University of Pisa
School of Engineering

Large Scale and Multi-Structured Databases

StockSim: Stock Portfolio Simulator

Supervisor
Prof. Pietro Ducange

Students
Marco Pinna
Rambod Rahmani
Yuri Mazzuoli

January 22, 2021

Contents

Part I

Documentation

Chapter 1

Introduction

StockSim is a Java application which, as main feature, allows users to simulate stock market portfolios. The StockSim application is composed by two main programs:

- **StockSim Server**: supposed to be running 24/7 to ensure historical data is always up-to-date;
- **StockSim Client**: can be launched in either `admin` or `user` mode and provides different functionalities based on the running mode.

The StockSim Server is not thought to be distributed to end users and is intended to be running on a Server machine, whereas the StockSim Client can be used by both administrators and normal users. The choice was made to provide the same program to both administrators and normal users with two different running modes. Administrators can add new ticker symbols, new administrator accounts, delete both administrator and normal user accounts. Normal users have access to stocks and ETFs historical data, day by day, starting from 2010. They can search for and visualize Stocks, create their own stock portfolios, run simulations and visualize the resulting statistics.

All the programs are terminal based but the StockSim Client, running in `user` mode, can display charts resulting from the different operations performed on stocks.

Before continuing with what follows, the following terms should be clarified:

- the **stock market** is any exchange that allows people to buy and sell stocks and companies to issue stocks; a stock represents the company's equity, and shares are pieces of the company;
- a collection of investments owned by an investor makes up their **portfolio**; you can have as few as one stock in a portfolio, but you can also own an infinite amount of stocks or other securities;
- a **stock symbol** is a one- to four-character alphabetic root symbol that represents a publicly traded company on a stock exchange; Apple's stock symbol is AAPL, while Walmart's is WMT;

- the NYSE and Nasdaq are open from Monday through Friday 9:30 A.M. to 4:00 P.M. (eastern time);
- the NYSE and Nasdaq close at 4 P.M., with after-hours trading continuing until 8 P.M.; the close simply refers to the time at which a stock exchange closes to trading;
- trading stocks after normal market hours through an electronic market, typically between 4:05 P.M. and 8:00 P.M., is **after-hours trading**;
- the **high** is the highest price at which a stock traded during a period;
- the **low** is the lowest price of the period;
- **open** means the price at which a stock started trading when the opening bell rang; it can be the same as where the stock closed the night before, but not always; sometimes events such as company earnings reports that happen in after-hours trading can alter a stock's price overnight;
- **close** refers to the price of an individual stock when the stock exchange closed shop for the day; it represents the last buy-sell order executed between two traders; in many cases, this occurs in the final seconds of the trading day;
- the **adjusted closing price** amends a stock's closing price to reflect that stock's value after accounting for any corporate actions; a stock's price is typically affected by some corporate actions, such as stock splits, dividends, and rights offerings; adjustments allow investors to obtain an accurate record of the stock's performance;
- **volume** is the total number of shares traded in a security over a period; every time buyers and sellers exchange shares, the amount gets added to the period's total volume;
- an **Open-high-low-close chart** is a type of bar chart that shows open, high, low, and closing prices for each period. OHLC charts are useful since they show the four major data points over a period, with the closing price being considered the most important by many traders; the chart type is useful because it can show increasing or decreasing momentum. When the open and close are far apart it shows strong momentum, and when the open and close are close together it shows indecision or weak momentum. The high and low show the full price range of the period, useful in assessing volatility.

Chapter 2

Actors and requirements

The main actors and the functional requirements can be defined using the simple application description provided in the introductory chapter. Non-functional requirements, described in detail in the second section of this chapter, are the characteristics that ensure satisfactory interaction with users and real world utility of the product.

2.1 Actors

Based on the application design, four main actors can be identified:

- A **Guest user** is someone who is not yet registered (does not have a valid account to be able to use the application); this actor does not own private credentials for the login; in order to exploit the application main functionalities, he must register a new user account; therefore, the registration is the only action allowed for a Guest;
- A **Registered user** is someone who signed up on the application; this actor owns private credentials (username and password) for the login; he can login as a user into the client application and utilize its main features to search for stocks, get historical information about them, compose portfolios and simulate them.
- An **Admin user** is someone who is registered on the application with administration credentials; this actor can login as an admin using the StockSim client running in `admin` mode to perform maintenance operations; these operations include, among others, adding a new stock to the database, adding new admin credentials, removing normal and admin users;
- The **Data Updater** is a thread running on StockSim Server; this thread is supposed to be always running, and it is in charge of updating the database historical data with the new information coming from the stock market daily trading sessions; it is also in charge of finding and fixing (or at least report) data integrity issues.

Please refer to Chapter 3 for the full use case diagram.

2.2 Requirements

2.2.1 Functional requirements

- The application is available to be used only by registered users;
- The application provides access to stocks and ETFs historical data, day by day, starting from 2010;

- A guest user, upon launching the application, should be able to sign-up;
- A registered user should be able to and sign-in;
- A registered user should be able to search for a stock and view related details;
- A registered user should be able to view charts of the historical data of a stock;
- A registered user should be able to create and delete portfolios;
- A registered user should be able to run simulations on a portfolio;
- A registered user should be able to visualize statistics about a simulation and view related charts.

- Only admins can add new stocks to the database;
- Only admins can create new admin accounts;
- Only admins can delete admin accounts;
- Only admins can delete user accounts;

