Binance Futures Order Bot – Project Report

1. Introduction

This project implements a CLI-based trading bot for Binance USDT-M Futures using the official Binance Futures API. The bot allows users to place futures orders directly from the command line without any graphical interface. The implementation focuses on:

- Correct API usage
- Safe testing using Binance Futures Testnet
- Input validation
- Structured logging
- Clean and reproducible project structure

No real money or real trades are involved.

2. Objectives

The main objectives of this project are:

- Build a command-line (CLI) trading bot
- Support mandatory order types (Market & Limit)
- Implement at least one advanced order strategy
- Log all actions and errors
- Ensure safe and reproducible execution

3. Technology Stack

- **Programming Language:** Python 3
- **API:** Binance USDT-M Futures API
- **Library:** python-binance
- **Environment:** Binance Futures Testnet
- **Logging:** Python logging module
- **Security:** Environment variables using .env

4. Project Architecture

The project follows a modular structure:binance_bot/

| ├── src/

| ├── market_orders.py

| ├── limit_orders.py

| ├── logger.py

| ├── validators.py

| | | └── advanced/

| └── twap.py

| ├── bot.log

├── README.md

└── report.pdf

Key Design Choices

- CLI-based execution (no UI)
- Separate modules for validation and logging
- Advanced strategies isolated in `advanced/` folder
- Centralized logging to `bot.log`

5. Implemented Order Types

5.1 Market Orders (Mandatory)

Market orders execute immediately at the current market price.

Command: `python src/market_orders.py BTCUSDT BUY 0.01`

Behavior:

- Validates inputs
- Places a market BUY or SELL order
- Logs the execution result

5.2 Limit Orders (Mandatory)

Limit orders execute only when the specified price is reached.

Command: `python src/limit_orders.py BTCUSDT SELL 0.01 90000`

Behavior:

- Places a GTC (Good-Till-Cancelled) limit order
- Order remains open until executed or canceled
- Logs the order placement

5.3 TWAP Strategy (Advanced Order)

TWAP (Time-Weighted Average Price) is implemented as a strategy, not a native Binance feature.

Command: `python src/advanced/twap.py BTCUSDT BUY 0.05 5 10`

Explanation:

- Total quantity: 0.05 BTC
- Split into: 5 smaller orders
- Delay: 10 seconds between each order

Purpose:

- Reduce market impact
- Demonstrate advanced order logic

6. Validation & Error Handling

Input validation is performed before placing any order:

- Symbol format (must end with USDT)
- Order side (BUY or SELL)
- Quantity and price must be greater than zero

Invalid inputs prevent API calls and are logged as errors.

7. Logging

All actions are logged to a structured log file: `bot.log`

Logged events include:

- Order placement
- Strategy execution (TWAP slices)
- Validation errors
- API errors

Each log entry includes:

- Timestamp
- Log level
- Action details

8. Security & Test Environment

- API credentials are stored securely using environment variables
- `.env` file is excluded from version control
- All operations are executed on Binance Futures Testnet
- No real funds are used

9. Results & Screenshots

The following screenshots are included:

- Successful market order execution
- Successful limit order placement
- TWAP strategy execution
- `bot.log` showing order logs

10. Limitations

- OCO orders are not natively supported in Binance Futures and were not implemented
- Grid strategy was not implemented due to time constraints
- Testnet responses may return minimal order details

11. Conclusion

This project successfully demonstrates:

- Practical use of Binance Futures APIs
- CLI-based trading system design
- Secure credential handling
- Logging and validation best practices
- Implementation of an advanced trading strategy (TWAP)

The bot is safe, reproducible, and aligned with real-world backend trading system design principles.