CashIT
A personal finance tracking app
DAT076
Grupp 10

Marcus Axelsson 991025-6016,
Joakim Ohlsson 950425-3072,
Daniel Karlkvist 990928-6636,
Theodor Khademi 980726-6136 &
Robin Repo Wecklauf 990428-5179

Chalmers University of Technology

16 mars 2021

Innehåll

1 Introduction

Why do we spend money? Money is something that we all need to survive. You need money for basic everyday-needs like keeping a roof over your head, food and clothes. In addition, it is extremely easy to spend money on materialistic things like a new car or the best espresso machine. Buying things with just a swipe of a card gives us an instant gratification, which causes us to want even more. This being said, we need to keep track of our personal economy in an easy and convenient way. The app developed in this course is a personal finance tracking app. When you have created an account, you can easily view your expenses for a month, a project or for multiple months at the same time.

The app is divided into six different pages: Login, Create Account, Dashboard, Monthly, Graph and Profile. The first page you interact with is either Login or Create Account since you need an account to be able to use the app. Once you have been logged in, there are four different pages to navigate through. The main screen is the Dashboard, where you have an overview of all your months and projects. Each of these have their own card with some easily accessed information, but also a button that takes you to the a more detailed view. This leads you to Monthly, where you can see all your transactions for specific categories in a drop down list. In addition, you can see your total expenses in a specific category and visualize the expenses in a doughnut chart.

The Graph page visualizes your expenses in more detail than the Monthly page. Here you can see all expenses in both a bar graph and a doughnut chart. You can also choose the time interval that you prefer.

Lastly we have the Profile page, where you can view you current personal information, but also update it if something is out of date. You can also see all the categories that you have as well as what type and color they are.

The GitHub repository is available here.

2 A list of use cases

1. A user can create an account
2. A user can log in
3. A user can update their personal information
4. A user can see an overview of savings, incomes and expenses for each month
5. A user can see a more detailed view of each month
6. A user can add a new transaction to a specific month and category
7. A user can add a new transaction to a specific month, with a choice of categories
8. A user can delete an existing transaction from a specific month and category
9. A user can update an existing transaction from a specific month and category
10. A user can filter which categories being displayed by the graph by clicking on the colors over the graph
11. A user can add new categories to organize their expenses
12. A user can see a visualisation of their transactions in the form of different graphs
13. A user can hover over a color in the doughnut chart to show an exact amount of that particular category
14. A user can hover over a color in the horizontal bar chart to show an exact amount of that particular category
15. A user can choose an interval for their transactions to visualize in different graphs
16. A user can sign out

3 User Manual

3.1 Login/Sign-up

The user opens the application and is faced with a sidebar with a button that says start. When pressed, the user is navigated to a login page. The user needs to type in their email and password, and when that's done press the button "Login". If the user does not have an account, he/she needs to create one by pressing "Don't have an account yet?". Thereafter, the user can fill out the form consisting of name, last name, email and password. After this is done, the user can press "Create account" to create the account. The user is then redirected to the login page to log in if the sign-up was successful.

3.2 Dashboard

The user is automatically redirected to the Dashboard page after logging in. Here, the user can see all their monthly overviews, if he/she has any. The overviews consists of the month name as the title, a doughnut chart and a button "Go to view". The dynamic doughnut chart shows three different colors:

- Green: Visualizes a total amount of all incomes.
- Red: Visualizes a total amount of all expenses.
- Purple: Visualizes a total amount of all savings.

The colors are changed proportional to the circle. For example, the doughnut chart is filled with green if the user only has 20 000 kr worth of income for a month with 0 expenses or savings. In addition, the user can hover over a color in the doughnut chart, showing an exact amount of that particular category.

