

StockHome - Web Application

User interface for a financial analysis tool

Gilad Geron

supervised by
Prof. Dr. Michele Lanza

Abstract

Stock exchange markets are dynamic and constantly produce vast amounts of information. Investors use this information to perform various statistical calculations in order to find what to invest in. This process can be considered a long and tedious task. In our project we introduce a solution, which automates this process and therefore allows investors to make decisions quicker and take into account much more information. Our role in the project is to introduce a user interface that portrays the constantly changing information and lets the user interact with this information with a strong emphasis on usability.

Acknowledgments

I would like to dedicate this thesis to my family who made it all possible for me.

A special thanks to Michele Lanza for standing by our side and assisting us during dark times.

Last but not least, I would like to thank my friends who spent those long sleepless nights in the lab with me. You guys are great.

Contents

Abstract	i
Acknowledgments	ii
1 Introduction	1
1.1 Structure of the Document	2
2 Problem & Related Work	3
2.1 General problems	3
2.2 Financial information providers	3
2.3 Financial statistical calculation tools	5
3 Solution	7
3.1 Back-end framework	11
3.1.1 Data fetching	11
3.1.2 Statistical calculation core	11
3.1.3 Filtering	11
3.1.4 Optimisation core	12
3.1.5 Summary	12
3.2 Database	12
3.3 Front-end framework	12
3.3.1 Portfolio database handler	13
3.3.2 Cart database handler	13
3.3.3 Authentication servlet	13
3.4 Browser	13
4 Validation	14
4.1 Getting started with StockHome	17
4.2 Portfolio Management	17
4.3 Market observation	20
4.3.1 Searching for a single stock	20
4.3.2 Filtering stocks	24
4.3.3 Cart	25
4.3.4 Optimisation	25
5 Conclusions	29
5.1 Goals achieved	29
5.2 Future Work	30

A Technologies	31
A.1 Ruby	31
A.2 MySQL database	31
A.3 Hibernate	32
A.4 Apache Tomcat Java server	32
A.5 JSP	32
A.6 JavaScript	32
A.7 Prototype	33
A.7.1 Scriptaculous	34
A.7.2 flotr	34
A.8 AJAX	34
A.9 JSON	34

Chapter 1

Introduction

Stock exchange markets are dynamic and constantly produce vast amounts of information from the thousands of stocks being traded all around the world. Therefore, choosing the right handful of stocks to invest in is like finding a needle in a haystack. To help Investors find out what stocks to invest in, they use various statistical calculations and estimations.

These calculations require gathering and manipulating large amounts of data, which makes it impractical to carry out by hand. The use of spreadsheets is a common way to carry out this task more easily. Although this method is effective, finding the data needed for the calculation, preparing the spreadsheet and manipulating the spreadsheet can be long and error prone. It is sufficient for one field in the spreadsheet to have an incorrect value and the rest of the data becomes corrupt. Another major flaw with this approach is that as information inevitably changes, the calculations become outdated and the whole process needs to be redone.

In our project we introduce a web-based solution, which automates this process even further. Firstly, the gathering of information is done by information fetching scripts that crawl through the web and dump financial information into a database. These scripts are also in charge of maintaining the information stored in the database up-to-date. Secondly, all the statistical calculations are done behind the scenes, which eliminates the need of the investors to manipulate any raw data. Additionally, a top-down filtering approach lets the user specify a certain criteria and find stocks that match it.

Our solution tries to reduce the amount of time spent by the user carrying out the calculation and increase the amount time spent by the user experimenting with new ways of investment. In essence, this tool helps investors become more efficient as they can make decisions quicker and take into account much more information than they normally would by using other methods. Furthermore, taking away from the user the responsibility of carrying out the actual calculations and replacing it by a tested automated procedure, largely diminishes the chance of errors occurring.

Our role in the project is to develop a user interface that portrays the constantly changing information and lets the user interact with this information with a strong emphasis on usability.

1.1 Structure of the Document

We start this paper by introducing the problem matter of our project, discussing the different approaches currently used in the industry alongside their flaws.

We then move on to explaining our solution and how each feature we offer in the solution minimizes or eliminates their respective problem.

Once the solution has been introduced, we move on to the solution validation part, where we show how our tool operates, discuss how we attempted to maximize usability and efficiency from the user interface point of view. Additionally, we explain briefly what happens in each feature from the technical point of view. This is done by going through various use-case scenarios.

Finally, we conclude the report by discussing how effectively our solution solves the problem and future work that can be done to improve our solution.

Chapter 2

Problem & Related Work

As mentioned in the introduction stock exchange markets continuously produce large amounts of data. Currently, there is no tool that is capable of gathering, storing and managing financial information as well as carrying out statistical calculations on it. The tools available offer either information providing related functionality or information processing related functionality. This means that if the user would like additional information, he would need to gather the necessary information and carry out the calculations by himself. First, we will discuss general problems that apply to all tools then proceed to introduce each approach and discuss problems related to tools of each approach.

2.1 General problems

Each tool requires some sort of installation phase. The installation phase may vary in system requirement and complexity. However, the majority of tools work only with Windows, meaning that mac OS and Linux users are left out. Furthermore, some of these tools are offered as an addition to another product, meaning that the user needs to buy or download the original software and install it before being able to install the addition. This fact may also add additional costs to the user.

2.2 Financial information providers

This type of tools lets the user log into the database of the provider and retrieve desired information. An example of such a tool is the service offered by Bloomberg called Bloomberg Professional. Bloomberg is the largest financial software services, news and data company in the world. They have an extensive coverage of real-time and historical information of thousands of financial assets. Bloomberg Professional offers a terminal access to that data. There are currently approximately 250,000 subscribers to this service world-wide. However, although Bloomberg Professional provides a broad range of features, it has several problems.

One such problem is the price. A basic subscription to Bloomberg Professional terminal starts at \$1,500 per month. Moreover, the installation phase of Bloomberg makes it even more expensive and complicated as it requires special hardware in order

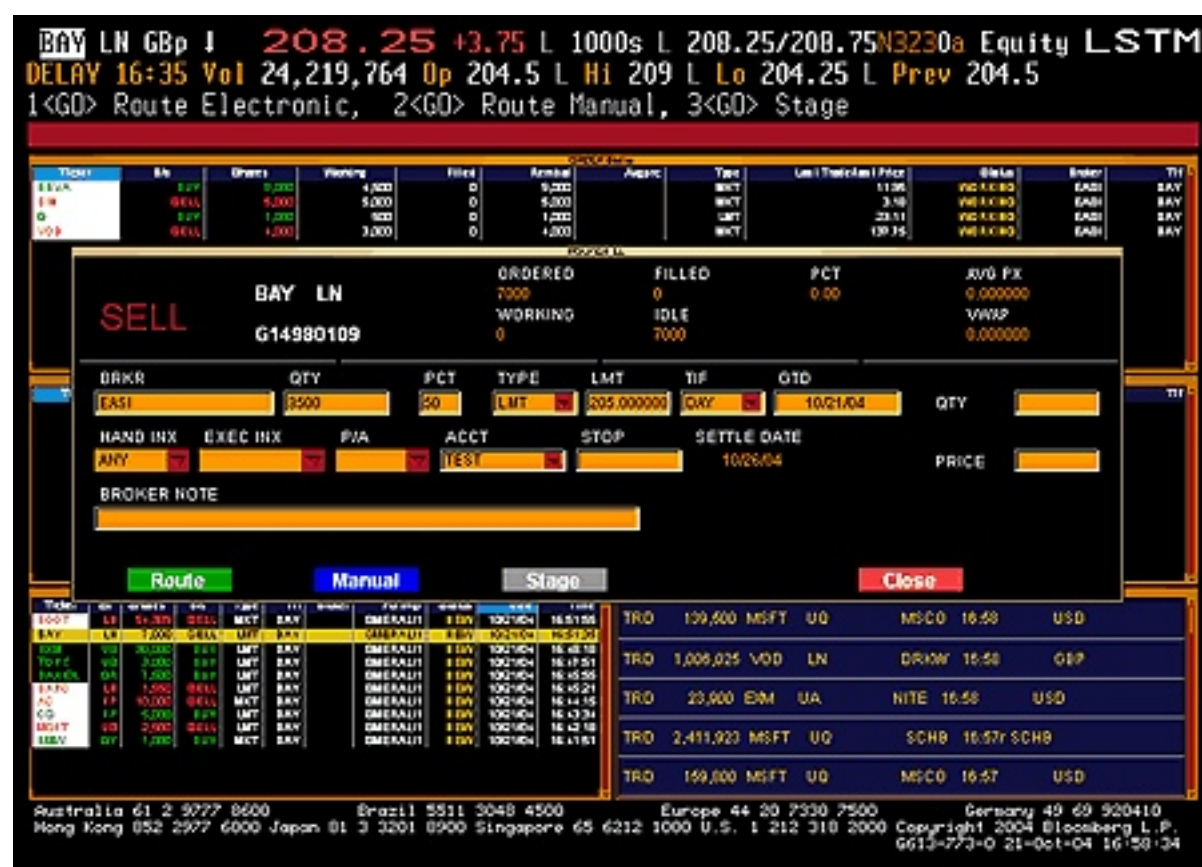


Figure 2.1: Bloomberg user interface

to enhance user-friendliness. Such hardware may include, the Bloomberg dual 17 inch screen set, the Bloomberg keyboard, an authentication kit and a communication kit. Therefore, the price of such a tool might make it undesirable for the average person and only appeal to big corporations who can afford this service.

Another major flaw with Bloomberg is its user interface. Despite the fact that Bloomberg offers full customizability of its user interface, the design and color layout gives the user an outdated feel and reduces usability. Figure 2.1 shows a screenshot of the Bloomberg user interface. The interface resembles the MS-DOS interface and has a black background. Moreover, the user interface is filled with an overwhelming amount of data. The combination of a dark background and the amount of information shown on the interface, makes it harder for the user to navigate and find the functionality and data he is looking for.

Even though Bloomberg provides a wide variety of financial data, it does not let the user carry out additional on-the-run calculations.

2.3 Financial statistical calculation tools

The main purpose of such tools is to carry out calculations. Different tools have different ways to carry out calculations. The most commonly used method of carrying out calculations on large amounts of data is using a spreadsheet. Spreadsheets are very powerful tools, yet they are subject to be error-prone due to several reasons.

Firstly, the user needs to find a reliable source of information and import the data from it. Usually, the user would carry out the same calculations on several stocks but would have to collect the data for each stock. This process is time consuming and repetitive, which may cause the user to collect the wrong data.

Once the user has fetched the data successfully, the user then needs to format the data in a way which is appropriate for carrying out calculations. The fact that the user has to manipulate thousands of cells, may cause the user to make a mistake. For example, it is enough for the user to format one cell in the wrong way and the entire calculation becomes incorrect.

