

# Binance Futures Trading Bot

## Project Report

---

### 1. Introduction

This project implements a Python-based Binance Futures Trading Bot designed to interact with Binance's Futures Testnet API. The bot supports placing various order types including market, limit, TWAP (Time-Weighted Average Price), and grid orders. The goal is to provide a modular, extendable, and well-logged trading framework.

---

### 2. Project Structure

The project is organized into the following key components:

- `src/`: Core source code
    - `market_orders.py`: Market order functionality
    - `limit_orders.py`: Limit order functionality
    - `advanced/`: Advanced order types (TWAP, grid, OCO, stop-limit placeholders)
    - `utils/`: Utility modules (API client, validation, logging)
  - Configuration files including `.env` for API credentials, `.gitignore`, `requirements.txt`, and documentation.
- 

### 3. Implementation Details

#### 3.1 Binance Client Setup

- Used `python-binance` library to connect securely to Binance Futures Testnet.
- API credentials are loaded from environment variables using `dotenv`.
- Client instance created centrally in `utils/binance_client.py`.

#### 3.2 Order Functions

- **Market Orders:**  
Simple immediate execution orders implemented in `market_orders.py`. Includes input validation and error logging.
- **Limit Orders:**  
Orders executed at a specified price implemented in `limit_orders.py`, with validation for price inputs.

- **TWAP Orders:**

Implemented in advanced/twap.py, breaks a large order into smaller chunks executed over set intervals to minimize market impact.

- **Grid Orders:**

Implemented in advanced/grid\_orders.py, places multiple limit orders at incremental price steps to capitalize on market volatility.

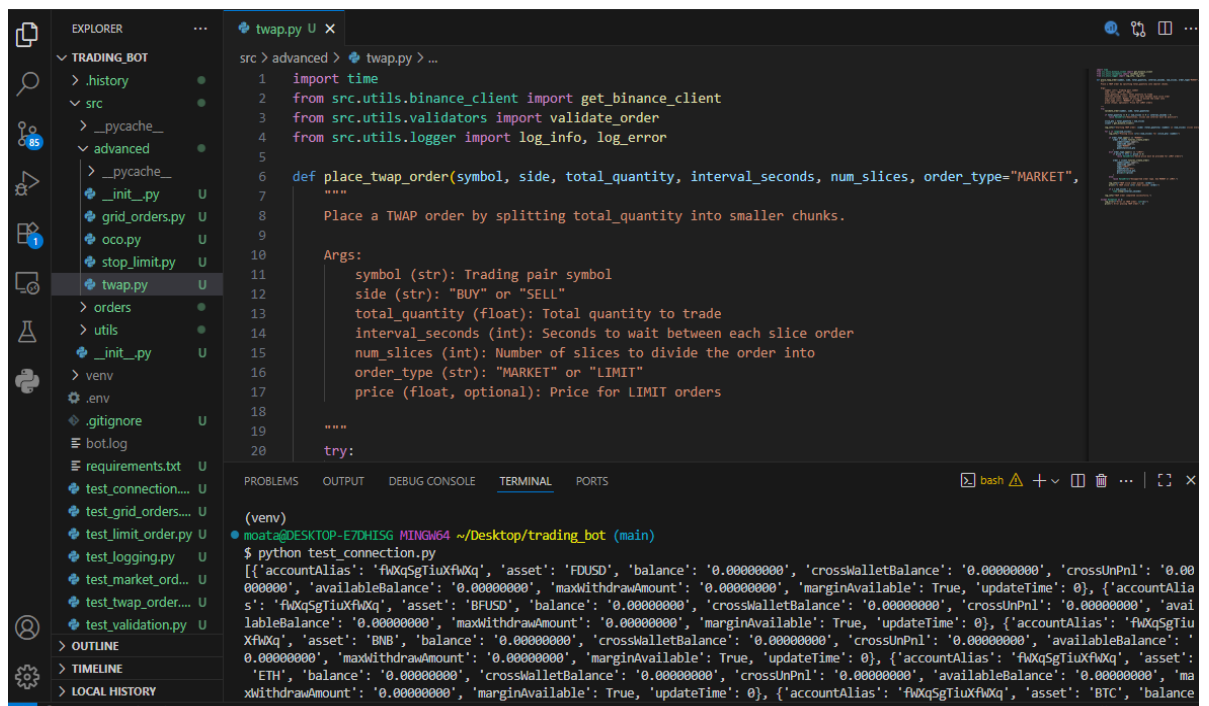
### 3.3 Utilities

- **Validation:** Checks symbol format, order side, quantity, and price validity.
- **Logging:** All actions and errors logged to bot.log for traceability.
- **Environment Management:** .env file used to manage sensitive keys securely.

