Binance Futures Trading Bot — Project Report

Name: Kavyashree M R
Course: B.Tech in Electronics and Communication
Date: 23 October 2025

1. Objective

The goal of this project is to design and implement a **command-line trading bot** for Binance USDT-M Futures that supports **multiple order types** including Market, Limit, Stop-Limit, OCO (One-Cancels-the-Other), TWAP, and Grid strategies.
The bot includes logging, input validation, and modular design.

2. Tools & Technologies Used

- **Programming Language:** Python 3.10+
- **Libraries:**
- requests — for API requests
- urllib3 — for HTTP connections
- python-dotenv — for environment configuration (optional)
- **Environment:** Windows PowerShell
- **Editor:** VS Code
- **Execution:** Command-line interface (CLI)

3. Implementation Details

The project structure follows modular design:

```
src/
 |----market_orders.py
|-----limit_orders.py
|-----utils.py
|-----config.py
 |-----advanced/
        |-----stop_limit.py
        |-----oco.py
        |-----twap.py
        |-----grid_orders.py
```

Each module implements a specific trading strategy:

- **market_orders.py** — executes immediate market trades
- **limit_orders.py** — places limit orders at specific prices
- **stop_limit.py** — sets stop-loss or breakout entries

- **oco.py** — manages simultaneous take-profit and stop-loss orders
- **twap.py** — splits large orders over time
- **grid_orders.py** — performs grid-based buy/sell automation

All execution logs are recorded in **bot.log** using the Python logging library.

4. Execution Procedure

Setup Steps:

1. Create virtual environment
   - python -m venv venv

2. Activate it
   - .\venv\Scripts\Activate

3. Install dependencies
   - pip install -r requirements.txt

4. Run sample commands
   - python -m src.market_orders --help
   - python -m src.market_orders BTCUSDT BUY 0.01
   - python -m src.limit_orders BTCUSDT BUY 0.01 45000
   - python -m src.advanced.stop_limit BTCUSDT SELL 0.01 43900 44000

5. Results and Logs

During execution, the bot successfully parsed CLI commands and generated structured logs.

Sample bot.log output:

```
2025-10-23 21:55:08,214 - utils - WARNING - API_SECRET not set — request unsigned (test mode).
2025-10-23 21:55:12,037 - utils - ERROR - API request failed: POST /fapi/v1/order -> 401 Client Error: Unauthorized for url
2025-10-23 21:55:12,079 - market_orders - ERROR - Failed to place market order
Traceback (most recent call last):
  ...
  requests.exceptions.HTTPError: 401 Client Error: Unauthorized for url
```

Explanation:
The bot attempted to connect to the Binance Testnet API without credentials, resulting in an *Unauthorized (401)* response.
This verifies that:

- API requests are formatted correctly
- Logging and error handling function as expected
- The bot can be easily switched to live mode by adding .env credentials

6. Screenshots

```
PS E:\KavyashreeMR_binance_bot> ni src\__init__.py, src\advanced\__init__.py


    Directory: E:\KavyashreeMR_binance_bot\src


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----        23-10-2025     21:42              0 __init__.py


    Directory: E:\KavyashreeMR_binance_bot\src\advanced


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----        23-10-2025     21:42              0 __init__.py
```

```
PS E:\KavyashreeMR_binance_bot> .\venv\Scripts\Activate
(venv) PS E:\KavyashreeMR_binance_bot> pip install --upgrade pip
Requirement already satisfied: pip in e:\kavyashreemr_binance_bot\venv\lib\site-packages (25.2)
```

```
(venv) PS E:\KavyashreeMR_binance_bot> pip install -r requirements.txt
Collecting requests==2.31.0 (from -r requirements.txt (line 1))
  Using cached requests-2.31.0-py3-none-any.whl.metadata (4.6 kB)
Collecting python-dotenv==1.0.0 (from -r requirements.txt (line 2))
  Downloading python_dotenv-1.0.0-py3-none-any.whl.metadata (21 kB)
Collecting urllib3==2.0.7 (from -r requirements.txt (line 3))
  Downloading urllib3-2.0.7-py3-none-any.whl.metadata (6.6 kB)
Collecting charset-normalizer<4,>=2 (from requests==2.31.0->-r requirements.txt (line 1))
  Using cached charset_normalizer-3.4.4-cp313-cp313-win_amd64.whl.metadata (38 kB)
Collecting idna<4,>=2.5 (from requests==2.31.0->-r requirements.txt (line 1))
  Using cached idna-3.11-py3-none-any.whl.metadata (8.4 kB)
Collecting certifi>=2017.4.17 (from requests==2.31.0->-r requirements.txt (line 1))
  Using cached certifi-2025.10.5-py3-none-any.whl.metadata (2.5 kB)
Using cached requests-2.31.0-py3-none-any.whl (62 kB)
Downloading urllib3-2.0.7-py3-none-any.whl (124 kB)
Downloading python_dotenv-1.0.0-py3-none-any.whl (19 kB)
Using cached charset_normalizer-3.4.4-cp313-cp313-win_amd64.whl (107 kB)
Using cached idna-3.11-py3-none-any.whl (71 kB)
Using cached certifi-2025.10.5-py3-none-any.whl (163 kB)
Installing collected packages: urllib3, python-dotenv, idna, charset-normalizer, certifi, requests
```

```
(venv) PS E:\KavyashreeMR_binance_bot> python -m src.market_orders --help
usage: market_orders.py [-h] [--position-side {BOTH,LONG,SHORT}] [--reduce-only] symbol {BUY,SELL,buy,sell} quantity

Place market order on Binance Futures

positional arguments:
  symbol
  {BUY,SELL,buy,sell}
  quantity

options:
  -h, --help            show this help message and exit
  --position-side {BOTH,LONG,SHORT}
  --reduce-only
```

```
(venv) PS E:\KavyashreeMR_binance_bot> python -m src.market_orders BTCUSDT BUY 0.01
❌ Order failed. Check bot.log for details.
```

```
2025-10-23 21:55:08,214 - utils - WARNING - API_SECRET not set — request unsigned (test mode).
2025-10-23 21:55:12,037 - utils - ERROR - API request failed: POST /fapi/v1/order -> 401 Client Error: Unauthorized for url: https://testnet.binancefuture.com/fapi/v1/order?
symbol=BTCUSDT&side=BUY&type=MARKET&quantity=0.01&positionSide=BOTH&timestamp=1761236708214&recvWindow=5000&signature=
```

7. Conclusion

The Binance Futures Trading Bot was successfully implemented with modular architecture, order management, and logging functionality.
Although executed in test mode, it demonstrates a full trading workflow and can be easily adapted for real Binance APIs.

Key Achievements:

- Modular Python architecture
- Command-line interface with argparse
- Structured logging with timestamps and error traces
- Extensible for future features like WebSockets, portfolio tracking, and backtesting

8. Future Enhancements

- Real-time monitoring using WebSocket streams
- Automated OCO order synchronization
- Telegram notifications for trade execution
- Web dashboard for portfolio visualization

Submitted by:
Kavyashree M R
B.Tech – Electronics and Communication Engineering