	A	B	C	D	E	F	G	H	I	J	K
1											
2											
3	#	INXV Tremont Emerging Markets Fund									
4	Jan-00	Return Time Series	Cumul NAV	%Cumulated Returns	%Drawdown	Mon. Max	Min	BP-5%*2			VMI MAX
5	Feb-00	2.08%	1,000.00	2.08%	0.00%	5.94%	-3.96%	-	-	1,359.96	1,439.924
6	Mar-00	5.57%	1,077.66	7.65%	0.00%	5.94%	-3.96%	-	-	1,359.96	1,439.924
7	Apr-00	2.96%	1,109.56	10.61%	0.00%	5.94%	-3.96%	-	-	1,359.96	1,439.924
8	May-00	-2.19%	1,085.59	8.45%	2.16%	5.94%	-3.96%	0.05%	-	1,359.96	1,439.924
9	Jun-00	-1.44%	1,069.96	7.01%	3.60%	5.94%	-3.96%	0.02%	-	1,359.96	1,439.924
10	Jul-00	3.74%	1,109.97	10.75%	0.00%	5.94%	-3.96%	-	-	1,359.96	1,439.924
11	Aug-00	3.39%	1,147.60	14.14%	0.00%	5.94%	-3.96%	-	-	1,359.96	1,439.924
12	Sep-00	3.43%	1,186.97	17.57%	0.00%	5.94%	-3.96%	-	-	1,359.96	1,439.924
13	Oct-00	-1.76%	1,166.08	15.81%	1.76%	5.94%	-3.96%	0.03%	-	1,359.96	1,439.924
14	Nov-00	-1.92%	1,143.69	13.89%	3.68%	5.94%	-3.96%	0.04%	-	1,359.96	1,439.924
15	Dec-00	-1.45%	1,127.10	12.44%	5.13%	5.94%	-3.96%	0.02%	-	1,359.96	1,439.924
16	Jan-01	0.98%	1,134.77	13.12%	4.45%	5.94%	-3.96%	-	-	1,359.96	1,439.924
17	Feb-01	4.31%	1,183.68	17.43%	0.14%	5.94%	-3.96%	-	-	1,359.96	1,439.924
18	Mar-01	-2.11%	1,158.70	15.32%	2.25%	5.94%	-3.96%	0.04%	-	1,359.96	1,439.924
19	Apr-01	-1.72%	1,138.77	13.60%	3.97%	5.94%	-3.96%	0.03%	-	1,359.96	1,439.924
20	May-01	-0.11%	1,137.52	13.49%	4.08%	5.94%	-3.96%	0.00%	-	1,359.96	1,439.924
21	Jun-01	3.45%	1,177.22	16.98%	0.59%	5.94%	-3.96%	-	-	1,359.96	1,439.924
22	Jul-01	2.54%	1,207.12	19.52%	0.00%	5.94%	-3.96%	-	-	1,359.96	1,439.924
23	Aug-01	-1.46%	1,189.49	18.06%	1.46%	5.94%	-3.96%	0.02%	-	1,359.96	1,439.924
24	Sep-01	2.29%	1,216.73	20.35%	0.00%	5.94%	-3.96%	-	-	1,359.96	1,439.924
25	Oct-01	-3.96%	1,168.55	16.39%	3.96%	5.94%	-3.96%	0.16%	-	1,359.96	1,439.924
26	Nov-01	2.86%	1,201.97	19.25%	1.10%	5.94%	-3.96%	-	-	1,359.96	1,439.924
27	Dec-01	5.77%	1,271.33	25.02%	0.00%	5.94%	-3.96%	-	-	1,359.96	1,439.924
28	Jan-02	3.28%	1,313.03	28.30%	0.00%	5.94%	-3.96%	-	-	1,359.96	1,439.924
29	Feb-02	2.73%	1,348.87	31.03%	0.00%	5.94%	-3.96%	-	-	1,359.96	1,439.924
30	Mar-02	2.65%	1,384.62	33.68%	0.00%	5.94%	-3.96%	-	-	1,359.96	1,439.924
31	Apr-02	2.39%	1,417.71	36.07%	0.00%	5.94%	-3.96%	-	-	1,359.96	1,439.924
32	May-02	1.82%	1,443.61	37.89%	0.00%	5.94%	-3.96%	-	-	1,359.96	1,439.924
33	Jun-02	-0.23%	1,440.19	37.66%	0.23%	5.94%	-3.96%	0.00%	-	1,359.96	1,439.924
34	Jul-02	-3.10%	1,395.54	34.56%	3.33%	5.94%	-3.96%	0.10%	-	1,359.96	1,439.924
35	Aug-02	-2.59%	1,359.96	32.01%	5.88%	5.94%	-3.96%	0.07%	1,359.96	1,359.96	1,439.924
36	Sep-02	3.87%	1,412.59	35.88%	2.01%	5.94%	-3.96%	-	-	1,359.96	1,439.924
37	Oct-02	-2.93%	1,371.20	32.95%	4.94%	5.94%	-3.96%	0.09%	-	1,359.96	1,439.924
38	Nov-02	4.50%	1,432.90	37.45%	0.44%	5.94%	-3.96%	-	-	1,359.96	1,439.924
39	Dec-02	2.71%	1,471.74	40.16%	0.00%	5.94%	-3.96%	-	-	1,359.96	1,439.924
40	Jan-03	0.72%	1,482.33	40.88%	0.00%	5.94%	-3.96%	-	-	1,359.96	1,439.924
41	Feb-03	0.52%	1,490.04	41.40%	0.00%	5.94%	-3.96%	-	-	1,359.96	1,439.924
42	Mar-03	2.64%	1,529.38	44.04%	0.00%	5.94%	-3.96%	-	-	1,359.96	1,439.924
43	Apr-03	1.37%	1,550.33	45.41%	0.00%	5.94%	-3.96%	-	-	1,359.96	1,439.924
44	May-03	5.94%	1,642.42	51.35%	0.00%	5.94%	-3.96%	-	-	1,359.96	1,439.924
45	Jun-03	4.40%	1,714.69	55.75%	0.00%	5.94%	-3.96%	-	-	1,359.96	1,439.924
46	Jul-03	1.81%	1,745.72	57.56%	0.00%	5.94%	-3.96%	-	-	1,359.96	1,439.924
47	Aug-03	0.07%	1,746.94	57.63%	0.00%	5.94%	-3.96%	-	-	1,359.96	1,439.924
48	Sep-03	2.74%	1,784.81	60.37%	0.00%	5.94%	-3.96%	-	-	1,359.96	1,439.924
49	Oct-03	2.42%	1,838.24	62.79%	0.00%	5.94%	-3.96%	-	-	1,359.96	1,439.924
50	Nov-03	1.85%	1,872.25	64.64%	0.00%	5.94%	-3.96%	-	-	1,359.96	1,439.924
51	Dec-03	1.47%	1,899.77	66.11%	0.00%	5.94%	-3.96%	-	-	1,359.96	1,439.924
52	Jan-04	3.07%	1,958.10	69.18%	0.00%	5.94%	-3.96%	-	-	1,359.96	1,439.924
53	Feb-04	2.47%	2,006.46	71.65%	0.00%	5.94%	-3.96%	-	-	1,359.96	1,439.924
54	Mar-04	1.81%	2,042.78	73.46%	0.00%	5.94%	-3.96%	-	-	1,359.96	1,439.924
55	Apr-04	1.93%	2,082.20	75.39%	0.00%	5.94%	-3.96%	-	-	1,359.96	1,439.924
56	May-04	-3.08%	2,018.07	72.31%	3.08%	5.94%	-3.96%	0.09%	-	1,359.96	1,439.924
57	Jun-04	-0.89%	2,000.11	71.42%	3.97%	5.94%	-3.96%	0.01%	-	1,359.96	1,439.924
58	Jul-04	0.58%	2,011.71	72.00%	3.39%	5.94%	-3.96%	-	-	1,359.96	1,439.924
59	Aug-04	0.52%	2,022.17	72.52%	2.87%	5.94%	-3.96%	-	-	1,359.96	1,439.924
60	Sep-04	2.79%	2,078.59	75.31%	0.08%	5.94%	-3.96%	-	-	1,359.96	1,439.924
61	Oct-04	2.73%	2,135.34	78.04%	0.00%	5.94%	-3.96%	-	-	1,359.96	1,439.924
62	Nov-04	1.54%	2,170.36	79.68%	0.00%	5.94%	-3.96%	-	-	1,359.96	1,439.924
63	Dec-04	2.02%	2,214.20	81.70%	0.00%	5.94%	-3.96%	-	-	1,359.96	1,439.924
64	Jan-05	1.71%	2,252.06	83.41%	0.00%	5.94%	-3.96%	-	-	1,359.96	1,439.924
65	Feb-05	0.91%	2,272.55	84.32%	0.00%	5.94%	-3.96%	-	-	1,359.96	1,439.924
66	Mar-05	4.18%	2,367.77	89.51%	0.00%	5.94%	-3.96%	-	-	1,359.96	1,439.924
67	Apr-05	-2.50%	2,308.58	86.01%	2.50%	5.94%	-3.96%	0.06%	-	1,359.96	1,439.924
68	May-05	-0.61%	2,284.50	85.40%	3.11%	5.94%	-3.96%	0.00%	-	1,359.96	1,439.924
69	Jun-05	0.45%	2,304.82	85.85%	2.66%	5.94%	-3.96%	-	-	1,359.96	1,439.924
70	Jul-05	3.45%	2,383.18	89.25%	0.00%	5.94%	-3.96%	-	-	1,359.96	1,439.924

Figure 2.2: An example of a spreadsheet used for statistical calculations

Finally, the user needs to input the formulas of the statistical calculations. These formulas may be confusing and unintuitive to insert into the spreadsheet, which makes

this process error-prone.

Moreover, there is no way of verifying if the results of the calculations are correct, let alone pinpointing the source of the error in case the results are wrong.

As data changes constantly, the calculations become outdated, meaning that the entire data preparation process has to be redone on a regular basis to portray the latest image of the stock exchange market.

Spreadsheets can only handle a certain amount of data, however, in order to carry out certain calculations thousands of values need to be taken into consideration. Therefore even a powerful tool such as Excel cannot handle some calculations.

There are also plug-in tools that are used on top of Excel to automate the calculation process. Yet, these tools do not automate the data gathering and preparation process.

Chapter 3

Solution

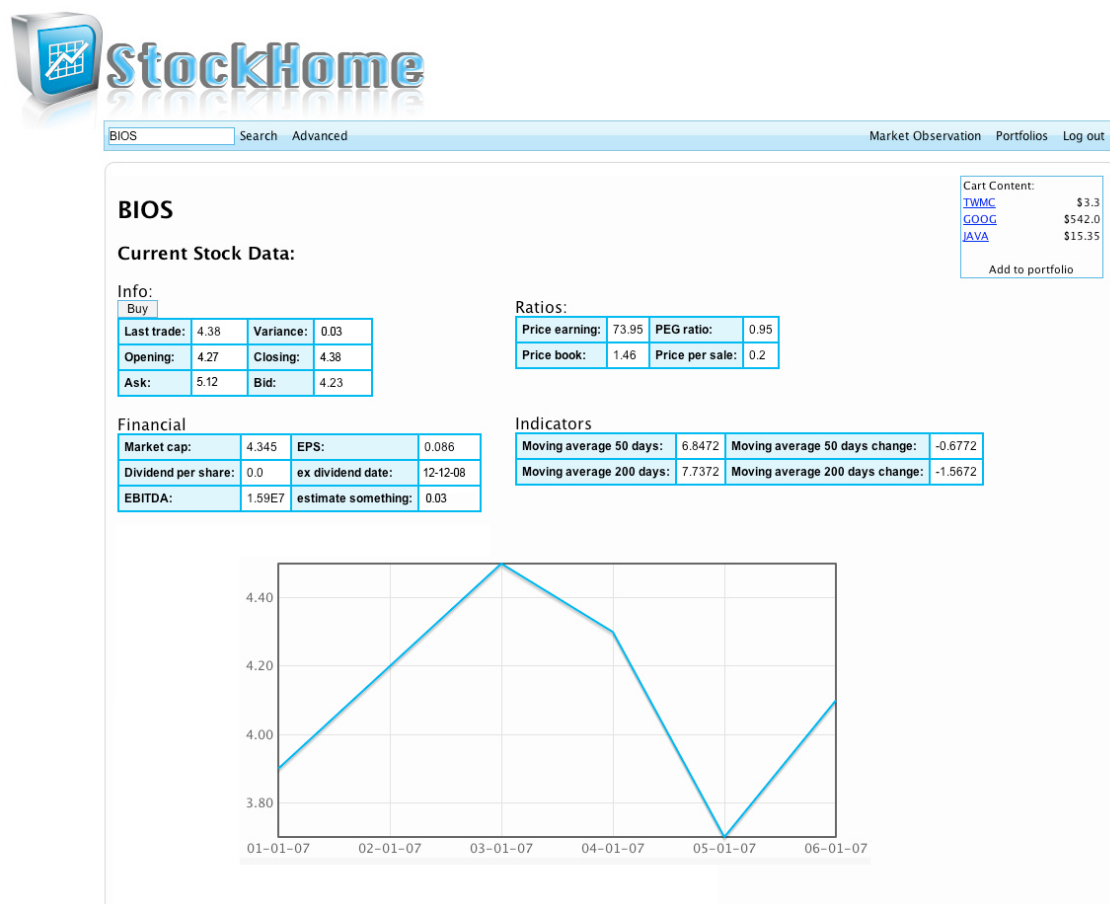


Figure 3.1: A screenshot of the single stock view interface

In the previous chapter we have introduced the problems of the stock exchange analysis tools currently available in the market. In this chapter we describe how we tackle these problems in our project.

