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 25, 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.

Github Repository: `https://github.com/rambodrahmani/stocksim`.

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.

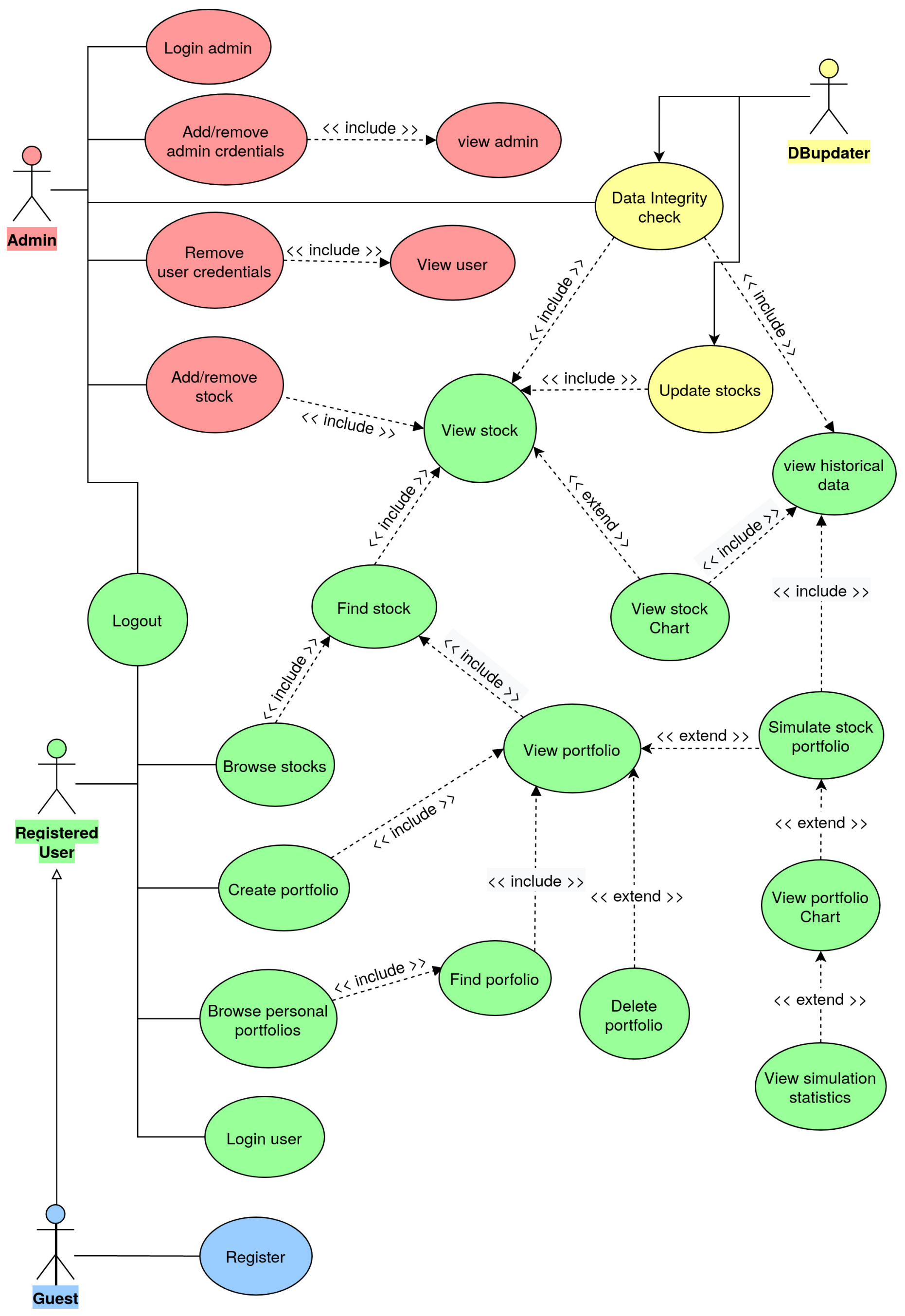
2.2.2 Non-functional requirements

- 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 to the latest market trading session;
- MongoDB and Apache Cassandra stocks data should always be consistent;
- 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

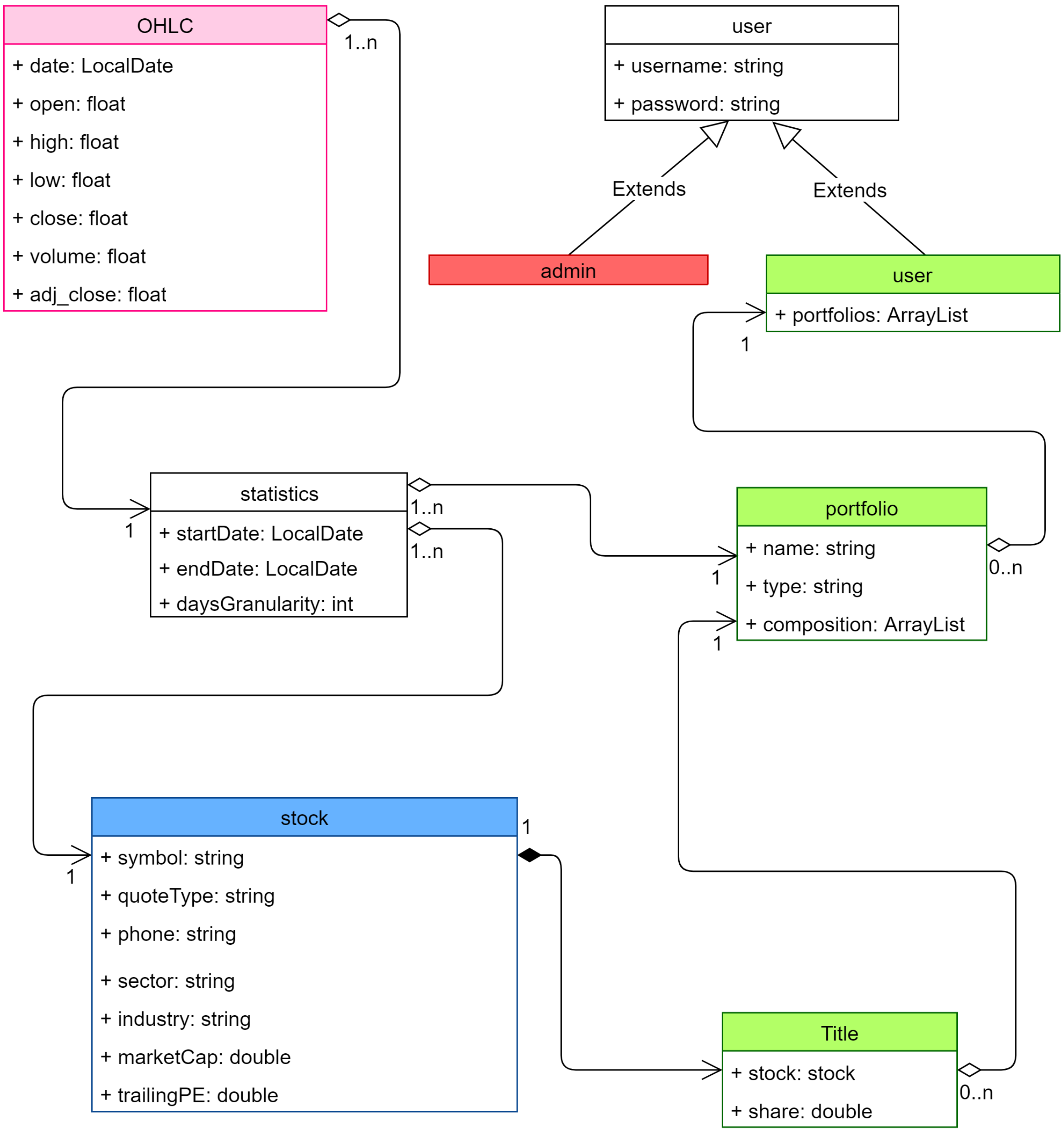
UML diagrams

3.1 Use Case diagram



3.2 Class diagram

In the following figure are shown the main entities of the application and the relationships among them.



Each user can be either an `admin` or a normal `user`. Each normal user can have zero or more portfolios. Each portfolio is composed by one o more Stock.

Chapter 4

Database

The database is composed by 8235 stocks and ETFs 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 family database (Apache Cassandra) for the storage of historical data; this is because historical data represents almost 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 family database where data storage is designed to optimize this type of operations column by column.
We decided to store any other information using a document database (MongoDB), in order to exploit the schemaless property to save memory; documents frequently needed together are stored in the same collection and indexes were created to speed up data retrieval;

4.1 Dataset

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

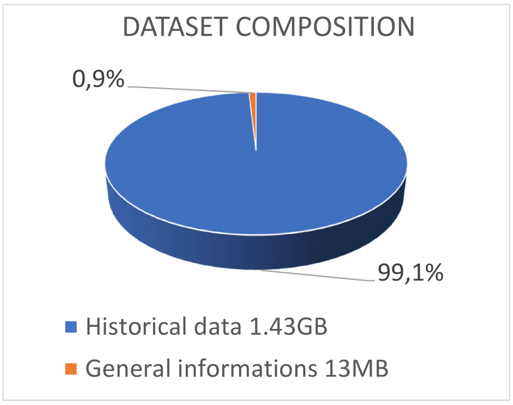
4.1.1 NasdaqTrader

The Nasdaq Stock Market (Nasdaq) is the largest U.S. equities exchange venue by volume.
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, etc.) but also to try smaller companies and/or minor sectors investments. Using NasdaqTrader we fetched the list of stocks quoted in the NASDAQ index starting from

1970 until today. For each of the retrieved stock symbol we queried **Yahoo! Finance** historical database to populate our own column family database.

