COMENIUS UNIVERSITY BRATISLAVA
FACULTY OF MATHEMATICS, PHYSICS AND
INFORMATICS

PAPER TRADING APPLICATION

Diploma Thesis

2022 **Bc. Eduard Kriváne**

COMENIUS UNIVERSITY BRATISLAVA
FACULTY OF MATHEMATICS, PHYSICS AND INFORMATICS

PAPER TRADING APPLICATION

Diploma Thesis

Study programme:	Applied Computer Science
Field of Study:	Computer Science
Workplace:	Faculty of Mathematics, Physics and Informatics
Supervisor:	RNDr. Peter Borovanský, PhD.
Consultant:	RNDr. Vladimír Karásek

2022 **Bc. Eduard Kriváne**k

Comenius University Bratislava
Faculty of Mathematics, Physics and Informatics

THESIS ASSIGNMENT

Name and Surname:	Bc. Eduard Kriváne k
Study programme:	Applied Computer Science (Single degree study, master II. deg., full time form)
Field of Study:	Computer Science
Type of Thesis:	Diploma Thesis
Language of Thesis:	English
Secondary language:	Slovak

Title: Paper trading application

Annotation: The student will design and implement a Paper trading application monitoring the current state and history of S&P 500 stock indices and offers real-time stock trading using virtual money.
The application gets current data from various stock market servers, and collects stock statistics and metadata.

Aim: The goal of the thesis is to implement a competitive Paper trading application, which enables tracking the growth of stock indices and mediates their trading in real time through virtual money. Applications of this kind are used in teaching financial literacy in economically oriented fields of study, also in the UK.
One of the student's challenges will be to create a server, which will obtain the necessary financial data and metadata for a ticker symbol, analyze the risk of the investment portfolio and statistically forecast the company's price using time series analysis.
From a technical point of view, the student must do an analysis between the REST and GraphQL server implementations, compare the differences and choose the appropriate method.
The client side solution must run on different platforms and be freely accessible via the cloud. The application will work with a large amount of data on the server, respectively on the client side, therefore, various optimization techniques, to speed up the application, are expected from the student.
The work assumes the publication and subsequent testing of the application on a larger group of users who are knowledgeable of the stock markets, which results will also be included in the thesis.

Keywords: technológie: Angular, Ionic, Flask, Apollo, Rest, GraphQL, Firebase DB, Firebase hosting, Google Cloud functions

Supervisor:	RNDr. Peter Borovanský, PhD.
Consultant:	RNDr. Vladimír Karásek
Department:	FMFI.KAI - Department of Applied Informatics
Head of department:	prof. Ing. Igor Farkaš, Dr.
Assigned:	20.09.2020

Approved: 06.10.2020 prof. RNDr. Roman Ďurikovič, PhD.
Guarantor of Study Programme

Comenius University Bratislava
Faculty of Mathematics, Physics and Informatics

.....

Student

.....

Supervisor

Acknowledgement

I sincerely declare that I have elaborated this thesis, entitled "Paper trading application", independently under the guidance of my supervisor and the consultant, using all the mentioned literature and resources.

.....

Bratislava, 2022 Bc. Eduard Kriváneк

Dedication

I would like to thank the following people for helping me to create and test this application. First to my supervisor RNDr. Peter Borovanský, PhD. for letting me to work on such a huge project and acquainting me with relevant people. To RNDr. Vladimír Karásek, who guided me in the right direction with my software in order to provide the most value for my users. Finally to doc. Mgr. Igor Melicherčík, PhD. and Mgr. Tatiana Jašurková who allowed me to test this application by their students of financial mathematics.

Abstract

The motivation behind this thesis is an observed lack of financial market understanding between students focused on different fields. The project's goal is to shrink this gap, to construct a trading simulator application, which allows creating a portfolio of different asset classes using virtual money, monitor its performance, and provide all available financial information of a given asset to the user.
At the beginning, in chapter 1 we take a look into time series, in hope to correctly model an asset class price prediction. We describe the most common ARIMA model, first mathematically, then we create a model in python (section 2.1.6), but by using computational forecasting we experience diverging from actual data and we conclude that more sophisticated methods are required, like machine learning, to get higher accuracy.
In the following section 4.2 we implement a REST data collector server, which gathers and modifies financial metrics of a given company from multiple external APIs.
To follow microservice architecture, which restricts mixing different module functionalities into one unit, we create a different GraphQL server (section 4.3), named data provider server, that provides market and user information from the firebase database. This chapter describes what is Apollo server, why GraphQL interface reduce server load and how to pass authenticated user's data to the backend.
Both servers have to be deployed to the cloud, to be freely accessible, therefore chapter 5 compares virtualization against dockerization and advocates the decision of using dockerization by a related study.
We also briefly touch on how users with or without an account can navigate in the system (section 4.5.5), in which the corresponding images can be found at the end of this thesis, in chapter Attachments.
Last but not least, after 3 months of a testing period, feedbacks from users were received, which is listed in the chapter 6, where the majority of students were satisfied, they proposed some feature improvements, but also two major bugs were fixed during this testing period (section 6.2).
By the time of submitting this thesis (year 2022), the production version of the application is accessible via url: https://stocktracker-prod.web.app/ .

Abstrakt

Motiváciou práce je spozorovaný nedostatok porozumenia finančnej gramotnosti medzi študentmi zameraných na rôzne oblasti. Cieľom projektu je zredukovať túto medzeru, vytvoriť aplikáciu simulátora obchodovania, ktorá umožňuje vytvárať portfólio rôznych tried aktív pomocou virtuálnych peňazí, monitorovať ich výkonnosť a poskytovať užívateľovi všetky dostupné finančné informácie o danom aktíve. Na začiatku, v kapitole 1 sa pozrieme na časové rady, s nádejou správne modelovať predikciu ceny aktív. Rozoberieme si najbežnejší model ARIMA, najprv matematicky, potom ho naprogramujeme v pythone (časť 2.1.6), lenže pri použití manuálnej výpočtovej predpovede dostávame odchýlky od skutočných údajov a dochádzame k záveru, že sú potrebné sofistikovanejšie metódy, ako napríklad strojové učenie, aby sme dosiahli vyššiu presnosť.
V nasledujúcej časti 4.2 implementujeme REST server, ktorý získava a upravuje finančné metriky danej spoločnosti od viacerých externých poskytovateľov údajov.
Aby sme dodržali architektúru mikro služieb, ktorá obmedzuje miešanie rôznych funkcií modulu do jedného celku, vytvoríme iný server pomocou GraphQL rozhrania (sekcia 4.3), ktorý poskytuje informácie o trhu a používateľoch z databázy firebase. Táto kapitola opisuje, čo je Apollo server, prečo GraphQL rozhranie znižuje zaťaženie servera a ako odovzdať údaje o overenom používateľovi do backendu.
Oba servery musia byť nasadené do cloudu, aby boli voľne prístupné, preto kapitola 5 porovnáva virtualizáciu s dockerizáciou a obhajuje rozhodnutie použiť dockerizáciu na základe súvisiacej štúdie.
V krátkosti sa tiež dotkneme toho, ako sa môže používateľ s vytvoreným účtom alebo bez neho pohybovať v systéme (časť 4.5.5), kde príslušné obrázky sa nachádzajú v sekcii Prílohy.
V neposlednom rade sa po 3 mesiacoch testovacieho obdobia zhromaždili spätné väzby od používateľov, čo je uvedené v kapitole 6, kde bola väčšina študentov spokojná, navrhli vylepšenia funkcií, ale taktiež sa vyskytli dve veľké chyby, ktoré boli opravené v tomto testovacom období (sekcia 6.2).
V čase odovzdania tejto práce (rok 2022) je produkčná verzia aplikácie prístupná cez url: https://stocktracker-prod.web.app/ .

Contents

Introduction

What is the fair value for a company? With the rise of the internet, plenty of information became available for the public and we started to manage our life from our convenience. One of the conveniences is investing. Within a few minutes we can create an online brokerage account, buy some stock and put our money to work. Simple as that, but then a question arises, why do the majority so called investors lose money? We could point to multiple mistakes, but I would like to highlight two of them, which, in the end, were the main motivation of writing this thesis.

The first group are psychological factors. We, as humans, are emotional creatures, that means, many times we tend to make decisions by pure impulse without any knowledge of the subject. In the world of finance, we refer to this phenomenon as fear of missing out. In simple terms, we want what everybody wants. Watching an increasing price of a ticker symbol may result in a need to jump to the trade and also the opposite side, seeing our position decreasing by its value can motivate us to sell our assets. This impulsive behavior may be translated into stock market by increased volatility in a short period of time, ignoring the company's long-term value.

The second point is lack of knowledge. If we see a ticker symbol trading for a specific value, how can we say that it is over or undervalued? Obviously, we need to analyse the company to make a judgment. There are two types of analysis. Technical, which tries to make predictions of future movement by evaluating historical daily prices from charts and fundamentals, which examines multiple financial metrics of a given company.

In the following chapters and sections we will focus on building a trading application, which allows for the users to create and manage their portfolio, find all the necessary asset class information in one place, but also we will dive deep into time series, in order to try to forecasting a company's future price.

Chapter 1

Time series analysis

When we observe behaviour of a single entity measured in data points and we would like to forecast its future value, we can use interpolation or extrapolation. Interpolation is a type of estimation where we construct a new data point within the range of known values obtained by sampling or experimentation. On the other hand, when we try to estimate some values beyond the original observation range, we use extrapolation, but the further we estimate from the known values the greater uncertainty and a higher risk of producing meaningless results arise.

Time series analysis is a type of extrapolation, where a set of numerical measurements of the same entity is taken at equally spaced intervals over time and its goal is to make a forecast for the future. It has four aspects of behaviour. Trend what represents the overall long-term direction of the data series. Seasonality occurs when there is repeated behaviour in the data which occurs at regular intervals. Cycles happen as a result of an up and down pattern that is not seasonal and can vary in length, which makes them more difficult to detect than seasonality. Lastly, random variations which can be found in all data. Some time series will be very regular with random variations while others may consist of not much else. The general equation for time series model is as follows [1]

$$y(t) = x(t)\beta + \epsilon(t)$$

Where $y(t) = y(t); t \in N$ is a sequence, indexed by the time subscript t, which is a combination of an observable signal sequence $x(t) = x_t$ and an unobservable independent random variation $\varepsilon(t) = \varepsilon_t$, also called white noise.

By definition [1], a time series is called white noise if the variables are independent and identically distributed with a zero mean and each value has no correlation with all other values in the data set. It has two meaningful concepts in forecasting. We cannot reasonably model a prediction

if it consists of only white noise, because by definition, it is random and ideally the error between our prediction and actual data should be white noise, which would indicate the possibility of further improvements to the forecasted model. As an example, we could use gaussian distribution with a mean of zero and a standard deviation of one. The result would be that the majority of data points will be located near our predefined mean value zero which represents a white noise.

In the following section we look at the most commonly used approaches to model time series analysis with the combination of observed signal and error. Before that, we need to discuss stationarity, because it can navigate us to which forecasting models should be used. A time series is called stationary if it satisfies the following criteria.

- Mean is constant
- Standard deviation (also referred as volatility) is constant
- There is no seasonality

Difference between stationarity and white noise is that the white noise has a constant zero mean, so if it is satisfied then also stationarity is satisfied, but not vice versa. A non-stationary time series data produces unreliable and spurious results and leads to poor forecasting. One feature which violates the constant mean criteria is trend. It is the slow, long-run evolution of preferences, technologies and demographics that is translated to always changing mean value. A non-stationary process with a deterministic trend becomes stationary after removing its trend.

1.1 Autoregressive model

The AR model specifies that the output variable y_t at some points t in time depends linearly on its own previous historical values y_{t-p} of order p. It can be only applied for stationary data with the following equation [20].

$$AR(p) = y_t = c + \sum_{t=1}^{p} \phi y_{t-i} + \varepsilon_t$$

Where c is constant, ε_t is white noise and p is the order of the process (i.e., how many lags to model). Lags are where results from one time period affect the following periods. ϕ is the corresponding coefficients for each lag up to order p which must meet the following criteria.

$$\phi_1 + \phi_2 + ... + \phi_k = 1$$

In section 1.4 we will discuss how to properly choose the order p to get the most accurate results. However, since autoregressive models base their predictions only on past information, they implicitly assume that the fundamental forces that influenced the past prices will not change over time, which is not always the case in real life.

1.2 Moving average model

In the moving average, the difference from the autoregressive model is that rather than using past values of the forecast variable in a regression, we will use past forecast errors [20].

$$MA(q) = y_t = \mu + \sum_{t=1}^{q} \theta_i \varepsilon_t - i + \varepsilon_t$$

μ is the mean of the series (also referred as expected value at y_t), t is white noise error term at time t and θ is a coefficient multiplayer of that error. Fitting the MA estimation is more complicated than in the AR model. An iterative solution is required because the lagged error terms are not observable. By using the lag operator, which operates on an element of a time series to produce the previous element, we can rewrite our formula into

$$y_t = \mu + (1 + \theta_1 \beta + ... + \theta_p \beta^p)\varepsilon_t$$

Moving average model should not be confused with moving average smoothing which simply states that the next observation is the mean of all past observations. It is used to smooth out short term fluctuations,

remove seasonal patterns and highlight long term trends. It requires a specific window size called window width which represents the number of raw observations used to calculate the new series.

1.3 Autoregressive moving average model

It's natural to predict the future values of some equity based on its previous values adjusted with some errors. This is where we can use the combination of AR part, which involves regressing the variable on its own lagged values and MA part, modelling the error term as a linear combination of error terms in the past [20].

$$ARMA(p,q) = y_t = c + \sum_{t=1}^{p} \phi_i y_{t-1} + \varepsilon_t + \sum_{t=1}^{q} \theta \varepsilon_{t-i}$$

In situations like the stock market, where we try to predict the future value of the company's share price, we also have to account for trends in our computation. Using a simple ARMA model will not be efficient, because it expects the time series to be stationary and as we mentioned before, one of its criteria is to have a constant mean. When a stock follows a specific trend, its mean is either increasing or decreasing. If this is the case then we can use the ARIMA(p, d, q) model, where the letter I stands for integrated, and d is its order (i.e. number of transformations to make the time series stationary). Integrated means that instead of predicting the time series itself, we will transform our time series and calculate the difference of one timestamp to the previous timestamp.

$$z_t = y_t - y_{t-1}$$

The usefulness of this transformation lies in the linearity (trend) of the original function. When we have an upward or downward trend and we take the difference of two side by side values, we expect the result to hover over some constant. This constant will be the new mean value for our transformed time series where we can use the ARMA model.

$$z_t = c + \sum_{t=1}^{d} \phi_i z_{t-1} + \varepsilon_t + \sum_{t=1}^{d} \theta \varepsilon_{t-i}$$

To choose the right candidates for our ARMA model we need to determine what order of AR and MA terms should be used. To solve this problem, we introduce two functions called Autocorrelation function (ACF), which will answer the AR part, and Partial autocorrelation function (PACF) for MA. Though ACF and PACF do not directly dictate the order of the ARMA model, their plots can facilitate understanding the order and provide an idea of which model can be a good fit for the time-series data.

1.4 Autocorrelation function

Describes how well the present value of the series y_t is related with its past values until y_{t-k}. ACF considers all concepts like trend, seasonality and cyclic and finds correlation between data points. The monitored dependency can be caused by correlation of other values between y_t and y_{t-k}. In case of a stationary time series, we express the ACF function as follows [20].

$$p(j) = \frac{\widehat{\mathrm{Cov}}(y_t, y_{t-j})}{\widehat{\mathrm{Var}}(y_t)} = \frac{\sum_{t=j+1}^{n}(y_t - \bar{y})(y_{t-j} - \bar{y})}{\sum_{t-1}^{n}(y_t - \bar{y})^2}$$

By plotting the graphical representation of the ACF function we can pick only the statistically significant values and by counting them we will obtain the order of AR model.

1.5 Partial autocorrelation function

The key assumption behind ACF model is that a variance y_{t-1} captures all values older than itself and is able to explain the correlation between every single data point up until y_{t-k}. That means the end result in y_{t-1} is influenced by all previous correlations in $y_{t-2}, y_{t-3}...y_{t-k}$ from the time series.

But what if the assumption is not true and the result in y_{t-1} is not able to explain all the variances contained in y_{t-2} ? We would like to also capture the unexplained portion of y_{t-2} and feed it into y_t. So, in example of the MA model using ACF we could end up with the following formula

$$y_t = \mu + 1 + \theta_1 y_{t-1} + \theta_2 y_{y-2} + \varepsilon_t$$

Unfortunately, by using simple ACF it is not possible, because y_{t-2} is already captured in y_{t-1} and by adding it again, it can drastically influence the end result.

