user-defined `interval` (in seconds) between each chunk. This minimizes market impact for large trades.



E. Grid Orders (`src/advanced/grid_orders.py`)

This strategy places a series of alternating Limit BUY and SELL orders across a designated price range, from `lower_price` to `upper_price`, split by the number of `grids`. Orders below the midpoint are BUYs, and orders above are SELLs, capturing volatility within a channel.



F. Order Status and Management (`src/check_order_status.py`)

Provides two utility functions for trade audit and management:

1. Check Specific Order: Retrieves detailed status, executed quantity, and average price for a given Order ID.
2. List Recent Orders: Fetches and displays the most recent orders for a symbol.

4. Summary of Results

All scripts produce console confirmations and corresponding bot.log entries.
Post-execution log analysis confirms:

- Successful execution of all order types on Binance Futures Testnet.
- Proper handling of invalid parameters (e.g., insufficient notional, precision errors).
- Reliable error logging and prevention of crashes during invalid API calls.

bot.log

```
2025-10-29 12:45:53,054:INFO:Market Order Success | BTCUSDT | BUY | Qty: 0.01
2025-10-29 12:50:11,276:INFO:Fetched last 5 orders for BTCUSDT
2025-10-29 19:00:56,899:INFO:Limit Order Success | BTCUSDT | BUY | Qty: 0.01 | Price: 67000.0
2025-10-29 19:01:26,315:INFO:Fetched last 5 orders for BTCUSDT
2025-10-30 20:15:15,652:INFO:Stop-Limit Order Placed | BTCUSDT | SELL | Stop: 66000.0 | Limit: 65800.0 | Qty: 0.003
2025-10-30 20:18:05,548:INFO:OCO Simulated | BTCUSDT | SELL | TP: 106000.0 | SL: 103000.0
2025-10-30 20:18:47,479:INFO:TWAP 1/5 Success | BTCUSDT | SELL | Qty: 0.01
2025-10-30 20:18:57,750:INFO:TWAP 2/5 Success | BTCUSDT | SELL | Qty: 0.01
2025-10-30 20:19:08,015:INFO:TWAP 3/5 Success | BTCUSDT | SELL | Qty: 0.01
2025-10-30 20:19:18,598:INFO:TWAP 4/5 Success | BTCUSDT | SELL | Qty: 0.01
2025-10-30 20:19:28,792:INFO:TWAP 5/5 Success | BTCUSDT | SELL | Qty: 0.01
2025-10-30 20:20:15,682:INFO:Grid BUY Order @ 60000.0
2025-10-30 20:20:15,878:INFO:Grid BUY Order @ 62000.0
2025-10-30 20:20:16,327:INFO:Grid BUY Order @ 64000.0
2025-10-30 20:21:47,614:INFO:Grid BUY Order @ 101000.0
2025-10-30 20:21:48,158:INFO:Grid BUY Order @ 101600.0
2025-10-30 20:21:48,361:INFO:Grid BUY Order @ 102200.0
2025-10-30 20:21:48,670:INFO:Grid SELL Order @ 102800.0
2025-10-30 20:21:48,977:INFO:Grid SELL Order @ 103400.0
2025-10-30 20:21:49,487:INFO:Grid SELL Order @ 104000.0
```

5. Conclusion

This project demonstrates strong competency in algorithmic trading and API-driven system design. It implements secure credential management, robust logging, modular design, and advanced order execution with precision. The system provides a solid foundation for extending into live-market environments or integrating machine-learning-based strategy modules in the future.