

**TRƯỜNG ĐẠI HỌC KỸ THUẬT CÔNG NGHIỆP
KHOA ĐIỆN TỬ**

BỘ MÔN: CÔNG NGHỆ THÔNG TIN



**BÀI TẬP LỚN
LẬP TRÌNH PYTHON**

ĐỀ TÀI: HỆ THỐNG THEO DÕI CỔ PHIẾU

Giáo viên hướng dẫn:

Sinh viên thực hiện:

Ngành học:

Lớp:

Đỗ Duy Cốp

Đặng Thị Hà

Kỹ thuật máy tính

K56KMT.01

Thái Nguyên 2024

NHIỆM VỤ BÀI TẬP LỚN MÔN LẬP TRÌNH PYTHON

Sinh viên: Đặng Thị Hà

MSSV : K205480106015

Lớp: K56KMT

Khoá: K56

Ngành học: Kỹ thuật máy tính

Giáo viên hướng dẫn: Đỗ Duy Cốp

1. Tên đề tài : **HỆ THỐNG THEO DÕI CỔ PHIẾU**

2. Nội dung thực hiện:

- Tạo cơ sở dữ liệu SQL để lưu trữ dữ liệu về cổ phiếu, bao gồm thông tin như giá cả, thay đổi giá, thời gian.
- Sử dụng FastAPI để tạo các endpoint API để truy xuất dữ liệu cổ phiếu từ cơ sở dữ liệu.
- Sử dụng Node-RED để kết nối và lấy dữ liệu cổ phiếu từ các nguồn khác nhau như API của các trang web tài chính hoặc các dịch vụ cung cấp dữ liệu cổ phiếu.
- Hiển thị biểu đồ để biểu diễn giá cả cổ phiếu

3. Các sản phẩm, kết quả :

- a. Sản phẩm phần mềm theo yêu cầu của bài tập lớn.
- b. Thuyết minh đồ án theo mẫu chung của khoa Điện tử.

4. Ngày giao nhiệm vụ: 15/05/2024

5. Ngày hoàn thành nhiệm vụ: 26/05/2024

GIÁO VIÊN HƯỚNG DẪN

Đỗ Duy Cốp

MỤC LỤC

MỤC LỤC.....	3
CHƯƠNG 1 : TỔNG QUAN ĐỀ TÀI.....	4
1.1. Yêu cầu của đề bài	4
1.2. Mục tiêu của đề tài	4
1.3. Các mục cần làm	4
CHƯƠNG 2: CƠ SỞ LÝ THUYẾT	5
2.1. Giới thiệu ngôn ngữ lập trình Python	5
2.2. SQL Server Management Studio	6
2.3. Node-red.....	6
CHƯƠNG 3: QUÁ TRÌNH THỰC HIỆN.....	8
CHƯƠNG 4: KẾT LUẬN	21

CHƯƠNG 1 : TỔNG QUAN ĐỀ TÀI

1.1. Yêu cầu của đề bài

- Dùng FastApi của python, xây dựng API (tự đưa vào logic xử lý input => output)
- Cài đặt Node-Red trên windows, tạo chu trình tự động hoá gửi dữ liệu tới api, nhận về kết quả, lưu trữ vào database Sql server.
- Tạo web đơn giản (html+js+css) với backend có thể là c# asp dot net, php, node-red, hoặc chính là python FastApi để lấy dữ liệu từ database Sql server, vẽ biểu đồ dữ liệu đã lưu. (Chart có thể dùng tùy ý thư viện thích hợp)

1.2. Mục tiêu của đề tài

- Lấy dữ liệu cổ phiếu.
- Xử lý dữ liệu :sử dụng FastAPI và Node-RED, sau đó lưu vào cơ sở dữ liệu.
- Xây dựng trang web

1.3. Các mục cần làm

- Cài đặt Node-Red
- Tạo CSDL trên SQL
- Kết nối SQL với Node – red
- Hiển thị lên web

CHƯƠNG 2: CƠ SỞ LÝ THUYẾT

2.1. Giới thiệu ngôn ngữ lập trình Python

Python là một ngôn ngữ lập trình được tạo ra bởi Guido van Rossum và phát hành lần đầu vào năm 1991. Ban đầu, Python được phát triển như một dự án sở thích nhằm tạo ra một ngôn ngữ lập trình dễ đọc và dễ viết hơn so với các ngôn ngữ hiện có. Tên "Python" được lấy cảm hứng từ nhóm hài kịch Monty Python, thể hiện sự hài hước và dễ tiếp cận của ngôn ngữ này.



