Project Report

Dat076 - Group 4

Kilberg, Oscar - IT
oscarki@student.chalmers.se

Karlsson, Leo - IT
leoo@student.chalmers.se

Axel Bergrahm - IT
bergrahm@student.chalmers.se

March 17, 2019

CHALMERS
UNIVERSITY OF TECHNOLOGY

https://github.com/kaffe-work/dat-076-Yourcash

Contents

1 Introduction

This group has created a fullstack webapplication with the purpose of managing your financial expenses. This is done by adding the seperate expenses the user would have over a span of time into a list with different categories, where the users can track them seperately. This program is called YourCash.

The repository for the project can be accessed at: https://github.com/kaffe-work/dat-076-Yourcash

The intention is that each user should ahve their own login and expenses, and that everything should be working in a one-page fullstack webapplication. However, the application should not hold the user responsible for keeping to your budget, as this is both outside the scope of this webapplication, and outside the scope of what is desired in an application of this kind.

2 List of use cases

2.1 Login

This is a very basic CRUD login feature, where the the username and password input in react is compared to the postgres database.

2.2 Logout

This does very much the opposite of the login CRUD, and ensures that an user can sign out of the application.

2.3 Register

This usecase is very basic CRUD as well, and the only difference compared to the Login usecase is that the data is inserted into the database instead. However, to make the user experience more streamlined, as you register you are automatically logged in with your newly made account (if registration succeeds).

2.4 Show Registration form

This is an update CRUD, where one only allows the user to access the data that is needed, designed to give enough visual space for the application only when desired.

2.5 Hide registration form.

Conversely, this CRUD hides the registry form when it is not in use, to simplify the visual aspect of the YourCash application.

2.6 Expense list

This is a read CRUD that imports the list of expenses a user has from the Postgres database to the react frontend.

2.7 Add expenses

And conversely, this use case adds input from the frontend into the database.

2.8 Update expenses

This usecase updates the expense list only when asked i.e when pressing the update button. This is a choice so that the user need not be visually bothered with expenses until they are actually added.

2.9 Expense sorting

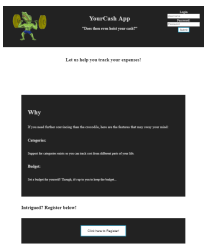
This is a more interesting usecase, as this is a specialised read. Here it organises the data of the expense list into different categories.

2.10 Expense search

The most complex usecase in the YourCash application is the search feature. Here it picks any substring in the input field and matches it to any substring (case sensitive) from the data imported from postgres. However, it is important to note that the entire search function runs in the frontend - This way it can not access data that is outside the scope of the current user.

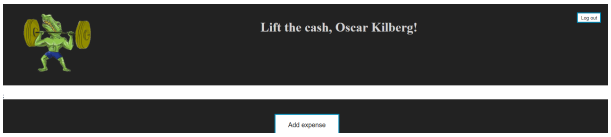
3 User manual

The YourCash web application's front-end is built on the Javascript library *React.JS* which is made by Facebook. One of the defined strengths with React.JS is that it has the ability to re-render individual components. This means that the page can have a non-confusing and unified design within a single window.

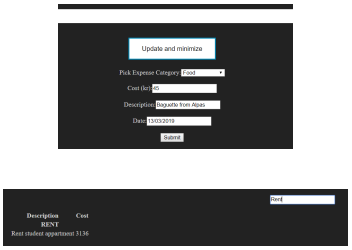


From this window we designed the page to be easy to use. From the looks of it, it is only a header and some information regarding the features of the application. Here is where the strengths of

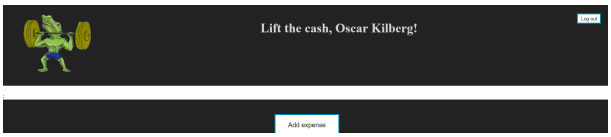
React.JS comes into play; as the user has finished reading the list of features of the app he or she will find a button, expanding a menu where they can register. If they decide to click on the button, a register-form component will appear without having to reload or land on a separate page.



Once a user has registered, you are automatically logged in and ready to use the app. The log-in form component can be found in the header of the application but only while no user is currently logged in. Once the user logs in, the components of the landing page area (all visible components from the) are again changed without having to reload the page and will be greeted via the local state of the user's log in.



From the logged-in state of the user, he or she is able to view the expenses that have been previously added. If it is the first time the user logs in, intuitively there should not be a list there. Though the user has the ability to click on a button which adds a React component to the page where the choice to add new expenses are possible.



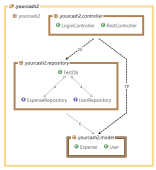
The different categories that appear are as of now set in stone, but future software updates may bring the user the ability to create own categories.

4 Design

- describe the technical construction of your app, including all libraries, frameworks and other technologies used.

Database: PostgreSQL database was used to store data and with help of JPA which served as our "Object to Relational Mapping" API data was easily added and accessed. Two tables were used, "Appuser" and "Expense".

Frameworks and Libraries: Spring Boot was used to create the Spring Framework based Java back-end with REST features. In comparison to JavaEE, Spring was easier to learn and use as it provided many helpful tools for creating a MVC structure.



ReactJS, a component based JavaScript library, was used to dynamically render our view similarly to a Single-Page app.

The jQuery library and Ajax helped make requests to the server.

5 Application Flow

Since the front-end of the application is made in React.JS the general flow of the application is simple. There are not many states which the application need to keep track of due to the re-rendering functions within the React library. However, the basic state of the application is as follows:

- The user lands on the first page
- The user confirms their will to register and a register-form appears
- The user registers, then logs-in
- The user wants to add expenses and is able to chose expenses from a list of categories.
- The user is done and logs out.

6 Responsibilities

DISCLAIMER
We had a lot of problems setting up the environments on all of our computers, especially on a Arch Linux system. In the end, after days of trying we got it running on two of the machines and the code uploads were shared between Axel and Leo.

The front-end of the application was created by Axel and Leo while back-end(Spring Boot + database (postgresql) was done by Oscar. In whole the joint effort was to learn the Javascript library React, where different code convetions than what we are used to made themselves apparent. Also, as part of the front-end the cascading style sheet wrestling is the result from the work. Additionally, and sadly, - since we didn't commit to the "Create-React-App" dependencies and chose to run JavaEE as the backend, we lost a whole lot of quality of life features with Babel and what felt like a million other things. Eventually we got our React.JS code running but we couldn't find the dependencies that were required for importing React files without downloading 200MB of other dependencies, so we ended up with a monsterfile where all the functions and classes where clusterd. Again, we are sorry but support and guides for React + JavaEE that we found were few and of poor quality.

After having encountered several of what seemed to be unsolvable problems trying to tie together front-end and back-end in a RESTful manner, a last-ditch effort was made through the use

of Spring Boot + Spring libraries for Java. With plenty of modern guides, we managed to construct a simple MVC architecture with REST characteristics. Project Lombok reduced time spent coding greatly and jQuery-Ajax requests succesfully returned the JSON data our React front-end was waiting for.