Introducing Partial ACF, as the name suggest, by finding the correlation of y_t and y_{t-k}, we eliminate all intermediate values $y_{t-1}, y_{t-2}...y_{t-k+1}$, so if the value y_{t-k} had some unexpected behaviour, it will be directly captured in y_t without any influence. As an example, the first value of ACF and PACF is the same because there are no intermediate measurements, but the second lag in PACF will measures the correlation between y_t and y_{t-2}. Partial autocorrelation $p(j)$ is defined as [20]

$$p(j) = \frac{\widehat{\mathrm{Cov}}[(y_t, \hat{y}_t), (y_{t+j}, \hat{y_{t+j}})]}{\sqrt{\widehat{\mathrm{Var}}(y_t - \hat{y}_t)}\sqrt{\widehat{\mathrm{Var}}(y_{t+j} - \hat{y_{t+j}})}}$$

1.6 Choosing the right time series application

A variety of different forecasting procedures are available and it is important to realize that no single method is universally applicable.
The most widely applied model for forecasting linear time series is currently the ARIMA model used by Box–Jenkins time series procedure in 1976. [4] The main difference compared to ARMA model is that ARIMA model makes a non-stationary time series, where for example a mean is increasing over time, so our time series may experience some upward or downward trend, into a stationary series before working on it. We tend to use the integrated part of the ARMA model, because instead of predicting the time series itself, we predict differences of the time series from one timestamp to the previous timestamp. ARIMA characterizes time series by going from three fundamental aspects [20]:

- Autoregressive terms (AR) that model past process information.
- Integrated terms (I) that model the differences needed to make the process stationary.
- The moving average (MA) that controls the past information of noise around the process

Researchers mainly focused on linear models, such as ARIMA. where future values are cramped to be linear functions of past data, due to their simplicity in comprehension and application.

1.7 Related study

In the research paper Forecasting of demand using ARIMA model [7], authors talk about forecasting the future supply manufacturing of a food company by utilizing historical demand data using ARIMA model. A survey shows that more than 74% poor forecasting accuracy and demand volatility are the major challenges to a supply chain flexibility.
To achieve their goal, the study was carried out in three parts: identification, estimation and verification, where they used large and consistent historical data from January 2010 until December 2015. By testing several models to identify the most suitable coefficient values, ARIMA (1,0,1) have been chosen to model forecasting the next 10 months of sales, but each time the needed to feed the historical data with the new data to improve the forecasting
Conclusion was that even if the chosen model was accurate, they experienced an error variate, but it was among the tolerance interval. The authors acknowledge that in order to minimize the emerging errors, manual algorithmic computation is less efficient than using ANN (artificial neural network) which are characterized by intervals with considerable variation of demand.

Authors Meftah Elsaraiti and Adel Merabet in paper of forecasting wind speed for renewable energy [14] compare two approchease, the most widely known ARIMA model to LTSM RNN trying to capture uncertainty and fluctuations in wind speeds. Researchers lean towards neural networks, because traditional computation methods are less likely to handle nonlinear and irregular data which resolves in higher error rate.
Concluding these two approaches, reseachers find that both have their limitations. Even if LSTM network followed more accurately the actual data

movement, it required a learning time in order to achieve precise values. On the other hand, ARIMA produced better results with smaller quantity of data, but we face the problem of fiding the right coeficients for the model.

In a paper written by Fathi Oussama [15] , author presents a hybrid modelling approach for time series analysis, combining ARIMA model with neural networks in hope to capture certain patterns that would be inaccessible without the support of each other and thus provide adequate results in time series prediction.
Despite the widespread use of the ARIMA model, it still has some major disadvantages, such as its high error rate as soon as normality and/or linearity in the data is lost which eventually leads to a decrease in inaccuracy. Also just by a manual computation to find seasonality, performing noise whiteness test, examining stationarity and finding coefficients are expensive in calculation time.
Research proposes using a hybrid combination of ARIMA model with recurrent neural networks, especially LSTM (long short term memory). RNN are a type of network that reuse the output from a previous step as an input for the next step alongside with the next element, but in case of LTSM the network has an internal state to function as a working memory space.
The idea of a hybrid method is as follows. Transforming the original time series into a smooth function by isolating it from its seasonal component using Box and Jenkins method and then by obtaining only a separated trend function without fluctuation approach it with LSTM model to know its dynamics.
To demonstrate effectiveness they use two popular datasets, the Wolf's sunspot data which contained 289 nonlinear, 10-year cycling observations from 1700 until 1988 and the US dollar/British pound exchange rate data daily observations from 1971 to 2018 giving 11939 data points.
The results of the research was that separated data prediction using ARIMA and then LTSM model yielded similar results, to similar error metrics. Authors concluded that LTSM could very well have a better performance if the operation were done correctly. Nevertheless the combination of both models led to the smallest error metrics and accuracy in prediction.

Chapter 2

Portfolio analysis

The recent global pandemic in 2020 showed that organizations, or individual investors, must establish project portfolio risk management to protect their portfolio and balance the level of risk. Depending on a chosen severity, an investor may choose to conduct various statistical modeling, sensitivity or timing analysis. It is important to understand that some companies are riskier, like the technology industry. compared to the healthcare sector. Proactive management with adequate portfolio risk can help to maximize returns, but to do that we have to know those measurable financial ratios, which will help us understand our investment strategy and also try to use statistical time series analysis to help us predict short term movement.

2.1 Portfolio risk

Modern portfolio theory (MPT) is a risk-return tradeoff optimization method for a diversified portfolio, based on the notion that markets are efficient and that diversity spreads investments across a number of assets. According to MPT, you can hold a high-risk asset type or investment, but the overall portfolio can be adjusted by combining multiple types of diversification, lessening the risk of the underlying assets or investments individually[6].

The entire risk of a portfolio of investments is referred to as portfolio risk, summing up the risks of all the investments in a portfolio. The weightings of the various components of a portfolio contribute to the amount to which the portfolio is exposed to certain risks. Market and other systemic risks are the most significant threats to a portfolio. To guarantee that a portfolio accomplishes its objectives, these risks must be handled. To better understand an individual's investment risk, we will calculate the following ratios: Alpha, Beta, Standard deviation, sharpe ratio, to provide necessary information about the current state of investments[6].

2.1.1 Alpha

Alpha, also referred to as excess return, is a term used to describe an investment strategy's ability to beat the market. Alpha gauges the performance of an investment against a market benchmark which represents the market's movement as a whole. It returns a value between -1 and 1, indicating whether the portfolio overperformed or underperformed a broader index in some percentage manner, but we are using this metric to compare similar funds with each other. Calculating alpha we are using the formula:

$$\alpha = R_p - R_f - \beta * R_m \tag{2.1}$$

$R_p =$ return of portfolio
$R_f =$ risk free rate
$R_m =$ broad market return

2.1.2 Beta

Beta helps understand stock's responsiveness to the overall market, providing insight about how volatile a stock is relative to the rest of market. it is a good metric to analyze short term volatility, but because it is calculated by historical data, it becomes less meaningful for forecast predicting. Its value is usually around number 1 and means the following

- When its value is less than 1.0, then the security is less volatile then the market
- When its value is greater than 1.0, then its price is more volatile than the market by by one percent per one hundredth of rise
- When the value is negative, it mean that the stock is inversely correlated to the market benchmark

$$\beta = \frac{\widehat{\mathrm{Cov}}[R_p, R_m]}{\widehat{\mathrm{Var}}(R_m)} \tag{2.2}$$

$R_p =$ return of portfolio
$R_m =$ broad market return

2.1.3 Standard deviation

It effectively measures a fund's volatility, or its proclivity for sharply rising or falling returns in a short brief period. A volatile investment is also regarded as posing a greater risk because its performance can swing dramatically in either way at any time. The standard deviation of a fund measures the degree to which the fund fluctuates in proportion to its mean return, which is a measure of risk.A fund with a consistent four-year return of the same percentage, would have a standard deviation of zero percentage, because its average does not depart from its four-year mean. However a different fund with returns of -5%, 17%, 2%, and 30% would have a mean return of 11%. , with a high standard deviation, because each year returns differ from each other. The standard deviation is calculated as the square root of variance by calculating the deviation of each data point from the mean. Greater difference between each data point indicates a wider price range and consequently more volatility. We calculate standard deviation by the following formula:

$$\sigma = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \hat{x})^2}{n-1}} \tag{2.3}$$

x_i = i-th point in data set
$\hat{x}$ = the mean value of the data set
n = the number of data points in data set

If the end result is not squared, we would have been calculating a statistical measurements called variance, which itself is a measurement of how far each number in the set is from the mean and thus from every other number in the set.

2.1.4 Sharpe ratio

The Sharpe ratio was developed by Nobel laureate William F. Sharpe and is used to help investors understand the return of an investment compared to its risk. The return earned in excess of the risk-free rate per unit of volatility or overall risk is referred to as the ratio. Volatility is a measure of an asset's or portfolio's price fluctuation. The Sharpe ratio can also be

used to determine if a portfolio's excess returns are the consequence of sound investment selections or excessive risk. Even if a portfolio or fund earns better returns than its peers, it is only a sound investment provided those higher gains are not accompanied by excessive risk. The Sharpe ratio of a portfolio determines its risk-adjusted performance, so If the ratio is negative, it signifies that the risk-free rate is higher than the portfolio's return, or that the portfolio's return is predicted to be negative.

$$sharpratio = \frac{R_p - R_f}{\sigma_p} \tag{2.4}$$

R_p = return of portfolio
R_f = risk free rate
σ_p = portfolio standard deviation

2.1.5 Calculation process

To calculate to above mentioned four portfolio risk statistical measurements, at the end of each day, a cloud function (section 4.4) is dispatched, which sends to our data collector server (section 4.2) a POST message, its payload contains an array of stock ticker symbols, that an individual person owns, and also how those symbol's weights are distributed in that person's portfolio.
At the beginning of each day, users can check their portfolio risk measurements on the dashboard page, like it is illustrated in the figure 7.14 So as an example we may have that the user's portfolio is diversified in companies such as 35% in AMD, 40% in Apple and 25% in Microsoft, then the request, which will be send to our server at the end of each day and its response is illustrated in the following listings.

Listing 2.1: Portfolio analysis request

```
{
        "symbols": ["AMD", "AAPL", "MSFT"],
        "weights": [0.35, 0.40, 0.25]
}
```

Listing 2.2: Portfolio analysis response


```
{
        "date": "2022-04-05T21:27:05.525436",
        "portfolioAlpha": 0.1817,
        "portfolioAnnualVariancePrct": 0.0796,
        "portfolioAnnualVolatilityPrct": 0.2821,
        "portfolioBeta": 1.5335,
        "portfolioEstimatedReturnPrct": 0.3582,
        "portfolioEstimatedReturnValue": 47.676,
        "portfolioSharpRatio": 1.1811,
        "portfolioVolatilityMeanPrct": 0.3233
}
```


2.1.6 Forecasting stock prices

Following up on chapter 1, we will try to implement ARIMA model in python on the ticker symbol AAPL (Apple). We will use ACF (section 1.4) and PACF (section 1.5) functions to plot a graph which could tell us the necessary order parameters for the ARIMA model, however, later we will see that basic graph evaluation is not sufficient enough and more additional tests are required to obtain the right order. We will split the time series, closing prices of this ticker symbols, into 80% training data and try to predict the remaining 20%.

To apply ARIMA model on any dataset, it has to be stationary, but to be qualified as such, the dataset has to have a constant mean, constant standard deviation (volatility) and no seasonality. To verify these conditions, we can run 2 independent tests to determine stationarity. First one is a visual graph plotting check, usually it is clear whether those criteria are satisfied, but to get a more measurable result, we use Augmented Dickey Fuller (ADF) Test.

The ADF test is a statistical significance test at its core, returning a p-value as a result which can infer whether a series is stationary or not based on the test statistic. ADF belongs to a 'Unit Root Test', determining the stationarity of the dataset, but also the number of differencing operations necessary to make the series stationary corresponds to the number of unit roots in the series. The key point to remember is that since we assume the presence of unit roots, the obtained p-value from the ADF test should be

less than the significance level 0.05, in order to assume stationarity [ToDO reference]. To get precise values and avoid computational errors, we are using the statsmodel library in python to estimate ADF value, but also the following operations described below.

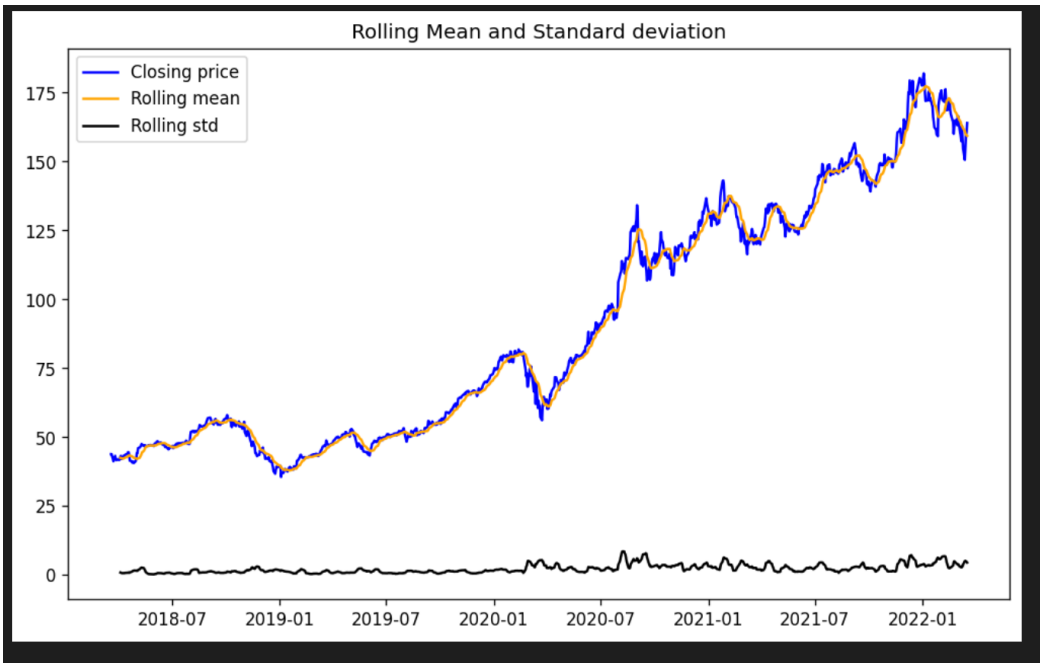


Figure 2.1: AAPL stock rolling mean & standard deviation

```
ADF test for symbol AAPL, using returns = False
1. ADF :  -0.0733400079255773
2. P-Value :  0.9520300753212004
3. Num Of Lags :  20
4. Num Of Observations Used For ADF Regression: 988
5. Critical Values :
        1% :  -3.4369860032923145
        5% :  -2.8644697838498376
        10% :  -2.5683299626694422
```

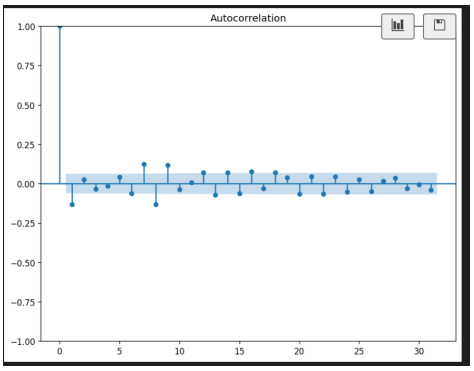
(2.2a) AAPL closed prices ADF test

```
ADF test for symbol AAPL, using returns = True
1. ADF :  -10.07875253918953
2. P-Value :  1.2041591800879e-17
3. Num Of Lags :  8
4. Num Of Observations Used For ADF Regression: 998
5. Critical Values :
        1% :  -3.4369193380671
        5% :  -2.864440383452517
        10% :  -2.56831430323573
```

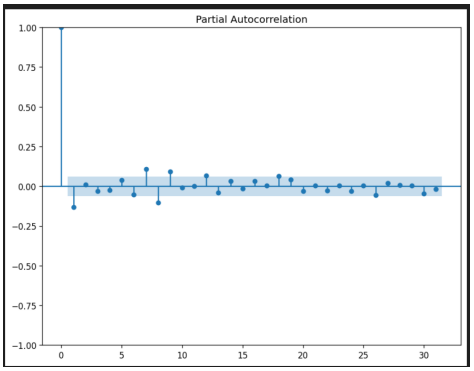
(2.2b) AAPL closed price returns ADF test

Figure 2.1 clearly shows an increasing rolling mean overtime, indicating that closed prices are not stationary. We verified this assumption by ADF test (figure 2.2a), which gave us a p-value 0.95, far above the threshold 0.05. Once we differentiate this dataset by one, subtracting each current value from its previous one, which will give us the daily price returns and running ADF test again, it results in a p-value far less than our threshold, confirming that the data became stationary (figure 2.2b).