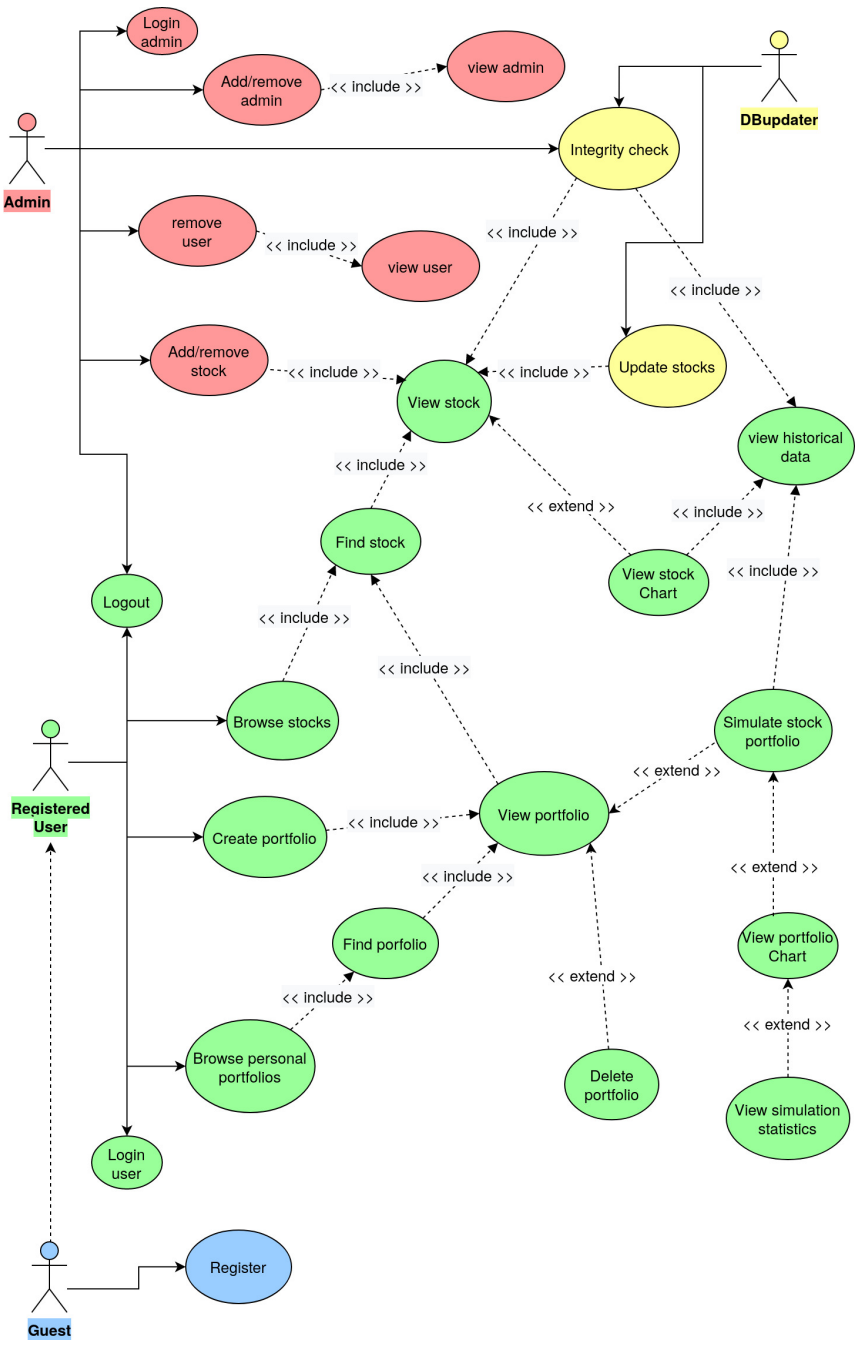
2.2.2 Non-functional requirements

- The terminal based menu should allow the user to accomplish the desired tasks with the minimum amount of commands;
- The application will store information in non-relation databases (MongoDB and Apache Cassandra);
- Stocks historical data and information should be always available and updated;
- The retrieval of stocks and portfolios historical data and information should be fast;
- Charting retrieved data should be fast;
- User password should be stored in a secure way (i.e. hashed);

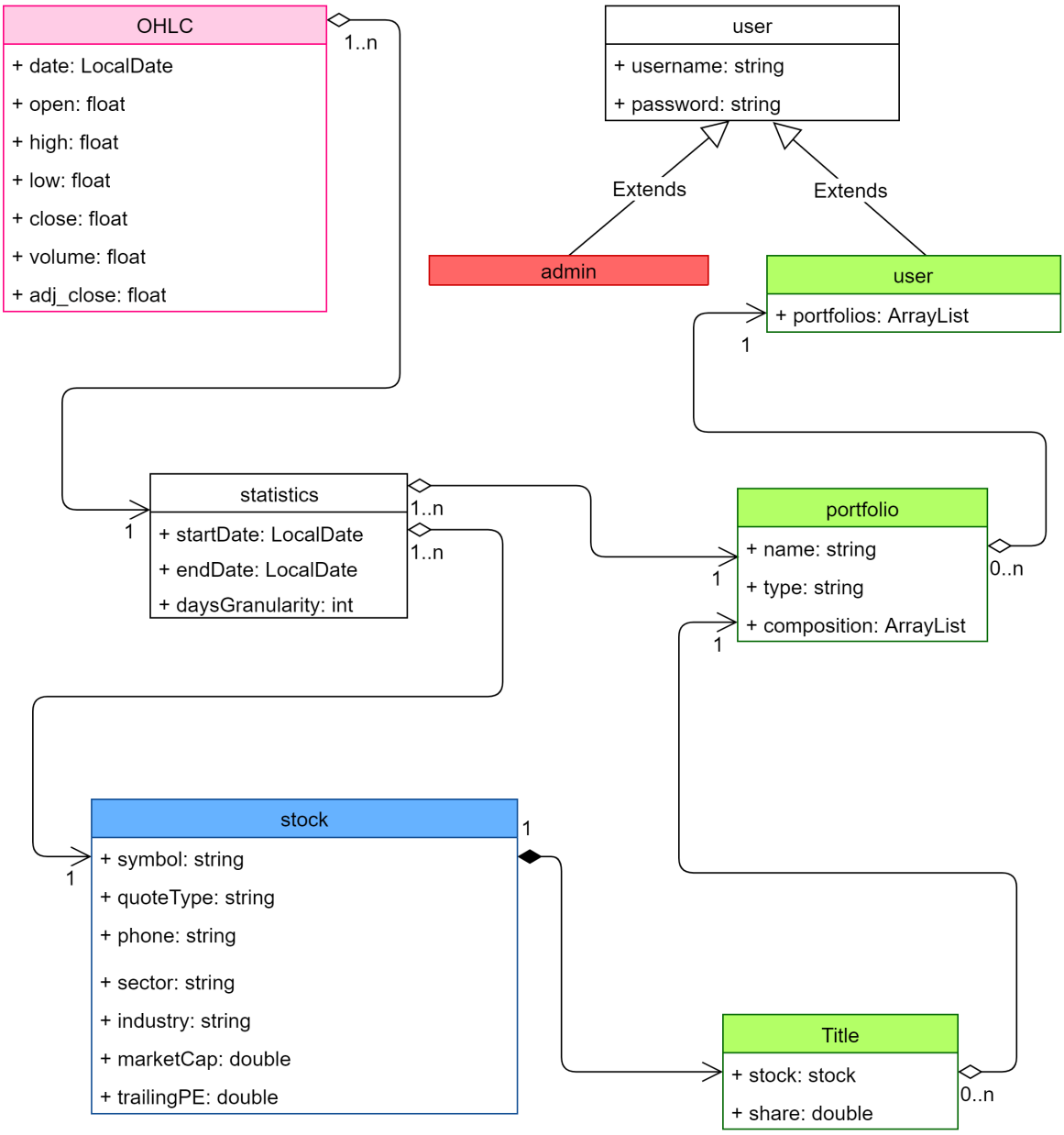
Chapter 3

UML diagrams

3.1 Use Case diagram



3.2 Class diagram



Chapter 4

Database

The database is composed by 8236 stocks from the US stock market, along with their general information and historical data; the application also needs to store users and admins login credentials, personal information and the composition and details of each user's portfolio.
We decided to use a column database (Apache Cassandra) for the storage of historical data; this is because historical data represents almost the 99% of the entire database and it is going to be growing very fast as days go by; aggregation and financial analytics on these volumes of data will perform better in a column database where data storage is designed to optimize this type of operations by column.
We decided to store any other information using a document database (MongoDB), in order to exploit the schemaless property to save memory; data frequently needed together is stored in the same document and indexes were created to speed up linking between documents;

4.1 Dataset

The initial set of data was fetched from the web, by means of Python scripts using `pandas`, `yfinance` and `JSON` as support libraries and relying upon `www.nasdaqtrader.com` and `finance.yahoo.com`.

4.1.1 NasdaqTrader

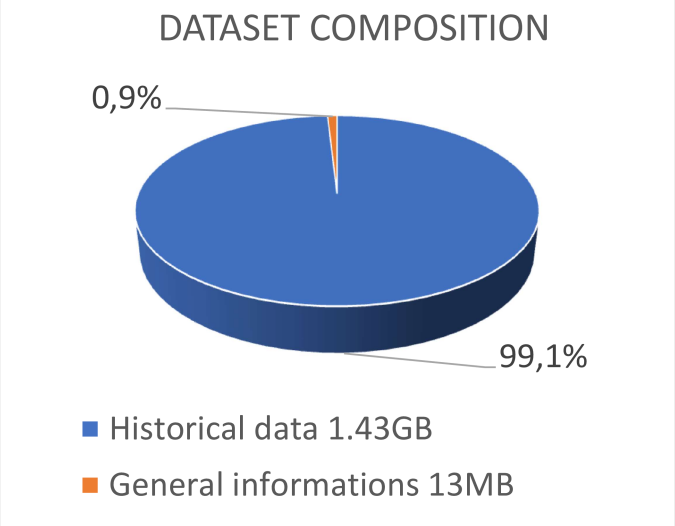
The Nasdaq Stock Market (Nasdaq) is the largest U.S. equities exchange venue by volume. [1]
We choose to take our set of stocks from the Nasdaq index, because it is very popular and includes a large number of stock, representative of different economy sectors. This will allow users to interact with big and famous companies stocks (like Google, Apple, Tesla...), but also to try smaller companies and/or minor sectors investments. Nasdaq-

[1] https://www.nasdaqtrader.com/

Trader provides us a stock symbols' list of all the stocks entered in the NASDAQ index from 1970 until now;

