NTNU – Trondheim
Norwegian University of
Science and Technology

IT2810 Web Development

Assignment 3: Main project

Kristian Skog
Cornelius Grieg Dahling
Christoffer Andre Nilsen
Joakim Lindgren
Martin Dørum
Eirik Fosse

November 13, 2016

Contents

1 Concept and Content

1.1 Introduction

This document is the documentation for the main group project in course *IT2810 Web Development* for group 08.

Chapter 1 describes the concept and the different pages and features of the application. Chapter 2 provides an installation guide for running the application locally, a file structure, and a description of the libraries we used. It also contains possible future functions. Chapter 3 explains how we fulfilled the requirements of the project.

1.2 Stockwatch

Stockwatch is your number one source for keeping track of stock values. Using data from Oslo Stock Exchange' API, stock values from all the listed businesses and funds in Norway are updated in real time. By creating your own profile, you can easily manage your equities and find new and exciting stocks all in one place.

1.3 Site content

1.3.1 Home

The home page is the main page of the site. This site displays a large graph of the latest marked activity/benchmark index, along with some additional information about use of the website.

1.3.2 Search

In the sub page Search we display all stocks and funds from the API we're using. The user can scroll or search to find the desired equity, or click to open a more detailed page about a single equity. The search result are able to ignore small margins of error (e.g five letters allows a margin of 1 wrong letter). We have chosen not to implement any sorting-functionality on this page, since the data set is so large that it makes more sense to just search in it.

The user can also click on an equity to display more information about it. This will also give the user the option of adding this equity to their personal portfolio.

1.3.3 Portfolio

This sub page displays equities that the user have added to their personal list. This way the user can easily keep track of the equities that are of interest. Equites can be sorted on all of its attributes, and it is also possible to filter equities by type and return on investment.

1.3.4 Statistics

The statistics sub page displays multiple graphs of the equities the user have added to their portfolio, helping the user to get a better understanding and overview of their

returns. The statistics updates once a day, giving the user a detailed and clear overview over his/her equities.

1.3.5 Log in/out

The user is able to log in to the site with their Facebook information, which enables restricted user only accessed tools and sub pages.

1.3.6 Notes

The user has the option to write their own notes while browsing and using the website. This way the user can keep notes between sessions as a way to keep track. The note tool is located in the top right, and is accessed when clicked.

2 Documentation

2.1 Setup

In this section, we explain how and where you can test the application, as well as how to run the application locally.

The React application is running on `http://it2810-08.idi.ntnu.no/`. The API can be accessed through `http://it2810-08.idi.ntnu.no:8008/api`, but you won't get much info without being authenticated.

2.1.1 Run application locally

Make sure that you are running node v6 or higher, and npm v3 or higher. Some configuration is required in order for the application to run properly. We have two config files and a database that must be set up. These are ignored by git, in order to support different configuration on different servers. We have included examples of everything, so it should not be too hard to configure. Start off by cloning the git directory with `git clone`, replace <username> with your own username:

```
git clone https://<username>@bitbucket.org/trondaal/it2810-08-oppgave-3.git
```

Navigate to `stockwatch/src/config` and `api/config`. Both of these directories has a filed called `apiConfigExample.js`. Create a copy of each of the files, and call them

`apiConfig.js`. Fill in correct URLs for the API and the site in the config files. Credentials for a Facebook app must also be included in the API config in order for Facebook login to work.

Navigate to `/api/database`, make a copy of `example-database.db` and call it `database.db`. Run the following command in both `/api` and `/stockwatch` (in two different terminal windows).

```
npm install #Installs node modules in your folder
npm start #Opens website on port 3000/Starts api for website to use
```

2.1.2 Test user

In order to test the application easily, we have created a test user that you can play around with. Stats are registered at 5:30PM every weekday, and you must therefore be registered for a while to experience the stats page properly. The test user has real test data from 1 year back in time, and can perform exactly the same actions as other users.

