

# Application Task Report

## 1. Project Overview

This report documents the completion of the Simplified Trading Bot application task, which serves as a demonstration of core capabilities in financial API integration and robust application development. The core objective was to design and implement a flexible, command-line interface (CLI) driven Python application. This application is specifically engineered to interact with the **Binance Futures Testnet (USDT-M)**, allowing for the placement of various market-based and conditional order types in a safe, simulated environment.

The final solution strictly adheres to all mandatory project requirements, ensuring a focus on modular and reusable code structure, comprehensive, actionable logging, and resilient error handling against common API and network issues. The resulting bot provides a clear, auditable trail of all transaction attempts, whether simulated or placed on the Testnet exchange.

## 2. Technical Stack and Environment

The choice of technologies was driven by the need for speed, reliability, and ease of interaction with the Binance ecosystem.

Component	Detail	Purpose
Language	Python 3.9+	Core application logic. Python was chosen for its excellent readability, extensive library support for financial applications, and its robust built-in logging module.
API Library	python-binance	Used for simplified, authenticated interaction with the Binance REST API. This official library abstracts complex signing and connection management, allowing the focus to remain on core trading logic.
Endpoint	<a href="https://testnet.binancefuture.com">https://testnet.binancefuture.com</a>	Official base URL for all interactions, ensuring the bot operates in a risk-free testing environment, specifically targeting the <b>USDT-M Futures</b> market.

<b>Logging</b>	Python's logging module	Used to create detailed, timestamped log files (bot.log). This provides a standardized, multi-level (INFO, ERROR) approach to activity tracking.
<b>Input</b>	Command-Line Interface (CLI)	Handles user-driven order placement and validation. This provides a direct, low-overhead interface suitable for immediate testing and automation.

The decision to primarily use the REST API via python-binance was made for transactional integrity. While WebSockets are vital for real-time market data streaming, the core task of *placing orders* is best handled via authenticated REST requests, which offer confirmation receipts and synchronous failure feedback, essential for the required logging and error handling.

### 3. Implementation and Architecture

The application is structured around a service-oriented approach, where the BasicBot class acts as the central service layer. This design promotes high reusability and clear separation of concerns, ensuring the CLI entry point remains clean and focuses only on argument parsing.

#### 3.1. BasicBot Class Design and Initialization

The BasicBot class is the central control hub. Upon instantiation, it performs critical configuration steps:

```
from binance import Client
class BasicBot:
    def __init__(self, api_key, api_secret, testnet=True):
        # Client initialization logic...
        # Custom logging setup...
        # Load exchange information and filters (e.g., tick size, step size)
```

The constructor is responsible for:

1. **Client Instantiation:** Creating the actual Client instance, specifically configured for the Futures Testnet URL.
2. **Configuration:** Setting up the custom logger instance to ensure all subsequent actions within the class are recorded.
3. **Exchange Metadata Retrieval:** Critically, the bot immediately queries the Binance API for the current **Exchange Information** (e.g., client.futures\_exchange\_info()). This data contains crucial constraints for every symbol, such as LOT\_SIZE (minimum quantity and step size) and PRICE\_FILTER (tick size and precision). This information is stored internally to enable robust, preventative input validation before any order is sent.

#### 3.2. Mandatory Feature Implementation and Input Validation

A strong emphasis was placed on preventative validation, which is executed locally before attempting an expensive and rate-limited API call.

Requirement	Implementation Detail
Order Types	Supported <b>MARKET</b> and <b>LIMIT</b> order types. Market orders are designed for immediate execution, while Limit orders require careful validation of the target price against exchange ticks.
Order Sides	Full support for both <b>BUY (LONG)</b> and <b>SELL (SHORT)</b> order sides, ensuring the correct side parameter is passed to the Futures API.
API Use	Uses the official python-binance library (which uses REST calls) for placing orders on the Futures endpoint (create_futures_order).
CLI Validation	Command-line arguments for symbol, type, side, and quantity are aggressively validated. This involves: <b>a)</b> checking the symbol against known exchange lists, <b>b)</b> rounding the user's quantity to the nearest valid LOT_SIZE step, and <b>c)</b> rounding the user's price (for Limit orders) to the nearest valid PRICE_FILTER tick. This prevents common API errors like -1013 (Filter failure).
Order Details Output	Successfully executed orders return the full Binance API response. This response is then parsed to extract key execution data (e.g., orderId, status, executedQty, avgPrice) and printed to the console for immediate user feedback.