After performing this operation, the next step is to determine ARIMA model order by ACF and PACF functions. By using the statsmodel library and its functions plot_acf and plot_pacf to make this calculation more precise, we receive the following results:



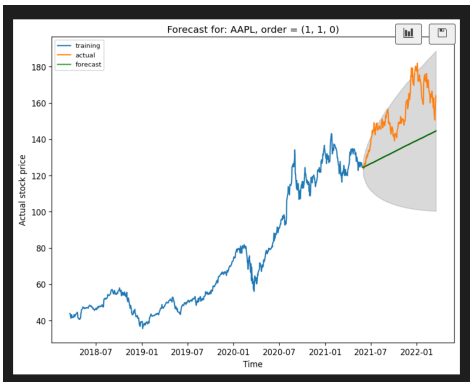
(2.3a) AAPL ACF plot



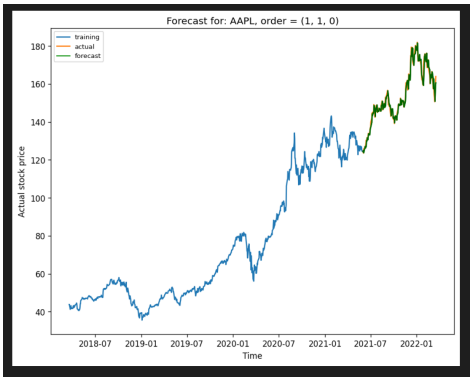
(2.3b) AAPL PACF plot

Evaluating figures 2.3a and 2.3b it may not obvious what order we should apply to the ARIMA model. The AR value should be the last value which is above the threshold in ACF chart, and similarly MA should have the last exceeding value in PACF chart. The problem with ARIMA model, or any statistical predicting model, is that usually we find the right model just by multiple trial and error attempts (section 1.6). To automate finding the optimal order we can use auto_arima library, for this purpose, which finds the best solution based on determined criteria.

Running the auto_arima function on 80% training data resulted in fitting the model with 807 historical observations and expecting it to predict the following 203 continuous values. The auto_arima function ended up with an order (1,1,0), representing the resulting predition in the figure 2.4a, where you can see that the overall prediction was correct, an upper trending line was generated, but its predicted values were far away from the actual ones. After evaluating this test we may have 2 hypotheses. First one is that the model (1,1,0) was not sufficient enough, or the second one is that the computational ARIMA model is really useful just to predict the first few values of the time series. The further we go in our prediction, without remembering and re-training our model with correct data, the more incompatible data we get. If we run our test once again with order (1,1,0), but only forecasting one step forward and then re-training the whole model with the actual correct data we tried to forecast, then we get more precise prediction, what can be seen in the figure 2.4b, but its limitation is that we can generate only one day forward.



(2.4a) AAPL price prediction



(2.4b) AAPL price prediction with memory

```python
from statsmodels.tsa.arima.model import ARIMA,
    ARIMAResultsWrapper
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.tsa.stattools import adfuller

def performArima(self, order = None, useMemory = False):
    ...
    self.__arimaWithCustomOrder(order, useMemory)

def __arimaWithCustomOrder(self, order, useMemory):
    if useMemory:
        history = list(self.traningData)
        forecast = []
        for i in range(len(self.predictData)):
            arima = ARIMA(history,order=order)
            fitModel = arima.fit()
            data = fitModel.forecast()[0]
            history.append(self.predictData[i])
            forecast.append(data)
        forecast = pd.Series(forecast)
        forecast.index = self.predictData.index
    else:
        arima = ARIMA(self.traningData,order=order)
        fitModel = arima.fit()
        forecast = fitModel.forecast(len(self.predictData))
        forecast.index = self.predictData.index
    return forecast
```

Listing 2.3: ARIMA implementation with memory

The above code (listing 2.3) shows that when running a custom arima function with *useMemory* set to True, we only predict one datapoint to the future (line 16.), but also before our next arima model training (line 11.) we save the correct value of that forecasted day (line 17.) and just after that we predict the following day. After handling multiple tests on multiple ticker symbols, their prediction resulted to a similar outcome as this described example.

The observed conclusion is that using a computational ARIMA model is insufficient, in our case, to forecast lots of data to the future, because everytime we diverge from actual data the more datapoints we try to predict. Arima can be used only to predict the nearest few data points with higher accuracy. The takeaway is that right now time series forecasting will not be implemented into our system, more sophisticated models are needed for better price prediction, like some kind of machine learning, which may be considered as a future implementation to the project.

Chapter 3

Application specification

In this section will have a brief overview of some non-technical specifications, describing what value we are bringing to our customers and address some technical requirements how the application has been built, whereas each module is described in later chapters.

3.1 Non-technical requirements

As the application's name suggests, the major feature in our system will be portfolio management. Meaning, all users will start with a predefined cash amount and we will provide them an interface to supervise their assets. Right now only basic trading functionalities, like buy or sell, will be implemented, advanced features like margin trading, limit or short positions will be skipped for now. At the end of each day the system will snapshot all user's balances, current cash on hand and invested amount, save the data into the database and visualize an individual's performance in multiple graphs. Calculating portfolio risk metrics, such as alpha, beta, volatility and sharpe ratio is also a must feature in analyzing investment strategy.

To simplify finding desired stocks an advanced search system will be implemented, where companies can be filtered out by their market capitalization, price, volatility, dividends, sector and exchange types. By selecting a ticker symbol, we will present all available information, financial metrics and historical data of that publicly listed company, in a nice pleasing visual representation. For that reason, an affordable third party data providers have to be found and probably use methods like web scraping to gather all possible data of that selected company.

A comment has to be addressed that no financial deposit will be needed into the system. This trading mechanism is done via virtual money, so called paper-trading for educational purposes, that's why users will also have the ability to reset their portfolio history.

Additional included non-technical features will be watchlists - having fast access into user's favorite symbols, allocating users into groups to increase competition spirit and a ticket submitting mechanism for whatever bug is found or suggestion is proposed by any members.

3.2 Technical requirements

A critical obstacle to overcome is to gather as much information as possible for any publicly traded company. Two issues arise by accomplishing this task. First one is that we have to find third party data providers which dispose with such information and the second one is that we do not know the shape of data which we will receive from one or multiple external APIs. Since we are trying to build an enterprise level application with the ability to scale based on the service usage, we will follow microservice architecture where the key benefit, compared to monolith, is to have independent, self-contained services, whereby catching an error will not stop the whole application, only the isolated service. In the figure 3.1 we can see a high overview of our application communication. Let's briefly describe why such a module separation is required.

Data collector server will be a responsible entity dedicated to obtaining all stock related financial metrics from external APIs. It will be written in python for convenient data formatting and we will expose some http endpoints using the flask framework, which is a useful tool to create web services, to allow our data collector service to get all these company information by a single http call.

We will restrict direct communication between frontend and the data collector service, instead of that we are going to have a middle server, a so called data provider, which will take care of each user's requests. So when a user will demand any stock information, the data provider server will receive that request, check data existence in firestore database, in case of data missing it will fetch resources from the data collector server, persist them in the database and finally return the result to the client. This data provider server will be written in Apollo which is more suitable when a large amount of data with multiple relations are requested.

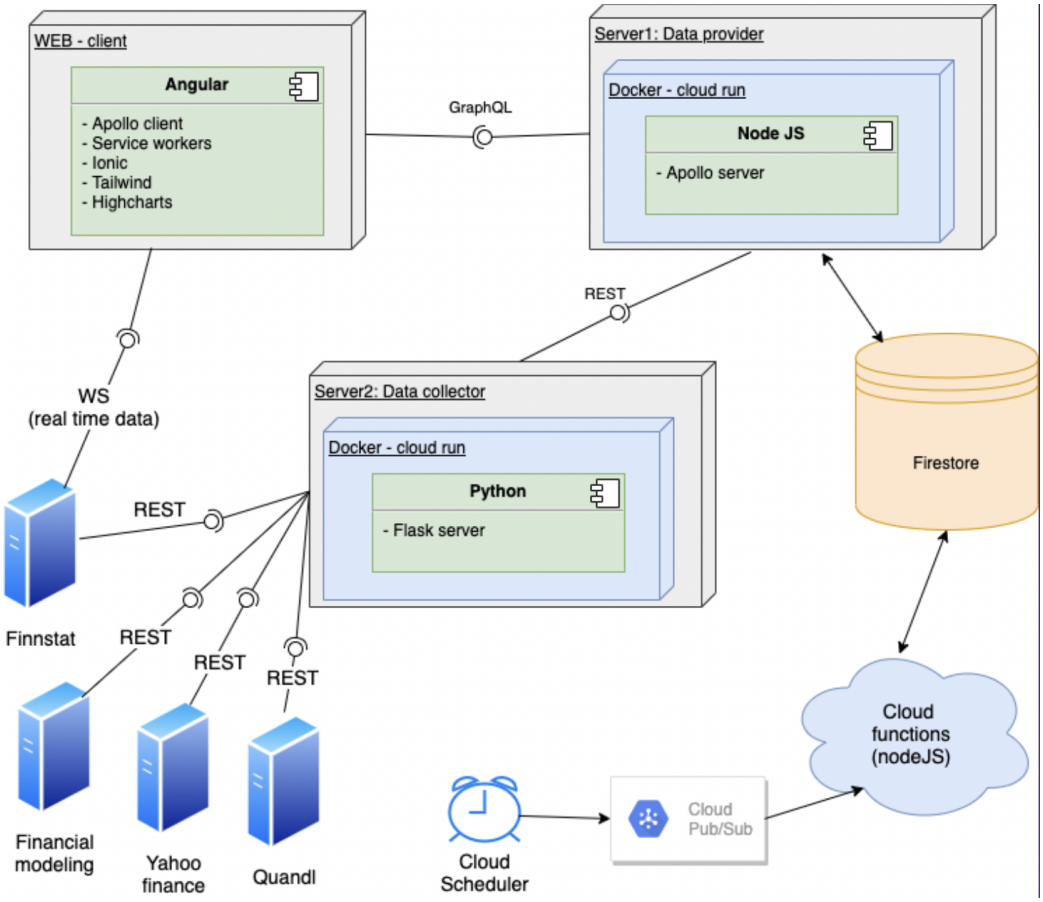


Figure 3.1: Architecture overview

By not yet knowing the shape of the data (stock information) which we will be working with, it is reasonable to lean towards a NoSql database such as firebase, that is a low pricing solution for a cloud storage. When thinking about databases, we may also consider traditional sql databases, which ensures ACID compliance and are great for complex queries, but since we will constantly add new features and change the data model based on how much information are we able to get for a single company, traditional sql databases provides less flexibility in comparison with NoSql databases.

When it comes to deployment, both servers will have its own docker files, which creates a lightweight executable processes, so called containers, that can be deployed to any environment, but in our case we will use a service provided from Google, called cloud run. In short, it can automatically scale our application vertically, meaning in peak times it can generate multiple instances of the same server, but also in lean times, when no user is present, shut down all running instances for a better computer resource utilization.

Client, or data consumer service, will be written as a single page application (SPA) using angular, which is one of the leading client side frameworks. As one major npm libraries we will use apollo-client to help us establish connection with a graphql server and provide in-memory caching mechanism, and also we will add service workers to persist asset files in the end user's computer to increase application load time. Considering real time stock price updates, an instruction will be shown after a successful registration, to create a free account on finnhub service, which dispose of such a functionality to allow 50 different symbol subscriptions for one websocket connection. In the future a paid service will be considered without total symbol restriction, but these are high in price solutions and right now we want to minimize our cost basis.

Last but not least, at the end of the day to periodically run a function, which will snapshot each user portfolio and save their historical performance into the database, we are going to use cron job schedulers to trigger cloud functions for that purpose.

Chapter 4

Application implementation

In this chapter we are going to focus on previously discussed application specifications. We are constructing two separated backend servers, which fulfill different purposes, but are written in different languages, so we will take a look at the distinction between them. We will also reason about why specific technologies and ways of implementation were chosen and finally on some major improvement solutions like asynchronous jobs (section 4.4), data caching (section 4.5.3) or cross-platform communication (section 4.5.4).

4.1 Design an API

By designing an API, a developer may lean toward a known technology without even considering its constraints. The constraint says that both the frontend and the backend should work separately without having to intervene with each other or without depending on each other. Clients should not know how the server is fetching the resources, what database is used on the server side, what language has been used to develop the program, and all other backend related stuff. Similarly, the server should not know how the client is using the all resources it has requested, or how the user interface has been designed nor any other frontend related stuff.
Exploring the types of constraints when designing an API as a cloud application [3], we can divide them into categories such as:

Business constraints Thinking about our product, what tasks it has to fulfill, will our solution be just a temporary, later replaceable piece of code, or should it be around for another 10 years. Also know what customer related requirements are supported. We may think about implementing a GraphQL interface for improving entity relation queries, but if our client can only process csv files, this change will cause more harm than good.

Complexity constraints The ability to resist changes without the need of a big refactoring, like for example, structuring and returning entities by endpoints, launching asynchronous tasks, considering the quantity and size of our components and their data communication. If we have a lot of clients making a lot of requests, we will probably face some size and resource related complexity issues.

Resource identification constraints When making a request for a resource, the server responds with a representation of this resource. This representation captures the current state of the resource in a format that the client can understand and manipulate with as a sequence of bytes along with metadata describing those bytes. This metadata is known as the media type of the representation. Typical API responses are HTML, JSON, and XML. Because the server sends a representation of the requested resource, it is possible for the client to request a specific shape of data that fits the client's needs. For example, a client can ask for JSON or XML type of the resource and the server may provide its answer if it is capable of doing so. This concept is called content negotiation. Content negotiation can be used in APIs to allow multiple clients to access different representations of the resource from the same URL.

When we apply the above mentioned constraints to the process of building a distributed system, we get some measurable properties which will eventually tell us how right we were by following a specific design pattern. Those properties are, first of all, performance, which can be also subdivided into network performance or user-perceived performance. Next is scalability of the size and operation of the hosting machine, i.e. vertically by increasing the resources of the computer where our system is running, or vertically by creating multiple instances of the same application and load balancing each and all incoming requests. Finally modifiability, which simply tells how easy it is to evolve an already large application, integrate changes, add new features and customize or restrict certain parts of the application to specific clients[3].

We could go on with other properties such as visibility, which includes monitoring the time and resource complexity of executing a specific part of our code, security by unauthorized access, caching data by database or just using in memory cache to increase execution performance. Last but not least, reliability of the system which includes the ability to recover from failure generated internally or by an external client.[3]

By the following sections we will take a look how to build a data provider system which will consist of two different APIs, one designed as an stateless rest server, implemented in python using flask micro web framework, and another one in nodejs following graphql query language specification by using apollo framework.
We will dive into the multiple options of deploying both of our servers and evaluating whether cloud based solutions, such as google cloud platform, sufficiently answers the scalability and visibility concerns.
Eventually we will answer the question of how to have asynchronous jobs, which update user's data in the database by leveraging the power of cloud functions with scheduler notification.

4.2 Data collector server

As part of our application will be an ability to display all possible information about publicly listed companies. The goal is to accommodate as much information of a given company as possible, that users do not have to browse multiple sites once evaluating the risk-reward factor of a stock. In this section we will describe the implementation of a data collector server, as an stateless REST API in python, which collects and aggregates information from multiple external data providers and expose an interface to further disseminate these data. We will also take a look at some optimization techniques for method completion improvements and in the end we will discuss the drawbacks of using an REST API.

The REST acronym is defined as a "REpresentational State Transfer" and is designed to take advantage of existing HTTP protocols when used for Web APIs. It is very flexible in that it is not tied to resources or methods and has the ability to handle different calls (GET, POST, PUT, PATCH, DELETE) and data formats (XML, JSON, Media) [3].

All REST interactions should be stateless. That means, each network request have to contain all of the information necessary for a connector to understand the demand, being independent of any network call that may have preceded it. This statelessness accomplishes four functions:

- it removes any need for the connectors to retain application state between requests, thus reducing consumption of physical resources and improving scalability
- it allows interactions to be processed in parallel without requiring that the processing mechanism understand the interaction semantics
- it allows an intermediary to view and understand a request in isolation, which may be necessary when services are dynamically rearranged
- it forces all of the information that might factor into the reusability of a cached response to be present in each request

A traditional web server does not understand or have any way to run python applications. WSGI servers handle processing requests from the web server and decide how to communicate those requests to an application framework's process, which are defined as the PEP 3333 standard. The segregation of responsibilities is important for efficiently scaling web traffic.

