

Project Title: StockWatch & CalcPro

Project Overview

StockWatch & CalcPro is a dual-purpose application that provides:

1. **StockWatch:** A real-time stock market monitoring dashboard with graphical representation and alerts.
2. **CalcPro:** A console-based calculator with optional GUI and history logging.

Features

StockWatch Module

1. Real-Time Stock Data:

- Fetch real-time stock prices using an API (e.g., Alpha Vantage or Yahoo Finance).
- Display data for user-specified stock symbols (e.g., AAPL, TSLA).

2. Graphical Representation:

- Use matplotlib or plotly to display stock price trends (e.g., line charts, candlestick charts).

3. Alerts:

- Set up alerts for specific conditions (e.g., price crossing a threshold, percentage change).

4. Data Storage:

- Store historical stock data in an SQLite database or CSV file.

5. Reports:

- Generate reports summarizing stock performance over time using Pandas.

CalcPro Module

1. Basic Arithmetic Operations:

- Addition, subtraction, multiplication, and division.
- Support for both integer and decimal inputs.

2. Error Handling:

- Handle invalid inputs (e.g., non-numeric values).
- Handle edge cases like division by zero.

3. User Interface:

- Console-based interface for simplicity.
- Optional: Use Tkinter for a GUI with buttons and input fields.

4. History Logging:

- Save performed operations to a text file or SQLite database.
- Optional: Use Pandas to store history in a table format.

Technologies

- **Python:** Core logic and functionality.

- **APIs:** Alpha Vantage, Yahoo Finance, or IEX Cloud for stock data.
- **Matplotlib/Plotly:** For graphical representation.
- **SQLite:** For storing historical data and calculator history.
- **Pandas:** For data manipulation and report generation.
- **Tkinter:** Optional GUI for the calculator.
- **Logging:** For generating logs.

StockWatch_CalcPro/

|

└─ stockwatch/

| └─ main.py # Main script for stock dashboard

| └─ data_fetcher.py # Script to fetch stock data

| └─ visualizer.py # Script for graphical
representation

| └─ alerts.py # Script for market trend alerts

| └─ database.py # Script for database operations

| └─ reports.py # Script for generating reports

| └─ README.md # Documentation for

StockWatch

|

└─ calcpro/

	— main.py	# Main script for calculator
	— gui_calculator.py	# Optional GUI-based calculator
	— history_logger.py	# Script for saving history
	— reports.py	# Script for generating reports
	— README.md	# Documentation for CalcPro
	— requirements.txt	# List of dependencies
	— README.md	# Overall project documentation

Deliverables

1. Python Scripts:

- Scripts for StockWatch and CalcPro modules.
- Optional GUI script for the calculator.

2. Database:

- SQLite database for storing stock data and calculator history.

3. Reports:

- CSV or PDF reports summarizing stock performance and calculator logs.

4. Documentation:

- Instructions for setting up and using the application.

- Explanation of features, error handling, and optional components.

Steps to Build the Project

Step 1: Set Up the Environment

1. Install Python (if not already installed).
2. Install required libraries:

code

`pip install matplotlib pandas requests sqlite3 tkinter`

Obtain an API key for stock data (e.g., from Alpha Vantage).

Step 2: Develop StockWatch Module

1. Fetch Real-Time Stock Data:

- Use an API to fetch stock prices.
- Example:

import requests

code

```
def fetch_stock_data(symbol, api_key):
```

```
    url =
```

```
    f"https://www.alphavantage.co/query?function=TIME_SERIE  
S_INTRADAY&symbol={symbol}&interval=5min&apikey={api_  
key}"
```

```
    response = requests.get(url)
```

```
    data = response.json()
```

return data

Display Graphical Representation:

- Use matplotlib to plot stock prices.
- Example:

code

```
import matplotlib.pyplot as plt
```

```
def plot_stock_prices(prices):
```

```
    plt.plot(prices)
```

```
    plt.title("Stock Price Trend")
```

```
    plt.xlabel("Time")
```

```
    plt.ylabel("Price")
```

```
    plt.show()
```

Set Up Alerts:

- Check for specific conditions (e.g., price > threshold).
- Example:

Code

```
def check_alert(price, threshold):
```

```
    if price > threshold:
```

```
        print(f"Alert: Price crossed {threshold}!")
```

Store Historical Data:

- Use SQLite to store stock data.
- Example:

```
import sqlite3
```

```
def save_to_database(symbol, price, timestamp):  
    conn = sqlite3.connect("stocks.db")  
    cursor = conn.cursor()  
    cursor.execute("CREATE TABLE IF NOT EXISTS stocks  
(symbol TEXT, price REAL, timestamp TEXT)")  
    cursor.execute("INSERT INTO stocks VALUES (?, ?, ?)",  
(symbol, price, timestamp))  
    conn.commit()  
    conn.close()
```

Step 3: Develop CalcPro Module

1. Implement Basic Arithmetic Operations:

- Example:

```
def add(x, y):  
    return x + y
```

```
def subtract(x, y):  
    return x - y
```

```
def multiply(x, y):
```

```
    return x * y
```

```
def divide(x, y):
```

```
    if y == 0:
```

```
        raise ValueError("Cannot divide by zero!")
```

```
    return x / y
```

Handle Errors:

- Example:

```
try:
```

```
    result = divide(10, 0)
```

```
except ValueError as e:
```

```
    print(e)
```

1. Optional GUI:

- Use Tkinter to create a GUI for the calculator.

Save History:

- Example:

code

```
def save_history(operation, result):
```

```
    with open("history.txt", "a") as file:
```



```
file.write(f"{operation} = {result}\n")
```

Step 4: Test the Application

- Test for edge cases (e.g., invalid inputs, API rate limits).
- Ensure all features work as expected.

Step 5: Document the Application

- Write clear instructions for setup and usage.
- Explain features, error handling, and optional components.

Step 6: Package and Submit

- Include all Python scripts, configuration files, and documentation.
- Optionally, create an executable using pyinstaller.