We decided to implement our solution as a web application in order to avoid the dependencies on other software and/or the operating system. By doing so, we have removed the whole installation phase which is required in order to run the application. All one needs is a web-browser and an internet connection. This means that the application can be executed from any computer anywhere in the world.

We have chosen to use an Apache Tomcat as the base for our application. Tomcat runs the Java Enterprise Edition platform. The enterprise edition version of Java has the functionality of the standard edition with additional libraries to support web applications. The HTML markup language was originally designed for publishing static web pages, and is therefore insufficient for our needs (creating and maintaining a static system is inefficient). Thus, alongside Tomcat, we have chosen the JavaServer Pages (JSP) technology to create our dynamic web contents.

We have decided to implement the system using the Model-View-Controller (MVC) design pattern [Mic02]. This design pattern separates the business logic from the user interface. This design pattern enables the appearance and behavior to be completely independent of each other. In other words, by doing so we can change the appearance without affecting any of the functionality.

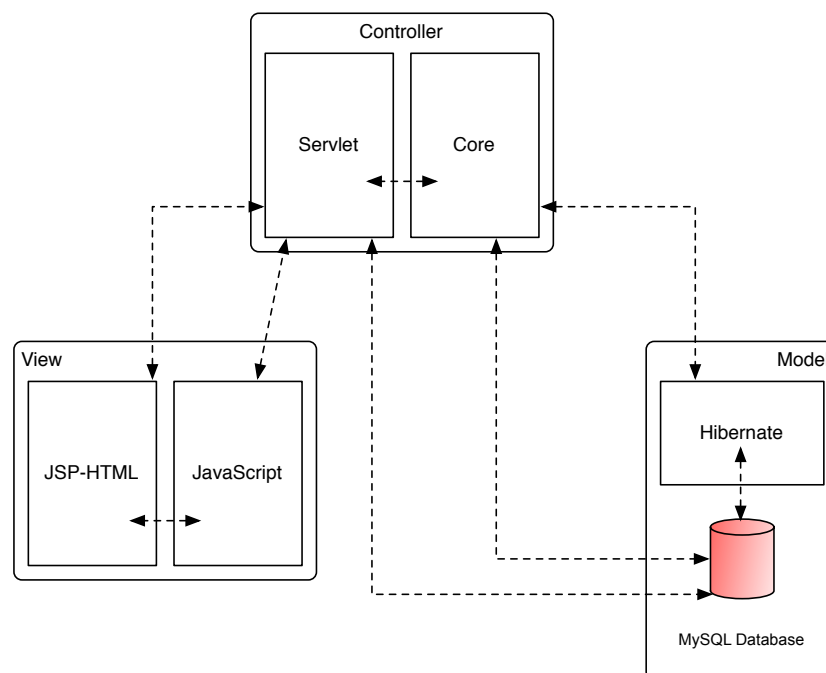


Figure 3.2: Model-View-Controller

An application applying the MVC design pattern is composed of three components. The model part of the application handles the database. The controller part of the application receives requests from the user and interacts with the model to get information which it then uses to generate the view. The view represents the user interface of the

application.

Figure 3.2 illustrates the control flow of such an application. In our case, the view part represents the browser on the client side where the JSP is interpreted and rendered. The view may interact with the controller which is found on the server side by either submitting HTML forms via HTTP requests or by sending AJAX requests using javascript. The controller receives these requests accordingly and calls the appropriate function in the core to prepare the response JSP. This function may interact directly with the database in the model using SQL or by using an abstraction persistence layer called Hibernate. Once the response JSP is complete, the servlet sends it back to the view, where the page or a section of a page is updated using the response JSP.

Figure 3.3 shows an overview of the solution we came up with. In this figure there are a few main groups, namely Tomcat server, Browser, back-end framework, and front-end framework. The tomcat server is where all the business logic and the database reside. The server is split into 3 groups The database, the front-end framework and the back-end framework.

The back-end framework has been implemented by Francesco Rigotti [Rig08]. The user interface, which consists of both the browser functionality and the front-end framework .

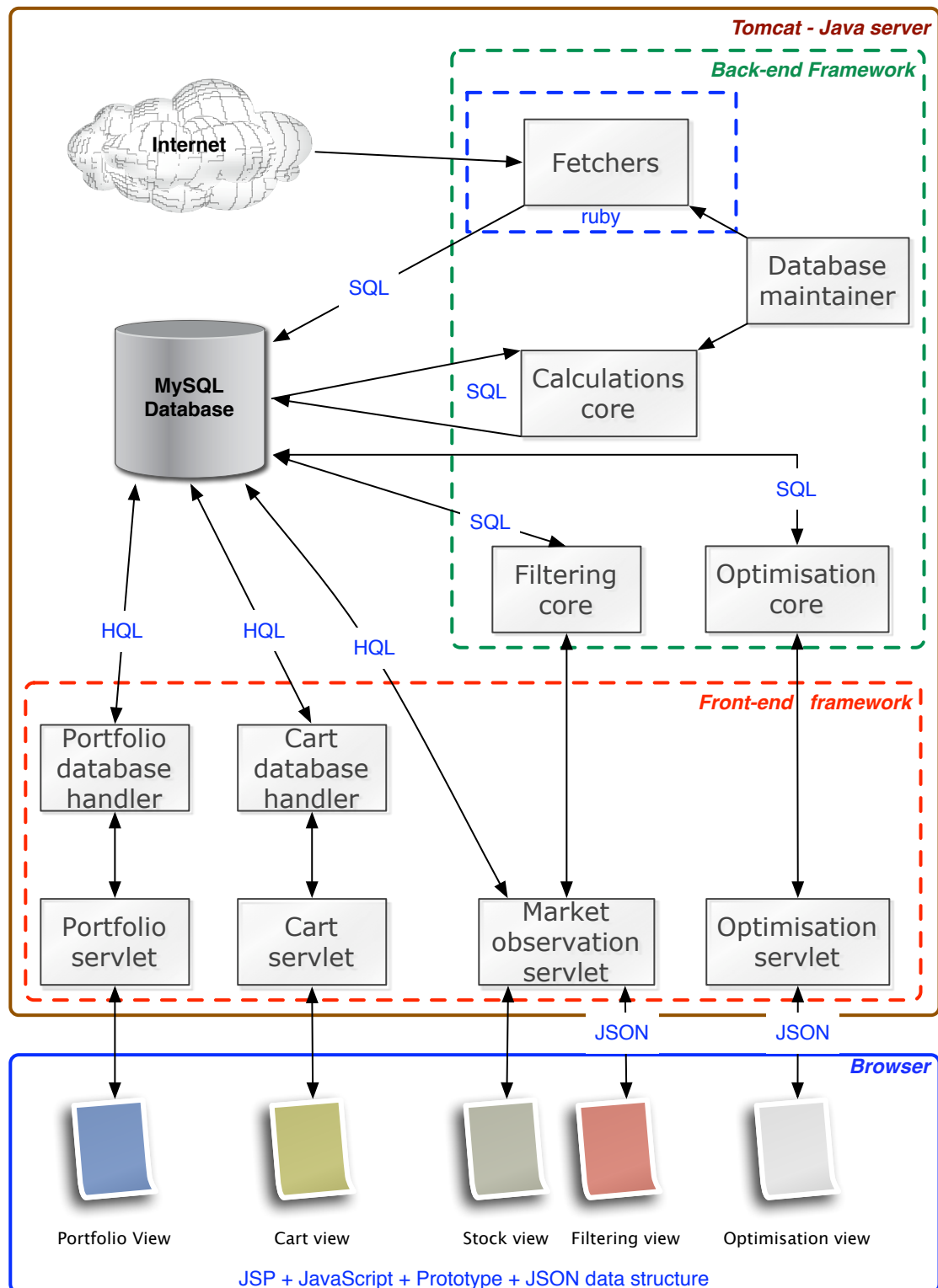


Figure 3.3: System overview

3.1 Back-end framework

As mentioned before, this part has been implemented by Francesco Rigotti. The back-end framework is in charge of carrying out 4 main features: data fetching, statistical calculations filtering, optimisation core.

3.1.1 Data fetching

We have created a series of 'web crawlers' implemented in ruby that connect Yahoo! finance and fetch all the required data in order to show how the stocks behave and to carry out the statistical calculations.

These crawlers are split into 2 groups:

1. Setup fetchers - fetch all historical data. These are used in the initial setup stage where the database is populated.
2. Maintenance fetchers - fetch the latest data in order to keep the database up-to-date. The execution of these fetchers is scheduled by the database maintainer package.

By adding this feature we have tackled the problem of the user having to fetch the information by himself in order to carry out calculations.

3.1.2 Statistical calculation core

We implemented a series of calculators to perform different statistical calculations and store the results in the database. These calculations are done on a regular basis and therefore keep the data in the database up to date. By doing so, the user no longer needs to manipulate any raw data in order to get the results he desires.

3.1.3 Filtering

In order to help the users choose what to invest in, we offer a filtering feature that allows the user to specify a certain stock description and find all stocks that match that description. This feature helps the user because, as mentioned, the stock exchange market is enormous and there are many stocks to choose from, therefore this feature helps the user narrow down and select only the stocks he is interested in.

Since a lot of data is sent to the servlet, we have decided to attach a JSON data structure object to the request (Section A.9). The conversion to and from JSON objects is done in the browser with the use of Prototype (Section A.7), and in the servlet with the use of Java. This makes it cleaner and easier to carry out the filtering, because by using the JSON object we can persist with the same naming convention on both the client and server side.

3.1.4 Optimisation core

In order to help the user choose how to invest, we have introduced the optimisations feature. This feature is similar to the calculators features only that instead of being done in the background, it is done on the user's demand. An example of such optimisation is 'style analysis', where the user specifies which stocks he is interested investing in and the style analysis optimisation tells him how much of each stock to buy.

3.1.5 Summary

By providing the back-end framework, the user no longer needs to fetch the information and manipulate it. All one needs to do is specify a stock or a description of a stock and the website handles all the rest. Due to the fact that the data is handled "behind the scenes", the user can be reassured that the data he receives is safe as we have verified the calculations with financial experts. The information provided to the user is also up-to-date as the web application maintains itself by fetching all new incoming data and carrying out the calculations with the new values.

3.2 Database

Figure 3.4 represents the database concerned only with the front-end framework. Its use will be described in the front-end framework section.

