HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

GRADUATION THESIS

Vietnamese stock data collecting and processing

NGUYỄN NGỌC KHÁNH
khanh.nn204915@sis.hust.edu.vn

Major: Data Science
Specialization: Data Science

Supervisor: PhD Nguyễn Hữu Đức _____

Signature

Department: Computer Science

School: Information and Communications Technology

HANOI, 07/2024

ACKNOWLEDGMENTS

I would like to thank my family, and my friends for their constant encouragement and belief in me. Without them, I can not overcome the difficulties I faced in my journey pursuing bachelor degree.

A deep thanks to my supervisor Phd Nguyễn Hữu Đức for his dedicated guidance that leads to the completion of this thesis.

PLEDGE

Student's Full Name: Nguyễn Ngọc Khánh
Student ID: 20204915
Contact Phone Number: 0378453188
Email: khanh.nn204915@sis.hust.edu.vn
Class: CTTT Data Science AI 02-K65
Program: CTTT Khoa học dữ liệu và Trí tuệ nhân tạo 2020

I, Nguyễn Ngọc Khánh, commit that the Graduation Thesis (GT) is my own research work under the guidance of Phd Nguyễn Hữu Đức.The results presented in the GT are truthful and are my own achievements, not copied from any other works. All references in the GT, including images, tables, data, and quotations, are clearly and fully cited in the bibliography. I take full responsibility for any violations of the school's regulations concerning plagiarism.

Hanoi, 03/07/2024

Student

ABSTRACT

Vietnamese stock market is one of the most popular investments besides real estate, gold, cryptocurrency,... Each year, Vietnamese stock market attracts tremendous amount of newcomers and hundreds of thousands of billions dong are poured into the market. To help investors make better decisions therefore reducing the chance of losing money, mathematical formulas based on historic price, volume,... or so-called technical indicators have been developed by experienced traders. These technical indicators can be further used to define automatic trading strategies.

With the objectives of helping investors gain more insight into the stock market and enhance their confidence when participating in trading activities, I decide to develop a data processing system with a web interface that allows users monitoring the stock market in real-time and provide them the ability to define their own technical indicators.

Student

TABLE OF CONTENTS

CHAPTER 1. INTRODUCTION 1

1.1 Motivation 1

1.2 Objectives and scope of the graduation thesis 1

1.3 Tentative solution 1

1.4 Thesis organization 2

CHAPTER 2. REQUIREMENT SURVEY AND ANALYSIS 3

2.1 The Vietnamese stock market overview 3

2.2 Technical Indicators 3

2.2.1 On-Balance Volume (OBV) 4

2.2.2 Accumulation/Distribution Line (ADL) 4

2.2.3 Aroon 5

2.2.4 Simple Moving Average (SMA) 5

2.2.5 Exponential Moving Average (EMA) 5

2.2.6 Moving Average Convergence Divergence (MACD) 6

2.2.7 Relative Strength Index (RSI) 6

2.3 Functional requirement 7

2.3.1 General use case diagram 7

2.3.2 Monitor the Vietnamese stock market 8

2.3.3 Manage customized indicators 9

2.4 Functional description 10

2.4.1 Description of use case "Create a new indicator" 10

2.4.2 Description of use case "Monitor user defined indicators" 11

2.5 Non-functional requirement 12

CHAPTER 3. TECHNOLOGY 13

3.1 Distributed event store and stream-processing platform: Apache Kafka 13

3.1.1 Overview 13

3.1.2 Applying to the Vietnamese stock data system 14

3.2 Time series database: Apache Druid 14

3.2.1 Overview 14

3.2.2 Applying to the Vietnamese stock data system 15

3.3 Web development framework: Streamlit 16

3.3.1 Overview 16

3.3.2 Applying to the Vietnamese stock data system 16

3.4 Graphing Library: Plotly 16

3.4.1 Overview 16

3.4.2 Applying to the Vietnamese stock data system 17

3.5 Template engine: Jinja2 17

3.5.1 Overview 17

3.5.2 Applying to the Vietnamese stock data system 17

CHAPTER 4. DESIGN 18

4.1 System design 18

4.2 Indicator package design 19

4.3 Database design 22

4.4 Streamit application design 23

CHAPTER 5. IMPLEMENTATION, AND EVALUATION 24

5.1 Application building 24

5.1.1 Libraries and Tools 24

5.1.2 Achievement 25

5.1.3 Illustration of main functions 25

5.2 Testing 29
5.2.1 Test collecting data functionality 30
5.2.2 Test indicator computing functionality 31
5.2.3 Test create an existing customized indicator functionality 32
5.3 Deployment 33
CHAPTER 6. CONCLUSION AND FUTURE WORK 34
6.1 Conclusion 34
6.2 Future Work 34
REFERENCE 35
APPENDIX 37
A. Python file rendering process 37
B. Kafka streams indicator calculator implementation 39

LIST OF FIGURES

Figure 2.1 General use case diagram 7
Figure 2.2 Monitor the Vietnamese stock market 8
Figure 2.3 Manage customized indicators 9

Figure 3.1 A Simple Kafka Cluster [1] 13
Figure 3.2 Druid Storage [2] 15
Figure 3.3 Druid schema model [3] 15
Figure 3.4 Jinja rendering process 17

Figure 4.1 System Design Overview 18
Figure 4.2 Indicator package 19
Figure 4.3 IndicatorCalculator abstract class 20
Figure 4.4 SeriesType class 20
Figure 4.5 Utils class 20
Figure 4.6 MACDCalculator class 21
Figure 4.7 Database schema 22
Figure 4.8 Streamlit application design 23

Figure 5.1 Initial UI 25
Figure 5.2 Full dashboard 26
Figure 5.3 Manage your indicators page 27
Figure 5.4 Create an indicator 27
Figure 5.5 Create an indicator successfully 28
Figure 5.6 Triple EMA on dashboard 28
Figure 5.7 StockPrices.csv 29

Figure A.1 Python file rendering process 37
Figure A.2 custom-indicator.py.jinja2 38

Figure B.1 createTopology method in SMACalculator class 39

LIST OF TABLES

Table 2.1 Description of use case **Create a new indicator** 10
Table 2.2 Description of use case **Monitor user defined indicators** . . 11

Table 5.1 List of libraries and tools used 24

LIST OF ABBREVIATIONS

Abbreviation	Full Expression
API	Application Programming Interface
OBV	On-Balance Volume
ADL	Accumulation/Distribution Line
SMA	Simple Moving Average
EMA	Exponential Moving Average
MACD	Moving Average Convergence Divergence
RSI	Relative Strength Index
OHLCV	Open, High, Low, Close, Volume
OHLC	Open, High, Low, Close
TCBS	Techcom Securities
ID	Identifier
UI	User Interface

CHAPTER 1. INTRODUCTION

1.1 Motivation

After nearly 25 years of formation and development, the Vietnamese Stock market has achieved many encouraging achievements. It is becoming increasingly stable, transparent, efficient and sustainable thanks to the efforts of the government. With attractive properties such as does not require a large amount of capital to participate or high liquidity, many investors has chosen the Vietnamese stock market to become one of their own promising investments.

However, being profitable in the market is not easy, the market is dependent on many variables therefore make it so complex to capture. To cope with this problem, investors often make use of price data, trading volumes, and other technical indicators provided by a securities company in their investing activities. Moreover, to stand out in the market and maximize profits, many advanced investors want to define their own technical indicators.

If the requirements for efficient monitoring of the market in real time and customizing of technical indicators are met, it would greatly improve the decision-making process for investors, reducing the risk of financial loss and increasing confidence in their trading activities. Furthermore, the methodologies developed for the Vietnamese stock market could potentially be adapted and applied to assets other than stock or market in other country.

1.2 Objectives and scope of the graduation thesis

With the high demand for the system that aids investors in their trading process, many applications come from securities companies have appeared. Some of the most popular ones are VPS SmartOne (by VPS Securities), VNDirect (by VNDirect), SSI iboard (by SSI joint stock company). These applications provide users with unified toolkit that equips them the ability to buy or sell financial products, monitor the market with technical indicators. However, these technical indicators are fixed, if some users need to define custom indicators, they simply can't.

Knowing that fact, I decided to build a system that not only processes and visualizes stock data in real time, but also gives investors a way to define their indicators they are interested in. The system has a web interface to make user interactions more easily.

1.3 Tentative solution

To achieve the objectives stated 1.2, I proposes a solution as follows:

The system uses VNStock as the open-high-low-close-volume stock data crawler from TCBS (Techcom Securities) source. The stock data are then go to a Kafka topic. Kafka Streams is then used to process data in exactly-once semantics and real-time manner. The output of Kafka Streams are technical indicators that are also stored in their own Kafka topics. Apache Druid, which has exactly once semantics guarantee ingestion from Kafka and efficient time-series query, ingests the data from all the Kafka topics and serves as the back-end of a web application interface. Streamlit is a fast and easy way to develop a web application. Plotly provides interactive plots, jinja2 helps with code generation process, and streamlit-code-editor is integrated in the Streamlit app to help user define indicators more efficiently.

The solution results in a system with functionalities that visualize Vietnamese stock data in real time and help users manage their own indicators with basic operations such as adding a new indicator, modifying or deleting an existing indicator.

1.4 Thesis organization

The remainder of this thesis is organized as follows:

Chapter 2 Requirement Survey And Analysis presents the high-level overview of the Vietnamese stock market, existing stock data processing systems, and requirements of the system.

Chapter 3 Technology introduces important technologies that make up the system: Apache Kafka, Apache Druid, Streamlit, Plotly and Jinja2. This chapter summarizes the major points of technologies and how to apply them to build the system.

Chapter 4 Design describes the overall design of the system, database design, how the applications in server and end-user device are designed.

Chapter 5 Implementation, and Evaluation lists the libraries and tools used to build the system.The final appearance of the end user application, tesing and deployment processes are also contained in Chapter 5.

Chapter 6 Conclusion and Future Work restates the achievement of the thesis and proposes several directions to enhance the system.

CHAPTER 2. REQUIREMENT SURVEY AND ANALYSIS

This chapter first introduces some terminologies relating to Vietnamese stock market. The second section covers the formulas for seven useful technical indicators. The chapter ends with requirements for the system.

2.1 The Vietnamese stock market overview

Securities are assets including stocks, fund certificates, derivative securities, bonds, warrants, covered warrants, share purchase rights, and depository certificates. What these types of assets have in common is a proof of legal ownership of the owner (collectively referred to as the investor) with the assets of the business or issuing organization.

Stock is a type of security that confirms ownership of shares by the issuing organization.

Shareholders are stock holders.

The stock market is where offering, listing, trading, investment, information disclosure, public company management... The stock market operates according to the principle of respecting ownership rights, freedom of transactions and openness and transparency, investors are responsible for their own risks.