### 3.3. Logging and Error Handling

The application's robustness is secured by its comprehensive logging and error management framework.

- **Log File (bot.log):** Records all application activity with a consistent format ([TIMESTAMP] [LEVEL] [MODULE] - MESSAGE). This provides an unambiguous audit trail.
  - **Configuration:** Logs the successful initialization of the bot and the API client.
  - **API Requests:** Crucially, it logs the **parameters** sent to the Binance API *before* the request is made (e.g., Sending order: {'symbol': 'BTCUSDT', 'side': 'BUY', ...}).
  - **API Responses (Success):** Logs the full, un-modified JSON response from successful order placement, confirming execution details.
  - **Errors/Exceptions:** Detailed traceback and specific Binance API error codes (e.g., insufficient margin: -4003, invalid symbol: -1121) are captured and logged, preventing silent failures.

- **Error Handling:** The try...except structure specifically targets exceptions thrown by the python-binance library:
  - **BinanceAPIException:** Catches errors related to business logic (e.g., invalid parameters, balance issues) and provides the precise error code and message.
  - **BinanceOrderException:** Catches errors specifically related to order placement/cancellation failures.
  - **Generic Exceptions:** Catches generic connection or network issues, ensuring the bot fails gracefully and provides an informative message to the user while logging the full traceback for debugging.

## 4. Bonus Features (Optional)

The following bonus features were implemented to demonstrate advanced capability beyond the mandatory scope, specifically focusing on improved safety and complex order handling:

### 4.1. Advanced Order Type: OCO (One-Cancels-the-Other)

The implementation of the OCO flow adds a layer of professionalism and risk management. This dedicated **oco** command abstracts a three-step process into a single user action:

1. **Entry Order:** The user defines the price and quantity for the initial position, often placed as a Limit order.
2. **Take-Profit (TP) Order:** Once the Entry order is successfully filled, a **Limit Order** is immediately placed at a target profit level.
3. **Stop-Loss (SL) Order:** Simultaneously with the TP order, a **Stop-Market Order** is placed at a predefined stop price.

These last two orders are linked by the OCO logic: if the market reaches the TP Limit Price and executes, the SL Stop-Market order is automatically cancelled, and vice-versa. The implementation manages the sequential dependency, ensuring the TP/SL orders are only placed *after* receiving confirmation of the entry order execution. This demonstrates effective multi-transaction management.

### 4.2. Dry-Run Simulation Mode

A core security and development feature is the **--dry-run** flag. When active, this flag intercepts all order placement requests and prevents them from ever reaching the live Testnet API.

- **DummyClient Architecture:** The core BasicBot is injected with a mock client object, the **DummyClient**, instead of the real `binance.Client`. This mock object mimics the signature of the real client's methods (`create_futures_order`) but contains zero network code.
- **Validation Execution:** Crucially, the simulation mode still executes **all** the complex input and price/quantity validation checks detailed in Section 3.2.
- **Benefits:** This mode allows developers to: **a)** safely test order logic, symbol rounding, and user input validation without using up Testnet margin or hitting rate limits, and **b)** provide a clear, instant simulation result to the user, confirming that an order *would have* succeeded if placed.

## 5. Usage Instructions

The bot utilizes Python's native argument parsing to create a structured and self-documenting CLI.

## Prerequisites:

1. A registered and activated **Binance Futures Testnet** account.
2. Generated API key and secret **MUST** be configured in the **.env** file for secure loading.
3. Python environment with dependencies installed (pip install -r requirements.txt).