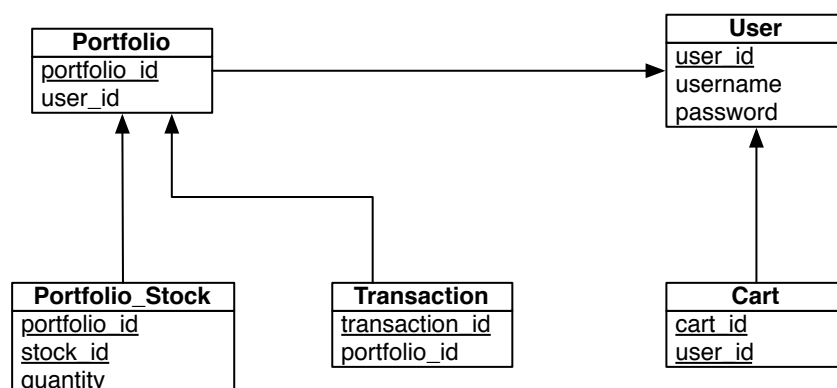


Figure 3.4: Front-end framework database representation

3.3 Front-end framework

This part of the project is dedicated to handling incoming requests from the user and generating the appropriate response JSP by accessing the database.

3.3.1 Portfolio database handler

The portfolio database handler is where all the portfolio management functionality is located. This handler allows for creation and removal of portfolios in the database. A portfolio is an information container that holds stocks which belong to it and transactions that have been carried out while buying and selling stocks.

What happens when a user buys a stock is the following. The request is received by the portfolio servlet that calls the logging function with the input values from the request and the current balance of the portfolio. It then attempts to buy the stock. When finished the servlet generates the response view and sends it back to the browser.

By adding this feature, we allow the user to save his investment ideas and monitor their performance as the prices of stocks fluctuate. Furthermore, the logging feature makes the system more robust and fault tolerant. In case of a failure in the system or in carrying out the transaction, data can be controlled and verified and in case of need, restored.

3.3.2 Cart database handler

The cart database handler enables functionality similar to a shopping cart of an online-shop, where the user can place the merchandise (in this case, potential stocks the user would like to purchase) until he is ready to checkout. The user can add stocks to the cart from the single stock view and from the filtering results view.

Our checkout functionality integrates the optimisations from the back-end framework and lets the user call them on demand. By using AJAX requests several optimisations can be done until the user is satisfied with the results without having to reload the page or recreate the cart.

3.3.3 Authentication servlet

The authentication servlet offers a simple authentication functionality which takes a username and password and matches them on the database.

3.4 Browser

This part of the solution is in charge of interacting with the server and providing a dynamic interface. It is described in much more detail in the validation part where we show the user interface and show how it communicates with the server.

Chapter 4

Validation

In the previous chapter we have described the solution and its different components. Now we show you how to use them and explain why they are useful for the user. This will be done as a walkthrough through different use-case scenarios, showing the typical usage of the system.

This section contains many references to technologies which are very recent. We therefore recommend to read the Technologies appendix before reading this section in order to make this section more comprehensible.

Before going through the scenarios, we introduce the user interface and explain a few design decisions we made. We wanted to design the interface in the most straightforward way possible to make it as user-friendly as possible. Therefore, we have decided to go for a minimal design, containing only the bare minimum text and images to avoid confusion on the user's behalf. Another reason for having a minimal design is to minimize the loading time of each component, hence, improving the responsiveness of the program and the overall user experience.

The main idea was to put as much functionality as possible on one web page without overwhelming the user with information. We did so by grouping together components according to their functionality. For example, in Figure 4.1, we have grouped together the functionality in the following manner: market observation, cart and navigation bar. The market observation groups together all the stock searching functionality. It is where the user can either specify a stock's name or a stock description. The cart is part of the market observation view, it is where the user holds potential stocks before purchasing them. The navigation bar groups together links to the main components of the program.

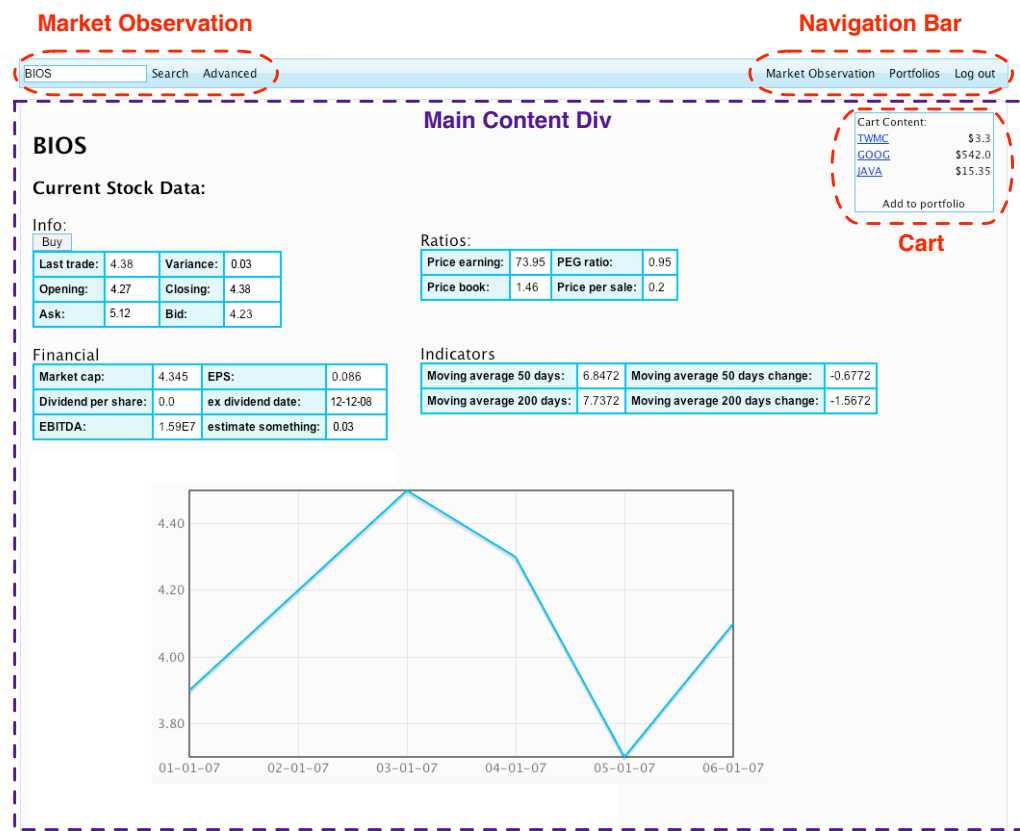


Figure 4.1: Main components of the user interface

The bar holding the market observation and navigation functionality is fixed no matter what the current view is. All the rest of the website is loaded dynamically according to the user's needs. This is done by sending AJAX requests to the server and then updating the appropriate div within the "Main Content" div. A 'div' is an HTML element offer a generic mechanism for adding structure to HTML pages. We have decided to group these functionalities there, in order to improve usability, as they are the most frequently used functionalities. More specific examples of how the divs are updated is given during the use-case scenarios.

Figure 4.2 is an overview of the data flow of the application. Each section will be describe more thoroughly in the following sections.

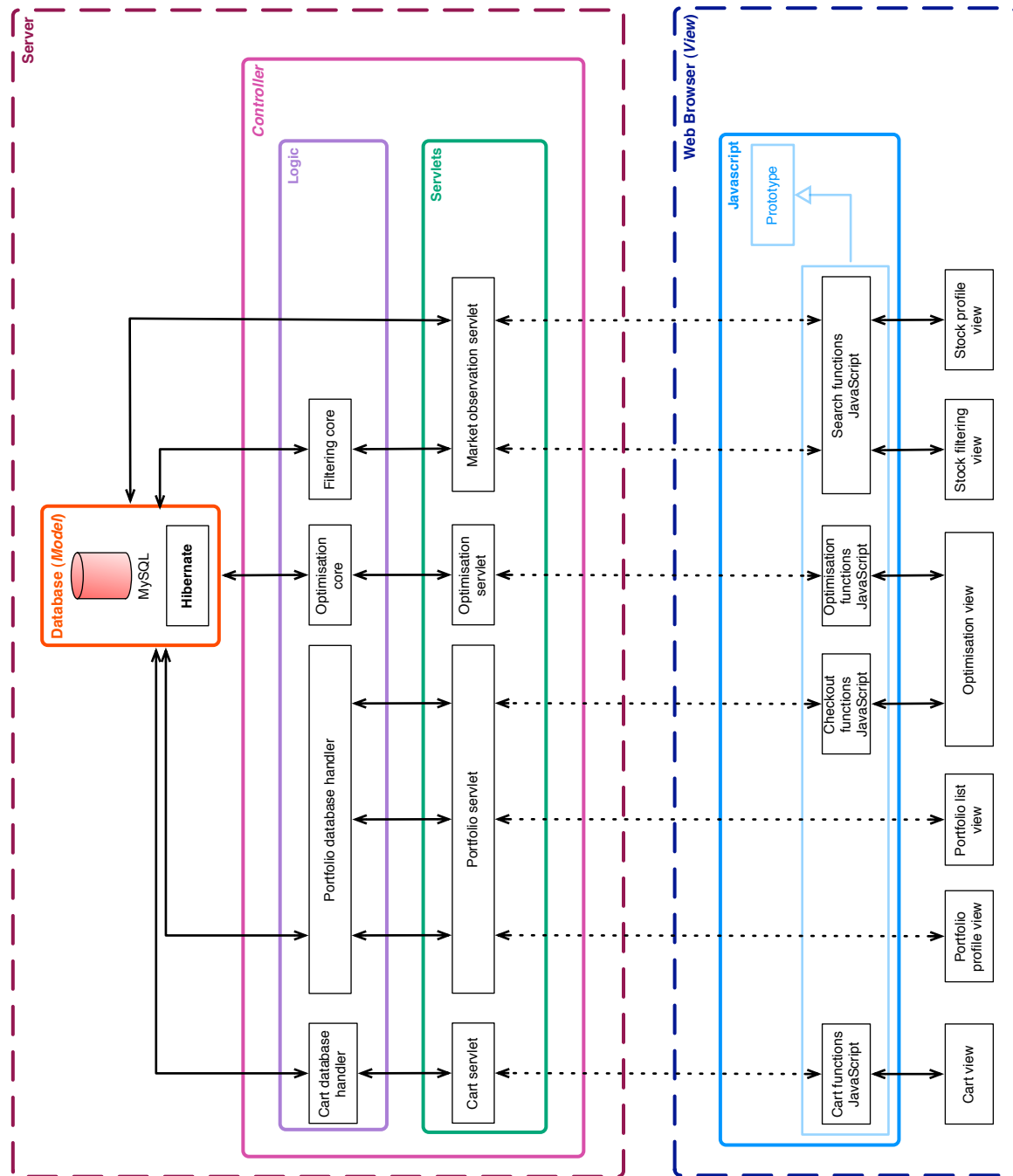


Figure 4.2: Data flow using MVC

4.1 Getting started with StockHome

Our user, Bob, has just won the lottery and would like to try out his chances with the stock exchange market. He decided to use StockHome to get some investment ideas. Bob has average computer and financial knowledge. We show the essential actions Bob carries out as he is interacting with the user interface.

Since our tool is a web-site, Bob does not need to download or install anything. He simply opens up his favorite web-browser and types in the address of StockHome. He is then prompted to log in to his account as shown in Figure 4.3. By logging into his account he can access his account portfolios.

When he logs in, the username and password are sent to the authentication servlet via an HTTP form using the HTTP POST method. The HTTP POST method is a way of submitting information to the server. These values are then matched with the username and password from the database. If the match is successful, the servlet puts the user id on the session so that personal data will be shown where possible. Otherwise, the user is requested to retry to log in.