4.1.2 Yahoo! Finance

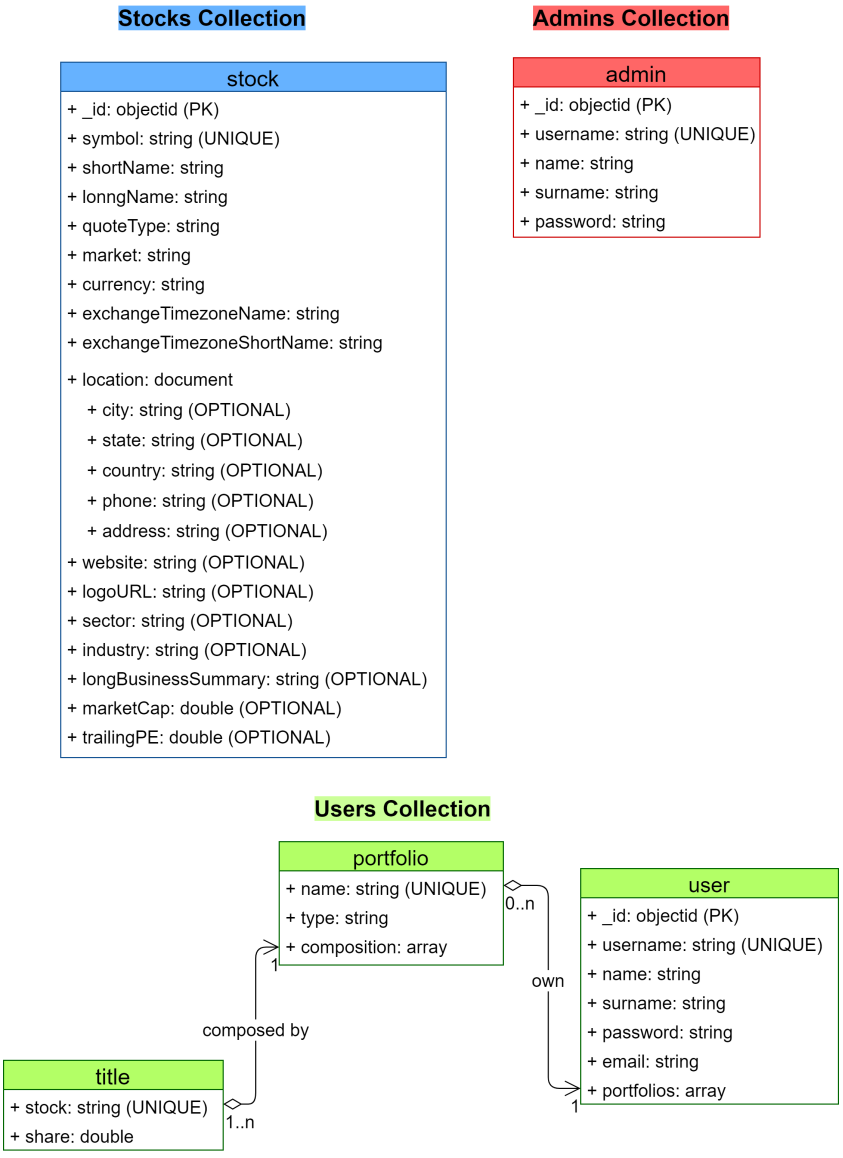
"Yahoo Finance provides free stock quotes, up-to-date news, portfolio management resources, international market data, social interaction and mortgage rates that help you manage your financial life." [2]
Yahoo Finance is a service, been part of the Yahoo network, that provides several of information about stocks and companies; they are frequently updated, reliable and well organized.
We decided to use this service to retrieve the starting dataset of stocks; we extract only the fields that we need, and parse into a JSON file. In this way it is possible to rebuild from scratch this dataset into MongoDB with few commands (including mongoimport). With Yahoo Finance it is also possible to retrieve historical data of market values for every stocks. Using this service it has been possible to build a dataset of all the market values of each stocks coming from NasdaqTrader; values are collected daily, and we decided to take all the values from 2010 to 2020; this dataset (around 1.43 GB) has been parsed to CVS files and than imported into a Cassandra Cluster. Thanks to the Yahoo Finance service, it's possible to update every day the database with the last session results. It's also possible to add a new stock to the dataset, coming from every market exchange of every country.



[2] https://finance.yahoo.com/

4.2 MongoDB

"MongoDB is a general purpose, document-based, distributed database build for modern application developers and for the cloud era." Taken from www.mongodb.com.
MongoDB is a very famous document database with a great support for cloud operations, which will improve the availability of our application. It also supports several analytic functions and the creation of custom indexes in order to speedup read operations. In order to organize data in a meaningful and memory-optimal way, we opted for this structure:



This scheme is composed by 3 collections: **stocks**, **users**, **admins**;

- The stocks collection contains one document for each stock; inside this document all the general information about the stock is stored; each stock is identified by the attribute SYMBOL. Some basic information is always present, while other may be missing for some stocks; we decided to keep these last type of information were possible, exploiting the schemaless property of the document database;
- The **users** collection contains one document for each user registered on the application; in each of these documents, login credentials are stored, along with few personal information; for every user there is also an array of documents named PORTFOLIOS: this array contains the portfolios of the user. Each portfolio has a scheme, which includes an array of TITLEs, named COMPOSITION, which describes what the portfolio consists of. This nested structure has been preferred over splitting data in different collections, because all the information of a user, including their portfolios, is frequently needed at once; on the other hand, there are not operations that involve portfolios owned by different users.
- The **admins** collection contains the admins login credentials together with few personal informations about them; we decided to create a separated collection for administrators to improve the security of the administration features: in this way is impossible to inject administration privileges through the login command.

4.2.1 Aggregations

One of the main features of our application is the possibility to choose some stocks from the market and combine them into a portfolio. When a user is looking for a stock, they want to know statistics about **industries** and **sectors**, along with classification by **level of capitalization** and **PE ratio** (*Price-Earnings Ratio*); in order to do so, we will provide these aggregation pipelines:

- the total market capitalization of each sector
- the total market capitalization of each industry
- the total market capitalization of stocks coming from the same country
- the average PE ratio of stocks working in the same sector
- the average PE ratio of stocks working in the same industry
- the average PE ratio of stocks coming from the same country
- the average PE ratio of stocks being in a specific range of market capitalization

We provide here an example of an aggregation Mongo query:

```
1   /**
2    * Aggregates data with filtering and grouping by an attribute, can compute
3    * sum, avg ecc.
4    *
5    * @param collection the collection where to perform the operation;
6    * @param filter filter to be used to find the documents.
7    * @param groupField the filed used to group the aggregation
8    * @param aggregator the aggregator function and field
9    *
10   * @return iterable object containing the result of the aggregation.
11   */
12  public AggregateIterable<Document> aggregate(final Bson filter, final String groupField, final
        BsonField aggregator, final MongoCollection<Document> collection) {
13      Bson match = Aggregates.match(filter);
14      return collection.aggregate(
15              Arrays.asList(match, Aggregates.group("$"+groupField, aggregator)));
16  }
```

```
1   MongoCollection<Document> collection1 =
2       dbManager.getCollection(
3           StocksimCollection.STOCKS.getCollectionName()
4       );
5   // aggregate examples
6   final Bson equity= eq("quoteType", "EQUITY"); //filter(s)
7   // name of the field projected, field to accumulate
8   // type of accumulation (sum, avg...)
9   final BsonField marketCapAccumulator=Accumulators.sum("totalCap","$marketCap");
10  AggregateIterable<Document> aggregateList =
11                                          // grouping attribute
12          dbManager.aggregate(equity, "sector", marketCapAccumulator, collection1);
13  for (Document document : aggregateList) {
14      System.out.println(document);
15  }
16  // avg example with nested attribute
17  final BsonField PEAccumulator=Accumulators.avg("avgPE","$trailingPE");
18  aggregateList =
19          dbManager.aggregate(equity, "location.country",
20                  PEAccumulator, collection1);
21  for (Document document : aggregateList) {
22      System.out.println(document);
23  }
```