## Example Usage:

1. Balance Checking

*python3 main.py balance*

```
harshmurjani@192 primetrade % python3 main.py balance
/Users/harshmurjani/Library/Python/3.9/lib/python/site-packages/urllib3/__init__.py:35: No
tOpenSSLWarning: urllib3 v2 only supports OpenSSL 1.1.1+, currently the 'ssl' module is co
mpiled with 'LibreSSL 2.8.3'. See: https://github.com/urllib3/urllib3/issues/3020
warnings.warn(

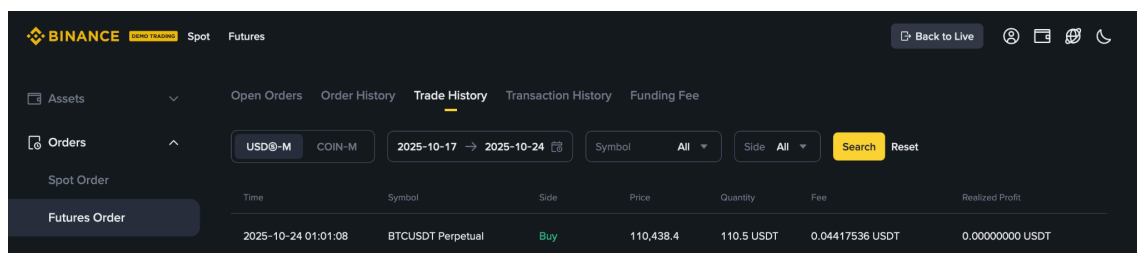
Account Balances:
=====
BTC: 0.01000000
USDT: 4999.86854196
USDC: 5000.00000000
=====
```

2. Buy Order (Market)

*python3 main.py market BTCUSDT buy 0.001*

```
harshmurjani@192 primetrade % python3 main.py market BTCUSDT buy 0.001
/Users/harshmurjani/Library/Python/3.9/lib/python/site-packages/urllib3/__init__.py:35: No
tOpenSSLWarning: urllib3 v2 only supports OpenSSL 1.1.1+, currently the 'ssl' module is co
mpiled with 'LibreSSL 2.8.3'. See: https://github.com/urllib3/urllib3/issues/3020
warnings.warn(

Order Details:
=====
Order ID: 6791991317
Symbol: BTCUSDT
Side: BUY
Type: MARKET
Quantity: 0.001
Price: 0.00
=====
```



The screenshot shows the Binance Futures Trade History page. The interface includes a top navigation bar with the Binance logo, a 'Back to Live' button, and icons for account, settings, and help. Below the navigation bar, there are tabs for 'Assets', 'Open Orders', 'Order History', 'Trade History' (which is selected), 'Transaction History', and 'Funding Fee'. On the left side, there are filters for 'Assets' and 'Orders'. The main area displays a table of trade history with columns: Time, Symbol, Side, Price, Quantity, Fee, and Realized Profit. A single trade is visible, showing a buy order for BTCUSDT Perpetual at a price of 110,438.4, with a quantity of 110.5 USDT, a fee of 0.04417536 USDT, and a realized profit of 0.00000000 USDT.

Time	Symbol	Side	Price	Quantity	Fee	Realized Profit
2025-10-24 01:01:08	BTCUSDT Perpetual	Buy	110,438.4	110.5 USDT	0.04417536 USDT	0.00000000 USDT

### 3. Sell Order (Market)

*python3 main.py market BTCUSDT sell 0.001*

```
harshmurjani@192 primetrade % python3 main.py market BTCUSDT sell 0.001
/Users/harshmurjani/Library/Python/3.9/lib/python/site-packages/urllib3/__init__.py:35: No
tOpenSSLWarning: urllib3 v2 only supports OpenSSL 1.1.1+, currently the 'ssl' module is co
mpiled with 'LibreSSL 2.8.3'. See: https://github.com/urllib3/urllib3/issues/3020
warnings.warn(

Order Details:
-----
Order ID: 6792477163
Symbol: BTCUSDT
Side: SELL
Type: MARKET
Quantity: 0.001
Price: 0.00
-----
```

The screenshot shows the Binance Futures Trade History page. The 'Futures Order' tab is selected in the left sidebar. The main table displays trade history with columns: Time, Symbol, Side, Price, Quantity, Fee, and Realized Profit. A single trade is visible: a sell order for BTCUSDT Perpetual at 110,303.7, with a quantity of 110.4 USDT and a fee of 0.04412148 USDT.