2.2 Choice of framework: React

All of the team members felt that they got a better grip of React. We also think that we get the functionality that we need from React. Since Angular is a more comprehensive framework, we believe it is unnecessary to use this. During the development of the prototype in Angular, the group also encountered some unexpected problems with git. More functionality was also implemented in the React prototype, and we think we have a better basis there.

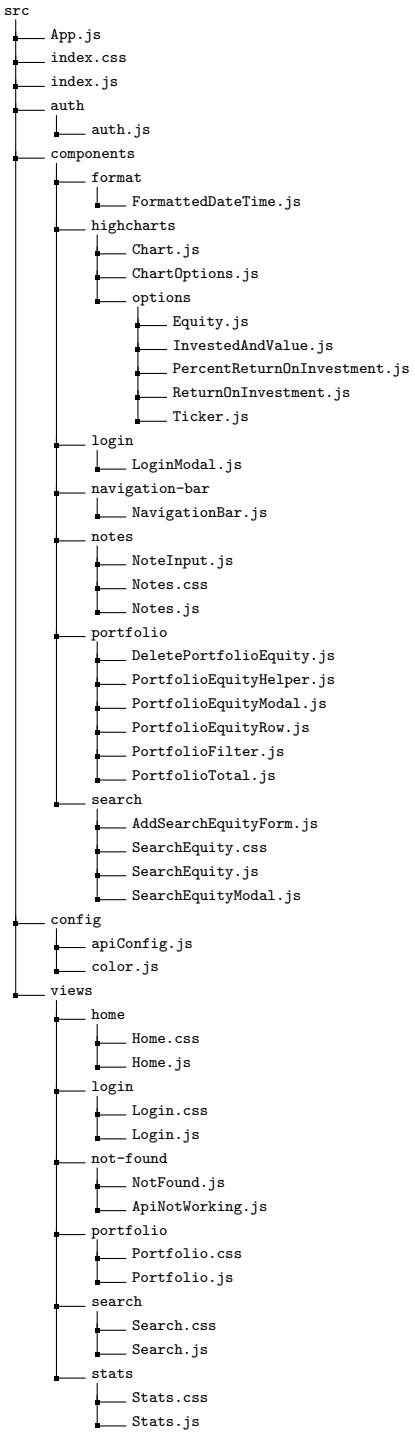
React's compile-time error versus Angular's run-time error, having React display more of a comprehensive error-log helping us the developers solve issues faster. React will provide both line number, and missing unclosed tags making finding issues quicker.

Most of Angular's documentation is written in TypeScript, while this can be converted to, or replaced with JavaScript, our team have more experience with JavaScript, and feel it will be easier to apply our knowledge with React.

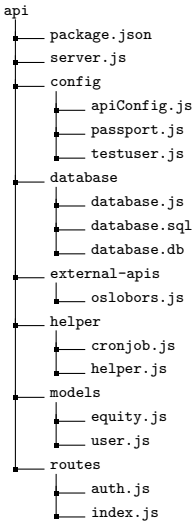
2.3 File structure

The main pages are set up in the `.../views` folder, whereas the components that's not site-specific are placed in `.../components`. This way the file-structure is clear and concise, which makes it easy to navigate. We have seperated the api files from the site files, with the page files being in `/stockwatch` and the api files being in `api`. The following tree structures starts from the `/stockwatch/src` folder and `/api` folder.

Tree view of `stockwatch/src`



Tree view of `api`



2.4 Built With

2.4.1 React

React is the main library used for our front-end view. Extended explanation can be found in the Choice of framework: React section.

2.4.2 NodeJS

Node fits our project very good because it's lightweight and fast. Our project does not have many heavy algorithms and does not consume a lot of CPU cycles. Node is written in JavaScript which the group already have experience with. It also includes a great package manager, NPM.

