VIETNAM NATIONAL UNIVERSITY, HANOI

UNIVERSITY OF ENGINEERING AND TECHNOLOGY

Le Vu Quang

BUILDING AUTOMATED TRADING BOT ON BINANCE EXCHANGE

UNIVERSITY GRADUATE THESIS

Major: Information Technology

HA NOI - 2023

VIETNAM NATIONAL UNIVERSITY, HANOI

UNIVERSITY OF ENGINEERING AND TECHNOLOGY

Le Vu Quang

BUILDING AUTOMATED TRADING BOT ON BINANCE EXCHANGE

Major: Information Technology

Supervisor: Assoc. Prof. Truong Anh Hoang (PhD.)

Co-Supervisor: Dr. Nguyen Van Vinh

HA NOI - 2023

Tóm Tắt

Thị trường tiền điện tử đã trở nên sôi động hơn trong những năm gần đây. Kéo theo là các hệ thống giao dịch dựa trên thuật toán cũng đã được phát triển và tối ưu hóa một cách rộng rãi để phù hợp với các đặc tính của thị trường tiền điện tử. Mục tiêu chính của luận văn này là giới thiệu và mô tả quy trình toàn diện để xây dựng một bot giao dịch tự động trên sàn giao dịch tiền điện tử nổi tiếng Binance.

Dựa trên các chỉ báo thường được sử dụng trong giao dịch, tôi đã phát triển các chiến lược giao dịch của riêng mình. Những chiến lược này được xây dựng bằng cách quan sát kỹ lưỡng các biến động của thị trường trong khung thời gian ngắn. Mục tiêu là xây dựng một hệ thống phần mềm thực hiện các chiến lược này một cách hiệu quả, nhanh chóng, chính xác và đáng tin cậy. Trong luận văn này, tôi đã trình bày quy trình xây dựng sản phẩm phần mềm của mình, bao gồm phương pháp phát triển phần mềm đã sử dụng, các công nghệ liên quan, yêu cầu phần mềm, thiết kế hệ thống, phát triển mã nguồn, kiểm thử, triển khai hệ thống và giám sát.

Sau khi hệ thống được phát triển, tôi đã cho chạy trên cả môi trường testnet và mainnet của sàn giao dịch Binance, và kết quả cho thấy những dấu hiệu khả quan về khả năng và tính khả thi của hệ thống.

***Từ khoá**: Thị trường tiền điện tử, Hệ thống giao dịch dựa trên thuật toán, Sàn giao dịch Binance.*

Abstract

The cryptocurrency market has become increasingly active in recent years. As a result, algorithmic trading systems have also been developed and optimized extensively to match the characteristics of the cryptocurrency market. The primary objective of this thesis is to showcase the comprehensive procedure involved in constructing an automated trading bot on the renowned Binance exchange.

Based on commonly used indicators in trading, I have developed my own trading strategies. These strategies were devised by closely observing market movements within short time frames. The aim was to build a software system that executes these strategies efficiently, quickly, accurately, and with high reliability. In this thesis, I have presented the process of building my software product, which includes the methodology used, related technologies, software requirements, system design, system implementation, testing, system deployment, and monitoring.

Following the development of the system, I executed it on both the testnet and mainnet platforms of the Binance Exchange, and the findings revealed encouraging indications of the system's capability and feasibility.

Keywords*: Cryptocurrency Market, Algorithmic Trading System, Binance Exchange.*

Acknowledgements

I would like to express my heartfelt gratitude to Assoc. Prof. Truong Anh Hoang (PhD.) for providing invaluable guidance and support throughout the process of writing this thesis. His feedback, insightful comments, and suggestions have been instrumental in shaping my ideas and improving the quality of this thesis. I am also grateful for his availability and willingness to discuss any issues and concerns that arose during this journey.

I also would like to thank my co-supervisor, Dr. Nguyen Van Vinh, for his dedicated guidance and helpful response to my inquiries during the thesis writing process.

I wish to express my sincere gratitude to my esteemed colleagues at Kyber Network. I thank Engineering Manager Dinh Van Tien for accepting me as an intern and teaching me valuable lessons about software systems, as well as providing me with many opportunities to develop and prove myself. I also thank Engineer Nguyen Viet Anh and Engineering Manager Nguyen Khanh Ha for their constant support since I started working at Kyber Network. They have taught me a lot about software development, coding skills, and system design. Last but not least, I thank Product Manager Ha Anh Son and Engineer Nguyen Huu Dung for teaching me about the crypto market and providing me with valuable lessons for self-development.

Finally, I would like to express my appreciation to my family and friends who have supported me throughout this journey with their love, encouragement, and understanding.

Thank you all for your invaluable support and encouragement.

Declaration

I declare that this thesis, entitled “Building Automated Trading Bot On Binance Exchange” is entirely my original work, except where otherwise acknowledged, and has not been submitted for any degree or examination at any other university.

I confirm that I have fully acknowledged all sources of information in the bibliography and that any data or ideas not my own are clearly indicated as such, with the appropriate references cited.

I declare that this thesis does not contain any material that has been previously published or written by another person, except where due reference has been made in the text.

I understand that any false statement made in this declaration may result in the revocation of my degree by University of Engineering and Technology, VNU.

Student

Le Vu Quang

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

With the first appearance of Bitcoin [4] in 2009, blockchain technology has developed rapidly over the past decade. Along with that, cryptocurrencies are continuously emerging and the market has gradually formed and grown with the all-time high market capitalization of over 3 trillion US dollars [1].

Cryptocurrency market is increasingly attracting many investors due to some advantages. Firstly, it offers the potential for high returns due to its high volatility. Secondly, the market operates 24/7, allowing traders to buy and sell at any time. Thirdly, the market is decentralized and operates independently of governments and financial institutions, offering greater freedom and autonomy. Additionally, the technology behind cryptocurrencies, such as blockchain, is seen as innovative and has the potential to disrupt traditional financial systems.

Recognizing these opportunities, I decided to build a trading bot with the aim of profiting from the market as well as gaining deeper insights into the cryptocurrency market. Furthermore, since developing an automated trading bot requires a high level of technical knowledge, I believe it is an excellent chance to improve my professional skills.

[1] Data from CoinGecko. `https://www.coingecko.com/en/global-charts`

1.2 Research Objectives

The aim of this thesis is to build an automated trading bot able to connect to Binance Exchange - the largest cryptocurrency exchange in the world, via their API. The software can collect and process data related to the market and the exchange in real-time. Based on the collected data and algorithm, the bot can place buy and sell orders on the exchange to make a profit. In addition, the administrator can manage the bot's behavior and monitor statistics through Telegram application.

1.3 Thesis Layout

My thesis includes five main Chapters and one Conclusion, as follow:

Chapter 1: Introduction. This chapter is an introduction to thesis **Building Automated Trading Bot On Binance Exchange** through presenting motivations, project goals and thesis structure.

Chapter 2: Background. I introduced the knowledge I learned to finish this thesis, including the concepts of blockchain, cryptocurrency exchanges, and details about the technical indicators used. Additionally, I presented the software development methodology and the technologies used to build the system.

Chapter 3: Trading Strategy. In this chapter, I have elaborated in detail on my personal trading strategy which I have utilized to implement the system.

Chapter 4: System Development. This chapter presents the software development process through sections describing the system requirements, system design, system implementation, and system deployment and monitoring.

Chapter 5: Evaluations. I presented the results of running the software on both the testnet and mainnet environments, and provided my own evaluation of the developed software based on these results.

Chapter 5: Conclusion. This segment encompasses three main components. Firstly, I have provided an overview of the work that has been accomplished throughout the thesis. Secondly, I have disclosed my personal reflections and insights on the process of writing the thesis. Finally, I have highlighted the upcoming functionalities that will be incorporated into the product in the future.

Chapter 2

Background

2.1 Blockchain

Blockchain [5] is a distributed database that is run by a network of nodes. In a blockchain, a block is a unit of data that contains a set of transactions that have been verified and added to the chain after achieving consensus from the network. After being added to the chain, block data or transactions cannot be altered or deleted, ensuring the integrity of the system. Each block in the chain is linked to the previous block, forming a chain of blocks, hence the name "blockchain". (Figure 2.1).

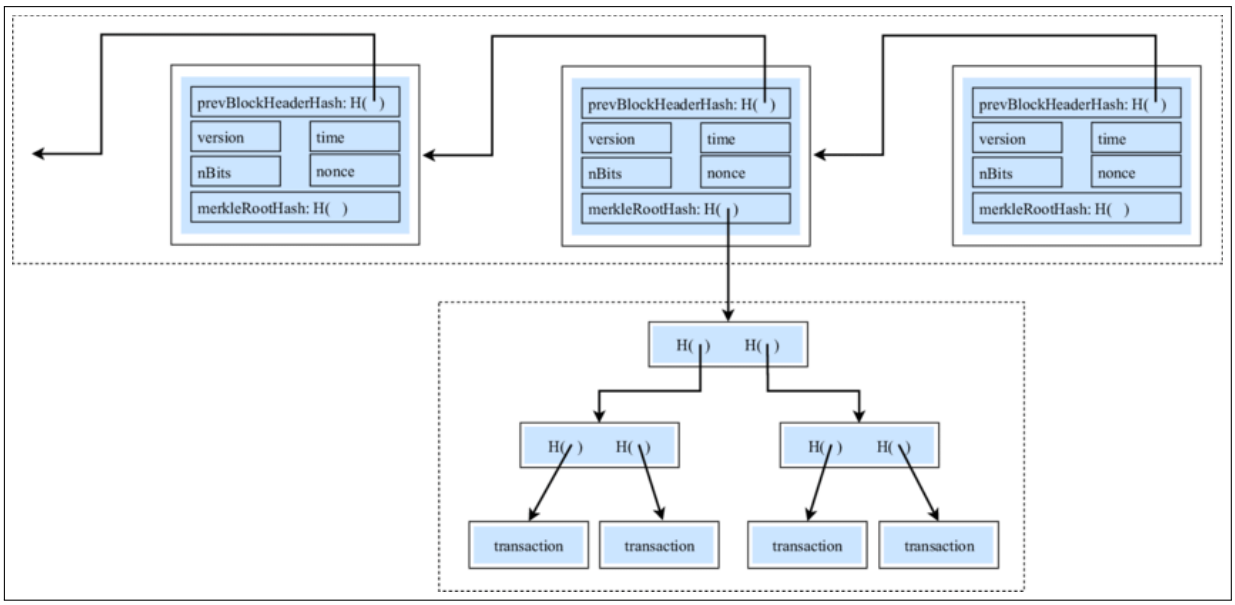

Figure 2.1: A simplified structure of the Bitcoin blockchain. Reprinted from Research-Gate: Towards Efficient Cross-Blockchain Token Transfers. [12]

A block contains a cryptographic hash of the previous block, a timestamp, and a set of transactions. The size of each block in a blockchain is typically limited, to ensure that the network can process transactions quickly and efficiently. The size limit can vary depending on the specific blockchain and its protocol.

In the case of Bitcoin, a new block is added to the chain every 10 minutes [1]. A block is considered valid if it gets up to 6 confirmations, or in other words if 6 blocks have appeared directly after it, estimated to take about 1 hour, although this time is variable and may depend on the seller and the monetary amount.

2.2 Cryptocurrency

Cryptocurrencies are assets on blockchain networks. They are stored and transferred using digital wallets [6], which are secured with private keys that only the owner has access to.

Cryptocurrencies can be used for a variety of purposes, such as peer-to-peer payments, online purchases, investments, and trading on cryptocurrency exchanges. Some of the most popular cryptocurrencies include BTC[1], ETH[2], LTC[3], and XRP[4], among others.

2.3 Cryptocurrency Exchanges

There are many types of cryptocurrency exchanges, but let's focus on the two most common ones: Decentralized Exchanges and Centralized Exchanges.

2.3.1 Decentralized Exchanges

DEXs are peer-to-peer marketplaces allowing traders to make transactions directly without relying on intermediaries or custodians. Transactions on DEXs are executed using self-executing agreements called smart contracts which are programs running on blockchain networks.

DEXs enable peer-to-peer trading of cryptocurrencies, linking buyers and sellers

[1]Native asset on Bitcoin blockchain.
[2]Native asset on Ethereum blockchain.
[3]Native asset on Litecoin blockchain.
[4]Native asset on Ripple blockchain.

without the involvement of a centralized authority. These exchanges are typically non-custodial, meaning users have control over their wallet's private keys, which provide access to their cryptocurrencies. With private keys, users can access their crypto balances immediately after logging into the DEX, without having to submit personal information such as names and addresses, thus ensuring greater privacy. Prominent DEXs include Uniswap [3], KyberSwap [20], PancakeSwap, and Curve [11].

2.3.2 Centralized Exchanges

CEXs are cryptocurrency exchanges that are owned and operated by a single company or organization. CEXs act as intermediaries between buyers and sellers, they make money by charging transaction fees or commissions for their services. In a centralized exchange, the exchange operator has control over funds of users, and users need to deposit their digital assets into the exchange's wallet before they can begin trading. CEXs are often more user-friendly and have more liquidity compared to DEXs, but they are also subject to potential security risks and regulatory scrutiny. Popular examples of CEXs include Binance, Coinbase, Kraken, and Huobi.

2.4 Spot Market

The Spot Market [28] is different from the Futures Market [25], where users can trade futures contracts. On Spot Market, users can trade using various order types such as market orders, limit orders, and stop-limit orders. Market orders are executed immediately at the best available price, while limit orders are executed at a specified price or better. Stop-limit orders are similar to limit orders but only become active when the price of the asset reaches a specified level.

Automated Trading Bot presented in this thesis is currently able to execute trades on the Binance Spot Market using limit orders.

2.5 Algorithmic Trading

Algorithmic trading [22] is a trading method that uses computer software. These programs are capable of automatically placing trading orders in order to generate profits based on algorithms and data analysis. Algorithmic trading has become increasingly popular in recent years due to advances in technology and the growing availability of

financial data.

Automated trading has the potential to enhance efficiency and lower trading expenses. With the trading process being automated, trades can be conducted with higher accuracy and speed, which mitigates the impact of human error.

Below is a detailed presentation of the indicators used by the Bot to make trading decisions.

2.5.1 Simple Moving Average

SMA [8] is a widely used statistical method in financial analysis. It is a calculation that gives equal weight to each data point in a time series, and it is frequently used to smooth out short-term fluctuations and highlight longer-term trends in a data set. The SMA is calculated by summing up a specified number of data points, then dividing the total by the number of data points. In the figure 2.2, the blue line represents the SMA indicator applied on 1-hour timeframe candlestick chart of BTCUSDT trading pair.

Figure 2.2: SMA indicator.

The SMA is often used in technical analysis as a trend-following indicator. If the current price of an asset is above its SMA, it may be considered as an indication of an uptrend, while a price below its SMA may be considered as a sign of a downtrend. SMA

can also be used to identify support and resistance levels, as well as potential entry and exit points for trades. Traders often use multiple SMAs with different time periods to gain a more complete picture of an asset's price movements.

2.5.2 Exponential Moving Average

The Exponential Moving Average [14] is a commonly used technical indicator in financial analysis that is used to smooth out the price data of an asset over a certain time period. It is a type of moving average that places greater weight on more recent data points, making it more responsive to changes in the price of the asset. By analyzing the EMA line, traders can identify potential buying or selling opportunities based on the direction of the trend. The EMA can also be used to indicate levels of support or resistance, as well as to identify trend reversals or trend strength. The blue line in figure 2.3 represents the EMA indicator applied on 1-hour timeframe candlestick chart of BTCUSDT trading pair.