## 4. Testing

- Created dedicated test scripts (test\_market\_order.py, test\_limit\_order.py, test\_twap\_order.py, test\_grid\_orders.py) for each order type.
- Successfully placed orders on Binance Futures Testnet with expected responses.
- Verified logging outputs for both success and error cases.

- **4.1 Test Connection**



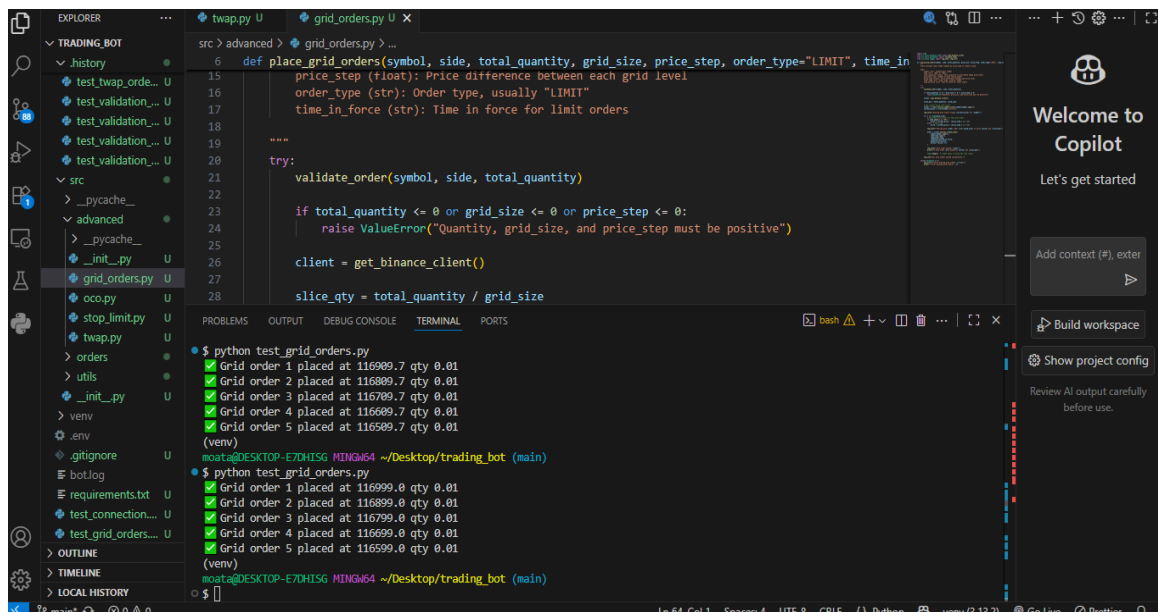
The screenshot shows a VS Code editor with the file explorer on the left displaying the project structure. The main editor window shows the `twap.py` file with the following code:

```
src > advanced > twap.py ...
1 import time
2 from src.utils.binance_client import get_binance_client
3 from src.utils.validators import validate_order
4 from src.utils.logger import log_info, log_error
5
6 def place_twap_order(symbol, side, total_quantity, interval_seconds, num_slices, order_type="MARKET",
7
8     """
9     Place a TWAP order by splitting total_quantity into smaller chunks.
10
11     Args:
12         symbol (str): Trading pair symbol
13         side (str): "BUY" or "SELL"
14         total_quantity (float): Total quantity to trade
15         interval_seconds (int): Seconds to wait between each slice order
16         num_slices (int): Number of slices to divide the order into
17         order_type (str): "MARKET" or "LIMIT"
18         price (float, optional): Price for LIMIT orders
19
20     """
21     try:
```