2.4.3 NPM

Package manager for node programs. NPM is fast, consistent and robust. It keeps packages isolated from other projects and avoids version conflicts. It makes it easy for all the group members to be up to date and install all missing packages that other members have installed.

2.4.4 React-bootstrap

React implementation for bootstrap, a front-end framework. Bootstrap gives us an easy and smooth baseline for the styling of the website. Bootstrap was mainly chosen to make it easier to build a responsive website using bootstrap's grid system.

2.4.5 React-router

React-router is a routing library to keep the UI in sync with the URL. It also provides a simple API for navigating through the different pages. React-router was chosen because of its simplicity, as well as it integrated well with the rest of our dependencies. React-router-bootstrap was used to integrate react-bootstrap with react-router.

2.4.6 Fuse.js

Fuse.js is a front-end library for searching that gives a real-time responsive search letter for letter. Fuse also supports threshold, which is very useful because the developer can be decided to what extent a user can have spelling errors in their search term and still get the result they were looking for. It also has support for `shouldSort: <Boolean value>`, which if true means the search returns the results with the closest resemblance to the search at the top. Fuse was chosen because it can search in large data set efficiently compared to other search libraries we looked into.

2.4.7 Passport

Authentication middleware for Node. Passport takes requests from the application and authenticates it. It is made to work with several OAuth providers, such as Facebook and Twitter, in addition to local authentication. As applications have unique authentication requirements, Passport have mechanisms, known as strategies, packaged as individual modules. Applications can choose which strategies to employ, without creating unnecessary dependencies. Passport was chosen because of its modularity and the possibility to select the login strategies that we wished to use.

2.4.8 SQLite

Lighter version of SQL-database, with only the bare bone commands needed for us to handle our data. SQLite is preferable over the heavier SQL-databases due to the usage of the website and the size benefit. SQLite saves the data directly in a single file on disk, which makes it easy to import locally. This also allows for quick and simple setup of test database files. SQLite has no support for user management, however we did not feel this was pertinent issue, since the application was implemented with an API.

2.4.9 Highcharts

Highcharts shows interactive graphs in pure JS. We use Highcharts to display and visualize tickers, return on investement, and values of equities. This way the user can easily get a good overview of the desired equity. Highcharts is free in non-commercial projects and has an API that is easy to work with, which is why we chose it.

2.4.10 React-intl

Components and an API to format dates, numbers, and strings, including pluralization and handling translations. Using React-intl, all dates, times and different formats (such as decimal points) are automatically translated based on your computer's own settings.

2.5 Possible future functions

We initially had a profile page in the project. This page would display information about the user, as well as giving the user options to edit his/her profile or turning of/on portfolio-sharing. Portfolio-sharing would serve as a sub page where the user can find other people's portfolio's or share their own. We later removed this because we would not have enough time to properly implement it.

Another possible function would be to integrate other API's, e.g Yahoo Finance API. This way the user could also keep track of their international equities.

By implementing AI into the application, we would create recommendations to the various users, which would be displayed on the front page. These suggestions would be based on what the user would be most interested in investing based on their portfolio and earlier investments, as well as what is most likely to be of value to them.

There are several ways the website can be expanded, like lists of the top stocks and funds, and showing news and relevant changes in the marked.

3 How the requirements were fulfilled

Ta utgangspunkt i en applikasjon av typen "katalog" hvor brukeren skal få presentert en ***liste*** *av noe.*

- We have list views both on the search page and on the portfolio page.

Webapplikasjonen skal kjøres på gruppaas virtuelle maskin og bruke ***node.js*** *på serversiden. Det er selvsagt en fordel å kunne utvikle og teste på egne maskiner.*

- Both the API and the React application is using node.js.

Webapplikasjonen skal bruke ***React*** *eller Angular 2.0, men det er greit å i tillegg bruke* ***andre bibliotek*** *eller løsninger som dere finner hensiktsmessig.*

- The application is using React. We also use other libraries, which are listed above and in the `README.md` file.