3.3 Monthly

The user can navigate to the Monthly page by pressing "Monthly" in the side navigation to the left in the application. Here, the user can see their expenses in different categories for a specific month, if the user has any. The title of the page is the name of the month. This makes the user know which month he/she are looking at. If the user doesn't decide to navigate to the monthly

page through a dashboard card but through the sidebar when first logging in to the app, the page will load the month that is on the users computer. If the user decides to navigate to the monthly page through a dashboard card the monthly page will load information for the specific month. The info for the page for the selected month is then saved for when the user navigates to the monthly page through the sidebar. If the user wants to change the month being displayed in the page, they have to go to the dashboard and select another card with a different month. Furthermore, the user can add a transaction to a specific category by pressing "Add new transaction" at the bottom of a specific category table. The user is being faced with a pop-up window, consisting of three types of input for the user: "Name/Description", "Amount" and which date the transaction was made. When adding an transaction to the category table, it shows up as a row in the dropdown table. In addition, the total of the transaction gets calculated at the bottom of the dropdown table above the button "'Add new transaction". The transactions in the category table can be edited or deleted when you bring down the dropdown. A new transaction can also be made by pressing the "Add New Transaction" floating button in the bottom right corner. Here the user is faced with another popup, where they can specify what category they want to add the transaction to through a dropdown list. The user can add their own categories by pressing the button "Add new category" in the bottom right corner. The user is then once again being faced with a pop-up window, where the user can insert the category name, pick a color for the category and mark it as income, expense or savings. Lastly, all the transactions in each category is being added to a doughnut chart to the right of the application. The user can hover over a color in the doughnut chart, showing an exact amount of that particular category. The color added to a category is the color that's being used in the chart. The user can also filter which categories being displayed by the graph by clicking on the colors over the graph.

3.4 Graph

The user can navigate to the graph page by pressing "Graph" in the side navigation to the left in the application. This page allows the user to visualize all their transaction within a time interval with beautiful graphs. The user first chooses the date where they want to calculate from and then which date they want to calculate to. Finally they can press the "See Statistics" button below to see all of their transaction data for that specific time interval. In the

middle of the screen the user can see all their expenses in a bar graph, where each category have their own bar. There is also a doughnut chart to the right of the bar graph. This chart shows the expenses in another way where each section displays a different category, similarly to the doughnut chart on the Monthly page.

3.5 Profile

The user can navigate to their profile by pressing "Profile" in the side navigation menu to the left in the application. The user is faced with their profile information on the left of the page consisting of "Name", "Last name" and "Email". In addition, the user can edit their profile by pressing "Edit profile" at the bottom center below "Email". When pressing "Edit profile", the user can type in their new name, last name and email and save it by pressing "Save". On the right-hand side of the page, the user can see what categories they have, with the type and the color of the category being shown as well.

4 Design

This project is built up by two different parts, the frontend and the backend. The frontend is based on the JavaScript framework "React.JS", which helps us to create an object oriented web application with reusable components. This decision was made since it is a modern way to create web apps, but also since it let us to reuse components rather than duplicating code. In the react project we are using different libraries to make the code more modular. We are using *BrowserRouter*, *Link* and *Route* from the "react-router-dom" library. This helped us to link pages together to navigate between them in an easy and convenient way. Another library that we have used is also from React, called "react-color". This library is used to show and choose a color in the color picker when adding a new category. We are also using "react-bootstrap" to keep the app responsive between different screen sizes. "React-bootstrap" also comes with some existing components that we have used in the app, e.g. the accordion on the graph page. On the graph page we are also using another library called "chart.js", this library helps us to visualize the expenses and creates beautiful bar graphs and doughnut charts. Furthermore, we used the library "FontAwesome" for the icons used in the application.

The backend is a object-relational model built in Java with the help of several different libraries. The library "Lombok" was used on the entity classes of the application to quickly generate setter and getter methods and to minimize boilerplate code. The entities we have in the database are Users, Categories and Transactions. The entities are connected to individual "Data Access Objects" (DAO) to provide an interface to our persistence or database with the help of Jakarta Enterprise Beans(EJB). Each DAO class also inherits methods from an abstractDAO class that contains more general functionality such as creating or removing. The complete SQL database is built and connected to with the help of Apache Derby. Finally, to communicate with the frontend, API classes are implemented for each entity. These API classes contain all the functionality that are used for the communication between the frontend and the backend, for example when adding categories, transactions and users to the database, as well as providing data for our dashboard cards, the monthly page etc. Each API call that modifes data in the database has the @Transactional annotation which makes sure there will be a rollback if something goes wrong and also ensures that the modified data is synced to the database. We also use a SessionScopedBean to determine if the client

browsing the website is logged in or not. If a client isn't logged in the only option will be to log in and the API only allows to be called when a user is logged in. Both the backend and frontend is then deployed on a Payara Server. The testing of our backend and database is facilitated with the help of the Arquillian testing platform library. We use a separate test database to conduct these tests.