The terminal window at the bottom shows the output of running `python test_connection.py` in a virtual environment. The output displays account information for various assets, including balances and wallet balances.

```
(venv)
moata@DESKTOP-E7DHISG MINGW64 ~/Desktop/trading_bot (main)
$ python test_connection.py
[{'accountAlias': 'fWqSgTiuXfWq', 'asset': 'FDUSD', 'balance': '0.00000000', 'crossWalletBalance': '0.00000000', 'crossUnPnl': '0.00000000', 'availableBalance': '0.00000000', 'maxWithdrawAmount': '0.00000000', 'marginAvailable': True, 'updateTime': 0}, {'accountAlias': 'fWqSgTiuXfWq', 'asset': 'BFUSD', 'balance': '0.00000000', 'crossWalletBalance': '0.00000000', 'crossUnPnl': '0.00000000', 'availableBalance': '0.00000000', 'maxWithdrawAmount': '0.00000000', 'marginAvailable': True, 'updateTime': 0}, {'accountAlias': 'fWqSgTiuXfWq', 'asset': 'BNB', 'balance': '0.00000000', 'crossWalletBalance': '0.00000000', 'crossUnPnl': '0.00000000', 'availableBalance': '0.00000000', 'maxWithdrawAmount': '0.00000000', 'marginAvailable': True, 'updateTime': 0}, {'accountAlias': 'fWqSgTiuXfWq', 'asset': 'ETH', 'balance': '0.00000000', 'crossWalletBalance': '0.00000000', 'crossUnPnl': '0.00000000', 'availableBalance': '0.00000000', 'maxWithdrawAmount': '0.00000000', 'marginAvailable': True, 'updateTime': 0}, {'accountAlias': 'fWqSgTiuXfWq', 'asset': 'BTC', 'balance': '0.00000000', 'crossWalletBalance': '0.00000000', 'crossUnPnl': '0.00000000', 'availableBalance': '0.00000000', 'maxWithdrawAmount': '0.00000000', 'marginAvailable': True, 'updateTime': 0}]
```

