Aptech COMPUTER EDUCATION
aptechvietnam.com.vn

THE 2nd SEMESTER FINAL PROJECT

"Developing Automatic System for Data Warehouse & Analysis"

Developed by Kieu Thi Huyen Trang

Members:

No.	Student Name	Student ID
1	Kieu Thi Huyen Trang	Student1465625

- ✔ **Class No.:** DT2304L
- ✔ **Start Date:** Aug 5th, 2024.
- ✔ **End Date:** Aug 27th, 2024.
- ✔ **Name of the Coordinator:** MR. Huynh Nam

Date of Submission: Aug 27th, 2024

Table of Contents

I. DATA INTRODUCTION

1. About the website

There are many websites that provide public data of stocks, from inside or outside Vietnam such as vietstock.vn, yahoo.com, investing.com, cafef.vn … Among these, I have had good experience working with vietstock.vn the most, so it seems to be the most trust-worthy source for me to process. Moreover, Yahoo or Investing can have at least 10-year-time for the data of foreign stocks but that for Vietnamese stocks are more limited than Vietstock's.

However, Vietstock just publicizes very few information of stock trading transactions. For examples, they give one-month data for non-account users and one-year for free-account ones. Due to the requirement of the project that we need to analyze the relation of prices, volumes and news over a period a year for each of 10 chosen stocks, I decided to agree with a one-year span for our results.

The primary requirement of the project is that I need to get information of 10 listed stocks from the vietstock.vn website. Therefore, it is necessary to build a system that is able to crawl data and push it into databases to clean it before continue to write it into a data warehouse connecting to power BI for visualization.

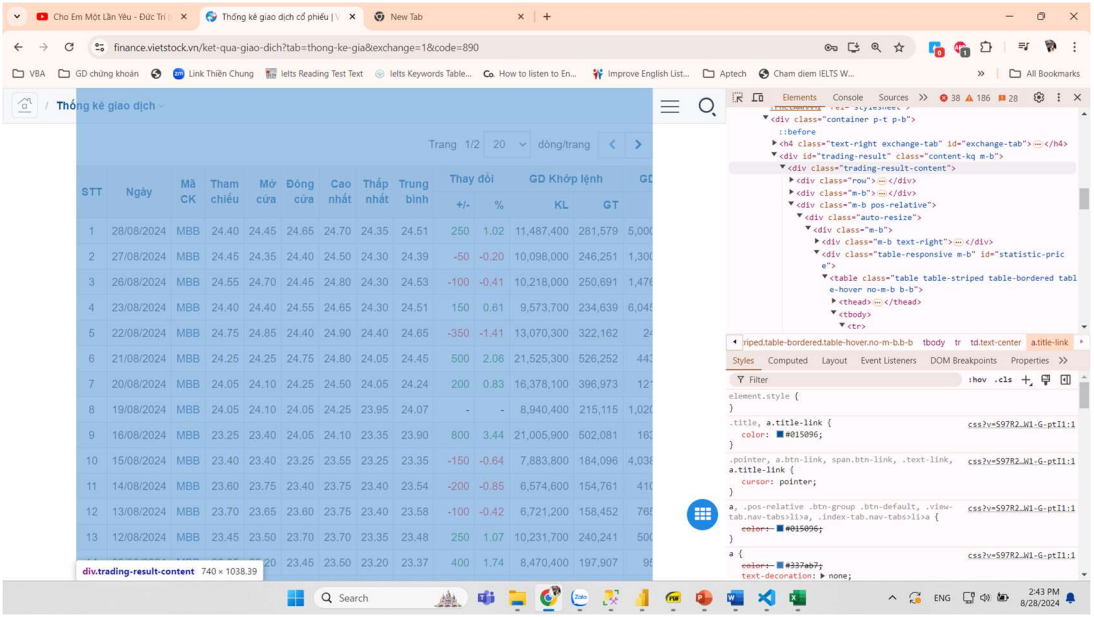
2. Data structure

2.1. Columns and rows of numerous details

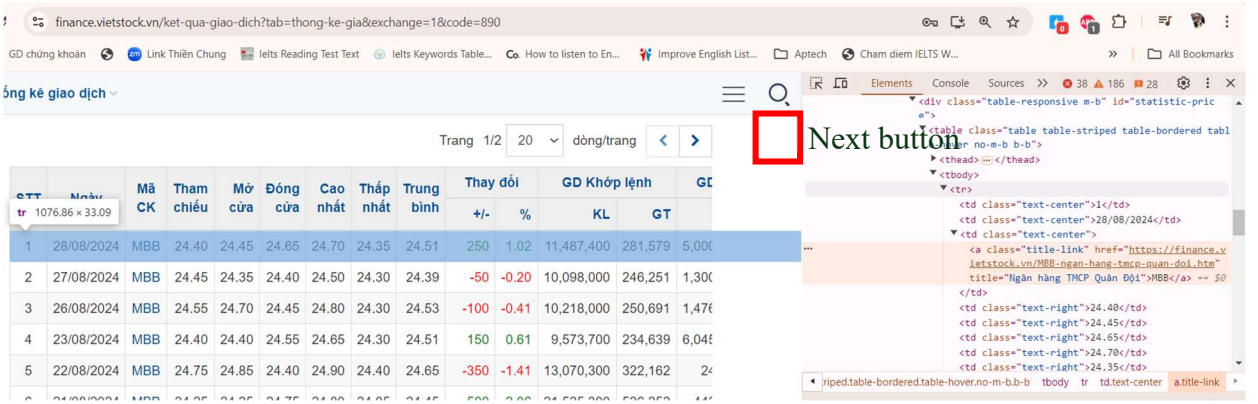
2.1.1. URL link structure:

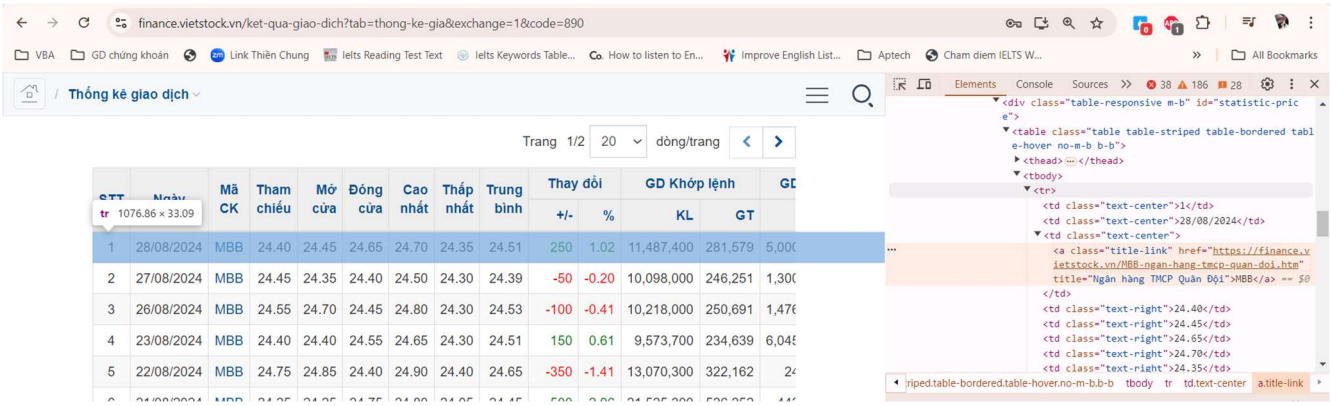
Vietstock has the same structure in the url link for each stock which starts with https://finance.vietstock.vn/ket-qua-giao-dich?tab=thong-ke-gia string and appended by "&exchange=" + exchangelist1[i] + "&code=" + linkcode1[i]" string. Here, "exchangelist1[i]" is the number code, for example: 1 for "HOSE" and "2" for "HNX" and so on while linkcode1[i] is the number code of each stock. For example, the one for MBB stock is 890, that of TCB is 13240.

2.1.2. Page source:

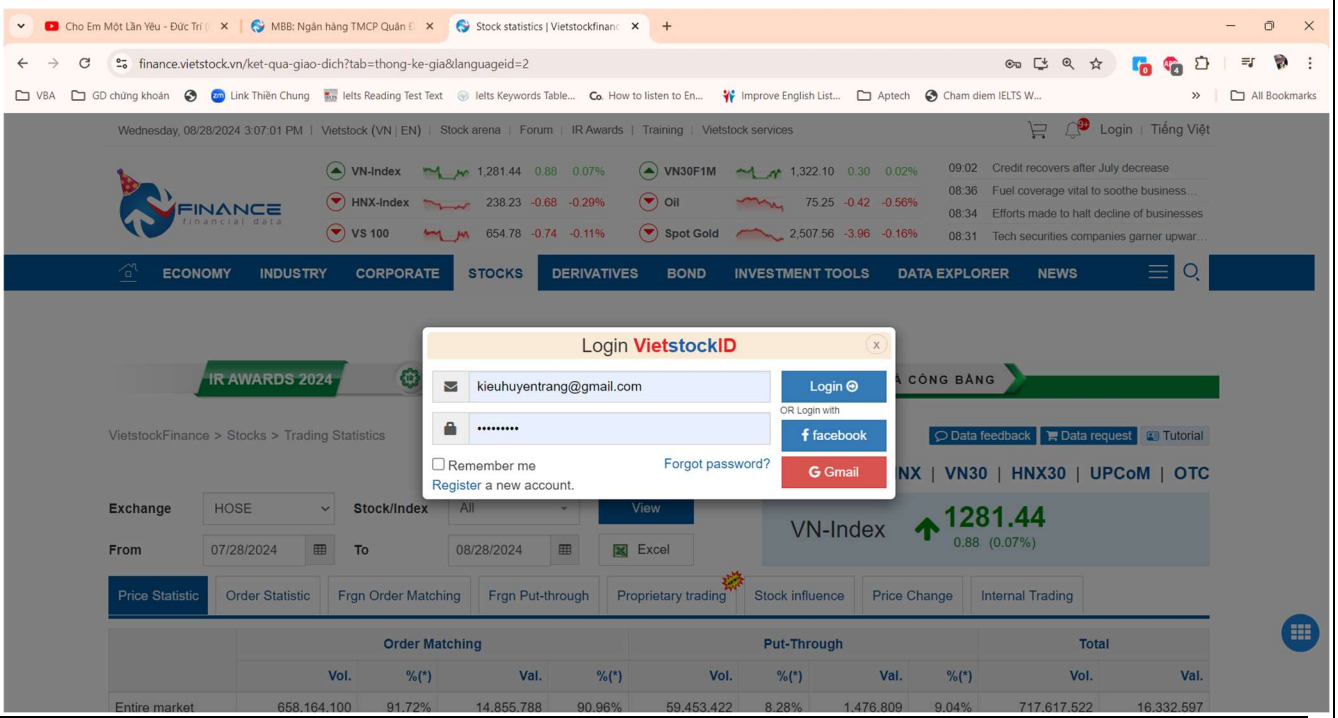


The whole table is included in “trading-result” class divided into 20 separate pages. This table is dynamically loaded content that there are 20 rows and 18 columns in each page. We need to click on the Next button for loading details of the next page.

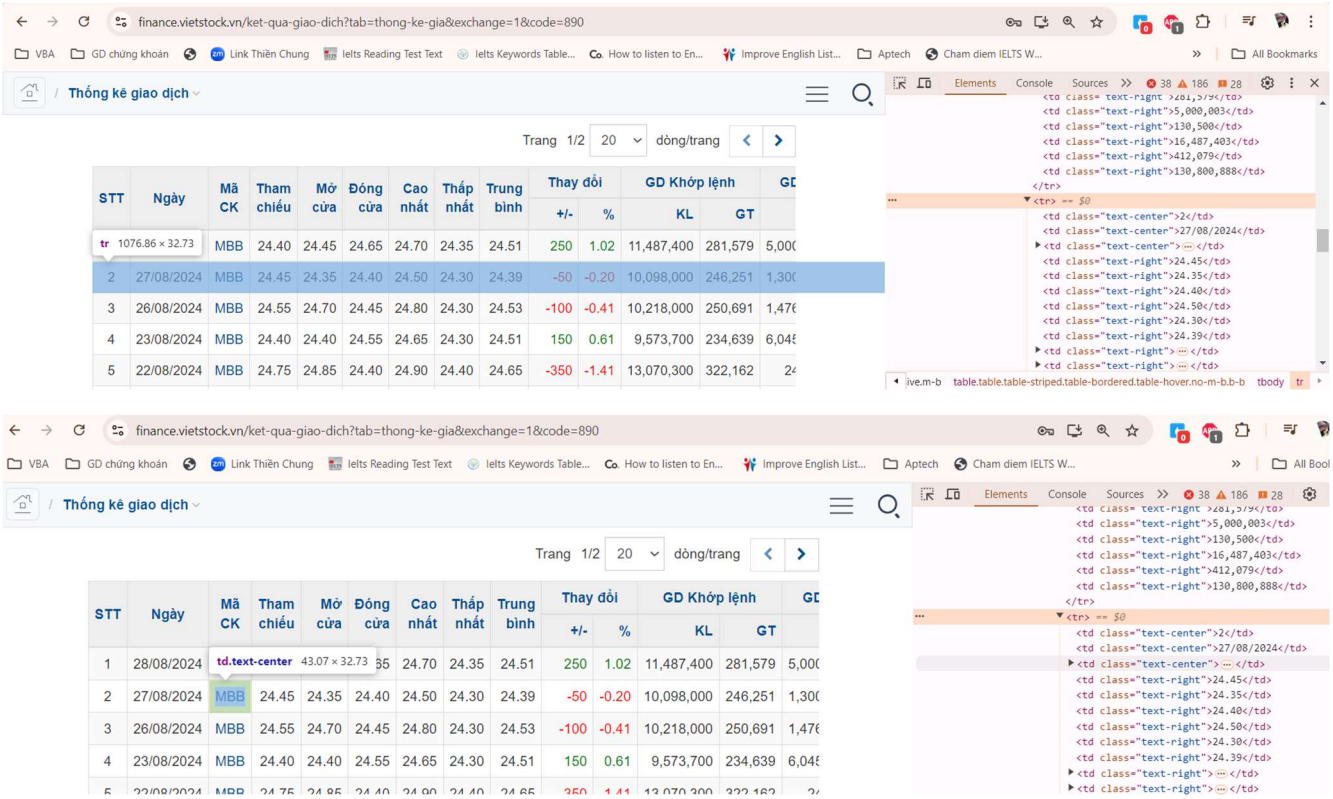




Noticeably, unregistered user can see only the first page of the table, which means that the result is shown in one-month period only. In contrast, registered user can have the latest one-year information. Therefore, I need to log in to the website before scrawling data.



Specifically, each row is wrapped in a “tr” class under “trading-result”, and each cell of the row contained in a “td” class as shown in the below pics:



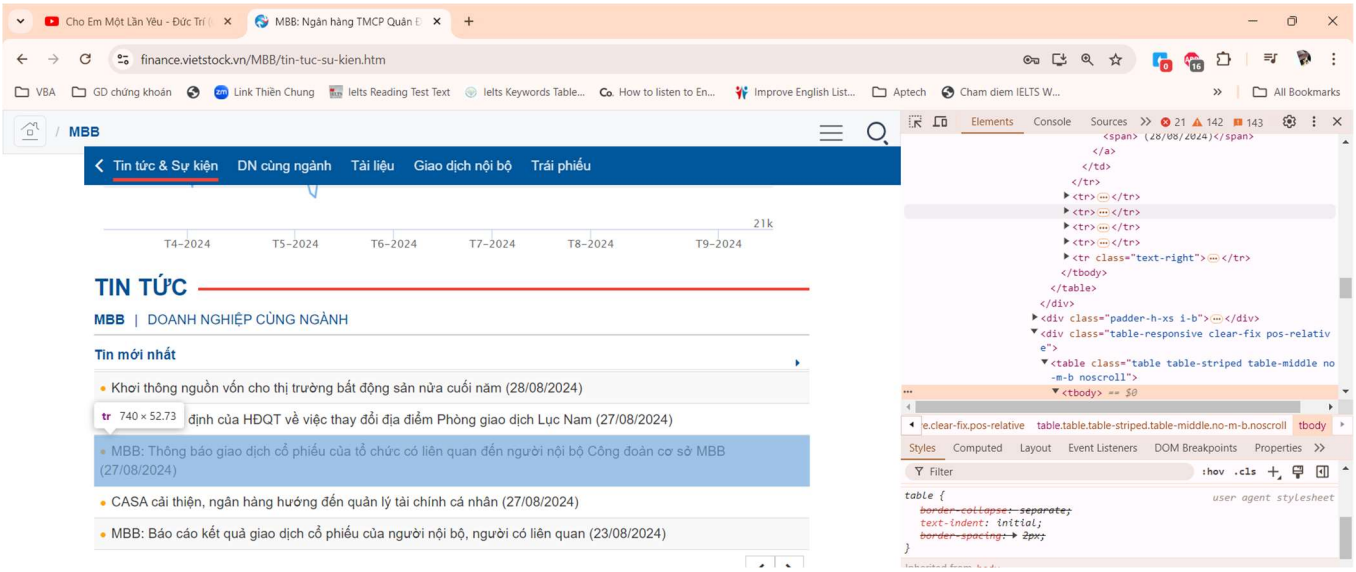
2.2. Columns and rows of news related to each stock:

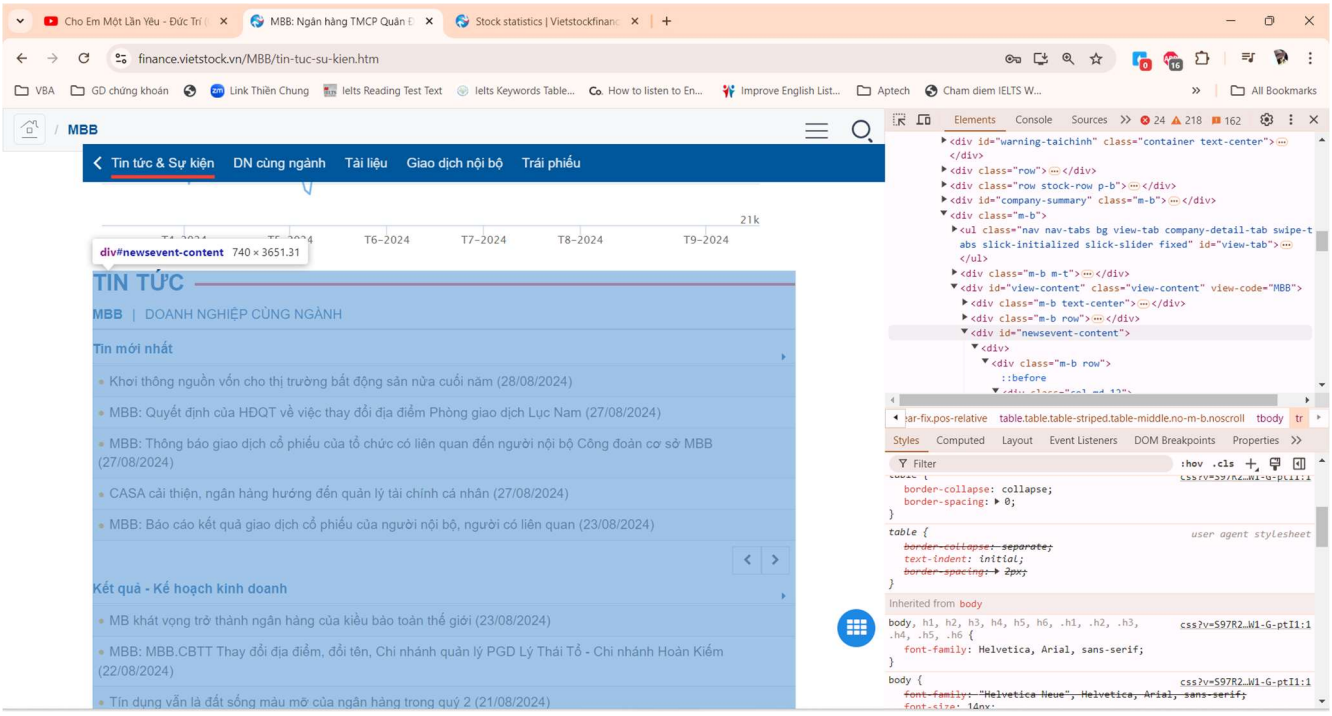
2.1.1. URL link structure:

Each stock has its own URL link where all latest news is contained. The same structure of the URL for each stock is “https://finance.vietstock.vn/"+stocklist1[i]+"/tin-tuc-su-kien.htm”. Here, stocklist1[i] is the stock code such as MBB, TCB and so on. Thus, what I need to do is just replacing stocklist1[i] with each stock code I need to analyze.