Đặc điểm chính của python

- Dễ Học và Sử Dụng
- Đa Nền Tảng
- Thư Viện Phong Phú
- Mã Nguồn Mở

Các công cụ và môi trường phát triển của Python

- IDLE: Một môi trường phát triển tích hợp đơn giản đi kèm với Python, phù hợp cho người mới bắt đầu.
- PyCharm: Một IDE mạnh mẽ dành cho Python, hỗ trợ nhiều tính năng như gỡ lỗi, quản lý phiên bản, và tự động hoàn thành mã.

- Jupyter Notebook: Một công cụ mạnh mẽ cho phép viết và chạy mã Python trong một môi trường tương tác, rất phổ biến trong lĩnh vực khoa học dữ liệu. Nó cho phép người dùng tạo và chia sẻ tài liệu chứa mã nguồn, hình ảnh, và các chú thích.

2.2. SQL Server Management Studio

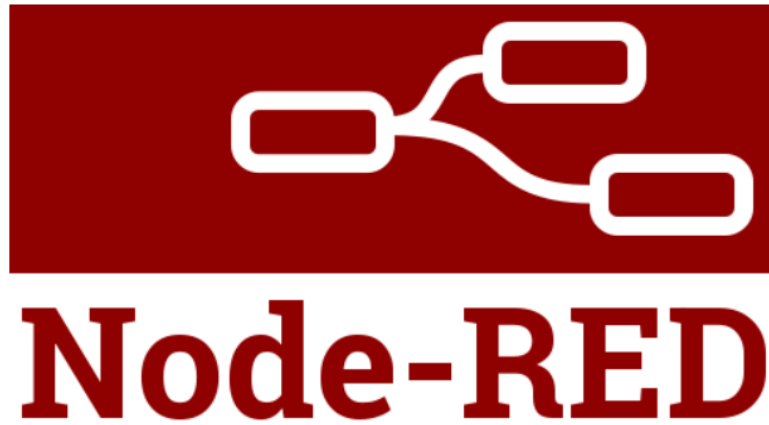
SQL Server Management Studio được viết tắt là SSMS, là một ứng dụng phần mềm được giới thiệu lần đầu với Microsoft SQL Server 2005, được dùng để quản lý, cấu hình tất cả các thành phần trong Microsoft SQL Server.



SQL Server cung cấp cho người dùng các công cụ và tính năng để quản lý, lưu trữ, xử lý các truy vấn dữ liệu, kiểm soát truy cập, xử lý giao diện và hỗ trợ tích hợp dữ liệu từ nhiều nguồn khác nhau.

2.3. Node-red

Node-RED là một công cụ mã nguồn mở và trực quan được sử dụng để xây dựng các luồng làm việc (workflows) và ứng dụng Internet of Things (IoT). Nó cung cấp một giao diện đồ họa dựa trên trình duyệt web, cho phép người dùng kết nối các nút (node) với nhau để xử lý dữ liệu và tương tác với các thiết bị và dịch vụ khác nhau.



Node-RED được xây dựng trên nền tảng Node.js và sử dụng trình duyệt web để tạo ra một giao diện dễ sử dụng. Người dùng có thể kéo và thả các nút từ thư viện có sẵn để tạo ra luồng làm việc theo ý muốn. Các nút có thể thực hiện các nhiệm vụ khác nhau, bao gồm xử lý dữ liệu, kết nối và tương tác với các dịch vụ web, cơ sở dữ liệu, thiết bị IoT và nhiều hơn nữa.

Node-RED có một cộng đồng rộng lớn và có sẵn nhiều bộ nút mở rộng, cho phép người dùng tương tác với các nền tảng và dịch vụ phổ biến như MQTT, HTTP, MySQL, MongoDB, Twitter, Raspberry Pi, Arduino và nhiều hơn nữa. Công cụ này rất linh hoạt và phù hợp cho việc phát triển các ứng dụng IoT, tự động hóa và quản lý dữ liệu.