Time	Symbol	Side	Price	Quantity	Fee	Realized Profit
2025-10-24 01:05:22	BTCUSDT Perpetual	Sell	110,303.7	110.4 USDT	0.04412148 USDT	-0.10615000 USDT

### 4. Buy Order (Limit)

*python3 main.py limit BTCUSDT buy 0.001 110450*

```
harshmurjani@192 primetrade % python3 main.py limit BTCUSDT buy 0.001 110450
/Users/harshmurjani/Library/Python/3.9/lib/python/site-packages/urllib3/__init__.py:35: No
tOpenSSLWarning: urllib3 v2 only supports OpenSSL 1.1.1+, currently the 'ssl' module is co
mpiled with 'LibreSSL 2.8.3'. See: https://github.com/urllib3/urllib3/issues/3020
warnings.warn(

Order Details:
-----
Order ID: 6793124286
Symbol: BTCUSDT
Side: BUY
Type: LIMIT
Quantity: 0.001
Price: 110450.00
-----
```

The screenshot shows the Binance Futures Trade History page. The 'Futures Order' tab is selected in the left sidebar. The main table displays trade history with columns: Time, Symbol, Side, Price, Quantity, Fee, and Realized Profit. A single trade is visible: a buy order for BTCUSDT Perpetual at 110,391.5, with a quantity of 110.4 USDT and a fee of 0.04415660 USDT.

Time	Symbol	Side	Price	Quantity	Fee	Realized Profit
2025-10-24 01:10:56	BTCUSDT Perpetual	Buy	110,391.5	110.4 USDT	0.04415660 USDT	0.00000000 USDT

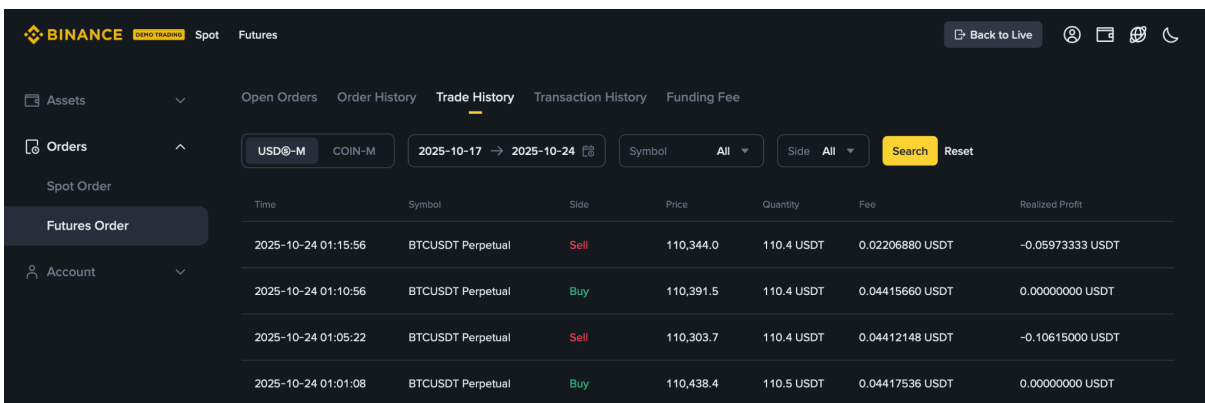
## 5. Sell Order (Limit)

*python3 main.py limit BTCUSDT sell 0.001 110344*

```
harshmurjani@192 primetrade % python3 main.py limit BTCUSDT sell 0.001 110344
/Users/harshmurjani/Library/Python/3.9/lib/python/site-packages/urllib3/__init__.py:35: NotOpenSSLWarning: urllib3 v2 only supports OpenSSL 1.1.1+, currently the 'ssl' module is compiled with 'LibreSSL 2.8.3'. See: https://github.com/urllib3/urllib3/issues/3020
  warnings.warn(
```