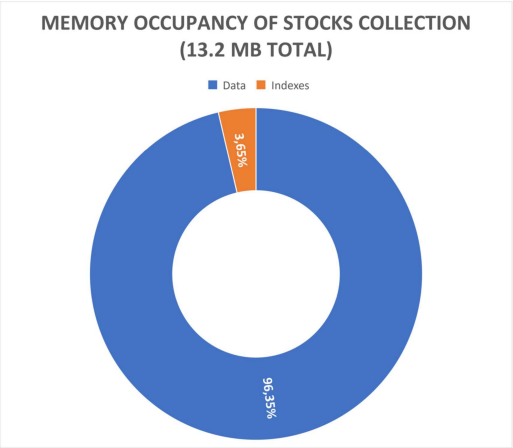
4.2.2 Indexes

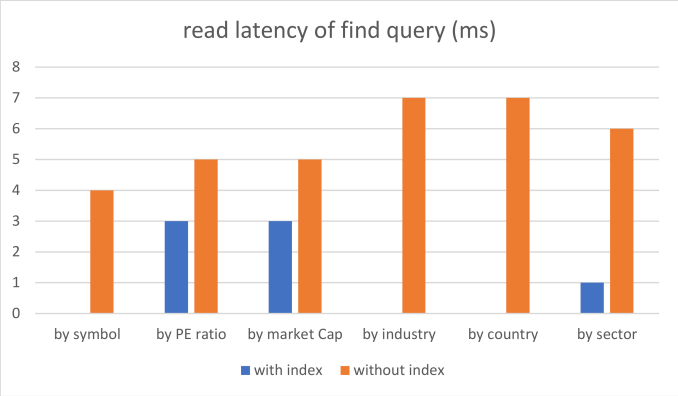
In order to speed up read operation in the document database, we decided to introduce some custom indexes:

- a REGULAR and UNIQUE index on the attribute **symbol** in the collection stocks;

- a REGULAR index on the attribute **marketCAP** in the collection stocks;
- a REGULAR index on the attribute **trailingPE** in the collection stocks;
- a REGULAR index on the attribute **sector** in the collection stocks;
- a REGULAR index on the attribute **industry** in the collection stocks;
- a REGULAR index on the attribute **country** in the collection stocks;
- a REGULAR and UNIQUE index on the attribute **username** in the collection users;

We provide some statistic that endorse our indexes choices





Analog results can be found about the username index in the users collection.

4.3 Apache Cassandra

"The Apache Cassandra database is the right choice when you need scalability and high availability without compromising performance. Linear scalability and proven fault-tolerance on commodity hardware or cloud infrastructure make it the perfect platform for mission-critical data. Cassandra's support for replicating across multiple datacenters is best-in-class, providing lower latency for your users and the peace of mind of knowing that you can survive regional outages." [3]
Apache Cassandra is a database designed for high scalability and availability; it is capable to handle a huge amount of data and manage it in a decentralized architecture across multiple nodes. It is built to be write optimized, but with the right indexes choices read latency can be improved too; tables schemas and analytic functions can be customized. This is the schema of our Cassandra database:

[3] https://cassandra.apache.org/

Ticker		
PK	PARTITION KEY CLUSTERING KEY	Symbol: string Date: LocalDate
		Open: float Close: float High: float Low: float Volume: float Asjusted_close: float

4.3.1 Aggregations

In order to provide snapshots and statistics of stocks and portfolios trends over time, we exploit the customization functionalities of Cassandra; a custom aggregation has been created, specifically to provide aggregate values for periods of time longer than one day; this allows us to obtain customizable granularity for stock market data, computed on server side; this will not over overwhelm the server, because the aggregator will execute, for each row, between a memory access and the following. This will greatly reduce the data to be transmitted from the node to the client, saving bandwidth and time.

```
1
2     /* State function to be executed for every row*/
3     CREATE OR REPLACE FUNCTION PeriodStateParam (
4
5     /* the state, containing the aggregation result till this row */
6         state map<date,frozen<map<text, float>>>,
7     /* the parameter ndays indicates the duration
8     * of the period aggregation, in days */
9         ndays int, data date,
10        open float, close float , high float, low float,
11        volume float , adj_close float)
12    CALLED ON NULL INPUT
13    RETURNS map<date,frozen<map<text, float>>>
14    LANGUAGE java AS '
15    if (data != null) {
16        int d=0;
17        Map<String, Float> statemap=null;
18
19        for(d=1; d<ndays;){
20            if((statemap=state.get(
21                    data.add(Calendar.DAY_OF_MONTH,d)
```

```
                ))!=null)
                break;
            d++;
        }
        if(d==ndays){
            statemap=new HashMap<String, Float>();
            statemap.put("open", open);
            statemap.put("close", close);
            statemap.put("high", high);
            statemap.put("low", low);
            statemap.put("volume", volume);
            statemap.put("adj_close", adj_close);
            state.put(data,statemap);
        }
        else{
                if(high>statemap.get("high"))
                    statemap.replace("high", high);
                if(low<statemap.get("low"))
                    statemap.replace("low", low);
                statemap.replace("volume",statemap.get("volume")+ volume);
                statemap.replace("open",open);
                state.replace(data, statemap);
        }
    }
    return state;'
     ;

    /* aggregate declaration
    * this aggregation generate a map data structure (JSON like):
    * the key is the end date of each period of nday days,
    * and the value is another map containing the aggregate
    * values of the period as:
    * the open of the first day
    * the close and adjusted close of the last day
    * the maximum of the highs
    * the minimum of the lows
    * the sum of the volumes
    */
    CREATE OR REPLACE AGGREGATE PeriodParam
        ( int, date,float, float,float, float,float, float )
    SFUNC PeriodStateParam
    STYPE map<date,frozen<map<text, float>>>
    INITCOND {}; /* no initial condition is necessary */

    /* example of usage, it can be used also with grouping by symbol */
    select PeriodParam(
        20, date, open, close, high, low, volume, adj_close)
        as Period from tickers where
        date<'2020-12-1' and date>'2020-6-10'
        and symbol='TSLA';
```

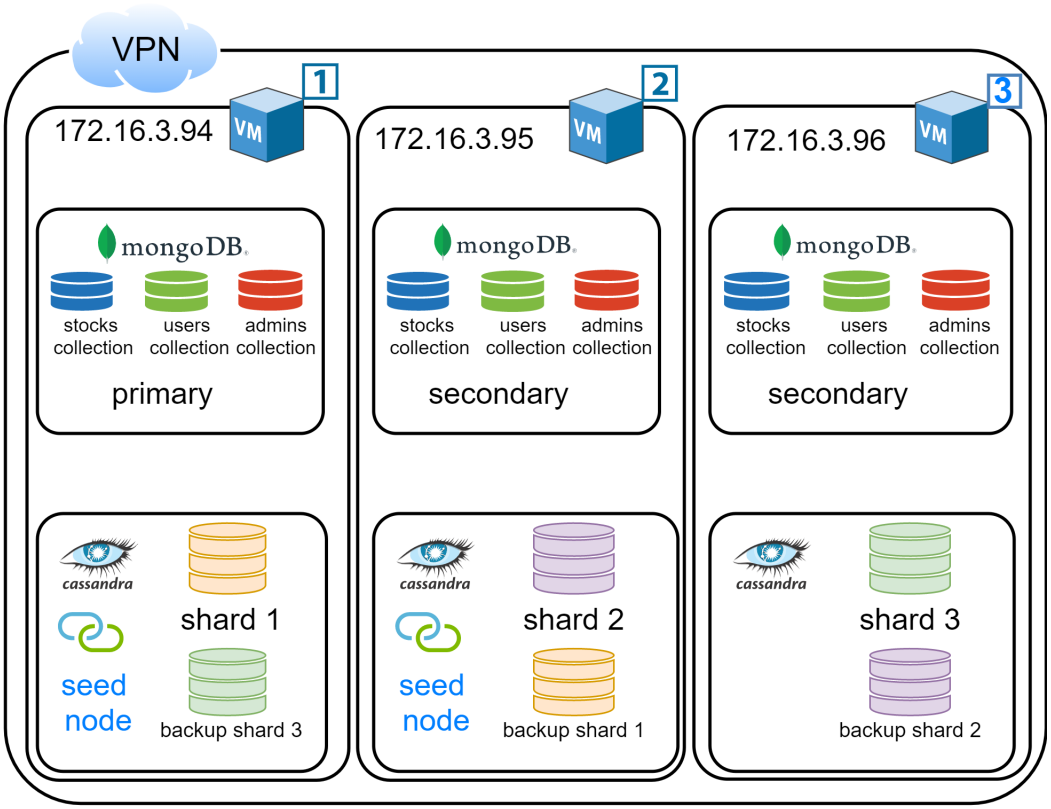
4.3.2 Indexes

- The PARTITION KEY index is part of the PRIMARY KEY and it is used to shard the dataset across the nodes. This index is build on the string symbol, unique for each stock;
- the CLUSTERING KEY i is also part of the PRIMARY KEY, and it is used to maintain rows in chronological order. This index is built on the attribute date;

4.4 Sharding and Replicas

The MongoDB cluster and the Cassandra cluster are deployed on 3 virtual machines provided by the University of Pisa; Our architecture is oriented to the availability of the service and built for the maximum scalability and decentralization.