Figure 2.3: EMA indicator.

Mathematically, EMA is calculated using a formula that takes into account the current price of the asset P, the previous period's EMA value EMA_{prev}, and a smoothing factor or constant K. The smoothing factor determines the rate at which the weight given to each previous data point decreases over time and its value equals $\frac{2}{N+1}$ where N is the

number of periods to compute EMA. The formula for calculating EMA is:

$$EMA = P * K + EMA_{prev} * (1 - K) \tag{2.1}$$

2.5.3 WaveTrend Oscillator

The Wavetrend Oscillator [15] is a technical analysis indicator used to identify trending markets and potential entry and exit points in trading. When the oscillator is above a certain threshold level, it is considered bullish and signals a potential uptrend. Conversely, when the oscillator is below the threshold, it is considered bearish and signals a potential downtrend. The Wavetrend Oscillator can also be used to identify overbought or oversold conditions in the market. When the oscillator reaches extreme levels, it may indicate that the market is overbought or oversold, which could signal a potential reversal in the trend (Figure 2.4). Traders can use this information to adjust their trading positions accordingly.

Figure 2.4: WaveTrend Oscillator indicator.

The calculation of the WaveTrend Oscillator involves steps:

1. This step computes array ap (Absolute Price) by calculating HLC3 of each element in candlestick array data. HLC3 stands for the typical price of a candlestick. This

value is calculated by taking the average of the high, low, and close prices. The formula for calculating HLC3 is as follows:

$$HLC3 = \frac{High + Low + Close}{3} \tag{2.2}$$

2. Calculate EMA of the above HLC3 data with $n1$ periods. This array is named as esa (Ehlers' Smoothing Average).

$$esa = EMA(ap, n1) \tag{2.3}$$

3. This step calculates d using EMA with $n1$ periods of the absolute difference between ap and esa. It is used to quantify the distance between the actual price and the equilibrium price. By taking the absolute difference between ap and esa, the step is calculating the deviation of the price from the equilibrium price. By applying EMA to this deviation, it smoothing out the value and makes it more responsive to recent price movements.

$$d = EMA(ABS(ap - esa), n1) \tag{2.4}$$

4. Channel index ci array is calculated in this step. The calculation of ci involves subtracting esa from ap. This difference is then divided by the product of 0.015 and d. 0.015 is a constant value that is chosen empirically to provide a good balance between the number of signals generated and the accuracy of those signals. The resulting ratio indicates the distance of the price from the centerline of the channel, which is represented by the zero level of the oscillator. A positive value of ci indicates that the price is above the centerline, and a negative value indicates that the price is below the centerline. The farther away the price is from the centerline, the higher the absolute value of ci. Therefore, the ci can be used to identify overbought and oversold conditions and potential trend reversals.

$$ci = \frac{ap - esa}{0.015 * d} \tag{2.5}$$

5. This step calculates the final value tci (Trading Channel Index) of the Wavetrend Oscillator by taking EMA of the ci values calculated in the previous step. By smoothing the ci values over a longer period $n2$, the tci helps to filter out short-term noise and provides a more reliable indication of the market trend.

$$tci = EMA(ci, n2) \tag{2.6}$$

6. Set value of first wavetrend $wt1$ equal to the value of tci.

$$wt1 = tci \tag{2.7}$$

7. Second wavetrend $wt2$ is calculated by taking SMA of the $wt1$ values calculated in the previous step, using a window size of 4. This helps to smooth out any short-term fluctuations and provides a more stable signal for trading decisions.

$$wt2 = SMA(wt1, 4) \tag{2.8}$$

After performing the necessary computations, the trading decisions of the Bot are made based on the values of $wt1$ and $wt2$.

2.6 Scrum Software Development Methodology

In the context of software development, Scrum [24] is an agile methodology that is widely used for managing complex projects. It is based on iterative and incremental development principles, with a strong emphasis on collaboration, flexibility, and adaptability. Scrum provides a structured framework that allows development teams to work in a collaborative and self-organizing manner, with a focus on delivering incremental value to stakeholders.

Scrum typically involves cross-functional teams working in short iterations called sprints, which are usually two to four weeks long. The development process is divided into several key roles, including the Product Owner, Scrum Master, and Development Team, each with specific responsibilities. The Product Owner is responsible for defining and prioritizing the product backlog, which contains the requirements or features to be developed. The Scrum Master is responsible for facilitating the Scrum process and ensuring that the team follows Scrum principles and practices. The Development Team is responsible for designing, coding, testing, and delivering potentially shippable increments of the product during each sprint.

Scrum also includes specific ceremonies or events, such as the daily Scrum stand-up meetings, sprint planning meetings, sprint reviews, and sprint retrospectives, which provide opportunities for team members to collaborate, plan, review, and continuously improve their work.

In this thesis, the Scrum methodology is being utilized as the primary approach for managing the software development process, with a focus on leveraging its iterative

nature to ensure efficient and effective development of the automated trading bot on the Binance exchange. The utilization of Scrum provides a structured framework for managing the project, promoting transparency, adaptability, and continuous improvement throughout the development lifecycle.

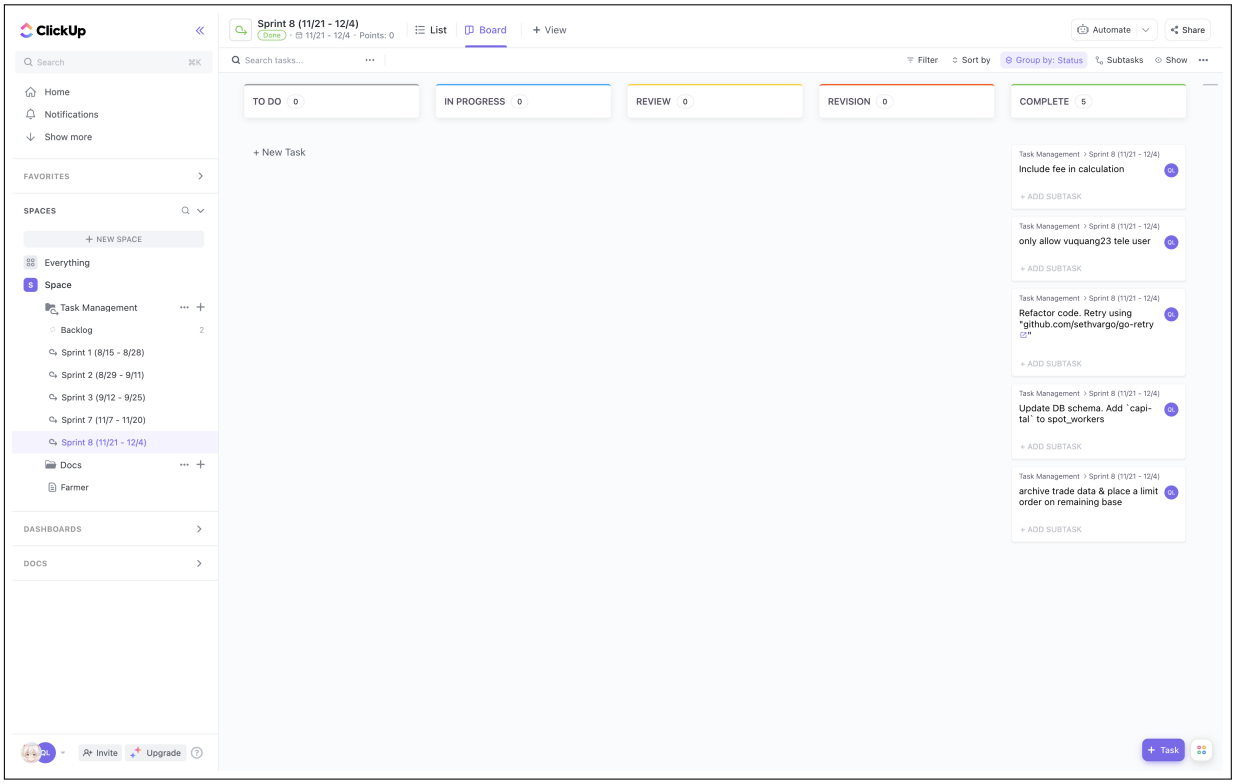

Figure 2.5: A sprint board.

In addition to utilizing the Scrum methodology, this project also employs ClickUp[5] as a project management tool to facilitate the development process (Figure 2.5). ClickUp is a versatile and user-friendly project management platform that offers a range of features that contribute to an efficient and effective development process for this project, including improving task management, and tracking progress.

2.7 Technologies

2.7.1 Golang

In this thesis, the Go programming language [10], also known as Golang, is utilized as the primary programming language for the development of the software system. Go is

[5] `https://app.clickup.com`

an open-source, statically-typed programming language developed by Google, designed to combine the efficiency and performance of a compiled language with the simplicity and readability of a dynamically-typed language. Go is renowned for its fast compilation times, efficient concurrent programming capabilities, and strong support for concurrent and parallel processing. It is particularly well-suited for building scalable, high-performance systems, networked applications, and distributed systems. Go's concise syntax and comprehensive standard library make it a popular choice for a wide range of applications, making it an ideal choice for the development of the software system in this thesis.

Goroutine

In this software, there are several tasks that require concurrent execution, including detecting trading opportunities, updating Binance Exchange info, and calculating Wavetrend Oscillator values, etc. To achieve concurrent processing, I utilize Goroutine, which is a fundamental feature of the Go programming language. It allows for the creation of lightweight threads, known as goroutines, which are efficiently managed by the Go runtime and can run concurrently with other goroutines within the same program.

Channel

Multiple Goroutines require intercommunication in this software, which is achieved using channels. Channels in Go are a powerful mechanism that enables seamless communication and synchronization between concurrent Goroutines. They serve as conduits for exchanging data among different Goroutines, facilitating safe and efficient coordination. Channels adhere to the "Do not communicate by sharing memory; instead, share memory by communicating" principle, which promotes a concurrent and safe programming paradigm. They are used for coordinated sending and receiving of data, ensuring prevention of race conditions and other concurrency-related issues.

2.7.2 MySQL

Considering the extensive adoption of MySQL [26] as an open-source relational database management system known for its user-friendly nature, scalability, and high reliability, it has been chosen as the database management system for this project.

2.7.3 Git and GitHub

In this project, I have utilized Git [16] and GitHub[6] as essential tools for version control and code management. Git provides me with a distributed version control system that allows me to effectively track changes and manage different branches. GitHub, being a web-based hosting service, serves as a centralized platform for hosting the Git repository, enabling issue tracking and pull requests. The use of Git and GitHub in this project enables me to maintain a well-organized and efficient development workflow, ensuring proper versioning and code management throughout the project lifecycle.

2.7.4 Docker and DockerHub

Docker [21] is an open-source platform that allows developers to create, package, and run applications in lightweight, portable containers. Containers encapsulate an application and its dependencies, making it easy to deploy and run consistently across different environments. I use Docker to run the software, including MySQL, for development and deployment of the system.

DockerHub[7], on the other hand, is a cloud-based registry for Docker images. It provides a centralized platform for storing and sharing Docker images, allowing developers to easily access and download pre-configured containers. DockerHub supports deployment of the software on cloud-based environments.

2.7.5 Google Cloud Platform

Google Cloud Platform[8] is a cloud computing service provided by Google that offers a wide range of cloud-based infrastructure and services for building, deploying, and managing applications and services. Compute Engine is one of the core services provided by GCP, which allows users to create and manage virtual machines in the cloud. It provides scalable and flexible computing resources that can be customized to suit specific requirements, and enables users to run a variety of operating systems and applications on virtual instances hosted on Google's infrastructure. I deployed the application and MySQL on the same Compute Engine to save resources.

[6] `https://github.com`
[7] `https://hub.docker.com`
[8] `https://cloud.google.com`

2.7.6 Grafana Loki and Promtail

I used Grafana Loki [9] and Promtail [7] for log aggregation and monitoring purposes.

Promtail is a lightweight log collector and shipper that can be easily deployed to collect logs from various sources and send them to Grafana Loki for storage and querying. Promtail can be configured to watch log files or log streams and can handle log rotation.

Grafana Loki is a horizontally-scalable, highly-available, multi-tenant log aggregation system inspired by Prometheus. It is designed to store logs in a cost-effective and efficient way, using an index-free architecture that enables efficient querying of logs. Grafana Loki provides a query language and an API to search, filter, and analyze logs in real-time.

Chapter 3

Trading Strategy

3.1 Definitions

Some variables, symbols, and phrases will be defined or introduced to support the presentation of the buying and selling strategy below.

The Bot utilizes the Wavetrend Oscillator in both 1-minute and 1-hour timeframe charts to facilitate decision-making for each trading pair. Building on the introduction of two arrays $wt1$ and $wt2$ in 2.5.3, let's define four variables $wt1\_1m$, $wt2\_1m$, $wt1\_1h$ and $wt2\_1h$. The variables $wt1\_1m$ and $wt2\_1m$ correspond to $wt1$ and $wt2$, respectively, and are calculated using 1-minute timeframe candlestick data. On the other hand, $wt1\_1h$ and $wt2\_1h$ are computed using 1-hour timeframe candlestick data. Subsequently, we introduce two variables, dif_wt_1h and dif_wt_1m, which are computed as follows:

$$dif\_wt\_1m = wt1\_1m - wt2\_1m \tag{3.1}$$

$$dif\_wt\_1h = wt1\_1h - wt2\_1h \tag{3.2}$$

In the Wavetrend Oscillator, the oversold and overbought levels are specific threshold values that can be used to identify potential trend reversals in the price of an asset. The oversold level $over\_sold$ is the lower threshold value, and it indicates that the price has reached a level that may be considered undervalued or oversold. It can be used as a signal for potential buying opportunities. The overbought level $over\_bought$ is the upper threshold value, and it indicates that the price has reached a level that may be considered overvalued or overbought. It can be used as a signal for potential selling opportuni-

ties. Figure 3.1 below visualizes the variables on the 1-hour candlestick chart of the BTCUSDT trading pair.

Figure 3.1: Wavetrend Oscillator on 1-hour candlestick chart of the BTCUSDT trading pair.

Since the variables $wt1\_1m$, $wt2\_1m$, dif_1m, $wt1\_1h$, $wt2\_1h$, and dif_1h are arrays of values, we use the syntax of some programming languages for the convenience of specifying values in the array. For example, the $i-th$ element in array a is denoted by $a[i]$, and the set of elements in the array with an index greater than or equal to $start$ and less than end is denoted by $a[start : end]$. Please note that these arrays are continuously updated by appending a new value at the end of each time period.

Let me introduce a new phrase called "trading unit", whereby when a buying opportunity is detected, the Bot will use a certain number of trading units to execute the transaction. When the bot detects a buying opportunity, it will use a number of trading units designated as $bullish\_buy$ to make the purchase if the market condition is bullish. Otherwise, it will use a number designated as $bearish\_buy$. The valuation of a trading unit is expressed as the dollar amount utilized to acquire the base currency[1], designated as $unit\_notional$, which is set by the admin. As a result of the finite capital allocated for

[1]The first currency quoted in a trading pair. It is the currency that the trader is buying or selling.

trading, the Bot will be assigned a predetermined upper limit on the number of trading units that can be utilized for trading purposes, which is referred to as $max\_buy\_allowed$.

I will now provide an illustrative example to facilitate comprehension. Let's say that I want the bot to trade on the BTCUSDT pair and I set the $max\_buy\_allowed$ to be 12 with $unit\_notional$ to be 50. From this, we can infer that the bot has 600 USDT available for trading. In a bullish market, the Bot uses three trading units to buy BTC when a buying opportunity is detected, using 150 USDT. In contrast, in a bearish market, the Bot executes the purchase of BTC using only a single trading unit.

