FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Assignment of bachelor's thesis

Title:	XpenseTracker Client - client part for personal finance manager
Student:	Amir Qamili
Supervisor:	Ing. Zdeněk Rybola, Ph.D.
Study program:	Informatics
Branch / specialization:	Web and Software Engineering
Department:	Department of Software Engineering
Validity:	until the end of winter semester 2022/2023

Instructions

The goal of the thesis is the creation of the web and mobile client application for the XpenseTracker app for managing personal finances. The key features include:
- managing of multiple personal accounts
- tracking transactions on the accounts with advanced categorization
- support for recurring and future payments
- statistics and reports

The thesis should consist of:
- analysis of the processes and requirements with a focus on user interaction
- research of existing applications and their comparison in the context of user experience
- user interface design for web and mobile client apps
- analysis of server-side API and functions and architectural design of the client apps
- implementation of the web and mobile client apps
- documentation of the solution, including a user guide and deployment guide

---

Electronically approved by Ing. Michal Valenta, Ph.D. on 16 September 2021 in Prague.

Bachelor's thesis

XPENSETRACKER CLIENT

Amir Qamili

Faculty of Information Technology
Department of Software Engineering
Supervisor: Ing. Zdeněk Rybola, Ph.D.
February 10, 2022

Czech Technical University in Prague
Faculty of Information Technology
© 2022 Amir Qamili. Citation of this thesis.
This thesis is school work as defined by Copyright Act of the Czech Republic. It has been submitted at Czech Technical University in Prague, Faculty of Information Technology. The thesis is protected by the Copyright Act and its usage without author's permission is prohibited (with exceptions defined by the Copyright Act).

Citation of this thesis: Qamili Amir. *XpenseTracker Client.* Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2022.

Contents

List of Figures

List of Tables

List of code listings

I would like to thank everyone who helped me during my studies and my thesis, specifically my supervisor Ing. Zdeněk Rybola, Ph.D. for his guidance and valuable feedback during the thesis, and my family for all the support.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as a school work under the provisions of Article 60 (1) of the Act.

In Prague on February 10, 2022 .....

Abstrakt

Cílem této práce je implementace webové a mobilní aplikace pro účely sledování financí se zaměřením na návrh uživatelského rozhraní s využitím aplikačního programovacího rozhraní (API), které vytvořil Jamal Azizov ve své bakalářské práci. Jako první krok před implementací je proveden výzkum a analýza existujících řešení. Další vylepšení jsou popsána v závěru jako závěry.

Klíčová slova sledování výdajů, finance, webová aplikace, mobilní aplikace, React, Typescript

Abstract

The goal of this thesis is to implement web and mobile applications for the purpose of finance tracking with a focus on user interface design, using the application programming interface (API) created by Jamal Azizov in his Bachelor Thesis. Research and analysis of existing solutions is done as the first step before implementation. Further improvements are described at the end as conclusions.

Keywords expense tracking, finance, web application, mobile application, React, Typescript

Acronyms

API	Application Programming Interface
CSRF	Cross-site Request Forgery
CSS	Cascade Style Sheets
DOM	Document Object Model
FIT	Faculty of Information Technology
HOC	Higher Order Component
HTML	Hypertext Markup Language
HTTP	HyperText Transfer Protocol
REST	Representational State Transfer
MVVM	Model-View-ViewModel
UI	User Interface
UX	User Experience
XML	Extensible Markup Language
XHR	XMLHttpRequest
VSCode	Visual Studio Code

Chapter 1
Introduction

Money is important in our day to day lives. Being able to track where the money is spent, can give us an accurate picture of where the money is going and instead plan where we would like it to go. That is the reason people are in search of the most convenient application that will help them make their finances improve.

1.1 Goals

The goal of this thesis is to create an expense tracking application, that will help users to track their finances. An application that is easy to use in terms of user experience and provides rich enough features.

Objectives of this thesis are to implement web and mobile applications fulfilling the following with a focus on user experience:

- Managing multiple
  - Accounts
  - Categories
  - Budgets
  - Projects
- Payment splitting
- Repeating payments
- Sharing account with different users
- Filtering of transactions
- Statistics and reports
- Mobile application for on-the-go updates

1.2 Roadmap

Before starting any practical work, it's important to find out what and how to implement something. For this reason an analysis is required as the first step.

Analysis should cover the current implementation including the functionalities it has, as this thesis is focusing on improvement of it. Existing solutions should be reviewed to have better understanding of the current standards required. Based on the results of the previous steps a resulting requirements should be enlisted.