- The Cassandra cluster is built among all 3 nodes, and data is sharded by ticker symbol; in this way every node stores 1/3 of the main dataset, and aggregation functions are computed on records that lie in the same node; each node also stores a backup of the data assigned to another node, giving us a replication factor of 2. There are 2 seed nodes, which are responsible for the cluster: the cluster is online as long as one of them keeps working. This is indeed a minor issue, because in any case, with only one node, the dataset would by incomplete. The decentralized behaviour of Cassandra ensures that even if all the node were to go offline, the cluster would return available as soon as one seed server went back online.
- The MongoDB cluster is also built among all 3 virtual machines, and the service is replicated; an initial primary server is elected, then another one will take its place if the former goes down. The cluster is available as long as one server is working, and incoming traffic could be balanced with the "nearest" preference on client connection; in this way the client would connect to the server with the lowest ping time.



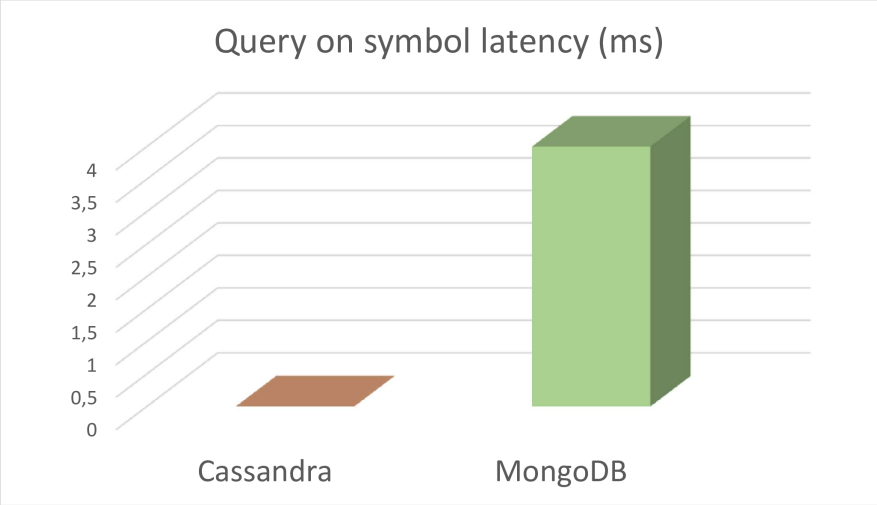
4.5 Apache Cassandra vs MongoDB

Cassandra and MongoDB are different database architectures but they share similar features. We know for sure that the usage of Cassandra can be avoided by adding a mongo Collection, which can reproduce the behaviour of the Cassandra table for historical data. We want to explain why in our opinion is better to store those data in Cassandra instead of Mongo, from a performance and flexibility perspective.

- First of all, Cassandra has a native decentralized behaviour; we can't replicate all the historical data in every server, because hardware limitation, so we are forced to shard data. Cassandra is specific designed for availability and partition tolerance, and can easy handle a node failure;
- Cassandra can also compress data with advanced algorithms, which allow a node to store a lot of backup data from other nodes, increasing the availability of the service;
- Even if Mongo can store and aggregate our dataset in a decent way, the opportunity to create custom aggregation directly in Java language make Cassandra the best

choice; this allow us to perform every aggregation function that we need, and to add new ones;

- From a performance perspective, we ran the same query, on the same dataset, with the same indexes structure, both in a Mongo collection and in our Cassandra keyspace; Cassandra always registered no latency, while mongo showed some ms of latency (4 in most cases); tests were performed on local storage, in the same machine, because remote latency performance would have been somehow distorted by network swings.



Chapter 5

Software architecture

In this section we show the general software architecture of the application.
Stocksim is a multi-module application written in Java. Its development was helped by the IntelliJ IDEA[1] IDE, together with the build automation tool Maven[2].
We opted for a multi-module approach so that logically independent sections could be kept separated and better maintained. Furthermore, we decided to follow a feature-oriented development process and the modularity of the project helped us to coding with more ease and efficiency.
The entire codebase can be found at https://github.com/rambodrahmani/stocksim .

5.1 Maven Multi-Module Structure

The application consists of three modules: Client, Server and Library.
Every module has its own .pom file, necessary for the building process and for keeping track of all the dependencies needed.
There is also a parent .pom that belongs to the 'parent' module Stocksim (which encapsulates the whole application), which contains all the dependencies and build settings that are common for all the modules.
All .pom files are available in Appendix A.
For every module an independent .jar file can be generated which allows to run that module *standalone*, without any dependency issue.

5.1.1 Client

The Client module is the main core of the application. It is that part of the application which is thought to be distributed to end users.
It has the following structure:

[1] https://www.jetbrains.com/idea/
[2] https://maven.apache.org/

```
CLIENT
└── src
    └── main
        └── java
            └── it.unipi.lsmsdb.stocksim.client
                ├── admin
                ├── app
                ├── charting
                ├── database
                └── user
```

The `app` directory contains the class with the entry point for the applications.
Since the Client program was thought to be possibly launched in two different modes (namely `admin` or `user` mode), these two functionalities were split into the homonym directories.
The `database` directory contains the classes necessary for the interaction with the databases and those necessary to store data retrieved from the formers.
The `charting` directory contains an API for the JFreeChart library, used by the Client to create charts of various types.

5.1.2 Server

The Server module takes care of keeping the data updated.
It is not supposed to be distributed: it should be always-on so that every day it can pull down from NasdaqTrader the most recent values about every stocks. The Server module has the following structure:

```
SERVER
└── src
    └── main
        └── java
            └── it.unipi.lsmsdb.stocksim.server
                ├── app
                └── database
```

The `app` directory contains the class with the entry point for the application.
The `database` directory contains the classes necessary for the interaction with the databases.

5.1.3 Library

There is also a third module, the `Library` module, which contains the database APIs both for MongoDB and Cassandra, together with the YahooFinance API and some utility functions. It has the following structure:

```
LIBRARY
└── src
    └── main
        └── java
            └── it.unipi.lsmsdb.stocksim.lib
                ├── database
                │   ├── cassandra
                │   └── mongoDB
                ├── util
                └── yfinance
```

The `database` directory contains the two APIs we wrote for the interaction with Cassandra and MongoDB.
The `yfinance` directory contains the API we used to retrieve data from Yahoo Finance.
The `util` directory contains some utilities functions, such as the ones used to set the log level both for Cassandra and Mongo, and an argument parser.

5.2 Apache Maven Assembly Plugin

The Assembly Plugin for Maven enables developers to combine project output into a single distributable archive that also contains dependencies, modules, site documentation, and other files.

Chapter 6

Conclusions

The modular architecture of the project (v. Chapter 5) allowed us to create stand-alone APIs (for both the databases and Yahoo Finance) that are not inherently tied with the StockSim application or with financial applications in general. They all can possibly be used stand-alone and could be integrated in different projects.