Technical analysis looks at the price movement and trading volume of a security and uses this data to predict future price movements.

Support level is a price level below which the price of an security, such as a stock, is unlikely to fall further.

Resistance level is a price level above which the price of an security, such as a stock, is unlikely to rise further.

Price Range is the difference between the highest price and the lowest price in an period

2.2 Technical Indicators

Technical indicator is a mathematical calculation based on historic price, volume of a security used by traders who follow technical analysis.

The mathematical formula in this section follows these notations:

O_i : open price of period i

H_i : the highest price of period i

L_i : the lowest price of period i

C_i : close price of period i

V_i : trading volume of period i

X_i : this can be any of the five O_i, H_i, L_i, C_i, V_i

$indicator_i$: indicator value of period i

N : number of periods

2.2.1 On-Balance Volume (OBV)

$$OBV_i = \begin{cases} OBV_{i-1} + V_i & \text{if } C_i > C_{i-1} \\ OBV_{i-1} - V_i & \text{if } C_i < C_{i-1} \\ OBV_{i-1} & \text{if } C_i = C_{i-1} \end{cases} \tag{2.1}$$

OBV is an indicator that measures the positive and negative volume flow of a stock over time. OBV suggests that when it increases or decreases dramatically without strong price volatility, the price goes up or down considerably at some point in the future.

2.2.2 Accumulation/Distribution Line (ADL)

$$MoneyFlowMultiplier_i = \frac{(C_i - L_i) - (H_i - C_i)}{H_i - L_i} = 2 * \frac{C_i - \frac{H_i + L_i}{2}}{H_i - L_i} \tag{2.2}$$

$$MoneyFlowVolume_i = MoneyFlowMultiplier_i * V_i \tag{2.3}$$

$$ADL_i = ADL_i + MoneyFlowVolume_i \tag{2.4}$$

The difference between the formula of ADL and the formula of OBV (2.1) is how the volume V_i is weighted. In this case, $MoneyFlowMultiplier_i$ gets negative value when the close price C_i ends up above the midpoint of the price range. In reverse, $MoneyFlowMultiplier_i$ gets a positive value when the close price C_i is below the midpoint of the price range.

The increase in ADL indicates an upward trend, while the decrease in ADL indicates a downward trend. Divergence is another phenomenon that traders look for. Divergence occurs when the price and the indicator go in opposite directions. The rise or fall of the ADL may be forecast the same behaviour of the price when divergence happens.

2.2.3 Aroon

$$AroonUp = 100 * \frac{N - PeriodsSinceN\_PeriodHigh}{N} \tag{2.5}$$

$$AroonDown = 100 * \frac{N - PeriodsSinceN\_PeriodLow}{N} \tag{2.6}$$

$$AroonOscillator = AroonUp - AroonDown \tag{2.7}$$

where $PeriodsSinceN\_PeriodHigh$: the number of periods since the highest price in N periods

$PeriodsSinceN\_PeriodLow$: the number of periods since the lowest price in N periods

The number of periods N is often chosen to be 25. $AroonUp$ measures the strength of the upward trend. In reverse, $AroonDown$ measures the strength of the downward trend. $AroonOscillator$, if negative, indicates an downward trend and indicates an upward trend if positive

2.2.4 Simple Moving Average (SMA)

$$SMA_i = \frac{\sum_{k=i-N+1}^{i} X_i}{N} \tag{2.8}$$

X_i is often chosen to be close price C_i

3 common types of SMA:

(i) Short-term SMA: SMA 10, SMA 14, SMA 20

(ii) Medium-term SMA: SMA 50

(iii) Long-term SMA: SMA 100, SMA 200

Simple Moving Average lines serve as support and resistance lines in technical analysis. When the price candlestick is above the simple moving average, it acts as a support zone. Conversely, when the SMA is above the price candle, it acts as a resistance zone. Depending on the size of the simple moving average (50,100,200) it will correspond to a strong support or resistance level.

2.2.5 Exponential Moving Average (EMA)

$$EMA_i = EMA_{i-1} + \alpha * (X_i - EMAi - 1) \tag{2.9}$$

where $\alpha = \frac{2}{N+1}$

X_i is often chosen to be close price C_i

Price data used in EMA is weighted, therefore it is quite sensitive to short-term fluctuations; thereby giving trading signals earlier than SMA. EMA is suitable for stocks with high market volatility. Because EMA is sensitive to stock fluctuations, it will give more timely trading signals than SMA.

2.2.6 Moving Average Convergence Divergence (MACD)

$$MACD_i = EMA[X_i, FastN]_i - EMA[X_i, SlowN]_i \tag{2.10}$$

$$MACDS_i = EMA[MACD_i, SignalN]_i \tag{2.11}$$

$$MACDH_i = MACD_i - MACDS_i \tag{2.12}$$

where $MACDS$: $MACD$ Signal

$MACDH$: $MACD$ Histogram

$FastN$, $SlowN$, $SignalN$ are often chosen to be 12, 26 and 9 respectively

X_i is often chosen to be close price C_i

The indicator comprises two lines: the MACD line and a signal line, which moves slower. When the signal line crosses above the MACD line, it indicates that the price is falling. When the signal line crosses below the MACD line, the price is rising.

MACD is often combined with a histogram that shows the difference between MACD and its signal line. If MACD is greater than its signal, the histogram will be above the MACD's baseline or zero line. If MACD is smaller than its signal, the histogram will be below the MACD's baseline. The MACD's histogram is used by traders to probe for overbought/oversold signals.

2.2.7 Relative Strength Index (RSI)

$$U_i = \begin{cases} C_i - C_{i-1} & \text{if } C_i > C_{i-1} \\ 0 & \text{otherwise} \end{cases} \tag{2.13}$$

$$D_i = \begin{cases} C_{i-1} - C_i & \text{if } C_{i-1} > C_i \\ 0 & \text{otherwise} \end{cases} \tag{2.14}$$

$$A_{U_i} = U_i + A_{U_{i-1}} * \frac{N-1}{N} \tag{2.15}$$

$$A_{D_i} = D_i + A_{D_{i-1}} * \frac{N-1}{N} \tag{2.16}$$

$$RSI_i = 100 - 100 * \frac{1}{1 + \frac{A_{U_i}}{A_{D_i}}} \tag{2.17}$$

where U_i: upward change of period i

D_i: downward change of period i

A_{U_i}: average of upward change of period i

A_{D_i}: average of downward change of period i

The number of periods N is often chosen to be 14

Traders use the RSI to identify overbought/oversold signals. When the RSI is over 70, the stock is considered overbought and could decline. When the RSI is below 30, the stock is oversold and could go up.

2.3 Functional requirement

2.3.1 General use case diagram

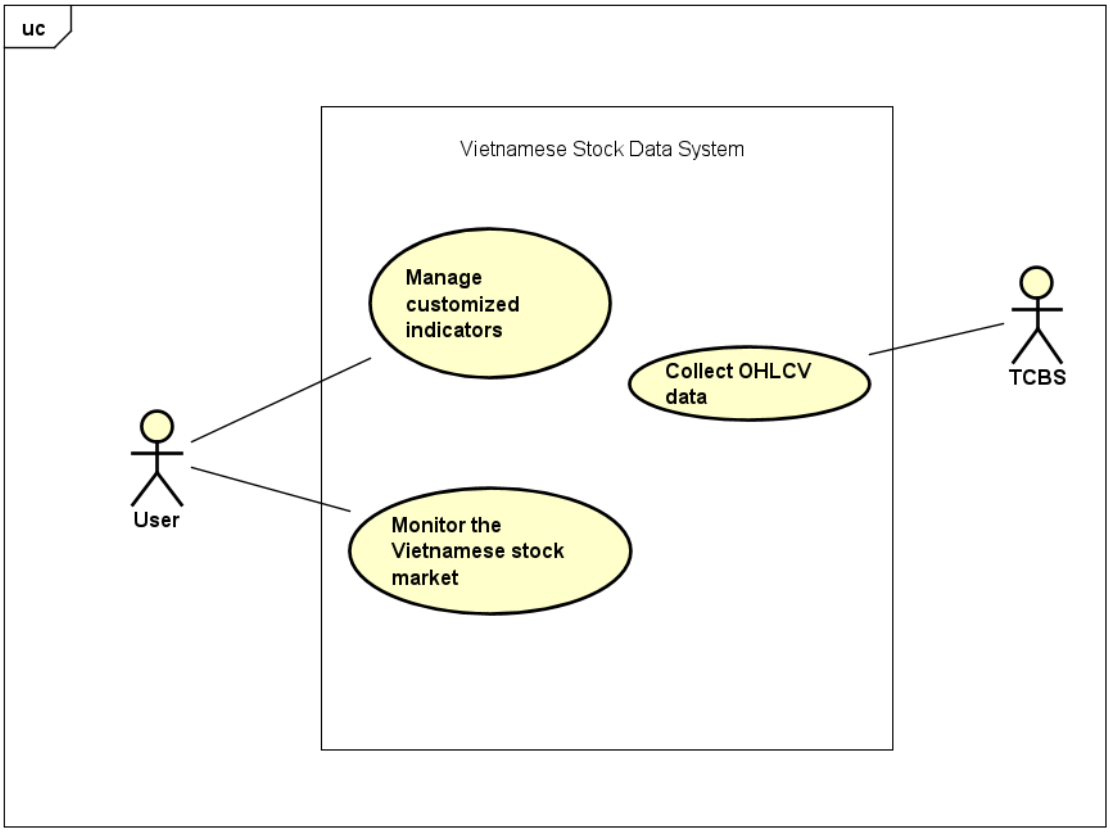

Figure 2.1: General use case diagram

The Vietnamese Stock Data System interacts with two actors:

(i) User: who uses the system and interacts with the web interface.

(ii) TCBS: the source of Vietnamese stock OHLCV data.

The Vietnamese Stock Data System have three main use cases:

(i) Manage customized indicators: allows users managing their customized technical indicators.

(ii) Monitor the Vietnamese stock market: User are able to perform real-time technical analysis by monitoring the interactive dashboard, which visualize stock OHLCV data and technical indicators.

(iii) Collect OHLCV data: crawl data from TCBS.