Next chapter will cover the design of the solution planned. To decide which technologies to choose and how to approach on a solution to specific requirements set in the analysis. The overall design of the solution should be included as mock-ups.

Implementation chapter should include the structure of solution. It should cover the approach implemented to the problems and specific use cases that were resolved.

In the final chapter a conclusion on what has been achieved over the course of the previous steps and if the goals were achieved in the end should be described. Possible future improvements could be listed for future work.

Chapter 2
Analysis

The focus of this chapter is to define the requirements of the work that will be done as a result of the analysis carried on. The analysis has multiple steps and with each step, we will get a better picture of what kind of application we are aiming to achieve.

As the main idea of the thesis is to improve the existing solution of XpenseTracker, we have to first understand what the current implementation is capable of, why do we want to make changes and how we can improve the working solution.

After having a clear understanding of the current implementation and the missing features of this application, a review of some of the existing applications follows. In this review, we would like to find out if there is any application that fulfils the requirements from the previous analysis of the current implementation of XpenseTracker.

The following step is to analyse the backend side of XpenseTracker, that has been implemented by Jamal Azizov in his Bachelor Thesis. The application that will be developed at the end of thesis, the client side, will be consuming this API and we need to have a clear understanding of the state of the API. As we can implement at most the same amount of functionality it is providing, the analysis in this step will give the final requirements and features we are able to work on.

Next we would like to list the functional and non-functional requirements of the application from the data we gathered in the previous steps completed.

Results achieved in this step, will guide us which technologies to choose, how to design, to realise the requirements and the goals we have set.

2.1 Current implementation

2.1.1 Architecture

Modern applications have a clear separation of concerns (SoC)[1]. Meaning each part of the application deals with a separate problem. There are many benefits of this approach, such as more freedom of design, deployment or usage. When the concerns are separated, there is better opportunity for modular upgrades, reuse and independent development. However this approach has it's associated execution penalty, as separation of concerns is a form of abstraction and this means adding additional code interfaces and this results in more code to be executed.

The current implementation of XpenseTracker used a monolithic approach, where frontend and backend was located in the same codebase, under the hood of a backend framework Laravel (PHP). Since Laravel is not designed for frontend development, it was hard to create good UI with just a templating engine Blade, provided by Laravel.

Separation of client side and server side has the benefits mentioned above. It helps to keep the codebase modular, easier to scale up and in case of a decision to change the technology used, it's much easier to do it in separated codebases.

One of the requirements of this thesis, is to have a mobile client that will be consuming the API and instead of having just the mobile application's codebase separated, it was decided to separate the two parts of the current implementation as well.

2.1.2 Functionalities

Current implementation is limited by it's functionalities. It satisfies the most basic requirements that were planned. Functionalities offered in the exisiting solution include:

- **Authentication**

  User is authenticated using it's credentials, to get access to the web application.

- **Accounts**

  User is able to create and manage accounts.

- **Categories**

  User is able to create and manage and modify categories. Categories can be in a tree like structure.

- **Categories**

  User is able to create, update, remove transactions. Basic filtering and pagination allows user to interact with multiple transactions. Transfer type of transaction is supported.

As it's seen above, the existing solution is just a basic application, that enables user to make simple expense tracking and finance management. It is far from being a good canditate for actual users.

2.2 Existing solutions

In the following, the performed research on the existing solutions in the market will be presented. An analysis of some applications in the market that offer features similar to the goals of this thesis, will help us understand where to focus and what are the most important features that are missing commonly in these applications. It will also help us to set the functional requirements needed to have a user friendly and attractive UI as a result.

When choosing the applications to analyse, I tried to choose the ones previously used by myself or people around me. This includes my supervisor, who had great ideas for an ideal application, based on the problems he actually was facing while using similar applications. This way we could find solutions to actual problems users might be experiencing. This way we could also bring some new ideas about missing features in those applications used previously.

2.2.1 Wallet by BudgetBakers

It is a Czech product that offers Web, iOS and Android applications. Not every finance tracking application offer a web UI alongside the mobile applications. As not many people would like to use the web application on a daily basis. Having all applications available on all platforms is a big plus for Wallet, compared to other options.

