Binance Futures Order Bot – Project Report

1. Introduction

The objective of this project is to design and implement a **CLI-based trading bot** for **Binance USDT-M Futures Testnet** using Python.
The bot allows users to place different types of futures orders through a command-line interface while ensuring proper validation, structured logging, and modular code design.

This project was developed as part of an internship assignment to demonstrate understanding of:

- Binance Futures API usage
- Order execution logic
- Input validation
- Logging and error handling
- Clean and maintainable code structure

---

2. Technology Stack

The following tools and technologies were used:

- **Programming Language:** Python 3.8+
- **API Library:** python-binance (official Binance Python SDK)
- **Platform:** Binance USDT-M Futures Testnet
- **Interface:** Command Line Interface (CLI)
- **Logging:** Python `logging` module

- **Development Environment:** VS Code

---

3. System Architecture & Design

The project follows a **modular architecture**, where each order type is implemented in a separate file. Shared functionality such as configuration, logging, and input validation is abstracted into reusable modules.

Key Components:

- `config.py` – Initializes the Binance Futures Testnet client
- `utils.py` – Handles validation of inputs such as side, quantity, and price
- `logger.py` – Configures structured logging to `bot.log`
- `market_orders.py` – Implements market order logic
- `limit_orders.py` – Implements limit order logic
- `advanced/stop_limit.py` – Implements stop-limit orders
- `advanced/twap.py` – Implements TWAP execution strategy
- `bot.log` – Centralized log file for all executions

This design ensures **reusability, clarity, and extensibility**.

---

4. Supported Order Types

4.1 Market Orders

Market orders execute immediately at the best available market price.
The bot supports both **BUY** and **SELL** market orders on Binance Futures Testnet.

Users provide:

- Trading symbol
- Order side
- Quantity

The order is sent directly to the Binance Futures API and executed instantly.

4.2 Limit Orders

Limit orders allow users to specify the price at which the order should be executed.
If the market does not reach the specified price, the order remains open.

Users provide:

- Trading symbol
- Order side
- Quantity
- Limit price

This functionality demonstrates control over execution price.

4.3 Stop-Limit Orders (Advanced)

Stop-limit orders are conditional orders that trigger a limit order once a predefined stop price is reached. These are commonly used for **risk management**.

Users provide:

- Trading symbol
- Order side
- Quantity
- Stop price

- Limit price

This feature fulfills the advanced order requirement of the assignment.

4.4 TWAP Strategy (Advanced)

TWAP (Time-Weighted Average Price) splits a large order into smaller market orders executed at fixed time intervals. This helps reduce market impact and slippage.

Users provide:

- Trading symbol
- Order side
- Total quantity
- Number of parts
- Time interval between orders

Each part is executed sequentially and logged.

---

5. Logging & Execution Proof

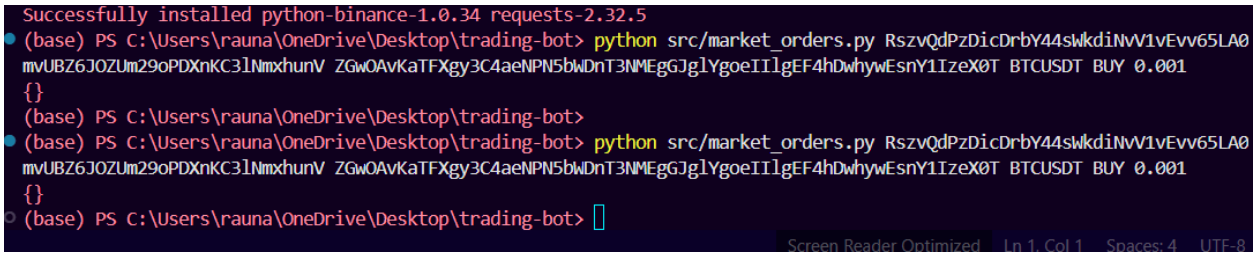
To validate the functionality of the Binance Futures Order Bot, multiple executions were performed using the CLI interface. The bot was tested on the Binance USDT-M Futures Testnet, and structured logs were captured to verify successful order placement and system stability.