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."[1]
Yahoo Finance is a service, 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 initial 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 (using Python `mongoimport` support library).
With Yahoo Finance it is also possible to retrieve historical data of market prices for each of the stocks. Using this service it has been possible to build a dataset of all the market prices of each stocks retrieved from NasdaqTrader; prices are collected daily, and we decided to take all the values from 2010 to 2020; the obtained dataset (around 1.43 GB) has been parsed to CVS files and then imported into the Apache Cassandra Cluster. Thanks to the Yahoo Finance service, it's possible to update every day the database with the last trading session data. It's also possible to add a new stocks to the dataset, quoted in any market exchange of any country.



For additional information regarding the implementation of the scripts used to fetch the raw data please refer to:

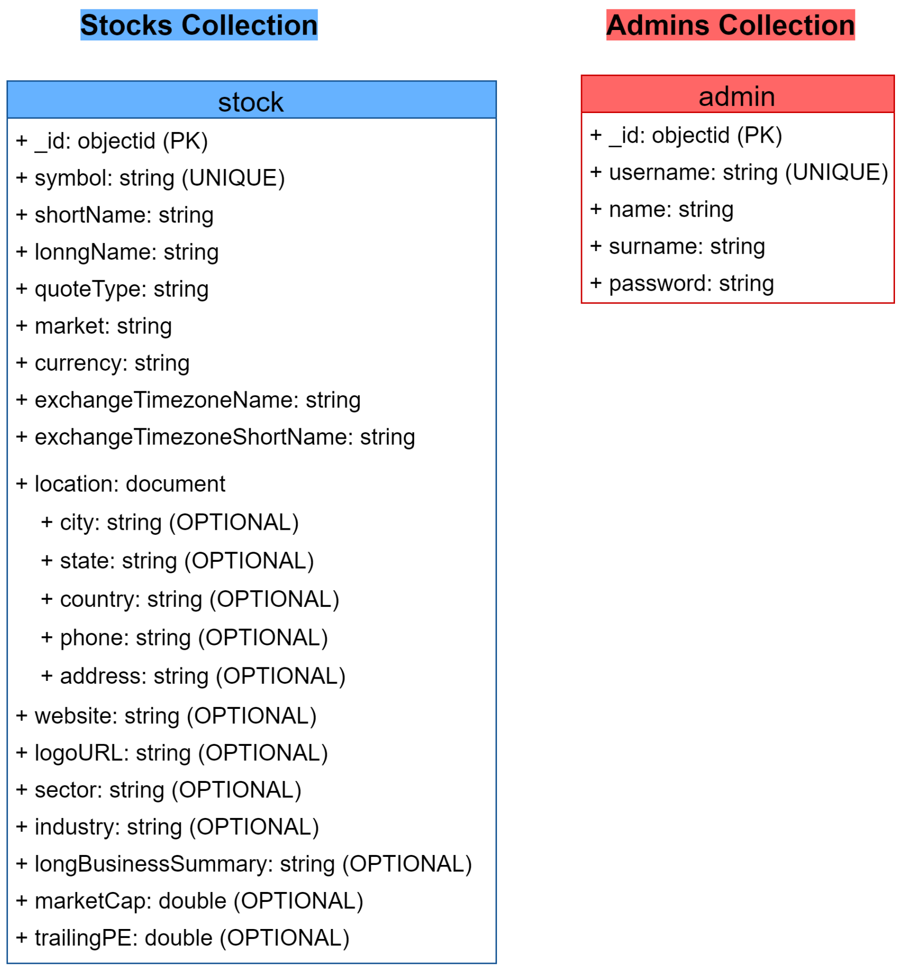
```
stocksim/scripts/
├── fetch_db_content.py
├── generate_cql.py
└── merge_json_documents.py
```

[1] www.finance.yahoo.com

4.2 MongoDB

"MongoDB is a general purpose, document-based, distributed database built for modern application developers and for the cloud era."[2]

MongoDB is a very famous document database with a great support for cloud operations, which will improve the data 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:

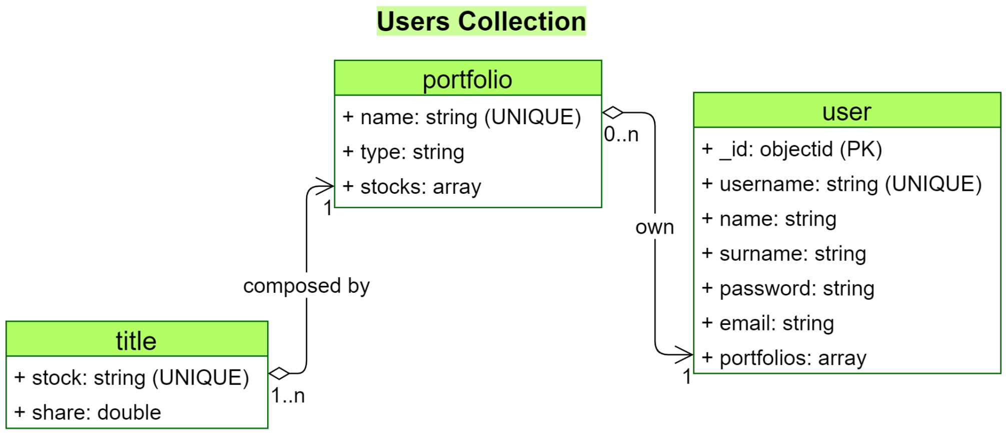

Stocks Collection

stock
+ _id: objectid (PK)
+ symbol: string (UNIQUE)
+ shortName: string
+ lonngName: string
+ quoteType: string
+ market: string
+ currency: string
+ exchangeTimezoneName: string
+ exchangeTimezoneShortName: string
+ location: document
    + city: string (OPTIONAL)
    + state: string (OPTIONAL)
    + country: string (OPTIONAL)
    + phone: string (OPTIONAL)
    + address: string (OPTIONAL)
+ website: string (OPTIONAL)
+ logoURL: string (OPTIONAL)
+ sector: string (OPTIONAL)
+ industry: string (OPTIONAL)
+ longBusinessSummary: string (OPTIONAL)
+ marketCap: double (OPTIONAL)
+ trailingPE: double (OPTIONAL)

Admins Collection

admin
+ _id: objectid (PK)
+ username: string (UNIQUE)
+ name: string
+ surname: string
+ password: string

[2]https://www.mongodb.com/



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

- The **stocks** collection contains one document for each stock; inside each document general information about the stock is stored; each stock is identified by the attribute `symbol`; some other basic information is always available too, while other may be missing for certain stocks; we decided to keep these last type of information were possible, exploiting the schemaless property of the document database; available or missing attributes have been handled properly Java-side, during the implementation;

- 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 stocks, 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 his portfolios, is frequently used all together and fetched once; on the other hand, there is no operation that involves portfolios owned by different users;

- The **admins** collection contains the admins login credentials together with few personal information about them; we decided to create a separated collection for administrators to improve the security of the administration features: in this way it 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 **countries** and **sectors**, along with classification by **level**

of capitalization and **P/E 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 stocks from the same country;
- the average P/E ratio of stocks working in the same sector;
- the average P/E ratio of stocks from the same country;

Here you are an example from the application source code:

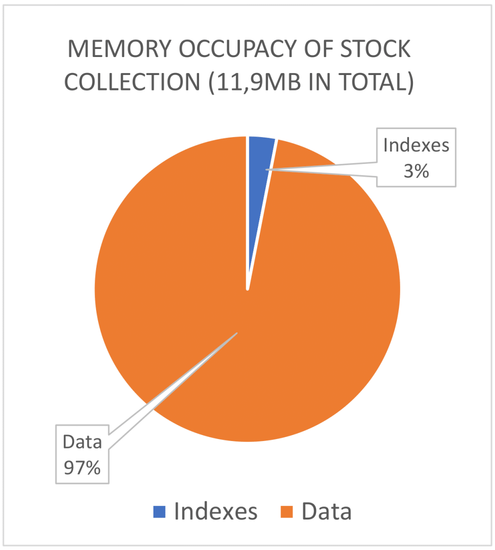
```java
/**
 * @return sectors market capitalization and trailing P/E aggregation.
 */
public ArrayList<SectorAggregation> getSectorsAggregation() {
    final ArrayList<SectorAggregation> ret = new ArrayList<>();

    // mongodb sectors aggregation
    final MongoCollection<Document> stocks = getMongoDB().getCollection(StocksimCollection.STOCKS.getCollectionName());
    final AggregateIterable<Document> industriesAggregationIterable =
        stocks.aggregate(Arrays.asList(
            Aggregates.match(Filters.and(Filters.ne("sector", null), Filters.ne("sector", ""))),
            Aggregates.group("$sector",
                                    Accumulators.sum("marketCapitalization", "$marketCap"),
                                    Accumulators.avg("avgTrailingPE", "$trailingPE")),
            Aggregates.match(
                    Filters.and(
                            Filters.ne("marketCapitalization", null), Filters.ne("marketCapitalization", ""),
                            Filters.ne("avgTrailingPE", null), Filters.ne("avgTrailingPE", "")
                    )
            )
    ));

    // iterate mongo aggregation results
    final MongoCursor<Document> iterator = industriesAggregationIterable.iterator();
    while (iterator.hasNext()) {
        final Document next = iterator.next();
        final String sector = next.getString("_id");
        final Double marketCapitalization = next.getDouble("marketCapitalization");
        final Double avgTrailingPE = next.getDouble("avgTrailingPE");
        final SectorAggregation sectorAggregation = new SectorAggregation(sector, marketCapitalization, avgTrailingPE);
        ret.add(sectorAggregation);
    }

    return ret;
}
```