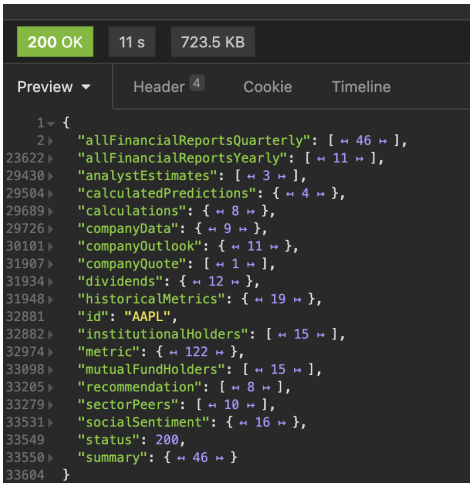
As mentioned in the beginning of this chapter, the main responsibility of this data collector server is to fetch, format and retrieve necessary information about publicly listed companies. Its role is to function as a middleware between our data provider server (section 4.3) and external data providers, but also compute statistical measurements both for a symbol and also for user's portfolio holdings (section 2.1).

To expose a communication interface we must first create a controller layer in our application. Those are just simple functions which react to a specific URI (Uniform resource identifier), that makes up a global namespace to identify a key resource with a unique ID.
When a client asks for some resource, its request consists of a request method (basic CRUD operations), URI, header and body (specially for PUT & POST requests). When a controller layer accepts a request, a common strategy is to pass received data, such as query parameters or body payload, to a service provider layer, whose purpose is to complete some business logic. As a demonstration for such a business logic in our case may be a method, which job is to parallelly fetch stock information from multiple URLs and return all the data.

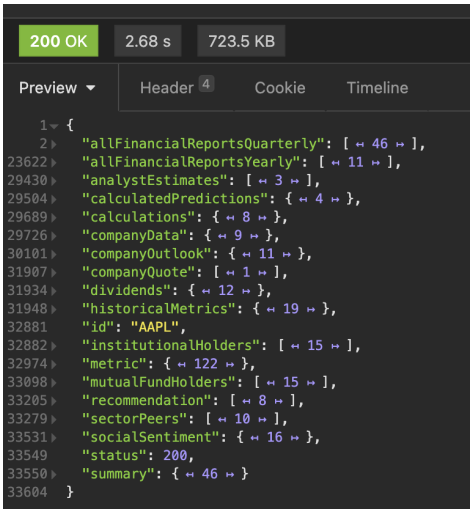
Figure 4.1: Data collector server service logic

Picture 4.1 shows that each and every method call in this block is handled as a separate thread. The reason for this is that every method is just a wrapper round a http call to an external resource server. These http calls in python are handled as a synchronous blocking operations, which in the execution context only means that if we had not used threads, our code would be blocked at each line until that call to an external resource is fulfilled. So to execute the following child methods to gather data, we would have to wait the completion of the previous method, even if those operations are independent from each other.

The operating system manages the threads itself and is capable of distributing them between available CPU cores. Threads are lighter than processes. In essence, it means we can generate more threads than processes on the same system. We can hardly run 10,000 processes, but 10,000 threads can be easy. On the other hand, there is no isolation, i.e. if there is any crash, it may cause not only one particular thread to crash but the whole process to crash. And the biggest difficulty is that memory of the process, where threads work, is shared by threads. Sharing memory means that there is a need of synchronize access to it. While the problem of synchronizing access to shared memory is the simplest case, we have to beware not to override the data that another thread is working with [23]. When we evaluate the following resource requests with and without using threads, we can see a significant time improvement by up to 5 times as shown in the figure 4.2b. In practise it means, that the user would only see a spinner, as a placeholder, until the content is not yet loaded, for a 2,5second, compared to a previous, without thread usage, 11 second (figure 4.2a).



(4.2a) Rest call without threads



(4.2b) Rest call with threads

However, the increased use of REST has exposed some limitations that harmed the performance of such APIs in crucial aspects. In general, clients with complex routes require nested object searches with multiple relationships. Due to the characteristics of a REST API, that exposes resources exclusively, it is necessary to perform several searches on the server before some routes are processed by the client, since not all information is sent in a single reply message. Also, most of these calls will return unnecessary data to the route context that executed it, which is known as over-fetching. Thus, multiple solutions have been proposed to increase the efficiency of data search, one considering the requests and replies formats, while other ones are optimizing the number of requests in the network[9].
REST APIs are modeled around the resource they return, which makes them general purpose and reusable from client to client. However, a recent trend involves a declarative data query model, in which client applications specify data they need. Then, such models optimized the communication with the server to get data in an efficient way. This is the purpose of GraphQL, which aims to mitigate some chronic problems of REST design, such as multiple round trips and excessive data traffic on the network.

For that reason, we are also going to implement another server, this time as a graphql server using apollo server, which will be used as a data provider server to directly communicate with a client. This means, that client will have an ability to form one or multiple queries, with only those fields, which are required for a specific view, therefore optimizing network bandwidth by only fetching the necessary data.

In section 4.3.1 we describe why having a graphql server is more convenient for a client than a REST endpoint and also we will look at some performance metrics comparing these two implementations from a study (section 4.3.3). We will leave our data collector server as a small independent microservice, which will be deployed to GCP (google cloud provider) (chapter 5). By isolating our application into multiple microservices, we can later better monitor their performance, and only scale those parts of the system, which are needed.

4.3 Data provider server

The goal of this section is to implement a second server instance, written in nodeJS using apollo-server, where its objective will be to load, modify or persist user's data into the database. By choosing a graphql server instead of a REST server for communication, we will have an ability to define multiple queries for different screens resolutions (desktop, mobile), removing fields from objects which are not displayed on a smaller device, therefore reducing network bandwidth and saving end user mobile data.

One of the pain points when using REST APIs is that the data coming from the server should be mapped many times to different object types in order to be presented on the screen or vice versa. These hard coded mappings are basically matching statically typed code, like creating object interfaces, with the unstructured JSON, to know what shape of response is expected when executing a http call. These custom castings or validations might fail as we know the server is always changing and may deprecate some fields and objects.

4.3.1 What is GraphQL ?

GraphQL is an application layer server-side technology which was developed by Facebook for executing queries with existing data. GraphQL can optimize RESTful API calls and gives a declarative way of fetching and updating data. Instead of working with rigid server-defined endpoints, we can send queries to get exactly the data we are looking for in one request. GraphQL is a query language specification, independent of the language implementation, which resembles a json data format without any values [4].

There are a few components to a graphql query. A selection represents the set of data we are trying to resolve from the server. Selection sets are composed of many fields in which one field describes a specific slice of data we wish to request from the server. The data we resolve from our fields can come from anything. It could come from a complex computation done on the server to output some computed value, or straight from a database table.

The only similarity with REST is that we still have an endpoint, as some URL to our remote server ,ending with an /graphql postfix, and we are sending a POST message where in the payload of the data is the expected returning structure message from a server. Let's have a look at a query in our system which is used to get authenticated user's data.

```
fragment StockSummaryFragment on Summary {
  ceo
  companyName
  id
  marketCap
  marketPrice
  sector
}

fragment STHoldingFragment on STHolding {
  symbol
  breakEvenPrice
  units
  summary {
    ...StockSummaryFragment
  }
}

query AuthenticateUser($id: String!) {
  authenticateUser(id: $id) {
    id
    nickName
    userPrivateData {
      roles
    }
    holdings {
      ...STHoldingFragment
    }
  }
}
```

(4.3a) Graphql query


```
{
  "data": {
    "authenticateUser": {
      "id": "Rd6txjVNB7UkiVTop1YZVrweUtE2",
      "nickName": "krivanek1234",
      "userPrivateData": {
        "roles": [
          "ROLE_ADMIN",
          "ROLE_GROUP_CREATE"
        ]
      },
      "holdings": [
        {
          "symbol": "MMM",
          "breakEvenPrice": 183.68654046463115,
          "units": 13,
          "summary": {
            "ceo": "Mr. Michael Roman",
            "companyName": "3M Company",
            "id": "MMM",
            "marketCap": 104491958272,
            "marketPrice": 183.66,
            "sector": "Industrials"
          }
        },
        { ↔ },
        { ↔ },
        { ↔ },
```


(4.3b) Graphql response

Picture 4.3a shown a query named AuthenticatedUser which gets authenticated user's data. A keyword fragment is only used to create a reusable data structure across multiple operations without defining the same logic twice. By leveraging the power of graphql, we only query those fields which we are interested in. Note that type Summary may contain plenty of more fields, which are just attributes of a given object, persisted in the database, or they are resolved in runtime, but these fields are not important

for us right away, so we do not include them in our initial query, hence they do not need to be resolved.

When we execute a query [18], it goes through phases such as parsing into a abstract syntax tree (AST), validating the query against its schema, checking the correct query syntax, fields existence and finally executing the query by walking through the AST from the root invoking resolvers and collecting results. A resolver is a function, which can also be asynchronous, that resolves a value for a field in a schema by returning a scalar or an object type, in which the execution continues to the next child field until it halts or null is returned. The main advantage of resolvers are that they are not executed until the users calls those fields in its schema. Resolvers are lazy, meaning if we query a database entity, which has many relations with another tables, but we do not include those relational fields into our schema, then no join operations will be performed from the database side, what actually will lead to a faster response time. Also in each invocation, arguments like root object, querying arguments or context can be passed.

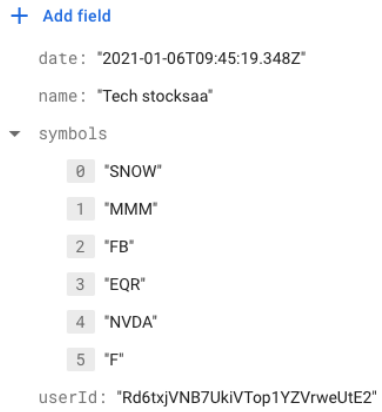
- **Root** is a result from parent type. By resolving a field for an object, it is possible to pass the whole object into to function for some computation operations.
- **Args** are arguments provided to the field, used mainly in situations for querying a specific object identified by an unique key
- **Context** is a mutable object, created and destroyed in each query, where related information to an query is provided, such as authentication token

GraphQL also provides real time message updates, also called subscriptions [4], by listening on a particular event that occurs on the server side. Comparing to a traditional bidirectional communication channel used by websockets, subscriptions function in a publish-subscription model.
The client (subscriber) first opens a communication channel with a server (publisher), sends all the subscription queries it is interested in and when a specific event is triggered, server executes the stored query and the result is sent through the same communication channel opened by the client. This type of communication is stateful, thus server remembers each variable, arguments and context, but can be canceled by the client or by server error anytime.

4.3.2 Apollo server

Note that graphql is a query language communication about data, to fetch only a partial part of a desired object, more or less just a specification, but apollo is the most used implementation of it. Apollo is a server and client independent library, allowing us to exchange information via graphql. This independence means, that in the future we may change our client framework from the current one and still keep all apollo configuration and queried data types. Configuring a custom ApolloServer using node, we can put a bunch of settings like cors, extensions, cache control, persistance, etc. This only shows that this implementation is configurable based on the project scaling.

For our data provider server, we only gonna focus on 3 parts which are type definitions, resolvers and context. A schema is a collection of type definitions, which defines queryable fields for any entity exposed to clients. Usually a graphql object type mirrors fields of a given entity which is saved in the database. Generally we define for a graphql object type the same fields which are for a given entity saved in the database (fig. 4.4a) to mirror its structure, however that exposed type may be also extended with additional attributes, which right away don't exist, but are resolved at runtime. For example an array attribute summaries for STStockWatchlist (fig. 4.4b) which is a collection of summary information about stored ticker in symbols array. To keep data normalization, these summary information are persisted in a different table and for each symbol, using resolvers, we are performing a database query into another table. This operation can be compared to joins at relational databases.



(4.4a) Database entity

(4.4b) Graphql schema

Another major benefit from the frontend perspective is the auto generated documentations of the backend schema structure. This documentation, or also called playground, is an interactive in-browser IDE, which allows us to see the whole graphql schema (mutations, queries and subscriptions) exposed by the server. This is a convenient way of documentation for the developers to describing backend structure without the need of a third party library. Also by adding an exclamation mark (!) for a specific entity (fig. 4.4b), we are telling the consumer that this entity cannot be null, so no null condition checking is necessary.

Lastly, we are, in most cases, dealing with queries per user. Meaning, in the database we have multiple users and their data, so we want to only query resources associated with that specific requester. Solving this problem by putting an additional argument for each and every query like *requestedId* would be tedious, but fortunately, apollo provides us in its configuration an object type called context. This object is passed through every query and resolver. It is created and destroyed by each http request. It runs as the first function before anything else, like a middleware, and we can access the http request object from its argument, therefore performing a validation whether a token or any other user identification exists in that http header. If not, we reject the connection or else we perform the operation only for the user whose identification is located in the request header.

4.3.3 Comparing graphql to REST

Sayan Guha compapre in her paper [10] a performance evaluation between graphql and rest stateless services, tested out fetching large data structure by both rest and graphql, where each request is independent from each other and no data is cached from previous response.
The experiment was concluded on a simulated social media application with a bidirectional relationship between two entities users & posts with variable data volumes. For the graphql endpoint, when fetching users, all posts could be loaded for each user in one query, the whole required response by client was resolved in one server-client iteration and additional requests were not needed. On the other hand, in case of rest architecture, by returning all users form from one endpoint, for each user, additional network operations were required to be made to fetch their posts.
The result of performance evaluation was that by a small volume of data there were almost no difference in response time, however a significant improve-

ment (40%) were measured towards graphql architecture when the volume increased up to 100K.
Although, an acknowledgement were given to rest web services to becoming industry standards for its ease of use and simplicity, despite of that, in cases where a large number of entities are involved and data is required at scale, the study will favour to graphql.

4.3.4 Disadvantages of graphql

One disadvantage [8] is that queries always return a HTTP status code of 200, regardless of whether or not that query was successful. If a query is unsuccessful, the response JSON will have a top-level errors key with associated error messages and stacktrace. This can make it much more difficult to do error handling and can lead to additional complexity for things like monitoring.
Another disadvantage is the lack of built-in caching support. Because REST APIs have multiple endpoints, they can leverage native HTTP caching to avoid refetching resources. With GraphQL, there may be a need to set up an own caching support which means relying on another library.
This brings us to the final drawback - complexity. Graphql can eliminate many of the pain problems associated with REST APIs for firms what have the engineering resources to devote to rearchitecting their API platforms, otherwise simple rest API is sufficient enough.

4.4 Event base communication

As mentioned in the non-technical requirements (section 3.1) the system needs to have some asynchronous jobs, which will be fired at specific time to perform a specific task, like for example evaluating a user's portfolio at the end of the day. The most suitable solution for these tasks are bringing cron jobs[21] into the picture. It takes a string of five slots as its argument (minute, hour, day, month, weekday) and creates an idle process executing a passed callback function at a given period of time.

Following SOLID principles[17], especially single responsibility principle, it would make a sense to decouple these scheduled processes into a self contained services, where each could have its own unique responsibility and

a sole reason behind its change, therefore we can leverage a function type from firebase called Pub/Sub. It's a publisher-subscriber function which allows us to pass messages in between google services that is also integrated with google cloud scheduler. This means we can run any backend code at any specific point in time using crontab scheduler. To register our custom services (like portfolio evaluation), we will use cloud functions. It is an event driven serverless compute platform, which scales automatically based on the load and customer is charged only when they are running. So in idle time, the cost of maintaining these functions is zero.

Figure 4.5 shows a registered cloud function (functionCreateUserPortfolioSnapshot), which is triggered every evening at 23:45 by a crontab scheduler, to evaluate the user's portfolio and its logs are visible in the figure 4.6. For security reasons all cloud functions have a maximum running threshold of 5 minutes, exceeding this amount the execution is abandoned.