Many choices we made during both the designing and the development process proved to be effective. We opted for a column database architecture to store the historical data because it really lends itself to the usage we thought of. Performance analysis of the query latency (as shown in paragraph 4.5) has confirmed out hypothesis, although tests have been run only locally.
It would be interesting to repeat the measurements with a bigger database and in a controlled environment where network jittering does not skew the results too much, something that we were not able to do due to limited resources and due to the need of a VPN.

At the moment the database is limited to US market only, but it would be really simple to integrate data from other stock markets around the world.
This would not be a problem since the column database architecture is by nature suitable for geographical distribution and for dealing with huge amounts of data. Another test worth doing to evaluate the performance of the application would be to deploy several geographically distributed servers and increase the data volume by some orders of magnitude: the current database amounts to around 1.43 GB and it would be interesting to see how it behaves with tens of terabytes of data.

Chapter 7

Appendices

Appendix A

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/
             maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>it.unipi.lsmsdb.workgroup4</groupId>
    <artifactId>Stocksim</artifactId>
    <packaging>pom</packaging>
    <version>1.0</version>

    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.8.1</version>
                <configuration>
                    <source>8</source>
                    <target>8</target>
                </configuration>
            </plugin>
        </plugins>
    </build>

    <modules>
        <module>Client</module>
        <module>Server</module>
        <module>Library</module>
    </modules>

    <dependencies>
        <dependency>
```

```
34              <groupId>org.apache.maven.plugins</groupId>
35              <artifactId>maven-compiler-plugin</artifactId>
36              <version>3.8.1</version>
37          </dependency>
38
39          <!-- Dastastax Apache Cassandra driver -->
40          <dependency>
41              <groupId>com.datastax.oss</groupId>
42              <artifactId>java-driver-core</artifactId>
43              <version>4.9.0</version>
44          </dependency>
45          <dependency>
46              <groupId>com.datastax.oss</groupId>
47              <artifactId>java-driver-query-builder</artifactId>
48              <version>4.9.0</version>
49          </dependency>
50          <dependency>
51              <groupId>com.datastax.oss</groupId>
52              <artifactId>java-driver-mapper-runtime</artifactId>
53              <version>4.9.0</version>
54          </dependency>
55
56          <!-- JUnit -->
57          <dependency>
58              <groupId>junit</groupId>
59              <artifactId>junit</artifactId>
60              <version>4.13.1</version>
61              <scope>test</scope>
62          </dependency>
63
64          <!-- MongoDB -->
65          <dependency>
66              <groupId>org.mongodb</groupId>
67              <artifactId>mongodb-driver-sync</artifactId>
68              <version>4.1.1</version>
69              <scope>compile</scope>
70          </dependency>
71
72          <!-- https://mvnrepository.com/artifact/log4j/log4j -->
73          <dependency>
74              <groupId>org.slf4j</groupId>
75              <artifactId>slf4j-api</artifactId>
76              <version>1.7.30</version>
77          </dependency>
78
79          <!-- LogBack -->
80          <dependency>
81              <groupId>ch.qos.logback</groupId>
82              <artifactId>logback-classic</artifactId>
83              <version>1.2.3</version>
84          </dependency>
85
86          <!-- https://mvnrepository.com/artifact/commons-cli/commons-cli -->
```

```
87          <dependency>
88              <groupId>commons-cli</groupId>
89              <artifactId>commons-cli</artifactId>
90              <version>1.3.1</version>
91          </dependency>
92      </dependencies>
93  </project>
```

pom_files/parent_pom.xml

```
1   <?xml version="1.0" encoding="UTF-8"?>
2   <project xmlns="http://maven.apache.org/POM/4.0.0"
3            xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4            xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/
                 maven-4.0.0.xsd">
5       <parent>
6           <artifactId>Stocksim</artifactId>
7           <groupId>it.unipi.lsmsdb.workgroup4</groupId>
8           <version>1.0</version>
9       </parent>
10
11      <modelVersion>4.0.0</modelVersion>
12      <artifactId>Client</artifactId>
13      <packaging>jar</packaging>
14
15      <dependencies>
16          <dependency>
17              <groupId>it.unipi.lsmsdb.workgroup4</groupId>
18              <artifactId>Library</artifactId>
19              <version>1.0</version>
20              <scope>compile</scope>
21          </dependency>
22
23          <dependency>
24              <groupId>org.jfree</groupId>
25              <artifactId>jfreechart</artifactId>
26              <version>1.5.1</version>
27          </dependency>
28
29          <!-- https://mvnrepository.com/artifact/commons-codec/commons-codec -->
30          <dependency>
31              <groupId>commons-codec</groupId>
32              <artifactId>commons-codec</artifactId>
33              <version>1.15</version>
34          </dependency>
35      </dependencies>
36
37      <build>
38          <plugins>
39              <plugin>
40                  <groupId>org.apache.maven.plugins</groupId>
41                  <artifactId>maven-assembly-plugin</artifactId>
42                  <executions>
43                      <execution>
```

```
44                          <phase>package</phase>
45                          <goals>
46                              <goal>single</goal>
47                          </goals>
48                          <configuration>
49                              <archive>
50                                  <manifest>
51                                      <mainClass>it.unipi.lsmsdb.stocksim.client.app.Client</
                                            mainClass>
52                                  </manifest>
53                              </archive>
54                              <descriptorRefs>
55                                  <descriptorRef>jar-with-dependencies</descriptorRef>
56                              </descriptorRefs>
57                          </configuration>
58                      </execution>
59                  </executions>
60              </plugin>
61          </plugins>
62      </build>
63  </project>
```

pom_files/client_pom.xml

```
1   <?xml version="1.0" encoding="UTF-8"?>
2   <project xmlns="http://maven.apache.org/POM/4.0.0"
3            xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4            xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/
                 maven-4.0.0.xsd">
5       <parent>
6           <artifactId>Stocksim</artifactId>
7           <groupId>it.unipi.lsmsdb.workgroup4</groupId>
8           <version>1.0</version>
9       </parent>
10
11      <modelVersion>4.0.0</modelVersion>
12      <artifactId>Server</artifactId>
13      <packaging>jar</packaging>
14
15      <dependencies>
16          <dependency>
17              <groupId>it.unipi.lsmsdb.workgroup4</groupId>
18              <artifactId>Library</artifactId>
19              <version>1.0-SNAPSHOT</version>
20              <scope>compile</scope>
21          </dependency>
22      </dependencies>
23
24      <build>
25          <plugins>
26              <plugin>
27                  <groupId>org.apache.maven.plugins</groupId>
28                  <artifactId>maven-assembly-plugin</artifactId>
29                  <executions>
```

```
30                      <execution>
31                          <phase>package</phase>
32                          <goals>
33                              <goal>single</goal>
34                          </goals>
35                          <configuration>
36                              <archive>
37                                  <manifest>
38                                      <mainClass>it.unipi.lsmsdb.stocksim.server.app.Server</
                                            mainClass>
39                                  </manifest>
40                              </archive>
41                              <descriptorRefs>
42                                  <descriptorRef>jar-with-dependencies</descriptorRef>
43                              </descriptorRefs>
44                          </configuration>
45                      </execution>
46                  </executions>
47              </plugin>
48          </plugins>
49      </build>
50  </project>
```

pom_files/server_pom.xml

```
1   <?xml version="1.0" encoding="UTF-8"?>
2   <project xmlns="http://maven.apache.org/POM/4.0.0"
3            xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4            xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/
                 maven-4.0.0.xsd">
5       <parent>
6           <artifactId>Stocksim</artifactId>
7           <groupId>it.unipi.lsmsdb.workgroup4</groupId>
8           <version>1.0</version>
9       </parent>
10
11      <modelVersion>4.0.0</modelVersion>
12      <artifactId>Library</artifactId>
13      <packaging>jar</packaging>
14
15      <properties>
16          <maven.compiler.source>8</maven.compiler.source>
17          <maven.compiler.target>8</maven.compiler.target>
18      </properties>
19
20  </project>
```

pom_files/library_pom.xml

Part II

User Manual

Chapter 8

StockSim Server Manual

For an application dealing with the stock market, it is essential to always provide up-to-date, consistent and reliable data. This is the purpose of the **StockSim Server** program. It is not intended to be distributed to end users. It is thought to be running 24/7.

The StockSim Server has two different startup modes: the first one

```
$ java -jar Server.jar

Welcome to the StockSim Server.