CHƯƠNG 3: QUÁ TRÌNH THỰC HIỆN

CÁC BƯỚC THỰC HIỆN

Bước 1. Tạo file python sử dụng FastAPI để lấy dữ liệu cổ phiếu

Em sẽ tạo một file python và sử dụng code py FastAPI trên trang web:
<https://finnhub.io/> để lấy dữ liệu cổ phiếu mà mình muốn

```
import asyncio
from fastapi import FastAPI, HTTPException
import aiohttp
from datetime import datetime, timedelta, timezone

app = FastAPI()

API_KEY = 'cp3mdl9r0lqs3665mih0cp3mdl9r0lqs3665mihg'
BASE_URL = 'https://finnhub.io/api/v1'
STOCK_SYMBOL = 'BCM'

# Define UTC and VN timezones
UTC_timezone = timezone.utc
VN_timezone = timezone(timedelta(hours=7)) # UTC+7

# Biến toàn cục để lưu trữ dữ liệu cổ phiếu
stock_data_cache = {}

async def fetch_stock_data(symbol):
    params = {
        "symbol": symbol,
        "token": API_KEY
    }
    async with aiohttp.ClientSession() as session:
        async with session.get(f"{BASE_URL}/quote", params=params) as response:
            data = await response.json()
            print("API response data:", data) # In ra kết quả API call
            if 'c' not in data:
                raise HTTPException(status_code=404, detail="Stock symbol not found")
            # Extract relevant information
```



```

        stock_data = {
            "symbol": symbol,
            "price": data["c"],
            "change": data["d"],
            "last_refreshed":
datetime.utcfromtimestamp(data["t"]).strftime('%Y-%m-%d %H:%M:%S')
        }

        return stock_data

async def update_stock_data():
    while True:
        try:
            global stock_data_cache
            stock_data_cache[STOCK_SYMBOL] = await
fetch_stock_data(STOCK_SYMBOL)
            print("Updated stock data cache:", stock_data_cache)
        except Exception as e:
            print(f"Error updating stock data: {e}")
            await asyncio.sleep(30)

@app.on_event("startup")
async def startup_event():
    # Khởi động nhiệm vụ cập nhật dữ liệu cổ phiếu
    asyncio.create_task(update_stock_data())

@app.get("/stock/{symbol}")
async def get_stock_info(symbol: str):
    try:
        if symbol not in stock_data_cache:
            raise HTTPException(status_code=404, detail="Stock symbol
not found in cache")

        stock_data = stock_data_cache[symbol]
        # Convert last refreshed time from UTC to VN timezone
        last_refreshed_utc =
datetime.strptime(stock_data["last_refreshed"], "%Y-%m-%d %H:%M:%S")
        last_refreshed_vn =
last_refreshed_utc.replace(tzinfo=UTC_timezone).astimezone(VN_timezone)
        stock_data["last_refreshed"] = last_refreshed_vn.strftime("%Y-
%m-%d %H:%M:%S %Z%Z")

        return stock_data
    except HTTPException as e:

```

```

        raise e
    except Exception as e:
        raise HTTPException(status_code=500, detail=str(e))

if __name__ == "__main__":
    import uvicorn
    uvicorn.run(app, host="127.0.0.1", port=8001)