Name	Region	State	Description	Frequency	Target	Last run	Last run result	Next run
createUserPortfolioSnapshot	europe-west1	Enabled	Creating portfolio snapshot about current state (invested + cash) - At 23:45 on every day-of-week	45 23 * * * (Europe/Bratislava)	Topic : projects/stocktrackertest-e51fc/topics/createUserPortfolioSnapshot	Feb 17, 2022, 11:45:00 PM	Success	Feb 18, 2022, 11:45:00 PM

Figure 4.5: Cloud function

```
2022-02-17T22:45:04.802Z  functionCreateUserPortfolioSnapshot  79xcscj6kzqz  Started updating at Thu Feb 17 2022 22:45:04 GMT+0000 (Coordinated Universal Time)
2022-02-17T22:45:04.802Z  functionCreateUserPortfolioSnapshot  79xcscj6kzqz  date: 2022-02-17T12:00:00.000Z
2022-02-17T22:45:06.141Z  functionCreateUserPortfolioSnapshot  79xcscj6kzqz  users with holdings: 6
2022-02-17T22:45:06.142Z  functionCreateUserPortfolioSnapshot  79xcscj6kzqz  updating user: s9ozFsP5eZSWfhWLUwNFm7DrUpu1, nickname: thezlateriksk, [1 / 6]
2022-02-17T22:45:11.054Z  functionCreateUserPortfolioSnapshot  79xcscj6kzqz  updating user: 9dQVMwMkGoZnxqIL3wZu4qdebaY2, nickname: FMFI, [2 / 6]
2022-02-17T22:45:18.971Z  functionCreateUserPortfolioSnapshot  79xcscj6kzqz  updating user: Rd6txjVNB7UkiVTop1YZVrweUtE2, nickname: krivanek1234, [3 / 6]
2022-02-17T22:45:42.011Z  functionCreateUserPortfolioSnapshot  79xcscj6kzqz  updating user: xSy6bn3JnRhO2H98T8l3piSmNaY2, nickname: Matt, [4 / 6]
2022-02-17T22:45:42.496Z  functionCreateUserPortfolioSnapshot  79xcscj6kzqz  updating user: teQZ01PUskOCm3GTt54j45zjwmB3, nickname: krivanek12345.6, [5 / 6]
2022-02-17T22:45:42.965Z  functionCreateUserPortfolioSnapshot  79xcscj6kzqz  updating user: iyGQLYhpQ8NVuyoEQzcpgA2E0sA3, nickname: krivanek1234.5, [6 / 6]
2022-02-17T22:45:43.457Z  functionCreateUserPortfolioSnapshot  79xcscj6kzqz  Distinct symbols: 32
2022-02-17T22:45:43.457Z  functionCreateUserPortfolioSnapshot  79xcscj6kzqz  Completed updating at Thu Feb 17 2022 22:45:43 GMT+0000 (Coordinated Universal Time)
2022-02-17T22:45:43.457Z  functionCreateUserPortfolioSnapshot  79xcscj6kzqz  Total run: 39 seconds
```

Figure 4.6: Cloud function logs

4.5 Data consumer

When choosing the right frontend framework, or any kind of framework, we first must evaluate some key aspects of our preferences and the application lifespan, whether our option will be able to support growing system requirements and also users demands. Some of those aspects are [12]:

- **Backbone** - All of these tools are developed and maintained by other teams. We should try to understand the long term vision of the responsible person or institution, because discontinued support of that tool may slowly start the fall of our project
- **Licensing** - We should be careful with licensing under which the framework is maintained and selecting only those which do not violate source distribution
- **Completeness** - We are looking to answer whether the framework gives us complete functionality with its built-in features or additional development for solving common problems will be needed
- **Performance** - Considers final bundle size, its initial load time and runtime performance. Application must be fast and responsive to meet all user needs
- **Cross-platform independence** - By building modern client-side applications, we want to bring the easiest accessibility to our users, which means web applications must be also installable as mobile or desktop apps and behave almost like native ones.

By choosing Angular, as one of many options, to build our client, we try to understand its value brought to the table by answering the above mentioned concerns.

4.5.1 Angular modularity

Modules are Angular's main building blocks[13]. In a simple term, modules allow grouping individual pieces of logic and components under one umbrella. They define a context for compiling templates. When HTML template parsing is performed, angular checks each individual HTML tag, creates a list of it and checks whether a custom element should be applied on top of it. The problem which modules solve is isolating some base functionality and exposing only those functionalities which should be used by other parts of the application. A module is composed of the most importantly 4 parts - declarations, imports, providers, exports.

- In **declarations** all module related elements are registered and are internally (at module level) available to other elements, but not exposed to the rest of the project
- In **exports** are located elements from the declaration array which are visible and usable cross the whole project
- In **imports** are registered other modules on which our current one is depended on
- In **providers** we include services which will be created in the module hierarchy. By default each service is registered at the root level, which makes them as singletons, but to create multiple instances of that service, we include them into providers section

The main benefit of modularity comes in hand when we describe lazy loading. It is a technique when we do not register our custom modules in the root modules (which would make them available by the start of the application), but they are asynchronously loaded when a specific route is activated. This splits the application into several bundles, multiple files with smaller size, which overall improves load time and are fetched only when those assets are needed.

4.5.2 Angular compilation

When it comes to compiling a client side application every code change has to be reflected in the browser without a need of constantly rebuilding the application. Angular solves this problem by its Just in Time (JIT) compilation mechanism[13]. It provides compilation during the execution of the program at runtime, meaning not all code is compiled at the initial time, only necessary components which are going to be needed at the starting and the rest of the functionality is compiled only when they are invoked. It helps to reduce the burden on the CPU and makes projects render fast. This method is preferable only on local development, because each component is complicated separately, which makes immediate change reflection in the browser, however the whole source code is inspectsable in the devtools, so when making a production shipment a different approach must be applied.

Compiling our frontend project to be production ready, Angular generates plenty of files, where we have to focus mainly on two of them, which are main.js and vendor.js. Main.js contains our application logic, converted typescript code into raw javascript, which is downloaded at the start of the application, and vendor.js includes any local or third party library imported to the app. When the above mentioned JiT mechanism would be used in production, not just we could expose our source code in the browser, but also, because project functionality is complicated in the browser only when they are invoked, for that reason, Angular would have to also ships its CLI to the customer's computer. The result would be reflected by a long waiting period until the system is loaded, because by each visit Angular always had to compile the app in the user's browsers.

By all means we want to avoid that, that's why another compilation method was introduced, called Ahead of Time (AoT). Generally, it is a process of compiling higher-level language or intermediate language into a native machine code. A longer compilation period is entered where Angular detects and reports template binding errors before users can see them, which reduces unwanted bugs, and a better security is applied. Angular compiles templates into Javascript files only once, before they are served into the client display, eliminating the risk of client-side HTML evaluation and reducing the size of main.js, because files are served to the user immediately, so there is no need of shipping Angular's CLI to the customer. When we utilize this AoT production build, Angular under the hood performs tree shaking, which is a method of dead code elimination. We may have some unused legacy

functions in our code which have not been removed yet, but those are not needed to properly run the application therefore are not shipped to the end user[13].

4.5.3 Data Caching

One of the key metrics which evaluates website performance is the speed of data displaying. As the user navigates through different routes, the application shows many dynamically loaded data to the customer, such as his portfolio on the dashboard section, or stock analytics on the search section. In order to fetch all of this data, XHR requests must be sent to the data provider server and asynchronously react on the server's responses. Show the results on a successful network call, or error message, if failure occurs, with a meaningful explanation of what went wrong. This is a common behavior how plenty of web apps work, however by this approach one issue arises, which is that we always execute a http operation by multiple visits of the same web section, or in other words we load multiple times the same data which have been already fetched, but have not been cached. So the question is how to cache server responses ?

As we described the data provider server (section 4.3), we have built a graphql backend, so first of all, an appropriate library is needed to establish communication between the client and server. A comprehensive state management library, called apollo-client[24], will be used, that enables local and remote data management with graphql. When we first initialize apollo's configuration, out of the box it includes a powerful caching mechanism, called InMemoryCache, which normalizes incoming data to keep consistency across multiple components. So how does it work ?

When we query any exposed type, which is a database entity representation (figure 4.3a), the majority of entities has some kind of identification, such as ID. What apollo does by data normalization is that, for example, when we query an array of entities, apollo creates a logically unique identifier for each entity, stores them separately and if another objects want to access these entities, because of maybe some relational connection, they can do it only by their reference, which is that unique identifier. See figure 4.7 for better illustration. This normalization brings us a benefit of updating an entity at one place and the change will be reflected in all the parts of the application or objects, which have some relation with that entity.

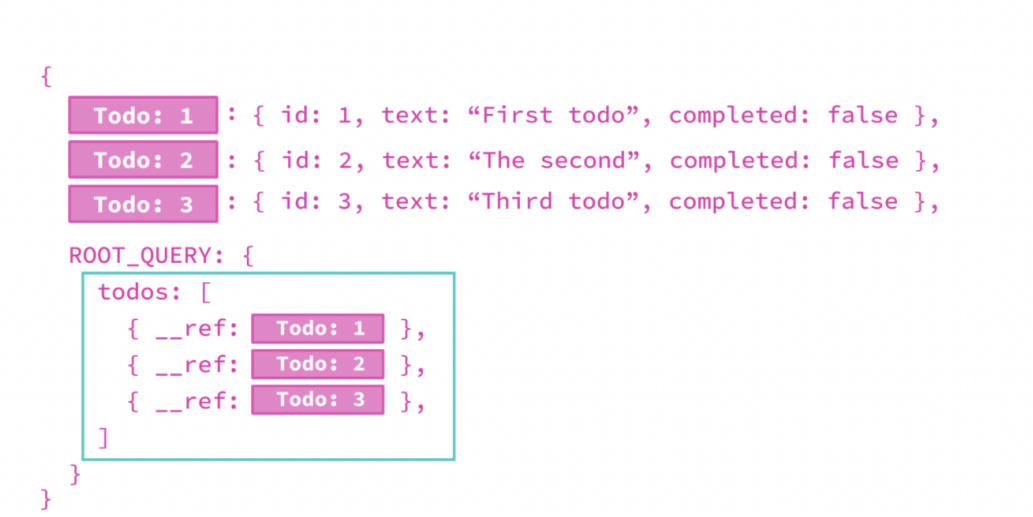


Figure 4.7: Apollo-client data normalization
https://www.apollographql.com/blog/apollo-client/caching/demystifying-cache-normalization/

Despite data normalization being a useful tool, it does not solve a problem of multiple http calls for the same resource. What does solve it however is apollo's, by default, behavior of transforming the execution query, with all of its parameters, into a SHA-256 hash format and confirming whether that hash has already been executed[24]. If the system has already carried out that query, Apollo will not send it to the backend, asking for the same information twice, after all, those data are already persisted in its cache, so it just loads them from there. We can definitely make a configuration to tell which server responses to cache or not, nonetheless by utilizing apollo's hashing algorithm, we can unload our backend with unnecessary requests.

4.5.4 Cross-platform independence

One of the main issues engineers face during software development is the question of running their application on multiple platforms. We are developing a web application, which is accessible via browser, but what if we would like to have an installable version in desktop PC's or having it available via mobile stores for IOS and Android ? As the first thought one may consider creating multiple separated native applications in the language supported by the selected platform. This is however an unwise decision, because in the end each feature change in our app would need to be made in all the projects, whereas these projects are maintained just for a specific platform. This is a very time consuming and costly approach. The issue we want to solve is how to have a single source of truth, i.e. only one project base, which will eventually be built and deployed on multiple platforms ?

Progressive web applications (PWA) is a hybrid between responsive web and native apps. PWAs are written in any web technology, look and behave like regular websites, but are both visible via search engines and offer similar functionalities to native apps, such as offline mode, push notifications and use device hardware (location, camera, ect.)[16]. They can also be placed on the home screen, like a fastlink, without needing to be directly installed, thus saving memory space on the device.

Progressive Web Apps is a concept created by Frances Berriman and Alex Russel to describe apps taking advantage of new features supported by modern browsers in 2015. A year later, during Google IO, this concept was introduced as a new standard in web development.

There is no clear definition when an application is considered as a PWA, whether it is qualified from the moment it is installable on any device or just after a proper offline mode implementation. But in general, the following three pillars qualify a PWA as a platform-specific application[16]:

- **Capable** - to have a native app experience, when we engage with the user, we can leverage its device tools like location, camera, file system to enhance the whole user experience
- **Reliable** - they must feel fast regardless of network connection. Basic functionalities such as logs, recent history, account data should be available even in offline mode
- **Installable** - they are launchable from the device home screen as a standalone window instead of a browser tab

To implement the PWA concept, we use service workers, which allows having control over network requests. They can be configured to cache assets like images, icons, javascript or any other static files, to be always available on the application load, even when there is slow or no internet connection.
A service worker lifecycle consists of self registration into the browser during which mentioned static assets are cached. If an error occurs during installation the whole process fails and assets will be downloaded from the remote CDN server once again. Once the installation is finished, they will control all pages that fall under their scope. A basic configuration of service works can be seen on the listing 4.1.

```
{
 "name": "app",
 "installMode": "prefetch",
 "resources": {
  "files": ["/favicon.ico", "/index.html", "/
   manifest.json", "/*.css", "/*.js"]
 }
}
```

Listing 4.1: Service workers configuration

In the resources.files section we include the necessary cacheable files. In our case those are mainly css and javascript files. Although service workers are powerful tools, they can hijack connections, fabricate, and filter responses, that's why they are, by default, allowed to only be used via HTTPS connection to avoid man-in-the-middle attacks.

After we registered service workers and told them to cache asset files such as JS and CSS, the end result was 1 second improvement in application fully initialization time (figure 4.9), which overall is not so much a noticable difference, nevertheless now all users can place our app onto their deceive desktop, mobile or PC, access it via icon like a native application and it will even launch even when no internet connection is available.

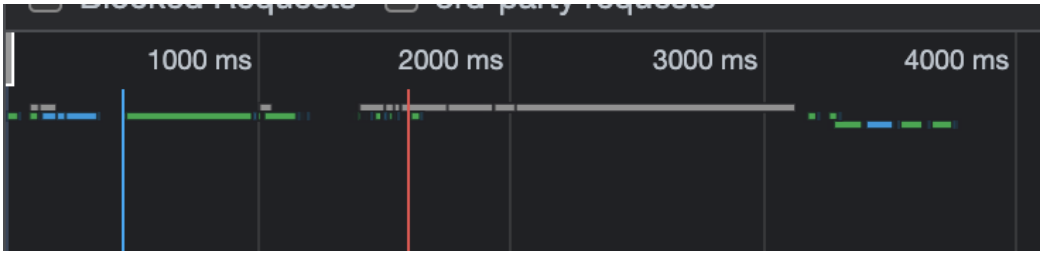


Figure 4.8: Network call without service workers

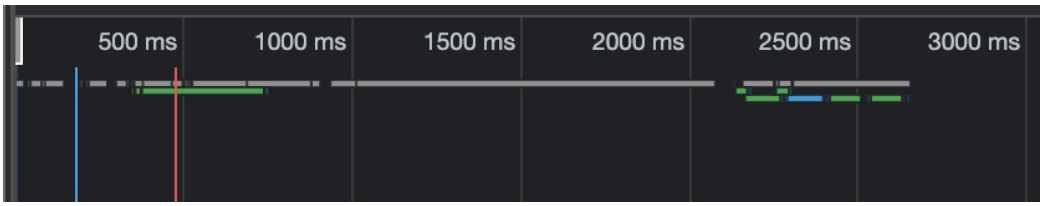


Figure 4.9: Network call with service workers

An appropriate case study is twitter[19] when in 2017 with more than 80% of users on mobile, they wanted to create a more engaging experience and have an all in one solution, an app which could be installed on any device. They created twitter lite, a PWA solution of their existing main application. It relies on cached data as much as possible, reduces data consumption by 70% and installation takes just 600KB compared to 23.5MB of their native app. By adding PWA as a business solution, they have noted a 75% increase in Tweets sending since then.

4.5.5 UI navigation

Let's briefly describe the application's overall navigation and behavior depending on the type of logged in user on the currently deployed version 0.9.8.2. We want to have some basic functionalities available to all incoming visitors without creating an account, to try out this system, and the rest of functionalities only to authenticated users. Right now we support two major features, which are stock analytics, that is available for all non-registered users, and portfolio tracking analytics for registered members. All the figures which we will be referred to by the following section are located at the end of the thesis, in chapter Attachments.

Non-registered user - by the initial load, the user is redirected to a page displaying present market conditions, we show the best and worst performing stocks on a current day with their daily price and volume percentage change (figure 7.1).
We also include crypto currency related data, historical statistics about bitcoin and price change for the top currencies (figure 7.2), but those data are limited, the whole segment is in development, so in the future more crypto related information will be shown.
Main advantage of this part is market overview, where plenty of historical data is displayed like S&P500 metrics, inflation rate for multiple countries, investor sentiment, corporate bond yield, consumer price index, etc. (figure 7.3). Users also have a possibility to create their own custom chart from these data and explore whether correlation exists between them (figure 7.4). Other areas of this section include displaying present news from different sources for different companies, calendars about ongoing or upcoming events like IPO, earnings dates and paid dividends. Last but not least data about major ETF indexes (figure 7.5) and market sector performance is provided.

