VYSOKÁ ŠKOLA BÁŇSKÁ – TECHNICAL UNIVERSITY OF OSTRAVA

FACULTY OF ECONOMICS

DEPARTMENT OF APPLIED INFORMATICS

Návrh a implementace aplikace pro vizuální analýzu a monitorování finančních instrumentů StockInSight

Design and Implementation of Application for Visual Exploration and Monitoring of Financial Instruments StockInSight

Student: Vít Chrubasík, BSc (Hons)

Bachelor thesis supervisor: doc. RNDr. Ivo Martiník, Ph.D.

Ostrava 2020

VŠB - Technical University of Ostrava
Faculty of Economics
Department of Applied Informatics

Bachelor Thesis Assignment

Student: **Vít Chrubasík**

Study Programme: B6209 Systems Engineering and Informatics

Study Branch: 6209R017 Informatics in Economics

Title: Design and Implementation of Application for Visual Exploration and Monitoring of Financial Instruments StockInSight
Návrh a implementace aplikace pro vizuální analýzu a monitorování finančních instrumentů StockInSight

The thesis language: English

Description:

1. Introduction
2. Theoretical and Methodological Basis of StockInSight Application Development
3. Analysis of the Current State
4. Design and Implementation of StockInSight Application
5. Conclusion

Bibliography
List of Abbreviations
Declaration of Utilization of Results from the Bachelor Thesis
List of Annexes
Annexes

References:

SUBRAMANIAN, Vasan. *Pro MERN Stack: Full Stack Web App Development with Mongo, Express, React, and Node, Second Edition.* Berkeley, CA: Apress, 2019. ISBN 9781484243916.
LIM, Mark Andrew. *The handbook of technical analysis + test bank: the practitioner's comprehensive guide to technical analysis (1st ed).* Singapore: Wiley, 2016. ISBN 9781118498927.
PRESSMAN, S. ROGER and Bruce MAXIM. *Software engineering: a practitioner's approach.* New York: McGraw-Hill Education, 2015. ISBN 9781259253157.

Extent and terms of a thesis are specified in directions for its elaboration that are opened to the public on the web sites of the faculty.

Supervisor: **doc. RNDr. Ivo Martiník, Ph.D.**

Date of issue: 22.11.2019

Date of submission: 07.05.2020

VYSOKÁ ŠKOLA BÁŇSKÁ-TECHNICKÁ UNIVERZITA Ekonomická fakulta 1 OSTRAVA

Ing. Petr Rozehnal, Ph.D.
Head of Department

doc. Ing. Vojtěch Spáčil, CSc.
Dean

Declaration

I hereby declare that I have elaborated this entire Bachelor Thesis, including the annexes, independently based on the used literature and under the guidance of my supervisor.

Ostrava 19. 7. 2020

Vít Chrubasík, BSc (Hons)

Thanks

I would like to express my thanks to my project supervisor doc. RNDr. Ivo Martiník, Ph.D. for the guidance and the helpful feedback.

I want to express my gratitude and appreciation to University of Huddersfield, and VSB Technical University of Ostrava for providing a great academic platform to carry out the project even during restrictive times.

Finally, I want to thank my family and friends for supporting me in my endeavours.

Contents

1 Introduction

The incorporation of "intelligent" machines is revolutionizing and transforming almost every aspect of the economy and the industry. The financial system is no exception to the rule. It is general knowledge that nowadays around 90 % of all orders on the stock markets are being placed by a machine based on algorithms that operate on historical data from the market. The infallibility of a machine is smaller in orders of magnitude in comparison with that of a human being, most of the time negligible, and the computation power astronomically larger. Given an accurate enough model of the economical universe of discourse, a machine can accurately infer its future behaviour and capitalize on predictions in a matter of fractions of a second. Those abilities are exploited by financial institutions acting on the financial markets to make even more informed economical decisions.

Private investors (individuals and small companies) do not have easy access to tools for analysing large amounts of historical data and must, therefore, either rely on only the traditional techniques from the fundamental analysis or sacrifice portions of their returns by vesting their liquidity into managed funds.

Nowadays, it is infeasible for an average individual to pursue the investing endeavour on their own. A private investor needs to compete against both private, and institutional investors, the institutional investors being specialized institutions (banks, funds, insurance companies) equipped with high levels of expertise. Those institutions thrive in the system by offering choices for investment to private investors, cutting their returns in exchange for lowered risk.

Most of the tools available are either too complicated or require a financial commitment. Paid tools are closed source sic non-transparent. This caveat emptor approach is known to bring potential security risks to its user base. There exist tools on the market that enable the investor to incorporate data analysis to some degree, those, however, are usually commercial, and closed-source, therefore they limit their use to what is currently offered and prohibit any extensions from third parties, thereby slowing the development.

The main purpose of this project was to explore ways of making investing more viable for a private investor with a focus on stocks investing, and to provide pointers that

in future would lead towards delivering an aurea mediocritas solution for a private investor.

The project was meant to explore the state-of-art techniques and open-source software tools for making informed financial decisions about stocks that are available on the market, and from those create a design of a possible solution. The other objective was to implement a prototype to experiment with data reporting.

2 Theoretical and Methodological Basis of StockInSight Application Development

This chapter outlines the theoretical foundation for the StockInSight project. The chapter touches the topics relating to the big data and financial markets with the focus of stock markets and stock exchanges.

2.1 Data Science

Data Science is a broad word for an inter-disciplinary field that is now evolving more than ever. The data science discipline always starts with data and with the application of scientific, mathematical and computational methods tries to uncover-*mine* new information from that data.

(Kotu, et al., 2018) outlines *6 major fields* associated with data science.

Descriptive statistics provides a language to communicate the characteristics of the dataset. Descriptive statistics possesses an arsenal of techniques from mathematics. Those are e.g. mean, standard deviation, kurtosis, skewness, correlation and many more. *Exploratory visualization* is oriented towards the human understanding of data. It is integral to the data science process. To pick a salient subset of the dataset is the task of *Dimensional slicing.* It entails grouping the data, filtering it and generally working with data queries, mainly in the environment of databases. The *Hypothesis testing* (also called *confirmatory data analysis*) serves to confirm or reject beliefs about the world by utilizing a data-driven approach.

Data engineering process is concerned with sourcing, organizing, assembling, storing and distributing data for effective analysis and usage. *Business intelligence (BI)* is a field that interconnects the previously mentioned fields to help businesses use data both more efficiently and efficiently.

2.2 Stock Markets

t is the purpose of every financial system to facilitate the flow of loanable funds from entities with a surplus to entities with a demand. Those funds are being represented by various financial instruments. Financial markets are vital subsystems that provide a platform for such flows. Capital markets are part of the financial markets' taxonomy. They operate with long-term debt and equity. Companies offer their share of equity in the

form of stock security to raise the capital for their ventures. The stock market is a capital market, that houses trades of those stocks. The stock markets are housed on stock exchanges. Stock exchanges allow their members to trade stocks. It is hard/expensive to get a membership on the exchange, therefore, many members provide broker services for outside investors. (Henning, et al., 1978)

The value of a stock is an intersection between the supply and demand on the exchange. Participants in the exchange place buy orders and sell orders – they offer to buy/sell some quantity of a given stock for a given price. A transaction happens when a buy order and sell order have matching prices. Each such transaction serves as a current stock price for a given stock for a given market at a given time. Over time the historical data accumulated is grouped into time intervals, each time interval has the following values:

- open price. The transaction price which the interval starts with.
- close price. The transaction price which the interval ends with.
- high price. The transaction with the highest price for the given interval.
- low price. The transaction with the lowest price for the given interval.
- volume. The number of transactions for the given interval.

2.3 Technical analysis

An analysis is conducted on the stock market to gain insight into its emergent behaviour with the intention of accurate prediction for designing a portfolio that has maximized return with the lowest possible risk. There are two main kinds of analyses. Fundamental and technical. The fundamental analysis focuses on the data from the company itself. Those can be financial records, public relations, company assets etc.

Technical analysis, on the other hand, focuses solely on the historical development of the company's stock. The model environment for technical analysis works with an assumption that the market price of the stock is a function of the supply and demand, and that historical trends affect the future development of the stock. It uses the past and the present data to construct prediction models for future development.

The development of the traded stock is abstracted into a stochastic system, as the underlying complexity resulting in the price of the stock cannot be comprehended by an individual, therefore, antecedent trading methods relied mainly on the fundamental

analysis of the company producing stock. However, we now have means of capturing and processing vastly larger quantities of data and nowadays, data is the fundamental driver of contemporary trading. With the context, information can be mined from data and produce insight into stock's future development.