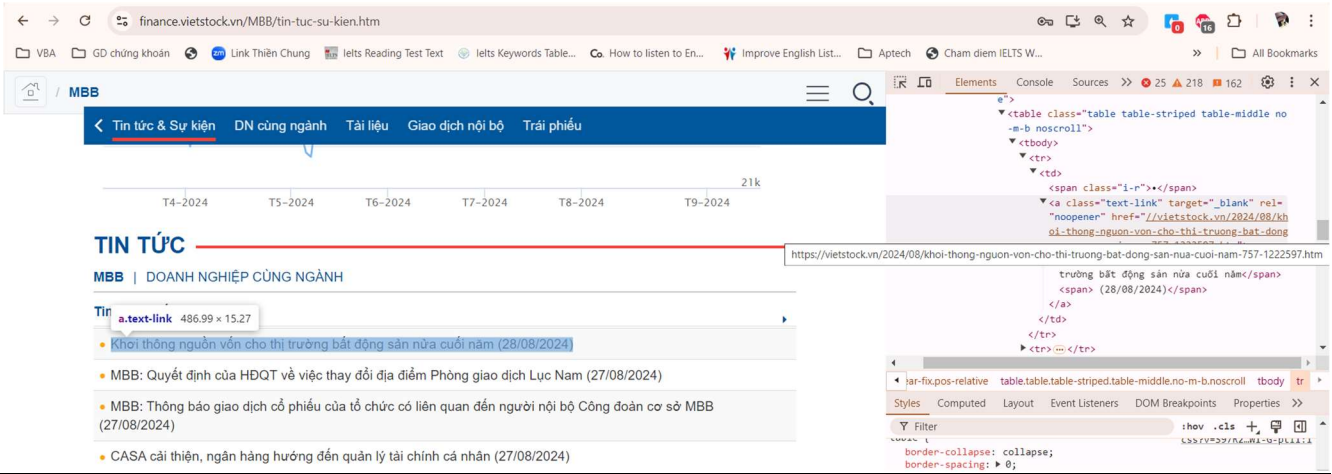
2.1.2. Page source:

Differing to numerous data, news URLs are all contained in “a” class/tag under the element with the “newsevent-content” ID:

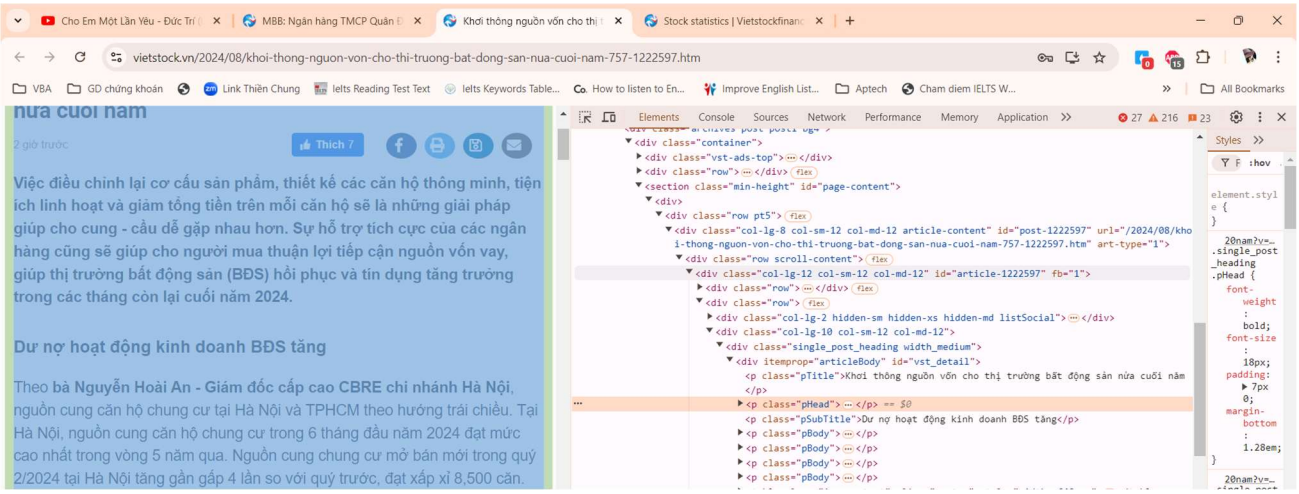




In detail, each URL link is wrapped in “href” attribute of the element with “a” tag:



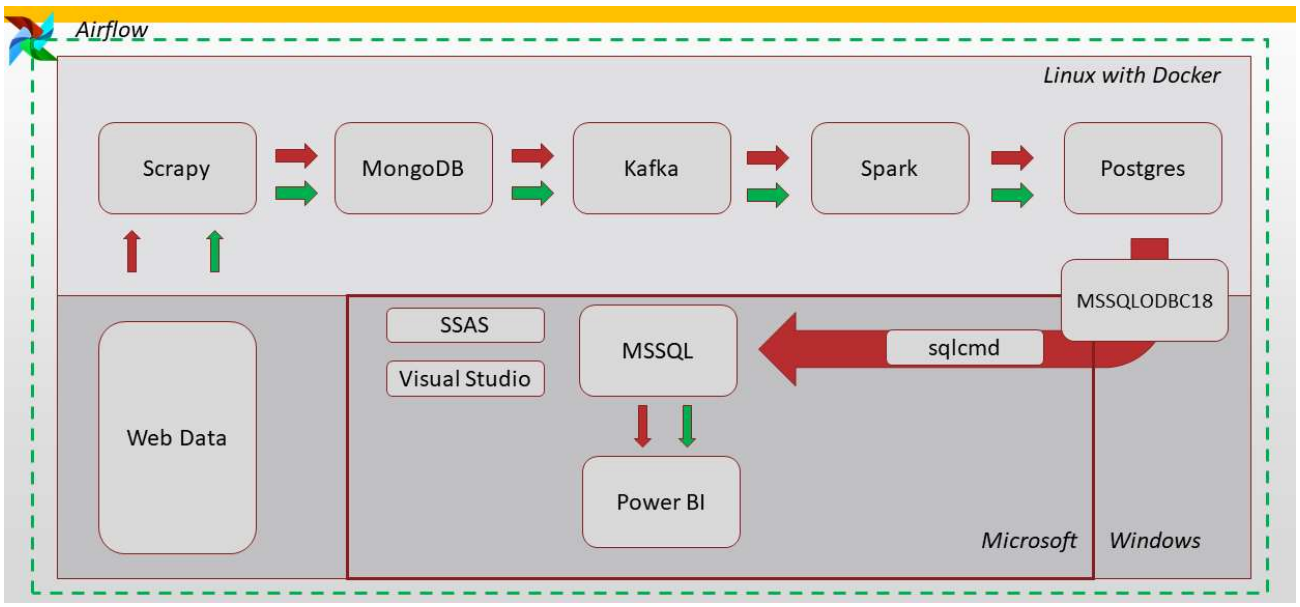
Finally, what I need to extract is the news's body text inside all elements of the "div.col-lg-10.col-sm-12.col-md-12" class for each URL link:



II. THE AUTOMATIC SYSTEM OF DATA PROCESSING

1. System overview

This diagram shows the organization and process of the system, starting with crawling data from the website using Scrapy and ending with visualizing my analysis using power BI dashboards:



The process is divided into three main parts:

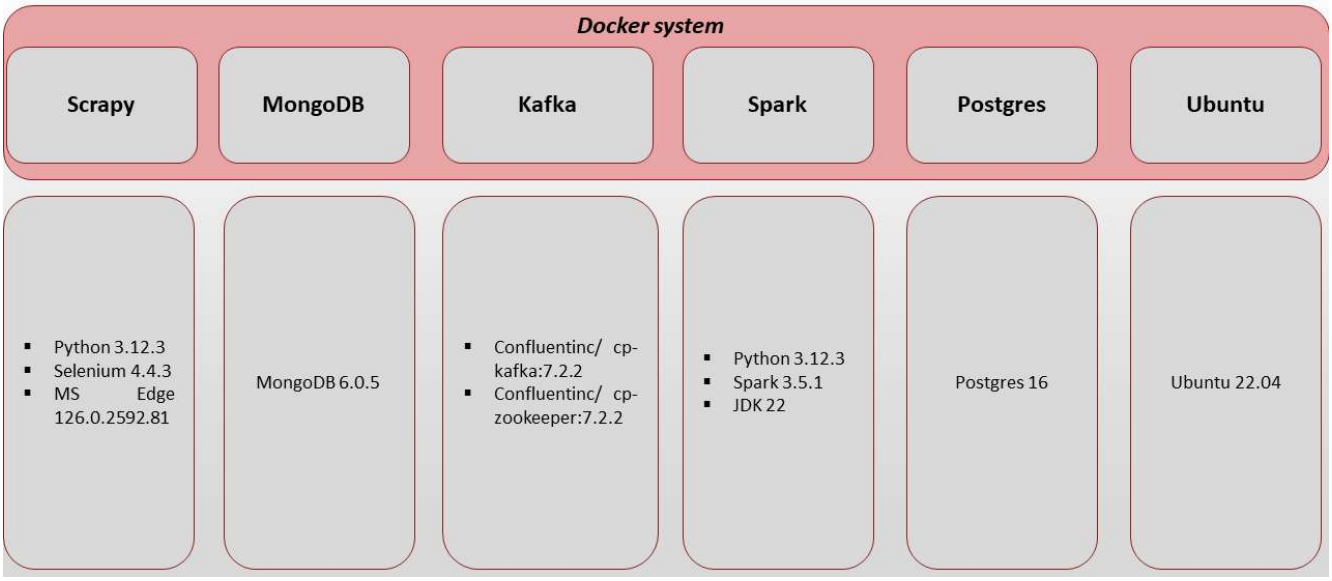
- The first one is to scraping raw data from the Vietstock website before transforming, cleaning and storing it into a Postgres database while the second phrase involves in building a data warehouse and a set of dashboards.
- When the first stage requests for numbers of Docker containers installed in Linux operating system, the second one relates to Microsoft ecosystem. Next, I will explain the components in details.

- The master controller of all the tasks is Airflow. It runs the process automatically, starting from scraping data and ending with inserting data into data warehouse.

2. Component details

2.1. Docker system

Docker is a software platform that allows us to build, test, and deploy applications quickly. Docker packages software into standardized units called containers that have everything the software needs to run, including libraries, system tools, code, and runtime. By using Docker, we can quickly deploy and scale applications into any environment and be confident that our code will run. In my case, there are six biggest set of containers participated in the back-end process to get raw data and process it as shown in the picture:



- **Scrapy:** Built from Python 3.12.3 base image, it is installed some more necessary packages such as Selenium 4.4.3, Microsoft Edge 126.0.2592.81 browser and driver for scraping data directly from the Vietstock website.
- **MongoDB:** Built from MongoDB 6.0.16 base image in replication mode according to the introduction from MongoDB official website. MongoDB does not support Kafka streaming connector to stand-alone mode, therefore I need to set up a replication mode based on their tutorials. This container helps storing raw data, both structured and unstructured ones at high speed. Besides, it remains and creates most essential information of data such as timestamp, which is important for real-time aggregation
- **Kafka:** Built from a set of five various services that are zookeeper, broker, connect, rest-proxy and schema-registry provided by MongoDB. While zookeeper is built from "confluentinc/cp-zookeeper:7.2.2" image, broker is built from "confluentinc/cp-kafka:7.2.2" image, connect is set up from "confluentinc/cp-kafka-connect:7.2.5" image, rest-proxy is based on "confluentinc/cp-kafka-rest:7.2.2" image, and schema-resigtry is based on "confluentinc/cp-schema-registry:7.2.2" image. Kafka helps boosting the speed of processing data, especially for real-time streaming data. I choose to connect Kafka between MongoDB and PySpark for two reasons:
    - Kafka can keep raw data from MongoDB in topics so that avoid the data loss when streaming and flowing inside the pipeline.

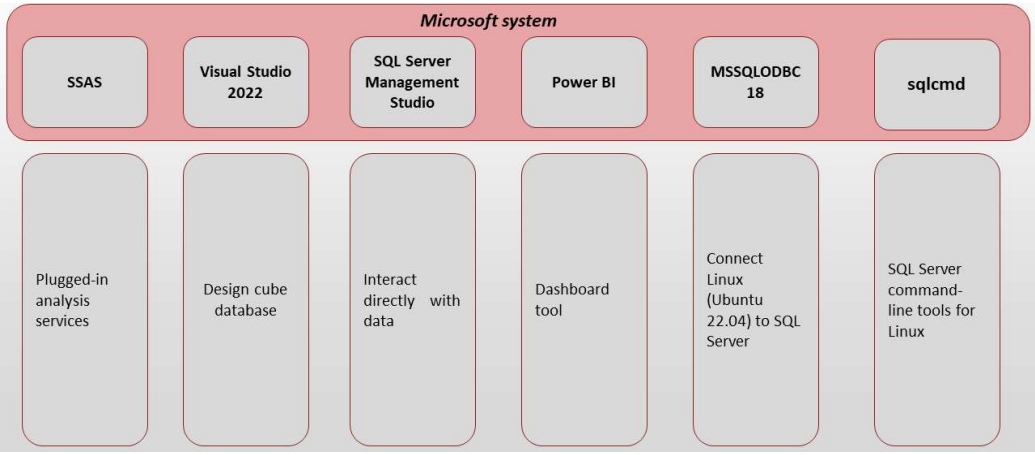
- Kafka can boost up the speed of processing streamed data nearly to immediately due to the distribution mechanism of partitions that divide and distribute data into smaller batches to process in parallel.

- **Spark:** Built from Python 3.12.3 base image. The supplementing packages to be installed are Spark 3.5.1, JDK 22, postgresql-42.7.3.jar (Postgres driver to help connect Spark and Postgres). Most of the data processing tasks including cleaning, transforming, dealing with null data, writing cleaned data into different external storages in various operating system such as Linux and Windows are handled by Spark container. Therefore, it is vital to install correct packages as well as acceptable drivers for connecting those applications.
- **Postgres:** Built from Postgres 16 image.
- **Ubuntu:** Built from Ubuntu 22.04 image, this is the environment for installing and running MS SQL ODBC 18 and SQL CMD (mentioned in the next content).

Above is a distinguishment of a set of containers based on the difference of their features and roles in the whole system. In fact, we can totally reduce the number of containers by building a bigger container based on Ubuntu 22.04 and install all needed packages inside it such as Python, Spark, JDK …, and other drivers.

2.2. Microsoft ecosystem:

When Linux in association with Docker is a back-end system with its strength in machine resource utility and its flexibility to deal with the difference in various versions of applications, Microsoft with Windows operating system takes advantage of its user-friendly interface which is the good choice for dash board presentation. In other words, Power BI is quite a common software for visualizing which means that choosing it leads me to an ecosystem of Microsoft with supported applications for building data warehouse such as MS Visual Studio with SQL Server Analysis Services (SSAS, in short) plugin as shown in the diagram:



- **MS SQL ODBC 18:** This is a Linux driver developed by Microsoft and considered as the key component to connect Linux and Windows. Because I plan to work with Power BI, I need to apply Microsoft driver for the compatibility between two systems. However, there is one issue that Microsoft only supports some listed versions of Ubuntu (like 18.04, 20.04, 22.04) which exclude my Ubuntu 23.10 virtual machine's. My solution for this problem is to build my own Ubuntu container based on ubuntu image version 22.04.

- **SQL CMD:** This is the command-line tool inside MS SQL ODBC 18 package, which is used for executing SQL commands on Linux machine for remotely manipulating databases in SQL Server Management Studio in Windows machine.
- **SQL Server Management Studio:** This is a front-end application for interacting with database. From here, I can see the database created by SQL commands from Linux machine as well as its tables with complete data. Also, it helps connect with SQL Server Analysis Services in Microsoft Visual Studio 2022 for working with cube database. My practical version is the latest release, 20.2.
- **Microsoft Visual Studio:** The practical version of my machine is Community 2022 which is used for developers.
- **SQL Server Analysis Services (SSAS):** This is a plugin installed inside Microsoft Visual Studio for designing cube database. It helps making some calculated columns and measures that pre-calculate the results of formulas for better performance of Power BI queries.
- **Power BI:** This is a common tool of Microsoft for visualizing. It can connect to SQL Server in import or live connect modes.

2.3. Airflow system

Airflow 2.9.3 is installed directly to my Ubuntu 23.10 virtual machine using "pip" installation under a Python virtual environment to avoid conflicts with built-in Python packages of Ubuntu 23.10. It is not necessary to create one more Docker container for Airflow due to the machine resource utility. I need to take advantage of all the installed packages inside my virtual machine. Moreover, the purpose of Airflow is running all the tasks automatically no matter what they are inside or outside my Docker system.