5 Application flow

Use case number 2 - Log in to the app

When a user wants to log in to the app, he/she fills in "Email" and "Password" connected to the account. This form is being rendered by the *Start.js* file. Furthermore, when the user decides to press "Log in", the form first validates that the user has a correct type of input, e.g. the email input has the type of an email with an "@". The method *handleSubmit* includes a POST request of a JSON object from the form consisting of email and password. This request communicates to the backend by fetching the url we have set for the *checkCredentials* method that we have implemented in the UsersAPI. In this method, we check in the database if a user with that email and password exists and sets the state of a session scoped bean to the Users object that just logged in. The Users object is later on referred to when new API calls are made to make sure the user only sees its own information everywhere in the application. The login API call then gives the user in the database back as a response and we check again if that an id exists and thus the login was successful so we redirect them to their dashboard page and they can now use the app. The sidebar is also updated to show all the pages the user can navigate to once logged in.

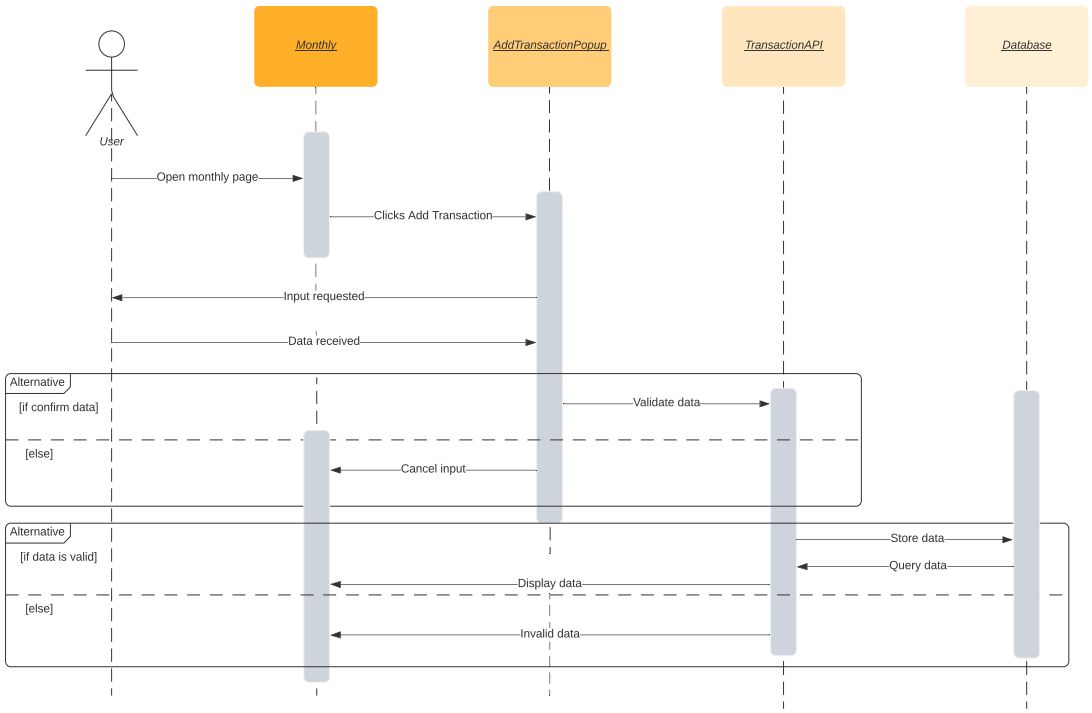
Use case number 6 - Add a new transaction

When a user wants to add a new transaction he/she needs to navigate to the Monthly page. There are two possible ways of adding a new transaction, either by clicking the button "Add new transaction" under the correct category or in the bottom right corner. The button under the category table and the floating button in the right-hand corner are connected to the classes *AddIncomePopup.js* and *AddTransactionPopup.js* respectively. The popups are connected to different components as the input needed for a new transaction are different depending on what button you press. No matter which button the user presses, the popup becomes visible in the center of the screen. In addition, it both blurs out the rest of the Monthly page by making it darker and disables the interactivity for everything besides the popup. To remove the popup and reset to the monthly page the user can press the "Cancel" button in the popup. The popup displays what inputs the user has to make to add a new transaction. For the popup when the user presses the "Add New Transaction" button in the category table, a description, an amount and

the date of the transaction has to be inserted. For the popup when the user presses the "Add New Transaction" floating button in the right-hand corner, the user also has to input what category the transaction should belong to.