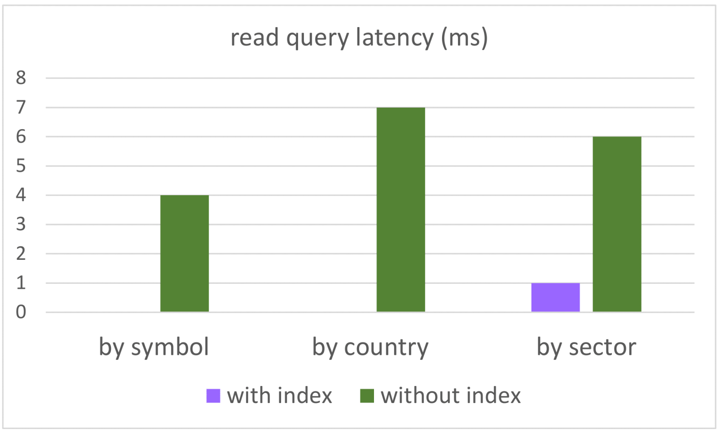
4.2.2 Indexes

In order to speed up read operations 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 **sector** 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 statistics that endorse our indexes choices:





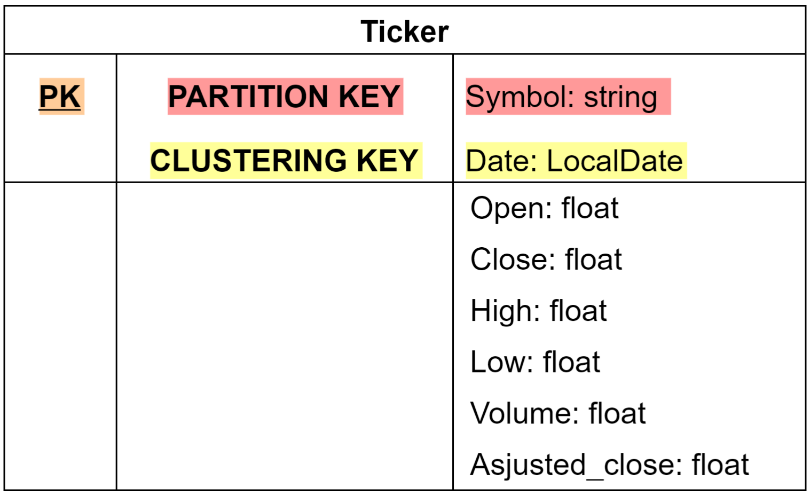
Similar results can be found about the username index in the `users` collection.

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]

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

Apache Cassandra is a database designed for high scalability and availability; it is capable of handling 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 Apache Cassandra database:



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

4.3.1 Indexes

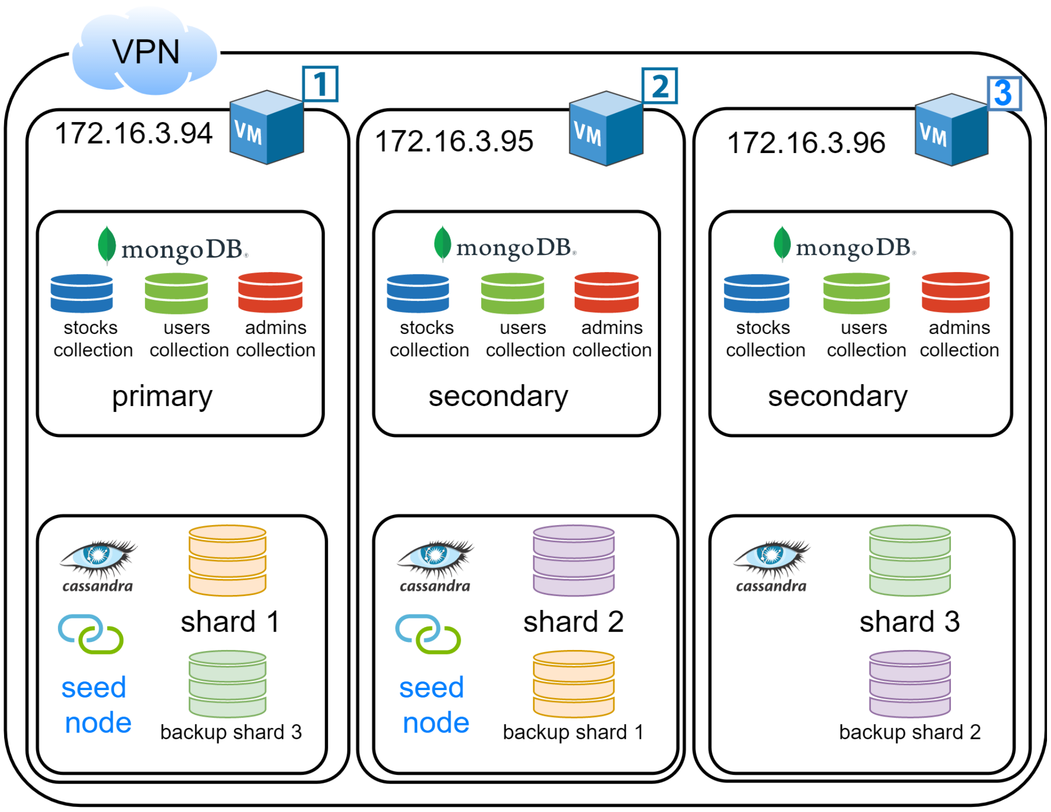
- The PARTITION KEY index is part of the PRIMARY KEY and it is used to shard the dataset across the nodes. This index is built on the attribute `symbol`, unique for each stock;
- The CLUSTERING KEY is also part of the PRIMARY KEY, and it is used to maintain rows in chronological order. This index is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

- The Cassandra cluster is built using 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 comes back online.

- The MongoDB cluster is also built using 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.



4.5 Apache Cassandra vs MongoDB

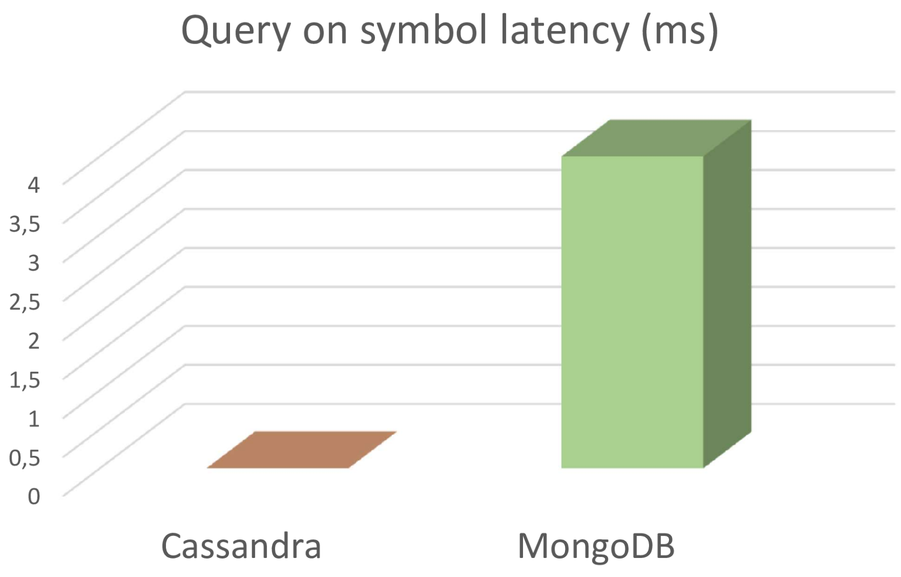
Cassandra and MongoDB are different database architectures but they share similar features. We know for sure that the usage of Cassandra can be avoided by adding a mongo Collection, which can reproduce the behaviour of the Cassandra table for historical data. We want to explain why in our opinion is 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); benchmarks were performed, on local storage[4], using the same machine:



4.5.1 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 intervals of time longer than one day; this allows us to obtain stock market historical data with customizable days granularity, computed server side; this will not over overwhelm the server: from CPU scheduling theory we know that the aggregator instructions will be executed, for each row, between one memory access and the next one (a combination of CPU bound and I/O bound