III. PROGRAMMING

A – BUILDING DOCKER CONTAINERS

1. Scrapy

- The Dockerfile used to build scrapy container:

```
FROM python:latest

# Install necessary packages
RUN apt-get update && apt-get install -y
RUN apt-get update && apt-get install -y gnupg2
RUN apt-get update && apt-get install -y gnupg
RUN apt-get -y install apt-transport-https
RUN apt-get -y install software-properties-common
RUN apt-get -y install libglib2.0-0
RUN apt-get -y install libnss3
RUN apt-get -y install libgconf-2-4
RUN apt-get -y install libfontconfig1
RUN apt-get -y install libxss1
RUN apt-get -y install libappindicator1
RUN apt-get -y install unzip
RUN apt-get -y install nano
```

```
RUN apt-get -y install vim


# Bổ sung để fix lỗi MS Edge crashed:
RUN apt-get -y install libatk-bridge2.0-0
RUN apt-get -y install libatspi2.0-0
RUN apt-get -y install libdrm2
RUN apt-get -y install libgbm1
RUN apt-get -y install libgtk-3-0
RUN apt-get -y install libu2f-udev
RUN apt-get -y install libvulkan1
RUN apt-get -y install libxkbcommon0
RUN apt-get -y install xdg-utils

RUN apt-get update && apt-get install -y \
    wget \
    xvfb \
    fonts-liberation \
    libappindicator3-1 \
    libgbm-dev
RUN rm -rf /var/lib/apt/lists/*

# Update the package list with the new repository
RUN apt-get update

# Copy the Microsoft Edge Browser setup file into the Docker image
COPY microsoft-edge-stable_126.0.2592.81-1_amd64.deb /tmp/
# Install Microsoft Edge Browser from the .deb file
RUN dpkg -i /tmp/microsoft-edge-stable_126.0.2592.81-1_amd64.deb || apt-get install -f
-y \
    && rm /tmp/microsoft-edge-stable_126.0.2592.81-1_amd64.deb
# Copy MS Edge Vdriver zip file from local machine vào image:
COPY msedgedriver /usr/local/bin

# Set execute permissions for msedgedriver
RUN chmod +x /usr/local/bin/msedgedriver
RUN chmod 777 /usr/local/bin/msedgedriver
# Export the path of EdgeDriver to the PATH environment variable
ENV PATH="/usr/local/bin/msedgedriver:${PATH}"
# Export the path of chromedriver to the PATH environment variable
ENV PATH="/usr/local/bin/chromedriver-linux64:${PATH}"

# set display port to avoid crash
ENV DISPLAY=:99
# Copy requirements.txt into the Docker image
COPY requirements.txt /tmp/
# Install Python packages from requirements.txt
RUN pip install --upgrade pip \
```

```
    && pip install -r /tmp/requirements.txt
# Cleanup
RUN rm /tmp/requirements.txt

# Set the working directory
WORKDIR /

# Create a logs directory
RUN mkdir -p /log/
# Set the environment variable for logs directory
ENV LOG_DIR=/log/
# Copy the application files
COPY Ticker_table-xvfb.xlsx /
COPY dbstockcrawl.py /
COPY contentpubver3.py /
COPY start_service.sh /
COPY startpubver3.sh /
RUN chmod 777 /start_service.sh
RUN chmod +x /startpubver3.sh

# Set command
CMD ["bash"]
```

- "start_service.sh":

```
#!/bin/bash
# Set up the timezone for the Docker container
ln -snf /usr/share/zoneinfo/Asia/Ho_Chi_Minh /etc/localtime
echo "Asia/Ho_Chi_Minh" > /etc/timezone
# Start Xvfb and capture its PID
Xvfb :99 -screen 0 1920x1080x24 & python "dbstockcrawl.py"
```

- "startpubver3.sh":

```
#!/bin/bash
# Set up the timezone for the Docker container
ln -snf /usr/share/zoneinfo/Asia/Ho_Chi_Minh /etc/localtime
echo "Asia/Ho_Chi_Minh" > /etc/timezone
# Start Xvfb in the background and capture its PID
Xvfb :99 -screen 0 1920x1080x24 & python "contentpubver3.py"
```

2. MongoDB & Kafka

The docker compose file to build a set of MongoDB and Kafka service containers:

```
services:
  zookeeper: #1
    image: confluentinc/cp-zookeeper:7.2.2
    hostname: zookeeper
    container_name: zookeeper
```

```
    networks:
      - localnet
    environment:
      ZOOKEEPER_CLIENT_PORT: 2181
      ZOOKEEPER_TICK_TIME: 2000

  broker: #2
    image: confluentinc/cp-kafka:7.2.2
    hostname: broker
    container_name: broker
    depends_on:
      - zookeeper
    networks:
      - localnet
    environment:
      KAFKA_BROKER_ID: 1
      KAFKA_ZOOKEEPER_CONNECT: "zookeeper:2181"
      KAFKA_LISTENERS: LISTENER_1://broker:29092,LISTENER_2://broker:9092
      KAFKA_ADVERTISED_LISTENERS:
LISTENER_1://broker:29092,LISTENER_2://localhost:9092
      KAFKA_LISTENER_SECURITY_PROTOCOL_MAP: LISTENER_1:PLAINTEXT,LISTENER_2:PLAINTEXT
      KAFKA_INTER_BROKER_LISTENER_NAME: LISTENER_1
      KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: 1
      KAFKA_GROUP_INITIAL_REBALANCE_DELAY_MS: 0
      CONFLUENT_SUPPORT_CUSTOMER_ID: "anonymous"
      KAFKA_DELETE_TOPIC_ENABLE: "true"

  connect: #5
    build:
      context: .
      dockerfile: connect.Dockerfile
    ports:
      - "35000:35000"
    hostname: connect
    container_name: connect
    depends_on:
      - zookeeper
      - broker
    networks:
      - localnet
    environment:
      KAFKA_JMX_PORT: 35000
      KAFKA_JMX_HOSTNAME: localhost
      CONNECT_BOOTSTRAP_SERVERS: "broker:29092"
      CONNECT_REST_ADVERTISED_HOST_NAME: connect
      CONNECT_REST_PORT: 8083
      CONNECT_GROUP_ID: connect-cluster-group
      CONNECT_CONFIG_STORAGE_TOPIC: docker-connect-configs
```

```
    CONNECT_CONFIG_STORAGE_REPLICATION_FACTOR: 1
    CONNECT_OFFSET_FLUSH_INTERVAL_MS: 10000
    CONNECT_OFFSET_STORAGE_TOPIC: docker-connect-offsets
    CONNECT_OFFSET_STORAGE_REPLICATION_FACTOR: 1
    CONNECT_STATUS_STORAGE_TOPIC: docker-connect-status
    CONNECT_STATUS_STORAGE_REPLICATION_FACTOR: 1
    CONNECT_ZOOKEEPER_CONNECT: "zookeeper:2181"
    CONNECT_PLUGIN_PATH: "/usr/share/java,/usr/share/confluent-hub-components"
    CONNECT_CONNECTIONS_MAX_IDLE_MS: 180000
    CONNECT_METADATA_MAX_AGE_MS: 180000
    CONNECT_AUTO_CREATE_TOPICS_ENABLE: "true"
    CONNECT_KEY_CONVERTER: "org.apache.kafka.connect.json.JsonConverter"
    CONNECT_VALUE_CONVERTER: "org.apache.kafka.connect.json.JsonConverter"

rest-proxy: #4
  image: confluentinc/cp-kafka-rest:7.2.2
  depends_on:
    - zookeeper
    - broker
    - schema-registry
  hostname: rest-proxy
  container_name: rest-proxy
  networks:
    - localnet
  environment:
    KAFKA_REST_HOST_NAME: rest-proxy
    KAFKA_REST_BOOTSTRAP_SERVERS: "broker:29092"
    KAFKA_REST_LISTENERS: "http://0.0.0.0:8082"
    KAFKA_REST_SCHEMA_REGISTRY_URL: "http://schema_registry:8081"

schema-registry: #3
  image: confluentinc/cp-schema-registry:7.2.2
  hostname: schema-registry
  container_name: schema-registry
  depends_on:
    - broker
  networks:
    - localnet
  environment:
    SCHEMA_REGISTRY_HOST_NAME: schema-registry
    SCHEMA_REGISTRY_KAFKASTORE_BOOTSTRAP_SERVERS: "broker:29092"
    SCHEMA_REGISTRY_KAFKASTORE_CONNECTION_URL: "zookeeper:2181"
    SCHEMA_REGISTRY_LISTENERS: http://0.0.0.0:8081

mongo1:
  image: "mongodb-kafka-base-mongod:1.0"
  container_name: mongo1
  ports:
```

```
      - "35001:27017"
    build:
      context: .
      dockerfile: mongo.Dockerfile
    command: --replSet rs0 --oplogSize 128
    depends_on:
      - zookeeper
      - broker
      - connect
    networks:
      - localnet
      - smnet2 #
    restart: always
    volumes: #
      - mongo_data:/data/db
      - /home/osboxes/mongo1backup:/data/backup #

  mongo1-setup:
    image: "mongodb-kafka-base-setup-mongod:1.0"
    container_name: mongo1-setup
    build:
      context: .
      dockerfile: mongo.Dockerfile
    depends_on:
      - mongo1
    networks:
      - localnet
    entrypoint:
      [
        "bash",
        "-c",
        "sleep 10 && mongosh --host mongo1:27017 config-replica.js && sleep 10",
      ]
    restart: "no"

networks:
  smnet2:
  localnet:
    attachable: true
volumes:
  mongo_data:
```

3. Spark

- The Dockerfile to build spark container:

```
FROM python:3.12.3
# Upgrade pip
RUN pip install --no-cache-dir --upgrade pip
```

```
# Install OpenJDK (required for Spark) and other dependencies
RUN apt-get update && apt-get install -y
RUN apt-get -y update
RUN apt-get -y upgrade
RUN apt-get update
RUN apt-get install -y curl
RUN apt-get install -y python3-pip
RUN apt-get install -y nano
RUN apt-get install vim -y
RUN apt-get install -y iputils-ping
RUN apt-get install -y kafkacat
RUN apt-get install -y git
# Copy the local JDK tar.gz file into the Docker image
COPY jdk-22_linux-x64_bin.tar.gz /tmp/jdk-22_linux-x64_bin.tar.gz

# Install OpenJDK 22 from the local file
RUN apt-get update && apt-get install -y \
    wget && \
    tar -xzvf /tmp/jdk-22_linux-x64_bin.tar.gz -C /opt && \
    rm /tmp/jdk-22_linux-x64_bin.tar.gz

# Copy the Apache Spark tar.gz file from the local machine to the Docker image
COPY spark-3.5.1-bin-hadoop3.tgz /tmp/spark-3.5.1-bin-hadoop3.tgz
# Install Apache Spark 3.5.1
RUN tar -xzf /tmp/spark-3.5.1-bin-hadoop3.tgz -C /opt && \
    rm /tmp/spark-3.5.1-bin-hadoop3.tgz

# Set environment variables
ENV PYSPARK_PYTHON=python3
ENV PYSPARK_DRIVER_PYTHON=python3
ENV PYSPARK_VERSION=3.5.1

ENV SPARK_HOME=/opt/spark-3.5.1-bin-hadoop3
ENV JAVA_HOME=/opt/jdk-22
ENV PATH=$JAVA_HOME/bin:$SPARK_HOME/bin:$PATH

# Copy the PostgreSQL JDBC driver into the Docker image
COPY postgresql-42.7.3.jar ${SPARK_HOME}/jars/
# Set working directory
WORKDIR /app

# Copy Python script and requirements file
COPY kafka-clients-3.8.0.jar /app/
COPY pyspark-requirements.txt /app/pyspark-requirements.txt

# Install Python dependencies
RUN pip install -r pyspark-requirements.txt
CMD ["bash"]
```

After building the spark container, we can write more entrypoint scripts and copy them into spark container.

- "entrypoint.sh":

```
#!/bin/bash
# Set up the timezone for the Docker container
ln -snf /usr/share/zoneinfo/Asia/Ho_Chi_Minh /etc/localtime
echo "Asia/Ho_Chi_Minh" > /etc/timezone
# Start PySpark with the required packages
pyspark --packages org.apache.spark:spark-sql-kafka-0-10_2.12:3.5.1 <<EOF
exec(open("/app/dbstockread.py").read())
EOFO
```

- "entrypoint_content.sh":

```
#!/bin/bash
# Set up the timezone for the Docker container
ln -snf /usr/share/zoneinfo/Asia/Ho_Chi_Minh /etc/localtime
echo "Asia/Ho_Chi_Minh" > /etc/timezone
# Start PySpark with the required packages
pyspark --packages org.apache.spark:spark-sql-kafka-0-10_2.12:3.5.1 <<EOF
exec(open("/app/dbcontent-15.py").read())
EOFO
```

- "post2ssms.sh":

```
#!/bin/bash
# Set up the timezone for the Docker container
ln -snf /usr/share/zoneinfo/Asia/Ho_Chi_Minh /etc/localtime
echo "Asia/Ho_Chi_Minh" > /etc/timezone
# Start PySpark with the required packages
pyspark --packages org.apache.spark:spark-sql-kafka-0-10_2.12:3.5.1 --jars
/app/sqljdbc_12.8/enu/jars/mssql-jdbc-12.8.0.jre11.jar \
--conf "spark.driver.extraJavaOptions=-Duser.timezone=Asia/Ho_Chi_Minh" --conf
"spark.executor.extraJavaOptions=-Duser.timezone=Asia/Ho_Chi_Minh" <<EOF
exec(open("/app/ssms5tables2.py").read())
EOF
```

4. Postgres & Adminer

I build a docker-compose file to create postgres along with adminer services in order to interact with Postgres database using Adminer Web UI. This is the content of the "adminerpostgres.yml" file:

```
version: "3.8"
services:
    postgres:
        image: postgres
        container_name: postgres
        restart: always
        environment:
```

```
            - POSTGRES_USER=admin
            - POSTGRES_PASSWORD=admin
            - POSTGRES_HOST=ppnet #
            - POSTGRES_DB=db249
        volumes:
            - datavol:/var/lib/postgresql/data
        ports:
            - "5432:5432"
        networks:
            - ppnet
            - panet # ket noi adminer

    adminer:
        image: adminer # uses an image from a registry
        container_name: adminer
        restart: always
        environment:
            - PGUSER=admin
            - PGPASSWORD=admin
            - PGDATABASE=db249
            - PGHOST=postgres # ppnet
            - PGPORT=5432
        ports:
            - "8080:8080"
        networks:
            - panet

networks:
    panet:
    ppnet:

volumes:
    datavol:
```

5. Ubuntu 22.04

I build the container from an ubuntu 22.04 image, then I install MS SQL ODBC package inside it using bash shell commands.

- The Dockerfile:

```
# Use the latest Ubuntu 24.04 image as the base
FROM ubuntu:22.04
# Set environment variables to non-interactive to avoid prompts during build
ENV DEBIAN_FRONTEND=noninteractive
# Update and install required packages
RUN apt-get update && \
    apt-get upgrade -y && \
    apt-get install -y \
```

```
    python3 \
    python3-pip \
    curl \
    gnupg \
    apt-transport-https \
    unixodbc-dev \
    && rm -rf /var/lib/apt/lists/*

# Create and activate a Python virtual environment, then install packages
RUN python3 -m venv /opt/venv && \
    /opt/venv/bin/pip install --upgrade pip && \
    /opt/venv/bin/pip install pyspark pyodbc pandas kafka

# Copy the local JDK tar.gz file into the Docker image
COPY jdk-22_linux-x64_bin.tar.gz /tmp/jdk-22_linux-x64_bin.tar.gz

# Install OpenJDK 22 from the local file
RUN apt-get update && apt-get install -y \
    wget && \
    tar -xzvf /tmp/jdk-22_linux-x64_bin.tar.gz -C /opt && \
    rm /tmp/jdk-22_linux-x64_bin.tar.gz
# Set environment variables
ENV PYSPARK_PYTHON=python3
ENV PYSPARK_DRIVER_PYTHON=python3
ENV JAVA_HOME=/opt/jdk-22
ENV PATH=$JAVA_HOME/bin:$SPARK_HOME/bin:$PATH
# Set working directory
WORKDIR /
# Command to run when the container starts
CMD ["/bin/bash"]
```

- Bash shell commands to install MS SQL ODBC 18:

```
curl https://packages.microsoft.com/keys/microsoft.asc | tee /etc/apt/trusted.gpg.d/microsoft.asc

curl https://packages.microsoft.com/keys/microsoft.asc | tee /etc/apt/trusted.gpg.d/microsoft.asc

curl https://packages.microsoft.com/config/ubuntu/22.04/prod.list | tee
/etc/apt/sources.list.d/mssql-release.list

apt-get update

apt-get install mssql-tools18 unixodbc-dev

apt-get install -y unixodbc unixodbc-dev

apt --fix-broken install

ACCEPT_EULA=Y apt-get install -y msodbcsql18
```

- "entrypoint.sh":

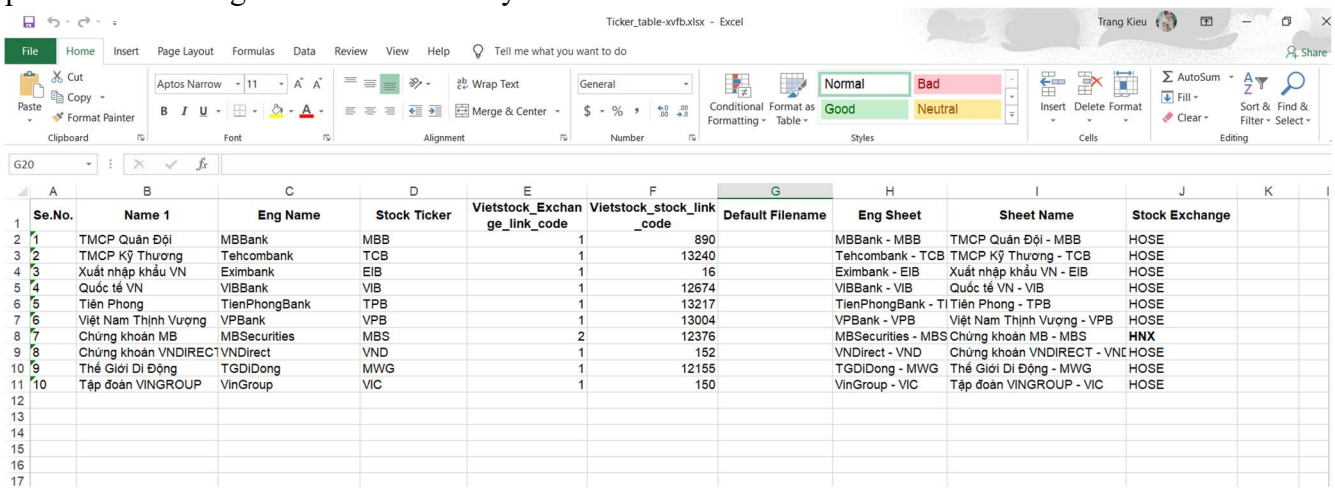
```
#!/bin/bash
```

```
# Set up the timezone for the Docker container
ln -snf /usr/share/zoneinfo/Asia/Ho_Chi_Minh /etc/localtime
echo "Asia/Ho_Chi_Minh" > /etc/timezone
# Create the new database and tables
/opt/mssql-tools18/bin/sqlcmd -S 192.168.10.100 -U trangkieu4 -P 123456 -N -C -i
/createstockdt.sql
```

B – BUILDING CONTAINER APPLICATIONS

1. Crawling raw data and inserting crawled data into MongoDB

I create an Excel file named "Ticker_table-xvfb.xlsx" which contains basic info of 10 stocks in my portfolio including variables used in my source codes :



Se.No.	Name 1	Eng Name	Stock Ticker	Vietstock_Exchange_link_code	Vietstock_stock_link_code	Default Filename	Eng Sheet	Sheet Name	Stock Exchange
1	TMCP Quân Đội	MBBank	MBB	1	890		MBBank - MBB	TMCP Quân Đội - MBB	HOSE
2	TMCP Kỹ Thương	Techcombank	TCB	1	13240		Techcombank - TCB	TMCP Kỹ Thương - TCB	HOSE
3	Xuất nhập khẩu VN	Eximbank	EIB	1	16		Eximbank - EIB	Xuất nhập khẩu VN - EIB	HOSE
4	Quốc tế VN	VIBBank	VIB	1	12674		VIBBank - VIB	Quốc tế VN - VIB	HOSE
5	Tiên Phong	TienPhongBank	TPB	1	13217		TienPhongBank - TI	Tiên Phong - TPB	HOSE
6	Việt Nam Thịnh Vượng	VPBank	VPB	1	13004		VPBank - VPB	Việt Nam Thịnh Vượng - VPB	HOSE
7	Chứng khoán MB	MBSecurities	MBS	2	12376		MBSecurities - MBS	Chứng khoán MB - MBS	HNX
8	Chứng khoán VNDIREC	VNDirect	VND	1	152		VNDirect - VND	Chứng khoán VNDIRECT - VNI	HOSE
9	Thế Giới Di Động	TGDiDong	MWG	1	12155		TGDiDong - MWG	Thế Giới Di Động - MWG	HOSE
10	Tập đoàn VINGROUP	VinGroup	VIC	1	150		VinGroup - VIC	Tập đoàn VINGROUP - VIC	HOSE