2.3.1 Big Data

If we want to reduce the uncertainty in the universe of discourse, we need more information, and consequently, more data to work with. Taking the competition into account, quicker data processing results in faster decision-making thus gaining and sustaining the competitive edge in the market. With more voluminous, and velocious data, it is inevitable for its nature to become even more variegated and imposing any kind of structure presents an exhaustive task. (Laney, 2001) (D'Angelo, 2018)

The relationship between the increase in the V's (Laney, 2001) of data and the quality of information mined is not necessarily a linear one. Knowledge is not power, it is only potential power, in this case, therefore only good employment of tools, techniques and models of data analytics can be the harbinger of a successful data-driven investing venture. (Márquez, et al., 2017)

2.3.2 Generic Reference Architecture

(Márquez, et al., 2017) split big data processing into five phases:

1. Data Generation
2. Data Ingestion
3. Data Storage
4. Data Analytics
5. Presentation

Data is generated by various applications, systems, sensors etc. Then it is ingested/collected by a big data system – this phase usually involves cleaning and pre-processing data, introducing some order into it. After that, the data is stored by the system. It can then be read, mutated, or potentially deleted as of a part of the data analytics process. The data is then reported as a part of the presentation. Those phases are not strictly sequential, Figure 2.1 visualizes the architecture, along with the directions of the flow of the data.

2.3.3 Information Visualisation

Extracting patterns from the data (data mining) is one of the desired objectives in data science-related work. The prevalent final customer of data mining is still a human being, therefore there is an everlasting need to tailor the reports to the comprehension of a human mind. An individual gets most of the information visually, and there are shortcomings the mind suffers from, the inability to scale the volume of comprehension being the ever-lasting challenge to overcome.

The human mind is, however, equipped to deal with highly non-homogeneous and noisy data, and it is intuitive, therefore it does not require any understanding of complex mathematics, statistics, computational algorithms etc. It is believed that exploring data this way is more reliable and leads to better results than by utilizing machines.

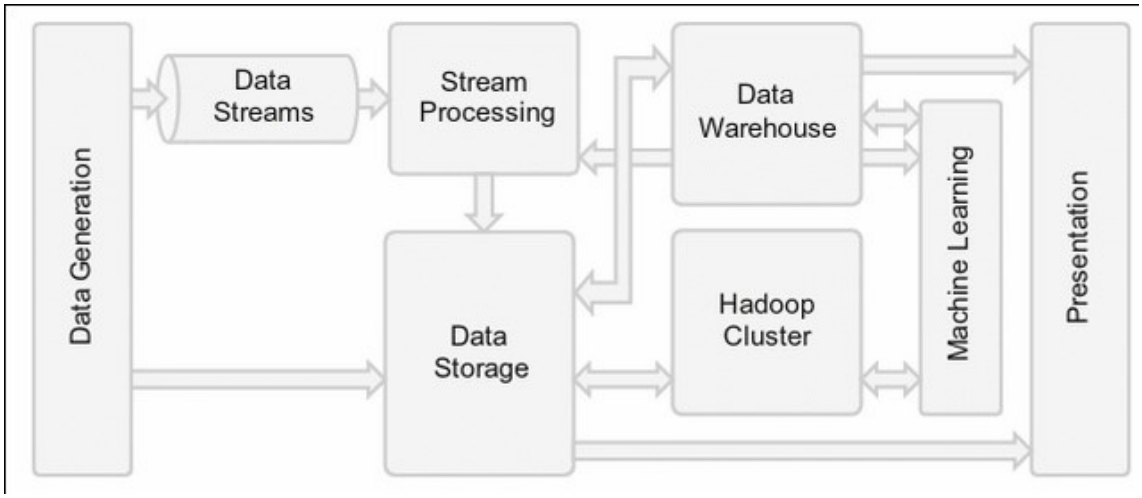


Figure 2.1 Generic Reference Architecture

(Keim, 2002) introduces the three-step process, that is believed to be followed in each iteration of visual data exploration. *Overview (i)* – interesting patterns are identified, then *zoom and filter (ii)* techniques are applied. Lastly, the dataset is expanded, *details-on-demand (iii)* are provided, making the process ready for potential next iteration.

2.3.4 Visualization Techniques

There is an abundance of techniques to visualize data. Just one dataset can me infinitely visualized, and there is no panacea visualization, there is no right or

wrongdoing. Visualizations are created for human individuals, their perception and cognition abilities need to be taken into account, along with the purpose of the visualization itself.

2.3.5 Data Dimensionality

Data dimensionality refers to the number of attributes in a dataset. The problem grows non-linearly in complexity with each additional attribute. Hence, to convey information through visualization comes with an inherent information trade-off. With high dimensional data, there may be a need for multiple visualizations to communicate all information.

2.3.6 Visualizing the Stock Data

As mentioned in Section 2.2, the stock data are temporal. The basic visualization of time-series data is the line chart. Popular and convenient visualization of the stock price data is a candlestick chart (see Figure 2.2 for an example). Each candle in the candlestick chart represents one price interval and shows. The bottom knot represents the lowest price point, the top knot represents the highest price point.

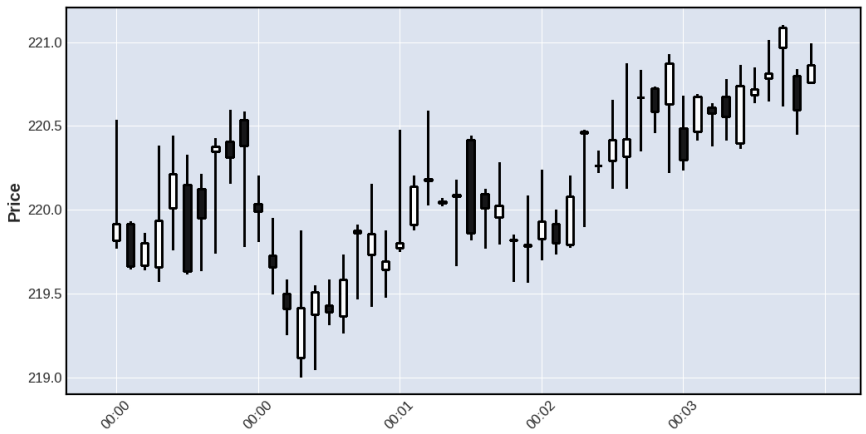


Figure 2.2 Example Candlestick Chart

There are two types of candles. The first type represents an increase in price (bullish candle, light colour in Figure 2.2), the second type represents a decrease in price (bearish candle, dark colour in Figure 2.2). In a bullish candle, the bottom of the wide part represents the open price, and the top the close price, in a bearish candle, the order is vice versa.

2.3.7 Visualization on the web

When it comes to data visualization on the web, web browsers utilize Scalable Vector Graphics (SVG) to render. SVG is fit for simple graphics, such as drawing graphs. Vectors are described in terms of mark-up (XML is used).

The most used tool for manipulating SVG is JavaScript framework D3.js.

Web browsers go with two main techniques to represent the data visually. They can either use SVG or bitmap graphics called Canvas. Table 2.1 provides a brief comparison.

SVG	CANVAS
Vector	Bitmap
More flexible	Less flexible
Heavier on resources	Lighter on resources

Table 2.1 SVG vs. Canvas

Handling visualization on the web brings all the benefits, but also all the downsides of web applications. The applications are contained within the browser, they are portable and multiplatform.

Implementing applications in the web browser comes with restrictions because developers are forced to work with the abstractions of the browser environment (through JavaScript).

2.4 Methodology

The whole StockInSight system required a set of methodologies to tackle individual problems.

The prototype was developed in vitro, i.e. in an environment that was isolated from the real world.

2.4.1 Data Generation

A mock stock exchange was implemented to generate individual stocks. Each stock of company A was abstracted into randomly generated time-series for which the

name (symbol) of the stock served as an initial seed. Seed-based generation was chosen, should there arise the need to reproduce situations with the same results.

The stock analysis and prediction themselves were not in the scope of the project, therefore a simple data generator was implemented. The data generated with the generator exhibit fluctuations and trends over time, which was convenient for some technical indicators, however, it was not necessary for the project.

For further information on the data generation see Annex 1: Stock Simulator.

2.4.2 Data Ingestion and Transport

The requirement for the ingestion and transport was to be scalable with the increasing load of data. The methodology for data ingestion and transport entailed the adoption of a horizontally scalable distributed system to provide elastic scalability and fault tolerance.