```

Khởi chạy FastAPI

```

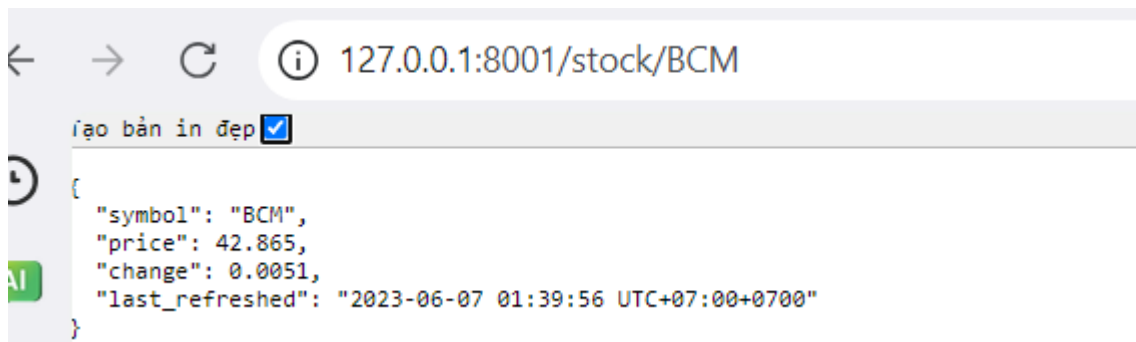
D:\KHOAHOC\DULIEU\Test\.venv\Scripts\python.exe D:\KHOAHOC\DULIEU\Test\main.py
D:\KHOAHOC\DULIEU\Test\main.py:49: DeprecationWarning:
  on_event is deprecated, use lifespan event handlers instead.

  Read more about it in the
  [FastAPI docs for Lifespan Events](https://fastapi.tiangolo.com/advanced/events/).

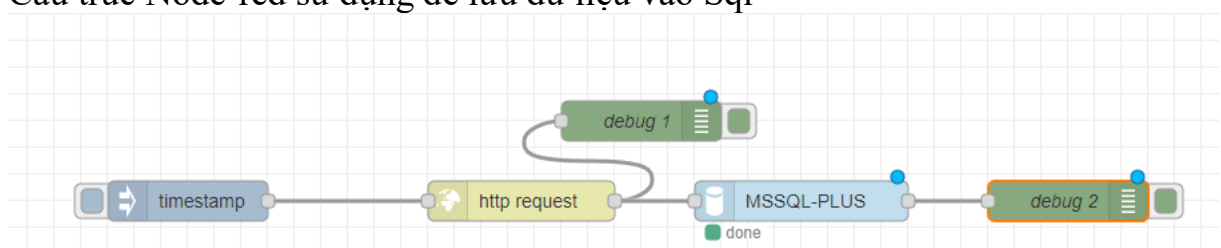
@app.on_event("startup")
INFO:      Started server process [52212]
INFO:      Waiting for application startup.
INFO:      Application startup complete.
INFO:      Uvicorn running on http://127.0.0.1:8001 (Press CTRL+C to quit)

```

Sau khi khởi chạy sẽ trả về kết quả một chuỗi json trên local của mình



Bước 2. Sử dụng node-red để lấy dữ liệu từ địa chỉ local của FastAPI
Cấu trúc Node-red sử dụng để lưu dữ liệu vào Sql



Trong đó :

- Timestamp : Bắt đầu quy trình bằng cách truyền dữ liệu vào

Edit inject node

Delete Cancel Done

Properties

Name

msg. payload = timestamp

msg. topic = a_z

+ add inject now

☐ Inject once after 0.1 seconds, then

Repeat interval

every 30 seconds

-> Sau mỗi khoảng thời gian này sẽ tự động lưu dữ liệu vào Database

- Http request : Nút này được sử dụng để gửi yêu cầu HTTP đến API và nhận phản hồi từ đó. Nút này thường được sử dụng để lấy dữ liệu truyền đến function đẩy dữ liệu lên SQL