Wallet has good and simple UI for creating and managing accounts, transactions and categories. However some of the features are not supported in the web application compared to it's

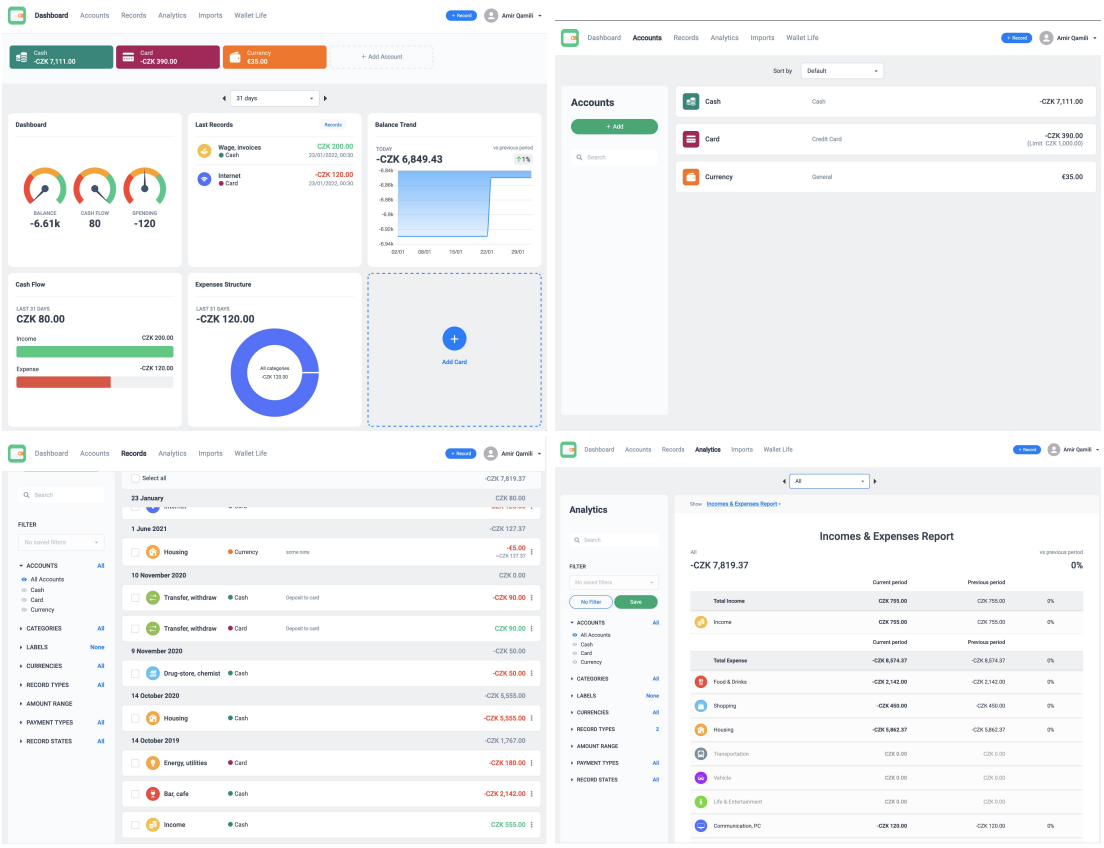

Figure 2.1 Wallet by BudgetBakers - Web application

mobile applications. With friendly and straightforward statistical graphs, it is easy to keep track of expenses on a broader scale. Wallet is one of the most successful applications in the market, on a personal level it is my favorite application.

The web application of this thesis is heavily inspired by the UI and functionalities Wallet is providing. When brainstormed to find a solution on how to implement some of the UI components, I tried to reverse-engineer those features and components the way it works on Wallet web application.

However as good as it might be Wallet is missing some functionalities that we considered could be beneficial to have in a finance tracking product. Most wanted functionalities missing in Wallet are not being able to split transactions and a functionality to group transactions for a specific goal for one time thing. Also being limited to a number of accounts, is also not very user friendly, but of course this is understandable as they are monetising their product.

2.2.2 Spendee

Another Czech product that was developed in Prague in 2013, that has Web, Android and iOS applications. However I was not able to use the web application at the multiple times I tried using it, some problems on their application I assume.

On visual aspects, the application is also well done and deserves a praise for good graphics. User experience is also good in general, you know what you are supposed to do when you use the application. The graphics for statistics related to expenses are also nice. I find Spendee limited in functionalities compared to Wallet and an ideal application I had in mind.

In the free plan, you are not allowed to do much other than creating 1 payment account, adding transactions to that account and making 1 budget to keep track of your expenses periodically. Your ability to use the application is quite limited. In Wallet you are at least allowed to have 3 payment accounts, without the need to pay. It is also not offering any kind of filtering for transactions other than a search field. It does not provide the functionalities that Wallet is also missing.

Overally, it is a standard application that offers limited functionality with a mindset to force users to pay so they can get the same type of features but with more quantity. Too much monetisation focused.