Trading strategies are developed through market observation. The following sections outline these strategies in detail.

3.2 Buying Strategy

The Bot uses 1-minute timeframe candlestick data to ascertain whether it should execute a buy order, whereas 1-hour timeframe candlestick data is employed to determine the quantity of trading units to be used.

From the previous section, we learned that the current values of the Wavetrend Oscillator are values at the end of the arrays. We assume that the maximum index of the "1-minute timeframe" arrays is i, and the maximum index of the "1-hour timeframe" arrays is j.

The Bot will immediately place a limit order if the following conditions are true:

1. The Bot has not used up all of its allowed trading units.
2. The last buying time is more than 2 minutes ago from the current time. Since the oversold period lasts from 7 to 15 minutes in 1-minute timeframe, the Bot does not purchase excessively during this period. Moreover, it also helps the Bot execute the DCA strategy[2], which provides an opportunity to improve the entry price.
3. Base asset is oversold in the last 5 minutes, which means the inequality below is true:

$$wt1\_1m[i-4:i+1] <= over\_sold \tag{3.3}$$

[2]A famous strategy in financial investment. `https://en.wikipedia.org/wiki/Dollar_cost_averaging`.

Figure 3.2: A buying opportunity is detected on BTCUSDT 1-minute timeframe chart.

4. Base asset has a sign of price growth.

$$dif\_wt\_1m[i-1:i+1] > 0 \tag{3.4}$$

and

$$dif\_wt\_1m[i-4:i-1] <= 0 \tag{3.5}$$

After identifying a buying opportunity, the bot will verify the validity of the inequality below to determine the appropriate number of trading units to use.

$$dif\_wt\_1h[j-1:j+1] > 0 \tag{3.6}$$

If this statement is true, the Bot infers that the market is bullish in the 1-hour timeframe, so it will choose $bullish\_buy$ trading units; otherwise, it will use $bearish\_buy$ units.

The Bot places a FOK limit order[3] with a price 0.05% higher than the market price to increase the chance of order fill. The figure 3.2 is a description of a buy opportunity on BTCUSDT 1-minute timeframe chart.

[3]Order is immediately canceled if not completely filled.

3.3 Selling Strategy

The Bot uses the Wavetrend Oscillator to detect if the current period is suitable for selling. The current time is considered as a selling opportunity if the following conditions are met:

1. Base asset is overbought in the last 5 minutes, which means the inequality below is true:

$$wt1\_1m[i-4:i+1] >= over\_bought \tag{3.7}$$

2. Base asset has a sign of price decrease.

$$dif\_wt\_1m[i-1:i+1] < 0 \tag{3.8}$$

and

$$dif\_wt\_1m[i-4:i-1] >= 0 \tag{3.9}$$

After detecting a selling opportunity, the Bot will select from the orders it previously bought and choose the ones that satisfy a 0.5% profit margin to execute the sell order.

Due to the high volatility of the crypto market, sometimes shortly after the bot places a buy order, the price increases sharply. Because this time period is relatively short, the Wavetrend indicator has not yet reached the overbought level, so the Bot cannot identify selling opportunities when using the indicator. Therefore, the Bot will analyze exceptions within the past 100 minutes. The Bot continuously scans through the previously purchased orders within a maximum time frame of 100 minutes from the current moment. If the Bot sees that the market is in a bearish condition in the 1-hour timeframe, it only checks whether the order satisfied a 0.5% profit and places a sell order. On the contrary, the Bot will check whether the order meets the profit requirement based on the time elapsed from when the order was filled to the current time. (Table 3.1).

The Bot sets a limit order using the Fill-Or-Kill (FOK) option, where the price is set at 0.05% below the current market price, in order to increase the probability of the order being executed. Figure 3.3 describes a selling opportunity on BTCUSDT 1-minute timeframe chart.

Figure 3.3: A selling opportunity is detected on BTCUSDT 1-minute timeframe chart.

Table 3.1: Profit requirements based on elapsed time when analyzing market exceptions

Elapsed time	**Profit**
0 - 20 minutes	1.1%
20 - 40 minutes	1.4%
40 - 60 minutes	1.7%
60 - 80 minutes	2.0%
80 - 100 minutes	2.3%

Chapter 4

System Development

4.1 Requirements Analysis

In this section, I will expound upon the functional and non-functional requirements of the system. Functional requirements entail a detailed explanation of the system's expected behaviors, including specific functions that the system must perform. Non-functional requirements, on the other hand, encompass a wide range of parameters that the system must adhere to in terms of performance, security, usability, reliability, and other key attributes.

4.1.1 Functional Requirements

The functional specifications of the Bot will be detailed in the subsequent sections.

API Integration

API integration allows the trading bot to interact with the Binance exchange programmatically, providing access to a wide range of functions and data, including prices, order books, account information, and more. This enables the trading bot to make informed trading decisions based on real-time market data and execute trades automatically according to pre-defined rules and parameters. From there, I have summarized this requirement into the two points:

- The system should integrate with the Binance API to interact with the Binance

Exchange and access market data, including real-time prices, order book, and historical data.

- The system should be able to authenticate with the Binance API using API key and secret key to securely access user's trading account.

Telegram Admin Authentication

The functional requirement for Telegram Admin Authentication, stating that the system should only allow whitelisted Telegram accounts to interact with it, is an important security measure for a Telegram bot. This requirement ensures that only authorized users with approved Telegram accounts are allowed to interact with the bot, helping to prevent unauthorized access or malicious attacks. Therefore, summarizing, we have the following functional requirement:

- The system should only allow whitelisted Telegram accounts to interact with it.

Trade Execution

The functional requirement for trade execution involves the ability of the system to autonomously execute buy and sell orders on the Binance exchange based on pre-defined trading strategies. This is a critical component of building an automated trading bot as it allows the system to make informed and rapid trading decisions based on real-time market data and predefined trading strategies. To summarize, the following requirement is presented:

- The system should be able to autonomously execute buy and sell orders on the Binance exchange based on pre-defined trading strategies.

Wavetrend Oscillator Calculation

The Wavetrend Oscillator indicator is an important factor in the system, as it is the data source that the system relies on to make trading decisions. Therefore, we have the following functional requirement:

- The system should continuously calculate Wavetrend Oscillator to ensure that the values are up-to-date.

Data Storage

The significance of data storage in the system lies in the fact that the decision-making process for trading and system management is completely reliant on the data storage layer. As a result, there are two requirements that need to be fulfilled:

- The system should store data in a structured and organized manner, adhering to defined data models.
- The system should ensure data integrity and security, and provide mechanisms for data retrieval, update, and deletion.

Portfolio Management

Portfolio is a very important factor in trading, as it can provide many insights to make investment decisions. For example, we can increase the capital for a trading pair that are generating good profits and conversely stop trading for pairs that are not performing well. Thus, we have the following two requirements:

- The system should provide portfolio management features, such as tracking and displaying the current portfolio holdings, balances, profit, and loss.
- The system should provide the administrator with the ability to modify the portfolio positions.

Reporting and Logging

Having system debugging support is crucial during software development. With an effective strategy, the process of identifying the error's root cause can be significantly expedited and require minimal effort. I have established three functional requirements for the system as follows:

- The system should regularly generate health reports.

- The system should allow the admin to retrieve the current Wavetrend Oscillator values to ensure the correctness of the calculations.
- The system should provide structured logging mechanisms to help with system troubleshooting, monitoring, and analysis.

4.1.2 Non-functional Requirements

Below are the non-functional requirements outlined for the system:

- **Performance**: the system should be able to process a large number of trade orders within a short timeframe, ensuring low latency and minimal downtime.
- **Security**: the system should implement robust security measures, such as secure communication protocols, and authentication mechanisms, to protect against unauthorized access, data breaches, and other security risks.
- **Reliability**: the system should be reliable and stable, with minimal system failures, errors, or crashes, ensuring continuous and uninterrupted trading operations.
- **Maintainability**: the system should be easy to maintain, with well-organized and documented code, modular design, and appropriate logging and monitoring capabilities to facilitate debugging and troubleshooting.
- **Documentation**: the system should be well-documented, including user manuals, and technical documentation to facilitate understanding, maintenance, and troubleshooting.

4.2 System Design

4.2.1 Entity Relationship Diagram

The system has a data model consisting of 3 tables. (Figure 4.1)

Table trading_workers

The table stores the configurations of the currently active trading workers. Attributes of the records in table:

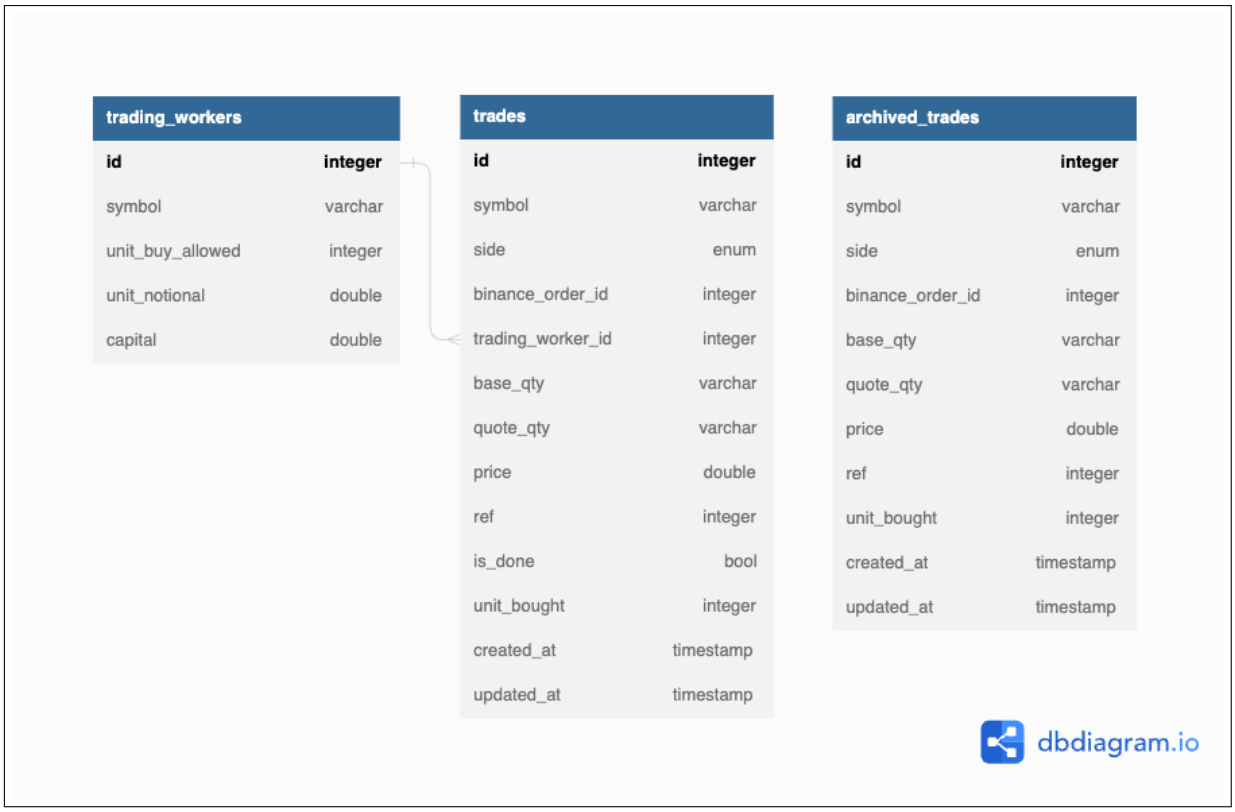

Figure 4.1: Entity Relationship Diagram.

- **id**: The identity of the trading workers.
- **symbol**: The trading pair symbols, such as BTCUSDT, ETHUSDT, etc.
- **unit_buy_allowed**: The number of trading units that can be utilized for trading purposes. (3.1)
- **unit_notional**: The valuation of a trading unit is expressed as the dollar amount utilized to acquire the base currency. (3.1)
- **capital**: The USD value capital allocated for trading.

Table trades

The table stores the trades of trading workers. The record attribute set contains:

- **id**: The identity of the trade.
- **symbol**: The trading pair symbols, such as BTCUSDT, ETHUSDT, etc.

- **side**: The side of the order, which can be either BUY or SELL.
- **binance_order_id**: The identity of the limit order on Binance Exchange.
- **trading_worker_id**: The identity of trading worker.
- **base_qty**: The base quantity that the trading worker has bought or sold.
- **quote_qty**: The quote quantity that the trading worker has bought or sold.
- **price**: The price of the base currency that is used to create the trade.
- **ref**: The SELL trade refers to "id" of the BUY trade.
- **is_done**: This field is always set to true for SELL trades. For BUY trades, this value is set to true if there exists a SELL trade that has sold a quantity of base currency bought by this trade.
- **unit_bought**: The number of trading units that have been used by the trade.
- **created_at**: The timestamp at which a record has been created.
- **updated_at**: The timestamp at which a record has been updated.

Table archived_trades

Sometimes, we want to run the Bot for a trading pair, yet historical trading data of this pair is still present as the Bot has previously run for that pair. We will archive the old data for the purpose of data analysis. Old trading data in the **trades** table will be moved to the **archived_trades** table. The attributes in this table are a subset of those in the **trades** table, so no further explanation is needed.

4.2.2 Architecture

Firstly, let me introduce the components of the system (Figure 4.2), and further details about the system's flows will be provided in the next section.

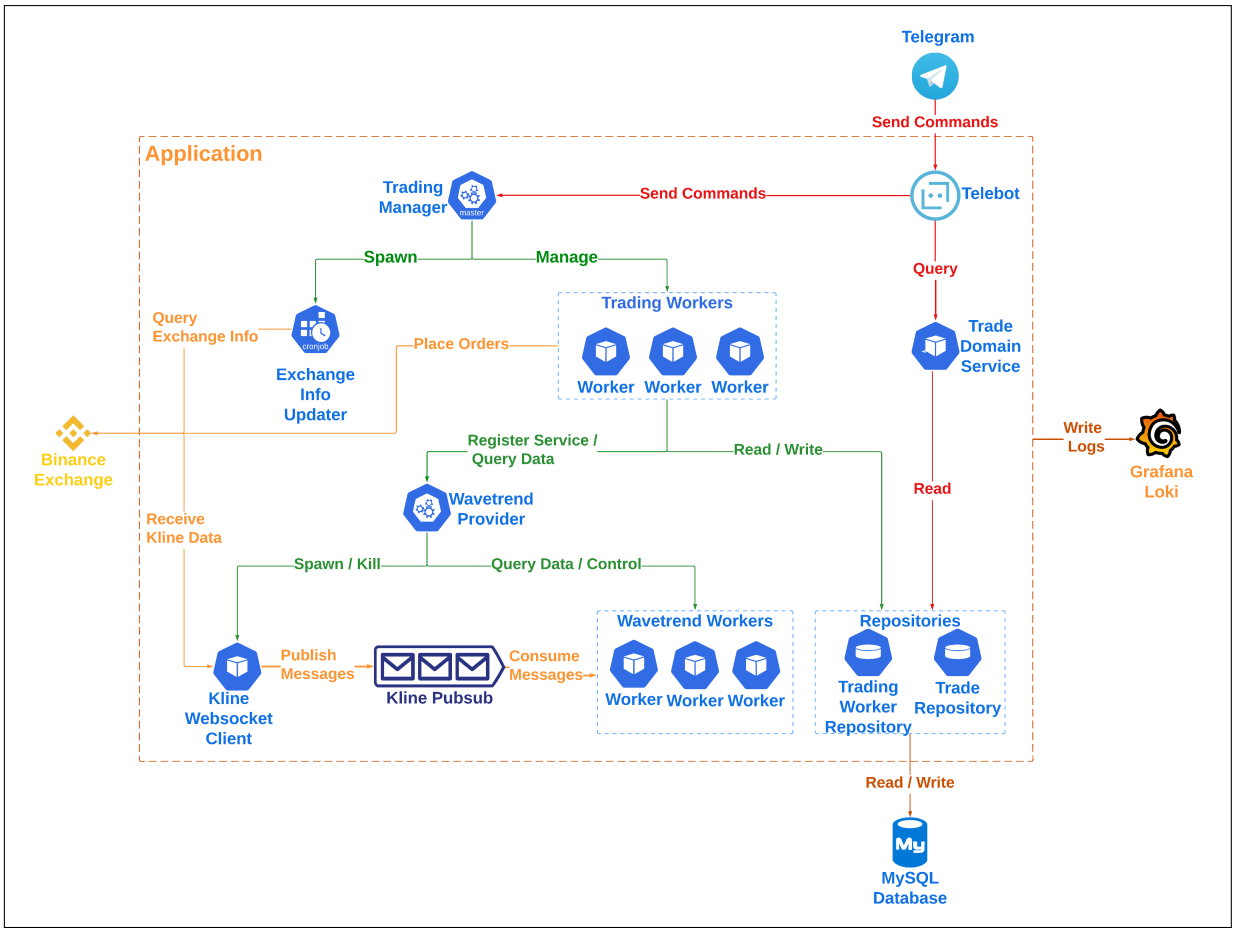

Figure 4.2: System architecture.