Edit http request node

Delete Cancel Done

Properties

Method GET

URL http://127.0.0.1:8001/stock/BCM

Payload Ignore

☐ Enable secure (SSL/TLS) connection

☐ Use authentication

☐ Enable connection keep-alive

☐ Use proxy

☐ Only send non-2xx responses to Catch node

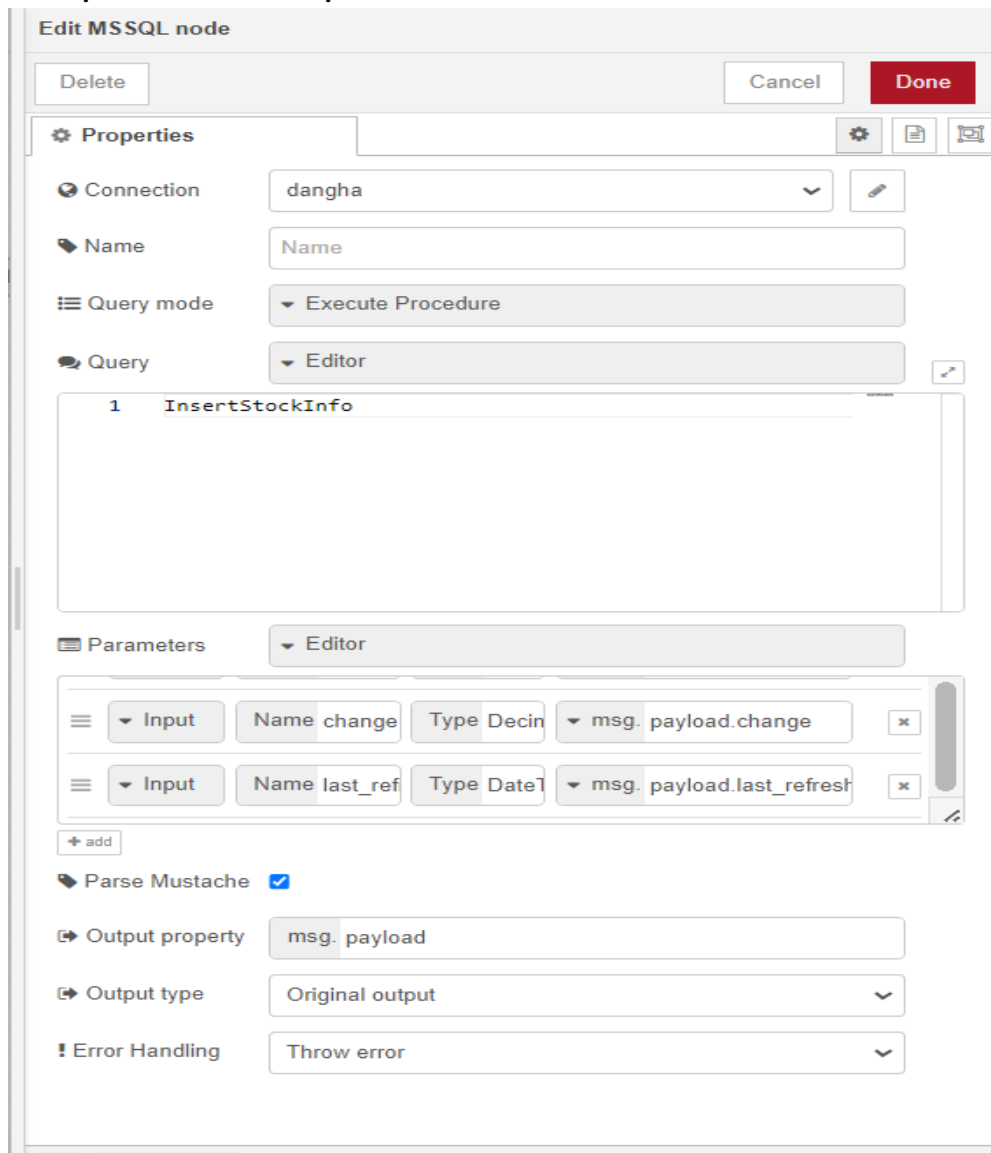
☐ Disable strict HTTP parsing

Return a parsed JSON object

Tip: If the JSON parse fails the fetched string is returned as-is.

Headers

- **MSSQL-PLUS:** Nút này được sử dụng để kết nối và truy vấn dữ liệu từ cơ sở dữ liệu Microsoft SQL Server. Để thực hiện các lệnh SQL và nhận dữ liệu từ cơ sở dữ liệu.



Edit MSSQL node

Delete Cancel Done

Properties

Connection: dangha

Name: Name

Query mode: Execute Procedure

Query: Editor

1 InsertStockInfo

Parameters Editor

Input	Name	Type	Value
▼ Input	change	Decin	msg. payload.change
▼ Input	last_ref	Date	msg. payload.last_refresh

+ add

Parse Mustache ☒

Output property: msg. payload

Output type: Original output

Error Handling: Throw error

- debug : Đây là một nút Debug khác trong Node-RED. Nút này cũng được sử dụng để xuất dữ liệu của thông điệp vào bảng điều khiển Debug để bạn có thể kiểm tra dữ liệu sau khi đã được xử lý bởi nút MSSQL-PLUS.