1.1. Numerous data:

For each stock code, the URL link has a rule for its string structure. Variables in the string are from "Vietstock_exchange_link_code" and "Vietstock_stock_link_code" columns of the Excel file. I create a "for" loop to iterate through all stock lists and scrape data in each loop. Additionally, I use Selenium for simulating the actions of human when suffering the internet in order to avoid crawling prevention from the website administrator. It does not only help me input the log-in information by clicking on "Log in" button but can also help load more dynamically loaded data by clicking on the Next button of each page and wait for some seconds until the element is visible. Thus, each page is displayed with full html sources and all the XPath of elements are available for scraping. I create another "for" loop to ietrate through all pages of the table to scrape data. This is the content of the source code file – "dbstockcrawl.py":

```
print("Let's start crawling!")
import scrapy
import selenium
import pandas as pd
import time
from selenium import webdriver
from selenium.webdriver.edge.service import Service
from selenium.webdriver.edge.options import Options
from selenium.webdriver.common.desired_capabilities import DesiredCapabilities
from selenium.webdriver.edge.service import Service
```

```
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from scrapy.selector import Selector
import pymongo
from pymongo import MongoClient
# Delete import-to-mongo def
#class Spider9(scrapy.Spider):
name = "spider9"
url1 = "https://finance.vietstock.vn/ket-qua-giao-dich?tab=thong-ke-gia"
import pandas as pd
#tickertable = pd.read_excel("D:\\1_Docker-compose\\scrapy-con - 100 -
stocks\\Ticker_table-100.xlsx")
tickertable = pd.read_excel("/Ticker_table-xvfb.xlsx")
#tickertable = pd.read_excel("C:\\stockenv\\env\\Scripts\\crawl-con\\python-3-12-slim-
continue\\Ticker_table.xlsx")
stocklist = tickertable['Stock Ticker'].tolist()
stocklist1 = [str(stock) for stock in stocklist]
linkcode = tickertable['Vietstock_stock_link_code'].tolist()
linkcode1 = [str(link) for link in linkcode]
exchangelist = tickertable['Vietstock_Exchange_link_code'].tolist()
exchangelist1 = [str(exchange) for exchange in exchangelist]
data = []
for i in range (len(stocklist)):
    url = url1 + "&exchange=" + exchangelist1[i] + "&code=" + linkcode1[i]
    # Selenium webdriver setup
    service = Service("/usr/local/bin/msedgedriver")
    options = webdriver.EdgeOptions()
    options.add_argument("--window-size=1920,1080")
    options.add_argument("headless")  # Optional: Run in headless mode
    options.add_argument("--no-sandbox")
    options.add_argument("--disable-dev-shm-usage")
    options.add_argument("--disable-gpu")
    # avoid timeout error:
    #prefs = {
    #"profile.managed_default_content_settings.images": 2,
    #"profile.managed_default_content_settings.stylesheets": 2,
    #"profile.managed_default_content_settings.javascript": 2
    #}
    #options.add_experimental_option("prefs", prefs)

    driver = webdriver.Edge(service=service, options=options)
    #driver.set_page_load_timeout(600) # avoid timeout crash
    driver.get(url)
    try:
        driver.maximize_window()
    except:
        print("can not maximize the window")
```

```
    # Click on Excel download button:
    # Find the button by its class name and click it
    try:
        button_container = WebDriverWait(driver, 4).until(
            EC.visibility_of_element_located((By.XPATH, "//div[@class='col-md-6']"))
        )
        # Now find the ExCEL Download button within the div:
        # Find the <a> element
        a_element = WebDriverWait(driver, 4).until(
            EC.visibility_of_element_located((By.XPATH, "//a[@title='Export Excel']"))
        )
    except Exception as e:
        print("Can not click the button: ", e)
    try:
        a_element.click()
    except Exception:
        print("Can not click Xem button")

    # Log in:
    # Find the container:
    login_container = driver.find_element(By.XPATH,"//a[@class='title-link btnlogin-
link']")

    # Click on the log in container to activate it
    #login_container.click()

    # Log in:
    # Find the email input field by its ID
    email_input = driver.find_element(By.ID,"txtEmailLogin")

    # Input your email into the field
    email_input.send_keys("kieuhuyentrang@gmail.com")

    # Find the password input field by its ID
    password = driver.find_element(By.ID,"txtPassword")

    # Input your password into the field
    password.send_keys("767122119")

    # Optionally, you can submit the form after inputting the email
    login_button = driver.find_element(By.ID, "btnLoginAccount")
    login_button.click()

    # Wait for the page to stabilize after login
    WebDriverWait(driver, 10).until(EC.staleness_of(button_container))

    # Input values:
```

```
    # Wait for the input field container to be visible
    input_container1 = WebDriverWait(driver, 4).until(
        EC.visibility_of_element_located((By.XPATH, "//*[@id='txtFromDate']"))
    )

    # Get the current date
    end_date = pd.Timestamp.today()

    # Calculate the start date as 2 years (365*2 days) before the end date
    start_date = end_date - pd.Timedelta(days=365*2)

    # Format the dates as text in the format dd/mm/yyyy
    end_date_text = end_date.strftime('%d/%m/%Y')
    start_date_text = start_date.strftime('%d/%m/%Y')

    # Click on the input container to activate it
    input_container1.click()

    # Now, locate and input the value into the field
    input_field1 = WebDriverWait(driver, 4).until(
        EC.visibility_of_element_located((By.XPATH, "//div[@id='txtFromDate']/input"))
    )
    input_field1.clear()  # Clear existing value if any
    input_field1.send_keys(start_date_text) #  "01/04/2023"

    # Similar to ToDate:
    # Wait for the input field container to be visible
    input_container2 = WebDriverWait(driver, 4).until(
        EC.visibility_of_element_located((By.ID, "txtToDate"))
    )

    # Click on the input container to activate it
    input_container2.click()

    # Now, locate and input the value into the field
    input_field2 = WebDriverWait(driver, 4).until(
        EC.visibility_of_element_located((By.XPATH, "//div[@id='txtToDate']/input"))
    )
    input_field2.clear()  # Clear existing value if any
    input_field2.send_keys(end_date_text) # "23/03/2024"

    # Find the button Contains XEM button
    button_container = WebDriverWait(driver, 4).until(
        EC.visibility_of_element_located((By.XPATH, "//div[@class='col-md-6']"))
    )
    # Wait for the page to stabilize after login
    # WebDriverWait(driver, 10).until(EC.staleness_of(button_container))
    Xembt = WebDriverWait(driver, 10).until(
```

```
        EC.element_to_be_clickable((By.XPATH, "//button[text()='Xem']"))
    )
    Xembt = WebDriverWait(driver, 4).until(EC.element_to_be_clickable((By.XPATH,
"//*[@id='trading-result']/div/div[1]/div[1]/div/div[2]/button")))
    #Click the button
    Xembt.click()

    # Select 20 displayed lines for each page
    select = WebDriverWait(driver, 4).until(
        EC.visibility_of_element_located((By.XPATH, '//*[@id="trading-
result"]/div/div[3]/div/div/div[1]/div/select'))
    )
    select.find_element(By.XPATH, "//option[text()='20']").click()
    time.sleep(1)
    # thêm tên cột:
    columns = ['STT', 'Date', 'Stock', 'Basic', 'Open', 'Close', 'High', 'Low',
'Average', 'PriceChange', 'PriceChangeRate', 'OrdMatVol', 'OrdMatVal', 'PutThrVol',
'PutThrVal', 'TotalTranVol', 'TotalTranVal', 'MarCap']
    # Scraping each page:
    for p in range(1, 14): # for each page after "Next" clicking
        #tbody = response.meta['tbody']
        # Locate the parent <div> element
        #parent_div = driver.find_element(By.XPATH, "//div[@class='parent-div-
class']") # wrong xpath
        parent_div = driver.find_element(By.XPATH, '//*[@id="trading-
result"]/div/div[3]') # try another XPath level

        # Use the parent <div> to locate all <tr> elements representing rows in the
updated content
        new_rows = parent_div.find_elements(By.XPATH, './/tr')
        time.sleep(3) #
        client = pymongo.MongoClient('mongodb://mongo1:27017/') # thay mongodb =
mongo1
        db = client['dbstock']
        collection = db['tbl10stocks']
        # Process the new rows as needed
        for row in new_rows:
            #item = Spider007Item()
            row_selector = Selector(text=row.get_attribute("outerHTML"))
            cols = row_selector.xpath('.//td')
            row_data = []
            for col in cols:
                row_data.append(col.xpath('.//text()').get().strip())
                # Ensure that row_data has the same length as columns
            if len(row_data) == len(columns):
                document = {columns[i]: row_data[i] for i in range(len(columns))}

                ## IMPORT VÀO MONGODB:
```

```
                try:
                    collection.insert_one(document)
                    print("Row index {row} of the stock {stocklist1[i]} inserted
successfully into MongoDB")
                except Exception as e:
                    print(f"An error occurred: {e}")
            else:
                print(f"Data length mismatch: Expected {len(columns)}, got
{len(row_data)}")
                print(f"Row data: {row_data}")
        time.sleep(3)
        #df = pd.DataFrame(data)
        #time.sleep(5)
        try:
            # Wait for the button to be clickable, handling potential dynamic loading
            nextbutton = WebDriverWait(driver, 6).until(
            EC.element_to_be_clickable((By.XPATH, "//*[@id='btn-page-next']/i"))
            )
            nextbutton.click()
            time.sleep(3) #
            continue
        except:
            break
    #print("One stock has been scraped and inserted into MongoDB successfully!")
    print(f"The stock {stocklist1[i]} with index {i} has been scraped and inserted
into MongoDB")
driver.quit()
print("CRAWLING PROSCESS HAS JUST FINISHED!")
```

For each row scraped from the website, I immediately insert it into MongoDB. If it fails, pass it and continue with the next one with try – except block.

1.2. Text data:

Different from numerous data crawling, crawling content is much simpler. The only different part of the main URL link of each stock is indeed the stock code itself. I can get all the URL links of latest articles displayed in the main URL link through “find_elements” method of Scrapy. Since then, I can extract all needed text. This is the content of the source code file – “contentpubver3.py”:

```
print("Content Crawling Has Just Begun ...")
import scrapy
import selenium
import pandas as pd
import time
from selenium import webdriver
from selenium.webdriver.edge.service import Service
from selenium.webdriver.edge.options import Options
from selenium.webdriver.common.desired_capabilities import DesiredCapabilities
from selenium.webdriver.edge.service import Service
from selenium.webdriver.common.by import By
```

```
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from scrapy.selector import Selector
import numpy as np
import pymongo
from selenium.common.exceptions import NoSuchElementException, TimeoutException
import random
## INITIALIZE STOCK CODES, LISTS TO SAVE COLUMN VALUES
#tickertable = pd.read_excel("D:\\1_Docker-compose\\scrapy-con - 100 -
stocks\\Ticker_table-100.xlsx")
tickertable = pd.read_excel("/Ticker_table-xvfb.xlsx")
#tickertable = pd.read_excel("C:\\stockenv\\env\\Scripts\\crawl-con\\python-3-12-slim-
continue\\Ticker_table.xlsx")
stocklist = tickertable['Stock Ticker'].tolist()
stocklist1 = [str(stock) for stock in stocklist]
linkcode = tickertable['Vietstock_stock_link_code'].tolist()
linkcode1 = [str(link) for link in linkcode]
exchangelist = tickertable['Vietstock_Exchange_link_code'].tolist()
exchangelist1 = [str(exchange) for exchange in exchangelist]
stock = []
link = []
content = []
## IN THE LOOP OF EACH STOCK CODE:
links = []
contents = []
codes = []
import pymongo
client = pymongo.MongoClient('mongodb://mongo1:27017/')
db = client['dbstock']
collection = db['tbcontent']
for i in range (len(stocklist1)):
    # Open news link in headless mode
    url = "https://finance.vietstock.vn/"+stocklist1[i]+"/tin-tuc-su-kien.htm"
    service = Service("/usr/local/bin/msedgedriver")
    options = webdriver.EdgeOptions()
    options.add_argument("--headless")
    options.add_argument("--window-size=1920,1080")
    options.add_argument("--no-sandbox")
    options.add_argument("--disable-dev-shm-usage")
    options.add_argument("--disable-gpu")
    driver = webdriver.Edge(service=service, options=options)
    driver.get(url)
    # Log in:
    # Wait for the login button and click it if necessary
    WebDriverWait(driver, 10).until(EC.element_to_be_clickable((By.XPATH,
"//a[@class='title-link btnlogin-link']"))).click()
    login_container = driver.find_element(By.XPATH,"//a[@class='title-link btnlogin-
link']")
```

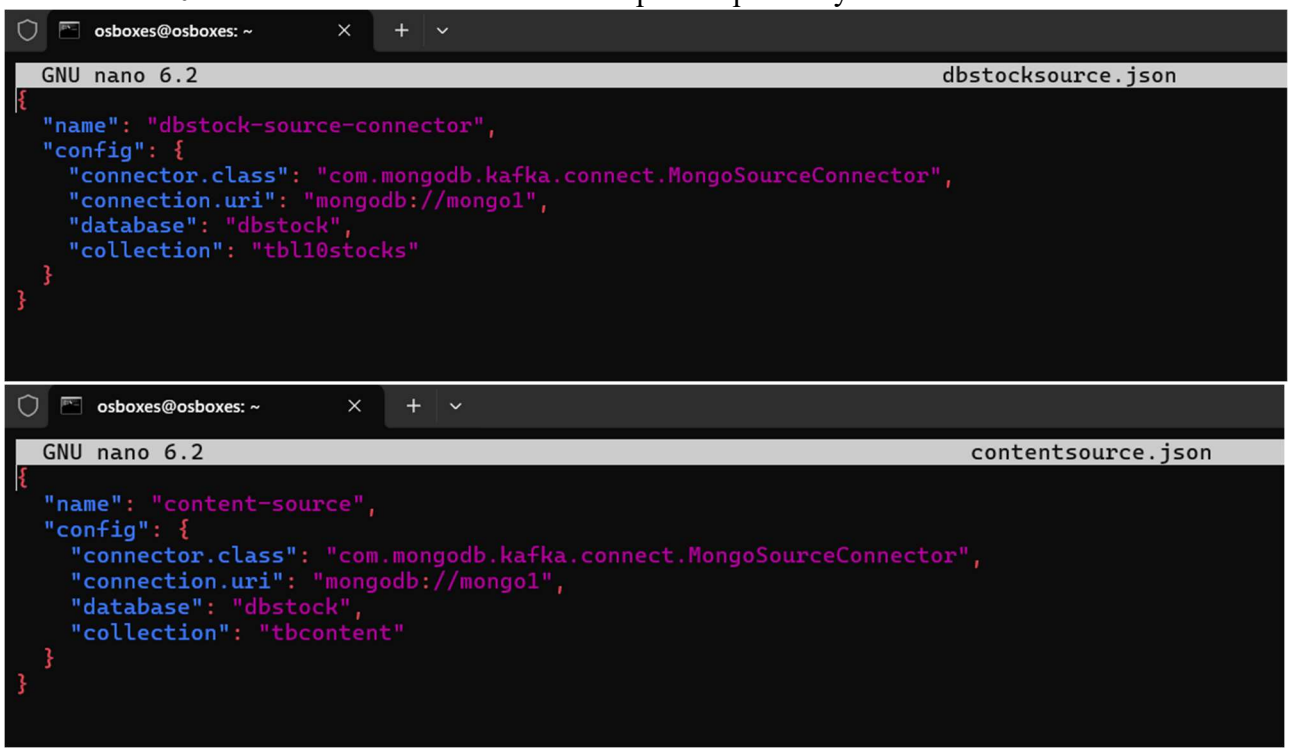
```
    time.sleep(3)
    email_input = driver.find_element(By.ID,"txtEmailLogin")
    email_input.send_keys("kieuhuyentrang@gmail.com")
    password = driver.find_element(By.ID,"txtPassword")
    password.send_keys("767122119")
    login_button = driver.find_element(By.ID, "btnLoginAccount")
    login_button.click()
    WebDriverWait(driver, 10).until(EC.staleness_of(login_button))
    # Get all article links:
    ## Find the element with the id 'newsevent-content'
    newsevent_content = driver.find_element(By.ID, 'newsevent-content') # The top
level contains the href and is common to all stock codes.
    # Find all <a> elements within 'newsevent-content'
    a_tags = newsevent_content.find_elements(By.TAG_NAME,'a') # the tag that contains
all hrefs
    hrefs = []
    for a_tag in a_tags:
        href = a_tag.get_attribute('href') # Each href is a link to an article.
        hrefs.append(href)
    print(hrefs)
    # Go to each link to scrape all text::
    #links = []
    #contents = []
    for href in hrefs: # for each article link
        if href != None:
            links.append(href) # them urlcontent vao links
            codes.append(stocklist1[i])
            driver.get(href)
            try:
                driver.maximize_window()
            except:
                print("can not maximize the window")
            wait = WebDriverWait(driver, 3)
            # Generate a random number of lines to scroll down (between 5 and 30) -
mimik human actions
            lines_to_scroll = random.randint(5, 30)
            # Assuming each line is approximately 20 pixels in height
            pixels_to_scroll = lines_to_scroll * 20
            # Scroll down by the calculated number of pixels
            driver.execute_script(f"window.scrollBy(0, {pixels_to_scroll});")
            # Pause to see the effect (optional)
            sec = random.uniform(1,3)
            time.sleep(sec)
            # Find all elements with the specified class name
            elements = driver.find_elements(By.CSS_SELECTOR, 'div.col-lg-10.col-sm-
12.col-md-12')
            # Initialize date_text to ensure it's defined
            date_text = None
```

```
            try:
                # Wait until the elements are visible
                element = WebDriverWait(driver, 10).until(
                    EC.presence_of_element_located((By.XPATH,
"/html/body/div[14]/div[5]/div[1]/div/section/div[1]/div/div[1]/div/div/div[1]/div[2]/
div[2]/div/div[1]/div/span"))
                )
                # Get the text of the elements:
                date_text = element.text
            except (NoSuchElementException, TimeoutException):
                # Handle the exception and print "Error"
                print("Error: Can't find PublishDate")
                date_text = "Unknown Date"
            title_text = None
            try:
                title_element = WebDriverWait(driver, 10).until(
                    EC.presence_of_element_located((By.XPATH,"//h1[contains(@class,'ti
tle')]")))
                # Get the text of the elements:
                title_text = title_element.text
            except (NoSuchElementException, TimeoutException):
                # Handle the exception and print "Error"
                print("Error: Can't find Title")
                title_text = "Unknown Title"
            text_contents = []
            for element in elements:
                text_contents.append(element.text)
            ## IMSERT DATA INTO MONGODB:
            # Create a dictionary representing the MongoDB document
            document = {
                "Stock": stocklist1[i],  # Renamed variable for valid field name
                "Link": href,
                "Contents": text_contents,
                "PublishDate" : date_text,
                "Title" : title_text
            }
            try:
                result = collection.insert_one(document)
                print("Data inserted successfully")
                print(document)
            except Exception as e:
                print(f"An error occurred: {e}")
        second = random.uniform(1, 4)
        time.sleep(second)
driver.quit()
print("\nContent Crawling has just finished!")
```

2. Creating and registering Kafka topics from MongoDB

Before creating a Kafka topic, be sure that all Kafka services are running in healthy status. Then, follow these steps:

- Firstly, a topic based on the name of database and its collection are created inside MongoDB container. The json file which contains information about this topic is called a "source connector". Below are the files named "dbstocksource.json" and "contentsource.json" to create "dbstock.tbl10stocks" and "dbstock.tbcontent" topics respectively:



- Then, these connectors are registered to Kafka using POST method using bash shell commands:

```
curl -X POST -H "Content-Type: application/json" -d @ dbstocksource.json http://connect:8083/connectors -w "\n"
```

```
curl -X POST -H "Content-Type: application/json" -d @ contentsource.json http://connect:8083/connectors -w "\n"
```

- Once it's done, check the status of topics:

```
osboxes@osboxes: ~

MongoDB Kafka Sandbox $status

Kafka topics:

    "topic": "dbcontent.tbcontent",
    "topic": "dbMBB.db249lines",
    "topic": "docker-connect-status",
    "topic": "dbstock.tbcontent",
    "topic": "dbstock.tbl10stocks",
    "topic": "dbstock.10stocks",
    "topic": "__consumer_offsets",
    "topic": "Tutorial1.orders",
    "topic": "docker-connect-offsets",
    "topic": "docker-connect-configs",
    "topic": "dbMBBver2.db249lines2",

The status of the connectors:

source  |  content-source            |  RUNNING  |  RUNNING  |  com.mongodb.kafka.connect.MongoSourceConnector
source  |  content-source1           |  RUNNING  |  RUNNING  |  com.mongodb.kafka.connect.MongoSourceConnector
source  |  contentsrc                |  RUNNING  |  RUNNING  |  com.mongodb.kafka.connect.MongoSourceConnector
source  |  dbMBB-source              |  RUNNING  |  RUNNING  |  com.mongodb.kafka.connect.MongoSourceConnector
source  |  dbMBB-source2             |  RUNNING  |  RUNNING  |  com.mongodb.kafka.connect.MongoSourceConnector
source  |  dbstock-source            |  RUNNING  |  RUNNING  |  com.mongodb.kafka.connect.MongoSourceConnector
source  |  dbstock-source-connector  |  RUNNING  |  RUNNING  |  com.mongodb.kafka.connect.MongoSourceConnector
source  |  mongo-simple-source       |  RUNNING  |  RUNNING  |  com.mongodb.kafka.connect.MongoSourceConnector

Currently configured connectors

[
  "content-source",
  "dbMBB-source2",
  "dbMBB-source",
  "contentsrc",
  "mongo-simple-source",
  "dbstock-source",
  "dbstock-source-connector",
  "content-source1"
]
```

As we can see from the output, the two topics are active.