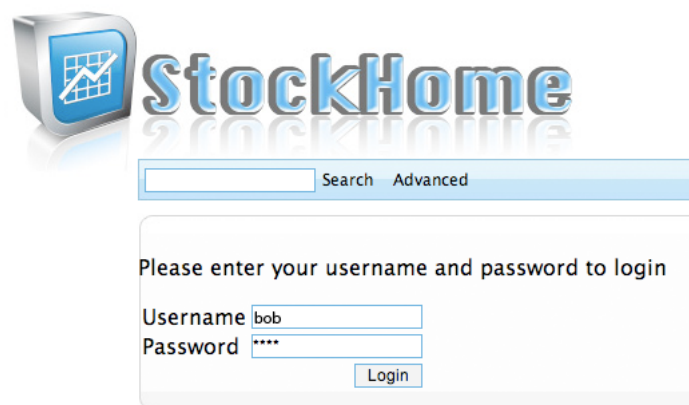


Figure 4.3: Screenshot of the login interface

4.2 Portfolio Management

In order for Bob to save his investment ideas, he needs to create a portfolio. Therefore, the first feature that the user needs to use is the virtual portfolio manager. Each portfolio has a name, a budget and a collection of stock names and their respective quantities.

To get to the portfolios view, Bob presses on the "Portfolios" link from the navigation bar. This updates the main content div and updates it with the portfolio list JSP file, which is generated with the user id to retrieve the appropriate portfolios owned by the user. This interface is split into 2 parts; a list of portfolios owned by the user and add a new portfolio. Below the portfolios list, the user can add a new portfolio to his collection.

Your portfolios

Portfolio name	Budget	
second	6781.56	<input type="button" value="Manage portfolio"/> <input type="button" value="Delete"/>
first	8817.1	<input type="button" value="Manage portfolio"/> <input type="button" value="Delete"/>
simo	462.86	<input type="button" value="Manage portfolio"/> <input type="button" value="Delete"/>

Portfolio name:
 Portfolio Budget:

Figure 4.4: Screenshot of the portfolio list interface

Bob would like to create a new portfolio. He therefore fills in the name and budget fields below the portfolios list, and presses on 'create portfolio'. By clicking the 'create portfolio' button, Bob submits POST request containing the filled in fields. On the server side, the portfolio servlet takes the input values and inserts a new portfolio into the database. After the database finishes adding the new portfolio successfully, it signals the servlet. The servlet then refreshes the page with the new portfolio list. This is illustrated in the UML-sequence diagram presented in Figure 4.5:

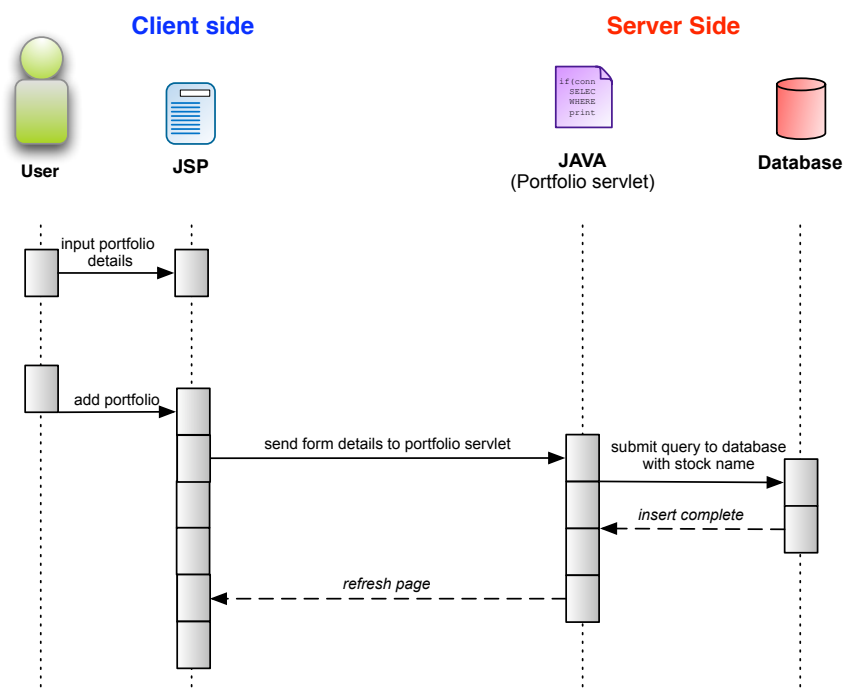


Figure 4.5: Use case scenario where a user adds a new portfolio

Throughout the rest of the portfolio interface we have used HTTP POST requests to communicate with the portfolio servlet. In order to avoid repetition, we assume a similar process happens when Bob clicks the 'Delete' button.

If Bob wants to inspect the contents of a portfolio, he presses on the 'Manage portfolio' button on the right of the portfolio name. This sends an HTTP POST request to the servlet, which then refreshes the page with the appropriate portfolio information.

Figure 4.6 is an example of a page that would be loaded after Bob clicks on the 'Manage portfolio' button:

Portfolio Management

Portfolio: second

Portfolio Budget: 6781.56

ZVUE	<input type="text" value="100"/>	<input type="button" value="Update shares"/>	<input type="button" value="Sell all shares"/>
ACME	<input type="text" value="10"/>	<input type="button" value="Update shares"/>	<input type="button" value="Sell all shares"/>
BIOS	<input type="text" value="132"/>	<input type="button" value="Update shares"/>	<input type="button" value="Sell all shares"/>
AMEN	<input type="text" value="57"/>	<input type="button" value="Update shares"/>	<input type="button" value="Sell all shares"/>

Stock name:

Amount:

Figure 4.6: Screenshot of the portfolio detail interface

This page contains the portfolio name and budget, and a list of the stocks currently owned in this portfolio. The list of stocks allows shows the names of the stock and next to it the quantity owned in this portfolio followed by the 'Update shares' and 'Sell all shares' buttons. The box containing the quantity of each stock is actually editable.

Therefore, if Bob wanted to buy or sell some of his currently owned stocks, he would put in the value accordingly and click 'Update shares'.

- In the case where the final value is smaller than the initial value, then the right amount of stocks is sold at Bid price and the budget is updated accordingly
- In the case where the final value is bigger than the initial value, then the right amount of stocks is bought at Ask price and the budget is updated accordingly. If the total price of a certain stock exceeds the portfolio's budget, then the maximum amount of that stock affordable is bought.

The 'Sell all shares' button sells all the shares of a certain stock.

Right below the stock list there is the functionality to buy a new stock and add it directly to the portfolio. Let us assume that Bob would like to buy some Google stocks. Bob inputs the name GOOG and specifies the amount of stocks he would like. He then clicks the 'Buy Stock' button and submits an HTML form to the portfolio servlet. The stock is added to the portfolio in the database by the servlet and then the page is refreshed with the new information. As Bob types in the name of the stock, an autocompletion suggestion box appears, letting Bob choose from a list of stocks that resemble the name that he has already input. This effect will be discussed in the market observation section.

The 'done with management' button takes Bob back to the portfolios list page.

4.3 Market observation

The market observation is the core feature of this application, because it is where most of the functionality is found and where the user spends most of his time. Therefore we attempted to maximize the usability for this interface.

The market observation allows the user to search for single stock details, or filter a list of stocks according to a stock description. Another feature of the market observation is the cart.

4.3.1 Searching for a single stock

After Bob has set up a new portfolio, he decides to review the financial information of some stocks he is interested in. He therefore starts typing the name of the stocks in the search bar. As he types in the name of the stock, the autocompletion suggestion box appears. This autocompletion suggestion box contains the names of stocks and their company's name. This is done with the use AJAX and the help of Prototype (Section A.7) and scriptaculous (Section A.7.1).



Figure 4.7: Autocompletion suggestions for stock name

This feature improves the usability because many stock names are similar and/or unintuitive for the user to remember. By offering the autocompletion suggestion box, the user can save time finding the stock he was interested in.

What happens "behind the scenes" to provide the user with the suggestion box is the following. The user types in a letter which triggers an AJAX request to the server using Prototype. In the server, the market observation servlet receives the AJAX request and uses hibernate to query all stock names that have either their name or their company's name that resembles the input string. The database returns a hibernate (Section A.3) object which is then interpreted and then generates a JSP response. The response JSP is sent back to Prototype. Prototype then updates a hidden div with the response JSP. This is illustrated in the UML sequence diagram in Figure 4.8.

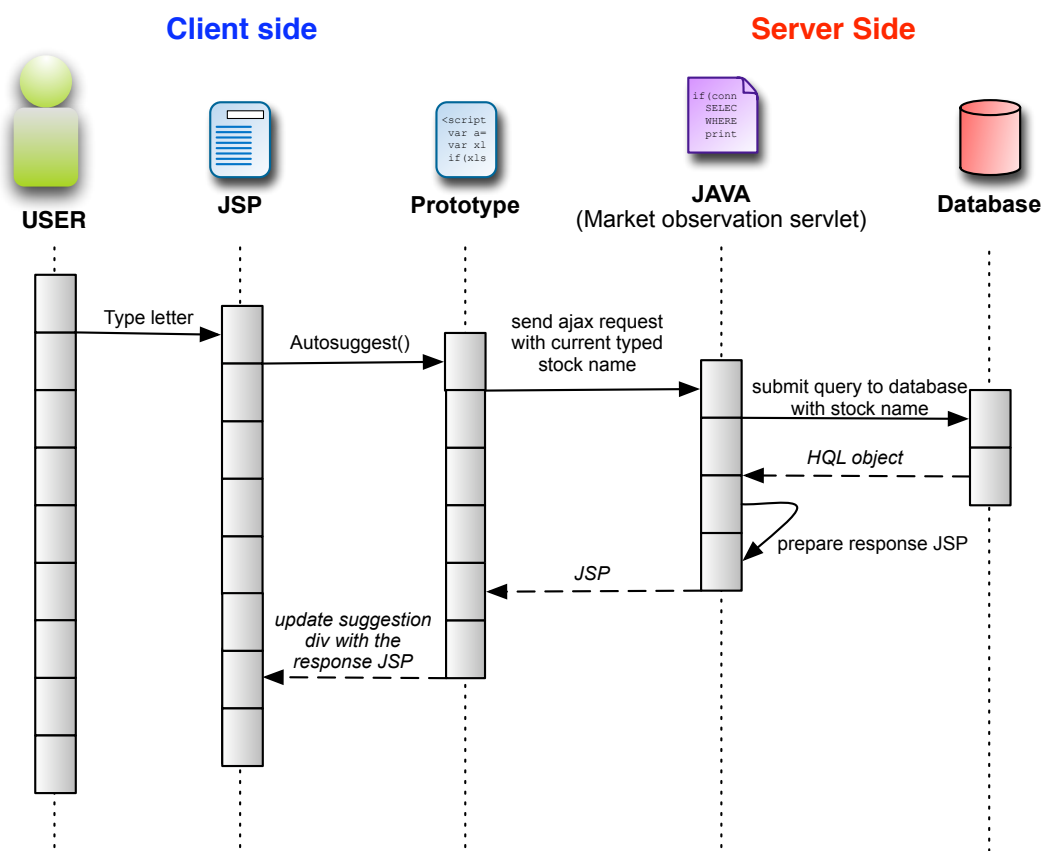


Figure 4.8: Use case scenario where user chooses a stock and receives possible stock suggestions

Bob then chooses the stock he is interested in and clicks on 'search'. This causes the stock's financial information to appear. To improve the usability, we have grouped together the different financial figures according to their nature. We have also added a graph that illustrates the evolution of the stock price over a period of time. Figure 4.9 is an example of a typical stock detail page.