Telebot

Telebot is an integral component of the system that is designed to facilitate communication between the admin and the trading system. It is implemented as a goroutine, a lightweight thread that is scheduled and managed independently by the Go runtime environment. The Telebot goroutine is always active and continuously listens to incoming messages and updates from the Telegram server via a persistent websocket connection. By leveraging the Telegram Bot API, Telebot enables seamless integration between the system and Telegram, allowing the admin to interact with the system in a secure and efficient manner.

Domain Service

In the automated trading bot on Binance exchange, the domain service component plays a crucial role in providing various services for the Telebot component. Essentially, the Telebot component leverages the services offered by the domain service component to efficiently manage the logic of data in the data storage layer. Some of the primary services that are provided include the ability to query portfolios, archive trading data, and carry out other important functions related to the management of data.

Repository

The repository component within the system is specifically designed to manage the persistence and retrieval of data from the database. Its key function is to act as a mediator between the database and the domain model, providing a well-organized and standardized interface for storing and retrieving data. Through its effective management of data persistence and retrieval, the repository component helps to ensure the system's reliability, efficiency, and scalability.

Trading Manager

The Trading Manager component in the system is designed to manage Trading Workers, which involves monitoring and coordinating the various aspects of trading activities. It provides a range of features for the Telebot, such as the ability to initiate or terminate a trading worker, as well as monitor its current status, and so on. In this way, the Trading Manager plays a critical role in ensuring that the trading system functions efficiently and effectively. Through its comprehensive suite of features and capabilities, the Trading Manager helps to ensure that trading operations are streamlined and optimized, enabling the admin to make more informed decisions and maximize profits.

Trading Worker

The Trading Worker component is a crucial module of the automated trading bot system that is responsible for implementing trading strategies and executing orders on the Binance exchange. It is designed to handle the complex process of trading with speed and accuracy, leveraging sophisticated algorithms and strategies to make informed and prof-

itable trading decisions. To ensure maximum efficiency and performance, the Trading Worker is designed to operate as a goroutine, allowing it to run independently and avoid interfering with other components.

Exchange Info Updater

The Exchange Info Updater component of the system is implemented as a goroutine that performs the periodic update of information for various trading pairs, which includes data such as the precision of quantity and price, minimum notional values, and other related details. Its main function is to ensure that the system has access to the most up-to-date and accurate information.

Wavetrend Provider

The Wavetrend Provider is a helpful component for Trading Workers. Trading Workers can register to use the wavetrend indicator for a specific trading pair and timeframe. Afterward, Trading Workers can query indicator values through the Wavetrend Provider. The Wavetrend Provider also manages the Wavetrend Workers.

Kline WebSocket Client

Kline WebSocket Client is responsible for initiating the WebSocket connection to a Binance WebSocket server, receiving Kline data in real-time, and handling the opening, closing, and error events of the WebSocket connection. Kline data received from WebSocket connection will be published to Kline PubSub.

Kline PubSub

The Kline PubSub component in the system is responsible for publishing the Kline data received from the Kline WebSocket Client to subscribers. The component is implemented using Go channel, which allows for an efficient and high-performance way of sending and receiving data. When the Kline WebSocket Client receives new Kline data, it forwards it to the Kline PubSub component, which then broadcasts it to all the subscribers who have subscribed to the Kline data.

Wavetrend Worker

The Wavetrend Worker component in the system is designed to subscribe to the Kline PubSub and continually calculate the Wavetrend Oscillator indicator values. These values are then made available to the Trading Workers to make informed and accurate trading decisions. The Wavetrend Worker component employs advanced algorithms to ensure that the calculations are precise and reliable, providing the Trading Workers with real-time data to help them make optimal trading choices. With its ability to provide high-quality data and analysis, the Wavetrend Worker component plays a vital role in the successful operation of the trading system.

4.2.3 Sequence Flows

In order to elucidate the interdependencies and interactions among the components within the system, sequence flows will be delineated below. In order to be clearer, the names of the components in the system will start with "Bot:".

Admin Starts A Trading Worker

In this section, I would like to describe the interaction between components in the system when the admin executes the command to start trading for a trading pair. (Figure 4.3).

Step 1: Admin sends a message using **Telegram chat**.

Step 2: Telegram Server sends the message to **Bot:Telebot**.

Step 3: Bot:Telebot calls **Bot:Trading Manager** to create a new trading worker.

Step 4: **Bot:Trading Manager** creates a new trading worker.

Step 5: Bot:Trading Manager requires **Bot:Trading Worker** to start running.

Step 6: Bot:Trading Worker registers symbol and timeframe to **Bot:Wavetrend Provider**.

Step 7: Bot:Wavetrend Provider spawns new goroutine named **Kline WebSocket Client**.

Step 8: Bot:Wavetrend Provider creates new wavetrend worker.

Step 9: Bot:Wavetrend Provider requires **Bot:Wavetrend Worker** to start run-

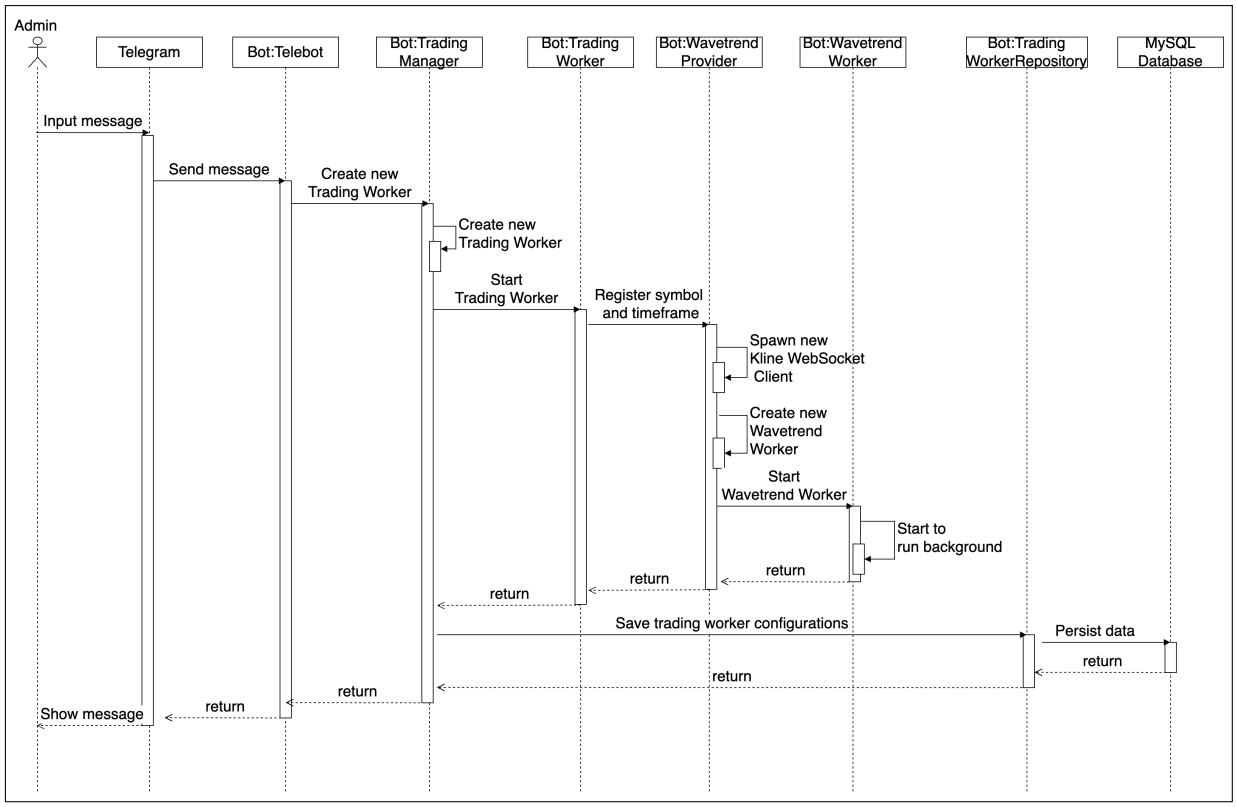

Figure 4.3: Admin starts a trading worker sequence diagram.

ning.

Step 10: Bot:Wavetrend Worker starts calculating Wavetrend Oscillator Indicator values in the background.

Step 11: Bot:Trading Manager calls **Bot:Trading Worker Repository** to save configurations of the trading worker.

Step 12: **Bot:Trading Worker Repository** persists data to **MySQL Database**.

Step 13: The result of the trading worker creation process is sent back to the **Admin**.

An example of an interaction with the Bot on Telegram Chat UI can be shown on Figure 4.4.

Admin Checks Health Of Trading Workers

Essentially, the main logic of a **Bot:Trading Worker** is a loop, in which a flag variable is updated every 30 minutes. We will check the most recent time the variable was updated.

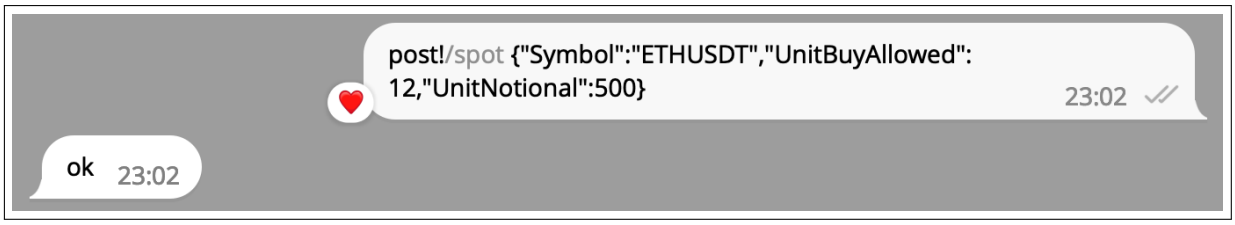

Figure 4.4: Admin starts a trading worker.

If it has not been updated for more than 30 minutes, we know that the worker's loop has encountered a problem. The detailed flow of operation is shown in Figure 4.5.

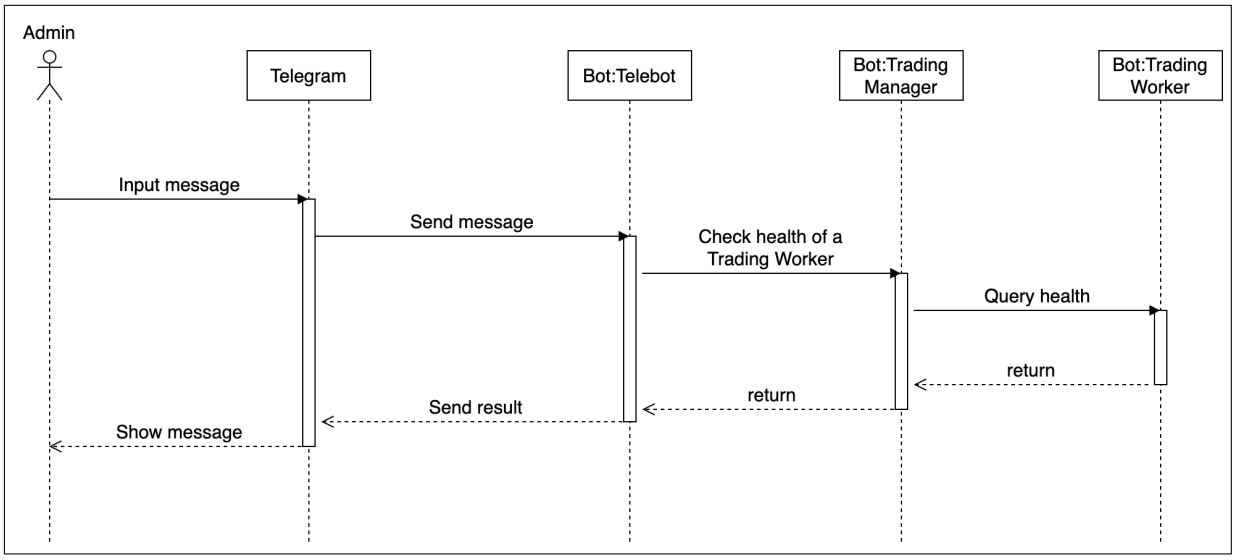

Figure 4.5: Admin checks health of trading workers sequence diagram.

Step 1: Admin sends a message using **Telegram chat**.

Step 2: Telegram Server sends the message to **Bot:Telebot**.

Step 3: Bot:Telebot calls **Bot:Trading Manager** to get heath of trading workers.

Step 4: **Bot:Trading Manager** calls all **Bot:Trading Worker** to get health.

Step 5: The health of trading workers is sent back to the **Admin**.

An example of an admin checking the health of the Bot is shown in Figure 4.6.

Admin Adds Capital For A Trading Worker

The system supports the admin in adding capitals for a trading pair. The workflow includes 8 steps as shown in Figure 4.7.