## 4.2 Grid Order

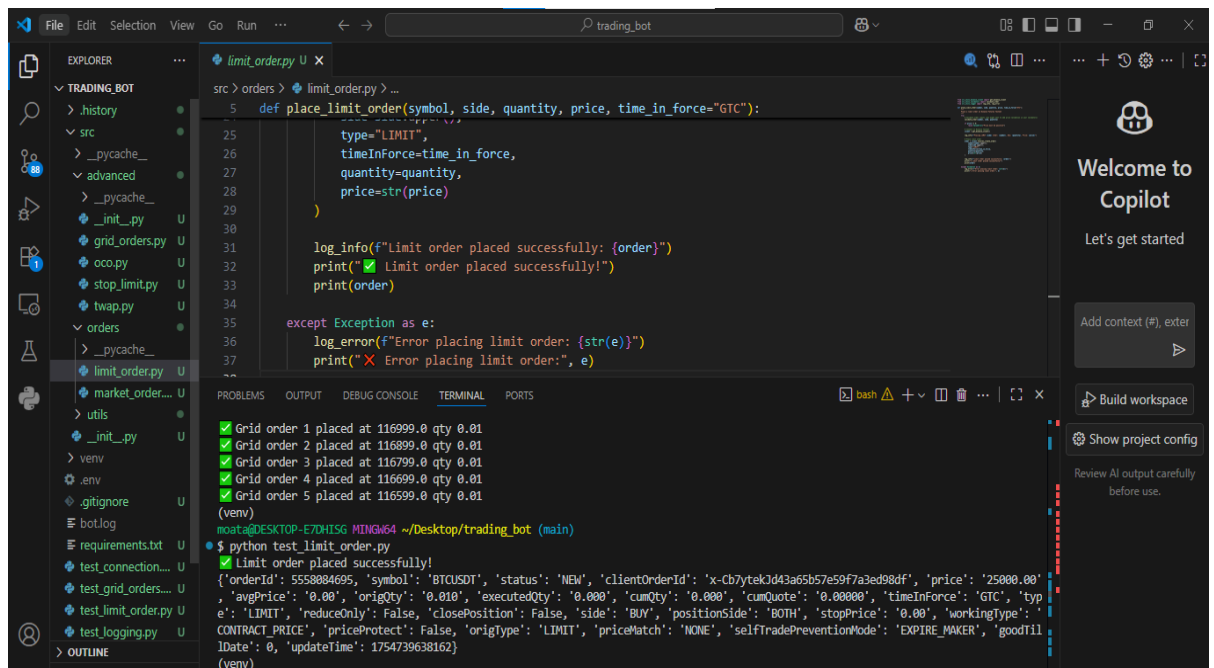


The screenshot shows a VS Code editor with a project named 'TRADING\_BOT'. The file explorer on the left shows a directory structure with files like 'test\_twap\_order.py', 'test\_validation.py', 'src', 'advanced', 'pycache', 'init.py', 'grid\_orders.py', 'oco.py', 'stop\_limit.py', 'twap.py', 'orders', 'utils', 'env', 'gitignore', 'botlog', 'requirements.txt', 'test\_connection.py', 'test\_grid\_orders.py', 'OUTLINE', 'TIMELINE', and 'LOCAL HISTORY'. The main editor window shows the 'grid\_orders.py' file with the following code:

```
src > advanced > grid_orders.py > ...
6 def place_grid_orders(symbol, side, total_quantity, grid_size, price_step, order_type="LIMIT", time_in_force="GTC"):
15     price_step (float): Price difference between each grid level
16     order_type (str): Order type, usually "LIMIT"
17     time_in_force (str): Time in force for limit orders
18
19     """
20     try:
21         validate_order(symbol, side, total_quantity)
22
23         if total_quantity <= 0 or grid_size <= 0 or price_step <= 0:
24             raise ValueError("Quantity, grid_size, and price_step must be positive")
25
26         client = get_binance_client()
27
28         slice_qty = total_quantity / grid_size
```

The terminal window at the bottom shows the execution of the 'test\_grid\_orders.py' script, which successfully places 5 grid orders at 116989.7, 116989.7, 116989.7, 116989.7, and 116989.7 with a quantity of 0.01 each.

## 4.3 Limit Order



The screenshot shows a VS Code editor with a project named 'TRADING\_BOT'. The file explorer on the left shows a directory structure with files like 'test\_twap\_order.py', 'test\_validation.py', 'src', 'advanced', 'pycache', 'init.py', 'grid\_orders.py', 'oco.py', 'stop\_limit.py', 'twap.py', 'orders', 'utils', 'env', 'gitignore', 'botlog', 'requirements.txt', 'test\_connection.py', 'test\_grid\_orders.py', 'test\_limit\_order.py', 'test\_logging.py', 'OUTLINE', 'TIMELINE', and 'LOCAL HISTORY'. The main editor window shows the 'limit\_order.py' file with the following code:

```
src > orders > limit_order.py > ...
5 def place_limit_order(symbol, side, quantity, price, time_in_force="GTC"):
25     """
26     type="LIMIT",
27     timeInForce=time_in_force,
28     quantity=quantity,
29     price=price
30     """
31
32     log_info(f"Limit order placed successfully: {order}")
33     print("✅ Limit order placed successfully!")
34     print(order)
35
36 except Exception as e:
37     log_error(f"Error placing limit order: {str(e)}")
38     print("❌ Error placing limit order:", e)
```

The terminal window at the bottom shows the execution of the 'test\_limit\_order.py' script, which successfully places a limit order at 116999.0 with a quantity of 0.01. The output shows the order details, including the order ID, symbol, status, client order ID, price, average price, original quantity, executed quantity, cumulative quantity, cumulative quote, time in force, order type, reduce only flag, close position flag, side, position side, stop price, working type, contract price, price protect flag, original type, price match flag, self-trade prevention mode, good till date, and update time.