When the user is done with the inputs they can press the "Confirm" button to complete the process and add the new transaction to the category if the validation succeeds. To do this, the inputs of the popups are converted into a JSON object and then sent as a POST request to the TransactionAPI. The POST method in the TransactionAPI is *addTransaction*. Here, we first check to see if the user is in a session and if they are we set a userId as the id of the user in the session. The date input is formatted so that it can be used to input into the Transaction table in the database. A data variable is then defined with all the attributes for a Transaction which is then checked to see if the attributes match the ones in the JSON object from the request. Both the amount, which is parsed into an int, and the categoryName attribute are defined, format checks are then made on them to make sure they are valid for the Transaction table in the database. The correct category for the transaction is then found and checked to see if it exists using the categoryName. The new transaction with a description, amount and category is then created and the date goes through a final check before inserted into the new transaction as well. The response is then sent out back to the frontend. The data is taken from the API and the transaction is inserted into the correct category table. The page is reset to the monthly page. If any of the validation steps goes wrong in the API an error message will be sent back and displayed for the user to easily correct his inputs.

See Figure 1 for sequence diagram.



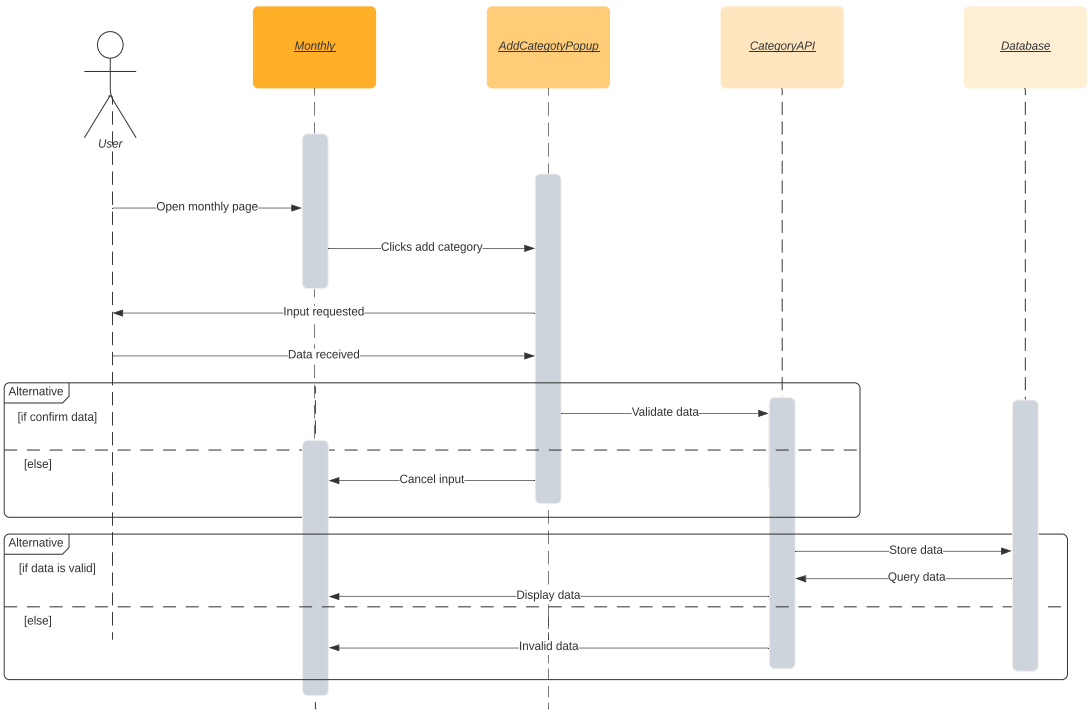
Figur 1: Transaction sequence diagram

Use case number 9 - Add new categories

When a user wants to add a new category he/she needs to navigate to the Monthly page. In this page, there is one button that adds a new category, and it is the floating "Add new category" button in the lower right corner. This button makes the component *AddCategoryPopup.js* visible, which is a pop up in the middle of the screen. This pop up asks the user to input a category name, choose a color for the category and lastly choose whether the category is an income, expense or saving. When this is done, the user can either press "Cancel" or "Confirm". The "Cancel" button does nothing else than hiding the pop up. The "Confirm" button is connected to a method called *addCategory* that sends the data to the API by creating a POST request with the body of a JSON object with the category name, color and what type it is. This request is sent to the API, which firstly checks if it exists a session with a valid user that is logged in. If no user exists in the session the server returns

a status 403(Forbidden) response. If a user exists the server checks for the correct amount of attributes and correct names, if any of those aren't fulfilled the API returns a status code of 400(Bad request). If no response is returned after checking the attributes it validates the input to be a real color and one of three types(INCOME, EXPENSE, SAVINGS). If the validation goes wrong, the API returns a status code of 400(Bad request) and if not it checks for a duplicate key and if there exists a category with the same name for the logged in user the API returns a status code of 409(Conflict). If everything works out as intended it is now safe to create a new category to the user and return a category object in JSON format together with a status code of 200(Success). The category can then be seen in the Profile page and in the drop down list used to add a new transaction to a category that is not shown in the Monthly page. If any of the validation steps goes wrong in the API a corresponding error message will be sent back and displayed for the user to easily correct his inputs.

See Figure 2 for sequence diagram.