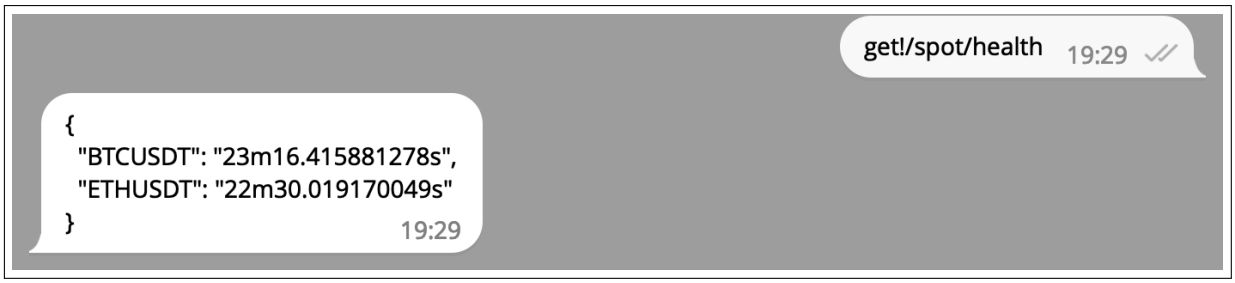

Figure 4.6: Admin checks health of trading workers

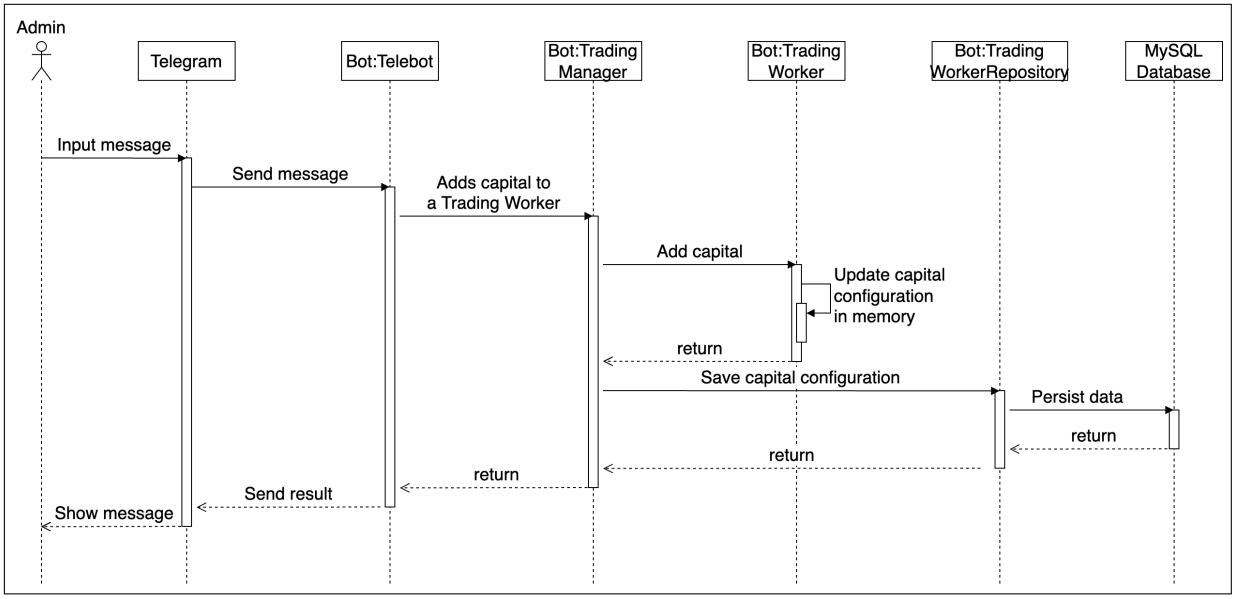

Figure 4.7: Admin adds capital for a trading worker sequence diagram.

Step 1: Admin sends a message using **Telegram chat**.

Step 2: Telegram Server sends the message to **Bot:Telebot**.

Step 3: Bot:Telebot calls **Bot:Trading Manager** to add capital for a trading worker.

Step 4: Bot:Trading Manager calls **Bot:Trading Worker** for updating capital configuration.

Step 5: Bot:Trading Worker updates capital configuration in memory.

Step 6: Bot:Trading Manager calls **Bot:Trading Worker Repository** to update data in database.

Step 7: Bot:Trading Worker Repository persists data into database.

Step 8: The result of the adding capital process is sent back to the **Admin**.

The action of the admin adding capital is illustrated in Figure 4.8.

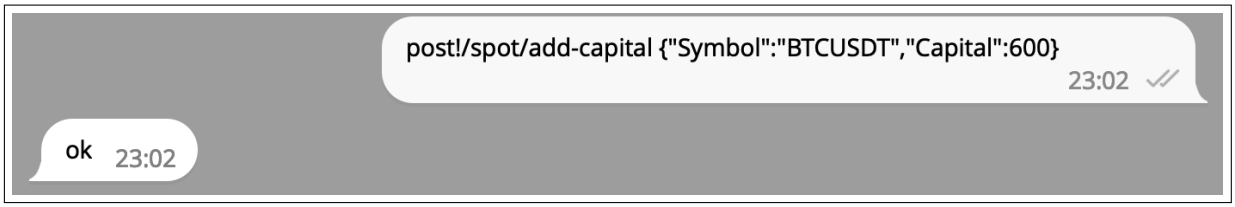

Figure 4.8: Admin adds capital for a trading worker.

Admin Queries Portfolio

The admin will need to know about their current portfolio. The data statistics flow is represented in Figure 4.9.

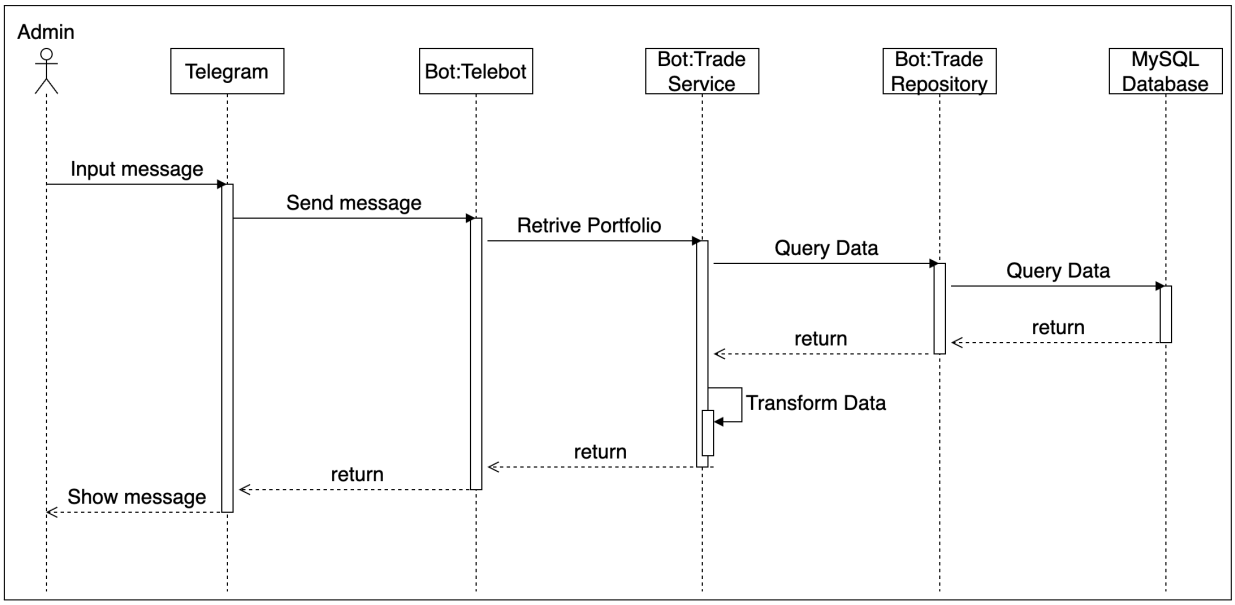

Figure 4.9: Admin queries portfolio sequence diagram.

Step 1: Admin sends a message using **Telegram chat**.

Step 2: Telegram Server sends the message to **Bot:Telebot**.

Step 3: Bot:Telegram calls **Bot:Trade Service** to retrieve portfolio.

Step 4: Bot:Trade Service queries data by calling **Bot:Trade Repository**.

Step 5: Bot:Trade Repository queries data in **MySQL Database**.

Step 6: Bot:Trade Service transforms data.

Step 7: The portfolio is sent back to the **Admin**.

Figure 4.10 simulates the admin querying the portfolio.

Figure 4.10: Admin queries portfolio.

Admin Stops A Trading Worker

Figure 4.11 demonstrates the process of sending a command to stop the trading worker for a trading pair by the admin.

Step 1: Admin sends a message using **Telegram chat**.

Step 2: Telegram Server sends the message to **Bot:Telebot**.

Step 3: Bot:Telebot calls **Bot:Trading Manager** to stop a trading worker.

Step 4: Bot:Trading Manager sends a signal to the **Bot:Trading Worker** to stop

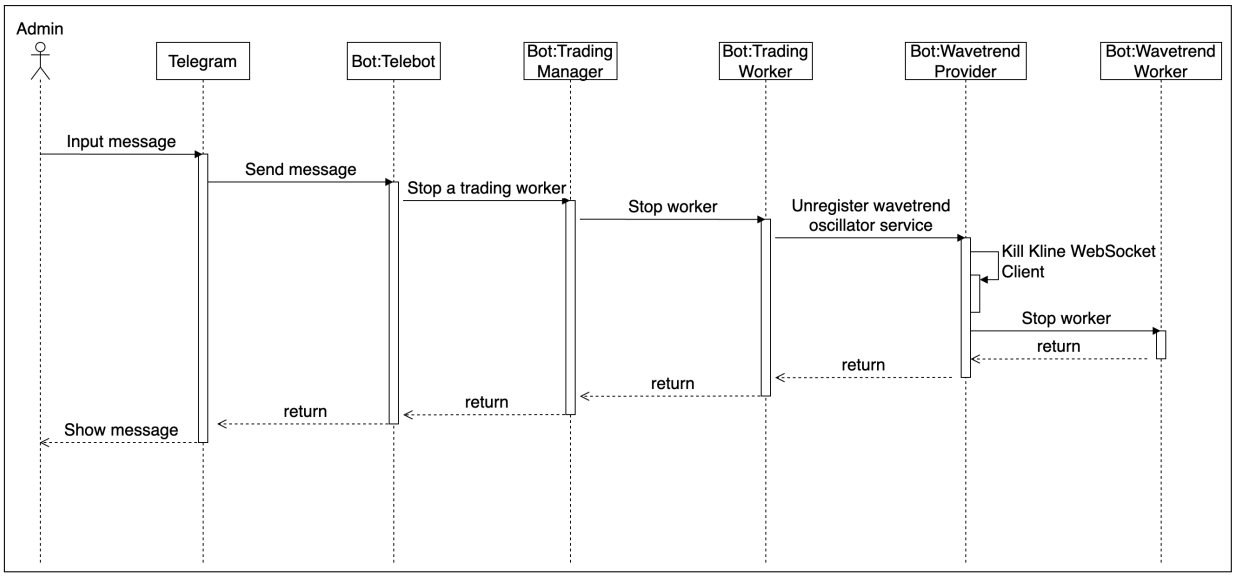

Figure 4.11: Admin stops a trading worker sequence diagram.

running.

Step 5: Bot:Trading Worker unregisters wavetrend oscillator service of **Bot:Wavetrend Provider**.

Step 6: Bot:Wavetrend Provider kills **Kline WebSocket Client** goroutine.

Step 7: Bot:Wavetrend Provider sends signal to stop **Bot:Wavetrend Worker**.

Step 8: The result of the stopping a trading worker process is sent back to the **Admin**.

An example the admin stopping a trading worker on Telegram Chat UI can be shown on Figure 4.12.

Figure 4.12: Admin stops a trading worker.

Admin Archives Trading Data

Figure 4.13 illustrates the interaction flow between system components when the admin sends a command to archive data.

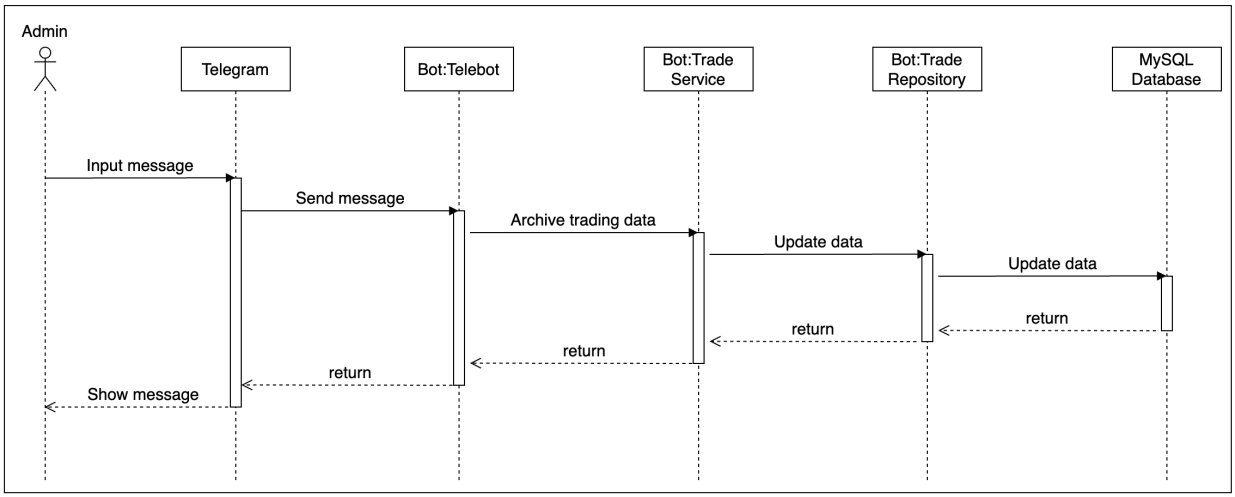

Figure 4.13: Admin archives trading data sequence diagram.

Step 1: Admin sends a message using **Telegram chat**.

Step 2: Telegram Server sends the message to **Bot:Telebot**.

Step 3: Bot:Telebot calls **Bot:Trade Service** to archive trading data.

Step 4: Bot:Trade Service calls **Bot:Trade Repository** to update trading data in database.

Step 5: Bot:Trade Repository updates database.

Step 6: The result of the archiving trading data process is sent back to the **Admin**.

An example of the admin archiving trading data can be seen in Figure 4.14.

Figure 4.14: Admin archives trading data.

Admin Queries Wavetrend Oscillator Values

This procedure is utilized to keep track of the precision of Wavetrend Indicator calculations, ensuring that the results are highly accurate. (Figure 4.15).

Step 1: Admin sends a message using **Telegram chat**.

Step 2: Telegram Server sends the message to **Bot:Telebot**.

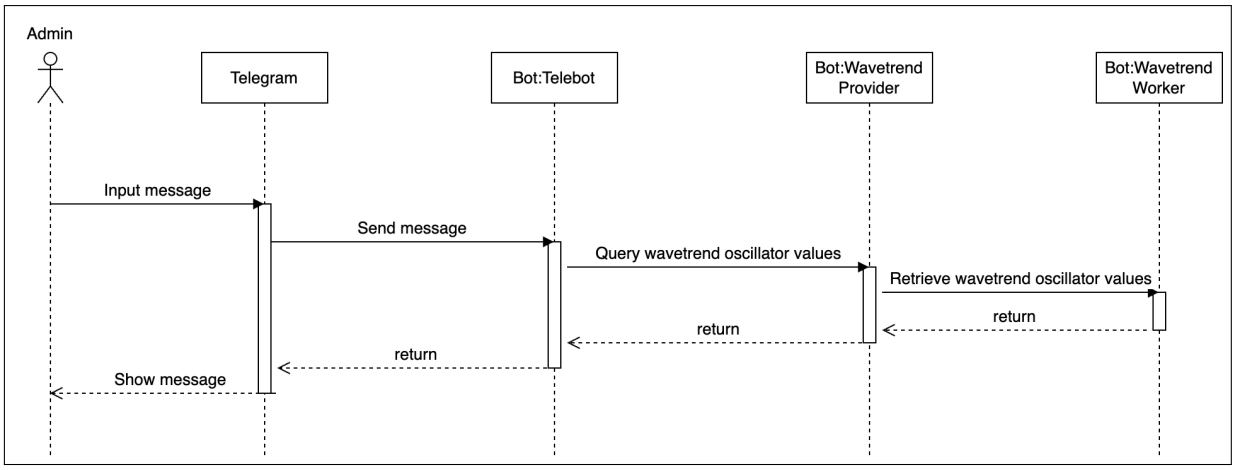

Figure 4.15: Admin queries wavetrend oscillator values sequence diagram.