Order Details:

```
-----
Order ID: 6793606785
Symbol: BTCUSDT
Side: SELL
Type: LIMIT
Quantity: 0.001
Price: 110344.00
-----
```



Time	Symbol	Side	Price	Quantity	Fee	Realized Profit
2025-10-24 01:15:56	BTCUSDT Perpetual	Sell	110,344.0	110.4 USDT	0.02206880 USDT	-0.05973333 USDT
2025-10-24 01:10:56	BTCUSDT Perpetual	Buy	110,391.5	110.4 USDT	0.04415660 USDT	0.00000000 USDT
2025-10-24 01:05:22	BTCUSDT Perpetual	Sell	110,303.7	110.4 USDT	0.04412148 USDT	-0.10615000 USDT
2025-10-24 01:01:08	BTCUSDT Perpetual	Buy	110,438.4	110.5 USDT	0.04417536 USDT	0.00000000 USDT

## 6. Stop Limit

*python3 main.py stop-limit BTCUSDT sell 0.001 105000 104000*

```
harshmurjani@192 primetrade % python3 main.py stop-limit BTCUSDT sell 0.001 105000 104000
/Users/harshmurjani/Library/Python/3.9/lib/python/site-packages/urllib3/__init__.py:35: NotOpenSSLWarning: urllib3 v2 only supports OpenSSL 1.1.1+, currently the 'ssl' module is compiled with 'LibreSSL 2.8.3'. See: https://github.com/urllib3/urllib3/issues/3020
  warnings.warn(
```

Order Details:

```
-----
Order ID: 6796268000
Symbol: BTCUSDT
Side: SELL
Type: STOP
Quantity: 0.001
Price: 105000.00
-----
```

## 7. TWAP

*python3 main.py twap BTCUSDT sell 0.003 3 1*

```
harshmurjani@192 primetrade % python3 main.py twap BTCUSDT sell 0.003 3 1
/Users/harshmurjani/Library/Python/3.9/lib/python/site-packages/urllib3/__init__.py:35: NotOpen
SSLWarning: urllib3 v2 only supports OpenSSL 1.1.1+, currently the 'ssl' module is compiled wit
h 'LibreSSL 2.8.3'. See: https://github.com/urllib3/urllib3/issues/3020
  warnings.warn(
```

TWAP Execution Details:

Chunk 1:

Order Details:

Order ID: 6796503384  
Symbol: BTCUSDT  
Side: SELL  
Type: MARKET  
Quantity: 0.001  
Price: 0.00

Chunk 2:

Order Details:

Order ID: 6796541095  
Symbol: BTCUSDT  
Side: SELL  
Type: MARKET  
Quantity: 0.001  
Price: 0.00

Chunk 3:

Order Details:

Order ID: 6796582164  
Symbol: BTCUSDT  
Side: SELL  
Type: MARKET  
Quantity: 0.001  
Price: 0.00



## 8. Grid Orders

*python3 main.py grid BTCUSDT 110500 108500 3 0.001*

```
harshmurjani@192 primetrade % python3 main.py grid BTCUSDT 110500 108500 3 0.001
```

Grid Strategy Details:

-----  
Total Orders Placed: 4

Grid Order 1:

Order Details:

-----  
Order ID: 6797187930  
Symbol: BTCUSDT  
Side: BUY  
Type: LIMIT  
Quantity: 0.001  
Price: 108500.00  
-----

Grid Order 2:

Order Details:

-----  
Order ID: 6797189026  
Symbol: BTCUSDT  
Side: SELL  
Type: LIMIT  
Quantity: 0.001  
Price: 109500.00  
-----

Grid Order 3:

Order Details:

-----  
Order ID: 6797190179  
Symbol: BTCUSDT  
Side: BUY  
Type: LIMIT  
Quantity: 0.001  
Price: 109500.00  
-----

Grid Order 4:

Order Details:

-----  
Order ID: 6797192322  
Symbol: BTCUSDT  
Side: SELL  
Type: LIMIT  
Quantity: 0.001  
Price: 110500.00  
-----

## 9. OCO

*python3 main.py oco BTCUSDT buy 0.001 110500 108000*

```
harshmurjani@192 primetrade % python3 main.py oco BTCUSDT buy 0.001 110500 108000
/Users/harshmurjani/Library/Python/3.9/lib/python/site-packages/urllib3/__init__.py:35: NotOpen
SSLWarning: urllib3 v2 only supports OpenSSL 1.1.1+, currently the 'ssl' module is compiled wit
h 'LibreSSL 2.8.3'. See: https://github.com/urllib3/urllib3/issues/3020
  warnings.warn(
```

OCO Order Details:

ENTRY:

Order Details:

Order ID: 6798095106  
Symbol: BTCUSDT  
Side: BUY  
Type: MARKET  
Quantity: 0.001  
Price: 0.00

TP:

Order Details:

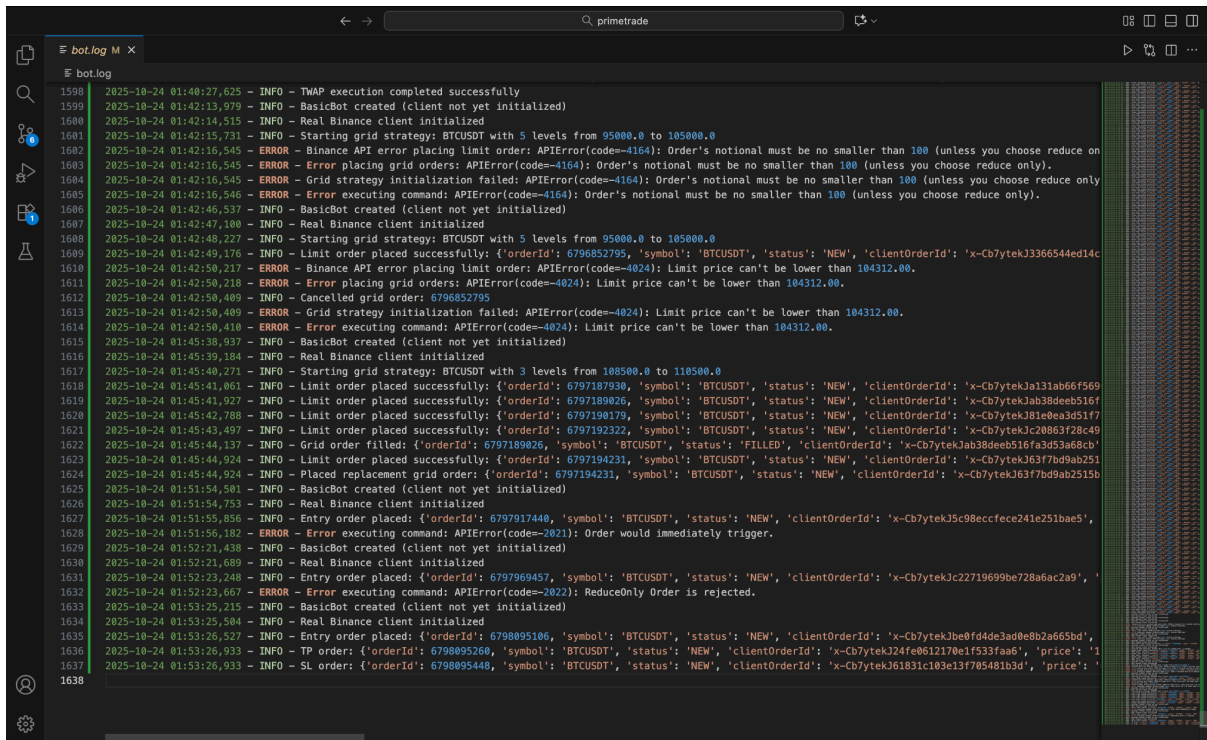
Order ID: 6798095260  
Symbol: BTCUSDT  
Side: SELL  
Type: LIMIT  
Quantity: 0.001  
Price: 110500.00