Sometimes when we are looking for a company, we do not look for a specific one, but we are trying to find multiple ones which satisfies our criteria. To input these criteria, an advance searching system has been created, using which we can filter out stocks, based on price, market capitalization, beta, trading volume and so on (figure 7.6). Right now only from NYSE and NASDAQ exchange are accessible.
After picking a stock, user is redirected into the company details page, where he can examine its financial strength or growth, valuation, recommendation and buy rating by other analysts, quarterly and yearly financial growth, institution and fund holder and plenty more. (figures [7.7, 7.8, 7.9, 7.10]).

At this part we also provide, but it is still work in progress, calculation of a company's intrinsic value. One thing is that a ticker is traded at some price, another is an ability to decide whether that trading price is worthy, that's why we added discounted cash formula and earning valuation (figures [7.11, 7.12]), which are hypothetical valuation formulas.

Registered user - once an account is created, additional options are accessible, however the first step at each account is to activate it. In this current version 0.9.8.2, to receive live data about stock symbols, users have to create an additional account at Finnhub service provider. Once they have a free account there, they just copy-paste their generated private key into the our account page to establish realtime connectivity with Finnhub up to 50 different symbols. They can find the whole explanation highlighted by yellow color (figure 7.13).

After account activation, trading is available. Whenever a user buys a symbol, statistics about his performance are generated at the page dashboard. He finds there his current balance, portfolio change over time, risk evaluation (figure 7.14), balance movement relation to S&P500 (figure 7.15), holdings and sector allocation (figure 7.16) and last transaction history (figure 7.17).

To party users who are from the same class, groups have been created. Only authorized users are able to create a group, but depending on the group settings, anyone can request an acceptance into the groups or just only the group owner is able to invite people. Groups are great to bring competition between members (figure 7.19) and the same statistics are computed for each group as for an individual (figure 7.18).

Last page, that is worth mentioning, is the admin page. This is right now only available for the admin, to the system owner, to see created accounts (figure 7.20) and check all open tickets from users. In the future I plan to add technical information like changelogs, system error buffering, time when cloud functions were executed and so on, admin stuff.

Chapter 5

Deployment

In the past, but still persisting to this day, machine virtualization was the most common way of application deployment. In short, virtual machines emulates the behavior of an entire system, complete with its own operating system, storage and I/O operations, but eventually shares a single physical machine. Hypervisor is the responsible software for managing and provisioning hardware resources, like memory and storage and schedules operations in VMs to prevent overrunning in resource usage. VMs are good for testing new applications, because they are isolated from the host OS and do not impact the underlying hardware which eventually boosts security. Finally they reduce cost since we can run multiple virtualizations of several applications on one computer[11].
Despite all its benefits, virtual machines still have their share of problems. They are large entities, where each one contains a full operating system, a system administrator has to manually scale the application based on user load and also replicate the same running environment as system resources fill up, which requires an engineer to clone and replicate the whole virtual machine with all its settings. Being aware of all pros and cons using virtual machines, a more modern way of application deployment will be needed, like docker containerization.

5.1 Deployment by docker

Docker is a tool that allows to package up an application with all its parts, such as libraries and other dependencies and ship it out as one package[5]. Unlike virtual machines, docker allows applications to use the same Linux kernel as the underlying system and containers are compartmentalized from one another. With the absence of an operating system in each container, we use less computer resources, like memory, and spin up times are significantly higher. Containers are an execution environment of any application, easily transmittable to another hosting operating system, created by docker images. Images are templates, also referred as snapshots, that represent an

application and its virtual environment at a specific point of time. Images are immutable files, built by dockerfiles, that contain source code, libraries, dependencies and files needed for an application to help us build one or more containers, which itself are just a running instance of an image. When we are talking about multiple containers, multiple instances of the same application, it's also worth pointing out tools like kubernetes to manage scaling these containers.

In the study Performance evaluation of Docker Container and Virtual Machine[2], researchers tested out a series of experiments measuring different benchmarks to which of these two options conduct a better result. The measuring parts were CPU performance, memory throughput, storage read and writes, load test, and operation speed measurement. Overall they observed that docker containers exceeded VM performance, as the presence of QEMU layer (open-source image virtualizer to emulate other OS on top of the main OS) made VM less efficient, but I would highlight 2 tests. First one was an operation speed test, where they tested eight queen's problems (placing eight queens on an 8×8 chessboard such that none of them attacked one another) and the second one was a load test measuring the number of requests per second a given system can tolerate in which docker achieved better outcome than VM. The above mentioned results are important to conclude which deployment technique is better to handle API calls and perform mathematical operations.

By dockerizing a node application, the data provider server (section 4.3), we desire to replicate our local environment and install all required dependencies and libraries to an isolated container to be able to deploy them on the cloud for global accessibility. Configuring a docker image is done just in a few step showed by the following commands (listing 5.1). Briefly described, we pull an existing node image from docker-hub, create a working directory, copy the metadata of our dependencies, install them, then copy our source code into the work directory, build the project and start it.

```
1 FROM node:14.16-alpine
2 WORKDIR /app
3 COPY package.json .
4 RUN npm install
5 ADD . /app
6 RUN npm run build
7 RUN ls -la /
8 CMD ["npm", "start"]
```

Listing 5.1: Docker image configuration

Building and running this containers takes roughly 325MB space on the hard drive, compared to an installed Ubuntu operating system, taking around 4.5GB, when choosing virtualization instead. By dockerization we only achieved creating a running instance of our server, but to expose it, a cloud deployment must be also accomplished. Sticking to google services, cloud run[22] is the best strategy. It is a managed compute platform that provision containers, automatically scale them up or down based on demands, and produce GUI metric monitoring. Automatic scaling is important, even to zero instance, because when there is no demand from users, the server can be shut down, avoiding paying unnecessary costs for unutilized time. Figure 5.1 shows details about our running instance, which requires only 521MB of memory, is limited to scale only to one instance and shuts down after 300 seconds of no usage.

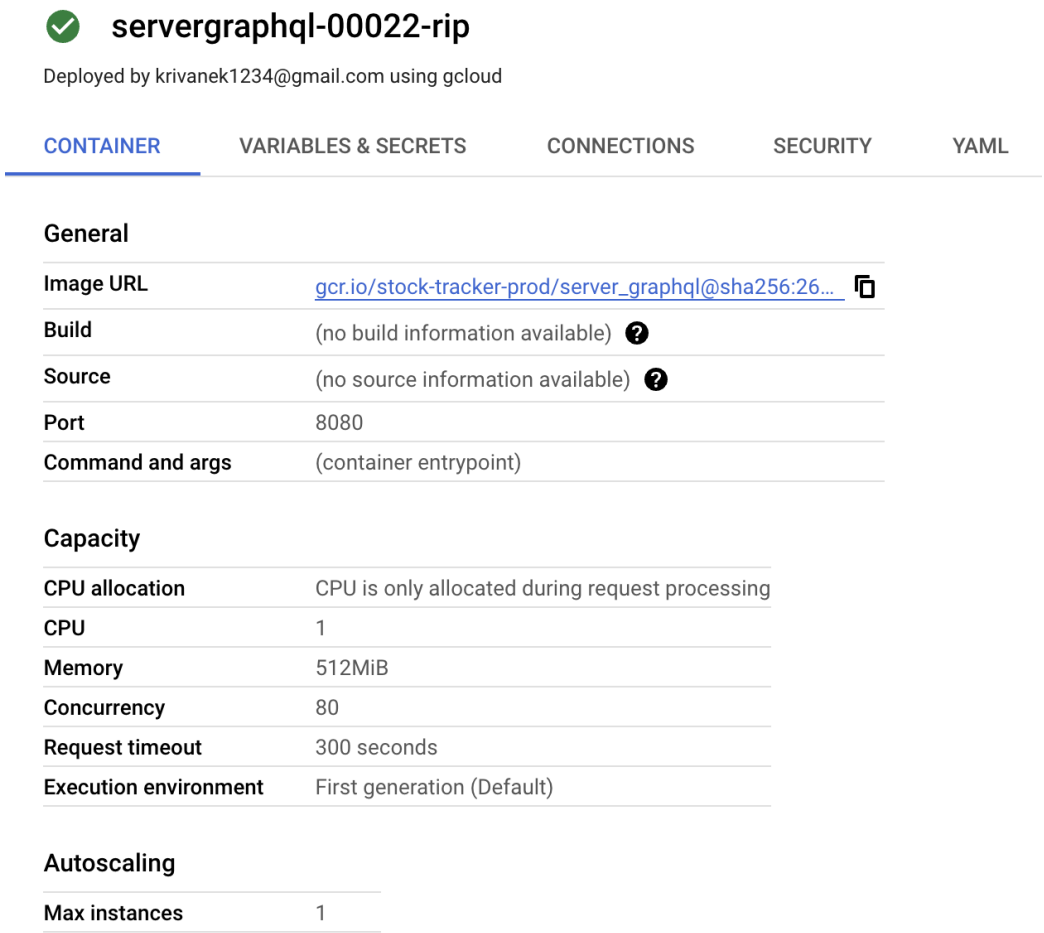


Figure 5.1: Cloud run details

Chapter 6

Testing & results

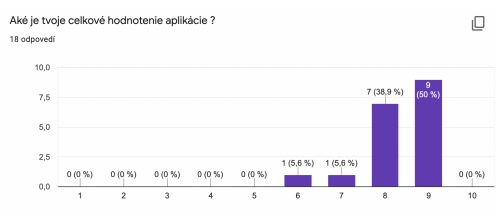
To verify the application's usefulness, all its functionalities, market details and portfolio building mechanism, a niche group of people, who are oriented in finance had to be found. To fulfill this task, in October 2021, I presented, for students at the lecture on financial mathematics, this application. The main idea was to replace the then many years used application from Finamis by a fresher one, which had to include all previously used functionalities, but also bring new ones. The testing period (3 months) was supervised by Mgr. Tatiana Jašurková and doc. Mgr. Igor Melicherčík, PhD. , teachers of this lecture, where the application has been showed and then used by 70 of their students. Students goals were to create a portfolio of stocks and diversify their risk into multiple assets following the Markowitz model.

6.1 Students and lecturer feedback

Students after completing their lectures, 18 of them submitted their feedback on the application. Overall everybody acknowledged the application's value brought to the course completion (fig. 6.1a) and would happily recommend it to other schools or classmates (fig. 6.1b). The average rating was between 8 - 9 points (fig. 6.2a), but this one may be misleading, because the majority of students haven't used any other exchange platform before, and couldn't compare it to anything else. To my surprise, when I asked them whether they would continue using the app after the course has ended, the majority of them answered "maybe" (fig. 6.2b). This may suggest that with future improvements the application may bring so much value to a person that no external motivation, like school courses, will be required to use it on a frequent time basis, whether it is just for practicing trading or analyzing stock details. The whole review of their lecturer Mgr. Tatiana Jašurková can be found in the attachments [TODO].

(6.1a) Feedback application purpose

(6.1b) Feedback suggest elsewhere



(6.2a) Feedback rating

(6.2b) Feedback future usage

As the biggest positive value, that students pointed out was data visualization. Seems like, in general, people tend to like more data representations through graphs compared to reading just raw data. In the future work I will focus on adding more visual elements whether we are talking about stock detail representation or user statistics.

A major benefit using this application oppose to others was in the functionality of portfolio comparison. Students were divided into groups where at the end of the day the position of each member changed based on their current assets balance which eventually resulted in a small competition[todo picture in attachments]. All of the users portfolio performance were also compared to the major index S&P500[todo picture in attachments]. Stretching out the idea of comparison, a new page will be introduced to users called Hall of fame, where they will be able to see users with the highest portfolio, best and worst performers in a specific time interval (day, week, month). Also users will be able to compare their performance with other indexes or even users, not just with S&P500, and see side by side their portfolio statistics, as this is one of the most requested addition from the gathered feedbacks. Another suggestions from students were:

- Adding more exchanges to find stocks (right now only NASDAQ and NYSE are available)
- Improve trading mechanism by allowing trading ETFs, crypto, treasuries and adding basic functionalities such as short selling and limit buy
- Possibility to download any personal or asset related historical data in a visual format (png) or as raw data (csv / json)
- Implement notifications, dispatched by actions related to an user to bring real time behaviour

6.2 Testing from a technical perspective

During testing, from the technical perspective, all functionalities worked as expected, but two major failures happened, both related to the same function, a cloud function, which evaluated users portfolio at the end of day. This function for each user adds up their cash on hand and their current invested amount, which is calculated by the current value of each asset they hold and save that value into the databse.

The first failure was a timeout failure of that cloud function, which is 5 minutes. Timeout happened because an user can hold up to 50 different stocks and we had around 70 active users. In the worst case scenario, each user could hold different symbols from each other, where eventually we would end up with 3500 unique stocks in the database. When this function is called and fetches the current value of any symbol it takes around 0,6 second per symbol, a http request is made to an external paid API, to get the data. Summing up the total time, we end up with 35 minutes for 3500 symbols, in our worst case scenario, which result in function timeout (5 minutes). This problem was resolved by creating a cron job to periodically run this function multiple times in a small time window and subsequently load only a tiny chunk of users (around 25), updating only their portfolio, but also caching received symbols current price in a key-value variable, to avoid multiple http round trips for the same symbol.

The second major failure of this function happened for lack of proper testing. By evaluating the user portfolio, its daily increase is also computed, which is just a difference of the porfolio current balance and the day before. However, when we have a fresh user, newly registered, and we run this function, the user doesn't have any previous balance saved and the computation results in an undefined value, which is saved in the database. This itself is not an error, but when we load those data from the database using graphql, an error has been thrown because only number values were expected, but somehow an undefined value was received. Overall this resulted in not returning user's values at the authentication section and they could not log in into their account. Solving this mistake was just by adding additional conditions to prevent storing undefined values in the database.

6.3 Technical improvements

This testing period showed that by deploying an application into the production, there will always be a change of some inaccuracy and bug occurrences, that's why an error caching mechanism will be needed to be introduced. As mentioned above, this testing period was supervised by Mgr. Tatiana Jašurková, who reported all mistakes immediately, but we have to account that not all users will be able to submit tickets and we, as developers, have to be notified by some external system if any fault arise.

Another important change will be disposing of an external service called Finnhub. Right now to achieve real time stock price updates, all users have to have an account in Finnhub system to get a private API key, which is used to initialize websocket connection with Finnhub and listen on real time stock price updates. This is inconvenient for the user, because he has to register to another website to use our application and also, more importantly, this real time service from Finnhub is an unpaid free service, meaning they can cancel it anytime and we will no longer receive real time price updates. Fixing this issue will be to find an affordable real time data provider and adjust the frontend to have a websocket connection with a new custom backend microservice, which will eventually be always connected to that new external data provider.

Chapter 7

Conclusion

The goal of this thesis was to implement a competitive Paper trading application, which enables tracking the growth of stock indices and mediates their trading in real time through virtual money. At first we dived deep into time series forecasting, in order to successfully predict a stock price movement using the ARIMA model. We chose Apple stock, divided its historical data into 80% training, which were put into the ARIMA model in hope to correctly forecast the remaining 20%, however, the further we forecasted, the more inaccurate data we received. Concurrently with this forecasting, we also examined portfolio risk analysis, computing crucial metrics for the users to understand their risk of investments.

After that we specified the technical and non-technical requirements for our application and built two servers for different purposes, one as a REST server, which gathers financial data from multiple APIs, and the second one as a GraphQL server, which communicates with our client side application, but also persist data in firebase database. Both were dockerized and deployed to the cloud. For asynchronous tasks, to perform some calculation at a specific time interval, we used cloud functions with cron scheduler. The client side application was written as a single page application, applying various caching mechanisms and assets persistence on the user's device for faster start time. The application was tested from October 2021, for 3 months, by roughly 70 students of financial mathematics in the UK. Students after completing all the lectures, they received a survey, submitted their overall opinion and suggested some feature additions for future development.

In conclusion the idea of having such a Paper trading system was received gratefully and was used frequently during that testing period. However to publicize this system to a broader group of people, for example to multiple schools, some fixes and additional functionalities are required. The first improvement will be to remove the dependency on Finnhub data provider and also bring a framework, like NestJS, alongside with Apollo server, to more efficiently build scalable web apps.

As non-technical improvements, students suggested enabling trading assets like cryptocurrencies, implementing limit and stop loss functions for trading, and finally adding more properties by which users can search ticker symbols.

Bibliography

[1] Agrawal R. K. Adhikari R. *An Introductory Study on Time Series Modeling and Forecasting*. LAP Lambert Academic Publishing, 2013. URL: https://arxiv.org/abs/1302.6613.

[2] Shivaraj K. Amit M P. Narayan D G. and Mohammed Moin M. "Performance Evaluation of Docker Container and Virtual Machine". In: (2020). URL: https://www.researchgate.net/publication/341907958_Performance_Evaluation_of_Docker_Container_and_Virtual_Machine.