2.4.3 Data Analytics

The analytics system was expected to deal with three types of problems: big volume, big velocity, and the two combined.

The data of a large volume was to be handled by a parallel programming model, which would enable the batch to be spread equally on the computing nodes.

To maintain the velocity, a stream processing would be executed on the incoming data, enriching it and sending it to the client.

Selected Indicators and Analysis Techniques

Few basic technical indicators were selected for the prototype.

Simple Moving Average (SMA) accompanies almost every technical analysis. SMA smoothens the time-series, eliminates seasonality, and is used to identify trend.

Formula (2.1) shows how the SMA is calculated. p_i denotes a price at time i, n is the lag in time units, t is the current time. The higher n is, the more inert the trend line becomes.

$$SMA_{(n,t)} = \frac{\sum_{i=t-n}^{t} p_i}{n} \tag{2.1}$$

Simple Moving Average treats all values equally, e.g. a value lagged n time unit has the same weight in the calculation as the most recent value.

α is the smoothing factor such that:

$$0 \leq \alpha \leq 1$$

Exponential Moving Average (EMA) for given time period n (in time units) is calculated:

$$EMA_{(t,n)} = \begin{cases} p_1, & t = 1 \\ \alpha_{(n)} \cdot p_t + \left(1 - \alpha_{(n)}\right) \cdot EMA_{(t-1)}, & t > 1 \end{cases} \quad (2.2)$$

The closer α approaches 1, the more weight more recent changes in time-series have, thus the lesser the smoothing effect is. Generally used formula ((2.3)) for α was used for the prototype.

$$\alpha_{(n)} = \frac{2}{n+1} \quad (2.3)$$

Moving Average Convergence/Divergence (MACD) is a technical indicator, a difference between two EMAs, used to indicate changes in the trend of a time-series.

$$MACD_{(t)} = EMA_{(t,n\_fast)} - EMA_{(t,n\_slow)} \quad (2.4)$$

$$n\_fast < n\_slow$$

For the sake of the prototype, values 12, and 26 were chosen for n_fast, and n_slow respectively.

2.4.4 Data Reporting and Visualization

The time-series is output on the client, this could be shown in the form of a table, but high-frequency data is not likely to be comprehended by an individual, hence a visualization is desired. Stock data is best visualized by a candlestick chart (see Figure 2.2).

The client is likely to communicate with the rest of the system throughout the entire life cycle, hence it makes sense to establish bidirectional channel for communication.

2.5 Tools and Software Selection

All tools (third-party development libraries, existing engines etc.) selected for development and implementation were non-commercial software with an open-source licence. This ensures the software is accessible to anyone for future adoption or contributions.

2.6 Project Implementation

With a one-person project, classical agile methodologies could not be applied strictly.

Core functionality was identified from the requirements and a prototype developed around that functionality. There was an idea of a potential customer, however, no actual customer dictated the requirements. Given the time and depth scope, the user-stories driven development was desirable, making the project more agile.

The system was going to be developed with an emphasis put on test-driven development. Test-driven development starts with unit-tests which fail. The code is then written for the test to succeed. This would keep the development goal-oriented, as the unit-tests focus on the value-added to the user.

2.7 Potential Project Issues

Some issues might affect not only carrying out the project, but also, affect the hypothetical full version of the system.

The theoretical deliverable system under the investigation of this project would be a complex work requiring an effort of a group of variously specialized individuals. There is risk tied to using third-party software. The licencing must be always subjected to investigation. The initial requirement of using only open-source software lowers the risk to a certain degree. The system operates on data, and therefore is subject to any legislation concerning data in the country of operation.

3 Analysis of the Current State

A literature review should overview the options an individual has when pursuing private investing endeavours. Fields related to the solution the project delivers are inspected in depth.

The literature for StockInSight project was surveyed by querying academic databases Summon from University of Huddersfield and Google Scholar. Articles were drawn from various sources, including IEEE journals. Software overviews are based on documentations and third-party reviews.

3.1 Commercial Data Analytics Solutions

There are many commercial solutions available on the market that can be used for analysing the data from the market. All offer general functionality that can also be applied to financial analysis. Few most used products are outlined in the following subsections.

3.1.1 Microsoft Excel

Microsoft Excel is a multi-purpose spreadsheet program developed by Microsoft Corporation since 1987. Excel offers a quick and easy way to manipulate small amounts of data and get a fast result within a friendly user environment. Compared to other software, it is the easiest to control.

The program gets data either from user input or from files in the appropriate format, thus it is a file-based system. Once the data load gets larger, Excel is no longer sufficient. Excel is also multi-purpose, which means it is very general. For analysis data, it does not offer as much as software designed for analysis.

Excel runs on Windows operating systems or in Azure cloud as a part of the Microsoft Office 365 office suite. At the time of writing Excel licence costs $139.99 for a one-time licence or $6.99 per month as a part of the Office 365 suite licence.

3.1.2 IBM SPSS

SPSS is a statistical analysis software owned and developed by IBM. It offers a graphical user environment for conducting statistical analysis. It is written in Java and equipped with tools for time series analysis and data visualization. The program is usually

purchased by companies and universities, for an individual, the price starts at $99.99 per month.

3.1.3 Tableau

Tableau is a BI platform developed by Tableau Software LLC. Tableau aims to provide a platform to allow everyone to understand their data. Tableau comes with a framework for exploratory data analysis and visualization. The software allows the user to query results through natural language and is equipped to communicate with various file systems and databases. Tableau can be hosted on-premises or accessed online.

Tableau is proprietary and starts from $70 per month billed annually.

3.2 Data Analysis Tools

Apart from full-blown solutions, there are also software tools. Those require higher technical expertise, but usually are free to use and have their source open to the public.

3.2.1 R

R is a programming language designed for statistical computing and statistical graphics. It was created in 1993. Nowadays, R is considered a standard in data science communities.

R is a specialized language, therefore its implementation into larger systems is harder than with other general-purpose languages.

3.2.2 Python

Python is a high-level, multi-platform general-purpose programming language created in 1991. Python's design is aimed toward code readability. It is dynamically typed and supports multiple programming paradigms – object-oriented, functional, imperative, and procedural.

Python finished as overall most popular language according to Developer Survey held on the Stack Overflow website. https://insights.stackoverflow.com/survey/2019

Python is still under active development as an open-source project. The currently supported version of the language is Python 3. The previous major versions are not backwards compatible.

The language has strong library management in the form of package managers and online repositories. Python is becoming a preferred choice for data scientists, for it offers libraries for each stage of the data science process.

Widely known data science packages are:

- pandas – a fast data manipulation library
- numpy – a library for matrix computing
- matplotlib – a library for data visualizations
- seaborn – a library based on matplotlib for statistical data visualization
- scipy – a library for scientific computing

Despite all its strengths, Python is not built for distributed computing and is used only as a wrapper when working with data analytics software (e.g.: PySpark to write Apache Spark Applications in Python)

3.3 Data Storage

There are two main types of Databases to choose from. Relational Databases, and NoSQL Databases.

3.3.1 Relational Databases

Relational databases build upon relational model (Codd, 2002), and are fit for structured data, that obey a predefined schema. Relational Database Management Systems (RDBMS) are, however not fit for big data, as they only offer vertical scaling (scaling by improving hardware)

3.3.2 NoSQL Databases

NoSQL databases offer flexibility by allowing to store data without a predefined schema. Data does not have to be structured to be stored. The database can be scaled horizontally (adding more machines) for higher loads, making it feasible for big data solutions.

There are many technologies in the NoSQL world and many types of database management systems built with those technologies. NoSQL is very heterogeneous in

terms of the software and its use. Due to relaxed schema, the SQL usually cannot be adopted by the database system.

3.4 Distributed Computing and Analytics

The data needs to be processed to extract information from it. Processing time matters and it is desirable for the time not to increase with increased data load. Distributed computing systems and models are being adopted to achieve this.

MapReduce is a programming paradigm that generalizes distributed computing across nodes.

MapReduce is fit for large batch processes that does not require many iterations, as the data needs to be moved around across nodes each time a MapReduce job is executed, and the results are being written into the filesystem.

Apache Hadoop is an open-source analytics engine providing distributed processing and storage of data. Hadoop uses MapReduce for parallel processing. Hadoop works with its own Hadoop Distributed File System (HDFS) which offer high fault tolerance and is designed to be deployed on low-cost, possibly heterogeneous hardware. HDFS is known to be able to easily handle files that are terabytes large.

