📘 Binance Futures Order Bot – Project Report

1. Introduction

This project implements a **CLI-based trading bot** for **Binance USDT-M Futures**.
The objective is to design a **robust, modular, and safe trading system** that supports multiple order types, input validation, structured logging, and real-world API limitations.

The focus of this project is **engineering quality and correctness**, not trading profitability.

---

2. Objectives

- Develop a CLI-based trading bot for Binance Futures
- Support both **basic and advanced order types**
- Ensure **input validation** before order placement
- Implement **structured logging**
- Handle **Binance Futures testnet limitations** gracefully
- Maintain a clean project structure and documentation

---

3. System Architecture

The project follows a **modular architecture**, where each responsibility is separated into dedicated modules.

Key Components

- **config.py** – Handles Binance API configuration and client creation
- **logger.py** – Centralised structured logging
- **validator.py** – Input validation layer

- **Order modules** – Separate files for each order type
- **Advanced strategies** – TWAP and OCO implementations

This design improves:

- Readability
- Maintainability
- Extensibility

---

4. Supported Order Types

4.1 Market Order

Executes immediately at the current market price.

Usage example:

```
python3 -m src.market_orders BTCUSDT BUY 0.01
```

4.2 Limit Order

Places an order at a specified price.

Usage example:

```
python3 -m src.limit_orders BTCUSDT SELL 0.01 45000
```

4.3 Stop-Limit Order

Triggers a limit order once the stop price is reached.

Usage example:

```
python3 -m src.advanced.stop_limit BTCUSDT SELL 0.01 44800 44700
```

Validation ensures:

- Correct BUY/SELL logic
- Proper relationship between stop and limit prices

4.4 TWAP (Time-Weighted Average Price)

TWAP splits a large order into smaller market orders executed over time to reduce market impact.

Usage example:

```
python3 -m src.advanced.twap BTCUSDT BUY 0.02 4 15
```

This executes:

- 4 market orders
- Every 15 seconds apart

4.5 OCO (One-Cancels-the-Other) – Simulated

Since Binance Futures does not support native OCO, it is **simulated** using:

- One Take-Profit order
- One Stop-Loss order
- Continuous monitoring

When one order is filled, the other is automatically cancelled.

Usage example:

```
python3 -m src.advanced.oco BTCUSDT SELL 0.01 45500 44500
```

---

5. Validation Layer

All inputs are validated before sending requests to Binance:

- Symbol format
- Order side (BUY / SELL)
- Quantity > 0
- Price > 0
- Logical price relationships (Stop-Limit & OCO)

This prevents:

- Invalid API calls
- Unexpected crashes
- Incorrect trading behaviour

---

6. Logging System

A centralised logging system records all activities into **`bot.log`**, including:

- Order requests
- Validation failures
- API responses
- Execution status
- Testnet limitations

Example Log Entry

```
2025-12-30 | INFO | binance_bot | Placing STOP-LIMIT order | Symbol: BTCUSDT | Side: SELL
```

Structured logging helps with:

- Debugging

- Auditability
- Evaluation transparency

---

7. Handling Binance Futures Testnet Limitations

During development, it was observed that Binance Futures **testnet may return incomplete responses**, especially for:

- LIMIT orders
- STOP orders
- Conditional orders

In some cases, responses do not include an `orderId`.

Design Decision

Instead of assuming success, the system:

- Detects missing fields
- Logs the limitation clearly
- Fails gracefully without crashing

Example log:

```
Stop-Limit order response missing orderId (testnet limitation)
```

This approach reflects **real-world defensive programming practices**.

---

8. Testing & Execution

All features were tested using:

- Binance Futures **TESTNET**

- CLI-based execution
- Log verification via `bot.log`

Screenshots included:

- Terminal command execution
- Corresponding log entries

---

9. Project Structure

```
src/
├── config.py
├── logger.py
├── validator.py
├── market_orders.py
├── limit_orders.py
└── advanced/
    ├── stop_limit.py
    ├── twap.py
    └── oco.py
```

---

10. Conclusion

This project successfully demonstrates:

- Correct usage of Binance Futures API
- Implementation of advanced trading strategies
- Clean and modular software design
- Robust validation and logging
- Awareness of real-world API limitations

The system is **safe, extensible, and evaluation-ready**.

---

11. Future Improvements

- WebSocket-based order monitoring
- Risk management rules
- Strategy backtesting
- Native OCO for spot trading
- UI dashboard for monitoring

---

12. Author

Kumar Gautam
Binance Futures Order Bot
Academic / Learning Project

---

13. Disclaimer

This project is for **educational purposes only**.
It does not constitute financial advice and should not be used for live trading without proper risk management.