[4]In order to avoid performance degradation due to network speed.

tasks). This will greatly reduce the data to be transmitted from the node to the client, saving bandwidth and time.

```
/**
 * Apache Cassandra Stocksim CQL aggregation function.
 *
 * author: Marco Pinna, Rambod Rahmani, Yuri Mazzuoli.
 */

 /**
  * AggregateOHLC State function.
  * Executed for every row.
  */
CREATE OR REPLACE FUNCTION AggregateOHLCSFunc (
    state map<date, frozen<map<text, float>>>,
    ndays int, data date, open float, close float , high float, low float, volume float , adj_close float
)
CALLED ON NULL INPUT
RETURNS map<date, frozen<map<text, float>>>
LANGUAGE java
AS '
    if (data != null) {
        int d = 0;
        Map<String, Float> statemap = null;

        for (d = 1; d < ndays;) {
            if((statemap=state.get(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 generates a map data structure (JSON like) where
 * the key is the end date of each time interval of ndays days, and the value is
 * a map containing the aggregation of the OHLC values of the period as follows:
 * - the open price is the one of the first day of the time interval;
 * - the close and adjusted close prices are the ones of the last day;
 * - the high price is the highest price of the time interval;
 * - the low price is the lowest price of the time interval;
 * - the volume is the sum of the volumes;
 */
CREATE OR REPLACE AGGREGATE AggregateOHLC(int, date, float, float, float, float, float, float
    )
SFUNC AggregateOHLCSFunc
STYPE map<date, frozen<map<text, float>>>
INITCOND {}; /* no initial condition is necessary */

/**
 * Usage example.
 * Aggregates the OHLC prices for the TSLA (Tesla) stock on time intervals of
 * 20 days starting from 2020-06-10 till 2020-12-01.
 * It can also be used with the group by directive.
 */
SELECT AggregateOHLC(20, date, open, close, high, low, volume, adj_close)
AS OHLCAggregation from tickers WHERE
date < '2020-12-01' and date > '2020-06-10'
AND symbol = 'TSLA';
```

Chapter 5

Software Architecture

In this section the general software architecture of the application is presented.
StockSim is a Java multi-module Maven application written using the IntelliJ IDEA IDE.
We opted for a multi-module approach so that logically independent modules of the applications could be kept separated and better maintained. This allowed

- each of us to focus on the development of a particular module;
- to avoid code replication.

Furthermore, we decided to follow a feature-oriented development process and the modularity of the project helped us code 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`.
The `parent .pom` defines the parent module Stocksim, with all the dependencies and build settings that are common to the child modules.

```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>

    <!-- child modules -->
    <modules>
        <module>Client</module>
        <module>Server</module>
        <module>Library</module>
    </modules>

    [...]
```

Each child module has its own `.pom` file, necessary for the building process and for keeping track of all the required dependencies.

```
<?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">
    <parent>
        <artifactId>Stocksim</artifactId>
        <groupId>it.unipi.lsmsdb.workgroup4</groupId>
        <version>1.0</version>
    </parent>

    <!-- child module -->
    <modelVersion>4.0.0</modelVersion>
    <artifactId>Server</artifactId>
    <packaging>jar</packaging>

    <!-- dependencies -->
    <dependencies>
        <dependency>
            <groupId>it.unipi.lsmsdb.workgroup4</groupId>
            <artifactId>Library</artifactId>
            <version>1.0-SNAPSHOT</version>
            <scope>compile</scope>
        </dependency>
        <dependency>
            <groupId>it.unipi.lsmsdb.workgroup4</groupId>
            <artifactId>Library</artifactId>
            <version>1.0</version>
            <scope>compile</scope>
        </dependency>
    </dependencies>

    [...]
```

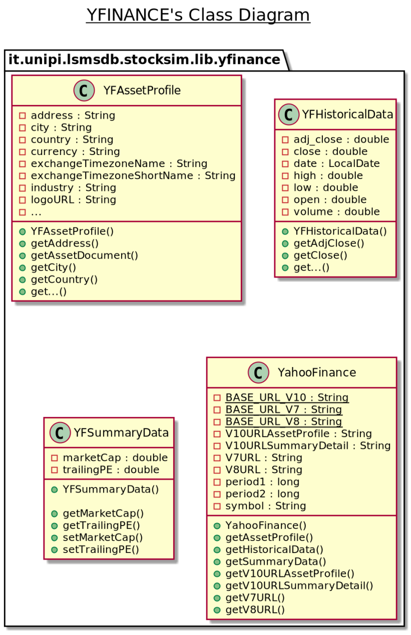
For the full content of the `.pom` files please refer to Appendix A.

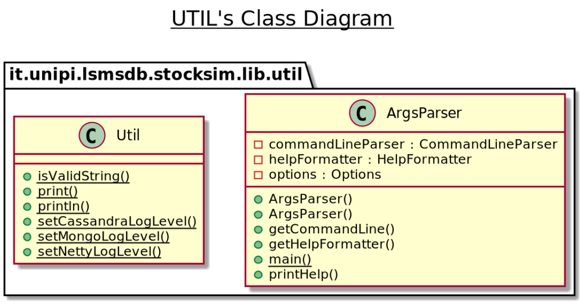
5.1.1 Library

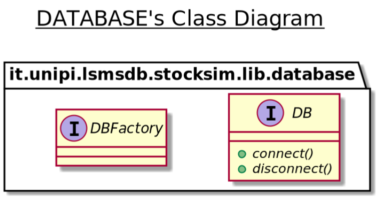
The `Library` module, contains the database APIs both for MongoDB and Apache Cassandra, together with the YahooFinance API and some utility functions. The `Library` module was created in order to avoid code replication. The `Client` and `Server` module relay on the APIs implemented in this module.
It has the following structure:

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

The `database` package contains the two APIs we wrote for the interaction with Apache Cassandra and MongoDB.
The `yfinance` package contains the API used to retrieve data from Yahoo Finance. After trying some of the offered APIs, we decided to implement our own from scratch.
The `util` package contains some utilities functions, such as the ones used to set the log level both for Apache Cassandra and MongoDB, and our custom argument parser derived from `Apache Commons CLI` used to parse command line arguments for both StockSim Server and StockSim Client.







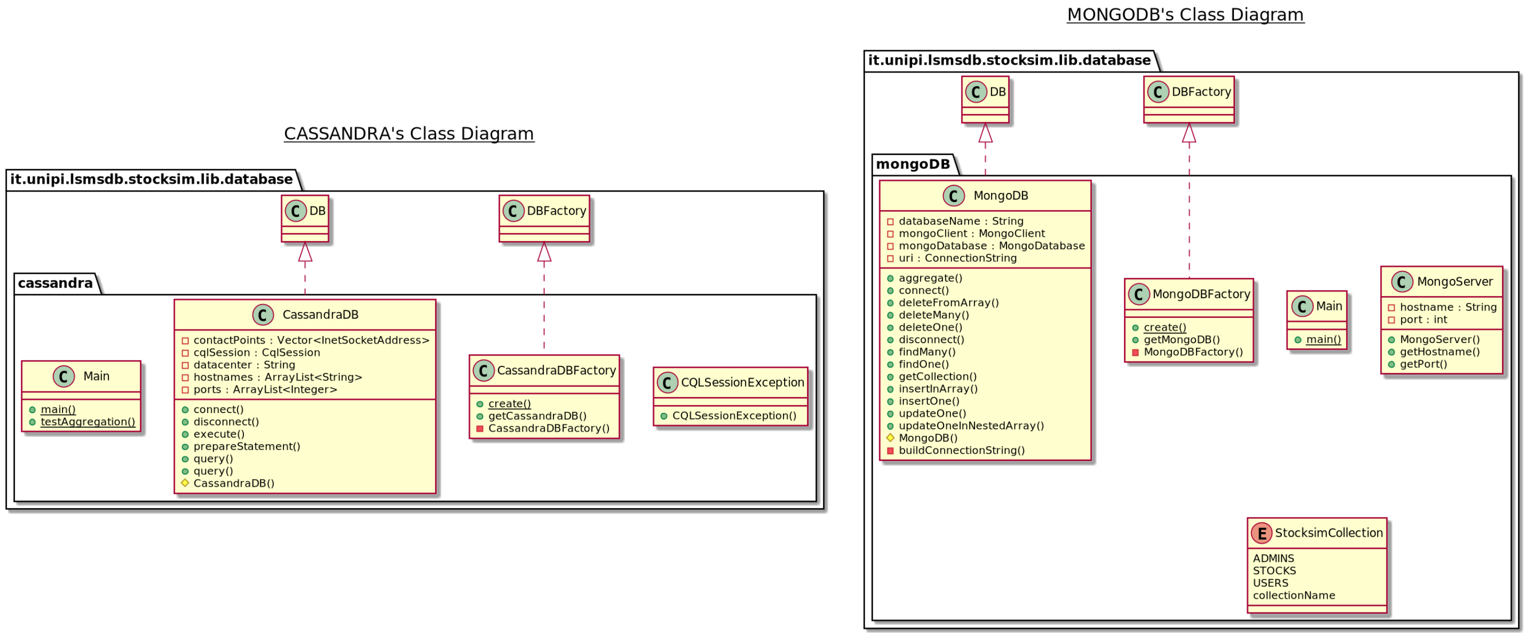


Figure 5.2: Library Module UML sample Diagrams

5.1.2 Client

The Client module contains the StockSim Client implementation. This program is intended to be distributed to end users.
It has the following structure:

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

The `app` directory contains the `Client.java` class with the entry point for the application.
Since the StockSim Client program was thought to be executed in two different running modes (namely `admin` or `user` mode), the `admin` and `user` packages contain the implementation of the functionalities available to the user in these two different running modes. The `database` directory contains the classes necessary for the connection man-

agement and the interaction with MongoDB and Apache Cassandra. Also here you can find classes intended for storing the retrieved data.
The `charting` directory contains an API for the `JFreeChart` library, used by the Client to create charts of various types.