Modern Hadoop works with cloud-based files systems that are sometimes being referred to as data lakes. All 3 largest cloud providers offer some version of a cloud-based file system. AWS offers S3, Google Cloud Platform offers Google Cloud Storage, Microsoft Azure offers Azure Blob. Data lakes remove the need to transport acquired data from a file to HDFS, thereby improving the data analytics process by removing one step of moving data from it.

Apache Spark is a cluster computing framework created in 2009 at AMPLab at University of California in Berkeley, it was later donated to the Apache Foundation and now is distributed under their open-source license.

Apache Spark introduces in-memory computation through Resilient Distributed Dataset (RDD). Nowadays, more advanced structures such as DataFrame (with the functionality similar to Python Pandas DataFrame) and Dataset are available. Apache Spark is known to be up to 100x faster than Hadoop, mainly thanks to in-memory computation.

Spark comes with many utility libraries, the important examples being Spark SQL enabling ANSI SQL syntax for querying data, Spark Streaming for streaming data, and MLlib for machine learning.

Spark provides rich API in languages Python, Scala, Java, and R. Table 3.1 shows a short overview.

Language	Property
Python	Simple, fast prototyping
Scala	Functional, good scalability
Java	Used in enterprise
R	Mainly used for machine learning

Table 3.1 Spark API languages

3.5 Web Service, Backend, and Frontend

There are three most recognized frameworks for building user interfaces in the web browser.

Angular.js is an enterprise-grade JavaScript framework developed by Google for building Single Page Applications (SPA). It was designed to help the developer separate the view logic from the application logic. It is now distributed under the MIT licence.

React.js is a JavaScript library/framework for creating user interfaces on the web. It was originally developed by Facebook, but now is in the domain of open source with MIT licence with Facebook being the maintainer and largest contributor.

Vue.js (read as view) is a progressive open-source JavaScript library for creating web user interfaces. It was created by Evan You in 2014 with an aim to provide a framework similar to Angular.js, but more lightweight. It is now adopted by a vibrant community and has over 3 000 commits on GitHub.

Nowadays, the differences between the functionality of the frameworks are very subtle, and the most significant are in the syntax.

3.5.1 JavaScript – MEAN and MERN

MER/AN (MongoDB Express, React/Angular, Node.js) refers to a popular stack used for web application development. It is perceived strength lies in the relative homogeneity. Apart from MongoDB and the HTML/CSS design part, the tools are JavaScript oriented. The framework uses JavaScript Object Notation (JSON) for communication within the system, offering interoperability along with convenience. The deployment of a MER/AN application is fast and well-documented. The popularity of the stack results in large community support.

3.5.2 Django Web Framework

Django is a web-framework which aims to ship with all facets a web application should have for fast, easy, and successful deployment. The domain language of Django is Python 3. Django supports a variety of database back-ends, Django is open-source and has large support from the Django and Python community. Since Django is written in Python, it natively supports all Python libraries, including the libraries for data science.

3.6 Web graphics for visualization – D3.js and Canvas

(Bostock, et al., 2011) introduced an abstraction for manipulating Scalable Vector Graphics (SVG) on the web client through a JavaScript library. The framework's graphics are manipulated based on the data passed through JavaScript. The entire project is open-source and distributed under the BSD licence.

3.7 Deployment

Infrastructure as a service model delegates infrastructure management to an online service. This approach has numerous advantages and is in a vast amount of cases cost-effective solution for software deployment. (Amies, et al., 2012)

The four largest providers of this service are Google (Google Cloud), Amazon (Amazon Web Services), Microsoft (Microsoft Azure), and Digital Ocean.

3.7.1 Docker

Docker is a virtualization tool mostly associated with DevOps methodology. Docker builds a virtual environment (container) on top of a Linux kernel that acts as a minimal virtual operating system.

In contrast to a fully-fledged virtual machine, Docker does not virtualize the kernel and uses the host's machine kernel. The container then ships only with the packages that are necessary to run the application.

Containers run everywhere where docker is installed, making all containerized software platform agnostic, and therefore portable. This eliminates the difference between production and testing environment and the risks tied to said difference.

3.7.2 Nginx

Nginx is an open-source web server developed as a response to the Apache Web Server. Nginx aims for a fast distribution of static content and balancing loads on services in production. The web server supports HTTP (HTTPS), SMTP, POP3, IMAP and SSL protocols. The source code of Nginx is distributed under the BSD licence.

3.7.3 Kubernetes

Originally developed by Google, Kubernetes is now free software under the Apache Licence for container orchestration in the cloud. Kubernetes is used for deploying the system as a microservices architecture. It aims to enable easy scalability, zero downtime deployment, and management (orchestration) of container networking layer.

4 Design and Implementation of StockInSight Application

The scope of this report covers the design of the prototype in depth. Some implicit facets of a production-ready system might have been omitted.

StockInSight was a big data-oriented project, the design needed to account for high volume and the high velocity of the data. The variety of data was low in the case of stocks, any stock price development of a given tick and name (symbol) could be captured in 7 data points:

- Timestamp
- Open
- High
- Low
- Close
- Volume

The system was split into several subsystems so they could be distributed across the network. This section presents a description and purpose of each part of the system along with the selected software tools. The underlying architecture was reused from the General Architecture Model (Márquez, et al., 2017). An overview of the hypothetical system can be seen in Figure 4.1.

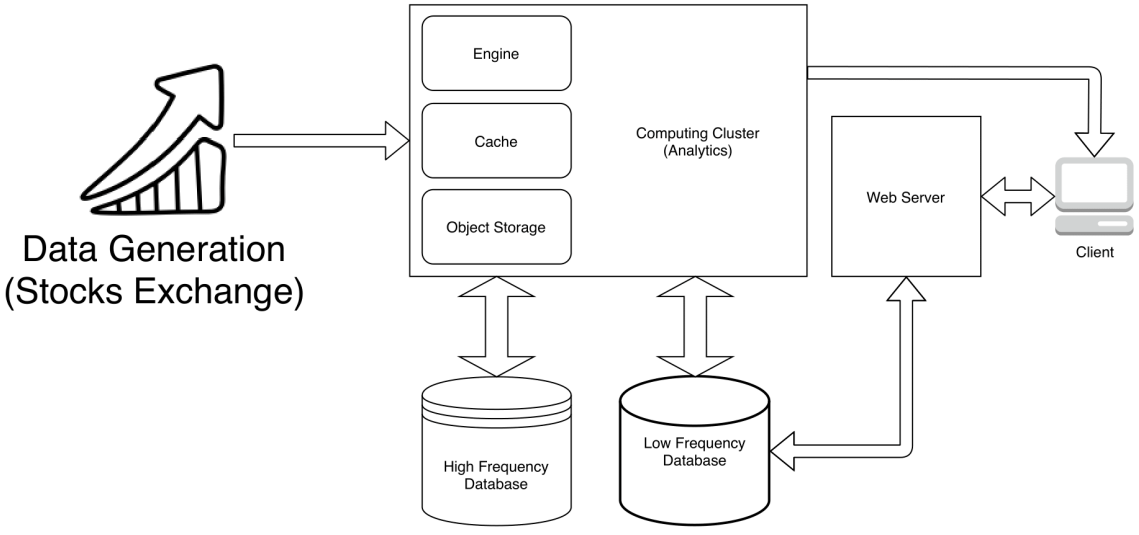


Figure 4.1 Initial System Design Diagram

4.1 Data generation

It was vital to provide a testing environment which was a sufficient simulation of the real one. After a survey into the available API options for stock data, it was decided to implement a simulator that would emit fictional data. This meant the data load could be adjusted on-demand. The disadvantage of this approach was that the data would not resemble the real behaviour of the stock market. This was, however, not necessary, because there was no need for real data for testing.

The generator was implemented in Python 3 (see Annex 1: Stock Simulator), as it made it possible to develop a prototype quickly and easily. The built-in Random module and the Numpy package provided pseudo-random number generation for desired distribution.

