REPORT

Introduction

This assignment focuses on designing and executing **automated cryptocurrency trading** strategies using various order types, including market, limit, stop-limit, OCO, TWAP, and grid orders. By interacting with a **live trading API**, we analyze how each order behaves under real market conditions while adhering to exchange constraints such as minimum notional value. The work emphasizes understanding **risk management components** like stop-loss and take-profit triggers. Through hands-on experimentation, we develop a practical and technical understanding of modern trading mechanisms and their applications in algorithmic systems.

Project Structure

This project is structured as a **modular Python-based trading bot** that interacts with the Binance exchange using **authenticated REST API calls**. The directory layout cleanly separates core logic, advanced trading strategies, logging, reporting, and configuration. All API credentials are securely stored in a **.env** file to ensure keys are never exposed in code. The structure ensures clarity, scalability, and ease of navigation while supporting both basic and algorithmic order execution modules.

```
/Kushagra_Gupta_Binance_Bot/
│
├── .env                    # Stores API key + secret (never commit!)
├── bot.log                 # Log file storing order history and errors
├── Report.pdf              # Analysis (screenshots, explanations)
├── README.md               # How to run + install + examples (you fill in)
│
├── /src/                   # All Python source code
│   │
│   ├── chatbot.py          # Interactive CLI bot that accepts user commands
│   ├── client.py           # Creates and returns authenticated Binance client
│   ├── market_orders.py    # Market buy/sell order logic
│   ├── limit_orders.py     # Limit + Stop-Limit order logic
│   │
│   ├── /advanced/          # Bonus order strategies folder
│   │   ├── oco.py          # One-Cancels-the-Other order execution
│   │   ├── twa.py          # TWAP: splits large trades over time
│   │   └── ga.py           # Grid strategy: buys low, sells high across levels
│   │
│   └── __pycache__/        # Python compiled cache files (auto-generated)
│
└── venv/ (optional)        # Virtual environment if you created one
```

The **chatbot** serves as the **main user interface of the trading bot**, guiding the user interactively through every trade. It prompts for different inputs depending on the selected market type: simple quantity for market orders, price and size for limit orders, stop prices for stop-limit trades, and multiple parameters for **advanced types such as OCO, TWAP, and Grid.** Each input request dynamically adapts to the order style, ensuring safe and structured execution without requiring the user to remember syntax. All **advanced order implementations** are also supported, and stop-loss functionality has been integrated inside **limit_orders.py** as part of stop-limit logic, maintaining clear organization and reusability across order flows.

```
2026-01-10 13:47:23,574 - ERROR - LIMIT BUY FAILED: APIError(code=-4164): Order's notional must be no smaller than
2026-01-10 13:47:24,055 - ERROR - LIMIT SELL FAILED: APIError(code=-4024): Limit price can't be lower than 86083.46
2026-01-10 13:48:07,710 - ERROR - LIMIT BUY FAILED: APIError(code=-4014): Price not increased by tick size.
2026-01-10 13:48:07,883 - ERROR - LIMIT SELL FAILED: APIError(code=-4014): Price not increased by tick size.
2026-01-10 13:48:51,108 - ERROR - LIMIT BUY FAILED: APIError(code=-4014): Price not increased by tick size.
2026-01-10 13:48:51,673 - ERROR - LIMIT SELL FAILED: APIError(code=-4014): Price not increased by tick size.
2026-01-10 13:49:48,596 - ERROR - LIMIT BUY FAILED: APIError(code=-1111): Precision is over the maximum defined for
2026-01-10 13:49:49,171 - INFO - LIMIT SELL BTCUSDT qty=0.002@91495.0 SUCCESS {'orderId': 11560598871, 'symbol': 'B
2026-01-10 13:54:42,917 - ERROR - LIMIT BUY FAILED: APIError(code=-4164): Order's notional must be no smaller than
2026-01-10 13:54:45,453 - ERROR - LIMIT SELL FAILED: APIError(code=-4164): Order's notional must be no smaller than
2026-01-10 14:10:32,487 - ERROR - Market SELL FAILED: APIError(code=-2019): Margin is insufficient.
2026-01-10 14:32:20,341 - ERROR - User input error: Order type must be market or limit
2026-01-10 14:32:21,703 - ERROR - User input error: Order type must be market or limit
2026-01-10 14:32:22,367 - ERROR - User input error: Order type must be market or limit
2026-01-10 14:32:22,699 - ERROR - User input error: Order type must be market or limit
2026-01-10 14:32:22,875 - ERROR - User input error: Order type must be market or limit
2026-01-10 14:32:23,043 - ERROR - User input error: Order type must be market or limit
2026-01-10 14:32:23,213 - ERROR - User input error: Order type must be market or limit
2026-01-10 14:32:23,360 - ERROR - User input error: Order type must be market or limit
2026-01-10 14:37:43,763 - ERROR - User input error: Symbol FHEW is invalid. Choose from: BTCUSDT, ETHUSDT, BCHUSDT,
2026-01-10 14:38:34,205 - ERROR - OCO FAILED: APIError(code=-4005): Quantity greater than max quantity.
2026-01-10 14:39:46,013 - ERROR - OCO FAILED: APIError(code=-4024): Limit price can't be lower than 86078.69.
2026-01-10 14:40:49,418 - ERROR - OCO FAILED: APIError(code=-4164): Order's notional must be no smaller than 100 (u
2026-01-10 14:41:45,870 - INFO - OCO Take Profit ORDER: {'orderId': 11561521548, 'symbol': 'BTCUSDT', 'status': 'NE
2026-01-10 14:41:46,042 - INFO - OCO Stop-Loss ORDER: {'orderId': 11561521650, 'symbol': 'BTCUSDT', 'status': 'NEW'
2026-01-10 15:55:32,941 - ERROR - User input error: Side must be BUY or SELL
2026-01-10 15:55:47,327 - ERROR - Market BUY FAILED: APIError(code=-2019): Margin is insufficient.
```