2.3.2 Monitor the Vietnamese stock market

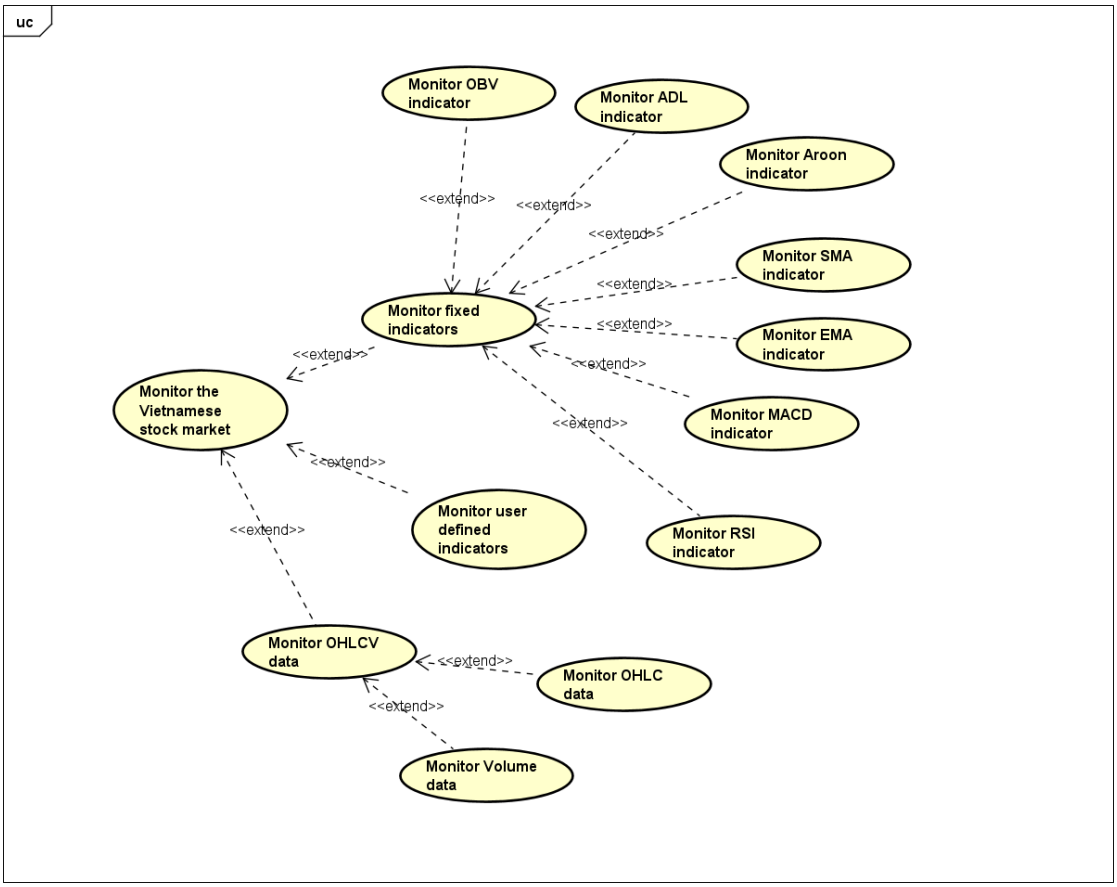

Figure 2.2: Monitor the Vietnamese stock market

(i) Monitor fixed indicators: User can monitor 7 useful technical indicators(OBV, ADL, Aroon, SMA, EMA, MACD, RSI) in the dashboard.

(ii) Monitor user defined indicators: The indicators that user defined can be visualized in the dashboard.

(iii) Monitor OHLCV data: OHLC data of a stock is visualized in form of a candlestick chart and Volume data of a stock is shown by a histogram.

2.3.3 Manage customized indicators

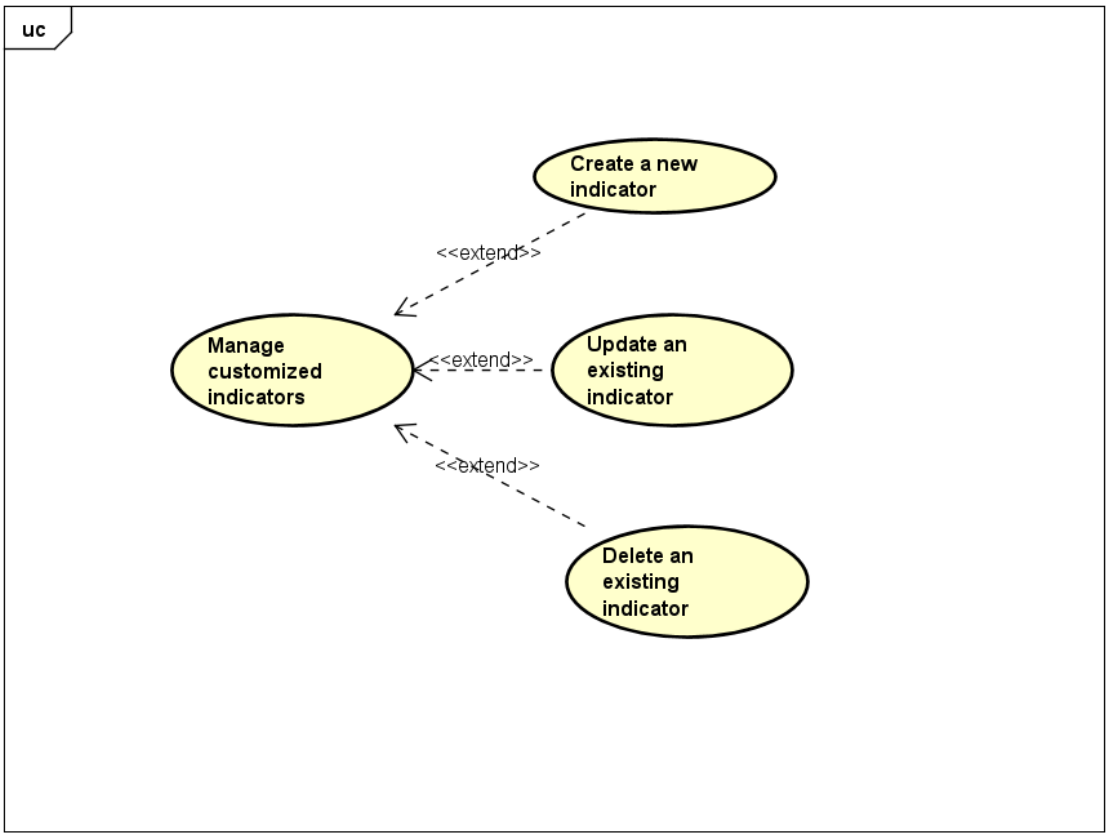

Figure 2.3: Manage customized indicators

(i) Create a new indicator: allows users creating their customized technical indicators by coding directly in the code editor, which is integrated in the web interface.

(ii) Update an existing indicator: User can update the technical indicator user defined before.

(iii) Delete an existing indicator: User can delete the technical indicator user defined before.

2.4 Functional description

2.4.1 Description of use case "Create a new indicator"

Table 2.1: Description of use case **Create a new indicator**

Use Case Name	**Create a new indicator**
ID	UC-01
Importance Level	High
Primary Actor	User
Stakeholders and Interests	User: wants to create a technical indicator of his/her interest.
Brief Description	This use case describes how an user can add a new indicator to the system.
Precondition	None
Trigger	The user selects the create option in the **Chose an operation** select box in the **Manage your indicators** page.
Postcondition	A new indicator is created in the system.
Relationships	Extend: **Manage customized indicators**
Normal Flow of Events	1. The system displays a form for entering the new indicator's information, such as name, python code, output variable. 2. The user fills in the required fields and clicks on the create button. 3. The system validates the input data and checks for any errors or conflicts. 4. The system creates a new python file which has the ability of computing the indicator defined by the user.
SubFlows	None
Alternate/Exceptional Flows	3a. The system detects an error or conflict in the input data such as a duplicate indicator name, python syntax error, output variable does not appear in the code. 3a1. The system displays an error message to the user. 3a2. The user corrects the input data and resubmits the form. 3a3. The system returns to step 3 of the normal flow.

2.4.2 Description of use case "Monitor user defined indicators"

Table 2.2: Description of use case **Monitor user defined indicators**

Use Case Name	**Monitor user defined indicators**
ID	UC-02
Importance Level	High
Primary Actor	User
Stakeholders and Interests	User: wants to monitor a technical indicator which he/she has defined.
Brief Description	This use case describes how an user can monitor a customized indicator.
Precondition	The user created at least one customized indicator before.
Trigger	The user scrolls the sidebar of the **Dashboard** page to the **Your indicators** section.
Postcondition	A line chart of a customized indicator is shown on the dashboard.
Relationships	Extend: **Monitor the Vietnamese stock market**
Normal Flow of Events	1. The system displays a control group under the **Your indicators** section. 2. The user selects an indicator in the select box and optionally changes the color of the line other than the default one. 3. The user checks on the **Show** checkbox. 4. The system creates a line chart and shows it on the dashboard.
SubFlows	None
Alternate/Exceptional Flows	None

2.5 Non-functional requirement

The system has the following non-functional requirements:

(i) Data should be collected, fully stored and without data loss.

(ii) Data should be processed in exactly-once semantics.

(iii) The latency of the system must be low to achieve real-time updating.

(iv) The system needs to respond quickly to user requests and operate stably.

(v) The web user interface must be user-friendly and intuitive.

CHAPTER 3. TECHNOLOGY

Chapter 2 has presented some concepts to have basic grasp of the Vietnamese stock market and the requirements of the system. This chapter then introduces the technologies that will be used in the system and the reasons why they are suitable.

3.1 Distributed event store and stream-processing platform: Apache Kafka

3.1.1 Overview

Apache Kafka is, in short, a Distributed Streaming Platform. Kafka is capable of storing streams of records in a fault-tolerant and durable way. Moreover, Kafka streams applications that leverage Kafka can process streams of records with tiny latency. Apache Kafka emerges as a solution to decouple the source systems and the target systems, therefore, reduces the complexity of the overall system.

Some fundamental concepts in Apache Kafka are : Message, Topic, Brokers, Producers, Consumers, Kafka Streams.

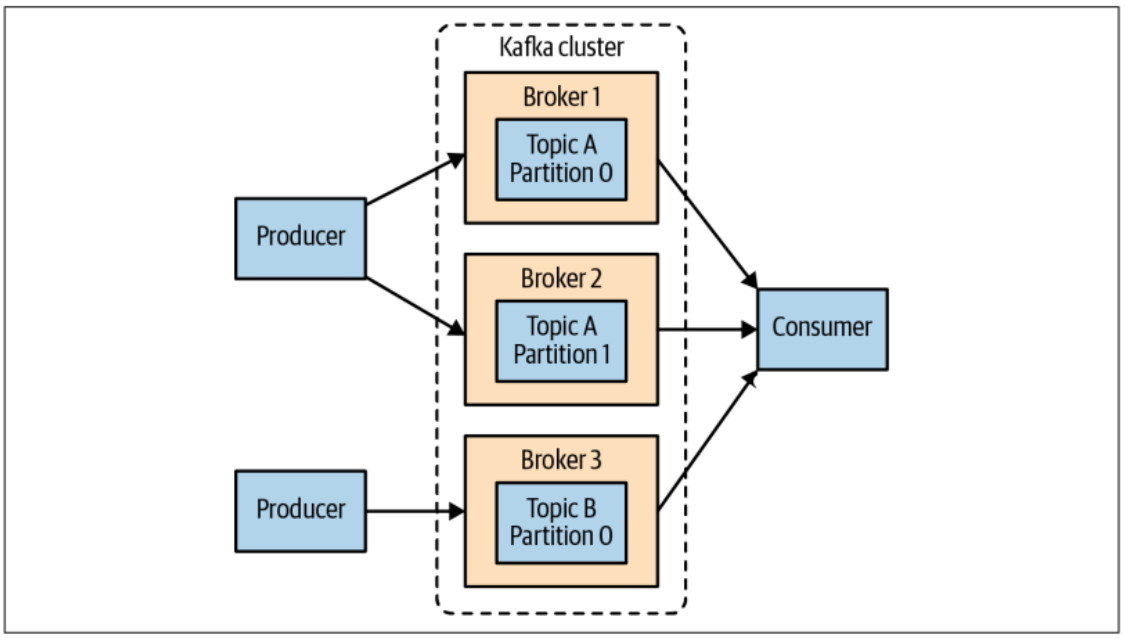

Figure 3.1: A Simple Kafka Cluster [1]

Message (or record or event) is data element in Kafka. Each message consists of a Key and a Value. The Key is used to distributed messages to cluster. The Value is the business relevant data.