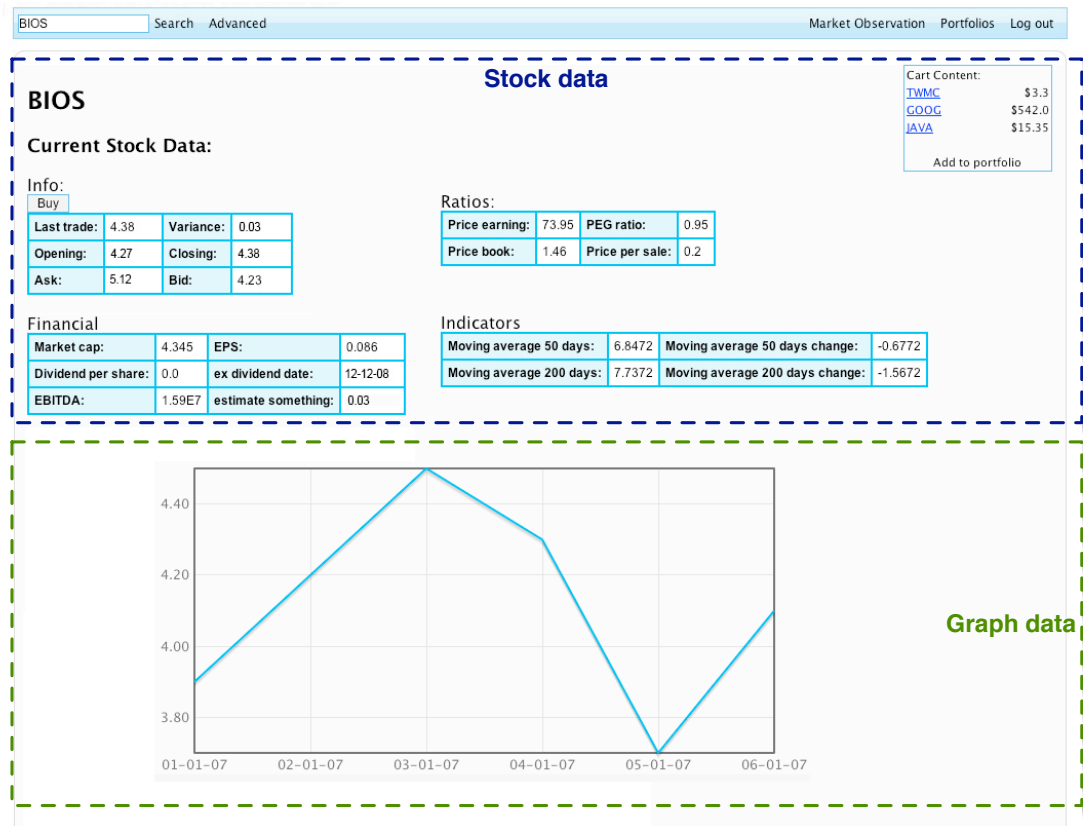


Figure 4.9: Screenshot of the main interface

When Bob clicks on the 'search' button, he triggers an AJAX request containing the stock name. The market observation servlet receives the request on the server side and uses hibernate to query for the stocks data. The resulting hibernate object is then used to generate the response JSP containing the tables. This JSP is then sent back to Prototype and the stock data div which is inside the main content div is updated with the response JSP. Once the response JSP is successfully loaded, another AJAX request is sent to the market observation servlet requesting for the graph of the stock. The servlet then requests for the appropriate data from the database and builds a flot (Section A.7.2) graph object. This object is sent back to Prototype and then rendered in the graph div.

We decided to load the page in 2 parts is to improve the responsiveness of the page, because fetching and showing the financial data is a rather light task, whilst the preparation of the graph and its rendering requires more time. Therefore, by loading the page in 2 parts lets the user read the financial information as the graph loads up. This is represented in Figure 4.10.

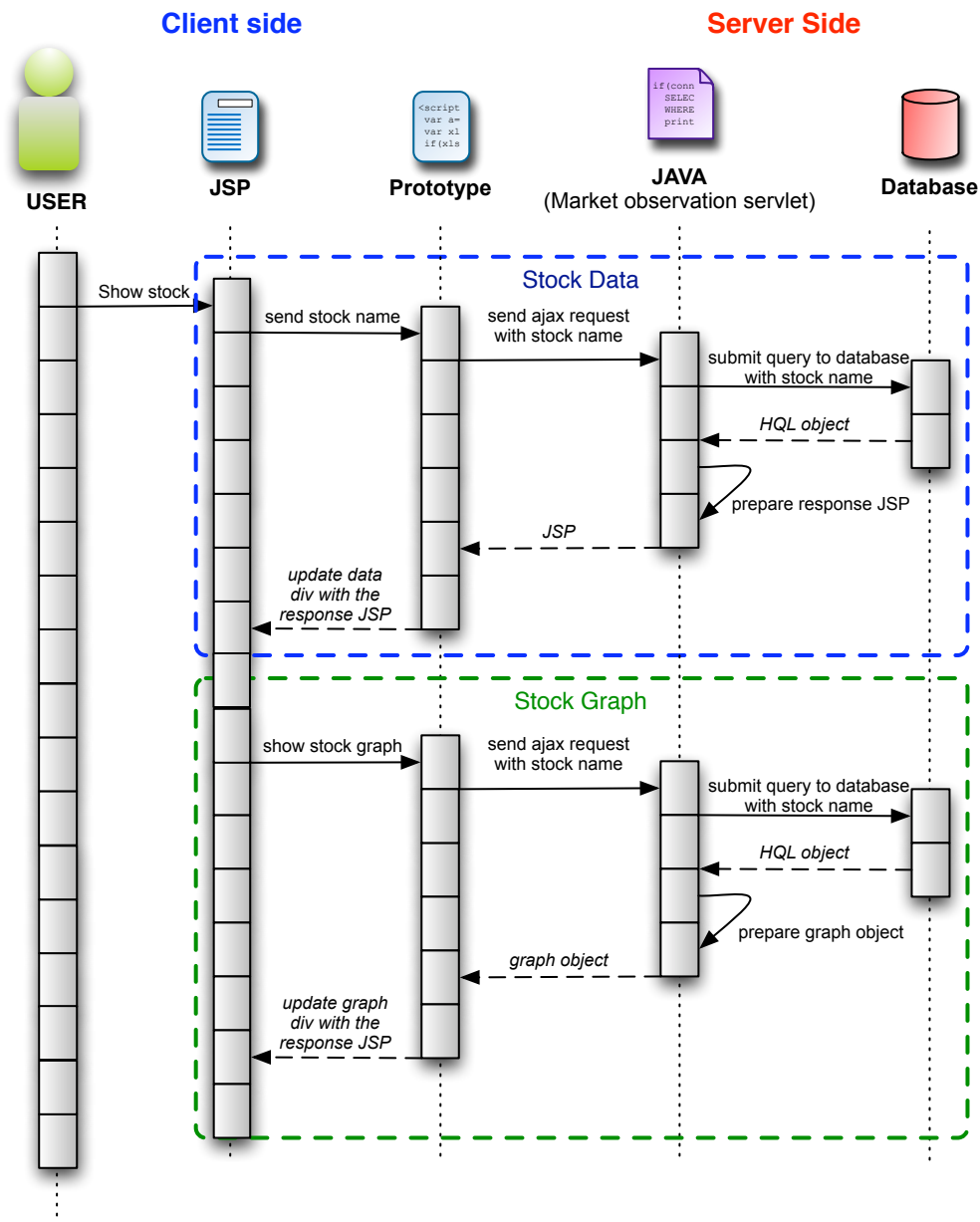


Figure 4.10: Use case scenario where user asks for data of a specific stock

After examining the stock he is interested in, Bob clicks on the 'buy stock' button. This adds the stock to the cart. More about the cart will be explained later on in the cart section.

4.3.2 Filtering stocks

Bob wants now to specify a stock description, he therefore presses on the 'advanced' button. This makes the advanced search box to appear in the interface. Bob then enters the values to match his stock description, and clicks 'search stocks'.

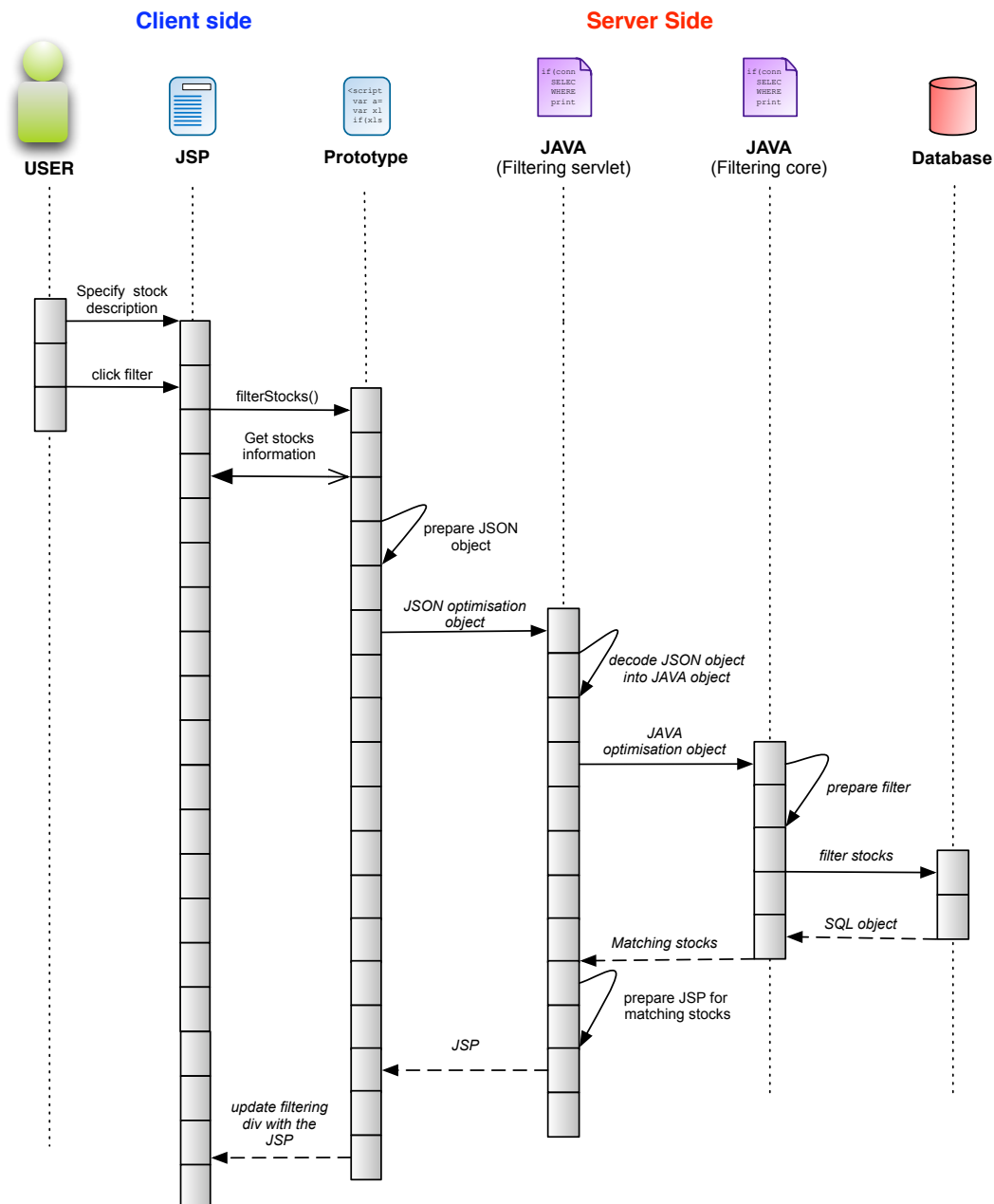


Figure 4.11: Use case scenario where user specifies stock description and filters all stocks that match this description

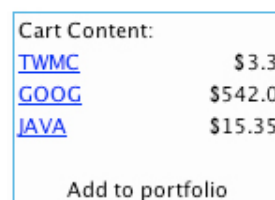
The resulting list of stocks matching the stock description (if any exist) appear on the main content div. The stocks are represented in a sortable table. If Bob wants to

sort the stocks by a certain criteria, he clicks on the criteria's name and the table is then sorted accordingly. In order to buy stocks Bob ticks the tick boxes next to the stock names he desires then clicks on the 'buy stocks' button which adds those stocks to the cart.