Step 3: Bot:Telebot calls **Bot:Wavetrend Provider** to retrieve wavetrend oscillator indicator values.

Step 4: Bot:Wavetrend Provider queries data of **Bot:Wavetrend Worker**.

Step 5: The result is sent back to the **Admin**.

Figure 4.16 is an example of the admin querying Wavetrend Oscillator indicator values.

Trading Workers Run In Background

Trading Workers run in a loop of 7 steps (Figure 4.17).

Step 1: Bot:Trading Worker queries **Bot:Wavetrend Provider** wavetrend oscillator indicator values.

Step 2: Bot:Wavetrend Provider retrieve calculated data of **Bot:Wavetrend Worker**.

Step 3: Bot:Trading Worker analyzes data to make trading decision.

Step 4: If data shows a trading opportunity, **Bot:Trading Worker** places a limit order on Binance Exchange by calling API.

Step 5: If the order is executed successfully, **Bot:Trading Worker** updates worker state in memory.

Step 6: If the order is executed successfully, **Bot:Trading Worker** calls **Bot:Trade Repository** to persist order data.

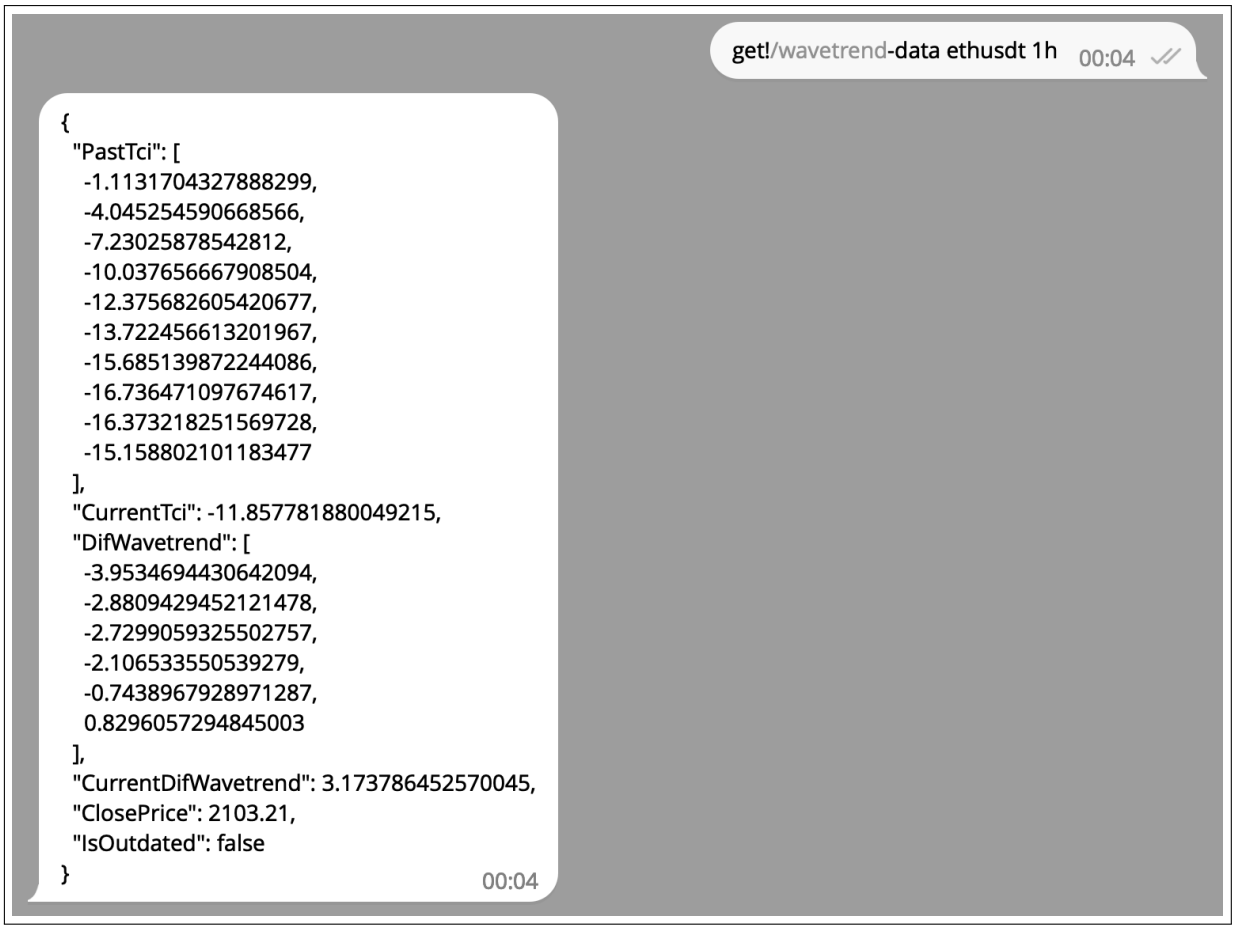

Figure 4.16: Admin queries wavetrend oscillator values.

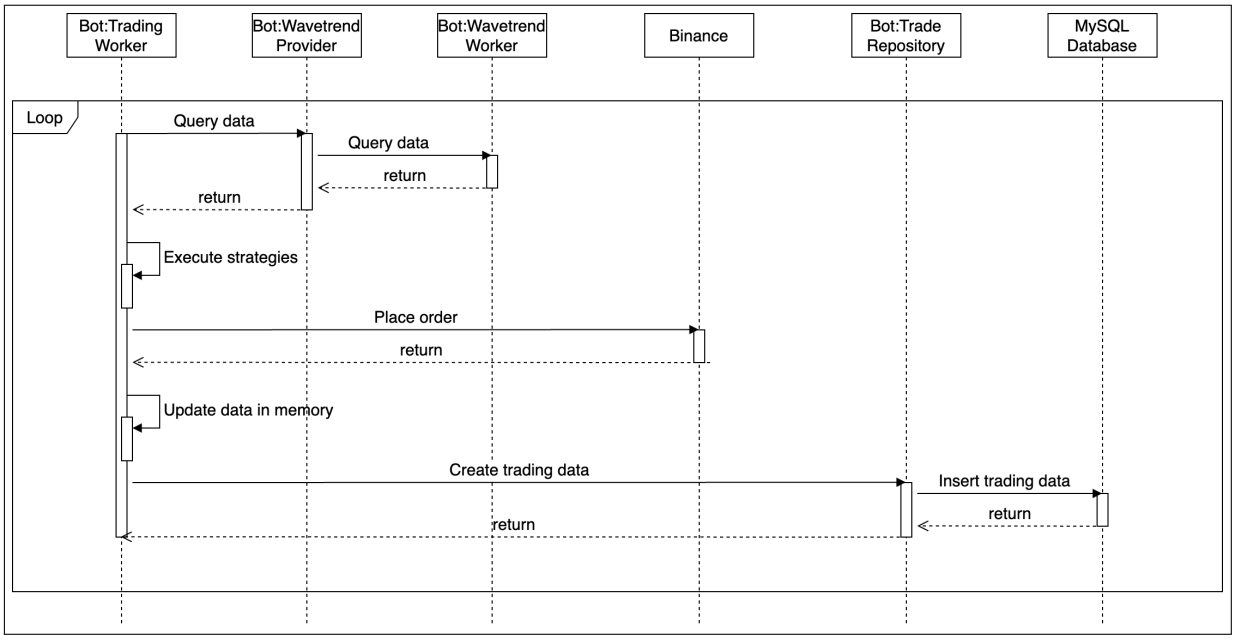

Figure 4.17: Trading Workers run in background.

Step 7: **Bot:Trade Repository** inserts a record in table **trades**.

Updating Exchange Info In Background

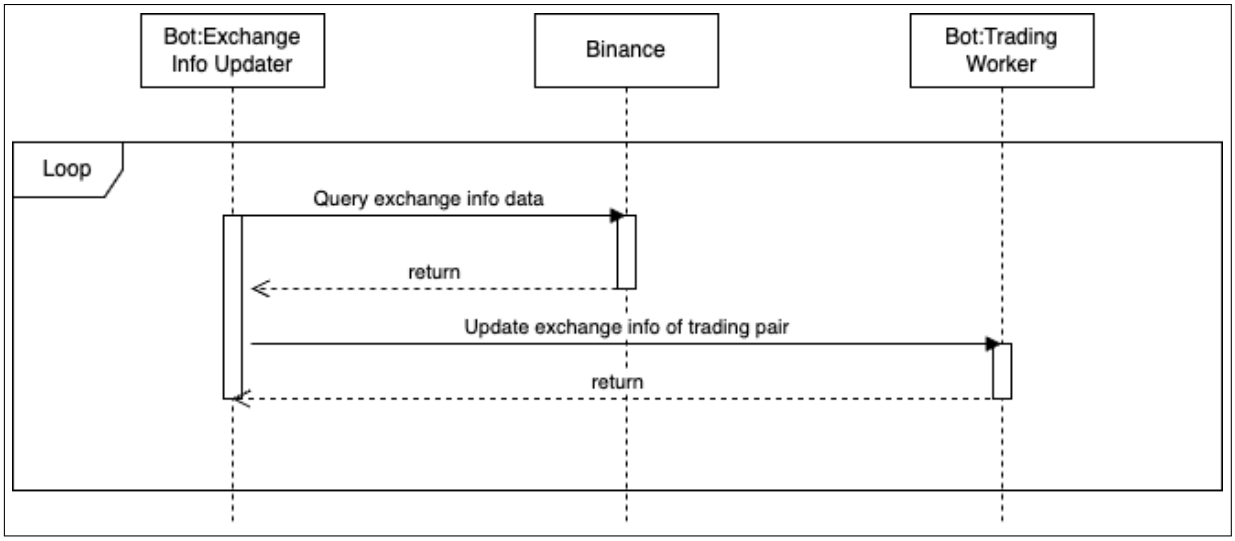

Figure 4.18: Updating exchange info in background sequence diagram.

Exchange Info Updater run in a loop of 2 steps as described in Figure 4.18.

Step 1: Bot:Exchange Info Updater queries exchange info data by calling Binance API.

Step 2: Bot:Exchange Info Updater calls **Bot:Trading Worker** to update exchange info in memory of its trading pair.

Calculating Wavetrend Oscillator Values In Background

Another crucial workflow in the system is the calculation of Wavetrend Oscillator indicator values, which is illustrated in Figure 4.19.

Step 1: Bot:Kline WebSocket Client receives Kline data in real time from Binance.

Step 2: Bot:Kline WebSocket Client publishes Kline data to **Kline PubSub**.

Step 3: Bot:Wavetrend Worker consumes Kline data from **Kline PubSub**.

Step 4: Bot:Wavetrend Worker updates wavetrend oscillator values in memory.

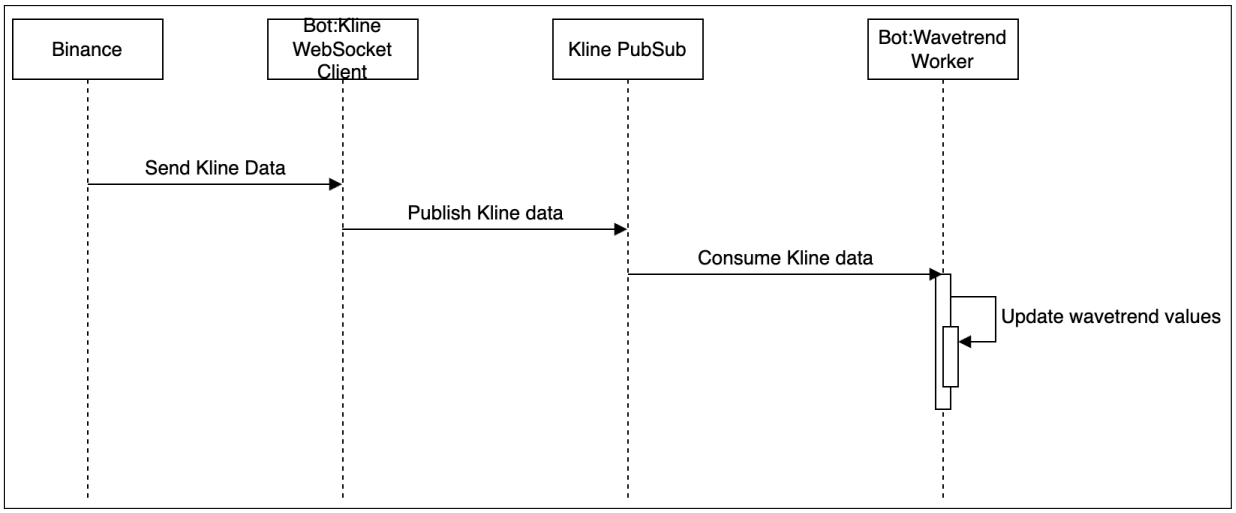

Figure 4.19: Calculating Wavetrend Oscillator indicator values in background sequence diagram.

4.3 System Implementations

In system implementation, the use of patterns can provide numerous benefits, such as improving system design, reducing complexity, and promoting reusability. Patterns are commonly used to solve recurring design problems and provide a shared vocabulary for communication among software developers. In the following sections, we will explore the use of patterns in system implementation in greater detail.

4.3.1 Dependency Injection

One key design pattern utilized in the system is dependency injection [27], which helps improve the flexibility, modularity, and testability of the system.

Dependency injection is a design pattern that allows for the decoupling of dependencies from the components that use them, by providing a way to manage and inject dependencies into components at runtime.

To abstract the behaviors that these dependencies must adhere to, I have defined interfaces. I have used constructor injection as the main technique for dependency injection in the system, where dependencies are injected through constructor parameters. This allows for clear and explicit dependencies in the codebase, making it easier to understand and test.

The use of dependency injection has provided several benefits to the trading bot.

It has improved the flexibility of the system by allowing developers to easily swap out different implementations of dependencies without modifying the core logic of the bot. It has also enhanced modularity by decoupling dependencies from the components that use them, making it easier to maintain and test individual components. Additionally, it has increased testability by facilitating the use of mock objects or test doubles during unit testing.

4.3.2 Singleton Pattern

I have also utilized the singleton design pattern [13] in the system. The singleton pattern ensures that a Go struct has only one instance and provides a global point of access to that instance, which can be beneficial in certain scenarios where a single instance of a struct needs to coordinate actions or manage resources across the system.

In the system, I have identified specific components that should have only one instance throughout the lifespan of the application, and have implemented them as singletons, such as “Trading Manager”, “Wavetrend Provider”, etc.

The use of singleton pattern in the trading system has provided several benefits. It has ensured that certain components are accessed consistently and manage resources effectively, preventing any conflicts or inconsistencies in the system configuration. It has also improved performance by reducing the overhead of creating multiple instances of the same type, as well as providing a global point of access to the single instance, simplifying the usage and management of these components in our system.

4.3.3 Repository Pattern

I have implemented the repository pattern [18] to provide a structured and organized approach for managing data persistence and interaction with external APIs. The repository pattern is a popular design pattern used in software development to separate the concerns of data access and manipulation from the rest of the application logic.

In the system, we have identified the need for managing various data entities, such as trading worker, trade, and trade history data. Instead of directly coupling these data entities with the business logic, I have implemented separate repository classes to encapsulate the data access and manipulation operations. The repository classes act as intermediaries between the data entities and the rest of the application logic, providing

an abstraction layer that decouples the details of data persistence and retrieval from the business logic.