Bước 3. Tạo bảng và cơ sở dữ liệu

Tạo bảng StockInfo để lưu trữ dữ liệu về cổ phiếu

```
SQLQuery3.sql - DE...S.CoPhieu (sa (67)) X DESKTOP-T80TUOI\S...u - dbo.StockInfo SQLQuery2.sql - t
/***** Script for SelectTopNRows command from SSMS *****/
SELECT TOP (1000) [symbol]
, [price]
, [change]
, [last_refreshed]
FROM [CoPhieu].[dbo].[StockInfo]
```

Dữ liệu được lưu vào dbo. StockInfo

SQLQuery6.sql - DE...P-T80TUOI\PC (57))			SQLQuery4.sql - DE...S.CoPhieu
symbol	price	change	last_refresh...
BCM	42.865	0.000	2024-05-22 ...
BCM	42.865	0.005	2023-06-07 ...
AAPL	123.450	1.230	2024-05-24 ...
GOOGL	987.650	2.340	2024-05-24 ...
MSFT	234.560	-0.450	2024-05-24 ...
FB	345.670	0.670	2024-05-24 ...
AMZN	876.540	-1.230	2024-05-24 ...
TSLA	678.900	3.450	2024-05-24 ...
NFLX	456.780	1.560	2024-05-24 ...
NVDA	567.890	-0.890	2024-05-24 ...
INTC	789.010	2.670	2024-05-24 ...
AAPL	123.450	1.230	2024-05-24 ...
GOOGL	987.650	2.340	2024-05-24 ...
MSFT	234.560	-0.450	2024-05-24 ...
FB	345.670	0.670	2024-05-24 ...
AMZN	876.540	-1.230	2024-05-24 ...
TSLA	678.900	3.450	2024-05-24 ...
NFLX	456.780	1.560	2024-05-24 ...
NVDA	567.890	-0.890	2024-05-24 ...
INTC	789.010	2.670	2024-05-24 ...

Sau đó em viết 1 store Procedure

```
USE [CoPhieu]
GO

SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
ALTER PROCEDURE [dbo].[InsertStockInfo]
    @symbol VARCHAR(10),
    @price DECIMAL(10, 3),
    @change DECIMAL(10, 3),
    @last_refreshed DATETIME
AS
BEGIN
    -- Chèn dữ liệu mới nếu không tồn tại bản ghi tương tự
    INSERT INTO StockInfo (symbol, price, change, last_refreshed)
    VALUES (@symbol, @price, @change, @last_refreshed);
END;
```

Bước 3: Em tạo 1 thư mục trong py có tên là templates trong đó sẽ chứa ba file table.html, index.html, chart.html để đưa dữ liệu lên web và vẽ biểu đồ

Index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Home Page</title>
</head>
<body>
    <h1>Welcome to the Co Phieu </h1>
    <a href="/data">View Data</a><br>
    <a href="/inde">View Table</a><br>
    <a href="/chart">View Chart</a>
</body>
</html>
```

Table.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Data Table</title>
</head>
<body>
    <table border="1">
        <thead>
            <tr>
                {% for column in columns %}
                <th>{{ column }}</th>
```

```

        {% endfor %}
    </tr>
</thead>
<tbody>
    {% for row in data %}
        <tr>
            {% for column in columns %}
                <td>{{ row[column] }}</td>
            {% endfor %}
        </tr>
    {% endfor %}
</tbody>
</table>
</body>
</html>

```

Chart.html

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Stock Price Bar Chart</title>
    <script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
</head>
<body>
    <canvas id="stockPriceChart" width="800" height="400"></canvas>
    <script>
        document.addEventListener('DOMContentLoaded', function() {
            var ctx =
document.getElementById('stockPriceChart').getContext('2d');
            fetch('/data')
                .then(response => response.json())
                .then(data => {
                    var labels = data.map(row => row.timestamp); // Sử
dụng thuộc tính timestamp từ dữ liệu
                    var prices = data.map(row => row.price); // Sử dụng
thuộc tính price từ dữ liệu

                    var stockPriceChart = new Chart(ctx, {
                        type: 'bar', // Thay đổi loại biểu đồ thành 'bar'
                        data: {
                            labels: labels,
                            datasets: [{
                                label: 'Stock Price',
                                backgroundColor: 'rgba(54, 162, 235,
0.5)', // Màu của các cột
                                borderColor: 'rgba(54, 162, 235, 1)', //
Đường viền của các cột
                                borderWidth: 1,
                                data: prices
                            }]
                        },
                        options: {
                            scales: {
                                xAxes: [{
                                    scaleLabel: {
                                        display: true,
                                        labelString: 'Time'

```