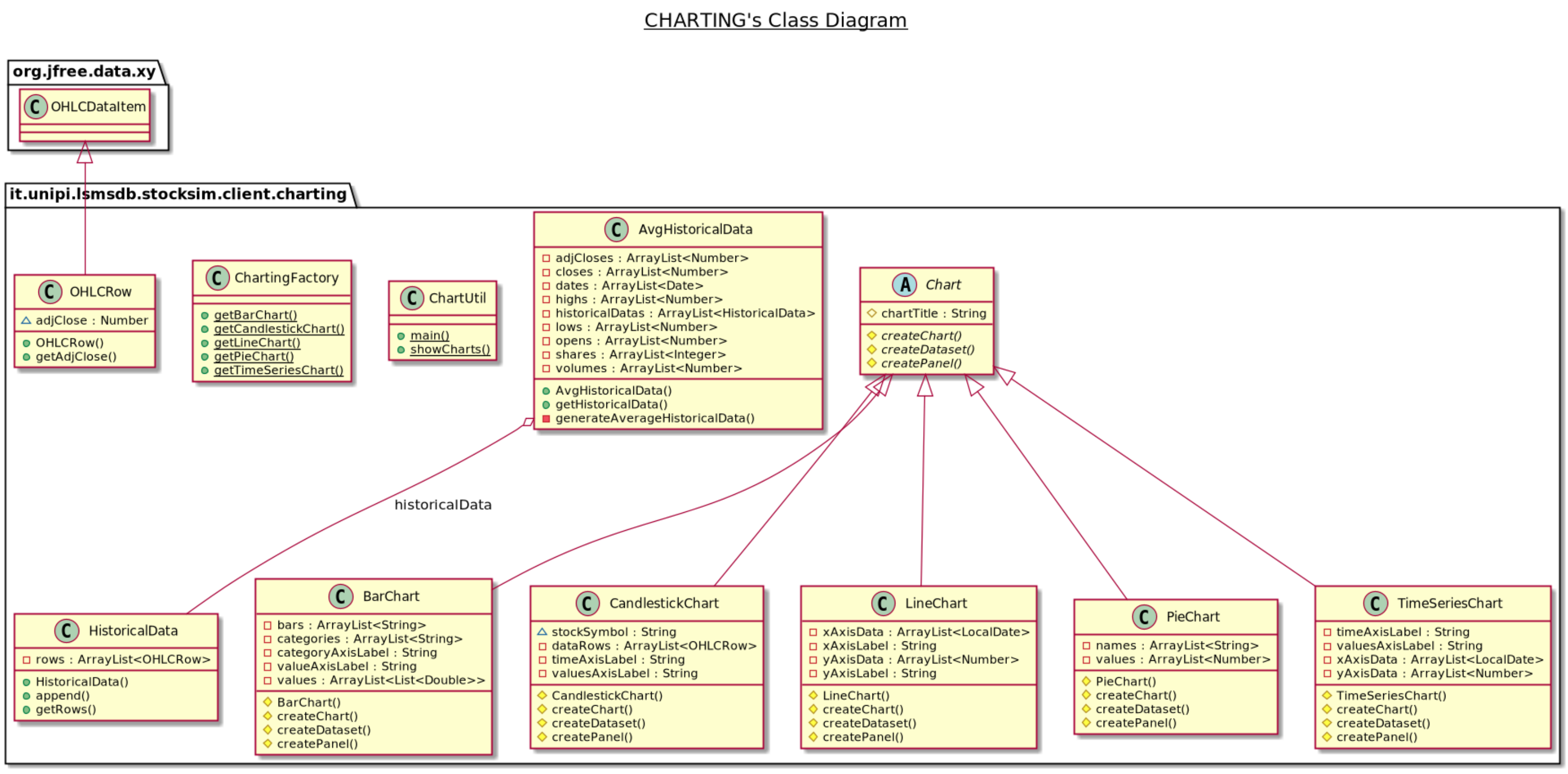


Figure 5.3: Client charting package UML Diagram as sample

5.1.3 Server

The Server module represents the StockSim Server program which takes care of keeping the data updated and consistent.
It is not supposed to be distributed: it should be always running in order to be able to retrieve the historical data prices relative to the latest trading session.
The Server module has the following structure:

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

```
│
│   ├── Server.java
│   └── ServerUtil.java
└── database
    ├── CassandraQueryBuilder.java
    └── DBManager.java
```

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

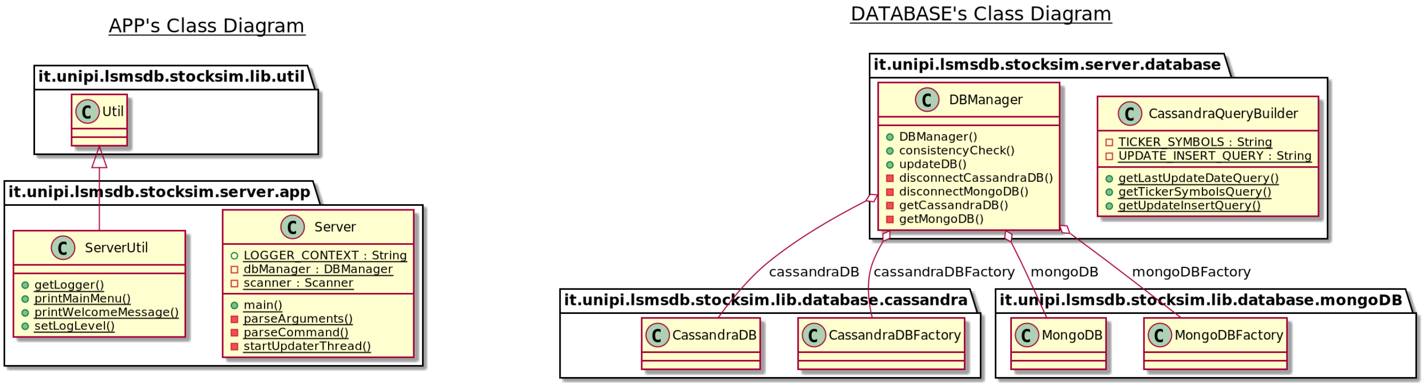


Figure 5.4: Server Module UML sample Diagrams

5.2 Apache Maven Compiler Plugin

The Compiler Plugin is used to compile the sources of the project.

```
$ mvn package
```

The command line will print out various actions, and end with the following:

```
[INFO] ------------------------------------------------------------------------
[INFO] Reactor Summary for Stocksim 1.0:
[INFO]
[INFO] Stocksim ........................................... SUCCESS [  0.001 s]
[INFO] Library ............................................ SUCCESS [  0.820 s]
[INFO] Client ............................................. SUCCESS [  6.287 s]
[INFO] Server ............................................. SUCCESS [  5.337 s]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  12.514 s
[INFO] Finished at: 2021-01-24T12:58:10+01:00
[INFO] ------------------------------------------------------------------------
```

Thanks to the proper configuration in the `.pom` file of each module, the following files are produced

```
Stocksim
├── Library
│   └── target
│       └── Library-1.0.jar
├── Client
│   └── target
│       ├── Client-1.0.jar
│       └── Client-1.0-jar-with-dependencies.jar
└── Server
    └── target
        ├── Server-1.0.jar
        └── Server-1.0-jar-with-dependencies.jar
```

The `jar-with-dependencies .jar` files provide all-inclusive Java Archives which can be used to execute the StockSim Server and the StockSim Client programs.

Chapter 6

Conclusions

The modular architecture of the project (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.

Most of the 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 such usages. Performance analysis of the query latency (as shown in paragraph 4.5) confirmed our 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 of the VMs and due to the fact that all traffic is filtered through a VPN.