4.3.4 Scheduler Pattern

I have utilized the Golang ticker in the trading bot as a means of scheduling periodic tasks [23]. The ticker is a built-in feature provided by the "time" package in Go, which allows us to create a repeating timer that fires at regular intervals.

I have used the ticker to schedule tasks such as retrieving exchange info, and executing trading strategies to place orders.

The ticker in Go works by sending a value to a channel at specified intervals, which triggers the associated task or action in the code. We can set the desired interval for the ticker using the *time.NewTicker* function, which returns a *Ticker* value. We can then use the *Ticker.C* channel to receive the ticker events and execute our scheduled tasks.

The ticker provides us with a simple and effective way to schedule periodic tasks in the trading bot, allowing us to automate repetitive actions and ensure timely and accurate execution of tasks based on our desired intervals. It helps me improve the efficiency of the system.

4.3.5 Pub-Sub Pattern

I have used the PubSub pattern [29] to calculate the values of the WaveTrend oscillator indicator. After receiving the values from the Binance WebSocket server, the Kline WebSocket client will push the kline values into a queue implemented using channels. The WaveTrend worker will listen to this queue and retrieve the values to continuously calculate and update the latest value of the indicator.

Maintaining the order of data is crucial when performing calculations that require a specific sequence of inputs. By pushing the data into a queue, we can ensure that the data is processed in the correct order, which can help guarantee the accuracy of the calculation results. This is particularly important in systems where calculations rely on historical or real-time data streams, as out-of-order data can lead to inaccurate or unreliable results. Therefore, using a queue to maintain the order of the data can be an effective way to ensure the integrity of the calculations and prevent errors from occurring.

4.3.6 Mutual Exclusion

In order to ensure thread safety and prevent race conditions in the system, I have utilized mutex locks [17]. A mutex is a synchronization primitive that provides exclusive access to a shared resource, allowing threads or processes to safely access and modify the shared resource without conflicts.

To ensure thread safety and prevent race conditions in the system, I have utilized Go's built-in mutex lock. Go provides a "sync" package that includes a "Mutex" type, which is a synchronization primitive used for exclusive access to shared resources.

In the system, I have identified critical sections of the code where multiple goroutines may concurrently access or modify shared resources, such as updating trading worker status and settings, etc. To prevent race conditions and maintain data integrity, I have used the Mutex type to enforce mutual exclusion, allowing only one goroutine at a time to access the shared resource.

The usage of Go's Mutex in the system follows a standard pattern. I create a Mutex object for each shared resource that needs to be protected, and I use its *Lock()* and *Unlock()* methods to lock and unlock the shared resource whenever it is accessed or modified. When a goroutine acquires a Mutex lock using the *Lock()* method, other goroutines that try to acquire the same lock will be blocked until the lock is released by the owning goroutine. This ensures that only one goroutine at a time can access the shared resource, preventing race conditions.

The use of Go's Mutex in the trading bot has provided effective synchronization and thread safety, preventing race conditions and maintaining data integrity in our system. However, it's important to note that the usage of Mutex should be carefully planned and implemented, as improper use can lead to issues such as deadlocks, contention, or performance degradation. I have thoroughly tested and optimized the usage of Mutex to ensure efficient and reliable synchronization in the trading bot.

4.3.7 Synchronization

Utilizing channels to control the flow of execution in the system can help ensure that different parts of the bot are coordinated and synchronized properly [2]. Channels provide a way for goroutines to communicate with each other and exchange data in a concurrent and safe manner.

For example, I use channels to signal when a specific event or condition has occurred, and other goroutines can block and wait for this signal before proceeding. This can help ensure that certain actions are executed in a specific order or when certain conditions are met.

Using channels to control the flow of execution can help ensure that different parts of the trading bot are synchronized and coordinated, and can prevent race conditions or other concurrency-related issues.

4.3.8 Retry Mechanism

The constant backoff retry mechanism [19] is implemented in the project to handle failures or errors that may occur during certain operations, such as sending Telegram messages to admin or saving trading data into database. This mechanism allows the system to automatically retry the operation a fixed number of times with a constant delay between retries, in case of failures, before giving up and raising an error.

4.4 System Testing

To ensure the reliability and accuracy of the software, I have run it on the development environment for an extended period of time to observe system logs. In addition, I have also implemented unit tests for the system. A comprehensive suite of unit tests has been implemented to ensure the correctness and robustness of the codebase. These unit tests are specifically designed to test individual functions and methods that are related to mathematics, such as those involved in calculating indicators, processing trading signals, and making trading decisions.

4.5 System Deployment And Monitoring

4.5.1 Infrastructure

For the deployment of my trading bot, I opted for an e2-small instance on the GCP. The e2-small instance is considered to be a general-purpose machine type that is capable of providing the necessary resources to run the trading bot effectively. It comes with 2 virtual CPUs and 2 GB of memory, which is sufficient for running small-scale trading bots. Despite its compact size, the e2-small instance is highly efficient and cost-effective,

making it an ideal choice for my trading bot deployment. With this instance type, I was able to balance the performance requirements of my trading bot with the cost constraints of running it on a cloud platform.

4.5.2 Setup Docker

To ensure efficient and independent operation of the various components of the system, including the trading bot, MySQL, and Promtail, the decision was made to run each component in a separate container. As a result, it was necessary to install Docker. With Docker, each component can be run independently and be managed easily, enhancing the system's performance and scalability. The process of installing Docker can be completed with great ease by simply following the guidelines and instructions available on the official website of Docker.

4.5.3 Setup MySQL Database

To ensure a smooth deployment of MySQL in the project, I chose to use the Docker Compose tool[1]. Firstly, I created and tested the Docker Compose file on my local machine, and then transferred it to the server for deployment. The file included all the essential configurations required to set up the MySQL container, including the database name that needed to be created and the root password.

Once the Docker Compose files are executed, the MySQL image will be fetched from DockerHub and run with the configurations specified in the file.

4.5.4 Setup Promtail

I also configured Promtail through the use of a Docker Compose file. To enable Promtail to transmit logs to the Loki server, it is necessary to have an API key for authentication purposes. This particular API key can be obtained from the official website of Grafana.

4.5.5 Setup Trading Bot

To streamline the deployment of the trading bot, I constructed a Docker image and uploaded it to my own repository on DockerHub. This involved building the image locally and then pushing it to the repository. On the cloud where the bot was to be run, I logged

[1] Docker Compose is a tool for defining and running multi-container Docker applications.

in to DockerHub and pulled the image from my private repository on DockerHub.

In order to run the trading bot successfully, various configurations are required, including database configurations, Binance API keys, Telegram bot token, and the admin Telegram ID. These configurations are crucial for the bot to access the necessary resources and to interact with the Binance exchange and the Telegram API.

The database configurations include the database host, port, name, username, and password. These configurations are needed for the bot to store and retrieve data from the database. The Binance API keys, on the other hand, are required for the bot to access Binance exchange data and execute trades. The Telegram bot token is necessary for the bot to interact with Telegram API and receive commands from the admin. Lastly, to ensure that only the admin has the authority to interact with the trading bot, it is crucial to provide the bot with the admin Telegram ID.

All of these configurations need to be set up correctly to ensure the trading bot runs smoothly and without any issues.

4.5.6 Metrics And Logs

I monitored the utilization of computational resources of the trading bot using the tools provided on GCP.

According to Figure 4.20, the trading bot utilizes the CPU resources efficiently, and the maximum CPU utilization recorded does not surpass the 15% mark.

Based on the Figure 4.21, it is noticeable that the maximum memory usage of the application is approximately 40%.

Log can be easily queried based on Grafana's query language (Figure 4.22), which allows users to easily search and analyze logs. With Grafana's query language, I can perform complex searches using various filters, such as time range, log level, and specific keywords.

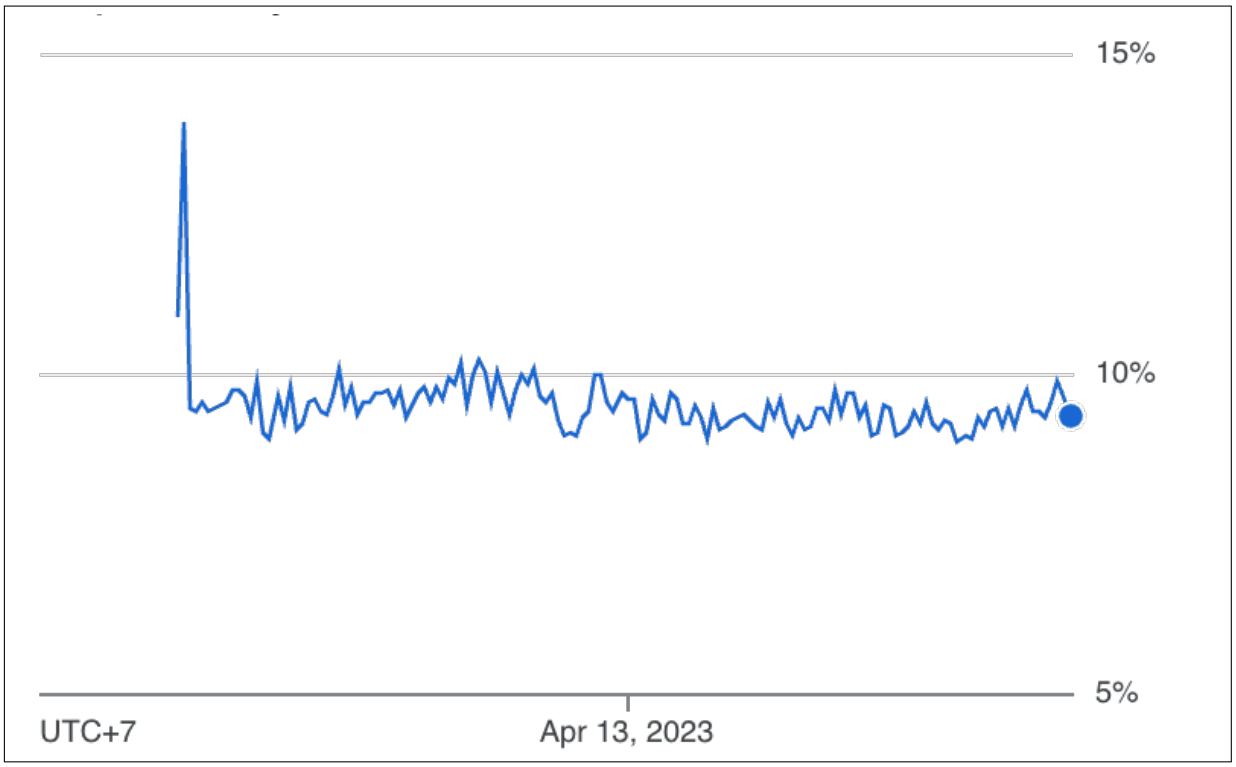

Figure 4.20: CPU utilization.

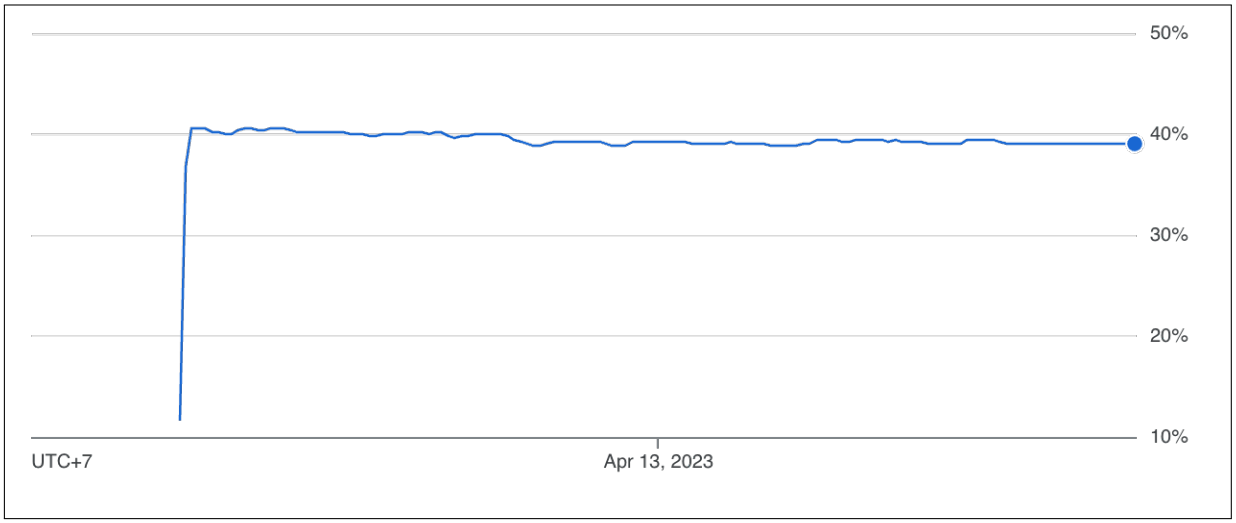

Figure 4.21: Memory utilization.

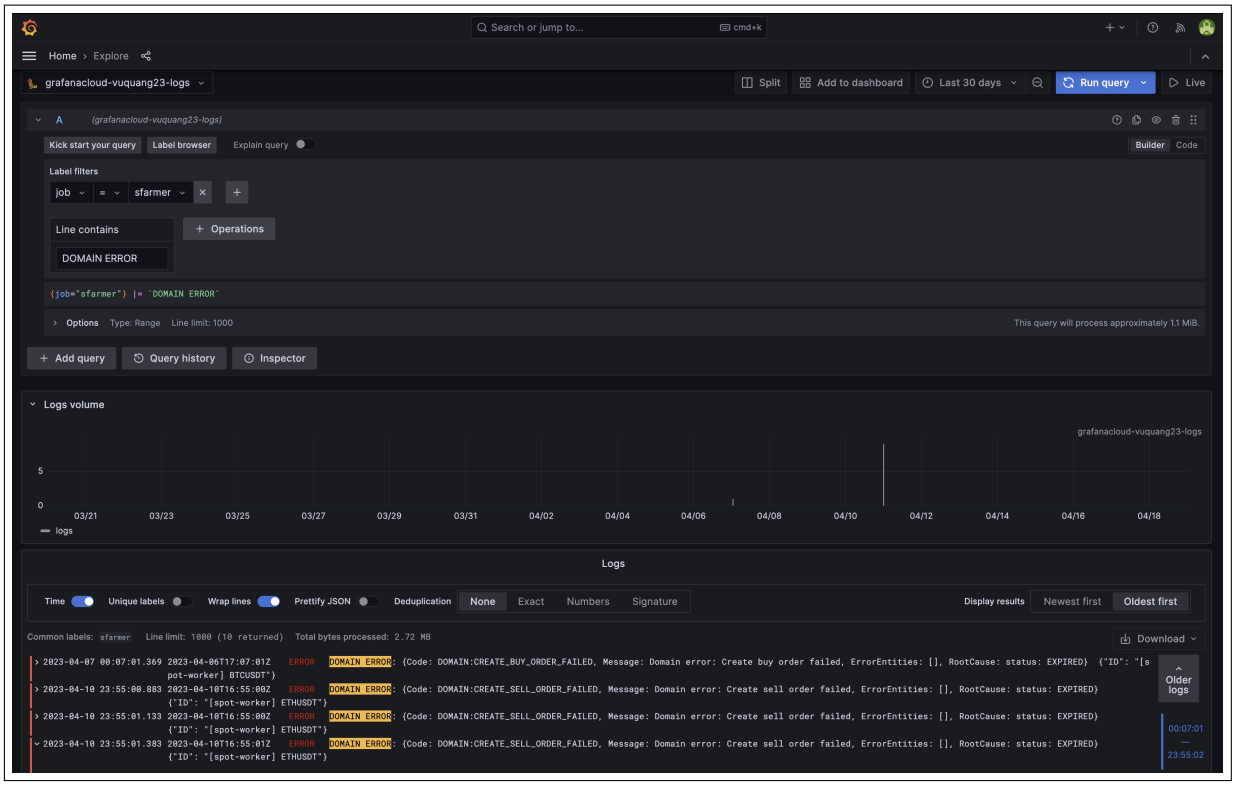

Figure 4.22: Log observability on Grafana Loki.