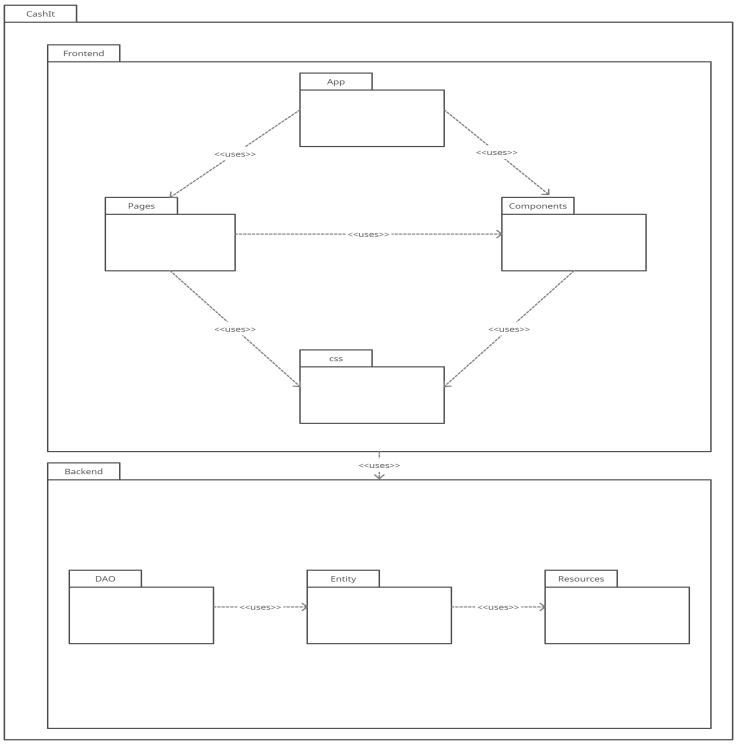


Figur 2: Category sequence diagram

6 Diagrams

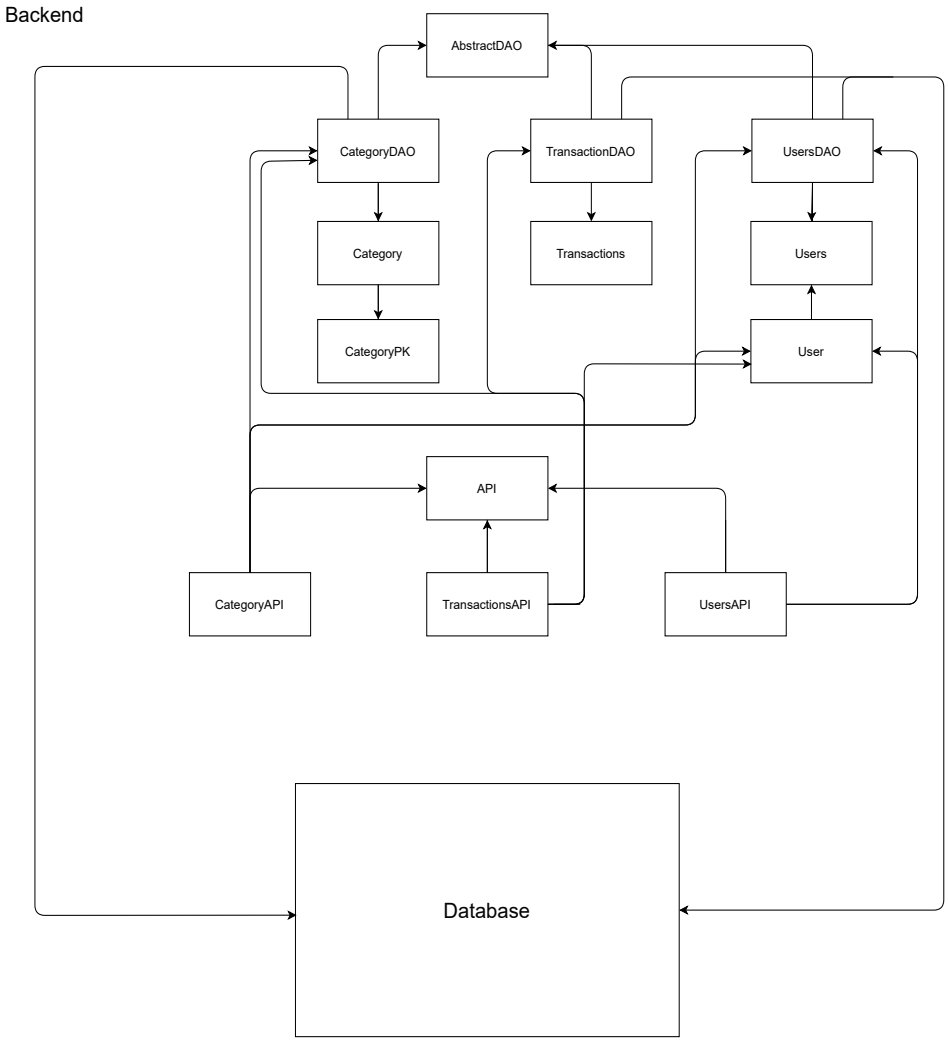
Figure 3 below shows the package structure of the application. The Figure is divided into two larger parts, backend and frontend. These parts are later divided into smaller packages with different responsibilities. The App package in frontend is the package that the user interacts with, it is basically the website. The App uses different pages which can be found in the Pages package. The pages are built up by different components, which can be found in the Components package. Both the components and the pages uses different styling that can be found in the css package.