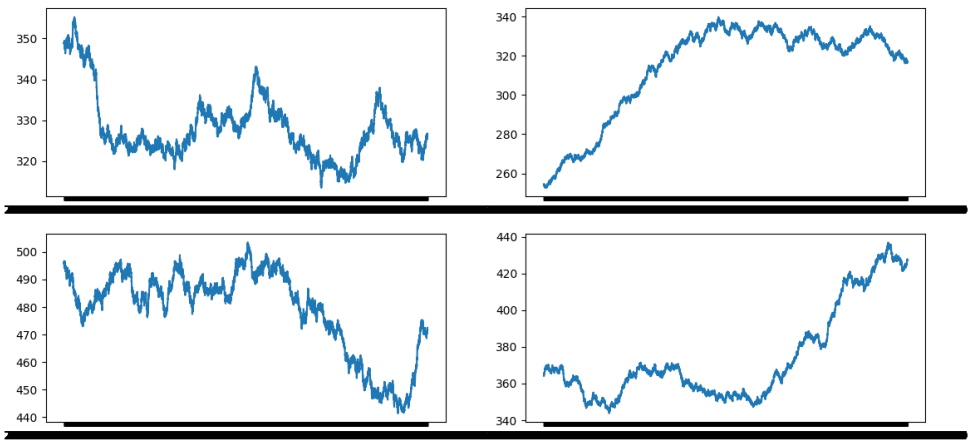


Figure 4.2 Generator time-series examples

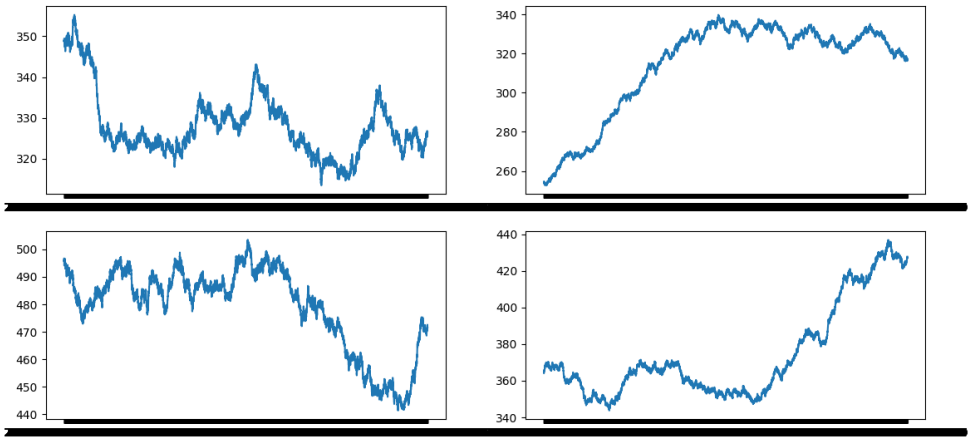


Figure 4.2 shows time-series of 10 000 Close prices for four random seeds.

4.2 Computing/Analytics Cluster – Apache Spark

Spark's performance outweighs its competitors by two orders of magnitude (see Section 3.4 in Chapter 3).

Spark Cluster is a master/slave architecture. Spark Driver (master) orchestrates Spark Executors (slaves), which execute the user's code as a series of tasks grouped into parallel jobs.

The heart of every Spark application is the Resilient Distributed Dataset (RDD). RDD is a data structure that serves as an abstraction from the distributed system.

4.3 Messaging – Apache Kafka

It was important for the system to be agnostic of the network environment it was going to live in. In future development, it is impossible to foresee the direction of the flow of the data.

Implementation of a messaging system was a feasible solution to the problem.

Kafka uses a binary TCP-based protocol and offers a commit log that can keep messages in it for weeks before they are deleted. Kafka is a distributed system and messages can be replicated on multiple nodes. These make Kafka highly reliable and scalable messaging system selected for StockInSight.

Apache Kafka works on a publish/subscribe basis. Kafka stores messages in a data structure called a topic.

Kafka topic is the base unit in the Kafka's architecture. It represents a stream of related data and is uniquely identified by topic name.

Each topic is split into partitions that themselves hold the messages. The messages within one partition are ordered, they are assigned an offset – an auto-incrementing id. There is no ordering between partitions.

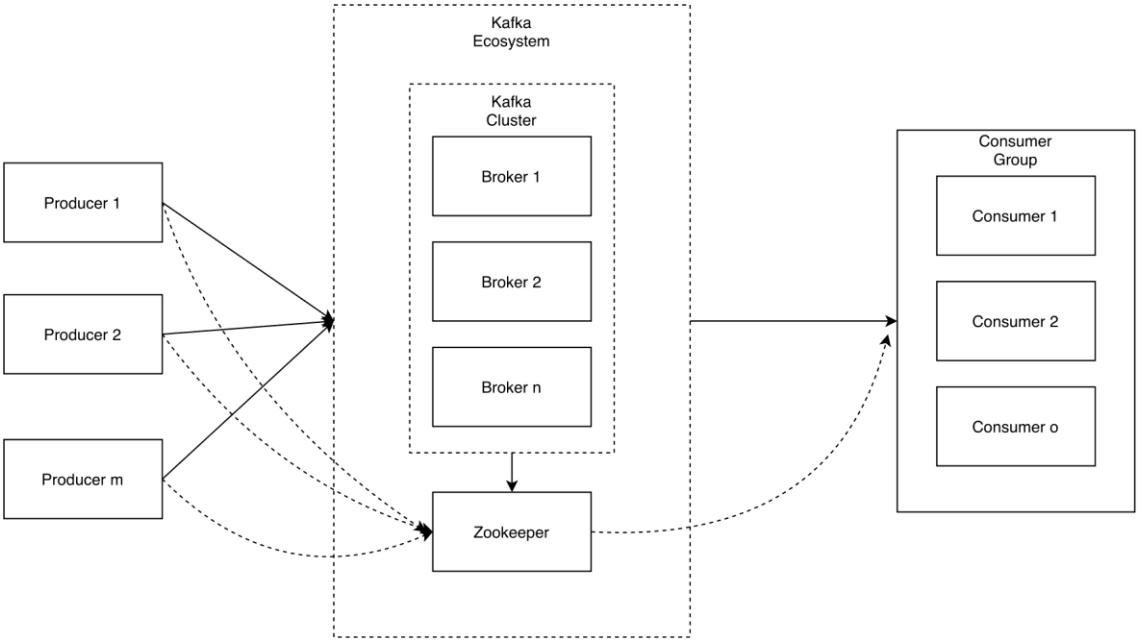
Once the data is written into a topic, it cannot be changed. The data is held in Kafka for a time period that can be specified in the configuration.

The topics are held by Kafka brokers. Brokers are servers (nodes) that make the Kafka cluster. Each broker holds at most one partition of a given topic, i.e. any topic can only have at most as many partitions as there are brokers in the cluster ($0 \leq$ Topic replicas $\leq$ No. of Kafka brokers). The no. of replicas the topic has is called a replication factor of a topic. Replicas ensure fault tolerance of the system, if a node goes down, the data is preserved on replicas. The more replicas, the less likely the system is to lose data.

Only one broker is able to serve and receive data on a given partition and other brokers synchronize partition replicas. Such a broker is called a leader for a partition. In the case of the leader crashing, another broker containing an in-sync replica is elected as a new acting leader. When the former leader broker goes up again, it tries to regain the leader status on the partition.

Producers write data to topics; thus they are adopted by data-generating applications. Producers are responsible for picking brokers and partitions to write to. Producers automatically recover in case of broker failures.

Figure 4.3 Kafka Cluster Diagram



There are options for producers to require acknowledgements for data writes. Producer can either require no acknowledgement, an acknowledgement from the leader partition, or for all replicas. Increasing acknowledgement level requirement decreases the likelihood of data loss but also decreases the speed of data transfer.

Producers can send a key with their messages to dictate the partition the data will be written to. If there is no key present, the producer sends the data round-robin across the partitions i.e. distributes the messages equally between brokers.

Consumers are used for subscribing and reading from topics. They are responsible for selecting which broker to read from and for selecting the partitions and offsets to read.

A consumer reads from one or more topic partitions. Consumers belong to consumer-groups, each consumer within a group reads from exclusive partitions, a partition serves only one consumer from a given group at the time. This means that a group subscribed to a topic with n partitions with $n + m$ consumers will result in m inactive consumers that could take over if some active consumers were to stop working, allowing the applications subscribing to the system through consumers to be resilient through consumer fault tolerance.

Consumers can choose to commit an offset into Kafka. When a consumer stops, it is then able to read the data from the offset it left of based on last committed offset for that consumer.

Each server in Kafka cluster is a bootstrap server – each time a client connects to a server in a Kafka cluster, it gains access to the whole cluster.

Zookeeper is a service responsible for Kafka broker management. Zookeeper keeps a list of brokers, takes part in performing partition leader elections and notifies nodes in the cluster in case of any changes, e.g. when a broker crashes, a new topic is created etc.

Kafka cluster was deployed in Docker as a part of the prototype, for the docker-compose YAML file see Annex 2: Docker-compose for scalable deployment of Kafka.

4.4 Web Server Backend

The web server is an intermediary that communicates with clients.