```

        }
    },
    yAxes: [{
        scaleLabel: {
            display: true,
            labelString: 'Price'
        },
        ticks: {
            beginAtZero: true
        }
    }]
},
legend: {
    display: false
},
tooltips: {
    enabled: true
},
animation: {
    duration: 1000
}
}
});
});
});
</script>
</body>
</html>

```

Bước 4. Em tạo một tệp .env trong thư mục dự án để kết nối tới cơ sở dữ liệu SQL Server bằng chuỗi kết nối :

```

DB_CONNECTION_STRING=DRIVER={ODBC Driver 17 for SQL
Server};SERVER=DESKTOP-T80TUOI\SQLEXPRESS;DATABASE=CoPhieu;UID=sa;PWD=123

```

Bước 5. Em tạo 1 file app thiết lập một ứng dụng web Flask kết nối với SQL Server, truy vấn dữ liệu từ bảng StockInfo, và cung cấp các route để hiển thị dữ liệu dưới dạng JSON, bảng HTML, và biểu đồ.

```

from flask import Flask, render_template, jsonify
import pyodbc
import os
import logging
from dotenv import load_dotenv

app = Flask(__name__)

# Load biến môi trường từ tập tin .env
load_dotenv()

# Thiết lập logging
logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)

# Chuỗi kết nối tới SQL Server
connection_string = os.getenv('DB_CONNECTION_STRING')

```

```

if not connection_string:
    logger.error("Database connection string is not set. Please set the
DB_CONNECTION_STRING environment variable.")
else:
    logger.info(f"Database connection string: {connection_string}")

# Kết nối tới SQL Server
def get_db_connection():
    if not connection_string:
        raise ValueError("Database connection string is not set.")
    return pyodbc.connect(connection_string)

# Route để hiển thị dữ liệu từ SQL trên web
@app.route('/')
def index():
    return render_template('index.html')

@app.route('/data')
def get_data():
    try:
        conn = get_db_connection()
        cursor = conn.cursor()
        cursor.execute("SELECT * FROM dbo.StockInfo") # Đảm bảo schema
"dbo" hoặc chỉ định schema chính xác
        columns = [column[0] for column in cursor.description]
        data = [dict(zip(columns, row)) for row in cursor.fetchall()]
        conn.close()
        return jsonify(data)
    except Exception as e:
        return jsonify({"error": str(e)})

@app.route('/inde')
def show_table():
    try:
        with get_db_connection() as conn:
            cursor = conn.cursor()
            cursor.execute("SELECT * FROM StockInfo")
            columns = [column[0] for column in cursor.description]
            data = [dict(zip(columns, row)) for row in cursor.fetchall()]
            return render_template('table.html', columns=columns,
data=data)
    except Exception as e:
        logger.error(f"Error occurred while fetching data: {str(e)}")
        return jsonify({"error": str(e)})

@app.route('/chart')
def show_chart():
    return render_template('chart.html')