Figure 4.11 illustrates what happens when Bob clicks on the 'search stocks' button. Bob calls a JavaScript function, which assembles the various specified stock information and forms a JSON object. This JSON object is then sent to the market observation servlet, where it is decoded into the equivalent java object. This java object is then used by the filtering core to formulate the appropriate database query in SQL form. The database returns a list of matching stocks, which are then used to generate the response JSP. The response JSP is used to update the main content div.

4.3.3 Cart

The cart is an intermediate container of stocks. The user may use the cart when he has found a stock which he would like to buy. The Cart contains the list of potential stocks to purchase, as well as their price.

A screenshot of a web application's cart interface. It features a table with two columns: stock names (TWMC, GOOG, JAVA) and their prices (\$3.3, \$542.0, \$15.35). The stock names are hyperlinks. Below the table is a button labeled 'Add to portfolio'.

Cart Content:	
TWMC	\$3.3
GOOG	\$542.0
JAVA	\$15.35
Add to portfolio	

Figure 4.12: Screenshot of a cart containing some stocks

In order to view the financial data of a certain stock contained in the cart, Bob presses on the stock's name and the stock information is displayed accordingly using the same AJAX call used in the single stock search. In order to add the selected stocks into a certain portfolio, Bob presses on the 'Add to portfolio' button that makes the optimisation interface to show.

4.3.4 Optimisation

When Bob clicks the 'Add to portfolio' button on the cart, the optimisation interface appears. When the optimisation interface appears, the background becomes darker and on top of it pops a window with all the portfolios list, a list of stocks and their amounts and a few optimisation buttons. We have decided to give the optimisation interface a dark background around it, in order to separate it from the rest of the interface. We wanted to focus all the users' attention on the optimisation or manual allocation of funds. Figure 4.13 is an example screenshot of the optimisation interface.

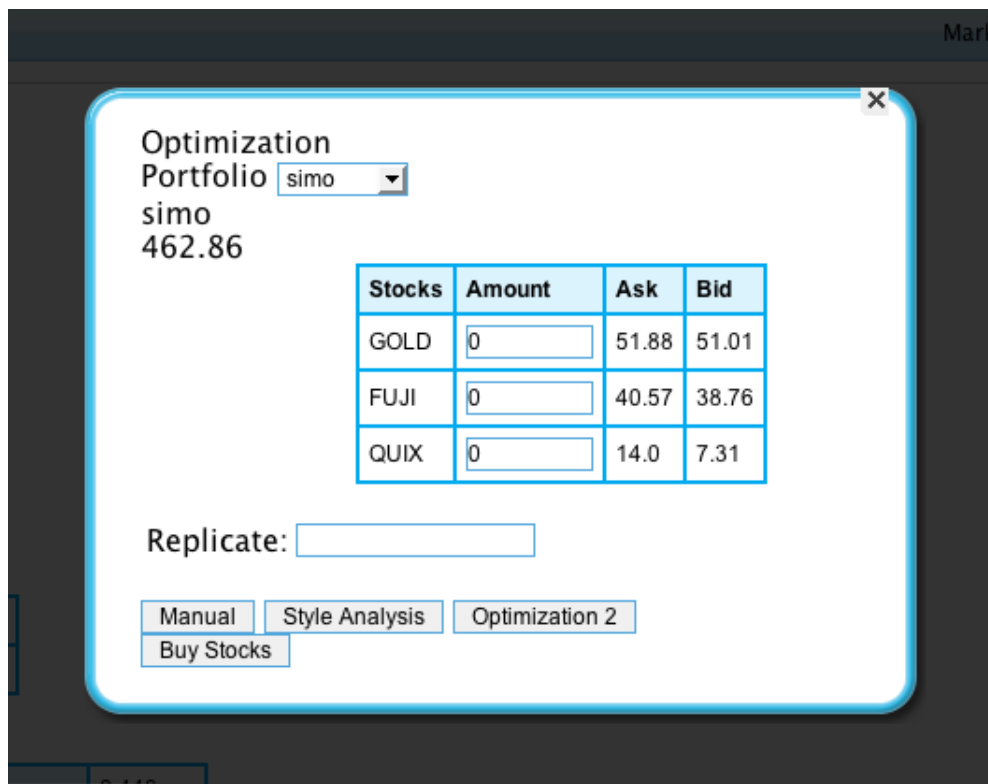


Figure 4.13: A screenshot of the optimisation interface

The optimisation interface has a few functionalities. It is where the user decides how much of the available budget to use for the transaction and how much of that amount to allocate to which stock.

We also offer a "Style analysis" optimisation feature. What this feature does is, it allocates the amount of each of the stocks in order to replicate the behavior of another stock. For example, if Bob would like to invest in a google stock but the price of the stock is too high, he might want to apply this optimisation on a bunch of cheaper stocks in order to imitate the google's stock performance.

This feature offers to calculate a statistical calculation on the fly, such a calculation would take much longer if it was to be carried out by hand. Therefore, this feature make it easier and faster for the user to make decisions.

In order to apply this optimisation, Bob chooses the portfolio he would like to apply the optimisation on. Then he enters the name of the stock in the 'Replicate' field, which has the autocomplete suggestion box and clicks on the 'Style analysis' button to apply the optimisation. This calls a JavaScript function that assembles together the information needed to apply the optimisation in a form of a JSON object. This JSON object is then sent as an AJAX request containing the JSON object. The optimisation servlet transforms the JSON objects into the equivalent Java object and passes it to the optimisation core. In the optimisation core the object is used to create a query to the database and then carry out calculations on the data returned from the query. The re-

sults are transformed into a JSON object in the servlet and then sent back to Prototype. Prototype then updates the values with the optimisation results. This is illustrated in the diagram below.

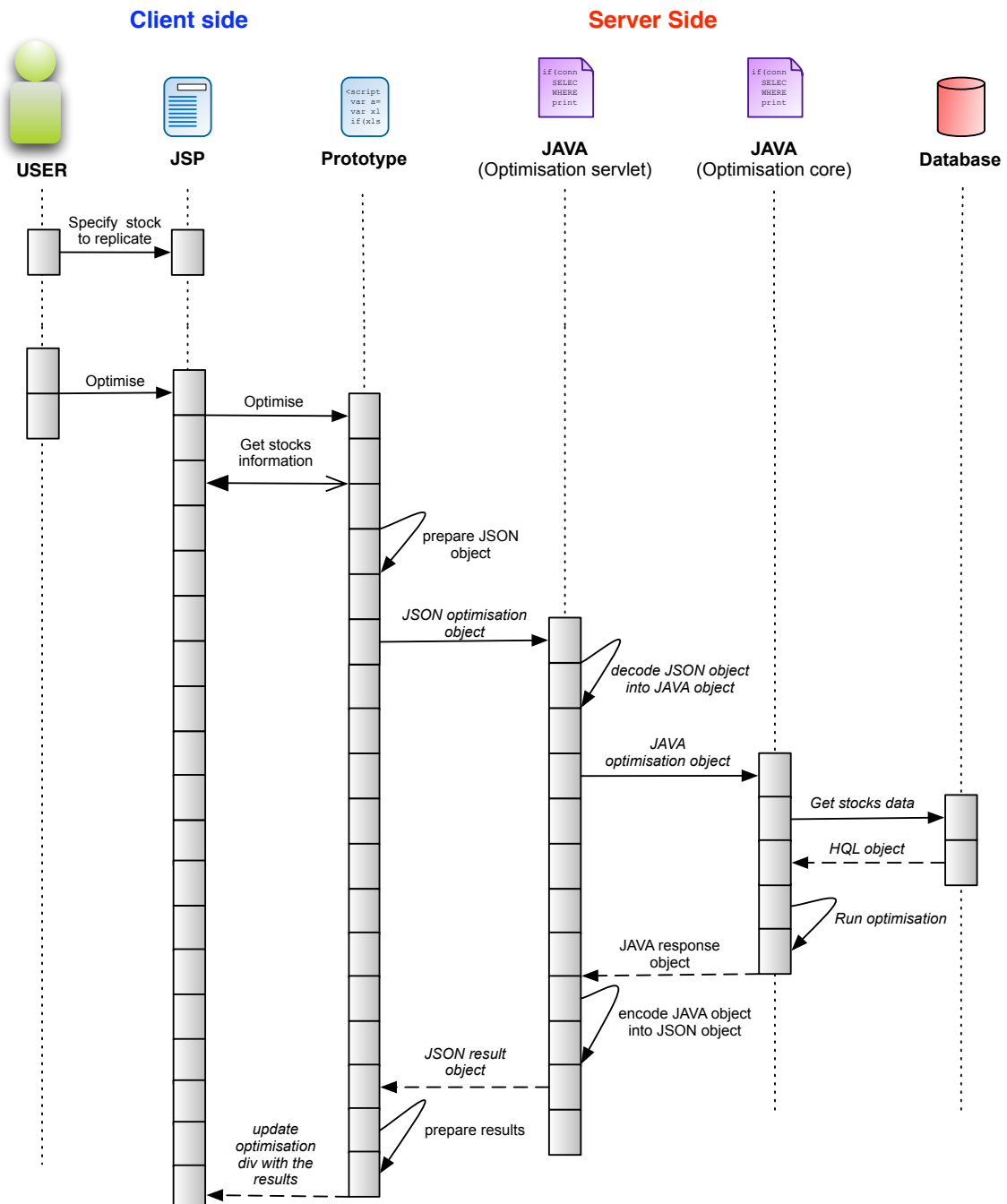


Figure 4.14: Use case scenario where user optimises his chosen stocks to replicate a certain stock behavior

Now that Bob has applied the optimisation on the stocks, he proceeds to the last feature of the optimisation interface. This feature is the 'checkout' feature. What this function does is use the buy stock function from the Portfolio servlet for each stock and its corresponding amount. To do so, Bob clicks on the 'Buy stocks' button.

Alternatively, Bob can just apply the amounts manually, as long as they are within budget, and press the 'buy stocks' button.

Chapter 5

Conclusions

In previous sections we have introduced the various problems related to the current financial analysis tools, provided a suitable solution that attempt to resolve these problems and showed how our solution resolves them.

In the beginning of the document we have mentioned that stock exchange markets produce a large amount of data which is constantly changing. Due to this fact, it is hard to examine all stocks one by one without the use of any tools, because there is simply too much information to handle and also by the time one would finish examining all the stock information in the world it will be already outdated.

To facilitate this task, analysts and investors use various calculations and tools to assist them. These tools also present their own problems. The main issues with the tools are the following:

- User interface is unintuitive and difficult to use.
- The user has to collect and format the financial data by himself, which is time consuming and error prone.
- The user must manipulate the financial data in order to carry out information, which is time consuming and error prone.
- The user has to continuously fetch data and carry out calculations on it in order to stay up-to-date.

5.1 Goals achieved

One of the most important feature we have implemented is the data fetching scripts. Although it is not directly part of the user interface, it greatly improves the user experience. The user no longer has to fetch and format the information before carrying out the calculations, which saves him time and ensures that the data is collected correctly.

Another important feature is the calculation core, like the data fetching scripts, this feature is done behind the scenes and greatly improves the user experience. It carries out several statistical calculations in the background, meaning that the user no longer has to carry out the calculations by himself or even know the formulas. Again, like the

previous feature, this saves the user a great deal of time and ensures that the calculations are done correctly as they are done automatically, which removes the human error factor from the equation.

Also, the server schedules running the scripts and calculation core so that the information in our database is always up-to-date.

Our filtering feature helps the user find what he wants more easily by specifying a stock's description which is then matched with all the stocks in the database and returns only the ones the user is interested in. Given the large number of stocks there are in the world, this feature saves the user a lot of time.

Our solution also offers the user with advice on how to invest his money and carry out calculations on the fly. This is done in the optimisation feature where a user can request for a calculation, the server carries out the calculations and sends back the results. Again, this feature saves the user time and ensures that the calculations are done correctly.