The entity package contains all the entities defined for the application, and some composite keys. The DAO package handles all the data access objects. The Resources package contains all the API classes so that the frontend can communicate with the backend.



Figur 3: Package diagram

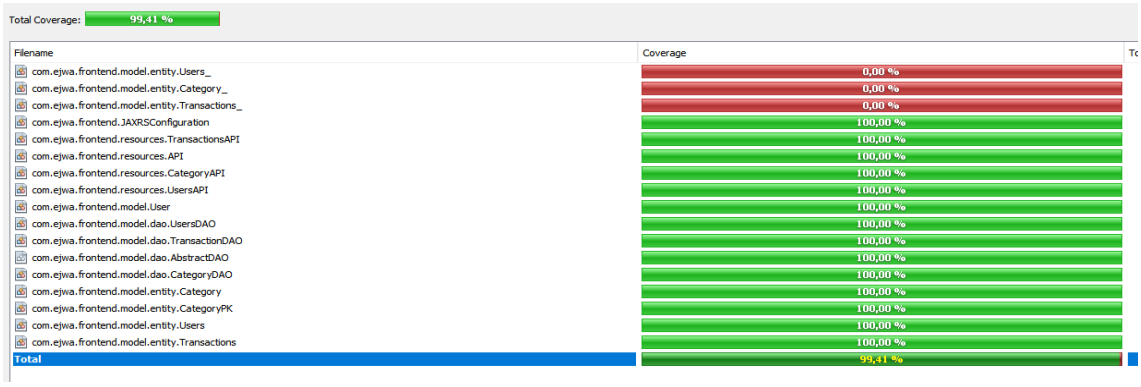
Figure 4 below shows a complete model diagram of the application. This clearly shows the dependencies between the modules. We decided to make a larger database entity to show that the DAO classes are editing the tables. This is done either by inserting new values, getting values, deleting already existing values or updating values in the table.