Topic is a particular stream of messages. Topic are split in partitions. Each new message only goes to the end of the partition. The messages ,located in the same partition, are ordered using offset. Offset is an incremental id, it never goes back.

Brokers are machines which compose a Kafka cluster. Each broker is given an integer id and contains certain topic partitions.

Producers are which write data to topics. Based on message key, producer know where to forward the message to. Producer can be configured to be idempotent. Idempotent producer guarantee each message is only written once, and each partition is ordered by the time message produced.

Consumers are which read data from topics. Consumers can be organized as a group and the Kafka automatically balance the output stream. Consumer makes use of a special topic called _consumer_offsets to know where to continue after failures.

Kafka Streams is a library aiming at developing Kafka Streams applications. Kafka Stream application is both a consumer and a producer, moreover, it processes data with high throughput. Kafka Streams guarantees that each message is only processed once.

3.1.2 Applying to the Vietnamese stock data system

Apache Kafka is used to store stream of stock data in a fault-tolerant and durable way. If there is more data to process, Kafka helps scale the system with ease by adding more machines to Kafka cluster. If there are more applications to come, we simply configure them as the new consumers to read from existed Kafka topic.

Kafka Streams is used to develop applications to process the stock data. In comparison with other stream processing technology such as Apache Spark, Kafka Streams has the advantage of no need to deploy another cluster (we just use the existed Kafka cluster). In terms of speed, Apache Spark relies on micro-batch processing, therefore, has the higher latency than Kafka Streams which processes each message as it comes.

3.2 Time series database: Apache Druid

3.2.1 Overview

Apache Druid is a time-series, columnar, distributed database. Common use cases of Druid are real-time ingestion, fast query, and high up-time.

Druid stores data in datasources, which resemble tables in traditional relational database. By default, each datasource is time-partitioned. Each time interval is called a chunk. Each chunk is further partitioned into segments. Each segment is stored in a single file. Data in a single file is converted to columnar format, indexed with bitmap indexes and compressed. Therefore, the storage requirement for big datasource and the time to return query result are reduced. Druid uses an

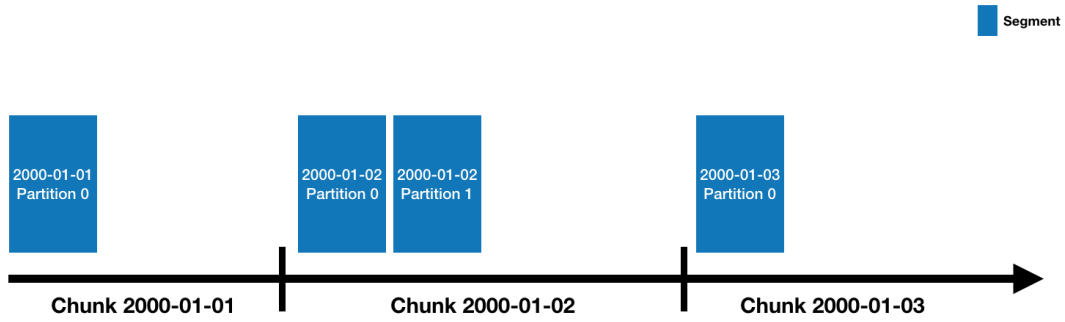

Figure 3.2: Druid Storage [2]

external deep storage to backup data. All the datasources are stored in deep storage.

Timestamp | Dimensions | Metrics

Timestamp	Page	Username	Gender	City	Characters Added	Characters Removed
2011-01-01T01:00:00Z	Justin Bieber	Boxer	Male	San Francisco	1800	25
2011-01-01T01:00:00Z	Justin Bieber	Reach	Male	Waterloo	2912	42
2011-01-01T02:00:00Z	Ke$ha	Helz	Male	Calgary	1953	17
2011-01-01T02:00:00Z	Ke$ha	Xeno	Male	Taiyuan	3194	170

Figure 3.3: Druid schema model [3]

Druid divides all columns of its schema model into three types: primary timestamp, dimensions and metrics. Primary timestamp is the type of column that always exists in Druid schemas. The primary timestamp column is used to partition and sort the datasource. The primary timestamp column is also used to optimize time involved read queries by pruning the set of segments to consider and manage datasource. Dimensions and metrics are columns with a support of a specialized operation at ingestion time named rollup. If rollup is allowed, metrics will be stored in an aggregated form and dimensions will be collapsed.

Data from Kafka topic can be ingested by Druid in real-time. The streaming ingestion process from Kafka is controlled by Druid supervisor. Each supervisor job is responsible to an indexing task, Supervisors base on the state of indexing tasks to coordinate handoffs, manage failures and guarantee that replication and scalability requirements are met.

3.2.2 Applying to the Vietnamese stock data system

The Vietnamese stock data are time-series by nature. Therefore, using a database optimized for time-series is beneficial. The compatibility of Druid and Kafka is another reason that makes it suitable for the system. Druid allows real-time querying the datasource when data is being ingested from Kafka. Moreover, Druid guaran-

tees that the data is ingested exactly-once.

3.3 Web development framework: Streamlit

3.3.1 Overview

Streamlit is an open-source framework specifically designed for creating data-driven web applications with remarkable ease and speed. Streamlit applications can be developed using only Python, without worrying about front-end technologies such as HTML, CSS, and JavaScript. This simplicity and focus on the data science community make Streamlit a powerful tool for delivering dashboards, data visualizations, and machine learning models to users.

User interaction in Streamlit is made possible by widgets (e.g., slider, button, text input). Furthermore, complex visualizations from popular plotting libraries such as Matplotlib, Plotly, and Altair are integrated in Streamlit frontend seamlessly with only some simple lines of code. If the default functionalities of Streamlit are not enough, users can use custom components using JavaScript or leverage third-party components.

3.3.2 Applying to the Vietnamese stock data system

Unlike other traditional web development frameworks, Streamlit significantly reduces development time by focusing on rapid prototyping and iterative development without the need for detailed front-end work. Streamlit also has simple Python syntax and less steep learning curve. However, as the tradeoff, Streamlit is lack of flexibility.

Streamlit is chosen to develop the system UI to save limited and valuable resources such as time. The time saving can further be invested into enhance the technical aspect of the system. To support the use case **Create customized technical indicator** introduced in 2.3.1, I rely on the third-party component streamlit-code-editor[1] to integrate a java code editor directly in the web UI.

3.4 Graphing Library: Plotly

3.4.1 Overview

Plotly is an Python open source, browser-based, interactive graphics library. The interactivity that Plotly brings makes charts more intuitive and interesting. Not only that, Plotly is fast, easy to use (thanks to detailed and well-written documentation), and combines seamlessly with other Python libraries like NumPy and Pandas. Ploly is a suitable solution for creating interactive and high-quality charts

[1] streamlit-code-editor is a code editor component for Streamlit apps, built on top of react-ace, with customizable themes and interface elements

from geographic, scientific, statistical or financial data.

3.4.2 Applying to the Vietnamese stock data system

Plotly is chosen to build an interactive dashboard, which allows user monitor the Vietnamese stock market(2.3.2). Plotly also does not have any compatible problems with Streamlit, because Streamlit has built-in support for Plotly as mentioned in 3.3.1.

3.5 Template engine: Jinja2

3.5.1 Overview

Jinja2 is a Python templating engine, which is primarily used to generate HTML from predefined templates with dynamic data. Jinja offers some powerful features, that are not existed in the built-in String Template Class in Python, such as Template Inheritance, Macros (analogous to function in usual programming), Filters and Tests.

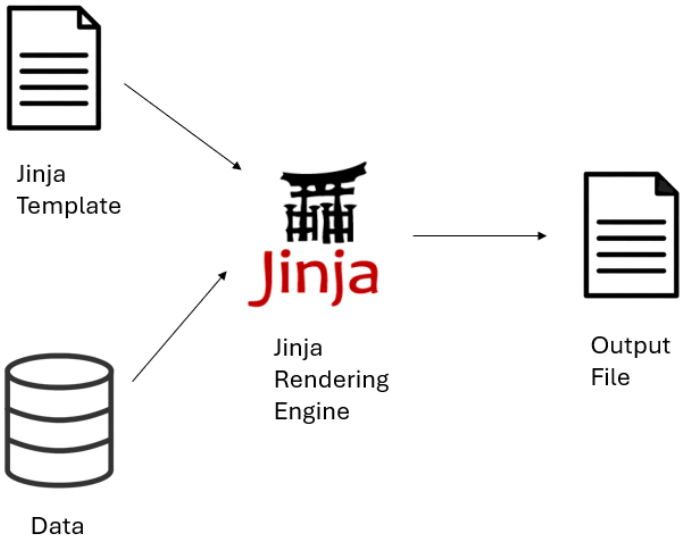

Figure 3.4: Jinja rendering process

The overview of Jinja rendering process is illustrated in Figure 3.4. Jinja Template has two parts: fixed part and placeholder part. The placeholder part defines where and how Jinja Engine puts data in Jinja Template to render the output file.

3.5.2 Applying to the Vietnamese stock data system

Jinja2 usage is to take the user python code input in the browser to render a python file, therefore, allowing user creating their own indicators (2.3.1). Although Jinja2 is mainly used to render HTML file (3.5.1), we can adapt it to render python file with minor changes.

CHAPTER 4. DESIGN

Chapter 3 has went over the technical aspects and usage of technologies made up the system. In the very beginning section of chapter 4, we will see how these technologies are organized into a system.