We have used the newest and most advanced web technology when implementing the user interface. This was done in order to provide the user with a dynamic and highly interactive interface.

In a sense, our implementation offers a black-box abstraction to stock exchange markets, because the user doesn't need to care about how the calculations are done and just rely on the application to do it for him. In fact, by creating such an abstraction, the level of pre-required financial knowledge is diminished. This means that less statistically savvy people can take advantage of our tool.

5.2 Future Work

Although our solution tackles many problems, there is still a great deal of improvements that can be done. Here is a list of things that may be done to improve our solution even more.

- Currently, we have only verified the calculations with a financial expert. However, to ensure that the calculation are always correct, they can be verified by using unit testing.
- So far, we have only offered a way to facilitate the creation of portfolios. However, there is no way of verifying how the portfolio will perform. It is possible to add a feature that simulates the performance of a portfolio and indicate how promising the investment is.
- We have built the solution in such a way that additional statistical calculators can be easily added. In order to improve the user-experience, more calculators should be added
- We have built the solution in such a way that additional optimisations can be easily added. In order to improve the user-experience, optimisations should be added

Appendix A

Technologies

We used a wide variety of technologies in our project. We have chosen to do so in order to take advantage of the speciality of each technology.

Before discussing each technology, here is a diagram illustrating the technologies used in the different layers of the application.

- Ruby
- MySQL database
- Hibernate
- Java on top of Apache Tomcat server
- JSP
- JavaScript
- Prototype
- AJAX
- JSON

A.1 Ruby

Ruby¹ is a dynamic, interpreted, open source programming language with a focus on simplicity and productivity. It is a pure object-oriented programming language that combines the syntax inspired by Python and Perl with Smalltalk-like features.

We have chosen to write our fetcher scripts in Ruby due to its flexibility and simplicity.

A.2 MySQL database

MySQL² is an open source database software. It is fast, reliable and easy to use. It is compatible with all of the other technologies we use. Therefore it was a perfect choice for our design.

¹<http://www.ruby-lang.org/en/>

²<http://dev.mysql.com/>

A.3 Hibernate

Hibernate³ is a relational persistence service for Java. It lets you query for data from the database and manipulate the result in an object orientated fashion. We chose to use this technology because it provides a higher level of abstraction, hence, making queries and manipulating data from the database more seamless. We chose to use this technology only when creating the web content. Carrying out calculations on thousands of objects is extremely heavy on the server, so instead we use simple SQL for that purpose.

A.4 Apache Tomcat Java server

Apache Tomcat⁴ is a servlet container, which implements the Java Servlet and JavaServer Page (JSP) specifications. It provides a pure Java HTTP web server environment for Java code to run on.

We have many calculations to be run, hence, performance and thread safety are constraints. Java offers many libraries for mathematical operations, it is thread safe, and it is fast. For these reasons we have chosen to implement our system based on Tomcat.

A.5 JSP

JSP⁵ is a technology that provides a simple way to create dynamic web content used alongside the Java code run on Tomcat. JSP can import java classes and use their functionality when writing the page contents. In JSP the code is written as a mix of HTML code and Java code. This code is compiled upon page request with the current dynamic values and generates an HTML code to be rendered by the web browser. We chose to use this technology because it comes alongside Java and Tomcat.

A.6 JavaScript

JavaScript is a rather "ancient" technology, it was first introduced in the early 1990s, when the world wide web was created and all web pages were static. This meant that all pages, once loaded, could not be interacted with. In order to add this interactivity, a certain form of language had to be used to describe the actions. This is where JavaScript came in. JavaScript came built in with web browsers because of the need to interact with the web page without having to reload the page which requires sending a request to the server.

We decided to use JavaScript in our project because it is the backbone of client-side interactivity.

³<http://www.hibernate.org/>

⁴<http://tomcat.apache.org/>

⁵<http://java.sun.com/products/jsp/docs.html>

A.7 Prototype

The Prototype JavaScript framework is an open-source framework that extends JavaScript functionality and aims to ease the development of dynamic web applications. Prototype creates a "ruby-like" abstraction to JavaScript, making coding easier and more concise. It can be argued that Prototype is a mixture between a library and a framework, because on one hand, Prototype offers library convenience methods to facilitate the code which is written most frequently. On the other hand, Prototype offers a class-based inheritance system, something which is not offered by JavaScript.

An example of a library convenience method is the dollar method (`$()`). The dollar method is a shorthand for `getElementById()`,

```
document.getElementById("element_id")
```

and reduces it to:

```
$("element_id")
```

One of the most important feature that Prototype offers its simple XML object handling such as `XMLHttpRequest`. which is in essence, AJAX.

Before prototype was created, using JavaScript to handle `XMLHttpRequest` was a tricky task and would look something like this [Cou],

```
function ajax(url, vars, callbackFunction)
var xhr;

// Firefox, Opera, Safari, IE7
if (window.XMLHttpRequest)
    xhr = new XMLHttpRequest();

// IE 5-6
try { xhr = new ActiveXObject("Msxml2.XMLHTTP"); }
catch (e) {
    try { xhr = new ActiveXObject("Microsoft.XMLHTTP"); }
    catch (e) { throw 'Ajax not supported!' }
}
```

However with prototype the same function is reduced to this:

```
new Ajax.Request('/some/url');
```

AJAX will be discussed in more detail in the following section.

Prototype can be easily extended by additional libraries such as Scriptaculous and flotr to give the web interface even more functionality.

A.7.1 Scriptaculous

Scriptaculous⁶ is a "special effects" add-on library to Prototype. This library lets users move div around, appear/disappear, change color and much more. Scriptaculous provides a user-friendly API and its own documentation.

Scriptaculous is used extensively throughout our project. For instance, its used to toggle the filtering panel in and out. Moreover, Scriptaculous offers an autocompletion suggestion feature, which in our case is used to help the user find the name of the stock that he wants.

A.7.2 flotr

Flotr⁷ is another library that works on top of Prototype. The purpose of this library is to plot graphs and therefore enhance user's experience and usability. This library allows the user plot various types of graphs with various amounts of user interactivity.

A.8 AJAX

Responsiveness is a very crucial issue when designing and developing web-interfaces. One way of improving the responsiveness of a web-page is by using AJAX. AJAX stands for Asynchronous JavaScript and XML. This means data is requested from the server and loaded "behind the scene" without interfering with the display and behavior of the existing page.

In order to retrieve data using AJAX, JavaScript is used to send an XMLHttpRequest object to the server, which then carries out the requested task and returns an XMLHttpRequestResponse object. The XMLHttpRequestResponse is then evaluated and rendered accordingly, refreshing only the affected part of the page.

A.9 JSON

Prototype allows for objects to be serialized and sent over to the server, which then decodes them to an equivalent object in the technology used by the server. This serialized object is a JSON object. JSON is a lightweight data format and stands for JavaScript Object Notation⁸. The JSON notation is text based, which makes it easy for humans to read and write and for machines to parse. A JSON object is basically an associative array or in other words an unordered set of "name : value" pairs. Such an object begins with a left brace and ends with a right brace. The key is a string followed by a colon and then the value is one of possible data types provided by JSON (see figure A.3). The name : value pairs are separated by a comma.

⁶<http://script.aculo.us/>

⁷<http://solutoire.com/flotr/>

⁸<http://www.json.org/>

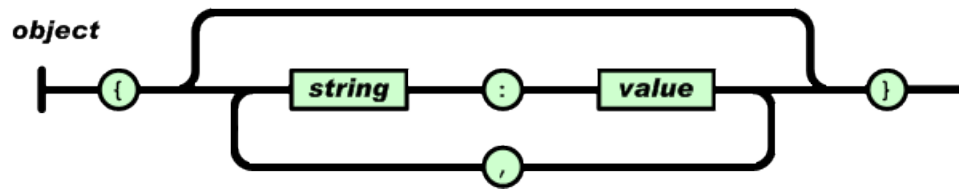


Figure A.1: JSON object data format

A JSON array works in a similar way. It begins with a left bracket ([), then possible values separated by a comma and ends with a right bracket (]).

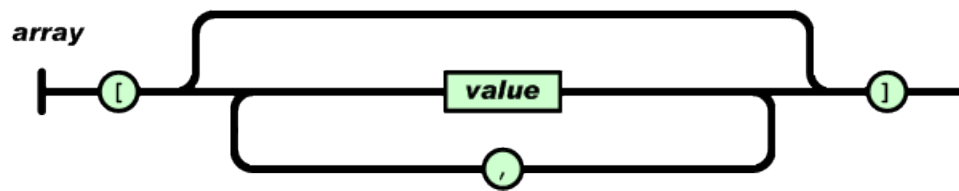


Figure A.2: JSON array data format

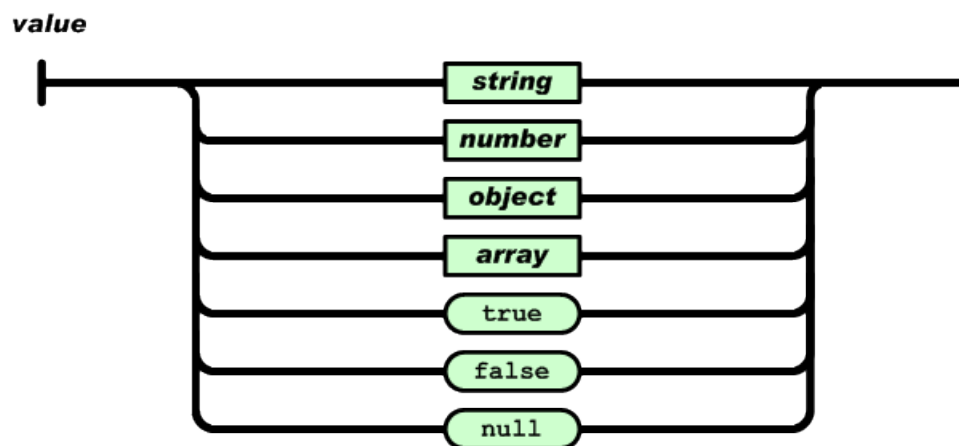


Figure A.3: JSON value data format

Here is a simple example of a JSON object declaration with prototype:

```
var data = {stockname: 'Google', statistics: [ 1, 2, 11, 5, 8 ], price: 25 };
var JSONObject = Object.toJSON(data);
```

List of Figures

2.1 Bloomberg user interface	4
2.2 An example of a spreadsheet used for statistical calculations	5
3.1 A screenshot of the single stock view interface	7
3.2 Model-View-Controller	8
3.3 System overview	10
3.4 Front-end framework database representation	12
4.1 Main components of the user interface	15
4.2 Data flow using MVC	16
4.3 Screenshot of the login interface	17
4.4 Screenshot of the portfolio list interface	18
4.5 Use case scenario where a user adds a new portfolio	18
4.6 Screenshot of the portfolio detail interface	19
4.7 Autocompletion suggestions for stock name	20
4.8 Use case scenario where user chooses a stock and receives possible stock suggestions	21
4.9 Screenshot of the main interface	22
4.10 Use case scenario where user asks for data of a specific stock	23
4.11 Use case scenario where user specifies stock description and filters all stocks that match this description	24
4.12 Screenshot of a cart containing some stocks	25
4.13 A screenshot of the optimisation interface	26
4.14 Use case scenario where user optimises his chosen stocks to replicate a certain stock behavior	27
A.1 JSON object data format	35
A.2 JSON array data format	35
A.3 JSON value data format	35

Bibliography

- [Cou] Prototype/Scriptaculous Crash Course. <http://www.refreshaustin.org/presentations/prototype-scriptaculous-crash-course/>.
- [Mic02] Sun Microsystems. *Sun ONE Architecture Guide*. <http://www.sun.com/software/sunone/docs/arch/>, 2002.
- [Rig08] Francesco Rigotti. *StockHome - Analytical Framework*. 2008.