The webserver is in immediate contact with the client, Node.js was chosen from the reviewed candidates along with the Express library which offers web application framework.

4.5 Client

The Client was assumed to run on a machine with low computing power and low memory as a browser-based application. That being the case, there were limitations to the data load it could operate on and what protocols could be used for communication.

There were solutions available to utilize Kafka consumers in the web browser, however, they were not well established, therefore they could produce unexpected behaviour. WebSockets was a viable option for data streaming, all contemporary browsers support them. Other types of asynchronous communication were going to be handled through HTTP.

The interface needed to be smooth and easily accessible to an average user. One session would require many re-renders, sending a pre-processed HTML/CSS was not a viable option. It was determined to be better to delegate the rendering to the browser itself through JavaScript and have all the features in one page. The single-page application (SPA) design is nothing new on the market. There are three major open-source JavaScript frameworks for developing SPAs. They differ very little in terms of functionality, React.js was selected because of the largest community of the three (Based on Stack Overflow survey from 2019: https://insights.stackoverflow.com/survey/2019).

React.js core functionality is in Model View Controller (MVC) design pattern. The controller takes input (user click event, API request) and converts it to commands, that can interact with the model. The model manages the data and rules of the application – React abstracts the model into React components. The view is what gets output into the browser Document Object Model (DOM – what the user will actually see.

React keeps a representation of the DOM in the browser's memory in the form of a JavaScript object. Unlike browser DOM, this virtual DOM does not need to render anything and therefore is substantially faster. React implements a diffing algorithm that compares the virtual DOM, against the browser DOM and creates a list of the minimum possible changes to the browser DOM. Once the list is complete, the changes are committed. React achieves this without triggering any unnecessary browser reflow.

UI Design was not an essential component. MIT licensed Material Dashboard was used in the prototype as it offers a basic generally accepted layout for good user experience.

4.6 Data Storage

The data for the business logic is the time-series for different stocks. The data points are of primitive types, however, there is no strict schema that needs to be designed for the system. Typical relational database management systems do not offer horizontal scaling, which is essential for a big data solution. Apache Cassandra was determined to be ideal for high-frequency data storage. Apache Cassandra is a distributed database management system that focuses on high speed in reads and writes. Cassandra shares some terminology with relational databases. The basic data structure in Cassandra is a table. Tables are made up of columns that store attributes. Tables in Cassandra have primary keys that uniquely identify rows. Since Cassandra is distributed, primary keys alone are not enough to find records, therefore, two additional keys are defined. Partition key identifies which node in a cluster to store a row in. Clustering key identifies the order in which the rows are stored.

There are no join operations, thus tables are denormalized, which results in data redundancy. Select queries cannot be ordered using "ORDER BY," the ordering must be done during storage.

This means that data modelling is query-driven, not entity-driven.

5 Conclusion

Apache Spark and Apache Cassandra are a well-established part of the big data ecosystem and their performance is sufficient by design because of the horizontal scalability.

The data from the generator was being sent through the Kafka cluster and the Node service to the React client where a candlestick chart was being re-rendered on the dom. Figure 5.1 shows a mock-up of the user interface (UI) with the SVG Chart component.

The initial chart was rendered after 100 data points were available on the client, and additional data points were appended. Two trials were done on one stock with frequencies 50, and 5 milliseconds.

Figures Figure 5.2 and Figure 5.3 shows rendering durations for new data point commits (5, and 50 ms, respectively). Figure 5.4 Histograms shows histograms of the frequency of various durations of commits.

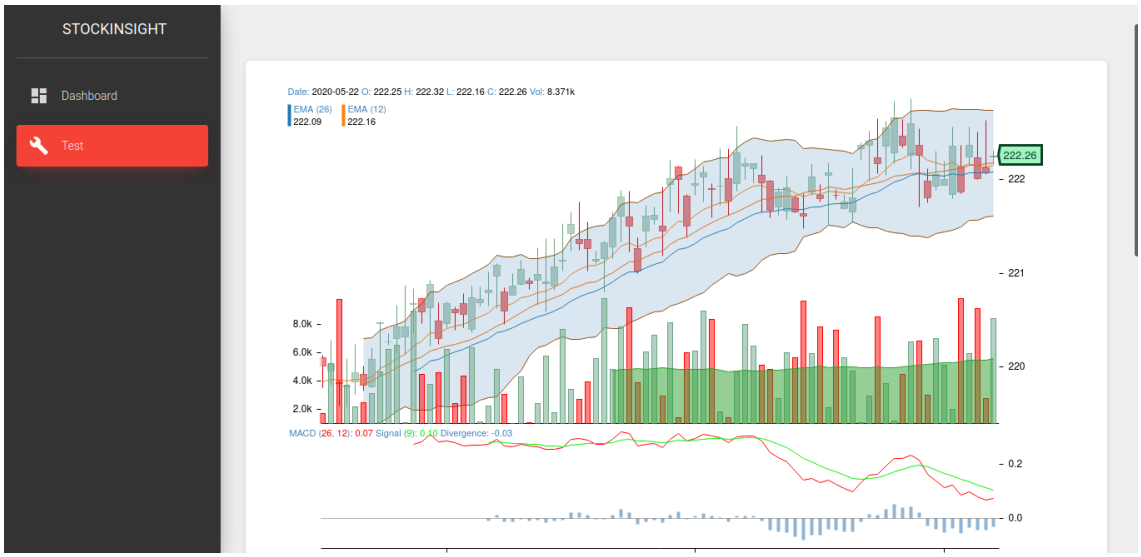


Figure 5.1 Real-Time Chart View

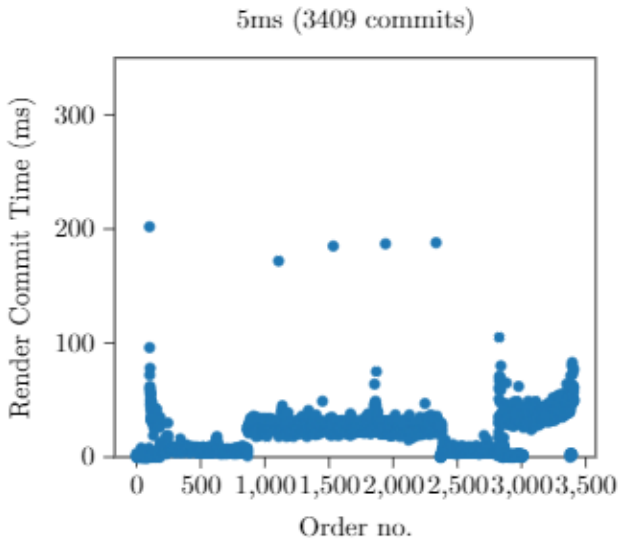


Figure 5.2 Scatterplot of Render Times, Time Interval 5 ms

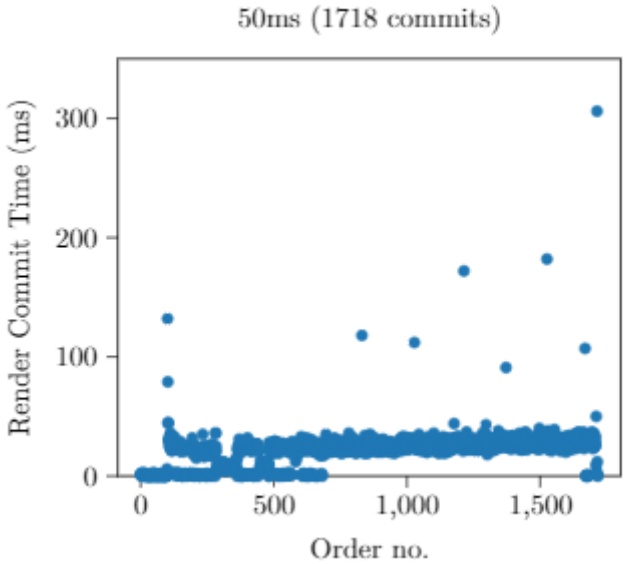


Figure 5.3 Scatterplot of Render Times, Time Interval 50 ms

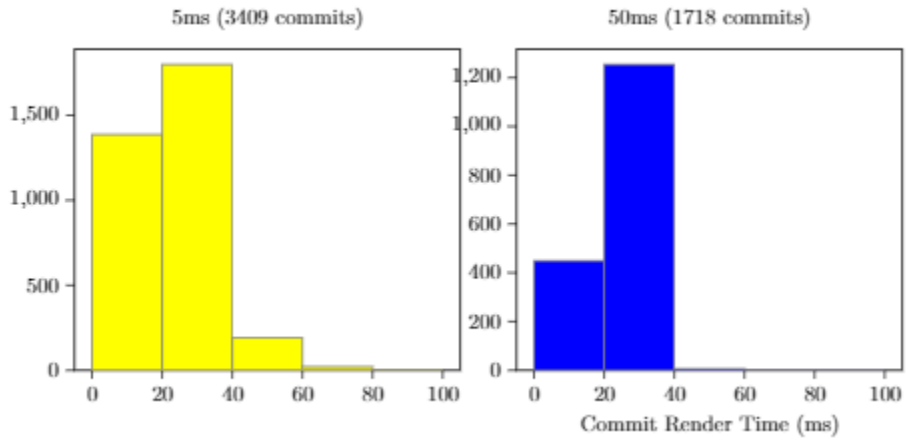


Figure 5.4 Histograms

5.1 Discussion

The figures in Chapter 5 show that the frequency of renders longer than 40 ms increases with the velocity of the data. This showed on the UI as a drop in perceived performance. The chart stopped rendering and then proceeded with a lag.