4.1 System design

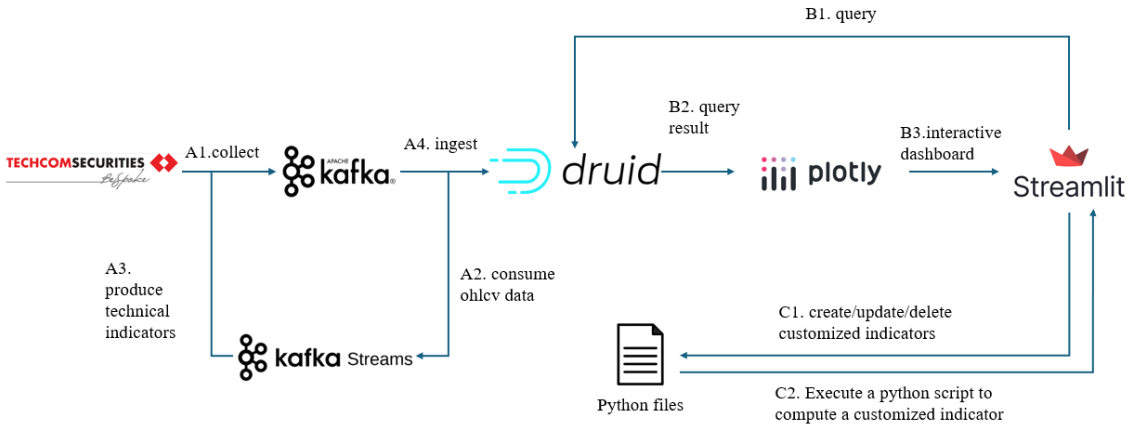

Figure 4.1: System Design Overview

Figure 4.1 presents how different components of the system are connected and three main system flows.

The collecting, computing indicators and storing flow is denoted by the letter A in the Figure 4.1. The ohlcv data of 1138 stocks are first collected from techcom securities and are produced to a topic named StockPrices in Kafka. Seven different kafka streams applications, dedicated to calculating seven fixed indicators of the system, each consumes data from StockPrices topic and produces to its own Kafka topic. Data from each topic of Kafka are ingested by a separate Druid ingestion task to store data securely in Druid.

The dashboard generating flow is denoted by the letter B in the Figure 4.1. As the user interacts with the dashboard, the streamlit front-end frequently calls the back-end api to get data from druid. The data are then visualized by plotly python package and the resulting interactive plots are shown in the dashboard.

The manage customized indicators flow is denoted by the letter C in Figure 4.1. Each time a user creates, updates, or deletes a customized indicator is actually that he/she creates, updates, or deletes a Python file. The process of producing Python file from user input ,which involves Jinja2, is decribed in details in Appendix A. When a customized indicator needs to be calculated, its corresponding Python file

is executed behind the scenes.

4.2 Indicator package design

The indicator package contains the classes that are used to compute fixed indicators. The computation of fixed indicators serves the Monitor fixed indicators use case mentioned in subsection 2.3.2

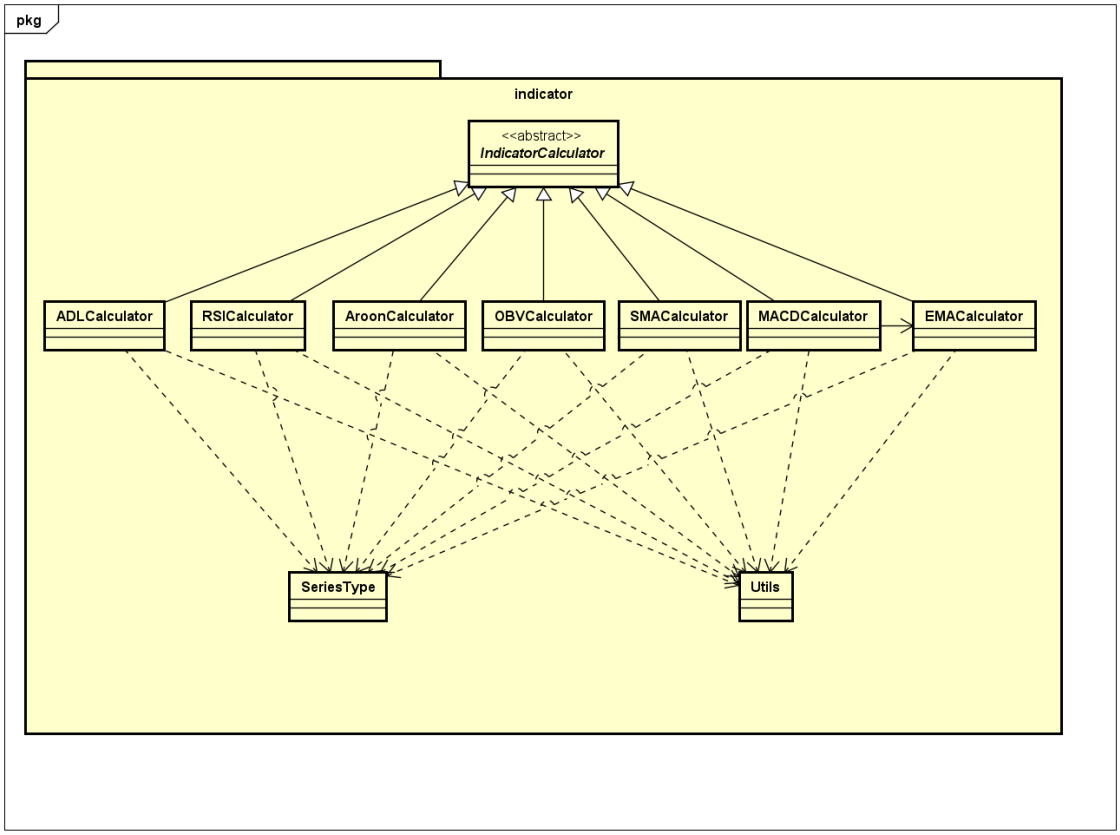

Figure 4.2: Indicator package

In Figure 4.2, there are seven classes: ADLCalculator, MACDCalculator, AroonCalculator, OBVCalculator, SMACalculator, RSICalculator, EMACalculator; all inherits an abstract class IndicatorCalculator and are designed to transform the input stream of OHLCV stock data to one output stream of these technical indicators: ADL, MACD, Aroon, OBV, SMA, RSI, EMA repectively. MACDCalculator associates to EMACalculator as infering from the formula of MACD indicator(2.2.6) which contains EMA. Utils class consists of useful functions that other classes requires. The purpose of the SeriesType class is to define a set of constants that represent different types of data series in stock market contexts. By using constants like these, the SeriesType class helps to avoid hardcoding string literals throughout the codebase, reducing the risk of errors (e.g., typos) and making the code easier to maintain. Instead of using raw strings, developers can refer to these predefined constants, ensuring consistency and improving readability.

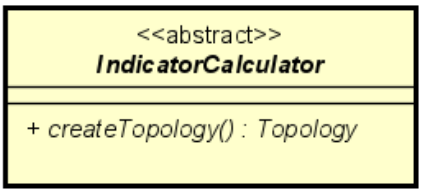

Figure 4.3: IndicatorCalculator abstract class

The detailed abtract class IndicatorCalculator in Figure 4.3 has only one abtract method that aims to create the Topology[1] of Kafka Streams application.

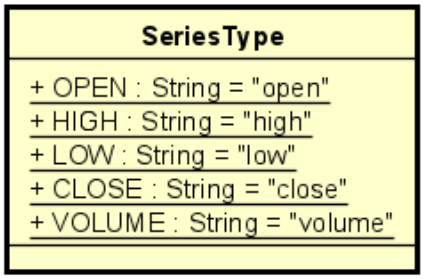

Figure 4.4: SeriesType class

As being mentioned in the beginning of the section 2.2, X_i in all of the indicator formulas can be any of the five open, high, low, close, volume. Therefore, again, the objective of SeriesType class in Figure 4.4 is to listing all possibilities of X_i.

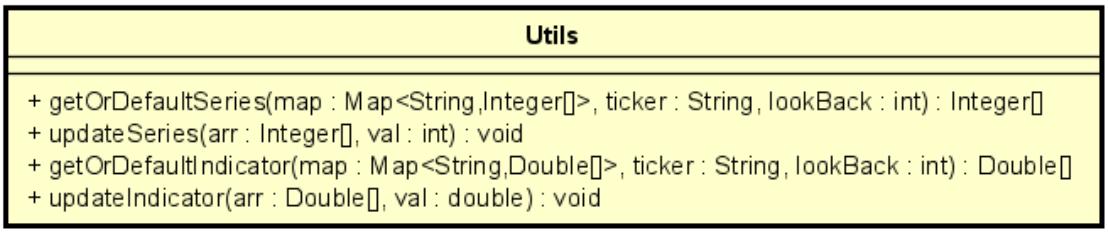

Figure 4.5: Utils class

Methods of Utils class:

(i) getOrDefaultSeries: if ticker exists in map then returns the list of integers correponding to ticker, otherwise, return a list of full zeroes. lookBack is the length of the list.

(ii) updateSeries: insert val in the head of arr and shift all other elements of arr one index back

[1]Topology defines the stream computation logic of a Kafka Streams application

(iii) getOrDefaultIndicator: same as getOrDefaultSeries, however, the list here stores doubles.

(iv) updateIndicator: same as updateSeries, however, the arr here stores doubles and the val is also double.

The rationality of the Utils class is made more clearer as we look in the Appendix B of the thesis.