if __name__ == '__main__':
    app.run(debug=True)

```

Bước 6: Giao diện Web

Giao diện web sau khi chạy chương trình



Giao diện web sau khi click vào View Data

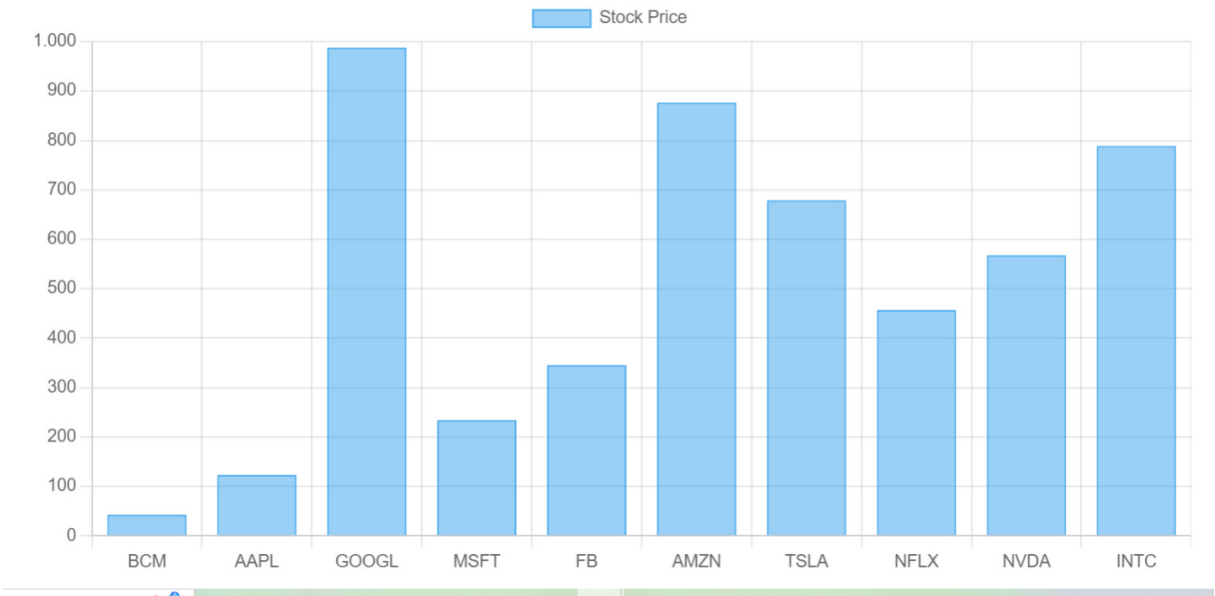


View table

The screenshot shows a web browser window with the address bar displaying '127.0.0.1:5000/index'. The main content of the page is a table with 4 columns: 'symbol', 'price', 'change', and 'last_refreshed'. The table contains 20 rows of data, including symbols like BCM, AAPL, GOOGL, MSFT, FB, AMZN, TSLA, NFLX, NVDA, and INTC.

symbol	price	change	last_refreshed
BCM	42.865	0.000	2024-05-22 07:00:00
BCM	42.865	0.005	2023-06-07 01:39:56
AAPL	123.450	1.230	2024-05-24 08:00:00
GOOGL	987.650	2.340	2024-05-24 08:05:00
MSFT	234.560	-0.450	2024-05-24 08:10:00
FB	345.670	0.670	2024-05-24 08:15:00
AMZN	876.540	-1.230	2024-05-24 08:20:00
TSLA	678.900	3.450	2024-05-24 08:25:00
NFLX	456.780	1.560	2024-05-24 08:30:00
NVDA	567.890	-0.890	2024-05-24 08:35:00
INTC	789.010	2.670	2024-05-24 08:40:00
AAPL	123.450	1.230	2024-05-24 08:00:00
GOOGL	987.650	2.340	2024-05-24 08:05:00
MSFT	234.560	-0.450	2024-05-24 08:10:00
FB	345.670	0.670	2024-05-24 08:15:00
AMZN	876.540	-1.230	2024-05-24 08:20:00
TSLA	678.900	3.450	2024-05-24 08:25:00
NFLX	456.780	1.560	2024-05-24 08:30:00
NVDA	567.890	-0.890	2024-05-24 08:35:00
INTC	789.010	2.670	2024-05-24 08:40:00

Theo dõi giá cổ phiếu



CHƯƠNG 4: KẾT LUẬN

Trong bài tập lớn này, em đã xây dựng một hệ thống thu thập, xử lý và hiển thị dữ liệu về cổ phiếu từ một nguồn dữ liệu công cộng. Sự kết hợp giữa FastAPI, Node-RED và cơ sở dữ liệu SQL đã tạo ra một giải pháp mạnh mẽ và linh hoạt, giúp chúng ta hiểu rõ hơn về quá trình xây dựng và triển khai các ứng dụng phức tạp.

Với giao diện người dùng, chúng ta có thể hiển thị dữ liệu một cách trực quan và thân thiện, giúp người dùng dễ dàng nắm bắt thông tin quan trọng về thị trường cổ phiếu. Hệ thống này cũng có tiềm năng để mở rộng và phát triển trong tương lai, với khả năng tích hợp thêm các tính năng mới và xử lý nhiều loại dữ liệu khác nhau.

Qua bài tập lớn lần em này giúp em hiểu hơn về ngôn ngữ lập trình Python cùng một số công cụ khác như Node-RED, SQL,...

Bên cạnh đó thì kiến thức cũng như kinh nghiệm của em còn hạn chế về nhiều mặt nên đề tài vẫn chưa được hoàn thiện tốt, và có nhiều sai sót. Em sẽ cố gắng học hỏi và tìm hiểu để hoàn thiện hơn đề tài này.

Em xin cảm ơn thầy Đỗ Duy Cốp đã nhiệt tình giúp đỡ để em hoàn thành bài tập lớn này.