This manifested in a drop in perceived performance of the UI. In a too fast setting, the chart would stop rendering for a brief period and then proceed lagged with respect to real-time messages.

Browser resource consumption points to a need for future work. The visualization can handle reasonable high-frequency data, however, with increased frequency, a limit becomes apparent. There is also a question of the usefulness of such high-frequency visualization, mainly concerning the level of visual comprehension.

A web browser offers the comfort of multi-platformity without the requirement of installation of additional software. The high-level features, including frameworks for vector rendering, proved to work only to some degree of data velocity. Higher throughput could be solved with a technique involving lower-level tweaking, lest the web-based application approach is dropped. High-frequency monitoring is still viable for monitoring interests, such as price and indicators, instead of the time-series as a whole.

The current prototype still can be useful for long-term trading, However, no foundation for justifying usefulness for higher frequency trading can be laid.

Future work concerning the analytics should touch the topic of algorithmic trading. This means an increased focus on the backend, mainly the analytics engine

(Apache Spark) and the actual algorithms, with visualization serving as a secondary overview tool, thus focused on post-hoc reporting, rather than the real-time. Implementation of algorithmic trading would entail integrating a low-latency mechanism for placing orders through a selected stock exchange broker.

The data in this project was in the form of historical data, which is used only for technical analysis. Full insight into the stock market cannot be drawn only from the technical analysis, the fundamental analysis should also be employed, and future projects should explore the ways the two combined can be implemented algorithmically.

Sticking with private investors as the target user base and moving to the question of algorithmic trading, it will be important not only to provide the user with the analytics tools but also to tailor the environment to be user-friendly. The topic of this question ranges from user experience to educating the user about the topic of investing.

The final question concerns the viability of a potential final product for a private investor. There are many services in the system whose functionalities are interdependent. If such a system were to be available for individuals that are less tech-savvy, its deployment would require a great deal of automation, possibly requiring many additional man-hours.

5.2 Final Summary

Stock trading is evolving and changing fast. The market is largely dominated by institutional investors and private investors usually resolve to invest through institutional investors as intermediaries, sacrificing returns for a lower risk. StockInSight project explored potential for a community solution that would allow private investors to gain insight into stocks through a self-hosted web-based application.

Visualization of data in the web browser showed to suffer from shortcomings, mainly concerning the render time, and perceived performance issues when reaching higher data throughput. From these results it seems to be inadvisable to try to visualize high frequency stock data in real time, rather orient the project towards post-hoc visualizations.

The concept of the StockInSight system was derived from General Reference Architecture. Based on the requirements criteria, the following tools were recommended

for implementation. Apache Spark was selected for the data analytics, Apache Cassandra for high frequency data storage and Apache Kafka for messaging.

A Prototype was developed to test high frequency visualization on the web. A Python data generation script was implemented. Using the React/D3.js charting library React-stockcharts, it was experimentally shown that the client is prone to creating a bottleneck effect in data with interval of less than 50 ms.

The recommendations given for future work included investigating and evaluating other environments for visualizing data that would offer the same convenience as web-based application with greater performance. Next, designing open-source frameworks for the usage of stock analysis for less tech-savvy users.

Bibliography

Technical literature

AMIES, Alex, SLUIMAN, Harm and TONG, Qiang G. 2012. *Developing and hosting applications on the cloud.* s.l. : IBM Corporation International Technical Support Organization, 2012. 9780133066869.

CODD, Edgar F. 2002. A relational model of data for large shared data banks. [book auth.] Manfred Broy and Ernst Denert. *Software Pioneers: Contributions to Software Engineering.* s.l. : Springer, 2002.

HENNING, Charles N., PIGOTI, William and SCOTT, Robert Haney. 1978. *Financial Markets And The Economy.* Englewood Cliffs, New Jersey : Prentice-Hall, Inc., 1978.

KOTU, Vijay and DESHPANDE, Balachandre. 2018. *Data science: concepts and practice.* Cambridge, MA : Morgan Kaufmann Publishers, an imprint of Elsevier, 2018. 9780128147627.

LANEY, Doug. 2001. 3D Data Management: Controlling Data Volume, Velocity, and Variety. *Application Delivery Strategies.* 2001.

MÁRQUEZ, García, FAUSTO, P. and LEV, Benjamin. 2017. *Big Data Management.* Cham : Springer International Publishing, 2017. 9783319454986.

Journal and conference articles

KEIM, D. A. 2002. Information visualization and visual data mining. *IEEE Transactions on Visualization and Computer Graphics.* 2002, Vol. 8, 1.

BOSTOCK, Michael, OGIEVETSKY, Vadim and HEER, Jeffrey. 2011. D3 Data-Driven Documents. *IEEE Transactions on Visualization and Computer Graphics.* 2011, Vol. 17, 12.

Electronic documents and websites

Amazon Web Services. 2020. *Amazon Web Services.* [Online] 2020. https://aws.amazon.com/.

D'ANGELO, Matt. 2018. What Is Big Data? Business News Daily. [Online] April 2, 2018. [Cited: March 25, 2020.] https://www.businessnewsdaily.com/4522-big-data.html.

DigitalOcean, LLC. 2020. *Digital Ocean – The developer Cloud.* [Online] DigitalOcean, LLC, 2020. https://www.digitalocean.com/.

Django Software Foundation. 2020. *Django.* [Online] 2020. https://www.djangoproject.com/.

Docker Inc. 2020. *Docker.* [Online] 2020. https://docker.com/.

F5, Inc. 2020. *Nginx.* [Online] 2020. https://nginx.com/.

Facebook Inc. 2020. *React – A JavaScript library for building user interfaces.* [Online] 2020. https://reactjs.org/.

Google. 2020. *Angular.* [Online] 2020. https://angular.io/.

Google Cloud. 2020. Google Computing Services. *Google Cloud.* [Online] 2020. https://cloud.google.com/.

IBM. *SPSS Software | IBM.* [Online] https://www.ibm.com/analytics/spss-statistics-software.

Microsoft. 2020. *Microsoft Azure.* [Online] Microsoft Corporation, 2020. https://azure.microsoft.com/.

Tableau Software, LLC, a Salesforce Company. 2020. *[Tableau] Business Intelligence and Analytics Software.* [Online] 2020. https://www.tableau.com/.

The Apache Software Foundation. 2020. *Apache Hadoop.* [Online] 2020. https://hadoop.apache.org/.

The OpenJS Foundation. 2020. *Electron | Build cross-platform desktop apps with JavaScript, HTML, and CSS.* [Online] The OpenJS Foundation, 2020. https://www.electronjs.org/.

The R Foundation. 2020. *R: The R Project for Statistical Computing.* [Online] 2020. https://www.r-project.org/.

YOU, Evan. 2020. *Vue.js*. [Online] https://vuejs.org/.

Acronyms

DOM	Document Object Model.
EMA	Exponential Moving Average.
MACD	Moving Average Convergence/Divergence.
MVC	Model-View-Controller.
RDD	Resilient Distributed Dataset.
SMA	Simple Moving Average.
SVG	Scalable Vector Graphics.
UI	User Interface.