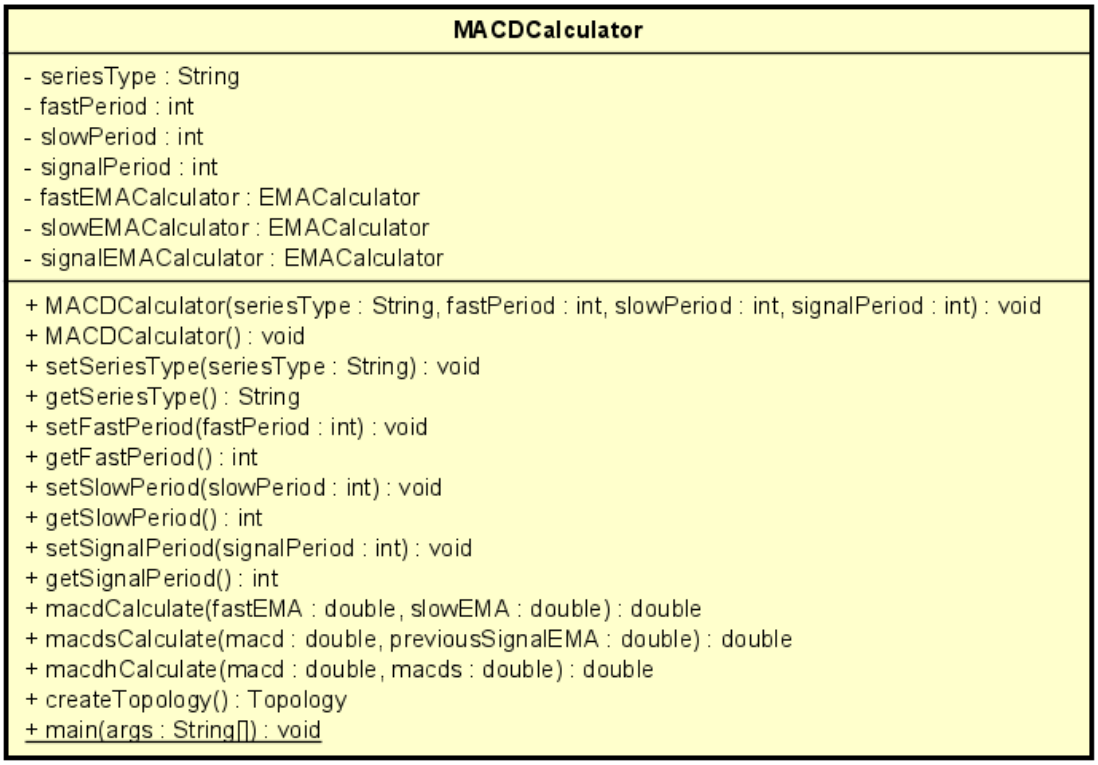

Figure 4.6: MACDCalculator class

In Figure 4.6, I presents one of the most complex indicator of the seven fixed indicators of the system. The attributes of MACDCalculator are self-explanatory with a noticeable point that it makes use of EMACalculator to do EMA calculation. MACDCalculator class has two contructors, in these two, the constructor without input constructs the MACDCalculator with default attributes. macdCalculate, macdsCalculate and macdhCalculate methods implements three formulas 2.10, 2.11, 2.12 respectively. createTopology method overrides the one of IndicatorCalculator abtract class. The public static void main method contains the code to starts the MACDCalculator.

4.3 Database design

Figure 4.7: Database schema

Figure 4.7 consists all the data models existing in Druid database. StockPrices datasource stores the OHLCV data for all stocks collecting from Techcom Securities. The other 7 datasources stores 7 fixed indicators of the system and their names indicate which indicator data they store. In all datasources, there is a __time column which is the mandatory column Druid requires. __time column has the meaning of the primary timestamp of the datasource. Because the computation of indicators are done by Kafka Streams applications and the value of indicators are stored in the datasources beforehand, the front-end only needs to call simple read queries to render a desire dashboard. Therefore, all columns other than __time are chosen to be dimensions. open, high, low, close, volume columns are of type bigint which is similar to their types in Techcom Securities. macd, aroon, ema, rsi, obv, sma, adl columns are of type double to keep them in high precision. Finally ticker column is the stock name.

4.4 Streamit application design

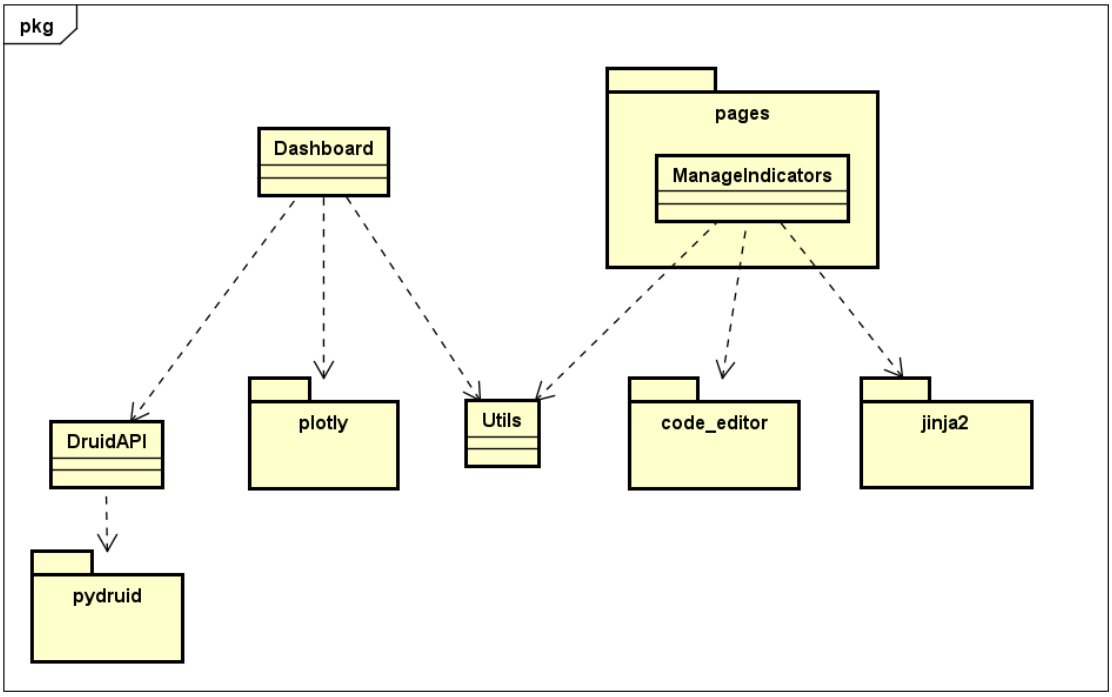

Figure 4.8: Streamlit application design

I design Streamlit application to have two pages which corresponds to two classes: Dashboard and ManageIndicators in Figure 4.8. To satisfy the constraint of Streamlit framework, all the pages other than the entry page have to be put in a package named pages. DruidAPI defines all the api needed to interact with the Druid database. The development of DruidAPI class is dependent on the existing pydruid package. Utils class is the set of useful functions that can do various tasks. The plotly, code_editor and jinja2 packages are described in 3

CHAPTER 5. IMPLEMENTATION, AND EVALUATION

Chapter 5 will go into the details of used libraries and tools. This chapter then evaluates the system designed in chapter 4.

5.1 Application building

5.1.1 Libraries and Tools

Table 5.1: List of libraries and tools used

Purpose	**Libraries Tools**	**Version**	**URL**
Code editor	Visual Studio Code	1.90.2	https://code.visualstudio.com/
Programming language	Python	3.10.12	https://www.python.org/
Programming language	Java	8	https://www.oracle.com/java/
Manage Java project	Apache Maven	3.6.3	https://maven.apache.org/
Distributed event store and stream-processing	Apache Kafka	2.7.0	https://kafka.apache.org/
Column-oriented, open-source, distributed data store	Apache Druid	29.0.1	https://druid.apache.org/
Web framework	Streamlit	1.35.0	https://streamlit.io/
Code editor component for streamlit application	streamlit-code -editor	0.1.14	https://pypi.org/project/ streamlit-code-editor/
Python Visualization package	Plotly	5.22.0	https://plotly.com/
Apache Kafka Java client and develop Kafka Streams application	org.apache. kafka	2.7.0	https://mvnrepository.com/ artifact/org.apache.kafka/ kafka-streams
Druid API Python package	pydruid	0.6.8	https://pypi.org/project/ pydruid/
Apache Kafka Python client	kafka-python	2.0.2	https://pypi.org/project/ kafka-python/
Templating engine	jinja2	3.1.4	https://jinja.palletsprojects.com/
Technical Anaysis Python package	pandas-ta	0.3.14	https://pypi.org/project/ pandas-ta/

5.1.2 Achievement

The implementation of indicator package design in section 4.2 results in a package containing 7 Kafka Streams applications. All these Kafka Streams applications aim at being deployed in the backend server to compute fixed indicators as new messages come in StockPrices Kafka topic.

The implementation of the design in 4.4 results in a package containing all the necessary code to start up a Streamlit application. This Streamlit application package aims to be deployed on the end-user device to help them monitor the Vietnamese stock market in real-time and manage their own customized indicators. However, the functionalities that allows users to create/update/delete customized indicators are still in initial development. To actually serve many different users, the application needs to have the ability to authenticate user and a database to store the indicator computing Python files associated to each user.

5.1.3 Illustration of main functions

Figure 5.1: Initial UI

Figure 5.1 illustrates the intial User Interface of the streamlit application. In the left hand side of the Figure 5.1 is the sidebar where users can navigate between two pages Dashboard and Manage your indicators. The sidebar also gives users the ability to configure the dashboard to meet their interests, for example, add a plot to the dashboard showing indicators or volume of a stock, change the color of a plot,... In the right hand side is the dashboard which currently shows the candlestick chart of the FPT stock.

Figure 5.2 shows the case when all of the 7 fixed indicators and volume histogram are enabled in the dashboard. Users can interact with the dashboard in several ways such as zoom in, zoom out by scrolling mouse wheel, drag the plot left