[3] *Best practices in cloud applications*. URL: https://docs.microsoft.com/en-us/azure/architecture/best-practices.

[4] Free code camp. *So what's this GraphQL thing I keep hearing about?* 2017. URL: https://www.freecodecamp.org/news/so-whats-this-graphql-thing-i-keep-hearing-about-baf4d36c20cf/.

[5] IBM Cloud Education. "What is Docker?" In: (2021). URL: https://www.ibm.com/cloud/learn/docker.

[6] Frank J Fabozzi and James L Grant. "Modern Portfolio Theory, Capital Market Theory, and Asset Pricing Models". In: (2001). URL: https://www.researchgate.net/publication/272157556_Modern_Portfolio_Theory_Capital_Market_Theory_and_Asset_Pricing_Models.

[7] Ezzine L. Fattah J. and Aman Z. "Forecasting of demand using ARIMA model". In: (2018). URL: https://journals.sagepub.com/doi/10.1177/1847979018808673.

[8] "GraphQL Advantages and Disadvantages". In: (). URL: https://www.javatpoint.com/graphql-advantages-and-disadvantages.

[9] "GraphQL vs. REST". In: (2021). URL: https://medium.com/codex/graphql-vs-rest-c0e14c9f9f1.

[10] Sayan Guha. "A Comparative Study Between Graph-QL Restful Services in API Management of Stateless Architectures". In: (2020). URL: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3640169.

[11] "How do virtual machines work?" In: (). URL: https://www.citrix.com/solutions/vdi-and-daas/what-is-a-virtual-machine.html.

[12] "How to choose your development framework and libraries?" In: (). URL: https://blog.devgenius.io/how-to-choose-your-development-framework-and-libraries-d36230960342.

[13] "Introduction to Angular concepts". In: (2022). URL: https://angular.io/guide/architecture.

[14] Elsaraiti M. and Merabet A. "A Comparative Analysis of the ARIMA and LSTM Predictive Models and Their Effectiveness for Predicting Wind Speed". In: (2021). URL: https://www.researchgate.net/publication/355429916_A_Comparative_Analysis_of_the_ARIMA_and_LSTM_Predictive_Models_and_Their_Effectiveness_for_Predicting_Wind_Speed.

[15] FATHI O. *Time series forecasting using a hybrid ARIMA and LSTM model*. URL: https://www.velvetconsulting.com/wp-content/uploads/2019/03/Seasonality_modeling_using_ARIMA_LSTM_Hybrid_Model.pdf.

[16] "Progressive Web Apps". In: (). URL: https://web.dev/progressive-web-apps/.

[17] "SOLID principles and Dependency Injection". In: (). URL: https://docs.microsoft.com/en-us/dotnet/architecture/microservices/microservice-ddd-cqrs-patterns/microservice-application-layer-web-api-design.

[18] Mark Stuart. *GraphQL Resolvers: Best Practices*. 2018. URL: https://medium.com/paypal-tech/graphql-resolvers-best-practices-cd36fdbcef55.

[19] "Twitter Lite PWA Significantly Increases Engagement and Reduces Data Usage". In: (). URL: https://developers.google.com/web/showcase/2017/twitter.

[20] William W. S. Wei. *Time Series Analysis. Univariate and Multivariate Methods*. 2nd ed. Addison Wesley, 2006.

[21] "What is a cron job". In: (). URL: https://www.hostinger.com/tutorials/cron-job.

[22] "What Is Google Cloud Run?" In: (). URL: https://www.bmc.com/blogs/google-cloud-run/.

[23] "What's the Diff: Programs, Processes, and Threads". In: (2017). URL: https://www.backblaze.com/blog/whats-the-diff-programs-processes-and-threads/.

[24] "Why Apollo Client?" In: (). URL: https://www.apollographql.com/docs/react/why-apollo.

Attachments

StockTracker - posudková správa

StockTracker je webová aplikácia ponúkajúca možnosť vyskúšať si obchodovanie na burze bez rizika straty vlastných finančných prostriedkov. Je tak vhodná pre každého kto sa zaujíma o dianie na finančných trhoch, sleduje vývoj finančných nástrojov, a chcel by si prakticky vyskúšať svoje vedomosti a znalosti pri zostavovaní portfólia a jeho následnom pozorovaní a zhodnocovaní. Keďže zároveň ponúka aj množstvo cenných informácii o dianí v ekonomike, rovnako ako aj správy týkajúce sa obchodovateľných spoločností, môže tak zaujať aj jednotlivcov, ktorý by sa chceli začať venovať práve finančnému trhu.

V rámci jednotlivých sekcii je možné sa dozvedieť nielen o novinkách týkajúcich sa jednotlivých spoločností, rovnako prístupné sú aj základné informácie o každej zo spoločností. Na jednom mieste tak nájdeme charakteristiku danej spoločnosti, množstvo ukazovateľov popisujúcich jej finančnú situáciu a v neposlednom rade i časový vývoj hodnoty akcií danej firmy. Toto všetko tak predstavuje výraznú pomoc pri hľadaní akcii, resp. ich ohodnocovaní, pri procese zostavovania efektívneho a úspešného portfólia, ktoré je následne možné v aplikácii nakúpiť a ďalej ho sledovať a pracovať s ním.

Aplikácia taktiež umožňuje vytvorenie skupín, ktorých členmi môžu byť viacerí jednotlivci a tak interaktívnym spôsobom poskytuje možnosť porovnať svoje portfólio s inými, či odpozorovať rôzne použité stratégie. Takouto hravou formou je tak jednotlivec nabádaný k zdokonaleniu svojho portfólia, pričom v tomto procese je rozvíjané jeho analytické myslenie.

Vďaka prehľadnému spracovaniu disponuje táto aplikácia intuitívnym ovládaním, a tak je ľahké sa v nej rýchlo zorientovať a nájsť všetko potrebné.

Vzhľadom na vyššie spomenuté kvality aplikácie StockTracker, tak z globálneho hľadiska je táto aplikácia veľmi perspektívnym nástrojom, ktorý môže poslúžiť či už jednotlivcom, alebo aj ako výučbová pomôcka, o čom sme sa napokon presvedčili i pri jej použití v rámci cvičení z Finančnej matematiky, kde si študenti vyskúšali práve zostavovanie portfólii na základe Markowitzovho modelu, jeho následný nákup a sledovanie jeho vývoja v predmetnej aplikácii.

Čo sa týka možných návrhov na zlepšenie, tak aplikácia síce už teraz ponúka bohatý výber dostupných aktív, ak by sme sa však zamerali aj na možnosť diverzifikácie portfólia z hľadiska meny, či krajiny, bolo by vítané doplnenie aj iných obchodovacích búrz, okrem NYSE a NASDAQ, ktoré sú zatiaľ v aplikácii dostupné. Z technického hľadiska by bolo vhodné popracovať ešte na funkcii vyhľadávania.

Keďže cieľom projektu z Finančnej matematiky, je nielen zostavenie portfolia, jeho nákup a správa, ale i finálne zhodnotenie výsledkov v rámci záverečnej správy, tak ďalším možným rozšírením aplikácie, ktoré by značne uľahčilo prácu používateľovi, by mohla byť možnosť priameho sťahovania obrázkov (rast portfólia, denné výnosy portfólia, porovnanie s inými portfóliami,...). Prípadne vhodné by mohlo byť aj doplnenie extrahovania údajov z grafov do csv, či xlsx súborov, vďaka čomu by mohol užívateľ ďalej s dátami pracovať a vykonávať na nich ďalšie analýzy. Práve v tejto možnosti StockTracker zaostáva v porovnaní s konkurenčným softvérom, kde stiahnutie údajov o výkonnosti portfólia je zakomponované. Rovnako žiaducim by mohlo byť i zakomponovanie nastavovania grafov priamo v aplikácii, ako napríklad voľba časového okna, ktoré sa má zobrazovať, prípadne pridanie možnosti porovnania portfólii dvoch, resp. viacerých používateľov, či iných akciových indexov.

Možnou nadstavbou aplikácie, vďaka ktorej by sa tento nástroj obchodovania ešte viac odlíšil od konkurenčných aplikácii, by tiež mohlo byť korektné nastavenie možnosti short-sellingu, teda prípustnosť krátkych pozícii v aktívach. Ďalším rozšírením je možnosť nákupu a predaja nielen celých kusov akcií, ale i ich čiastočných zlomkov. Táto možnosť by pomohla pri nákupe portfólia, ktoré presnejšie zodpovedá výstupu z optimalizácie Markowitzovho modelu, pokiaľ pracujeme s hodnotovými váhami jednotlivých aktív a nezavádzame dodatočné ošetrenie na cločíselný počet kusov akcií v portfóliu.

Napriek spomenutým návrhom na zlepšenie aplikácie StockTracker, považujem túto aplikáciu, už v terajšom stave, za veľmi dobre spracovanú, obsahujúcu širokú škálu informácii, či už o novinkách z finančného sveta, alebo o samotných aktívach. Aplikácia sa tak stala prínosom a obohatila i výučbu cvičení z finančnej matematiky, kde si žiaci na jednom mieste mohli vyskúšať hľadanie akcii, ich nákup a sledovanie vývoja. Z osobného hľadiska si myslím, že StockTracker má veľký potenciál stať sa a byť modernou, prehľadne spracovanou, jednoducho ovládateľnou aplikáciou, ponúkajúcou komplexný úvod do sveta obchodovania na burze.