I webappliksjonen skal det inngå en ***database*** *som kjøres på gruppas virtuelle maskin. Type database og hvordan denne brukes er opp til dere å bestemme. Dere skal demonstrere både* ***skriving og lesing til databasen*** *fra webapplikasjonen inklusive en form for* ***søk*** *(i praksis dynamisk brukerdefinert utvalg av det som skal vises).*

- We are running a SQLite database that is accessed through the API.
- Our API is also accessing the Oslo Børs API, which contains loads of real time and historic equity data.
- Reading from the database is done when a user is logging in, when we get a users portfolio, and when we get user stats.

- Writing to the database is done when creating a new user, when adding equities to a users portfolio, and when stats are updated once every weekday.
- Searching in the data is done on the search page of the application. The data set from Oslo Børs consist of 1800 equities. We introduced a threshold of 0.2 in our search, so that you don't need to type 100% correctly when searching for equities.

*Brukergrensensittet skal ha **listebasert visning** med få detaljer for hver enhet, og hvor målet er å vise brukeren hva som er i databasen eller hva som er resultatet av et søk. Brukeren skal ha mulighet til å se flere **detaljer for hver enhet** enten i et eget vindu, eller ved at listen enheten i lista har expand/collpase egenskap.*

- We have list views both on the search page and on the portfolio page.
- The portfolio page shows merged data from the database and Oslo Børs, so that the user can see how much his equities are worth.
- When clicking on a row in the search page, a popup is opening with more info about the equity, and you are also given the possibility to add the equity to your portfolio.
- When clicking on a row in the portfolio page, a popup is opening with more info about the equity and your holdings, as well as the possibility to delete the equity.

*Den listebaserte visningen skal kunne **sorteres** på minimum to forskjellge egenskaper. Eksempel: etter at brukeren har fått retunert en liste etter et søk skal brukeren kunne bytte mellom forskjellige sorteringer.*

- The portfolio can be sorted on all the seven attributes listed.
- We decided to not include sorting options in the search page since the data set is so large, and should mainly be used for searching and adding equities that you have invested in.

*Den listebaserte visningen skal kunne **filtreres** på minimum to forskjellige egenskaper. Eksempel: etter at brukeren har fått returnert en liste etter et søk skal brukeren kunne krysse av på en egenskap for å få begrenset antallet enheter i resultatsettet til kun de som har denne egenskapen.*

- In the portfoilio page, it is possible to filter on the type of the equity (funds or shares).
- It is also possible to filter on return on investment. This is implemented as a slider, where the minimum value is the equity with the lowest return, and the maximum is the equity with the highest return.

*Den listebaserte visningen skal ha **dynamisk lasting av data**. Eksempel: etter et søk vises de 10 første treffene, men flere lastes når brukeren scroller eller ved blaing i sider.*

- If we have more search results than what is showing on the search page, another 10 results will show when the user reaches 5px above the bottom of the page. We also included a button for loading 10 more rows in case the scroll spy fails.

*Webapplikasjonen skal ha "min side" funksjonalitet som i praksis betyr at **en bruker skal kunne logge seg på** og at det blir **registrert noe** fra søkeaktiviteten f.eks. hva brukeren har sett på eller søkene som er brukt.*

- It is possible to log in with Facebook. The user will be saved in the database. A user can also add/"register" equities, that will then be linked to the user.
- Many of the API routes are unaccessable if the user is not logged in.

*Webapplisjonen må implementere "**session**"-håndtering.*

- The application uses only one cookie, and that is the session cookie. The session cookie is used to identify if the request is authenticated and connected to a user.

*Webapplikasjonen skal ha et litt "**fancy" alternativ visning av listen** f.eks. visning på kart eller **visuell grafisk fremstilling av data**, ordsky ea.*

- We are using graphs on multiple pages. The graphs visualize equity value, exchange rate, your portfolio history etc.