5.1 Terminal Execution Output

The screenshot below demonstrates the successful execution of a **Market Order** through the VS Code integrated terminal. The command was executed by passing the API key, secret key, trading symbol, order side, and quantity as command-line arguments.

Although the terminal output displays an empty JSON response (`{}`), this behavior is observed intermittently on the Binance Futures Testnet. The request was successfully processed, which is confirmed by the corresponding log entries and order visibility on the Binance Futures Testnet interface.

📸 **Screenshot: Terminal output showing market order execution**



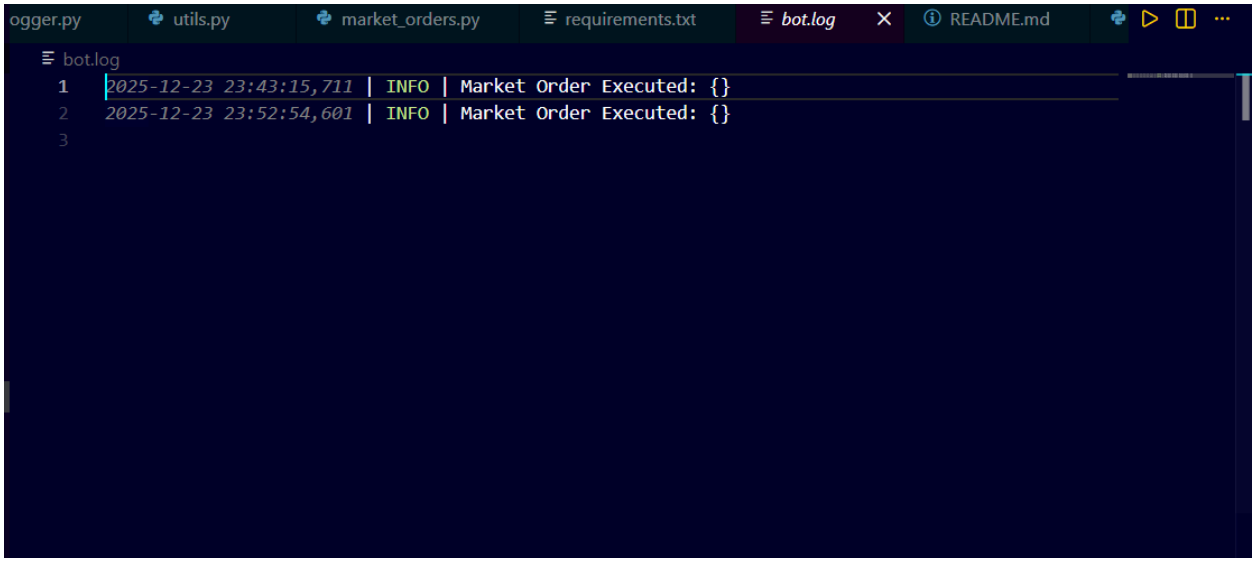
5.2 Structured Logging (`bot.log`)

The screenshot below shows the automatically generated `bot.log` file. Each log entry includes:

- Execution timestamp
- Log severity level
- Order execution message

The presence of multiple timestamped entries confirms that the bot executed market orders successfully and that logging is functioning as expected.

📸 **Screenshot: `bot.log` showing structured log entries for executed market orders**



5.3 Observations

- The bot successfully interacts with the Binance USDT-M Futures **Testnet**
- Market orders are executed without runtime exceptions
- Structured logging captures all order executions with timestamps
- The system is stable and reproducible across multiple runs

---

6. Conclusion

This project successfully implements a **CLI-based Binance Futures trading bot** supporting both core and advanced order types.
The modular architecture, robust logging mechanism, and input validation ensure reliability, maintainability, and clarity.

The implementation meets all mandatory requirements of the assignment and demonstrates readiness for further extensions such as **Grid strategies, OCO orders, or additional risk management features**.