DATA CONSISTENCY CHECK SUCCESS. PROCEEDING WITH UPDATE.

Updating historical data for: IRBT.
Last update date: 2020-12-30.
Days since last update: 2.
Historical data updated for IRBT. Moving on.

...

Updating historical data for: EWU.
Last update date: 2020-12-28.
Days since last update: 4.
Historical data updated for EWU. Moving on.

...


Updating historical data for: XRX.
Last update date: 2020-12-28.
Days since last update: 4.
Historical data updated for XRX. Moving on.

Historical data update terminated.
Elapsed time: 1 hrs, 37 mins, 3622 secs.
Exceptions during update process: 8.
```

```
Failed updates: [ARKG, HDS, IWFH, LDEM, WWW, XVV, GINN, AAAU].

Available Commands:

status  check databases status.
update  update databases historical data.
quit  quit Stocksim server.
>
[UPDATER THREAD] Current New York time: 2021-01-02T07:50-[America/New_York]
[UPDATER THREAD] Going to sleep for 13 hours before next update.
>
```

which executes the `historical data update` procedure right after startup. Whereas, the second startup mode can be triggered using the `--no-update` command line argument:

```
$ java -jar Server.jar --no-update

Welcome to the StockSim Server.

Available Commands:
status          check databases status.
update          update databases historical data.
quit            quit Stocksim server.
>
[UPDATER THREAD] Current New York time: 2021-01-02T06:29-[America/New_York]
[UPDATER THREAD] Going to sleep for 14 hours before next update.
> update
DATA CONSISTENCY CHECK SUCCESS. PROCEEDING WITH UPDATE.

Updating historical data for: AUPH.
Last update date: 2020-12-31.
Days since last update: 1.
Historical data for AUPH already up to date. Moving on.

...

Updating historical data for: EWU.
Last update date: 2020-12-31.
Days since last update: 1.
Historical data for EWU already up to date. Moving on.

...

Updating historical data for: XRX.
Last update date: 2020-12-31.
Days since last update: 1.
Historical data for XRX already up to date. Moving on.

Historical data update terminated.
Elapsed time: 0 hrs, 4 mins, 44 secs.
Exceptions during update process: 3.
```

```
37 Failed updates: [ARKG, XVV, DUAL].
38
39 Available Commands:
40 status       check databases status.
41 update       update databases historical data.
42 quit         quit Stocksim server.
43 >
```

in this mode, no update is executed right after startup, the main menu is shown and the user can decide the action to be performed.

In the previously shown examples, we should pay attention to the following

- `DATA CONSISTENCY CHECK SUCCESS. PROCEEDING WITH UPDATE.`
- `the stocksim server automatically detects the last update date, the number of days since the last update and fetches the required data using Yahoo Finance for EACH and EVERY ticker symbol.`
- `Historical Data Update logs.`
- `[UPDATER THREAD]`

Chapter 9

StockSim Client Manual

The StockSim Client has two different running modes: the first one

```
1 Welcome to the StockSim Client.
2
3 *** [RUNNING IN USER MODE] ***
4
5 Available Commands:
6 register        create a new user account.
7 login           login to your user account.
8 quit            quit StockSim client.
```

is the `user mode`. Whereas, the second running mode can be triggered using the `--admin` command line argument:

```
1 $ java -jar Client.jar --admin
2
3 Welcome to the StockSim Client.
4
5 *** [RUNNING IN ADMIN MODE] ***
6
7 Available Commands:
8 login       login to your admin account.
9 quit        quit StockSim client.
```

and is the `admin mode`. Although they might look like the same, the available menu actions differ once the user/admin login has been executed.

9.1 StockSim Client User Mode

Upon launching the application in *user* mode, the user is presented with three options: *register*, *login* and *quit*.
After selecting *register*, the user is asked to enter their info, such as name, surname, email, username and password:

```
> register
Name: John
Surname: Smith
E-Mail: jsmith@example.com
Username [login]: jsmith
Password [login]: hunter2
User sign up executed correctly. You can now login.
```

Once the user is registered and logged in, the application offers several options:

```
> login
Username: jsmith
Password: hunter2
User login executed correctly.
Welcome John Smith.

[jsmith] Available Commands:
search-stock            search for a stock ticker.
view-stock              view historical data for a stock ticker.
create-portfolio        create a new stock portfolio.
list-portfolios         list user stock portfolios.
view-portfolio          view user stock portfolio info.
simulate-portfolio      simulate user stock portfolio.
delete-portfolio        delete user stock portfolio.
logout                  logout from current user account.
quit                    quit StockSim client.
```

9.1.1 Search stock

The `search-stock` option allows the user to search for a specific stock in the database. The search can be done by **symbol**, by **sector** or by **country**.

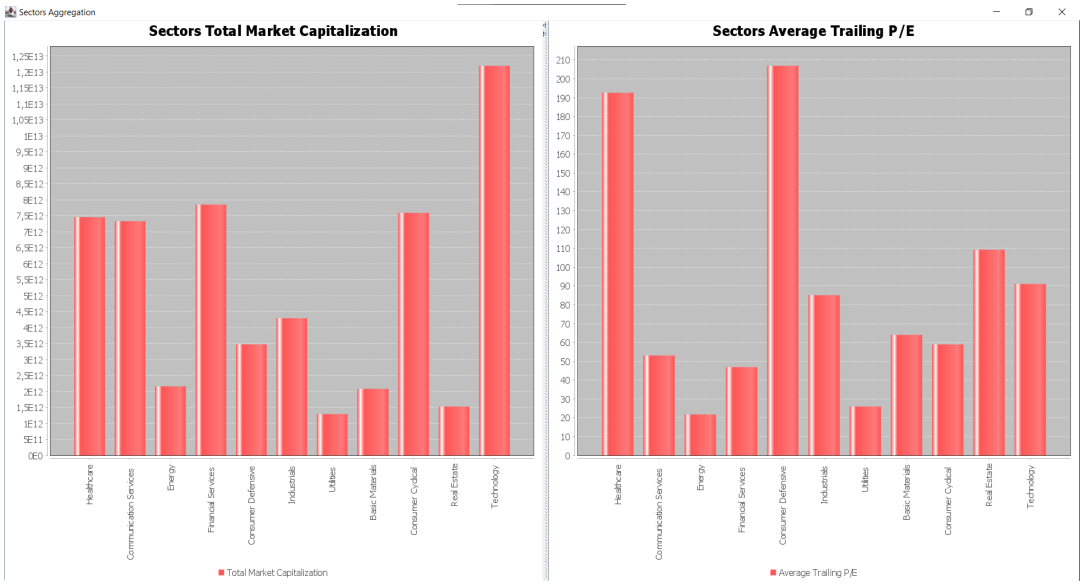
```
> search-stock
[jsmith] Available Search Commands:
symbol-search           search for a stock ticker using its ticker.
sector-search           search for a stock ticker using the sector.
country-search          search for a stock ticker using the country.
```

A search by symbol prompts the user for the symbol of the stock to be searched and returns all the information available in the database about that stock.

```
> symbol-search
Ticker Symbol: TSLA
```

```
Short Name: Tesla, Inc.
Long Name: Tesla, Inc.
Symbol: TSLA
Quote type: EQUITY
Market capitalization: 8.00041992192E11
PE ratio: 1265.3346
Market: us_market
Exchange timezone short name: EST
Exchange timezone name: America/New_York
Sector: Consumer Cyclical
Industry: Auto Manufacturers
Currency: USD
Location:  3500 Deer Creek Road Palo Alto CA United States
650-681-5000
Logo URL: https://logo.clearbit.com/tesla.com
Website: http://www.tesla.com
Long business summary:
Tesla, Inc. designs, develops, manufactures, leases, and sells electric
vehicles, and energy [...]
```

A search by sector opens a two bar charts showing aggregated data for all sectors (one for total market capitalizaion and the other for average trailing P/E) and prompts the user for the sector they are interested in. Once the desired sector is entered, a list with all related symbols is returned.