The **bot.log file** captures a complete and transparent record of the system's activity, tracking every order attempt made through the chatbot. It **logs both successful executions and failures,** including API errors, invalid user inputs, insufficient margin conditions, incorrect symbols, and constraint violations such as minimum notional requirements or tick-size limits. By timestamping each entry and **storing the exact message returned by Binance**, the log provides full traceability, helping debug issues, monitor performance, and verify that all order flows are working correctly end-to-end.

```
Order type (market/limit/stop-limit/oco/twap/grid, q to quit): marlet

Order type (market/limit/stop-limit/oco/twap/grid, q to quit): enc
Error: Order type must be one of: market, limit, stop-limit, oco, twap, grid

Order type (market/limit/stop-limit/oco/twap/grid, q to quit): market
Side (BUY/SELL): buy
Symbol (BTCUSDT, ETHUSDT, ...): btcusdt
Quantity: 1
Market BUY SUCCESS: {'orderId': 11567745516, 'symbol': 'BTCUSDT', 'status': 'NEW', 'clientOrderId': 'x-Cb7ytek
Qty': '1.000', 'executedQty': '0.000', 'cumQty': '0.000', 'cumQuote': '0.00000', 'timeInForce': 'GTC', 'type':
', 'positionSide': 'BOTH', 'stopPrice': '0.00', 'workingType': 'CONTRACT_PRICE', 'priceProtect': False, 'origT
PIRE_MAKER', 'goodTillDate': 0, 'updateTime': 1768057258724}

Order type (market/limit/stop-limit/oco/twap/grid, q to quit):
```

Overall, this project **demonstrates a fully functional, modular trading bot** built using the **Binance API**, capable of handling both basic and advanced order types in real-time. From **secure credential management and structured code organization to live order execution**, error handling, and detailed logging, the system closely models real-world algorithmic trading workflows. By **integrating market, limit, stop-limit, OCO, TWAP, and grid strategies, the bot showcases both flexibility and reliability**. This assignment not only strengthened my understanding of API-driven trading systems but also gave me hands-on experience with automation, system robustness, and disciplined software design.

Alongside this Binance Futures execution bot, I have also built a separate trading intelligence bot as part of another project. That system implements multiple algorithmic strategies including reinforcement learning, technical rule engines, hybrid machine learning models, and regime-adaptive pair systems to autonomously generate trade signals based on market data. While that project focuses on signal generation, backtesting, and model evaluation, the Binance bot showcased here focuses purely on order execution and automated trade placement on live markets.

GitHub Repository: https://github.com/KushagraGupta28/Quant-Hackathon