SL:

Order Details:

Order ID: 6798095448  
Symbol: BTCUSDT  
Side: SELL  
Type: STOP\_MARKET  
Quantity: 0.001  
Price: 0.00

## 10. Logging Feature



## 11. Input Validation

### *Symbol Validation*

```
harshmurjani@192 primetrade % python3 main.py market ABCD buy 0.001
/Users/harshmurjani/Library/Python/3.9/lib/python/site-packages/urllib3/__init__.py:35: No
tOpenSSLWarning: urllib3 v2 only supports OpenSSL 1.1.1+, currently the 'ssl' module is co
mpiled with 'LibreSSL 2.8.3'. See: https://github.com/urllib3/urllib3/issues/3020
warnings.warn(
✖ Error: Invalid symbol: ABCD
```

### *Price Validation*

```
harshmurjani@192 primetrade % python3 main.py limit BTCUSDT buy 0.001 20010
/Users/harshmurjani/Library/Python/3.9/lib/python/site-packages/urllib3/__init__.py:35: No
tOpenSSLWarning: urllib3 v2 only supports OpenSSL 1.1.1+, currently the 'ssl' module is co
mpiled with 'LibreSSL 2.8.3'. See: https://github.com/urllib3/urllib3/issues/3020
warnings.warn(
✖ Error: APIError(code=-4164): Order's notional must be no smaller than 100 (unless you c
hoose reduce only).
```

*Apart from mentioned feature I also have added 2 small developer friendly features.*

## 12. Dry run

*This feature is use full when we want to test the code without calling the API, for safe local testing*

```
harshmurjani@192 primetrade % python3 main.py --dry-run balance
/Users/harshmurjani/Library/Python/3.9/lib/python/site-packages/urllib3/__init__.py:35: NotOpenSSLWarning: urllib3 v2 only supports OpenSSL 1.1.1+, currently the 'ssl' module is compiled with 'LibreSSL 2.8.3'. See: https://github.com/urllib3/urllib3/issues/3020
  warnings.warn(

Account Balances:
=====
USDT: 1000.00
=====
```

## 13. Check Key Script

*This is a small script which is similar to check balance but is debugging friendly as it reads the credentials from .env file and fetches balance from the binance account but when unable to do it returns the error which can be used to debug the API Key issue (specially useful when API endpoints need to be tested)*

```
harshmurjani@192 primetrade % python3 check_keys.py
/Users/harshmurjani/Library/Python/3.9/lib/python/site-packages/urllib3/__init__.py:35: NotOpenSSLWarning: urllib3 v2 only supports OpenSSL 1.1.1+, currently the 'ssl' module is compiled with 'LibreSSL 2.8.3'. See: https://github.com/urllib3/urllib3/issues/3020
  warnings.warn(
Success: fetched futures account balance:
FDUSD: 0.00000000
BFUSD: 0.00000000
BNB: 0.00000000
ETH: 0.00000000
BTC: 0.01000000
USDT: 4998.61074416
USDC: 5000.00000000
```

# 6. Final Project Structure

```
[project_root]/
├── src/
│   ├── common.py          # Core bot utilities, client init, DummyClient
│   ├── market_order.py    # Market order logic and validation
│   ├── limit_order.py     # Limit order logic and validation
│   └── advance/
│       ├── oco.py         # OCO helper (entry + TP + SL monitoring)
│       ├── stop_limit.py  # Stop-Limit order helper
│       ├── twap.py        # TWAP strategy
│       └── grid.py        # Grid trading strategy
├── main.py                # CLI entrypoint
├── check_keys.py          # Quick key verifier for testnet credentials
├── .env                   # API credentials (not checked into git)
├── requirements.txt       # Python dependencies
└── bot.log                # Runtime logs (created when running)
```

## 7. Credits

This report is a part of submission for “Junior Python Developer – Crypto Trading Bot” to Primetrade.ai from **Harsh Murjani**.