```
> sector-search
Sector Name: Energy
[ BROG, PSXP, KOS, GMLP, CLMT, NCSM, PFIE, CCLP, NGS, WES, ENSV, FTSI, AXAS, EC, DEN,
    TTI, NBLX, E, GTE, PSX, PED, NNA, VVV, PVL, AR, HP, CEQP, MUR, DK, RTLR, LEU, NGL,
```

```
    NFG, PTEN, MMLP, PAGP, NESR, NR, PBFX, TRMD, BKR, NOG, ... ]
```

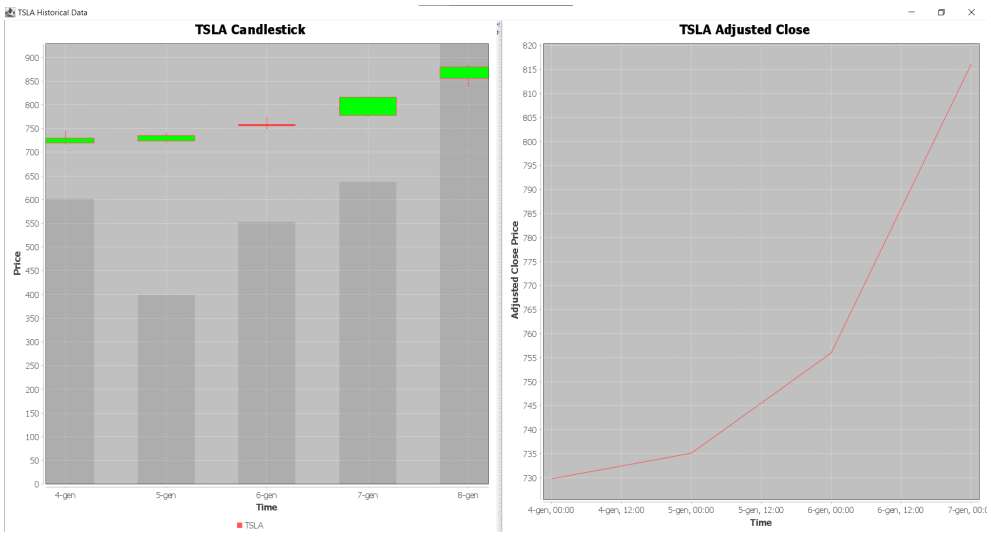
The search by country is similar to the search by sector: it also opens two bar charts with aggregated data by country and returns a list of the symbols of all the stocks belonging to the specified country.

```
1 > country-search
2 Country Name: Italy
3 [ E, NTZ, KLR, RACE, ]
```

9.1.2 View stock

The `view-stock` option allows the user to see the evolution of a stock in the market for a specific time range.
It asks the user for the stock symbol, the start and end dates and the day granularity, and then shows both a candlestick chart and a line chart of that stock for the desired time interval, along with printing all the information about that stock.

```
1  > view-stock
2  Ticker Symbol: TSLA
3  Start Date [YYYY-MM-DD]: 2021-01-01
4  End Date [YYYY-MM-DD]: 2021-01-10
5  Days granularity: 1
6  Short Name: Tesla, Inc.
7  Long Name: Tesla, Inc.
8  Symbol: TSLA
9  Quote type: EQUITY
10 Market capitalization: 8.00041992192E11
11 PE ratio: 1265.3346
12 Market: us_market
13 Exchange timezone short name: EST
14 Exchange timezone name: America/New_York
15 Sector: Consumer Cyclical
16 Industry: Auto Manufacturers
17 Currency: USD
18 Location:  3500 Deer Creek Road Palo Alto CA United States
19 650-681-5000
20 Logo URL: https://logo.clearbit.com/tesla.com
21 Website: http://www.tesla.com
22 Long business summary:
23 Tesla, Inc. designs, develops, manufactures, leases, and sells electric
24 vehicles, and energy [...]
```



9.1.3 Create portfolio

The `create-portfolio` option allows the user create a new portfolio and insert stocks into it.
The user is first required to insert a name for the new portfolio and then to type a list of comma-separated symbols of the stocks they wish to insert in the portfolio.

```
1 > create-portfolio
2 Portfolio name: Portfolio1
3 Ticker Symbols [comma separated]: TSLA, MSFT, RACE
4 Portfolio created correctly.
```

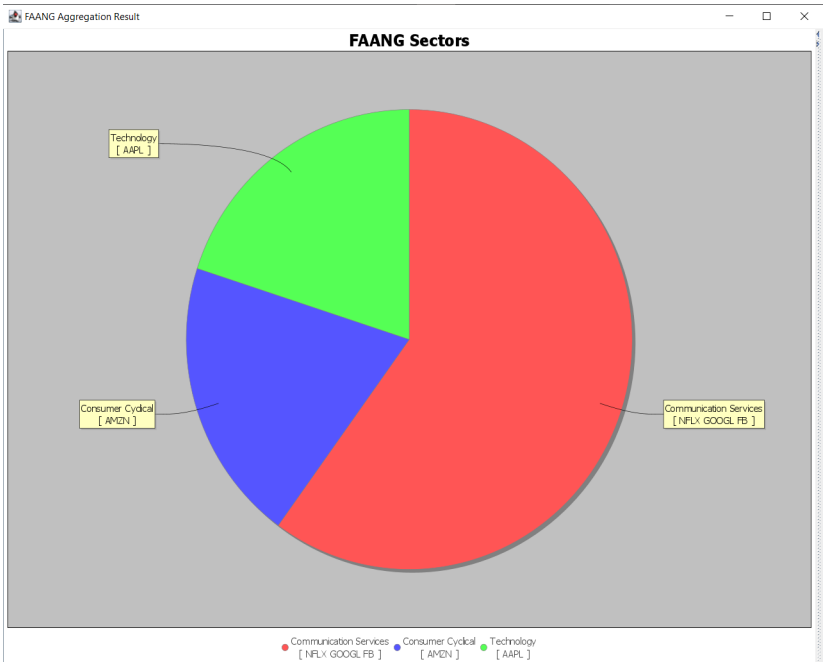
9.1.4 List portfolios

The `list-portfolios` option shows the user a list of all their portfolios and the stocks each one of them is made of.

```
1 > list-portfolios
2 Portfolio1: [ TSLA, MSFT, RACE, ]
3 FAANG: [ FB, AAPL, AMZN, NFLX, GOOGL, ]
```

9.1.5 View portfolio

The `view-portfolio` option allows the user to view the composition of one of their portfolios. It prompts the user for the name of the portfolio to be viewed and then shows a pie chart of the portfolio.



9.1.6 Simulate portfolio

TODO

9.1.7 Delete portfolio

The `delete-portfolio` option allows the user to delete one of their portfolios. It prompts the user for the name of the portfolio to be deleted and then it deletes the portfolio.

9.2 StockSim Client Admin Mode

After launching the application in *admin* mode and logging in with admin credential, the following options are available:

```
1 > login
2 Username: admin
3 Password: stocksim
4 Admin login executed correctly.
```

```
Welcome StockSim Admin.
Available Commands:

add-ticker          add a new ticker symbol to the database.
add-admin           create new admin account.
remove-admin        delete admin account.
remove-user         delete user account.
logout              logout from current admin account.
quit                quit StockSim client.
```

9.2.1 Add ticker

The `add-ticker` option allows an admin to add a ticker to the database, provided it exists in the Yahoo! Finance database. The underlying function retrieves the data from YFinance and loads it into the application database.

```
> add-ticker
Ticker symbol: LSMSDB
Asset Profile created with success. Updating historical data.
Historical data updated with success.
```

9.2.2 Add admin

The `add-admin` option allows an admin to add other admin.

```
> add-admin
Admin account name: John
Admin account surname: Doe
Admin account username: admin2
Admin account password: stocksim
New admin account created.
```

9.2.3 Remove admin

The `remove-admin` option allows an admin to remove another admin from their role.

```
> remove-admin
Admin account username: admin2
Admin account password: stocksim
Admin account deleted.
```

9.2.4 Remove user

The `remove-user` option allows an admin to delete a user from the database.

```
1 > remove-user
2 User account email: jsmith@example.com
3 User account deleted.
```