At the moment the database is limited to the US market data only, but it would be really simple to integrate data from other stock markets around the world using the StockSim Client in `admin` mode.
From the Software and Data Model architectural point of view, 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 size is 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

StockSim Module POM file

```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>

    <!-- child modules -->
    <modules>
        <module>Client</module>
        <module>Server</module>
        <module>Library</module>
    </modules>

    <!-- dependencies -->
    <dependencies>
        <dependency>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-compiler-plugin</artifactId>
            <version>3.8.1</version>
        </dependency>

        <!-- Dastastax Apache Cassandra driver -->
        <dependency>
            <groupId>com.datastax.oss</groupId>
            <artifactId>java-driver-core</artifactId>
            <version>4.9.0</version>
        </dependency>
        <dependency>
            <groupId>com.datastax.oss</groupId>
            <artifactId>java-driver-query-builder</artifactId>
            <version>4.9.0</version>
        </dependency>
        <dependency>
            <groupId>com.datastax.oss</groupId>
            <artifactId>java-driver-mapper-runtime</artifactId>
            <version>4.9.0</version>
        </dependency>

        <!-- JUnit -->
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.13.1</version>
            <scope>test</scope>
        </dependency>

        <!-- MongoDB -->
        <dependency>
            <groupId>org.mongodb</groupId>
            <artifactId>mongodb-driver-sync</artifactId>
            <version>4.1.1</version>
            <scope>compile</scope>
        </dependency>

        <!-- https://mvnrepository.com/artifact/log4j/log4j -->
        <dependency>
            <groupId>org.slf4j</groupId>
            <artifactId>slf4j-api</artifactId>
            <version>1.7.30</version>
        </dependency>

        <!-- LogBack -->
        <dependency>
            <groupId>ch.qos.logback</groupId>
            <artifactId>logback-classic</artifactId>
            <version>1.2.3</version>
        </dependency>

        <!-- https://mvnrepository.com/artifact/commons-cli/commons-cli -->
        <dependency>
            <groupId>commons-cli</groupId>
            <artifactId>commons-cli</artifactId>
            <version>1.3.1</version>
        </dependency>
    </dependencies>

    <!-- build configuration -->
    <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>
</project>
```

Listing 7.1: StockSim Module POM file

Library Module POM file

```
<?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">
    <parent>
        <artifactId>Stocksim</artifactId>
        <groupId>it.unipi.lsmsdb.workgroup4</groupId>
        <version>1.0</version>
    </parent>

    <!-- child module -->
    <modelVersion>4.0.0</modelVersion>
    <artifactId>Library</artifactId>
    <packaging>jar</packaging>
    <properties>
        <maven.compiler.source>8</maven.compiler.source>
        <maven.compiler.target>8</maven.compiler.target>
    </properties>
</project>
```

Listing 7.2: Library Module POM file

Client Module POM file

```
<?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">
    <parent>
        <artifactId>Stocksim</artifactId>
        <groupId>it.unipi.lsmsdb.workgroup4</groupId>
        <version>1.0</version>
```

```xml
    </parent>

    <!-- child module -->
    <modelVersion>4.0.0</modelVersion>
    <artifactId>Server</artifactId>
    <packaging>jar</packaging>

    <!-- dependencies -->
    <dependencies>
        <dependency>
            <groupId>it.unipi.lsmsdb.workgroup4</groupId>
            <artifactId>Library</artifactId>
            <version>1.0-SNAPSHOT</version>
            <scope>compile</scope>
        </dependency>
        <dependency>
            <groupId>it.unipi.lsmsdb.workgroup4</groupId>
            <artifactId>Library</artifactId>
            <version>1.0</version>
            <scope>compile</scope>
        </dependency>
    </dependencies>

    <!-- build configuration -->
    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-assembly-plugin</artifactId>
                <executions>
                    <execution>
                        <phase>package</phase>
                        <goals>
                            <goal>single</goal>
                        </goals>
                        <configuration>
                            <archive>
                                <manifest>
                                    <mainClass>it.unipi.lsmsdb.stocksim.server.app.Server</
                                        mainClass>
                                </manifest>
                            </archive>
                            <descriptorRefs>
                                <descriptorRef>jar-with-dependencies</descriptorRef>
                            </descriptorRefs>
                        </configuration>
                    </execution>
                </executions>
            </plugin>
        </plugins>
    </build>
</project>
```

Listing 7.3: Server Module POM file

Server Module POM file

```
<?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">
    <parent>
        <artifactId>Stocksim</artifactId>
        <groupId>it.unipi.lsmsdb.workgroup4</groupId>
        <version>1.0</version>
    </parent>

    <!-- child module -->
    <modelVersion>4.0.0</modelVersion>
    <artifactId>Client</artifactId>
    <packaging>jar</packaging>

    <!-- dependencies -->
    <dependencies>
        <dependency>
            <groupId>it.unipi.lsmsdb.workgroup4</groupId>
            <artifactId>Library</artifactId>
            <version>1.0</version>
            <scope>compile</scope>
        </dependency>

        <dependency>
            <groupId>org.jfree</groupId>
            <artifactId>jfreechart</artifactId>
            <version>1.5.1</version>
        </dependency>

        <!-- https://mvnrepository.com/artifact/commons-codec/commons-codec -->
        <dependency>
            <groupId>commons-codec</groupId>
            <artifactId>commons-codec</artifactId>
            <version>1.15</version>
        </dependency>
    </dependencies>

    <!-- build configuration -->
    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-assembly-plugin</artifactId>
                <executions>
                    <execution>
                        <phase>package</phase>
                        <goals>
                            <goal>single</goal>
                        </goals>
```

```xml
                        <configuration>
                            <archive>
                                <manifest>
                                    <mainClass>it.unipi.lsmsdb.stocksim.client.app.Client</mainClass>
                                </manifest>
                            </archive>
                            <descriptorRefs>
                                <descriptorRef>jar-with-dependencies</descriptorRef>
                            </descriptorRefs>
                        </configuration>
                    </execution>
                </executions>
            </plugin>
        </plugins>
    </build>
</project>
```

Listing 7.4: Server Module POM file

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.
Failed updates: [ARKG, XVV, DUAL].
```

```

Available Commands:
status       check databases status.
update       update databases historical data.
quit         quit Stocksim server.
>
```

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

- when the `update` command is used, the first output is "`DATA CONSISTENCY CHECK SUCCESS. PROCEEDING WITH UPDATE.`", this is because before updating the historical data for each stock an integrity check is performed to check the consistency of the data stored on MongoDB and Apache Casssandra; the same operation can be executed standalone with the command `status`; in case of inconsistencies, the program will show debugging information useful to find out the issues;
- the historical data update functionality was implemented to be as automated as possible; for each stocks, the StockSim Server automatically detects the last update date, the number of days since the last update and fetches the required missing data using the Yahoo Finance API;
- while the historical data update is being performed, some logs are printed relative to the stock being updated, the number of days since the last update; also, at the end, a report is provided in order to check for eventual errors in the update process; the log level can be increased to be more verbose using the `setLogLevel` utility method provided by `ServerUtil.java`;
- the `[UPDATER THREAD]`, which prints messages to the terminal in parallel to the `main` thread, is in charge of executing the historical data update each day at 20:00 P.M. (at the end of the **after-hours trading**) New York time.

Chapter 9

StockSim Client Manual

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

```
Welcome to the StockSim Client.

*** [RUNNING IN USER MODE] ***

Available Commands:
register        create a new user account.
login           login to your user account.
quit            quit StockSim client.
>
```

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

```
$ java -jar Client.jar --admin

Welcome to the StockSim Client.

*** [RUNNING IN ADMIN MODE] ***

Available Commands:
login            login to your admin account.
quit             quit StockSim client.
>
```

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

9.1 StockSim Client Admin Mode

After launching the StockSim Client in *admin* mode and logging in with admin credentials (*username*: `admin`, *password*: `stocksim`), the following commands are available:

```
> login
Username: admin
Password: stocksim
Admin login executed correctly.
Welcome StockSim Admin.

Available Commands:
add-stock          add a new stock to the database using its ticker.
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.
>
```

9.1.1 Add stock

The `add-stock` command allows an admin to add a new stock to the database, using its ticker (stock symbol) provided that it exists in the Yahoo! Finance database.
The underlying function retrieves both general and historical data from the YFinance databases and loads it into the application databases (MongoDB and Apache Cassandra).

```
> add-stock
Ticker symbol: LSMSDB
Yahoo Finance Quotes not found. Some information will be missing.
Yahoo Finance Asset Profile not found. Some information will be missing.
Could not add new ticker.

> add-stock
Ticker: RACE
Stock already available in the database.
Could not add new stock.

