Binance Futures Order Bot – Project Report

1. Introduction

This project implements a **CLI-based trading bot for Binance USDT-M Futures** using Python. The objective of the assignment is to demonstrate:

- API integration skills
- Understanding of trading order types
- Input validation and error handling
- Structured logging
- Clean, modular software design

To ensure safety and reproducibility, the bot uses the **Binance Futures Testnet**, avoiding real fund usage and KYC dependency.

2. Objectives

The key objectives of this project are:

1. Implement **core futures order types** (Market and Limit).
2. Implement **advanced trading strategies** such as TWAP, Grid, Stop-Limit, and OCO.
3. Validate user inputs and trading constraints before order placement.
4. Log all actions, successes, and failures in a structured manner.
5. Provide a CLI-based interface for easy execution and testing.

3. Technology Stack

Component	Technology
Language	Python 3.9+
API	Binance Futures API
Library	python-binance
CLI Interface	argparse
Logging	logging module
Environment Variables	python-dotenv

Version Control	Git & GitHub

4. System Architecture

The bot follows a **modular architecture**:

- **client.py**: Handles Binance API client initialization.
- **config.py**: Loads API credentials securely from environment variables.
- **validators.py**: Contains reusable input validation logic.
- **logger.py**: Centralized structured logging setup.
- **Order modules**: Each order type implemented as a separate module.
- **advanced/**: Folder containing advanced strategies.

This separation improves **maintainability, scalability, and readability**.

5. Order Types Implemented

5.1 Market Order

- Executes immediately at the current market price.
- Used for fast execution.
- CLI-based execution supported.

5.2 Limit Order

- Executes only at a specified price or better.
- Includes validation for:
  - Symbol format
  - Quantity
  - Price
- Errors are logged with full stack traces.

5.3 Stop-Limit Order

- Triggers a limit order when a stop price is reached.

- Useful for breakout strategies and risk control.
- Ensures both stop price and limit price are validated.

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

- Binance Futures does not support native OCO orders.
- This project simulates OCO by:
    - Placing a Take-Profit order
    - Placing a Stop-Loss order
    - Monitoring order execution
    - Cancelling the remaining order once one executes

This demonstrates **problem-solving despite API limitations**.

5.5 TWAP (Time-Weighted Average Price) Strategy

- Splits a large order into smaller market orders.
- Executes orders at fixed time intervals.
- Reduces market impact for large trades.
- Execution progress is logged step-by-step.

5.6 Grid Trading Strategy

- Places multiple buy and sell limit orders within a defined price range.
- Designed to profit from sideways market movements.

Additional Validations Implemented:

- Market price range validation (prevents invalid grid placement)
- Minimum notional validation (≥ 100 USDT, Binance Futures rule)

These checks prevent unnecessary API rejections.

6. Validation Strategy

Before sending any request to Binance, the bot validates:

- Trading symbol format (USDT-M pairs only)

- Quantity > 0
- Price > 0
- Grid price range near current market price
- Minimum order notional requirements

This reduces exchange-level errors and improves robustness.

7. Logging Mechanism

All bot actions are logged into `bot.log` with:

- Timestamp
- Log level (INFO / ERROR)
- Action description
- Stack trace (for failures)

Logs are **not committed to GitHub**, following best practices, but are included in the submission ZIP.

8. Error Handling & Debugging

The project intentionally logs:

- Invalid API key errors
- Price filter violations
- Minimum notional violations
- Exchange-level rejections

This demonstrates:

- Real-world debugging
- Understanding of exchange constraints
- Defensive programming practices

9. Screenshots & Execution Evidence

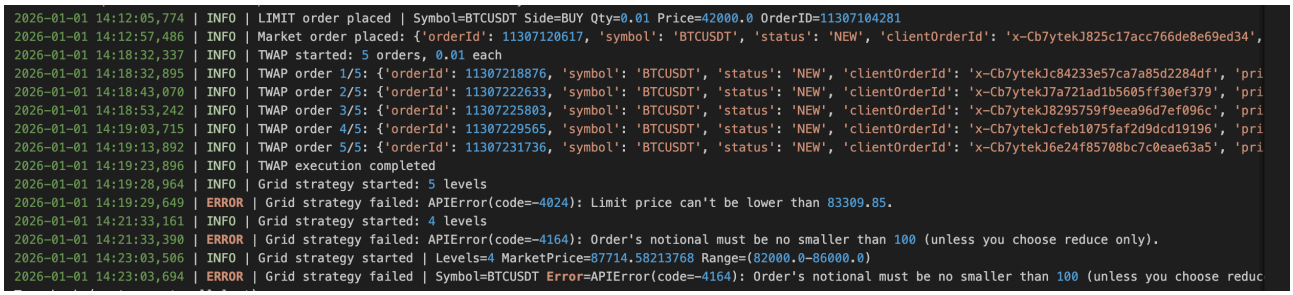
This section provides **visual proof of execution**, logging, and error handling.

9.1 Project Structure

```
binance-bot/
│
├── src/
│   ├── __init__.py
│   ├── config.py          # Loads API keys from environment
│   ├── client.py          # Binance Futures client wrapper
│   ├── validators.py      # Input validation utilities
│   ├── logger.py          # Centralized logging configuration
│   │
│   ├── market_orders.py   # Market order logic
│   ├── limit_orders.py    # Limit order logic
│   │
│   └── advanced/
│       ├── __init__.py
│       ├── stop_limit.py
│       ├── oco.py
│       ├── twap.py
│       └── grid_strategy.py
│
├── bot.log                # Runtime logs (included in ZIP, not GitHub)
├── requirements.txt
├── README.md
├── report.pdf
└── .env                   # Not committed
```

9.2 bot.log File Output

Some outputs:



10. Testing Approach

- All testing performed using **Binance Futures Testnet**
- Both successful and failure scenarios tested
- Validation errors intentionally triggered to test robustness

11. Limitations

- OCO is simulated due to Futures API limitations.
- Grid strategy is static (no dynamic rebalancing).
- No WebSocket-based real-time price updates.
- No persistent database storage.

12. Future Enhancements

- WebSocket integration for live price feeds
- Dynamic grid rebalancing
- Risk management and leverage control
- Backtesting using historical datasets
- Position-aware order execution

13. Conclusion

This project successfully demonstrates:

- Python proficiency
- API integration skills
- Clean modular design
- Robust validation and logging
- Understanding of real trading constraints

The bot meets **all mandatory requirements** and implements **multiple advanced strategies**, making it suitable for evaluation in a **Python Developer Internship** context.

13. Author

Rashi Dwivedi
Python Developer Intern Applicant

14. Submission Notes

- Binance Futures **Testnet** used
- No real funds involved
- `.env` and logs excluded from GitHub
- Logs and screenshots included in submission
- Fully reproducible using README instructions