V Bratislave 06. februára 2022 Mgr. Tatiana Jašurková

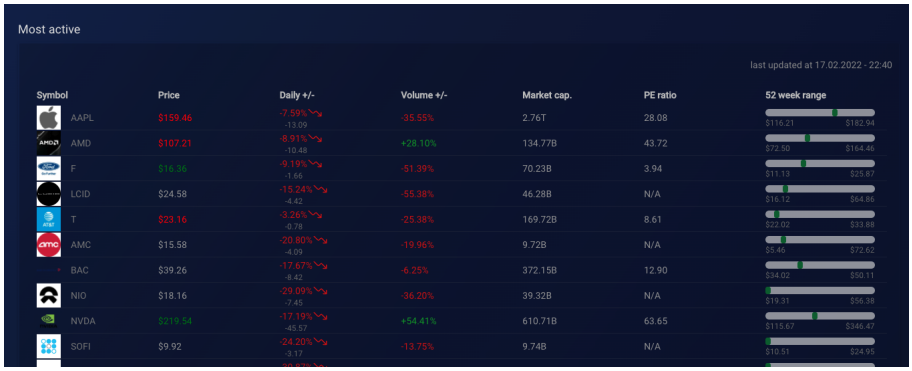


Figure 7.1: Market top stocks

Figure 7.2: Market top crypto currency

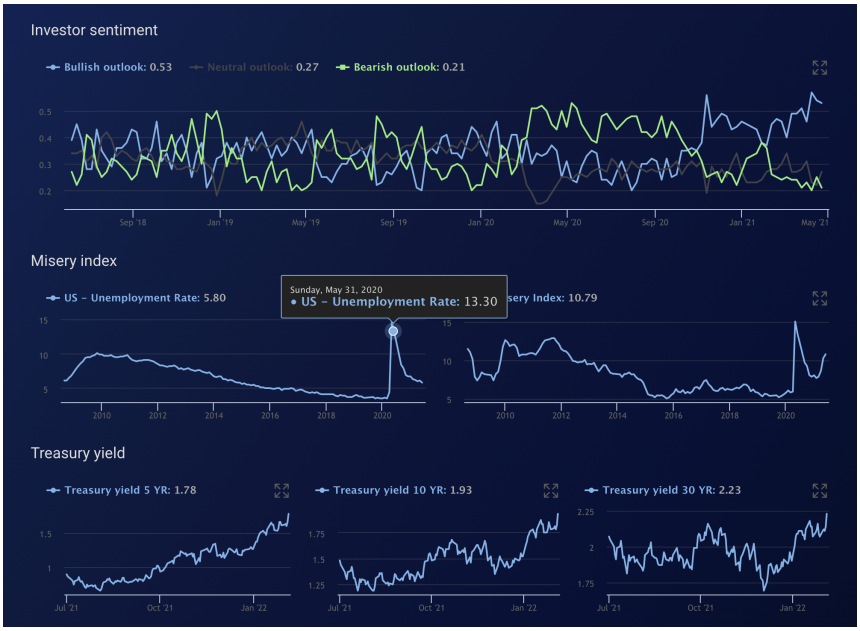


Figure 7.3: Market metadata

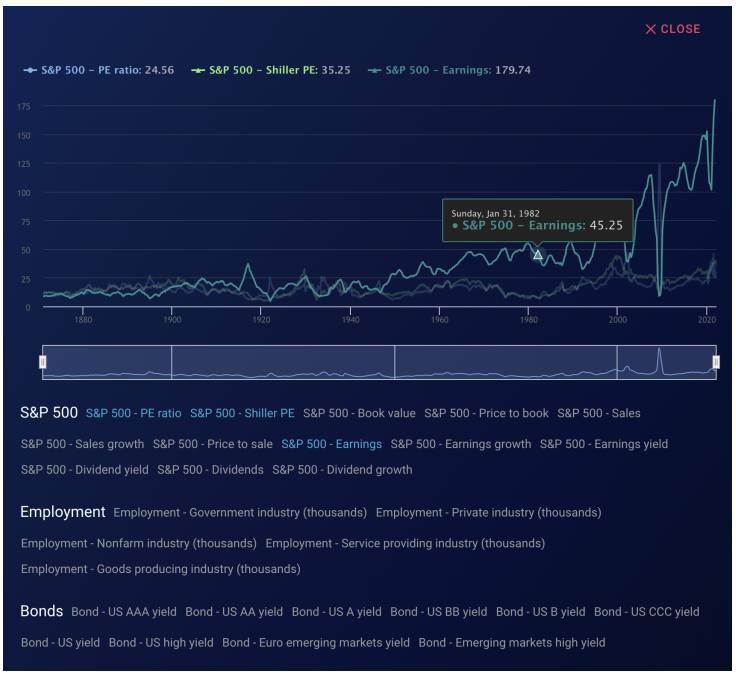


Figure 7.4: Chart builder

Figure 7.5: Market ETF

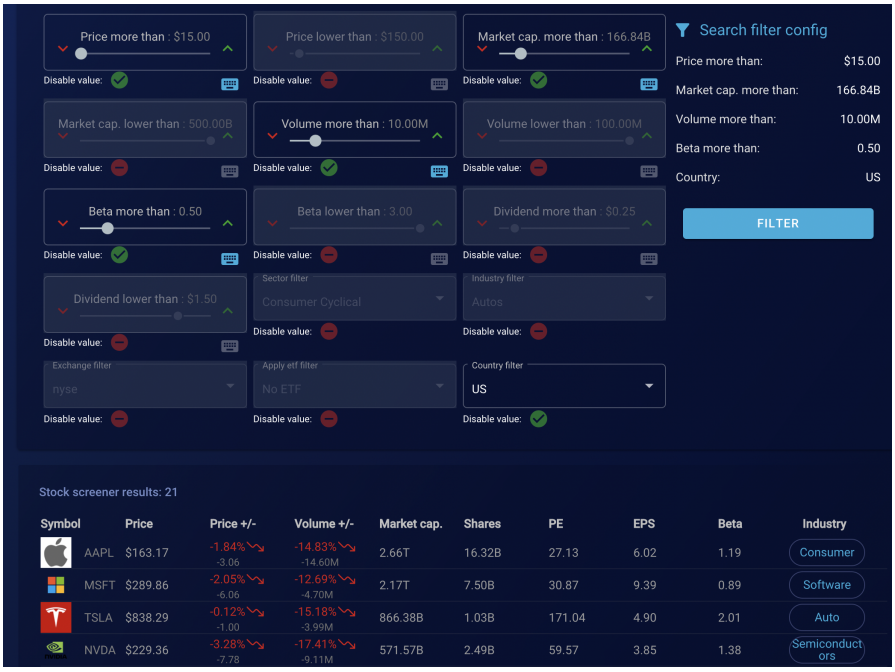


Figure 7.6: Search companies

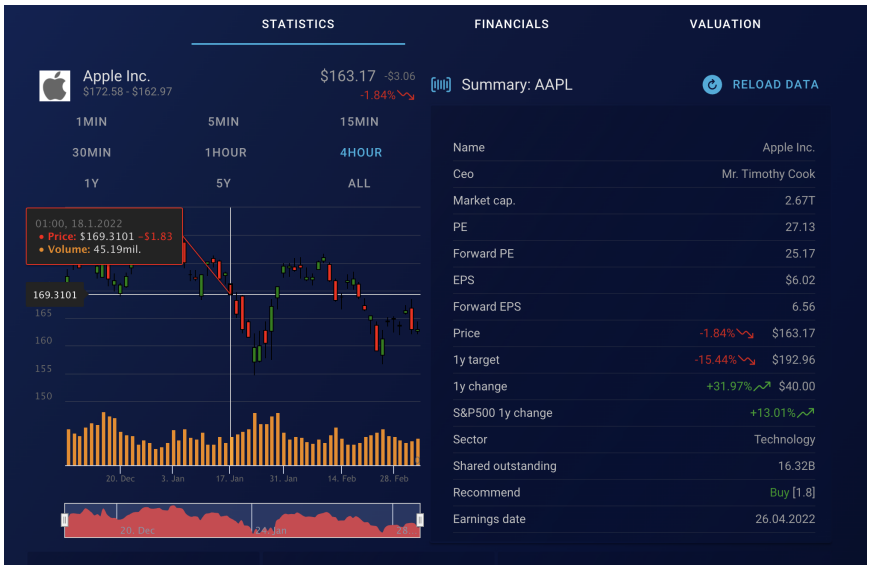


Figure 7.7: Company details 1

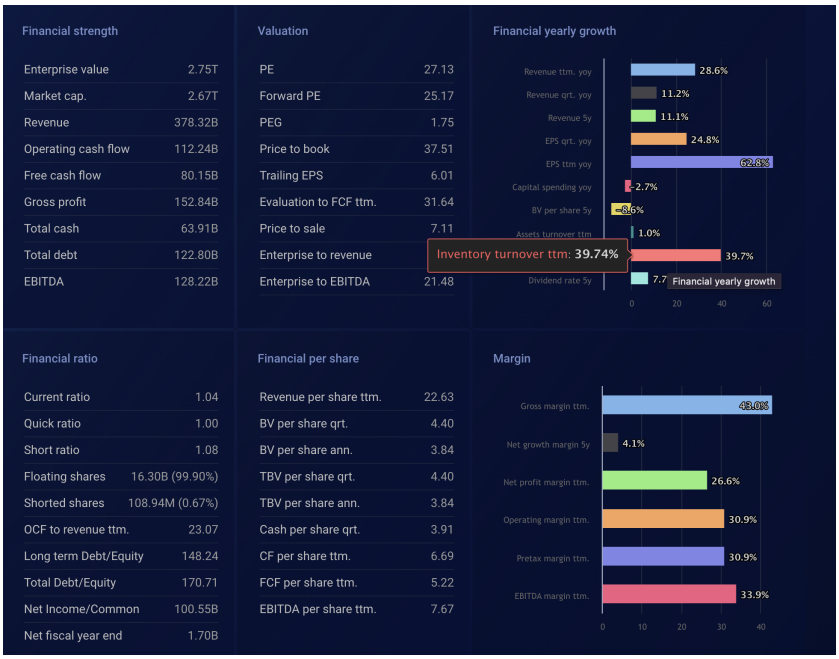


Figure 7.8: Company details 2

Figure 7.9: Company details 3

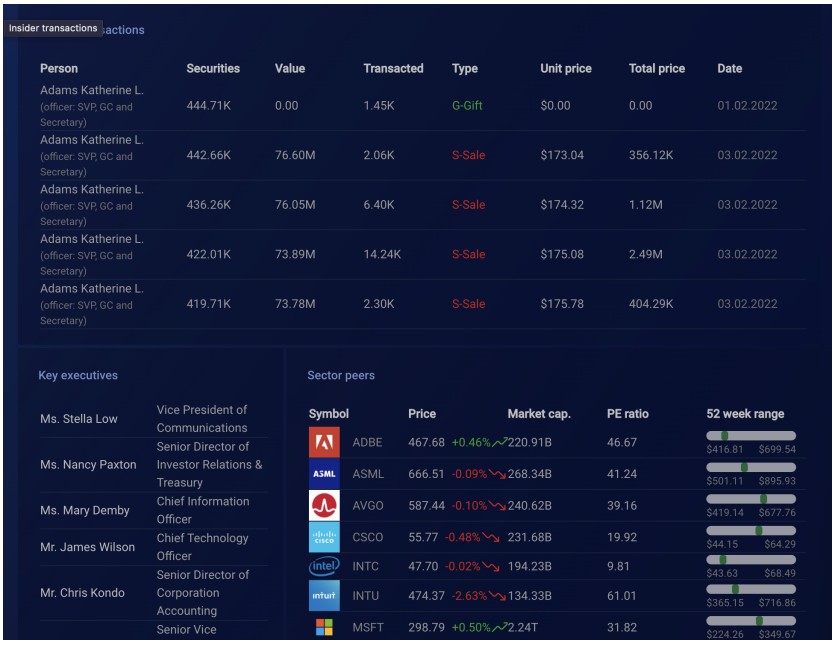


Figure 7.10: Company details 4

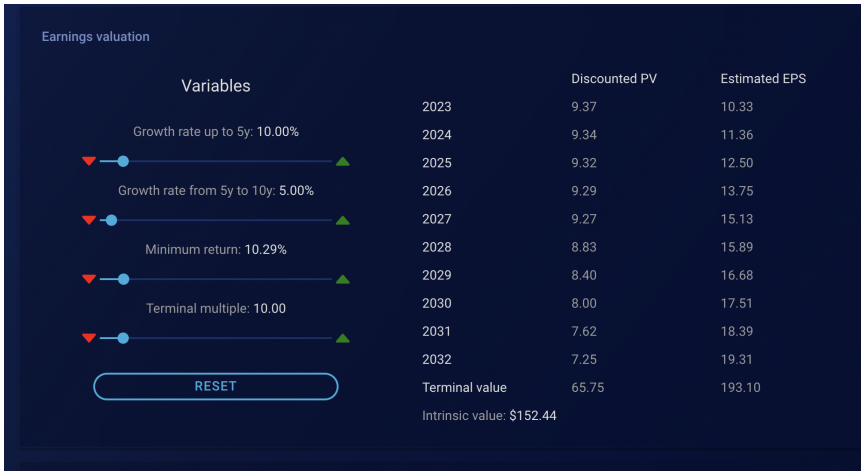


Figure 7.11: Earnings valuation

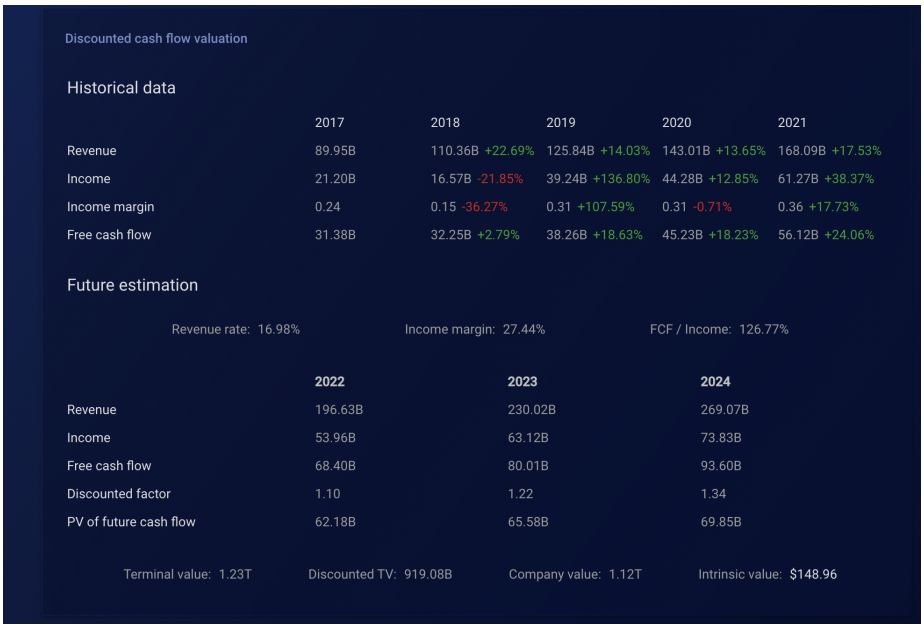


Figure 7.12: Discounted cash flow valuation

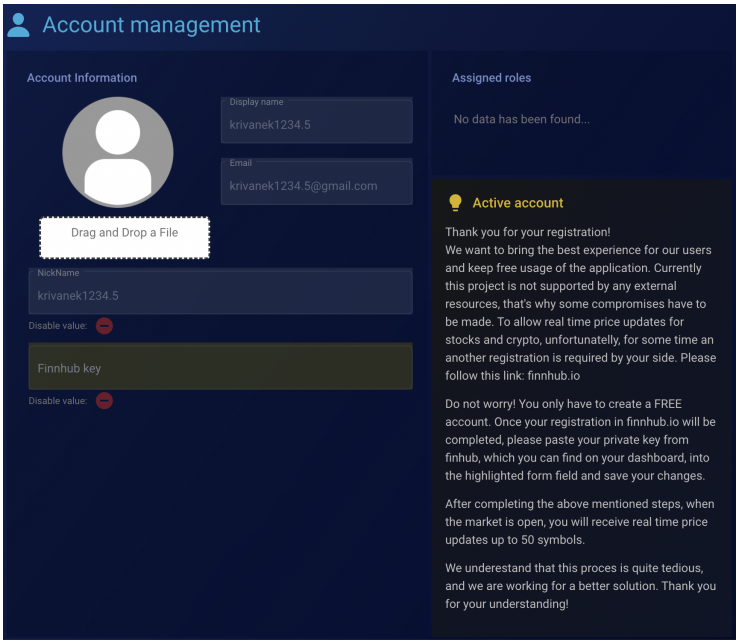


Figure 7.13: New user account settings

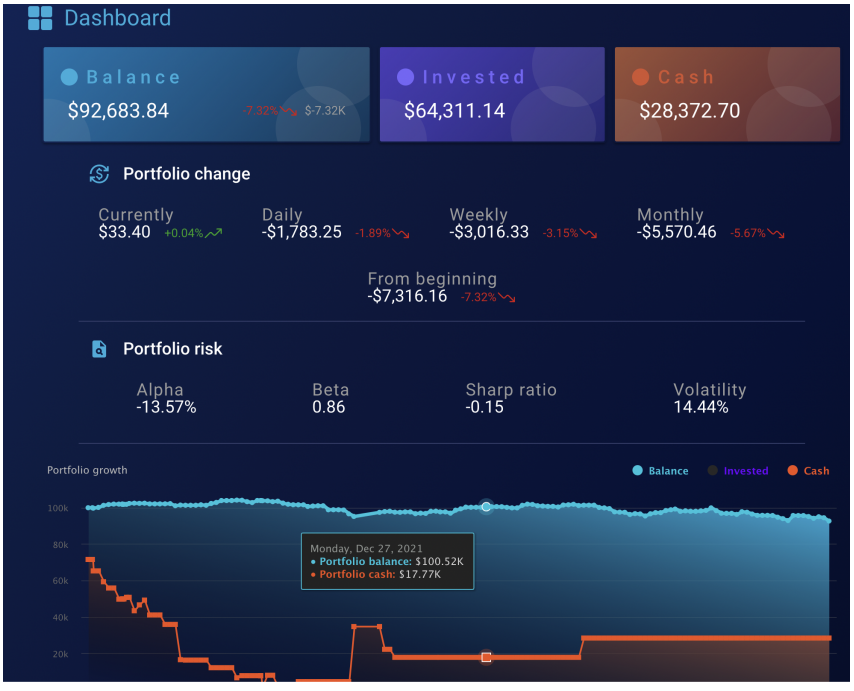


Figure 7.14: Dashboard - user portfolio

Figure 7.15: Dashboard - user portfolio against S&P 500

Figure 7.16: Dashboard - user portfolio holdings

Figure 7.17: Dashboard - transactions

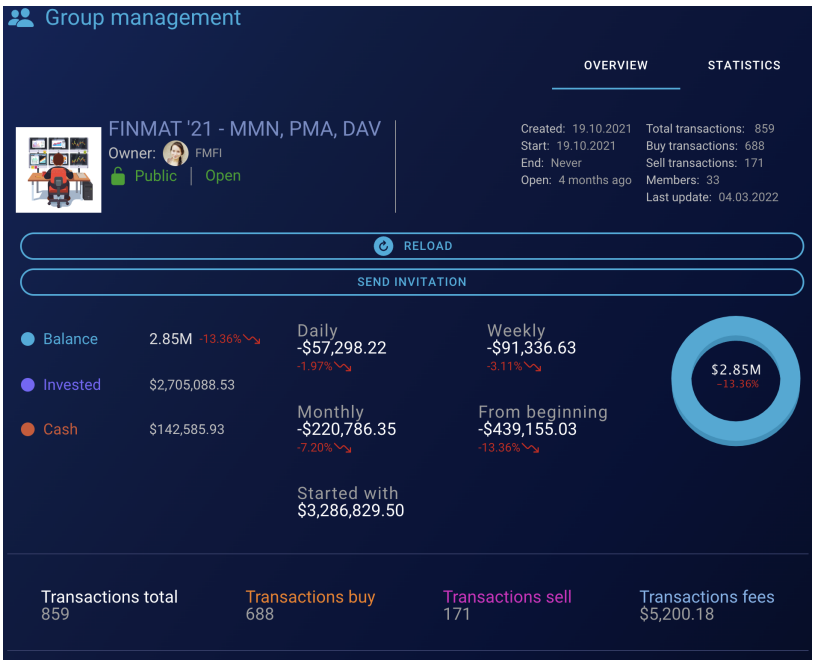


Figure 7.18: Group - statistics

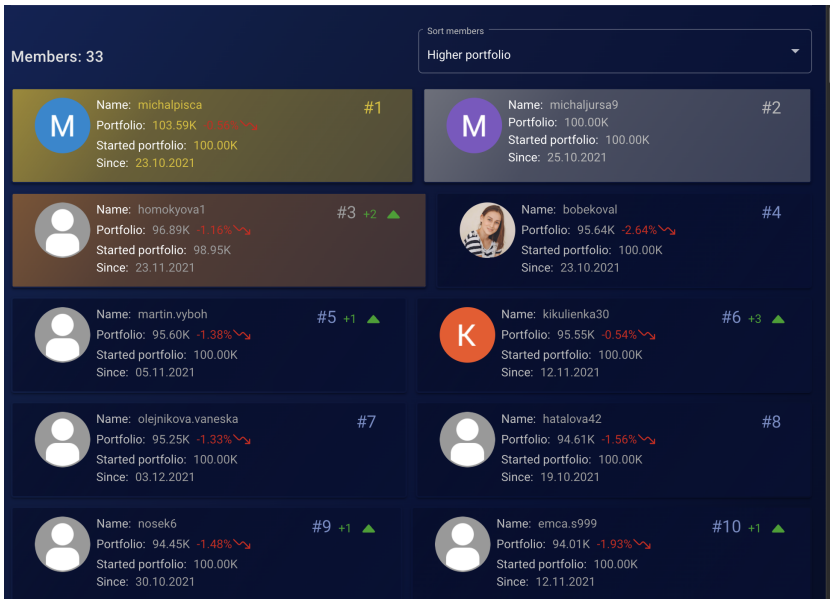


Figure 7.19: Group - members

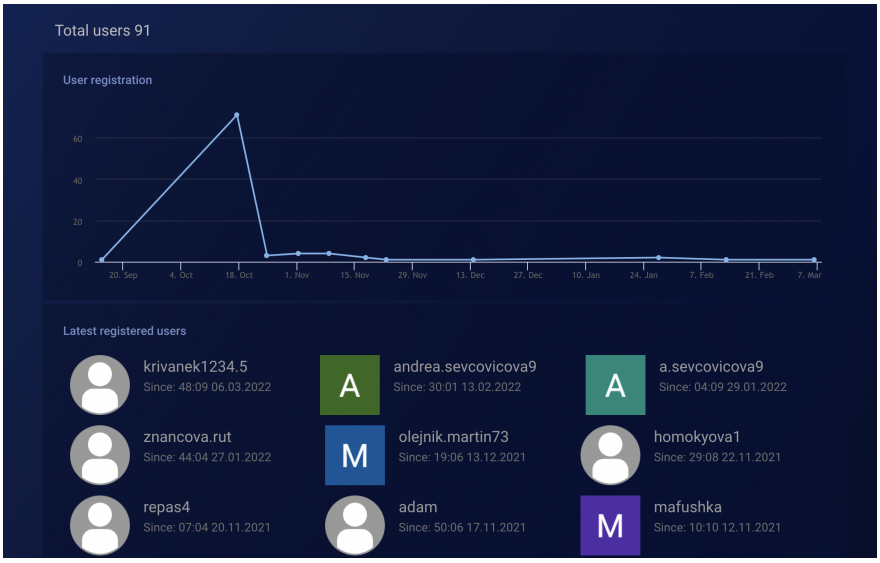


Figure 7.20: Admin - registrated users