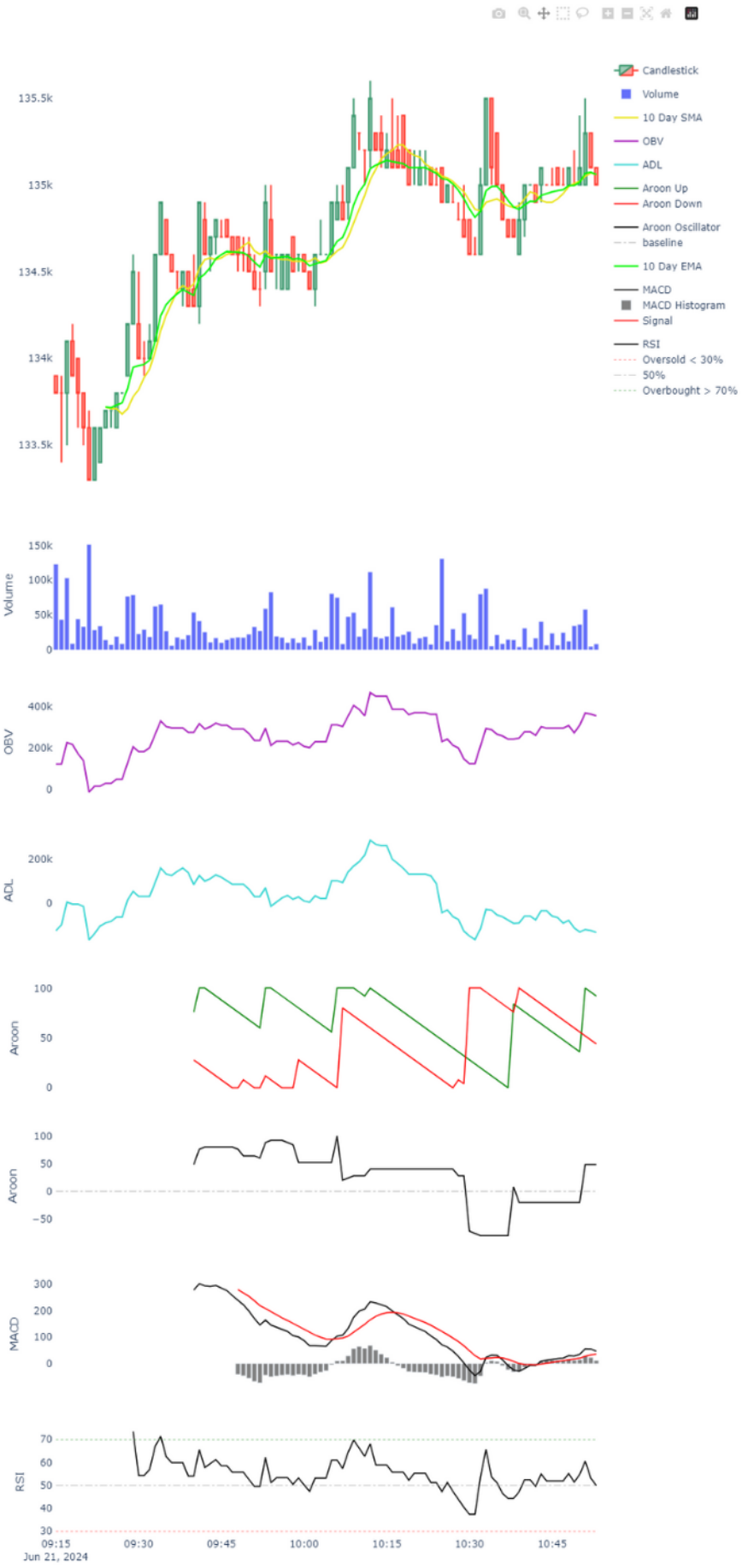

Figure 5.2: Full dashboard

or right,... User can even save a snapshot of the dashboard by click on the camera icon in the top right.

The following Figures 5.3, 5.4, 5.5, 5.6 illustrates the sequence of 4 steps to show a custom indicator named triple EMA on the dashboard.

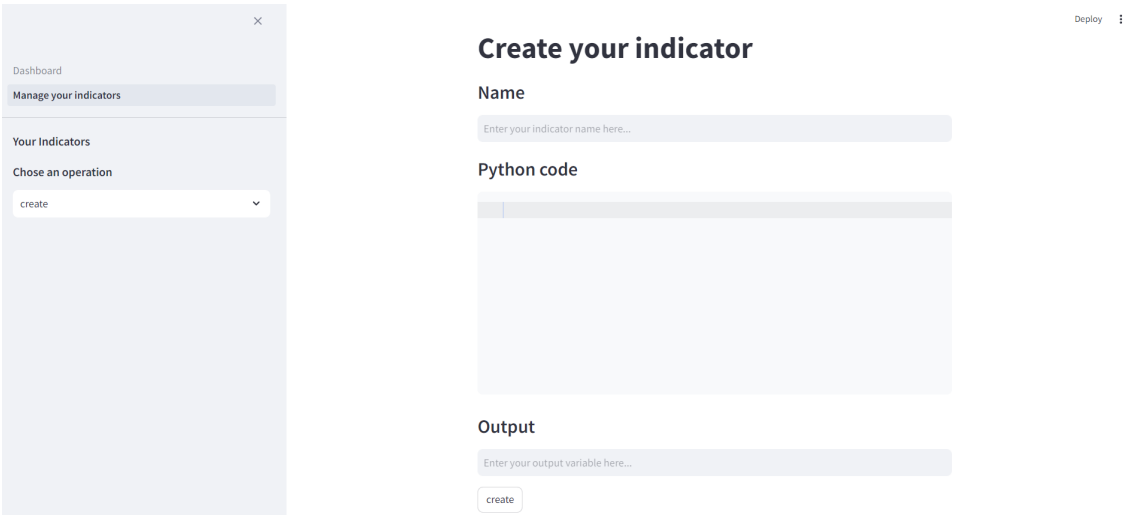

Figure 5.3: Manage your indicators page

The first step is navigate to the Manage your indicators page. Then choose create operation.

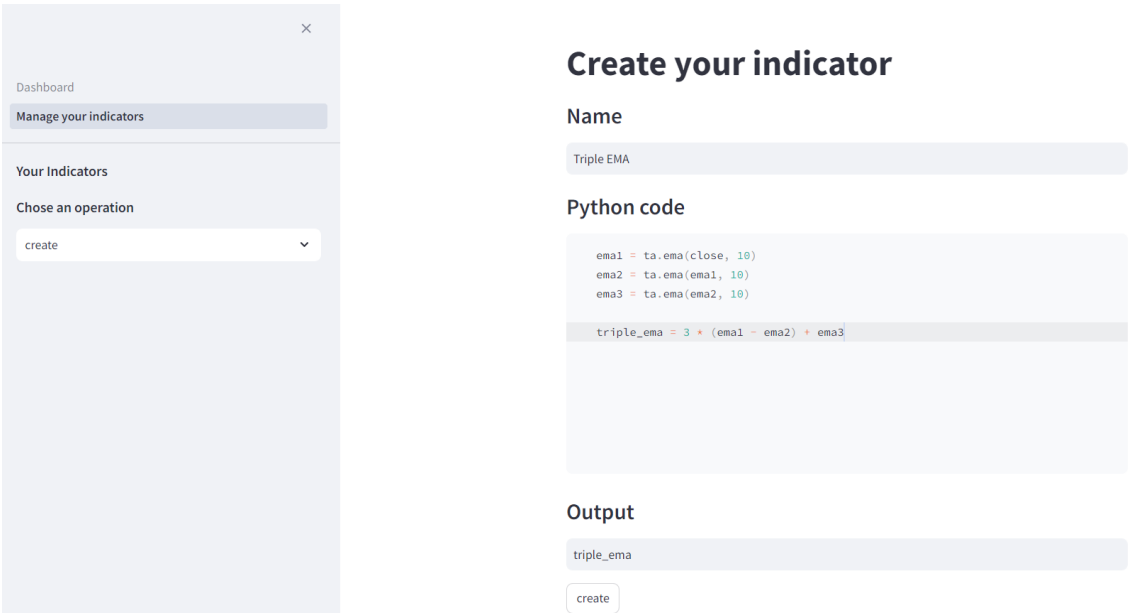

Figure 5.4: Create an indicator

The second step is fill all the blank in the form. The application helps the coding experience easier by providing python syntax highlighting and recommending.

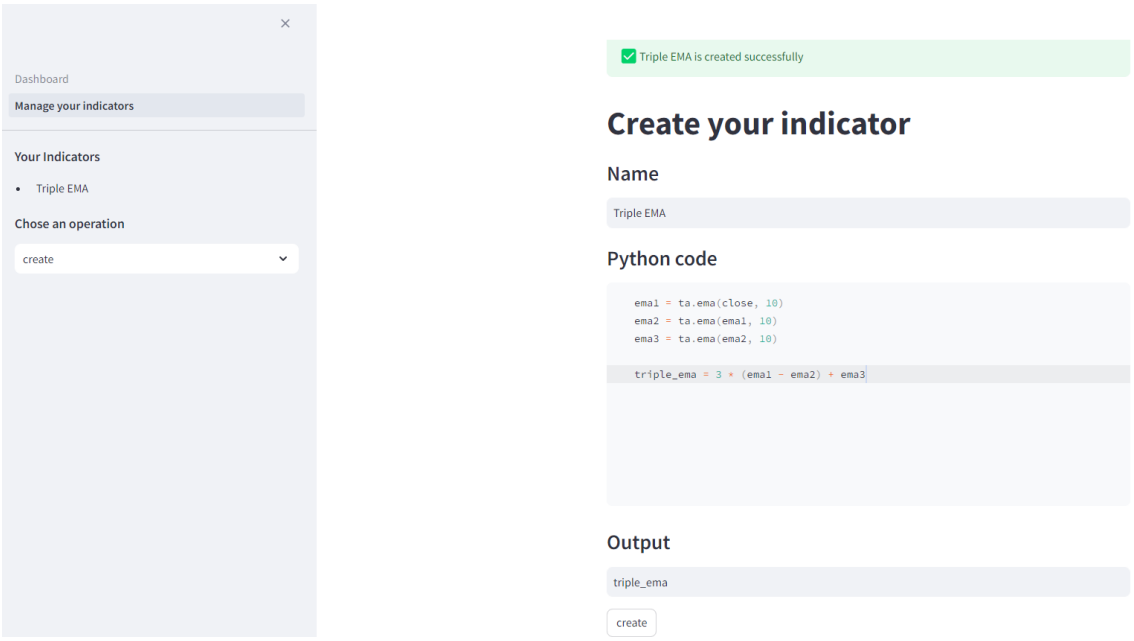

Figure 5.5: Create an indicator successfully

The third step is click on the create button. User receives a message informing that Triple EMA is successfully created. Your indicators section in the sidebar once again confirms that the creating operation is successful.

Figure 5.6: Triple EMA on dashboard

The final step is navigate back to the dashboard, select the indicator you want to show in your indicators section by choosing indicator from the drop box and click on show checkbox. The user finally happily see the Triple EMA line chart on the dashboard.

5.2 Testing

To test the functionalities of the system, the system needs to get the stock data from Techcom Securities, and Vnstock provides python api to achieve that. However, Vnstock only provides access to historical stock data and not in real time. Therefore, to simulate the stock data go in the system in real time, I first crawl historical stock data using Vnstock and store the data in a csv file named StockPrices.csv.

```
time,open,high,low,close,volume,ticker
2024-06-21 09:15:00,18800,18850,18800,18850,7800,NHH
2024-06-21 09:04:00,219600,220200,219600,220200,200,MCH
2024-06-21 09:00:00,11600,11600,11600,11600,1000,S99
2024-06-21 09:00:00,7100,7100,7100,7100,2000,MGC
2024-06-21 09:09:00,12800,12800,12800,12800,100,EBS
2024-06-21 09:19:00,32800,32800,32800,32800,200,GLT
2024-06-21 10:45:00,19700,19700,19700,19700,100,DIH
2024-06-21 09:07:00,22000,22000,22000,22000,1000,LDP
2024-06-21 09:15:00,75800,75800,75800,75800,100,LIX
2024-06-21 09:15:00,9680,9680,9680,9680,100,SMA
2024-06-21 09:00:00,30900,30900,30900,30900,100,NHC
2024-06-21 09:03:00,39000,39000,39000,39000,100,L18
2024-06-21 09:03:00,55100,55100,55100,55100,1700,CAP
2024-06-21 09:00:00,2700,2700,2700,2700,1000,MCG
2024-06-21 09:03:00,11600,11600,11600,11600,100,DPC
2024-06-21 09:04:00,26900,26900,26900,26900,3100,ABI
2024-06-21 09:15:00,32800,32800,32800,32800,300,CTF
2024-06-21 10:18:00,7600,7600,7600,7600,100,HKT
2024-06-21 09:15:00,6740,6740,6740,6740,800,TCD
2024-06-21 09:05:00,2100,2100,2100,2100,100,L62
```

Figure 5.7: StockPrices.csv

The first 20 rows of the StockPrices.csv is shown in Figure 5.7. The file contains 36817 records and has 7 columns. The data of the file are about OHLCV of 1138 stocks on Vietnamese stock market and the time spread one morning trading session of the date 2024-06-21.