## 4.4 Market Order

The screenshot shows a VS Code editor with a project named 'trading\_bot'. The Explorer panel on the left shows the file structure, including a 'src' directory with 'orders' and 'advanced' subdirectories. The 'market\_order.py' file is open in the editor. The code defines a function 'place\_market\_order' that takes 'symbol', 'side', and 'quantity' as arguments. It includes a docstring, a try-except block for error handling, and a log statement. The function uses a 'client' to create a market order. The terminal at the bottom shows the execution of a test script, which successfully places a market order for 'BTCUSD'.

```
def place_market_order(symbol, side, quantity):  
    """  
    Place a market order on Binance Futures Testnet  
    """  
    try:  
        validate_order(symbol, side, quantity)  
  
        client = get_binance_client()  
  
        log_info(f"Placing MARKET {side} order: {symbol}, Qty: {quantity}")  
  
        order = client.futures_create_order(  
            symbol=side.upper(),  
            side=side.upper(),  
            type="MARKET",  
            quantity=quantity  
        )
```

```
CONTRACT_PRICE', 'priceProtect': False, 'origType': 'LIMIT', 'priceMatch': 'NONE', 'selfTradePreventionMode': 'EXPIRE_MAKER', 'goodTillDate': 0, 'updateTime': 1754739638162}  
(venv)  
moata@DESKTOP-E7DHISG MINGW64 ~/Desktop/trading_bot (main)  
$ python test_market_order.py  
✓ Order placed successfully!  
{'orderId': 5558090140, 'symbol': 'BTCUSD', 'status': 'NEW', 'clientOrderId': 'x-Cb7ytekJd4fd4556e87c3a45b4486e', 'price': '0.00', 'avgPrice': '0.00', 'origQty': '0.018', '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': 1754739739704}  
(venv)
```

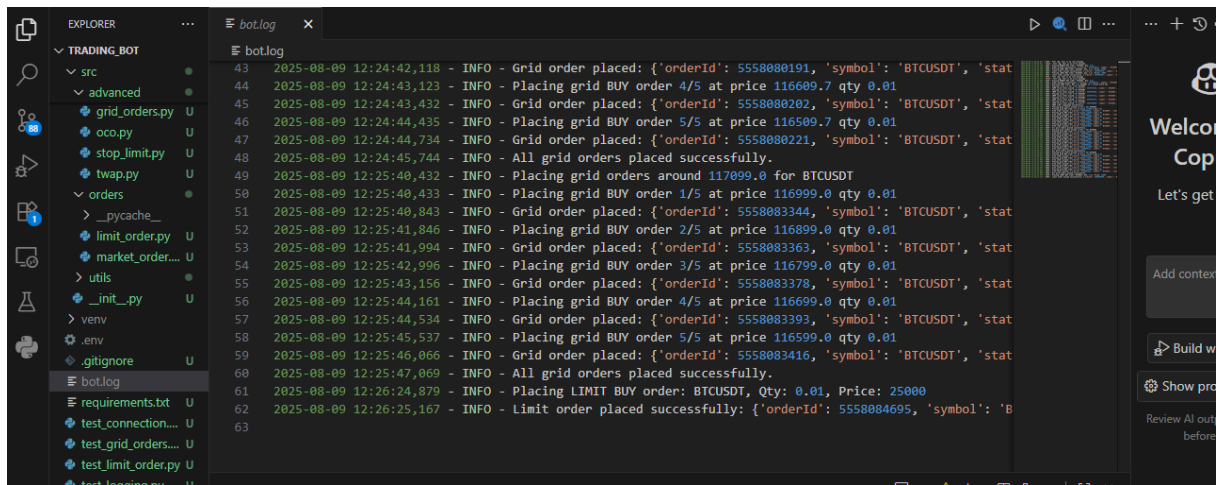
## 4.5 Twap

The screenshot shows a VS Code editor with the same 'trading\_bot' project. The 'twap.py' file is open in the editor. The code defines a function 'place\_twap\_order' that takes 'symbol', 'side', 'total\_quantity', 'interval\_seconds', 'num\_slices', and 'order\_type' as arguments. It includes a docstring, a try-except block for error handling, and a log statement. The function uses a 'client' to create a TWAP order. The terminal at the bottom shows the execution of a test script, which successfully places a TWAP order for 'BTCUSD'.

```
def place_twap_order(symbol, side, total_quantity, interval_seconds, num_slices, order_type="MARKET",  
    validate_order(symbol, side, total_quantity)  
  
    if total_quantity <= 0 or num_slices <= 0 or interval_seconds < 0:  
        raise ValueError("Quantities, slices and interval must be positive")  
  
    slice_qty = total_quantity / num_slices  
    client = get_binance_client()  
  
    log_info(f"Starting TWAP order: {side} {total_quantity} {symbol} in {num_slices} slices every {interval_seconds} seconds")  
  
    for i in range(num_slices):  
        log_info(f"Placing slice {i+1}/{num_slices} for {slice_qty} {symbol}")  
  
        if order_type.upper() == "MARKET":  
            order = client.futures_create_order(  
                symbol=side.upper(),  
                side=side.upper(),  
                type="MARKET",  
                quantity=slice_qty  
            )
```

```
'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': 1754739739704}  
(venv)  
moata@DESKTOP-E7DHISG MINGW64 ~/Desktop/trading_bot (main)  
$ python test_twap_order.py  
✓ TWAP slice 1 order placed: {'orderId': 5558093318, 'symbol': 'BTCUSD', 'status': 'NEW', 'clientOrderId': 'x-Cb7ytekJ2e14506af9a64d6f0df050', 'price': '0.00', 'avgPrice': '0.00', 'origQty': '0.002', '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': 1754739804946}  
(venv)
```