Figur 4: Model diagram

7 Show a code coverage report

In Figure 5 below is a screenshot of our code coverage report generated by JaCoCo. The classes that have not been tested is auto generated and that's the reason why these haven't been tested, which results in a report with almost 100% code coverage.



Figur 5: Code coverage

8 Responsibilities

Before diving in on each individual's contribution to this project, we would like to point out that we have had some personal issues throughout this project. Unfortunately, one member was out for 1.5 weeks because of the corona virus while another one has undergone a surgery and was incapable to contribute to the project during roughly 1.5 weeks as well. This means that we lost a lot of time. After speaking to Robin about this, he mentioned that we could hand in the project and get it graded in any "omtenta period". Although this was an option, we felt like we could finish this project and get a result that matches your requirements. Therefore, we hope that our final draft of "CashIT" has a good enough quality.

In addition, we have basically always been working in groups throughout the whole project. This means that everybody was essentially a part of everything done in the application. This also means that the contribution shown on GitHub is not completely accurate as the insights does not include co-authors. In other words, when doing pair-programming on one computer, the one person that commits is the one getting credit for it on GitHub. It has been somewhat difficult to do this project during the ongoing pandemic and to work in distance. This is what worked best for us as a group. What we stated below this is the *main area* of responsibilities done for each individual.

All

Everybody in the group were responsible for coming up with this idea and designing it in Figma. In addition, everybody played a role in doing Lab 2, and implement a mock up in HTML. After that all of the members was a part of moving the whole project over to React. Everybody played a part in deciding and editing the design choices for the application. Lastly, pull requests has been equally split up between all team members.

Joakim

Joakim was overall responsible for the backend-part of the project. Joakim did most of the testing and also most of the functionality in the frontend which involved communicating with the backend. Joakim also designed the first version of the CircleDiagram in the frontend part, which later was replaced.

Marcus

Marcus was responsible for the original sidebar that was later redesigned, which he also was a part of. After this he started and was responsible for lab3, together with other team members. Marcus was also a part in the implementation of the graphs which is used in Dashboard, Monthly and the Graph page. Furthermore he was a part of implementing the profile and graph page. Marcus was also part of the refactoring of the backend code at the end. Lastly Marcus was responsible for writing all sections in the report together with creating the diagrams.

Theodor

Theodor was part of doing lab 3. Overall, he helped with the implementation on both the backend and frontent parts of the application. His main responsibilities on the frontend were doing an initial design and implementing the graph, monthly and profile page. Later in production he also helped redesign many components and pages and refactored code. Theodor was also responsible for creating the redesigned sidebar in the application. Finally he also helped connect the login functionality to the frontend and made sure the user is redirected correctly when logged in. On the backend, Theodors main responsibility was connecting the frontend to the CategoryAPI and TransactionAPI by implementing the correct functionality to for example edit, add and delete categories and transactions.

Robin

Robin was responsible for creating the first draft of the Dashboard page. Furthermore, he abstracted the code at an early stage by creating components and making it more dynamic. He was also a part of doing Lab 4. In addition, he was mainly responsible for the log in and sign up page and a part of connecting it to the back end. Robin was a part of refactored the frontend code towards the end. Lastly, Robin was also responsible for writing all sections in this report together with creating the diagrams.

Daniel

Daniel was responsible for creating and redesigning the dashboard card. Furthermore he implemented the pop up window when adding new transition in the monthly page. He was also a part of doing lab 4. In addition he also was a part of the new sidebar. Closer to the end, there were a lot of clean up

that had to be done including restructuring parts of the HTML to be able to make the code more responsive, and also styling the pages a bit more to look better, which Daniel was responsible for. Daniel was also part of refactoring the backend code at the end.