Then, I use a kafka producer to produce data record by record from StockPrices.csv. The speed of producing I config is 0.01 second per record.

5.2.1 Test collecting data functionality

ID	Test case	Test process	Expected result	Actual result	Conclu -sion
1.1	Kafka ingestion recovery	The Druid supervisor controlling data ingestion from StockPrices Kafka topic is put in pending state while StockPrices topic continues to receive messages. After a while, the supervisor is resumed.	The supervisor continues where it left off	The supervisor continues where it left off	Pass
1.2	Kafka producer producing data from StockPrices.csv is down	Kafka producer producing data from StockPrices.csv is turned off	The supervisor continues to poll for new messages	The supervisor continues to poll for new messages	Pass

5.2.2 Test indicator computing functionality

ID	Test case	Test process	Expected result	Actual result	Conclu -sion
2.1	Kafka streams indicator calculator application scalability	Start one more Kafka streams application computing SMA indicator besides the one existing.	The number of partitions of StockPrices topic is distributed equally among two SMA calculator applications	The partitions assigned to each SMA calculator is redistributed equally right away when new SMA calculator application is started	Pass
2.2	Kafka streams indicator calculator application failure	Turn off one SMA calculator given that there are currently two SMA calculators running	The running SMA calculator takes over the work of the stopped one	All partitions of StockPrices topic is assigned to the running SMA calculator	Pass

5.2.3 Test create an existing customized indicator functionality

ID	Test case	Test process	Expected result	Actual result	Con -clu -sion
3.1	Create a customized indicator with existing name	Enter one of the customized indicators name in the Name field	An error message shows up informing the user that the indicator already exists and the create operation is aborted	An error message shows up informing the user that the indicator already exists and the create operation is aborted	Pass
3.2	Create a customized indicator with Python syntax error	Write Python code with syntax error in the Python code field	An error message shows up informing the user that their Python code containing syntax error and the create operation is aborted	An error message shows up informing the user that their Python code containing syntax error and the create operation is aborted	Pass
3.3	Create a customized indicator with output variable does not contained in Python code field	Input triple_ema in output field. Write code with triple_ema function and without triple_ema variable.	An error message shows up informing the user that triple_ema variable is not contained in the code and the create operation is aborted	An error message shows up informing the user that triple_ema variable is not contained in the code and the create operation is aborted	Pass
3.4	Create a customized indicator with not supported type of output variable.	Write Python code with output variable of type Int	An error message shows up informing the user that the output variable of type Int is not supported and the create operation is aborted	The new customized indicator is created	Fail

The test case 3.4 is failed because the system currently does not have any mechanism to validate the type of the output variable.

5.3 Deployment

Initially, I plan to deploy the system using Docker, Kubernetes technologies. However, I encounter with RAM limitation problem of my laptop. Therefore, the whole system is deployed in a single laptop. The laptop I use is Dell Insprison 5490 which equiped with intel i7-10510u cpu, 20 GB DDR4 Ram and SSD 512GB. I use the modest default config of Druid and default config of Kafka server to run the system. The Kafka server uses the same Zookeeper as the Druid service.

CHAPTER 6. CONCLUSION AND FUTURE WORK

6.1 Conclusion

The final result of this thesis is a system that can receive stock data in real-time, store data securely, calculate famous technical indicators and demonstrate them as an intriguing dashboard. Users are also able to easily configure the appearance of the dashboard and interact with it. Advanced users, who have prior experience in Python programming language, would find the system more useful with the manage customized indicators functionality the system provides.

However, there still exists some limitations of the system. The system is only capable of collecting historical stock data and the collecting process is not in real time as desire. Data validation process of creating new indicators and updating existing indicators use cases has not fully developed as it fails some of the test cases. The whole system have not tested in a distributed environment due to my lacking in computing resources.

After the completion of this thesis, my knowledge about data processing system development process, and developing common functionalities of a data processing system has been consolidated. I have acquired knowledge about modern technologies such as Apache Kafka, Apache Druid,... Moreover, I find that I become more independent, creative and persevering in work. My technical document reading comprehension and work arrangement skills have also been also enhanced. These valuable experiences will surely be the solid foundation for me to improve myself further in my future work and in my personal life.

6.2 Future Work

Based on the advantages and limitations presented above, the system can be further developed in these future directions:

Mitigating all the limitations presented in 6.1 by developing our own real time Vietnamese stock data crawling package, fixing all the fail test cases when creating or updating customized indicators, deploying the system on cloud.

Adding new components the system such as Postgres database to store customized indicators associated to each user.

Some functionalities that can be considered to develop: authenticating users, defining new trading strategies, backtesting trading strategies, forcasting future stock prices using machine learning models.

REFERENCE

[1] R. S. K. P. Gwen Shapira Todd Palino, *Kafka: The Definitive Guide 2nd Edition*. O'Reilly Media, Inc., 2021.

[2] A. Druid, *Storage overview*, 2023. [Online]. Available: `https://druid.apache.org/docs/latest/design/storage`.

[3] A. Druid, *Segments*, 2023. [Online]. Available: `https://druid.apache.org/docs/latest/design/segments/`.

APPENDIX

A. Python file rendering process

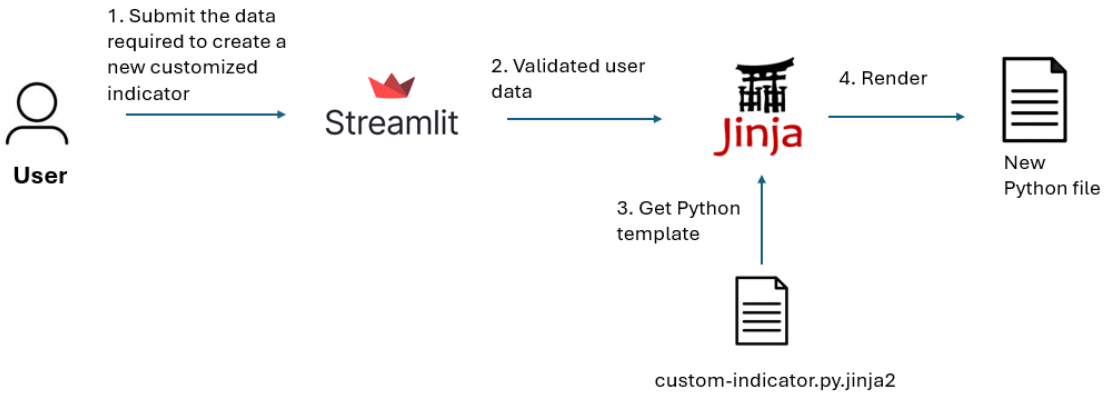

Figure A.1: Python file rendering process

Figure A.1 demonstrates how the system produces a new python file to compute user defined indicator. The user uses the create a new indicator functionality provided by the Streamlit application to submit Python code and other required data. The data is then validated by Streamlit application before going into Jinja2 engine. Jinja2 engine put the validated data into the annotated field in the predefined Python template named custom-indicator.py.jinja2. The outcome of the rendering process is an new Python file.

```python
import druid_api
import pandas as pd
import pandas_ta as ta
import numpy as np
import time

ticker = sys.argv[1]
curs = druid_api.get_cursor()

while 'StockPrices' not in druid_api.get_data_sources():
    time.sleep(5)

ohlcv = druid_api.get_query_result(f"""
    SELECT *
    FROM StockPrices
    WHERE ticker='{ticker}'
""", curs)

open = ohlcv['open']
high = ohlcv['high']
low = ohlcv['low']
close = ohlcv['close']
volume = ohlcv['volume']
n = len(open)

{{ user_code }}

out = {{ output_variable }}
```

Figure A.2: custom-indicator.py.jinja2

Figure A.2 shows a part of the content of the Python template custom-indicator.py.jinja2 which is also shown in A.1. In the top of the file, there are 3 commands to import 3 packages pandas, numpy and pandas_ta. These 3 packages are designed to support scientific computing, efficient data transforming and predefined technical indicators computing functions. User can call functions from these 3 packages to achieve their needs. The file also predefined some frequently used variables: open, high, low, close, volume and n which is the length of the iterable, therefore, users can use these variables right away without the need of defining in their code. The parts inside the are where Jinja2 engine put user data in the file.

B. Kafka streams indicator calculator implementation

In appendix B, we will delve into an implementation of a Kafka streams application to compute SMA indicator and see how Utils class is used.

```java
@Override
public Topology createTopology() {
    StreamsBuilder builder = new StreamsBuilder();

    KStream<String, String> stockPrices = builder.stream(topic:"StockPrices");
    ObjectMapper objectMapper = new ObjectMapper();
    Utils utils = new Utils();

    Map<String, Integer[]> seriesMap = new HashMap<>();

    KStream<String, String> smaStream = stockPrices.mapValues((ValueMapper<String, String>) value -> {
        try {
            JsonNode jsonNode = objectMapper.readTree(value);

            String time = jsonNode.get(fieldName:"time").asText();
            String ticker = jsonNode.get(fieldName:"ticker").asText();

            Integer[] series = utils.getOrDefaultSeries(seriesMap, ticker, this.period);
            utils.updateSeries(series, jsonNode.get(this.seriesType).asInt());
            seriesMap.put(ticker, series);

            double currentSMA = this.calculate(series);

            Map<String, Object> result = new HashMap<>();
            result.put(key:"time", time);
            result.put(key:"ticker", ticker);
            result.put(key:"sma", currentSMA);

            return objectMapper.writeValueAsString(result);
        } catch (Exception e) {
            e.printStackTrace();
            return null;
        }
    });
    smaStream.to(topic:"SMA", Produced.with(Serdes.String(), Serdes.String()));
    return builder.build();
}
```

Figure B.1: createTopology method in SMACalculator class

createTopology is the most important method in all IndicatorCalculator classes, because it outputs a Kafka Streams Topology which is the execution logic of a Kafka Streams application. The idea of the implementation is that we keep a Map<String, Integer[]> data structure named seriesMap in the memory. seriesMap stores all the historical data needed to do SMA calculation. Suppose that we want to calculate the 10-day SMA of close prices, seriesMap is the mapping of a ticker to a list of 10 latest close prices of that ticker. Each time we need to calculate a new SMA value, we use utils.getOrDefaultSeries to initialize or get a list of 10 latest close prices of a ticker. As we receive new close price, we call utils.updateSeries to put the newly close price into the 10-latest close prices list.