> add-stock
Ticker: G.MI
Asset Profile created with success. Updating historical data.
Historical data updated with success.
New stock added.
```

9.1.2 Add admin

The `add-admin` command 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.
```

9.1.3 Remove admin

The `remove-admin` command allows an admin to delete another admin account:

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

9.1.4 Remove user

The `remove-user` command allows an admin to delete a user account:

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

9.2 StockSim Client User Mode

Upon launching the StockSim Client in *user* mode, the user is presented with three options: *register*, *login* and *quit*.
Using the *register* command, the user can sign-up:

```
> 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 commands:

```
> 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.
```

9.2.1 Search stock

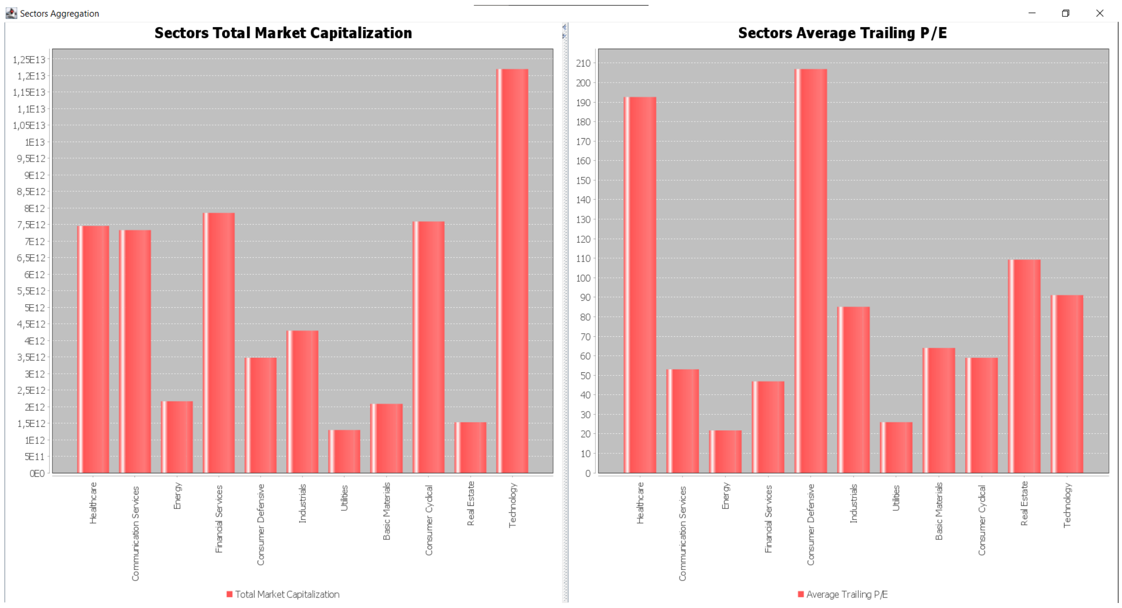
The `search-stock` command allows the user to search for a specific stock in the database. The search can be done by **symbol**, by **sector** or by **country**.

```
> 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.

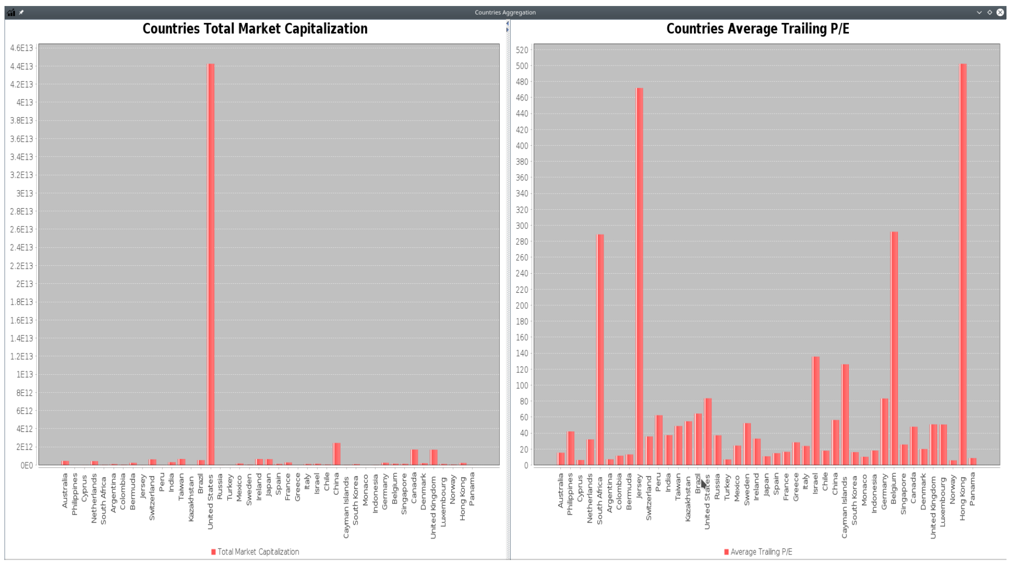
```
> symbol-search
Ticker Symbol: TSLA
Short Name: Tesla, Inc.
Long Name: Tesla, Inc.
Symbol: TSLA
Quote type: EQUITY
Market capitalization: 8.00041992192E11
PE ratio: 1265.3346
Market: us_market
Exchange timezone short name: EST
Exchange timezone name: America/New_York
Sector: Consumer Cyclical
Industry: 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 frame with two bar charts showing aggregated data for all available sectors (one chart with the total market capitalization and the other for the 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 available stocks 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 a frame with two bar charts with data aggregated by country and returns a list of the symbols of all the stocks belonging to the specified country.



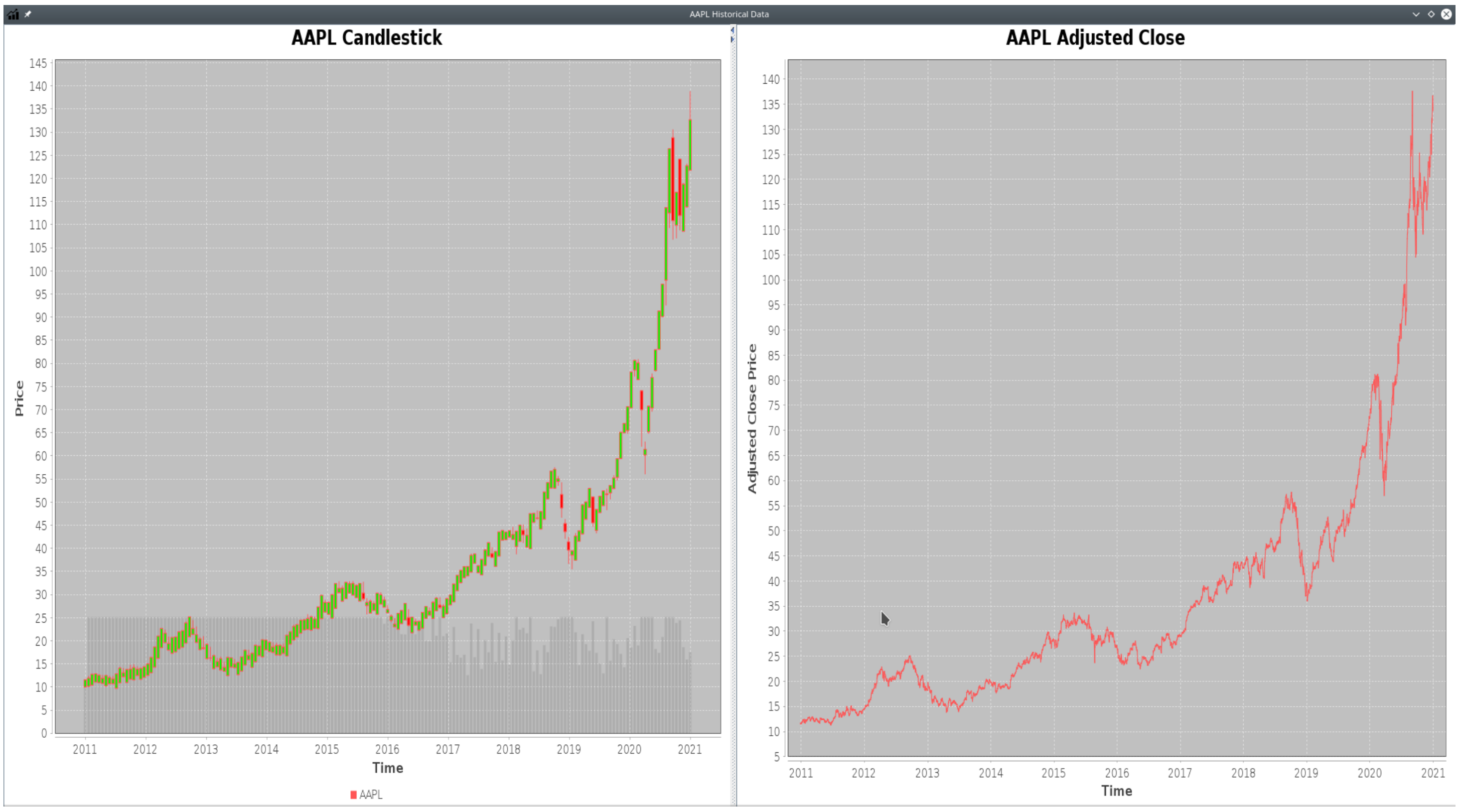
```
> country-search
Country Name: Italy
[ E, NTZ, KLR, RACE, G.MI, ]
```

9.2.2 View stock

The `view-stock` command allows the user to see the evolution of a stock in the market for a specific time interval.

It asks the user for the stock symbol, the start and end dates and the days granularity, and then shows both a candlestick chart and a time series chart of that stock for the desired time interval, along with printing all the information about that stock.