Glossary

bearish	signaling aversion in the market and possible downwards trend falling prices
bullish	signaling confidence in the market and possible increasing trend raising prices
DevOps	Methodology combining software development and operations
horizontal scaling	Scaling the system by adding more machines to it
reflow	Process performed by the browser. Recalculation of the positions, colours, and geometries of the elements on the page
tick	Measured interval of stock price change. 5-day tick – one candle represents 5 days
vertical scaling	Scaling the system by upgrading the hardware of the machines in the system

List of Figures

List of Tables

Declaration of Utilization of Results from the Diploma Thesis

Herewith I declare that

- I am informed that Act No. 121/2000 Coll. – the Copyright Act, in particular, Section 35 – Utilisation of the Work as a Part of Civil and Religious Ceremonies, as a Part of School Performances and the Utilisation of a School Word – and Section 60 – School Work, fully applies to my diploma (bachelor) thesis;
- I take account of the VSB – Technical University of Ostrava (hereinafter as VSB-TUO) having the right to utilize the diploma (bachelor) thesis (under Section 35(3)) unprofitably and for own use.
- I agree that the diploma (bachelor) thesis shall be archived in the electronic form in VSB-TUO's Central Library and one copy shall be kept by the supervisor of the diploma (bachelor) thesis. I agree that the bibliographic information about the diploma (bachelor) thesis shall be published in VSB-TUO's information system.
- It was agreed that, in case of VSB-TUO's interest, I shall enter into a license agreement with VSB-TUO, granting the authorization to utilize the work in the scope of Section 12(4) of Copyright Act.
- It was agreed that I may utilize my work, the diploma (bachelor) thesis or provide a license to utilize it only with the consent of VSB-TUO, which is entitled, in such a case, to claim an adequate contribution from me to cover the cost expended by VSB-TUO for producing the work (up to its real amount).

Ostrava 20. 7. 2020

Vít Chrubasík, BSc (Hons)

List of Annexes

Annex 1: Stock Simulator Snippet

```
# file generate_time_series.py
from random import Random
from datetime import datetime, timedelta
import collections
import hashlib
from lib.stock_exchange.psychology import Psychology
import numpy as np
import sys
import os
from kafka import KafkaProducer
from kafka.errors import KafkaError
import json
import time

CandleStick = collections.namedtuple(
    'CandleStick', ['symbol', 'date', 'open', 'low', 'high', 'close'])


class Stock:
    STOCK_PRICE_RANGE = (30, 500)

    @staticmethod
    def generate_candlestick(obj):
        """returns either starting candle based on seed, or next candle based o
n previous one"""
        def get_sigma(mu):
            return mu / 1000  # fixed deviation
        threshold = obj._apply_psychology()
        candle_roll = obj._random.uniform(0, 1)  # roll for bear of bull
        candlestick = None
        is_bull = True if candle_roll <= threshold else False
        if obj.order == 0:  # no previous datapoint
            mu = obj._random.uniform(*Stock.STOCK_PRICE_RANGE)
            sigma = get_sigma(mu)
            date = obj.start_date
        else:
            if is_bull:
                mu = obj._random.uniform(np.mean(
                    [obj._current_candlestick.open, obj._current_candlestick.cl
ose]), obj._current_candlestick.high)
            else:
                mu = obj._random.uniform(np.mean(
                    [obj._current_candlestick.open, obj._current_candlestick.cl
ose]), obj._current_candlestick.low)
            sigma = get_sigma(mu)
            obj.start_date += timedelta(seconds=obj.tick)
            date = obj.start_date
        obj.order += 1
        data = [obj._random.gauss(mu, sigma) for n in range(4)]
        data.sort()

        if is_bull:  # Bull
            candlestick = CandleStick(
                obj.name, date, data[1], data[0], data[3], data[2])
        else:  # Bear
            candlestick = CandleStick(
                obj.name, date, data[2], data[0], data[3], data[1])
        obj._current_candlestick = candlestick
        return candlestick
```

```
57.
58.     def __init__(self, name: str, start_date: datetime = datetime.now().replace
    (microsecond=0), tick=5):
59.         super().__init__()
60.         self.tick = tick
61.         self.name = name
62.         self.start_date = start_date
63.         self.data = []
64.         self._seed = int(hashlib.sha1(name.encode('utf-
    8')).hexdigest(), 16) % (
65.             10 ** 8)  # for starting prices
66.         self._random = Random(x=self._seed)
67.
68.         # set Stock based on seed
69.         self._psychology = Psychology(self._random)
70.         self.order = 0  # keep track how many candlesticks has been generated
71.         self._current_candlestick = Stock.generate_candlestick(self)
72.
73.     def _apply_psychology(self):
74.         """access coefficient and update psychology"""
75.         return self._psychology.tick()
76.
77.     def _get_psychology(self):
78.         """check the psychology without ticking"""
79.         return self._psychology.__str__()
80.
81.     def next_candlestick(self):
82.         return Stock.generate_candlestick(self)
83.
84.     def produce_to_kafka(self, producer: KafkaProducer, topic: str = 'stocks-
    topic', iterations=None):
85.         """Produce messages through kafka producer. Must be a JSON producer!"""

86.         if iterations:
87.             for n in range(iterations):
88.                 c = self.next_candlestick()
89.                 timestamp = datetime.timestamp(datetime.now())
90.                 producer.send(topic, key=c.date.isoformat().encode('utf-
    8'), value={
91.                     'Symbol': c.symbol,
92.                     'Date': c.date.isoformat(),
93.                     'DateUnix': datetime.timestamp(c.date),
94.                     'Open': c.open,
95.                     'Low': c.low,
96.                     'High': c.high,
97.                     'Close': c.close,
98.                     'Timestamp': timestamp
99.                 })
100.                 time.sleep(self.tick)
101.         else:
102.             print('Producing indefinitely.')
103.             while True:
104.                 try:
105.                     c = self.next_candlestick()
106.                     timestamp = datetime.timestamp(datetime.now())
107.                     producer.send(topic, key=c.date.isoformat().encode('utf-
    8'), value={
108.                         'Symbol': c.symbol,
109.                         'Date': c.date.isoformat(),
110.                         'DateUnix': datetime.timestamp(c.date),
111.                         'Open': c.open,
112.                         'Low': c.low,
113.                         'High': c.high,
114.                         'Close': c.close,
115.                         'Timestamp': timestamp
116.                     })
```

```
                    print(timestamp, ' Produced a message. ', c)
                    time.sleep(self.tick)
                except KeyboardInterrupt:
                    print('Stopping sending to kafka.')
                    break
        producer.flush()
```

```
# file psychology.py
from datetime import timedelta
from random import Random


class Psychology:
    tick_range = (1000, 6000)  # determines lengths of trends

    def __init__(self, random: Random):
        super().__init__()
        self._ticks = random.randint(*Psychology.tick_range)
        self._bull_coefficient = random.uniform(
            0.45, 0.55)  # determines slopes of trends
        # self._bull_coefficient = 0.4
        self.random = random

    def tick(self):
        """Returns a coefficient and updates ticks"""
        coefficient = self._bull_coefficient
        if self._ticks == 0:
            self.__init__(self.random)
        tick = self._ticks
        self._ticks -= 1
        return coefficient

    def __str__(self):
        s = f'TICKS LEFT {self._ticks}: '
        if self._bull_coefficient > 0.5:
            return s + f'{self._bull_coefficient}\t[BULL]'
        elif self._bull_coefficient < 0.5:
            return s + f'{self._bull_coefficient}\t[BEAR]'
        else:
            return s + f'{self._bull_coefficient}\t[NEITHER]'
```

Annex 2: Docker-compose for scalable deployment of Kafka

```
version: "2.1"
  services:
    zookeeper:
      image: wurstmeister/zookeeper
      ports:
        - "2181:2181"
    kafka:
      image: wurstmeister/kafka
      ports:
        - "9092-9094:9092"
      environment:
        KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181
        HOSTNAME_COMMAND: "echo $$HOSTNAME"
        PORT_COMMAND: "docker port $$HOSTNAME 9092/tcp | cut -d: -f2"
        KAFKA_LISTENERS: "INTERNAL://_{HOSTNAME_COMMAND}:9090,\
          EXTERNAL://:9092"
        KAFKA_ADVERTISED_LISTENERS: "INTERNAL://_{HOSTNAME_COMMAND}:9090,\
          EXTERNAL://popntb:_{PORT_COMMAND}"

        KAFKA_LISTENER_SECURITY_PROTOCOL_MAP: "INTERNAL:PLAINTEXT,\
          EXTERNAL:PLAINTEXT"
        KAFKA_INTER_BROKER_LISTENER_NAME: INTERNAL
        KAFKA_CREATE_TOPICS: "stocks-topic:1:3,enriched-stocks-topic:1:3,unit-
test-topic:1:1"
      volumes:
        - /var/run/docker.sock:/var/run/docker.sock
```