- Display data from the topic into the screen to see streamed data inserted into MongoDB by Scrapy application in real time. Also, a full structure schema of each item/document from MongoDB is shown at the same moment.

```
MongoDB Kafka Sandbox $kc dbstock.tbl10stocks
```

```
osboxes@osboxes: ~

cumentKey\": {\"_id\": {\"$oid\": \"66cd5f48c82a6c7416b91ecc\"}}}"}
------------------------


Partition: 0    Offset: 39920

Key (109 bytes):

{"schema":{"type":"string","optional":false},"payload":"{\"_id\": {\"$oid\": \"66cd5f48c82a6c7416b91ecd\"}}"}

Value (1003 bytes):

{"schema":{"type":"string","optional":false},"payload":"{\"_id\": {\"_data\": \"8266CD5F48000000092B022C0100296E5A10046C91E43D78784961B5F019907A07CE38466 45F69640064 66CD5F48C82A6C7416B91ECD0004\"}, \"operationType\": \"insert\", \"clusterTime\": {\"$timestamp\": {\"t\": 1724735304, \"i\": 9}}, \"wallTime\": {\"$date\": 1724735304289}, \"fullDocument\": {\"_id\": {\"$oid\": \"66cd5f48c82a6c7416b91ecd\"}, \"STT\": \"249\", \"Date\": \"29/08/2023\", \"Stock\": \"VIC\", \"Basic\": \"64.70\", \"Open\": \"65.70\", \"Close\": \"63.40\", \"High\": \"65.90\", \"Low\": \"63.20\", \"Average\": \"64.05\", \"PriceChange\": \"-1,300\", \"PriceChangeRate\": \"-2.01\", \"OrdMatVol\": \"20,650,300\", \"OrdMatVal\": \"1,322,741\", \"PutThrVol\": \"200,002\", \"PutThrVal\": \"12,790\", \"TotalTranVol\": \"20,850,302\", \"TotalTranVal\": \"1,335,531\", \"MarCap\": \"241,803,515\"}, \"ns\": {\"db\": \"dbstock\", \"coll\": \"tbl10stocks\"}, \"documentKey\": {\"_id\": {\"$oid\": \"66cd5f48c82a6c7416b91ecd\"}}}"}
------------------------


Partition: 0    Offset: 39921

Key (109 bytes):

{"schema":{"type":"string","optional":false},"payload":"{\"_id\": {\"$oid\": \"66cd5f48c82a6c7416b91ece\"}}"}

Value (1000 bytes):

{"schema":{"type":"string","optional":false},"payload":"{\"_id\": {\"_data\": \"8266CD5F480000000A2B022C0100296E5A10046C91E43D78784961B5F019907A07CE38466 45F69640064 66CD5F48C82A6C7416B91ECE0004\"}, \"operationType\": \"insert\", \"clusterTime\": {\"$timestamp\": {\"t\": 1724735304, \"i\": 10}}, \"wallTime\": {\"$date\": 1724735304301}, \"fullDocument\": {\"_id\": {\"$oid\": \"66cd5f48c82a6c7416b91ece\"}, \"STT\": \"250\", \"Date\": \"28/08/2023\", \"Stock\": \"VIC\", \"Basic\": \"63.50\", \"Open\": \"65.00\", \"Close\": \"64.70\", \"High\": \"66.70\", \"Low\": \"64.50\", \"Average\": \"65.34\", \"PriceChange\": \"1,200\", \"PriceChangeRate\": \"1.89\", \"OrdMatVol\": \"16,715,600\", \"OrdMatVal\": \"1,092,226\", \"PutThrVol\": \"79,000\", \"PutThrVal\": \"5,188\", \"TotalTranVol\": \"16,794,600\", \"TotalTranVal\": \"1,097,414\", \"MarCap\": \"246,761,631\"}, \"ns\": {\"db\": \"dbstock\", \"coll\": \"tbl10stocks\"}, \"documentKey\": {\"_id\": {\"$oid\": \"66cd5f48c82a6c7416b91ece\"}}}"}
------------------------

% Reached end of topic dbstock.tbl10stocks [0] at offset 39922
```

Such things are similar to “dbstock.tbcontent” topic:

```
MongoDB Kafka Sandbox $kc dbstock.tbcontent
```

Note that “status” and “kc” commands are user-defined functions provided by MongoDB. For more details, please view the full source codes attached.

3. Retrieving data from Kafka topics followed by transforming and cleaning data using PySpark

- The steps involved in this task are:
  - Creating a spark session.
  - Connecting Kafka server and reading a specific topic.
  - Assign a correct schema structure of data from Kafka which originally created by MongoDB.
  - Deserializing json structure of data and selecting needed columns.
  - Transforming data in columns into correct data type (string to date, integer, float types) using pyspark.sql.functions.
  - Writing to Postgres in writeStream mode.
- "dbstockread.py" source codes:

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import from_json, col, regexp_replace, date_format,
to_date, when
from pyspark.sql.types import StructType, StructField, StringType, LongType,
IntegerType, FloatType

# Create Spark session
spark = SparkSession.builder \
    .master("local") \
    .appName("KafkaSparkExample") \
```

```
    .config("spark.sql.streaming.kafka.consumer.cache.enabled", "false") \
    .config("spark.sql.streaming.kafka.consumer.pollTimeoutMs", "512") \
    .getOrCreate()

# Set the logging level for Kafka to ERROR to suppress warnings
spark.sparkContext.setLogLevel("ERROR")

# Define the Kafka bootstrap servers and topic
kafka_bootstrap_servers = "broker:29092"
kafka_topic = "dbstock.tbl10stocks"

# Define schema for the JSON content inside the 'payload' field
# 1
value_schema = StructType([
                    StructField("schema", StructType([
                        StructField("type", StringType(), True),
                        StructField("optional", StringType(), False)
                    ]),True),
                    StructField("payload", StringType(), True)
                    ])
df = spark.readStream \
    .format("kafka") \
    .option("kafka.bootstrap.servers", kafka_bootstrap_servers) \
    .option("subscribe", kafka_topic) \
    .load()
# 2
deserialized_df = df.select(
    from_json(col("value").cast("string"), value_schema).alias("parsed_value")
).select(
    from_json(col("parsed_value.payload").cast("string"),
              StructType([
                  StructField("_id", StructType([
                      StructField("_data", StringType(), True)
                  ]), True),
                  StructField("operationType", StringType(), True),
                  StructField("clusterTime", StructType([
                      StructField("$timestamp", StructType([
                          StructField("t", StringType(), True),
                          StructField("i", StringType(), True)
                      ]), True)
                  ]), True),
                  StructField("wallTime", StructType([
                      StructField("$date", StringType(),True)
                  ]), True),
                  StructField("fullDocument", StructType([
                      StructField("_id", StructType([
                          StructField("$oid", StringType(), True)
                      ]), True),
```

```
                    StructField("STT", StringType(), True),
                    StructField("Date", StringType(), True),
                    StructField("Stock", StringType(), True),
                    StructField("Basic", StringType(), True),
                    StructField("Open", StringType(), True),
                    StructField("Close", StringType(), True),
                    StructField("High", StringType(), True),
                    StructField("Low", StringType(), True),
                    StructField("Average", StringType(), True),
                    StructField("PriceChange", StringType(), True),
                    StructField("PriceChangeRate", StringType(), True),
                    StructField("OrdMatVol", StringType(), True),
                    StructField("OrdMatVal", StringType(), True),
                    StructField("PutThrVol", StringType(), True),
                    StructField("PutThrVal", StringType(), True),
                    StructField("TotalTranVol", StringType(), True),
                    StructField("TotalTranVal", StringType(), True),
                    StructField("MarCap", StringType(), True)
                ]), True),
                StructField("ns", StructType([
                    StructField("db", StringType(), True),
                    StructField("coll", StringType(), True)
                ]), True),
                StructField("documentKey", StructType([
                    StructField("_id", StructType([
                        StructField("$oid", StringType(), True)
                    ]), True)
                ]), True)
            ])
    ).alias("payload_json")
)
full_document_df = deserialized_df.select(
    col("payload_json.fullDocument.*")
)
# 3
# Remove commas as decimal separators from specific fields
cleaned_df = full_document_df.select(
    col("STT"),
    col("Date"),
    col("Stock"),
    regexp_replace(col("Basic"), ",", "").alias("Basic"),
    regexp_replace(col("Open"), ",", "").alias("Open"),
    regexp_replace(col("Close"), ",", "").alias("Close"),
    regexp_replace(col("High"), ",", "").alias("High"),
    regexp_replace(col("Low"), ",", "").alias("Low"),
    regexp_replace(col("Average"), ",", "").alias("Average"),
    regexp_replace(col("PriceChange"), ",", "").alias("PriceChange"),
    regexp_replace(col("PriceChangeRate"), ",", "").alias("PriceChangeRate"),
```

```
    regexp_replace(col("OrdMatVol"), ",", "").alias("OrdMatVol"),
    regexp_replace(col("OrdMatVal"), ",", "").alias("OrdMatVal"),
    regexp_replace(col("PutThrVol"), ",", "").alias("PutThrVol"),
    regexp_replace(col("PutThrVal"), ",", "").alias("PutThrVal"),
    regexp_replace(col("TotalTranVol"), ",", "").alias("TotalTranVol"),
    regexp_replace(col("TotalTranVal"), ",", "").alias("TotalTranVal"),
    regexp_replace(col("MarCap"), ",", "").alias("MarCap")
)

# Convert dd/MM/yyyy to MM/dd/yyyy format
df_transformed = cleaned_df.withColumn(
    "DateChange",
    date_format(to_date(col("Date"), "dd/MM/yyyy"), "MM/dd/yyyy")
)

# Convert specific columns from string to integer
integer_columns = ['OrdMatVol', 'OrdMatVal', 'PutThrVol', 'PutThrVal', 'TotalTranVol',
'TotalTranVal', 'MarCap']
for col_name in integer_columns:
    df_transformed = df_transformed.withColumn(col_name,
col(col_name).cast("integer"))

# Convert specific columns from string to float
float_columns = ['PriceChange', 'PriceChangeRate', 'Basic', 'Open', 'Close', 'High',
'Low', 'Average']
for col_name in float_columns:
    df_transformed = df_transformed.withColumn(col_name, when(col(col_name) != "-",
col(col_name).cast("float"))
                                   .otherwise(None))

# Define PostgreSQL JDBC URL and properties
postgres_url = "jdbc:postgresql://postgres:5432/dbstock"
postgres_properties = {
    "user": "admin",
    "password": "admin",
    "driver": "org.postgresql.Driver"
}

# Write DataFrame to PostgreSQL
def write_to_postgres(df, epoch_id):
    df.write.jdbc(
        url=postgres_url,
        table="tbl10stocks",
        mode="append",
        properties=postgres_properties
    )

# Write DataFrame to PostgreSQL in streaming mode
```

```
query1 = df_transformed.writeStream \
    .foreachBatch(write_to_postgres) \
    .outputMode("append") \
    .start()

# Write DataFrame to console for debugging
query2 = df_transformed.writeStream \
    .outputMode("append") \
    .format("console") \
    .option("truncate", "false") \
    .start()

# Keep the streaming query running
query1.awaitTermination()
query2.awaitTermination()
spark.stop()
```

- "dbcontent-15.py" source codes:

```
from pyspark.sql import SparkSession, Window
from pyspark.sql.functions import udf, from_json, col, regexp_replace, date_format,
to_date, when, expr, current_timestamp, to_timestamp
from pyspark.sql.functions import explode, split, col, sum, when, row_number, desc,
max, window
from pyspark.sql.types import StructType, StructField, StringType, TimestampType
import pyspark.sql.functions as F
import re
from datetime import datetime, timedelta

# Create Spark session
spark = SparkSession.builder \
    .master("local") \
    .appName("KafkaSparkExample") \
    .config("spark.sql.streaming.kafka.consumer.cache.enabled", "false") \
    .config("spark.sql.streaming.kafka.consumer.pollTimeoutMs", "512") \
    .getOrCreate()

# Set the logging level for Kafka to ERROR to suppress warnings
spark.sparkContext.setLogLevel("ERROR")

# Define the Kafka bootstrap servers and topic
kafka_bootstrap_servers = "broker:29092"
kafka_topic = "dbstock.tbcontent"

# Define schema for the JSON content inside the 'payload' field
# 1
value_schema = StructType([
                    StructField("schema", StructType([
                        StructField("type", StringType(), True),
```

```
                        StructField("optional", StringType(), False)
                    ]),True),
                    StructField("payload", StringType(), True)
                    ])
df = spark.readStream \
    .format("kafka") \
    .option("kafka.bootstrap.servers", kafka_bootstrap_servers) \
    .option("subscribe", kafka_topic) \
    .load()
# 2
deserialized_df = df.select(
    from_json(col("value").cast("string"), value_schema).alias("parsed_value")
).select(
    from_json(col("parsed_value.payload").cast("string"),
              StructType([
                  StructField("_id", StructType([
                      StructField("_data", StringType(), True)
                  ]), True),
                  StructField("operationType", StringType(), True),
                  StructField("clusterTime", StructType([
                      StructField("$timestamp", StructType([
                          StructField("t", StringType(), True),
                          StructField("i", StringType(), True)
                      ]), True)
                  ]), True),
                  StructField("wallTime", StructType([
                      StructField("$date", StringType(),True)
                  ]), True),
                  StructField("fullDocument", StructType([
                      StructField("_id", StructType([
                          StructField("$oid", StringType(), True)
                      ]), True),
                      StructField("Stock", StringType(), True),
                      StructField("Link", StringType(), True),
                      StructField("Contents", StringType(), True),
                      StructField("PublishDate", StringType(), True),
                      StructField("Title", StringType(), True)
                  ]), True)
              ])
    ).alias("payload_json")
)
df3 = deserialized_df.select(
    col("payload_json.fullDocument.*"),
    col("payload_json.clusterTime.$timestamp.t").cast(TimestampType()).alias("event_ti
me")
)
# Select columns: Stock, Link, Contents, Title, NewsDate:
```

```
df5=
df3.select(col("Stock"),col("Link"),col("Contents"),col("PublishDate"),col("Title"))

##Cleaning:
### Filter out where "PublishDate" = "Unknown Date":
# Filter out rows where PublishDate is 'Unknown Date'
df4 = df5.filter(col("PublishDate") != "Unknown Date")

# Handling the strings "n hours ago", "n minutes ago" in the "PublishDate" column:
from datetime import datetime, timedelta
import re
def parse_relative_time(relative_time_str):
    # Define the date format
    date_format = '%m/%d/%Y'
    if relative_time_str == 'Unknown Date':
        return relative_time_str
    # Attempt to extract hours from the string
    match = re.match(r'(\d+) giờ trước', relative_time_str)
    if match:
        # Extract number of hours
        hours_ago = int(match.group(1))
        # Calculate the date by subtracting the hours from the current time
        past_time = datetime.now() - timedelta(hours=hours_ago)
        return past_time.strftime(date_format)
    # Attempt to extract minutes from the string
    match_minutes = re.match(r'(\d+) phút trước', relative_time_str)
    if match_minutes:
        # Extract number of minutes
        minutes_ago = int(match_minutes.group(1))
        # Calculate the date by subtracting the minutes from the current time
        past_time = datetime.now() - timedelta(minutes=minutes_ago)
        return past_time.strftime(date_format)
    try:
        # If the format does not match, use the provided date
        date_obj = datetime.strptime(relative_time_str, '%d/%m/%Y %H:%M')
        return date_obj.strftime(date_format)
    except ValueError:
        # Handle any errors in date parsing
        return relative_time_str
    else:
        try:
            # If the format does not match, use the provided date
            date_obj = datetime.strptime(relative_time_str, '%d/%m/%Y %H:%M')
            return date_obj.strftime(date_format)
        except ValueError:
            # Handle any errors in date parsing
            return relative_time_str
    # Return the date in the specified format
```

```
    #return past_time.strftime(date_format)
# Register the function as a UDF
parse_relative_time_udf = udf(parse_relative_time, StringType())
# Apply the UDF to transform the relative time
df_transformed = df4.withColumn("formatted_date",
parse_relative_time_udf(col("PublishDate")))
# Convert the formatted date strings to date type
df5 = df_transformed.withColumn("DateChange", to_date(col("formatted_date"),
"MM/dd/yyyy"))
df6 =
df5.select(col("Stock"),col("Link"),col("Contents"),col("PublishDate"),col("Datechange
"),col("formatted_date"),col("Title"))
# Define PostgreSQL JDBC URL and properties
postgres_url = "jdbc:postgresql://postgres:5432/dbstock"
postgres_properties = {
    "user": "admin",
    "password": "admin",
    "driver": "org.postgresql.Driver"
}

# Write DataFrame to PostgreSQL
def write_to_postgres(df, epoch_id):
    df.write.jdbc(
        url=postgres_url,
        table="tbcontent",
        mode="append",
        properties=postgres_properties
    )

# Write DataFrame to PostgreSQL in streaming mode
query1 = df6.writeStream \
    .foreachBatch(write_to_postgres) \
    .outputMode("append") \
    .start()

# Write DataFrame to console for debugging
query2 = df6.writeStream \
    .outputMode("append") \
    .format("console") \
    .option("truncate", "false") \
    .start()

# Keep the streaming query running
query1.awaitTermination()
query2.awaitTermination()
spark.stop()
```

4. Creating database for data warehouse using SQL commands from Docker container in Linux

- Write an SQL script file named “createstockdt.sql” containing SQL commands:

```
-- Declare and initialize the database name variable
DECLARE @DatabaseName NVARCHAR(128) = 'dbstock4';

-- Create the new database
DECLARE @Sql NVARCHAR(MAX);
SET @Sql = 'CREATE DATABASE [' + @DatabaseName + '];';
EXEC sp_executesql @Sql;

-- Grant CONTROL permission to the user on the specified database
DECLARE @Sql1 NVARCHAR(MAX);
SET @Sql1 = 'USE [' + @DatabaseName + ']; GRANT CONTROL TO [' + 'trangkieu4' + '];';
EXEC sp_executesql @Sql;

-- Set the new database to multi-user mode (directly on the database name)
SET @Sql = 'ALTER DATABASE [' + @DatabaseName + '] SET MULTI_USER;';
EXEC sp_executesql @Sql;

-- Create tables directly in the new database without using USE
SET @Sql = '

CREATE TABLE [' + @DatabaseName + '].dbo.DataTime
(
  DateChange Date,
  Year int,
  Month int,
  Day int,
  PRIMARY KEY (DateChange)
);

CREATE TABLE [' + @DatabaseName + '].dbo.DataStock
(
  Stock VARCHAR (5),
  PRIMARY KEY (Stock)
);

CREATE TABLE [' + @DatabaseName + '].dbo.Fact
(
  Stock VARCHAR (5),
  DateChange Date,
  [Open] float,
  [Close] float,
  [Low] float,
  [High] float,
  PutThrVol Integer,
  OrdMatVol Integer,
  MarCap float,
  PRIMARY KEY (Stock, DateChange),
  FOREIGN KEY (Stock) REFERENCES [' + @DatabaseName + '].dbo.DataStock ([Stock]),
  FOREIGN KEY (DateChange) REFERENCES [' + @DatabaseName + '].dbo.DataTime
([DateChange])
);

CREATE TABLE [' + @DatabaseName + '].dbo.LinkDim
(
  Link_ID VARCHAR (40),
  [Link] VARCHAR (300),
```

```
        Title NVARCHAR (4000),
        Body NVARCHAR (MAX),
        PRIMARY KEY (Link_ID)
      );

      CREATE TABLE [' + @DatabaseName + '].dbo.LinkFact
      (
        Stock VARCHAR (5),
        DateChange Date,
        Link_ID VARCHAR (40),
        Link VARCHAR (300),
        Title NVARCHAR (4000),
        Body NVARCHAR (MAX),
        Count_of_phat Integer,
        Count_of_tra Integer,
        PRIMARY KEY (Stock, DateChange, Link_ID),
        FOREIGN KEY (Stock) REFERENCES [' + @DatabaseName + '].dbo.DataStock ([Stock]),
        FOREIGN KEY (DateChange) REFERENCES [' + @DatabaseName + '].dbo.DataTime
      ([DateChange]),
        FOREIGN KEY (Link_ID) REFERENCES [' + @DatabaseName + '].dbo.LinkDim ([Link_ID])
      );
      ';
      EXEC sp_executesql @Sql;
```

5. Removing duplicates, creating data frames and inserting data frames into SQL Server database using PySpark

```
from pyspark.sql import SparkSession
from pyspark.sql import Row
from pyspark.sql import DataFrameWriter
from pyspark.sql import SparkSession
from pyspark.sql.functions import length, explode, col, when, regexp_replace,
row_number, lpad, expr, split
from pyspark.sql.functions import current_timestamp, year, month, dayofmonth, to_date,
format_number, lit, lpad, concat
from pyspark.sql.window import Window
from pyspark.sql import functions as F
# I - CREATE dataframe for these SSMS tables: DataStock, DataTime, Fact
# Step 1: Initialize SparkSession - # Link:
https://www.machinelearningplus.com/pyspark/pyspark-connect-to-postgresql/
jdbc_driver_path = "/opt/spark-3.5.1-bin-hadoop3/jars/postgresql-42.7.3.jar"
odbc_driver_path = "/app/mssql-jdbc-12.8.0.jre11.jar"
spark = SparkSession.builder \
    .appName("PostgreSQL Connection with PySpark") \
    .config("spark.jars", jdbc_driver_path) \
    .config("spark.jars", odbc_driver_path) \
    .getOrCreate()
# Step 2: Define your PostgreSQL database connection details
url = "jdbc:postgresql://postgres:5432/dbstock"
properties = {
    "user": "admin",
    "password": "admin",
    "driver": "org.postgresql.Driver"
```

```
}
table_name = "tbl10stocks"
# Step 3: Load the PostgreSQL table into a PySpark DataFrame
df = spark.read.jdbc(url, table_name, properties=properties)
df.show()
print("number of rows: {}".format(df.count())) # 5,510 rows
df.dtypes
print(len(df.columns))
print(df.count())
# Step 4: TRANSFORMING DATA - REMOVE DUPLICATES:
df_non_STT = df.select("Date", "Stock", "Basic", "Open", "Close", 'High', 'Low',
'Average', 'OrdMatVol', 'OrdMatVal', 'PutThrVol', 'PutThrVal', 'TotalTranVol',
'TotalTranVal', 'MarCap', 'DateChange' )
df_no_duplicates = df_non_STT.dropDuplicates(df_non_STT.columns)
df_no_duplicates.dtypes
# Filter out rows where "OrdMatVol" is null or 0
df_fact_filtered = df_no_duplicates.filter(
    (F.col("OrdMatVol").isNotNull()) & (F.col("OrdMatVol") != 0)
)
df_fact_filtered.show()
df_fact_filtered.count() # 2,500 rows
print(len(df_no_duplicates.columns))
print(df_no_duplicates.count())  # 1,525 rows
df_sorted = df_fact_filtered.orderBy(["Stock", "DateChange"])
df_sorted.show()
# Transform "DateChange" data type into "Date" type:
df_transformed = df_sorted.withColumn("DateChange", to_date("DateChange",
'MM/dd/yyyy'))
# Show the result
df_transformed.dtypes
print("rows of df_transformed: {}".format(df_transformed.count()))
print("rows of df_sorted: {}".format(df_sorted.count()))
print("rows of df_fact_filtered: {}".format(df_fact_filtered.count()))
# Step 5: Create tables in SQL Server database as in the schema
# Table 1 - Data_time
# Extract unique values from "DateChange"
unique_dates = df_transformed.select("DateChange").distinct()
unique_dates.count() # 253 rows

data_time = unique_dates.withColumn("Year", year(col("DateChange"))) \
                        .withColumn("Month", month(col("DateChange"))) \
                        .withColumn("Day", dayofmonth(col("DateChange"))) \
# Sort the DataFrame by TimeID in ascending order
data_time1 = data_time.orderBy("DateChange")
data_time1.show()
print(data_time1.count()) # 253 rows
# Table 2 - Data_stock:
# Extract unique Stock Codes
```

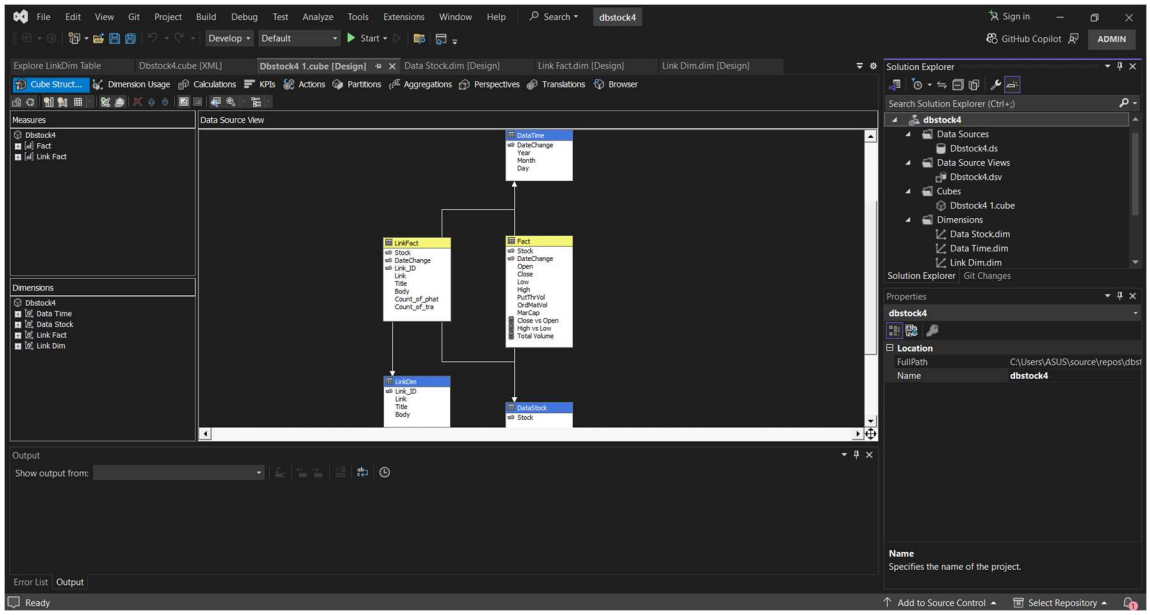
```
unique_stocks = df_transformed.select("Stock").distinct().orderBy("Stock")
data_stock = unique_stocks.orderBy("Stock")
data_stock.show()
# Table 3 - Fact table:
df_fact = df_transformed.select("Stock", "DateChange", "Open", "Close", "Low", "High",
"PutThrVol", "OrdMatVol", "MarCap")
df_fact.show()
df_fact.dtypes
print(df_fact.count())
df_fact.show()
df_fact.dtypes
print("number of columns: "+str(len(df_fact.columns)))
print("number of rows: "+str(df_fact.count()))

# II - CREATE dataframe for these SSMS tables: LinkDim, LinkFact:
# Step 1: Initialize SparkSession - # Link:
https://www.machinelearningplus.com/pyspark/pyspark-connect-to-postgresql/
jdbc_driver_path = "/opt/spark-3.5.1-bin-hadoop3/jars/postgresql-42.7.3.jar"
#odbc_driver_path = "/app/mssql-jdbc-12.8.0.jre11.jar"
# Step 2: Define your PostgreSQL database connection details
table_name1 = "tbcontent"
# Step 3: Load the PostgreSQL table into a PySpark DataFrame
df_read = spark.read.jdbc(url, table_name1, properties=properties)
df_read.show()
print("number of rows: {}".format(df_read.count())) #
df_read.dtypes
print(len(df_read.columns))
print(df_read.count())
# remove duplicates:
dfundup = df_read.dropDuplicates()
# Step 4: create dataframe to be inserted into SSMS table named "LinkDim" with these
columns "Link", "Link_ID", "Title", "Contents" ("Body")
linkdimdup = dfundup.select(col("Link"),col("Title"),col("Contents").alias("Body"))
linkdim = linkdimdup.dropDuplicates(["Link"])
## Add "Link_ID" column contains distinct values generated by timestamp:
linkdim1 = linkdim.withColumn("Link_ID", expr("uuid()")) # maximum length of uuid() :
36 characters
linkdim2 = linkdim1.select(col("Link_ID"),col("Link"),col("Title"),col("Body"))
# Add a new column that contains the number of character in url column
df_with_url_length = linkdim.withColumn("url_length", length(col("Link")))
# Show the resulting DataFrame
df_with_url_length.show()
# Step 5a: Create dataframe to be inserted into SSMS table named "LinkFact" with these
columns "Stock", "Datechange", "Link_ID", "Link", "Contents" ("Body"), "Title", "Count
of phat", "Count of tra"
## Select "Stock", "Datechange", "Link", "Contents" ("Body"), "Title" from dfundup:
```

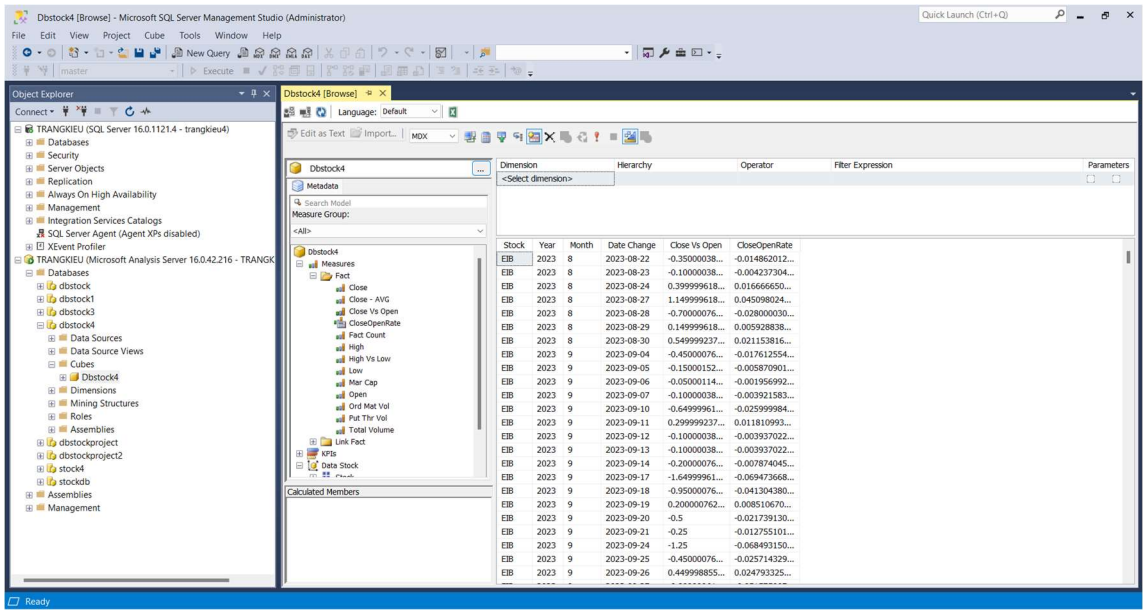
```
df11 =
dfundup.select(col("Stock"),col("DateChange"),col("Link"),col("Title"),col("Contents")
.alias("Body")) # 406 rows
# Step 5b: Filter out Dates in the scale of "DateChange" from dbstock.tbl10stocks :
## Find oldest and newest dates from data_time1:
data_time1.show()
print(data_time1.count()) # 251 unique rows
oldest_date = data_time1.agg(F.min("DateChange")).collect()[0][0]
newest_date = data_time1.agg(F.max("DateChange")).collect()[0][0]
print(f"Oldest Date: {oldest_date}") # Oldest Date: 2023-08-22
print(f"Newest Date: {newest_date}") # Newest Date: 2024-08-22

# Filter df11 "DateChange" rows that are in the scale of [oldest, newest]:
filtered_df11 = df11.filter((F.col("DateChange") >= F.lit(oldest_date)) &
(F.col("DateChange") <= F.lit(newest_date)))
# Show the result
filtered_df11.show()
print(filtered_df11.count()) # 358 rows (48 rows are eliminated from df11, which
accounts for 11.8%)
print(filtered_df11.agg(F.min("DateChange")).collect()[0][0]) # datetime.date(2023, 8,
22), matches Oldest Date
print(filtered_df11.agg(F.max("DateChange")).collect()[0][0]) # datetime.date(2024, 8,
22),  matches Newest Date
# Find mutual DateChange values between two dataframes:
mutual_dates = filtered_df11.join(data_time1, on="DateChange",
how="inner").select("DateChange").distinct()
# Show results
mutual_dates.show()
print(mutual_dates.count())  # The number of mutual dates: 71 unique values
# Filter df11 so that its "DateChange" includes only values from mutual_dates
alongwith all of other columns of df11:
df11_filtered = df11.join(mutual_dates, on="DateChange", how="inner")
## Step 5c: Create "Count of tra", "Count of phat" columns:
df_split = df11_filtered.withColumn("Contents_array", split(col("Body"), " "))
# Explode the array column `Contents_array`
df_exploded = df_split.withColumn("Content_exploded", explode(col("Contents_array")))
# Filter and count occurrences of "phát" and "trả"
linkfact = df_exploded \
    .filter(F.col("Content_exploded").like("%phát%") |
F.col("Content_exploded").like("%trả%")) \
    .groupBy("Stock", "DateChange", "Link", "Title", "Body") \
    .agg(
        F.sum(F.when(F.col("Content_exploded").like("%phát%"),
1).otherwise(0)).alias("Count_of_phat"),
        F.sum(F.when(F.col("Content_exploded").like("%trả%"),
1).otherwise(0)).alias("Count_of_tra")
    )
```

```
linkfactdup =
linkfact.select(col("Stock"),col("DateChange"),col("Link"),col("Title"),col("Body"),co
l("Count_of_phat"),col("Count_of_tra"))
linkfact1 = linkfactdup.dropDuplicates(["Stock","DateChange","Link"])
## Step 5d: Add "Link_ID" from linkdim into linkfact dataframe:
# Join linkfact with data_time based on matching dates
linkfact1_with_link_id = linkfact1.join(linkdim2, linkfact1["Link"] ==
linkdim2["Link"], "inner")
linkfact1_with_link_id.count() #  rows
# Select relevant columns:
linkfact1_with_link_id_1 = linkfact1_with_link_id.select(
    linkfact1_with_link_id["*"],  # Select all columns
    linkdim2["Link_ID"]  # Select Link_ID from linkdim2
)
linkfact2 = linkfact1_with_link_id_1.select(
    linkfact1["Stock"],linkfact1["DateChange"], \
    linkdim2["Link_ID"], \
    linkfact1["Link"], \
    linkfact1["Title"], \
    linkfact1["Body"], \
    col("Count_of_phat"),col("Count_of_tra"))
linkfact2.show()
print(linkfact2.count())
## Find rows that contain PutThrVol or OrdMatVol < 0:
neg = df_fact.filter((F.col("PutThrVol") < 0) | (F.col("OrdMatVol") < 0))
neg.show()
# III - INSERT INTO 5 SSMS TABLE IN AN ORDER: DataStock > DataTime > LinkDim > Fact >
FactLink:
# Define SQL Server database connection details
url_sql =
"jdbc:sqlserver://192.168.10.100:1433;database=dbstock4;encrypt=false;trustServerCerti
ficate=false;hostNameInCertificate=*.dat"
properties_sql = {
    "user":"trangkieu4",
     "password":"123456",
    "driver": "com.microsoft.sqlserver.jdbc.SQLServerDriver"
}
# create a loop for inserting each dimension dataframe into corresponding SQL Server
table:
tables = ['dbo.DataTime','dbo.LinkDim','dbo.Fact', 'dbo.LinkFact'] # pre-created SQL
tables in SSMS
dfs = [data_time1,linkdim2, df_fact, linkfact2]
for i in range(len(dfs)):
    try:
        dfs[i].write.mode("append").jdbc(url_sql, tables[i],
properties=properties_sql)
    except ValueError as error :
        print("Connector write failed", error)
```

```
print("Data inserted successfully into SSMS tables")
# Since the second time, inserting data excludes data from DataStock to avoid
duplicate conflicts.
spark.stop()
```

6. Creating cube database using MS Visual Studio with SSAS plugin





7. Creating Airflow tasks for automatic process

There are some notes when creating Airflow tasks:

- The orders of MongoDB and Kafka services to restart.
- Spark streaming session must be restarted in parallel with scrapy container, but scraping application runs only if PySpark shell is ready for retrieving streamed data in real time.
- Using subprocess method to run docker commands as if running these commands directly in Linux CLI.

- Creating an entrypoint (or .sh file) for the purpose of executing commands inside a bash shell of a Docker container.

This is the source codes of “all_in.py”:

```
from airflow import DAG
from airflow.providers.docker.operators.docker import DockerOperator
from airflow.operators.python import PythonOperator
from datetime import datetime, timedelta
import docker
import time
from airflow.exceptions import AirflowException
import subprocess
import os
from airflow.utils.dates import days_ago
from airflow.operators.dummy_operator import DummyOperator
from watchdog.observers import Observer
from watchdog.events import FileSystemEventHandler
import glob
from airflow.operators.bash_operator import BashOperator

# Define default arguments
default_args = {
    'owner': 'airflow',
    'depends_on_past': False,
    'retries': 1,
    'retry_delay': timedelta(minutes=5),
}

# DAG definition
with DAG(
    'all_in',
    default_args=default_args,
    description='Check if a container is healthy before restarting another container',
    schedule_interval=None,
    start_date=datetime(2024, 8, 20) ,
    catchup=False,
    concurrency=10,
) as dag:
    create_mongo_db_collection = BashOperator(
        task_id='create_mongo_db_and_collection',
        bash_command="""
        docker exec mongo1 mongosh --eval 'db = db.getSiblingDB("dbstock");
db.createCollection("tbl10stocks");'
        """,
    )
    create_mongo_db_collection1 = BashOperator(
        task_id='create_mongo_db_and_collection1',
```

```
        bash_command="""
        docker exec mongo1 mongosh --eval 'db = db.getSiblingDB("dbstock");
db.createCollection("tbcontent");'
        """,
    )
    # Function to execute postgres creation command inside Docker container
    def run_postgres_creation_func():
        result = subprocess.run(
            "docker exec postgres bash /newdbstock.sh",
            shell=True,
            stdout=subprocess.PIPE,
            stderr=subprocess.PIPE
        )

        print("STDOUT:", result.stdout.decode())
        print("STDERR:", result.stderr.decode())

        if result.returncode != 0:
            raise Exception(f"Command failed with return code {result.returncode}")
    postgres_creation = PythonOperator(
        task_id='postgres_creation',
        python_callable=run_postgres_creation_func
    )
    # Function to execute postgres rename command inside Docker container
    def postgres_rename_func():
        result = subprocess.run(
            "docker exec postgres bash /rename_db.sh",
            shell=True,
            stdout=subprocess.PIPE,
            stderr=subprocess.PIPE
        )

        print("STDOUT:", result.stdout.decode())
        print("STDERR:", result.stderr.decode())

        if result.returncode != 0:
            raise Exception(f"Command failed with return code {result.returncode}")
    postgres_rename = PythonOperator(
        task_id='postgres_rename',
        python_callable=postgres_rename_func
    )
    # Define the task: Restart the scrapy container
    def restart_schema_registry():
        import subprocess
        subprocess.call("docker restart schema-registry", shell=True)

    restart_schema_registry = PythonOperator(
        task_id='restart_schema_registry',
```

```
        python_callable=restart_schema_registry,
        dag=dag
    )
    # Function to check whether a container is running and restarting it:
    def check_and_restart_container(container_name, max_retries=8, wait_time=30):
        client = docker.from_env()
        for attempt in range(max_retries):
            container = client.containers.get(container_name)

            # Check if the container is running
            if container.status == 'running':
                print(f"Container {container_name} is running.")
                break  # Exit the loop if the container is running

            print(f"Container {container_name} is not running. Status:
{container.status}. Attempt {attempt + 1}/{max_retries}. Restarting...")
            container.restart()
            # Wait for the specified time before checking again
            time.sleep(wait_time)
        else:
            # This else block is executed if the loop finishes without hitting a
'break'
            raise ValueError(f"Container {container_name} is still not running after
{max_retries} attempts.")

        print(f"Finished checking the status of container {container_name}.")
    restartadminer = PythonOperator(
        task_id='restartadminer',
        python_callable=check_and_restart_container,
        op_args=['adminer']
    )
    restartpostgres = PythonOperator(
        task_id='restartpostgres',
        python_callable=check_and_restart_container,
        op_args=['postgres']
    )
    restartmongo1 = PythonOperator(
        task_id='restartmongo1',
        python_callable=check_and_restart_container,
        op_args=['mongo1']
    )
    restart_ubuntu = PythonOperator(
        task_id='restart_ubuntu',
        python_callable=check_and_restart_container,
        op_args=['ubuntu2204']
    )
    restartzookeeper = PythonOperator(
        task_id='restartzookeeper',
```

```
        python_callable=check_and_restart_container,
        op_args=['zookeeper']
    )
    restart_broker = PythonOperator(
        task_id='restart_broker',
        python_callable=check_and_restart_container,
        op_args=['broker']
    )

    # Function to retry restarting when health status checking is not healthy:
    def check_container_health_status(container_name, max_retries=8, wait_time=30):
        client = docker.from_env()
        for attempt in range(max_retries):
            container = client.containers.get(container_name)

            # Check if the health status exists
            if 'Health' in container.attrs['State']:
                container_status = container.attrs['State']['Health']['Status']
            else:
                #RAISE AIRFLOWEXCEPTION(F"CONTAINER '{CONTAINER_NAME}' DOES NOT HAVE A
HEALTH CHECK CONFIGURED.")
                print(f"Container '{container_name}' does not have a health check
configured.")
                break
            if container_status == 'healthy':
                print(f"Container {container_name} is healthy.")
                break  # Exit the loop if the container is healthy

            print(f"Container {container_name} is not healthy. Status:
{container_status}. Attempt {attempt + 1}/{max_retries}. Restarting...")
            container.restart()
            # Wait for the specified time before checking again
            time.sleep(wait_time)
        else:
            # This else block is executed if the loop finishes without hitting a
'break'
            raise ValueError(f"Container {container_name} is still not healthy after
{max_retries} attempts.")

        # Additional code can go here if needed after the loop is complete
        print(f"Finished checking the health status of container {container_name}.")

    # Task to check the health status of the 'broker' container
    check_broker_health = PythonOperator(
        task_id='check_broker_health',
        python_callable=check_container_health_status,
        op_args=['broker'],  # Replace 'broker' with your container's name
    )
```

```
    # Task to restart the 'rest-proxy' container
    restart_rest_proxy = PythonOperator(
            task_id='restart_rest_proxy',
            python_callable=check_and_restart_container,
            op_args=['rest-proxy']
    )
    # Task to restart the 'connect' container
    restart_connect = PythonOperator(
            task_id='restart_connect',
            python_callable=check_and_restart_container,
            op_args=['connect']
    )
    # Task to check the health status of the 'connect' container
    check_connect_health = PythonOperator(
        task_id='check_connect_health',
        python_callable=check_container_health_status,
        op_args=['connect'],  # Replace 'broker' with your container's name
    )

    # Function to check if a container is running
    def check_5container_status():
        client = docker.from_env()
        names=['zookeeper','broker','rest-proxy','connect','mongo1']
        for name in names:
            container = client.containers.get(name)
            if container.status.lower() == 'running':
                return True
            else:
                raise ValueError(f"Container {container_name} is not running. Current
status: {container.status}")
                break
    # Define tasks for each container
    check_5containers = PythonOperator(
        task_id='check_5containers',
        python_callable=check_5container_status,
    )
    # Define a function to check Kafka topics
    def check_kafka_topics():
        # Execute the command to open bash and run the status command
        result = subprocess.run(
            ["docker", "exec", "mongo1", "bash", "-c", "status"],
            capture_output=True,
            text=True
        )
        output = result.stdout

        # Check if "dbstock.tbl10stocks" is in the output
        if "topic\": \"dbstock.tbl10stocks\"" in output:
```

```
            print("Topic 'dbstock.tbl10stocks' found!")
            return True
        else:
            print("Topic 'dbstock.tbl10stocks' not found.")
            return False
    # Task to check Kafka topics
    check_kafka_topics = PythonOperator(
        task_id='check_kafka_topics',
        python_callable=check_kafka_topics,
        dag=dag,
    )
    # Task to check and restart the Spark container
    check_spark_running = PythonOperator(
        task_id='check_spark_running',
        python_callable=check_and_restart_container,
        op_args=['spark'],
        dag=dag,
        task_concurrency=3
    )
    # Function to run the entrypoint
    def run_entrypoint_func():
        result = subprocess.run(
            "docker exec spark bash entrypoint.sh",
            shell=True,
            stdout=subprocess.PIPE,
            stderr=subprocess.PIPE
        )

        print("STDOUT:", result.stdout.decode())
        print("STDERR:", result.stderr.decode())

        if result.returncode != 0:
            raise Exception(f"Command failed with return code {result.returncode}")

    # Task to start pyspark shell and execute python script
    run_entrypoint_task = PythonOperator(
        task_id='run_entrypoint_task',
        python_callable=run_entrypoint_func,
        dag=dag,
    )
    # Define function to check log file to know if the pyspark is ready:
    def check_log_file(**kwargs):
        dag_id = kwargs['dag'].dag_id
        task_id = 'run_entrypoint_task'
        execution_date = kwargs['execution_date']

        log_dir = f'/home/osboxes/airflow/airflowvenv/logs/dag_id={dag_id}'
        log_pattern = os.path.join(log_dir, '**', 'attempt=*.log')
```

```
        # Define the string to search for in the log files
        search_string = f"INFO - Running <TaskInstance: {dag_id}.{task_id}"

        max_wait_time = 600
        check_interval = 10  # Interval between checks in seconds
        start_time = time.time()

        while time.time() - start_time < max_wait_time:
            # Use glob to search for log files matching the pattern
            log_files = glob.glob(log_pattern, recursive=True)

            for log_file in log_files:
                try:
                    with open(log_file, 'r') as file:
                        log_contents = file.read()
                        if search_string in log_contents:
                            print(f"Found the string in {log_file}!")
                            return
                except IOError as e:
                    print(f"Error reading {log_file}: {e}")

            print(f"Log file matching pattern {log_pattern} not found or string not
found. Waiting...")
            time.sleep(check_interval)

        raise Exception(f"Log file matching pattern {log_pattern} not found or string
not found after waiting for {max_wait_time} seconds.")
    def delete_previous_log_files(task_id_to_delete, **kwargs):
        dag_id = kwargs['dag'].dag_id
        execution_date = kwargs['execution_date']

        log_dir = f'/home/osboxes/airflow/airflowvenv/logs/dag_id={dag_id}'

        # Define a pattern to match logs for the task to delete
        log_pattern = os.path.join(log_dir, f'{task_id_to_delete}', '**',
'attempt=*.log')

        # Use glob to find all log files matching the pattern for this task
        log_files = glob.glob(log_pattern, recursive=True)

        for log_file in log_files:
            try:
                # Check if the log file belongs to a previous execution
                log_timestamp = os.path.basename(os.path.dirname(log_file))  # Extract
timestamp from log file path
                if log_timestamp < execution_date.strftime('%Y%m%d%H%M%S'):  # Compare
with current execution time
```

```
                    os.remove(log_file)
                    print(f"Deleted previous log file: {log_file}")
            except OSError as e:
                print(f"Error deleting log file {log_file}: {e}")
    # Define the delete_previous_log_files task
    delete_logs_task = PythonOperator(
        task_id='delete_logs_task',
        python_callable=delete_previous_log_files,
        op_kwargs={'task_id_to_delete': 'run_entrypoint_task'},  # Pass the task ID of
check_log_file_task
        provide_context=True,
        dag=dag,
    )
    # Task to check log and start Scrapy container
    check_log_and_start_scrapy_task = PythonOperator(
        task_id='check_log_and_start_scrapy_task',
        python_callable=check_log_file,
        provide_context=True,  # Ensures kwargs like execution_date are passed
automatically
        dag=dag,
    )
    # Task to check and restart the Scrapy container
    scrapy_running = PythonOperator(
        task_id='scrapy_running',
        python_callable=check_and_restart_container,
        op_args=['scrapy'],
        dag=dag,
    )
    # Function to start crawling
    def start_crawl_func():
        subprocess.run("sudo pkill Xvfb || true",
        shell=True,
        stdout=subprocess.PIPE,
        stderr=subprocess.PIPE
    )
        result = subprocess.run(
            "docker exec scrapy ./start_service.sh",
            shell=True,
            stdout=subprocess.PIPE,
            stderr=subprocess.PIPE
        )
        print("STDOUT:", result.stdout.decode())
        print("STDERR:", result.stderr.decode())

    # Create the crawl task
    start_crawl = PythonOperator(
        task_id='start_crawl',
        python_callable=start_crawl_func,
```

```
        dag=dag,
    )
    # Dummy start and end tasks
    start = DummyOperator(
        task_id='start',
        dag=dag,
    )

    end = DummyOperator(
        task_id='end',
        dag=dag,
    )
    stop_xvfb_processes = BashOperator(
        task_id='stop_xvfb_processes',
        bash_command='sudo pkill Xvfb || true',
        dag=dag,
    )
    stop_spark = BashOperator(
        task_id='stop_spark',
        bash_command='docker stop spark',
        dag=dag,
    )
    stop_scrapy = BashOperator(
        task_id='stop_scrapy',
        bash_command='docker stop scrapy',
        dag=dag,
    )
    restart_scrapy = PythonOperator(
        task_id='restartscrapy',
        python_callable=check_and_restart_container,
        op_args=['scrapy']
    )
    restart_spark = PythonOperator(
        task_id='restart_spark',
        python_callable=check_and_restart_container,
        op_args=['spark'],
        dag=dag,
        task_concurrency=3
    )
    check_spark_running1 = PythonOperator(
        task_id='check_spark_running1',
        python_callable=check_and_restart_container,
        op_args=['spark'],
        dag=dag,
        task_concurrency=3
    )
    def run_entrypoint_func1():
        result = subprocess.run(
```

```
            "docker exec spark bash entrypoint_content.sh",
            shell=True,
            stdout=subprocess.PIPE,
            stderr=subprocess.PIPE
        )

        print("STDOUT:", result.stdout.decode())
        print("STDERR:", result.stderr.decode())

        if result.returncode != 0:
            raise Exception(f"Command failed with return code {result.returncode}")

    # Task to start pyspark shell and execute python script again:
    run_entrypoint_task1 = PythonOperator(
        task_id='run_entrypoint_task1',
        python_callable=run_entrypoint_func1,
        dag=dag,
    )
    # Task to check log and start Scrapy container again:
    check_log_and_start_scrapy_task1 = PythonOperator(
        task_id='check_log_and_start_scrapy_task1',
        python_callable=check_log_file,
        provide_context=True,  # Ensures kwargs like execution_date are passed
automatically
        dag=dag,
    )
        # Task to check and restart the Scrapy container
    scrapy_running1 = PythonOperator(
        task_id='scrapy_running1',
        python_callable=check_and_restart_container,
        op_args=['scrapy'],
        dag=dag,
    )
    def start_crawl_func1():
        subprocess.run("sudo pkill Xvfb || true",
        shell=True,
        stdout=subprocess.PIPE,
        stderr=subprocess.PIPE
    )
        result = subprocess.run(
            "docker exec scrapy ./startpubver3.sh",
            shell=True,
            stdout=subprocess.PIPE,
            stderr=subprocess.PIPE
        )
        print("STDOUT:", result.stdout.decode())
        print("STDERR:", result.stderr.decode())
    start_crawl1 =  PythonOperator(
```

```
        task_id='start_crawl1',
        python_callable=start_crawl_func1,
        dag=dag,
    )
    stop_xvfb_processes1 = BashOperator(
        task_id='stop_xvfb_processes1',
        bash_command='sudo pkill Xvfb || true',
        dag=dag,
    )
    stop_spark1 = BashOperator(
        task_id='stop_spark1',
        bash_command='docker stop spark',
        dag=dag,
    )
    def check_kafka_topics1():
        # Execute the command to open bash and run the status command
        result = subprocess.run(
            ["docker", "exec", "mongo1", "bash", "-c", "status"],
            capture_output=True,
            text=True
        )
        output = result.stdout

        # Check if "dbstock.tbl10stocks" is in the output
        if "topic\": \"dbstock.tbcontent\"" in output:
            print("Topic 'dbstock.tbcontent' found!")
            return True
        else:
            print("Topic 'dbstock.tbcontent' not found.")
            return False
    # Task to check Kafka topics
    check_kafka_topics1 = PythonOperator(
        task_id='check_kafka_topics1',
        python_callable=check_kafka_topics1,
        dag=dag,
    )
    restart_spark1 = PythonOperator(
        task_id='restart_spark1',
        python_callable=check_and_restart_container,
        op_args=['spark'],
        dag=dag,
        task_concurrency=3
    )
    def run_entrypoint_func2():
        result = subprocess.run(
            "docker exec spark bash post2ssms.sh",
            shell=True,
            stdout=subprocess.PIPE,
```

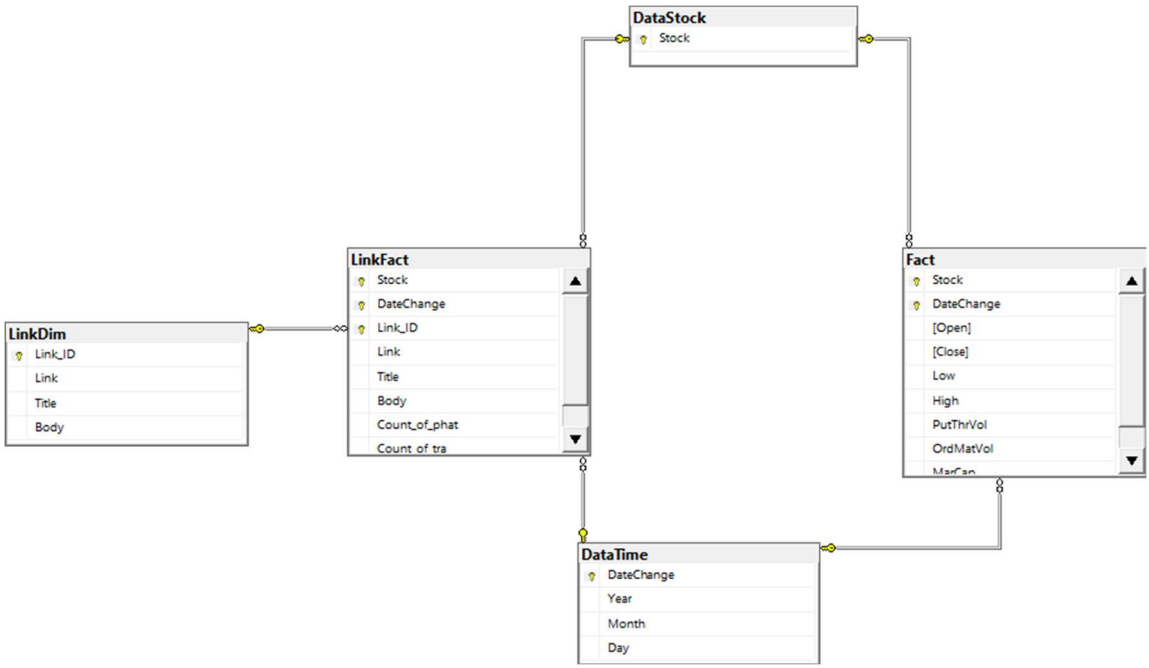
```
            stderr=subprocess.PIPE
        )

        print("STDOUT:", result.stdout.decode())
        print("STDERR:", result.stderr.decode())

        if result.returncode != 0:
            raise Exception(f"Command failed with return code {result.returncode}")