Chapter 5

Evaluations

I have run the trading bot on both the testnet and mainnet environments of the Binance Exchange. This involved executing the Bot in the testnet environment to ensure that it was functioning as expected without risking actual funds. After ensuring that the bot was working as intended, I then deployed it on the mainnet, which allowed me to monitor its performance in real-time trading with actual funds.

I ran the trading bot with a starting capital of $12,600 on the testnet environment, specifically to facilitate trading with two trading pairs: BTCUSDT and ETHUSDT. After two weeks of testing, the bot produced the result shown in Figure 5.1.

In the past, I ran the bot on Binance Exchange's mainnet with real capital of $588 for a period of two weeks. During this time, I monitored the bot's performance closely and was able to achieve promising results. The bot was able to execute trades efficiently and effectively, and I was able to see a positive return on investment. (Figure 5.2).

From the above two experiments, we can see that the return on investment when running on testnet and mainnet in two weeks is 3.3% and 6.1%, respectively. We can see a difference in the results, and the main reason is the market conditions. The market condition when running the mainnet was a strong sideway market with a slight upward trend. Meanwhile, the market condition when running the testnet was a slight downward trend. Moreover, the bot may encounter performance issues during a rapidly declining market as it has not yet implemented a stop-loss strategy. This is also an important feature that will be developed in the future.

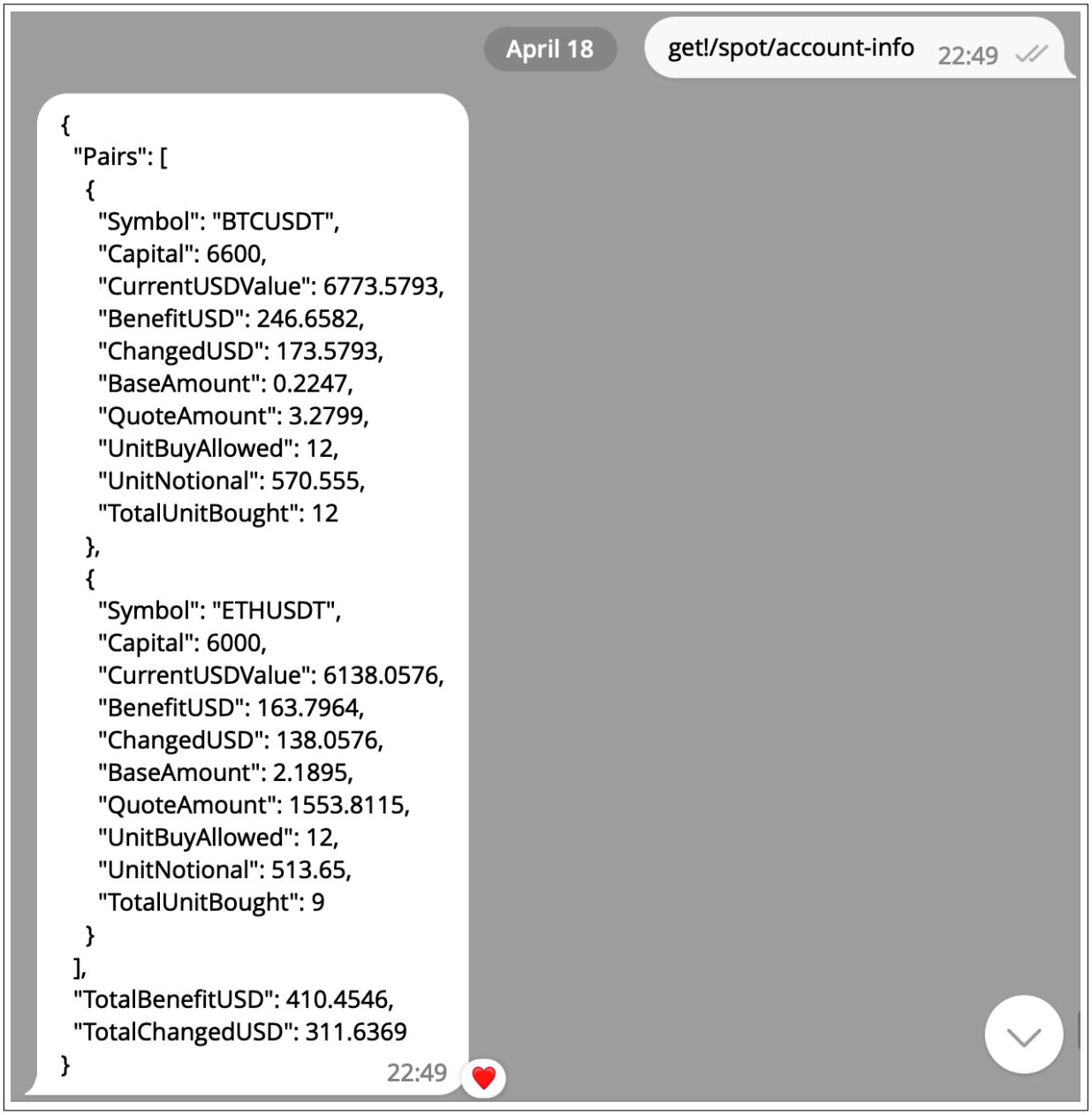

Figure 5.1: Bot trading performance on testnet in two weeks.

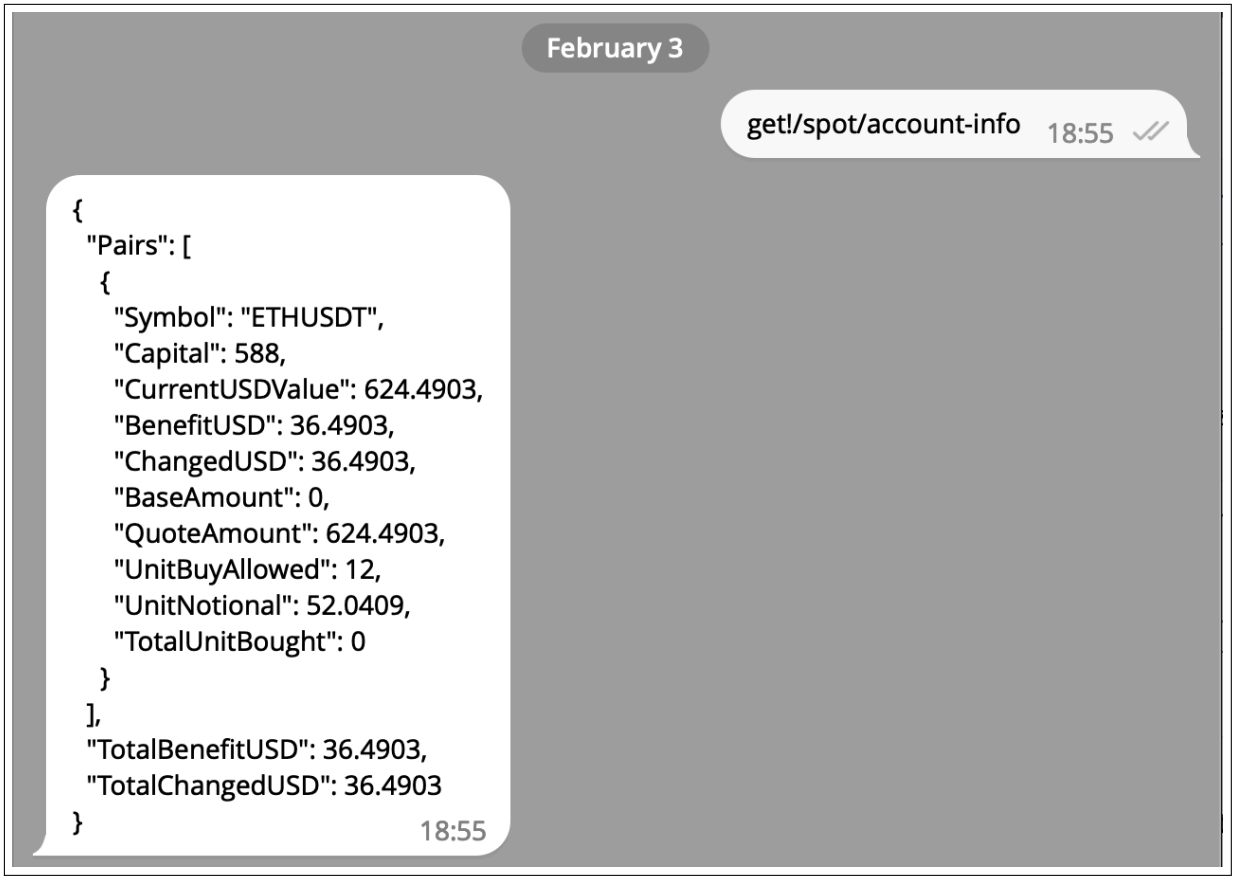

Figure 5.2: Bot trading performance on mainnet in two weeks.

Conclusion

This thesis has presented an in-depth exploration of the trading strategies implemented in the development of an automated trading bot, along with a detailed analysis of the associated software development aspects. The methodology utilized in the development process, the related technologies employed, the requirements of the system, as well as the system design, implementation, testing, deployment, and monitoring, have all been comprehensively described. Additionally, I have conducted experiments by running the bot and reported the results obtained. This has demonstrated the feasibility and potential of the project.

Selecting the topic for my project was a daunting task as I had no prior experience in developing an automated trading system. The thought of venturing into uncharted waters made me anxious, and I feared that I might not be able to deliver up to the expected standards. However, I sought solace in the words of the great German philosopher, Friedrich Nietzsche, who once said, "That which does not kill us makes us stronger." This phrase inspired me to take up the challenge and venture into the unknown.

As I began to delve deeper into the task at hand, my apprehension slowly began to subside, and my interest in tackling the project increased manifold. The challenge of developing an automated trading system was arduous, and I had to invest a great deal of time and effort to get the project off the ground. I encountered numerous roadblocks along the way, but I persisted with my efforts, learning from my mistakes and continuously refining my approach.

After many months of hard work and dedication, I was elated to have successfully built a software system with the bare minimum functions that were capable of generating profits in the crypto market. This achievement brought a sense of pride and accomplishment, and I knew that my hard work had paid off. Moreover, the process of developing the software system allowed me to gain invaluable insights into the field of software

engineering, particularly in the context of automated trading systems, and trading in the ever-evolving crypto market.

However, there are still many areas for improvement for the trading bot, and I aim to focus on implementing a stop-loss strategy and enabling the bot to perform trades on the futures market in further works.

References

[1] "Bitcoin average confirmation time (i:bmct)." [Online]. Available: https://ycharts.com/indicators/bitcoin_average_confirmation_time

[2] "Channel Synchronization." [Online]. Available: https://golangr.com/channel-synchronization

[3] H. Adams, N. Zinsmeister, M. Salem, R. Keefer, and D. Robinson, "Uniswap v3 core," 2021.

[4] A. Antonopoulos, *Mastering Bitcoin: Programming the Open Blockchain*, 2nd ed. O'Reilly Media, Inc, 2017.

[5] I. Bashir, *Mastering Blockchain: A deep dive into distributed ledgers, consensus protocols, smart contracts, DApps, cryptocurrencies, Ethereum, and more*, 3rd ed. Packt Publishing, 2020.

[6] G. Blokdyk, *Crypto Wallet A Complete Guide*, 1st ed. 5STARCooks, 2021.

[7] J. M. Bradshaw, *Software Agents*, 1st ed. MIT Press, 1997.

[8] S. Burns and H. Burns, *Moving Averages 101: Incredible Signals That Will Make You Money in the Stock Market*, 1st ed., 2015.

[9] A. Chuvakin, K. Schmidt, and C. Phillips, *Logging and Log Management: The Authoritative Guide to Understanding the Concepts Surrounding Logging and Log Management*, 1st ed. Syngress, 2012.

[10] A. A. A. Donovan and B. W. Kernighan, *The Go Programming Language*, 1st ed. Paperback, 2015.

[11] M. Egorov, "Stableswap - efficient mechanism for stablecoin liquidity," 2019.

[12] P. Frauenthaler, M. Sigwart, M. Borkowski, Taneli, Hukkinen, and S. Schulte, "Towards efficient cross-blockchain token transfers," 2019.

[13] E. Gamma, R. Helm, R. Johnson, J. Vlissides, and G. Booch, *Design Patterns: Elements of Reusable Object-Oriented Software*, 1st ed. Addison-Wesley Professional, 1994.

[14] F. KLINKER, "Exponential moving average versus moving exponential average," 2020.

[15] LazyBear, "Indicator: Wavetrend oscillator [wt]." [Online]. Available: https://www.tradingview.com/script/2KE8wTuF-Indicator-WaveTrend-Oscillator-WT/

[16] J. Loeliger and M. McCullough, *Version Control with Git: Powerful tools and techniques for collaborative software development*, 2nd ed. O'Reilly Media, 2012.

[17] A. Luo, W. Wu, J. Cao, and M. Raynal, "A generalized mutual exclusion problem and its algorithm," 2013.

[18] Microsoft, "Design the infrastructure persistence layer." [Online]. Available: https://learn.microsoft.com/en-us/dotnet/architecture/microservices/microservice-ddd-cqrs-patterns/infrastructure-persistence-layer-design#the-repository-pattern

[19] ——, "Retry pattern." [Online]. Available: https://learn.microsoft.com/en-us/azure/architecture/patterns/retry

[20] K. Network, "Dynamic market making v2," 2021.

[21] J. Nickoloff and S. Kuenzli, *Docker in Action*, 2nd ed. Manning, 2019.

[22] E. S. Ponomareva, I. V. Oseledetsa, and A. S. Cichockia, "Using reinforcement learning in the algorithmic trading problem," *Marchuk Institute of Numerical Mathematics, Russian Academy of Sciences, Moscow, Russia*, 2019.

[23] U. Sarma1, D. K. V. C. Rao, and D. P. Premchand, "Design patterns for scheduling tasks in real- time systems," *CVR Journal of Science & Technology*, vol. 2, 2012.

[24] K. Schwaber and J. Sutherland, *The 2020 Scrum Guide*, 1st ed. Creative Commons, 2020.

[25] J. D. Schwager and M. Etzkorn, *A Complete Guide to the Futures Market: Technical Analysis, Trading Systems, Fundamental Analysis, Options, Spreads, and Trading Principles*, 2nd ed. Wiley, 2017.

[26] B. Schwartz, P. Zaitsev, and V. Tkachenko, *High Performance MySQL: Optimization, Backups, and Replication*, 3rd ed. O'Reilly Media, 2012.

[27] M. Seemann and S. van Deursen, *Dependency Injection Principles, Practices, and Patterns*, 1st ed. Manning, 2019.

[28] K. Söze, *Cryptocurrency Trading: Strategies & Techniques for successful Portfolio Management*. Sabi Shepherd Ltd, 2019.

[29] S. Tarkoma, *Publish / Subscribe Systems: Design and Principles*, 1st ed. Wiley, 2012.