## 4.6 bot.log



```
43 2025-08-09 12:24:42,118 - INFO - Grid order placed: {'orderId': 5558080191, 'symbol': 'BTCUSDT', 'stat
44 2025-08-09 12:24:43,123 - INFO - Placing grid BUY order 4/5 at price 116609.7 qty 0.01
45 2025-08-09 12:24:43,432 - INFO - Grid order placed: {'orderId': 5558080202, 'symbol': 'BTCUSDT', 'stat
46 2025-08-09 12:24:44,435 - INFO - Placing grid BUY order 5/5 at price 116509.7 qty 0.01
47 2025-08-09 12:24:44,734 - INFO - Grid order placed: {'orderId': 5558080221, 'symbol': 'BTCUSDT', 'stat
48 2025-08-09 12:24:45,744 - INFO - All grid orders placed successfully.
49 2025-08-09 12:25:40,432 - INFO - Placing grid orders around 117099.0 for BTCUSDT
50 2025-08-09 12:25:40,433 - INFO - Placing grid BUY order 1/5 at price 116999.0 qty 0.01
51 2025-08-09 12:25:40,843 - INFO - Grid order placed: {'orderId': 5558083344, 'symbol': 'BTCUSDT', 'stat
52 2025-08-09 12:25:41,846 - INFO - Placing grid BUY order 2/5 at price 116899.0 qty 0.01
53 2025-08-09 12:25:41,994 - INFO - Grid order placed: {'orderId': 5558083363, 'symbol': 'BTCUSDT', 'stat
54 2025-08-09 12:25:42,996 - INFO - Placing grid BUY order 3/5 at price 116799.0 qty 0.01
55 2025-08-09 12:25:43,156 - INFO - Grid order placed: {'orderId': 5558083378, 'symbol': 'BTCUSDT', 'stat
56 2025-08-09 12:25:44,161 - INFO - Placing grid BUY order 4/5 at price 116699.0 qty 0.01
57 2025-08-09 12:25:44,534 - INFO - Grid order placed: {'orderId': 5558083393, 'symbol': 'BTCUSDT', 'stat
58 2025-08-09 12:25:45,537 - INFO - Placing grid BUY order 5/5 at price 116599.0 qty 0.01
59 2025-08-09 12:25:46,066 - INFO - Grid order placed: {'orderId': 5558083416, 'symbol': 'BTCUSDT', 'stat
60 2025-08-09 12:25:47,069 - INFO - All grid orders placed successfully.
61 2025-08-09 12:26:24,879 - INFO - Placing LIMIT BUY order: BTCUSDT, Qty: 0.01, Price: 25000
62 2025-08-09 12:26:25,167 - INFO - Limit order placed successfully: {'orderId': 5558084695, 'symbol': 'B
63
```

---

## 5. Challenges and Solutions

- **API Rate Limits:** Introduced small delays between multiple orders (e.g., in grid and TWAP) to prevent hitting rate limits.
- **Input Validation:** Ensured robust validation to prevent invalid order requests.
- **Error Handling:** Added comprehensive try-except blocks and logging to gracefully handle exceptions.

---

## 6. Future Work

- Implement additional advanced order types such as OCO and stop-limit.
- Add order status tracking and cancellation features.
- Enhance UI/UX for easier interaction (e.g., CLI or web dashboard).
- Integrate real-time market data streaming and analysis.

---

## 7. Conclusion

This project demonstrates a modular and extendable approach to building a Binance Futures trading bot with essential order types. The implementation balances functionality, validation, and error handling, providing a solid foundation for further enhancements.

---

## Appendix

- **Setup Instructions:** See README.md
- **Dependencies:** Listed in requirements.txt
- **API Documentation:** Binance Futures API