```
> view-stock
Ticker Symbol: AAPL
Start Date [YYYY-MM-DD]: 2010-01-01
End Date [YYYY-MM-DD]: 2021-01-01
Days granularity: 20

Short Name: Apple Inc.
Long Name: Apple Inc.
Symbol: AAPL
Quote type: EQUITY
Market capitalization: 2.152956821504E12
PE ratio: 40.234756
Market: us_market
Exchange timezone short name: EST
Exchange timezone name: America/New_York
Sector: Technology
Industry: Consumer Electronics
Currency: USD
Location:  One Apple Park Way Cupertino CA United States
408-996-1010
Logo URL: https://logo.clearbit.com/apple.com
Website: http://www.apple.com
Long business summary:
Apple Inc. designs, manufactures, and markets smartphones, personal computers,
tablets, wearables, [...]
```



9.2.3 Create portfolio

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

```
> create-portfolio
Portfolio name: Portfolio5
Stock Symbols [comma separated]: AAPL, MSFT, TSLA
Stock Shares [comma separated]: 14000, 20000, 40000
Portfolio created correctly.
```

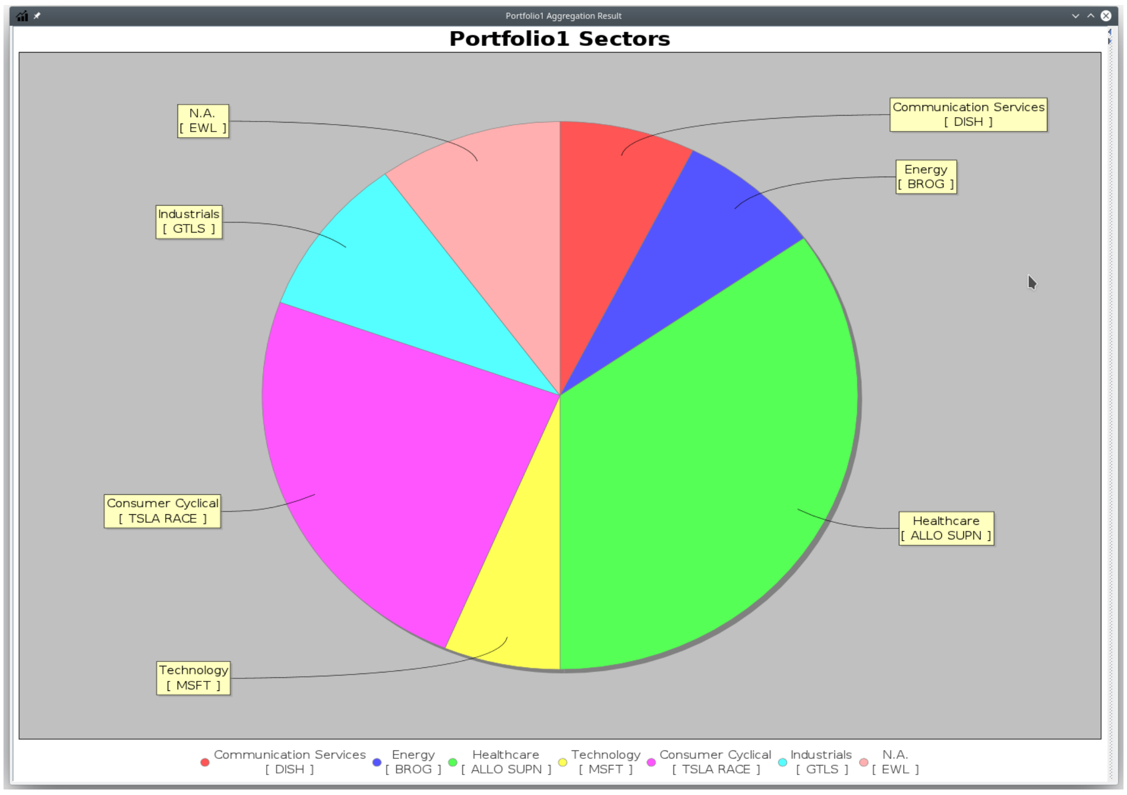
9.2.4 List portfolios

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

```
> list-portfolios
Portfolio1: [ TSLA (10), MSFT (12), RACE (13), ]
Portfolio2: [ TSLA (1), MSFT (2), RACE (3), DISH (4), ]
Portfolio3: [ TSLA (1), MSFT (30), RACE (3), DISH (4), BROG (5), ALLO (6), ]
Portfolio4: [ RACE (1), AAPL (10), MSFT (100), TSLA (1000), ]
Portfolio5: [ AAPL (14000), MSFT (20000), TSLA (40000), ]
```

9.2.5 View portfolio

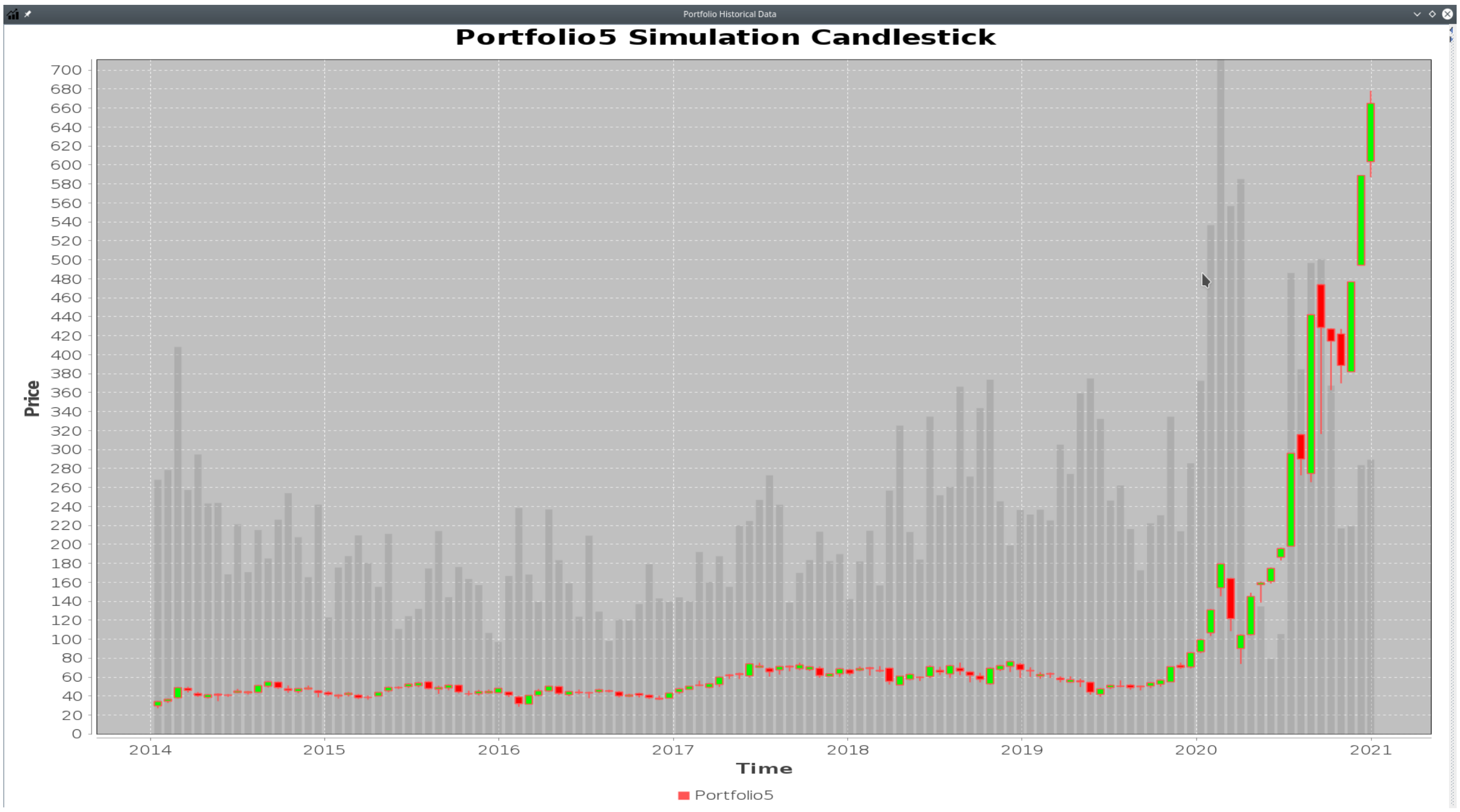
The `view-portfolio` command allows the user to view the composition of one of their portfolios. It prompts the user for the name of the portfolio to be viewed and then shows a pie chart of the portfolio.


Portfolio1 Aggregation Result
Portfolio1 Sectors
N.A. [EWL]
Communication Services [DISH]
Energy [BROG]
Industrials [GTLS]
Consumer Cyclical [TSLA RACE]
Healthcare [ALLO SUPN]
Technology [MSFT]
● Communication Services [DISH] ● Energy [BROG] ● Healthcare [ALLO SUPN] ● Technology [MSFT] ● Consumer Cyclical [TSLA RACE] ● Industrials [GTLS] ● N.A. [EWL]

9.2.6 Simulate portfolio

The `simulate-portfolio` command allows the user to simulate the trading of the stocks in their Portfolio for the desired time interval. The user is prompted for the name of the portfolio to be simulated, the start and end dates and the days granularity to be used in the aggregation. As a result, the candlestick resulting from the simulation is displayed:

```
> simulate-portfolio
Portfolio name: Portfolio5
Start Date [YYYY-MM-DD]: 2014-01-01
End Date [YYYY-MM-DD]: 2021-01-01
Days granularity: 20
```



9.2.7 Delete portfolio

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

```
> delete-portfolio
Portfolio name: Portfolio5
Portfolio deleted.
```

9.2.8 Quit

The `quit` command can be used to terminate the program.

```
> quit

Process finished with exit code 0
```