    # Task to start pyspark shell and execute python script again:
    run_entrypoint_task2 = PythonOperator(
        task_id='run_entrypoint_task2',
        python_callable=run_entrypoint_func2,
        dag=dag,
    )
    def sql_creation_func():
        result = subprocess.run(
            "docker exec ubuntu2204 bash /entrypoint.sh",
            shell=True,
            stdout=subprocess.PIPE,
            stderr=subprocess.PIPE
        )

        print("STDOUT:", result.stdout.decode())
        print("STDERR:", result.stderr.decode())

        if result.returncode != 0:
            raise Exception(f"Command failed with return code {result.returncode}")
    # Run rename SQL script inside the running container
    sql_creation = PythonOperator(
        task_id='sql_creation',
        python_callable=sql_creation_func
    )
    def run_rename_mongo_func():
        result = subprocess.run(
            "docker exec mongo1 bash /scratch_space/rename_mongo_db.sh",
            shell=True,
            stdout=subprocess.PIPE,
            stderr=subprocess.PIPE
        )

        print("STDOUT:", result.stdout.decode())
        print("STDERR:", result.stderr.decode())

        if result.returncode != 0:
            raise Exception(f"Command failed with return code {result.returncode}")
    # Task to run the script inside the MongoDB container
    rename_mongo = PythonOperator(
```

```
        task_id='rename_mongo',
        python_callable=run_rename_mongo_func,
        dag=dag,
    )
    # Define the task to stop airflows
    # Task to stop the Airflow Scheduler
    stop_scheduler = BashOperator(
        task_id='stop_airflow_scheduler',
        bash_command='pkill -f "airflow scheduler"',
        dag=dag
    )

    # Task to stop the Airflow Web Server
    stop_webserver = BashOperator(
        task_id='stop_airflow_webserver',
        bash_command='pkill -f "airflow webserver"',
        dag=dag
    )
    # Define the task to stop Docker containers
    stop_docker_containers = BashOperator(
        task_id='stop_all_containers',
        bash_command='docker stop $(docker ps -q)',
        dag=dag,
    )

    # Set up task dependencies
    restart_ubuntu >>  postgres_creation >> restartmongo1 >> restartpostgres >> \
    restartadminer >> restartzookeeper >> restart_broker >> check_broker_health >> \
    restart_schema_registry >> restart_rest_proxy >> restart_connect >> \
    check_connect_health >> check_5containers >> check_kafka_topics >> start
    start >> \
    check_spark_running >> [run_entrypoint_task, check_log_and_start_scrapy_task]
    check_log_and_start_scrapy_task >> scrapy_running >> start_crawl >> \
    stop_xvfb_processes >> stop_spark >> stop_scrapy >> restart_scrapy >> \
    restart_spark >> check_kafka_topics1 >> delete_logs_task >> check_spark_running1 \
    >> [run_entrypoint_task1, check_log_and_start_scrapy_task1]
    check_log_and_start_scrapy_task1 >> scrapy_running1 >> start_crawl1 >> \
    stop_xvfb_processes1 >> stop_spark1 >> sql_creation >> restart_spark1 \
    >> run_entrypoint_task2 >> \
    rename_mongo >> create_mongo_db_collection >> create_mongo_db_collection1 >> \
    postgres_rename >> stop_docker_containers >> stop_scheduler >> stop_webserver >> \
    end
```

IV. FINAL PRODUCTS

1. A galaxy schema data warehouse

Numerous data can tell the relationship between stock price and volume while content data can prove the close relation between the price and news published in the transferred day as well as an aggregation of total numbers of positive and negative words from the news contents. As a result, we can know which stock that giving its stakeholders good benefits. Thus, I create a galaxy schema with two fact tables named “Fact” and “LinkFact”.



Using “Select … from…” queries to check if data is inserted successfully into data warehouse

109 %

Results | Messages

	Stock
1	EIB
2	MBB
3	MBS
4	MWG

	Stock	DateChange	Open	Close	Low	High	PutThrVol	OrdMatVol	MarCap
1	EIB	2023-08-22	23.8999996185303	23.5499992370605	23.2999992370605	23.8999996185303	9096400	3603900	34743665
2	EIB	2023-08-23	23.7000007629395	23.6000003814697	23.3999996185303	23.7000007629395	5000000	3579500	34817431
3	EIB	2023-08-24	23.6000003814697	24	23.6000003814697	24.25	27390000	6290200	35407557
4	EIB	2023-08-27	24.3500003814697	25.5	24.1000003814697	25.5	25749000	10314700	37620529
5	EIB	2023-08-28	25.7000007629395	25	24.9500007629395	25.7000007629395	2747656	7410300	36882871

	DateChange	Year	Month	Day
1	2023-08-22	2023	8	22
2	2023-08-23	2023	8	23
3	2023-08-24	2023	8	24
4	2023-08-27	2023	8	27
5	2023-08-28	2023	8	28

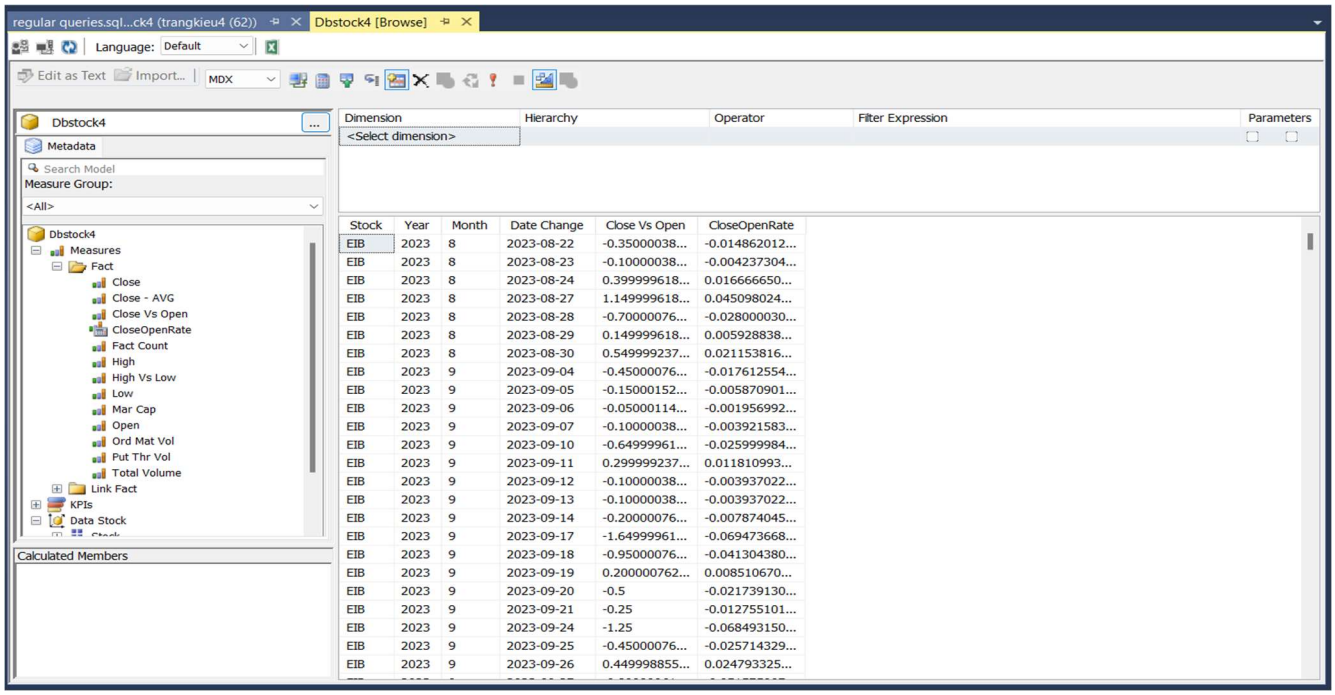
	(No column name)
1	2510

	Link_ID	Link	Title	Body
1	00d35a4b-2525-484e-8954-9f9bba9429f2	https://vietstock.vn/2023/06/vib-sap-phat-hanh-7...	VIB sắp phát hành 7.6 triệu cp ESOP	["Trang chủ Doanh nghiệp Tăng vốn - M&A","VIB sắp...
2	013d92d6-fabb-4880-9281-387c92258483	https://vietstock.vn/2024/08/tcb-nghi-quyet-hdqt-...	TCB: Nghị quyết HĐQT về việc triển khai phương ...	["Trang chủ Doanh nghiệp Hoạt động kinh doanh","T...
3	02549cf1-872a-4ba7-9f34-015682f0365a	https://vietstock.vn/2024/01/ong-pham-nhat-vuo...	Ông Phạm Nhật Vượng chuyển sang làm Giám đ...	["Trang chủ Doanh nghiệp Nhân vật","Ông Phạm Nh...
4	060b6613-91ed-47f4-99db-4afba418dc9f	https://vietstock.vn/2023/11/vndirect-mien-nhiem...	VNDirect miễn nhiệm Giám đốc Tài chính	["Trang chủ Doanh nghiệp Nhân vật","VNDirect miễn...
5	06ec69ac-52f5-4506-be6a-499d35d8c07a	https://vietstock.vn/2024/03/ors-bao-cao-ket-qua...	ORS: Báo cáo kết quả giao dịch quyền mua cổ ph...	["Trang chủ Chứng khoán Giao dịch nội bộ","ORS: B...

	Stock	DateChange	Link_ID	Link	Title	Body	Count_o
1	EIB	2023-08-22	fbfcd90b-1044-4c2f-90f5-371e12ca507e	https://vietstock.vn/2023/08/eximbank-duoc-nhnn-...	Eximbank được NHNN chấp thuận tăng vốn điều lệ t...	["Trang chủ Doanh nghiệp Tăng vốn - M&A","Eximba...	5
2	EIB	2023-09-13	2b70b57d-a628-4921-ab7f-e3b59cd0b502	https://vietstock.vn/2023/09/eib-chot-quyen-phat-h...	EIB chốt quyền phát hành cổ phiếu trả cổ tức, tỷ lệ 18%	["Trang chủ Doanh nghiệp Cổ tức","EIB chốt quyền ph...	3
3	EIB	2023-09-13	2c463286-b430-4df2-b786-a1e3475c0cc1	https://vietstock.vn/2023/09/eib-thong-bao-ngay-dk...	EIB: Thông báo ngày ĐKCC để phát hành cổ phiếu đ...	["Trang chủ Doanh nghiệp Cổ tức","EIB: Thông báo n...	3
4	EIB	2023-10-10	decbe1b8-5a17-4fd0-a4f8-de86d5b90f92	https://vietstock.vn/2023/10/eib-cbtt-nhan-duoc-co...	EIB: CBTT nhận được công văn của UBCK về báo cáo...	["Trang chủ Doanh nghiệp Cổ tức","EIB: CBTT nhận đ...	3
5	EIB	2024-08-12	dce6320e-1d09-48cf-9f34-c5f000963867	https://vietstock.vn/2024/08/gelex-tro-thanh-co-don...	GELEX trở thành cổ đông lớn của Eximbank	["Trang chủ Doanh nghiệp Hoạt động kinh doanh","G...	2

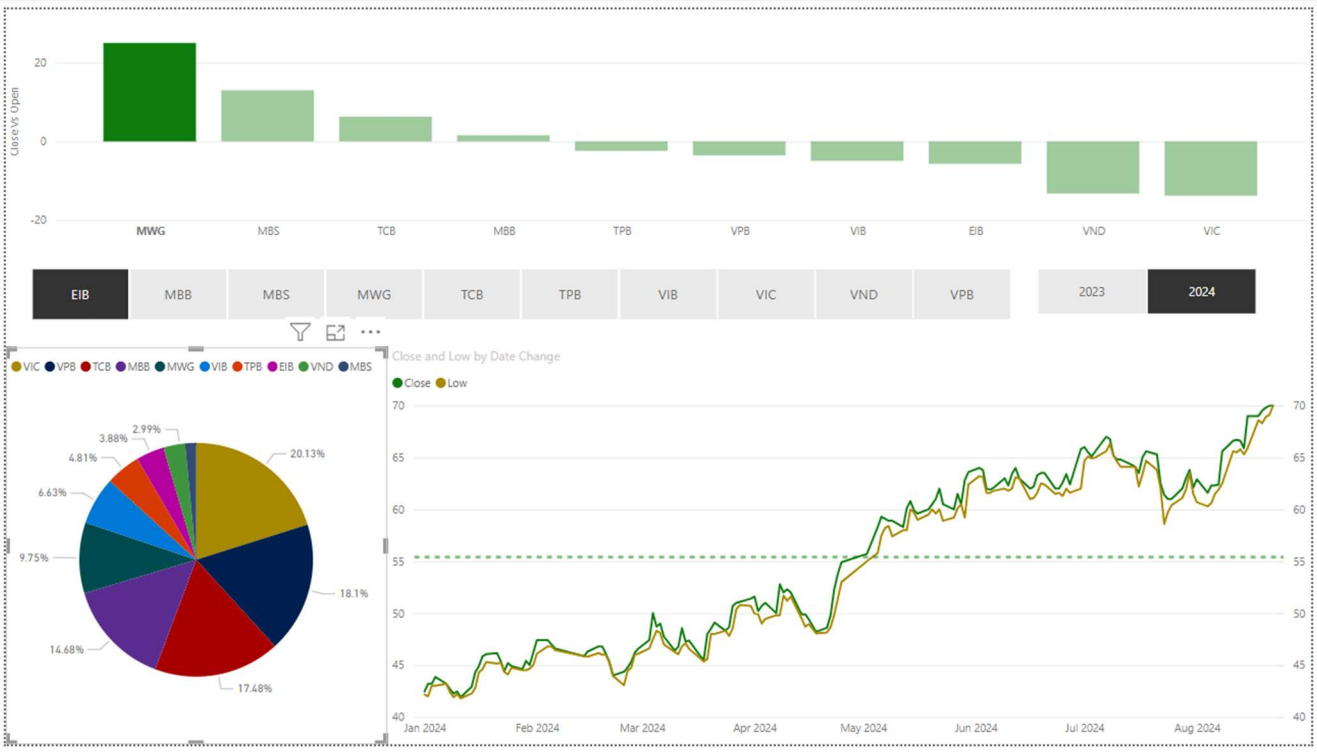
Query executed successfully. | TRANGKIEU (16.0 RTM) | trangkieu4 (62) | dbstock4 | 00:00:01 | 3,171 rows

Then, we can turn to the next step to create a cube database:



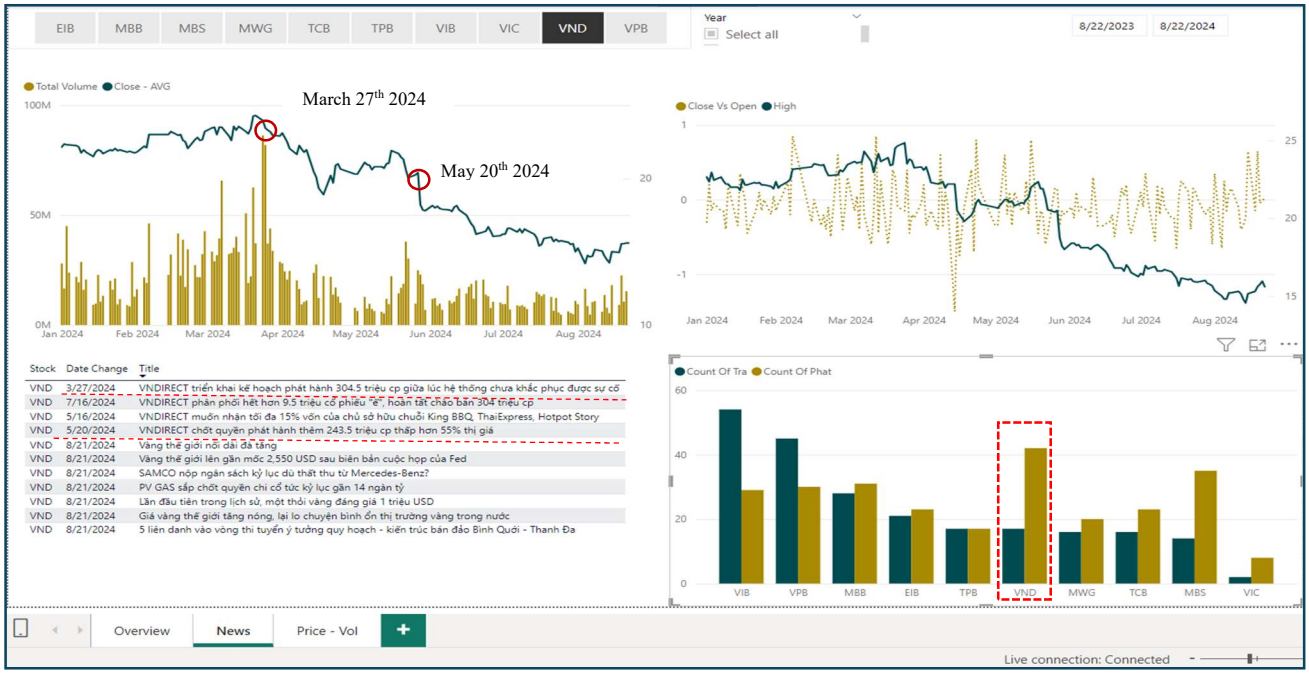
2. Dashboards and analysis

2.1. Overview of 10 stocks



From the pie chart, we can see the relative size of the market capitalization of the stocks. VIC has the largest proportion among these stocks while VPB, TCB stands at the second and third places respectively. Although MWG holds the fifth position with the moderate rate of under 10%, its price is in an upward trend. Meanwhile, VIC price shows a downward trend.

2.2. Relationship between news and stock price



Stock	Date Change	Title
VND	3/27/2024	VNDIRECT triển khai kế hoạch phát hành 304.5 triệu cp giữa lúc hệ thống chưa khắc phục được sự cố
VND	7/16/2024	VNDIRECT phân phối hết hơn 9.5 triệu cổ phiếu "e", hoàn tất chào bán 304 triệu cp
VND	5/16/2024	VNDIRECT muốn nhận tối đa 15% vốn của chủ sở hữu chuỗi King BBQ, ThaiExpress, Hotpot Story
VND	5/20/2024	VNDIRECT chốt quyền phát hành thêm 243.5 triệu cp thấp hơn 55% thị giá
VND	8/21/2024	Vàng thế giới nối dài đà tăng
VND	8/21/2024	Vàng thế giới lên gần mốc 2,550 USD sau biên bản cuộc họp của Fed
VND	8/21/2024	SAMCO nộp ngân sách kỷ lục dù thất thu từ Mercedes-Benz?
VND	8/21/2024	PV GAS sắp chốt quyền chi cổ tức kỷ lục gần 14 ngàn tỷ
VND	8/21/2024	Lần đầu tiên trong lịch sử, một thỏi vàng đáng giá 1 triệu USD
VND	8/21/2024	Giá vàng thế giới tăng nóng, lại lo chuyện bình ổn thị trường vàng trong nước
VND	8/21/2024	5 liên danh vào vòng thi tuyển ý tưởng quy hoạch - kiến trúc bán đảo Bình Quới - Thanh Đa

As shown in the column-line combo chart on the top left corner of the dashboard, VND announced the implementation of the plan to issue additional shares on March 27th 2024 which is a negative news to stakeholders as displayed in the Title column of the table under the combo chart. Therefore, the price had started to decrease gradually since then. The same trend repeated on May 20th 2024. During the observation period, VND issued more shares than other stocks in the portfolio.

2.3. Relationship between stock price and volume

When price of a stock is increasing but its total volume does not follow the same trend, it demonstrates a signs of price reversal in the near future. Look at the line-column combo chart on the top left corner of the dashboard which shows a reversal in price trend of VIC stock from the mid-March to late April in 2024. Although the price increased by 5% from 46.1 to 48.45 thousand VND during the period from March 17th to April 11th, the total volume decreased strongly from nearly 15.9 million stocks to 1/5 of the beginning of the period at only 3.1 million stocks. It can be said that there is no solid foundation for the incline in the price. As a result, the price started to drop sharply by 18% from the peak to hit the bottom at only 41 thousand VND in a very short time of 11 days since it hit the peak.

THE END