Figure 2.2 Spendee - iOS application

2.2.3 MoneyWiz

Originally a MacOS application, which is available on other platforms such as Windows, Android, iOS. It is quite rich in features it provides and the ability to have a desktop application could be interesting for some users.

One thing I learned from this application is that, having all the features possible in one place, can make the usage of the application quite hard. MoneyWiz has almost all of the features you could ask for in the finance tracking applications, but this makes the application not user friendly, this can be seen in figure 2.3. I would say user experience is quite bad. As a user I would not want to find the feature in a long list each time I need to add a transaction or view my budgets. MoneyWiz, made me realise we should be more careful when picking functionalities to implement as it can make the application unattractive to simple users who do not need that many features. Making the balance in between making the application feature rich and simple as possible, might not be an easy solution.

2.3 API analysis

The reimplemented API has been improved and modernised by Jamal Azizov as a result of his Bachelor Thesis work. Domain model represenation from his work can be seen in figure 2.4. As the applications that will be create in this thesis, will be consuming this API, it is important

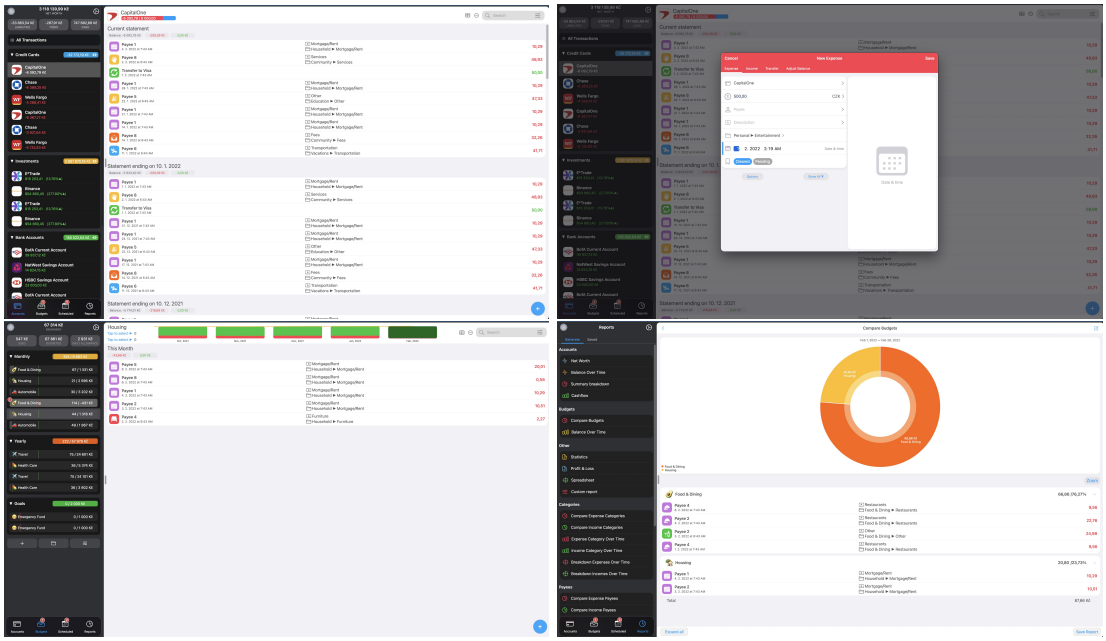

Figure 2.3 MoneyWiz - MacOS application

to analyse it for further definition of requirement in the client side of the applications. The documentation for API is provided on `https://xpensetracker-api.herokuapp.com/api/`.

List of functionalities based on endpoints and documentation:

- **Authentication**

  API provides endpoints for login, register and refresh token under `/auth` endpoint.

- **Accounts**

  This endpoint provides relevant resources to retrieve payment accounts, transactions regarding specific accounts, ability to update, delete and add new accounts under `/accounts` endpoint. It provides endpoints to share account with other users and update and retrieve the list of users accordingly account have been shared with.

- **Categories**

  API allows management of default categories set on registration. It's possible to update and delete them, also subcategories can be created.

- **Transactions**

  API provides endpoints to fetch, create, update and delete transactions under `/transactions` endpoint. It's also possible to split transactions and retrieve information, update and delete them, by using endpoint `/transactions/${id}/subtransactions`

- **Budgets**

  In the endpoint `budgets` provided by API, it is possible to create, update and delete budgets. Existing budgets can be closed or reopened. It is also possible to see the history of specific budgets.

- **Projects**

  Projects can be created, updated, deleted and retrieved under endpoint `/projects`. Projects can be activated and deactivated as well.

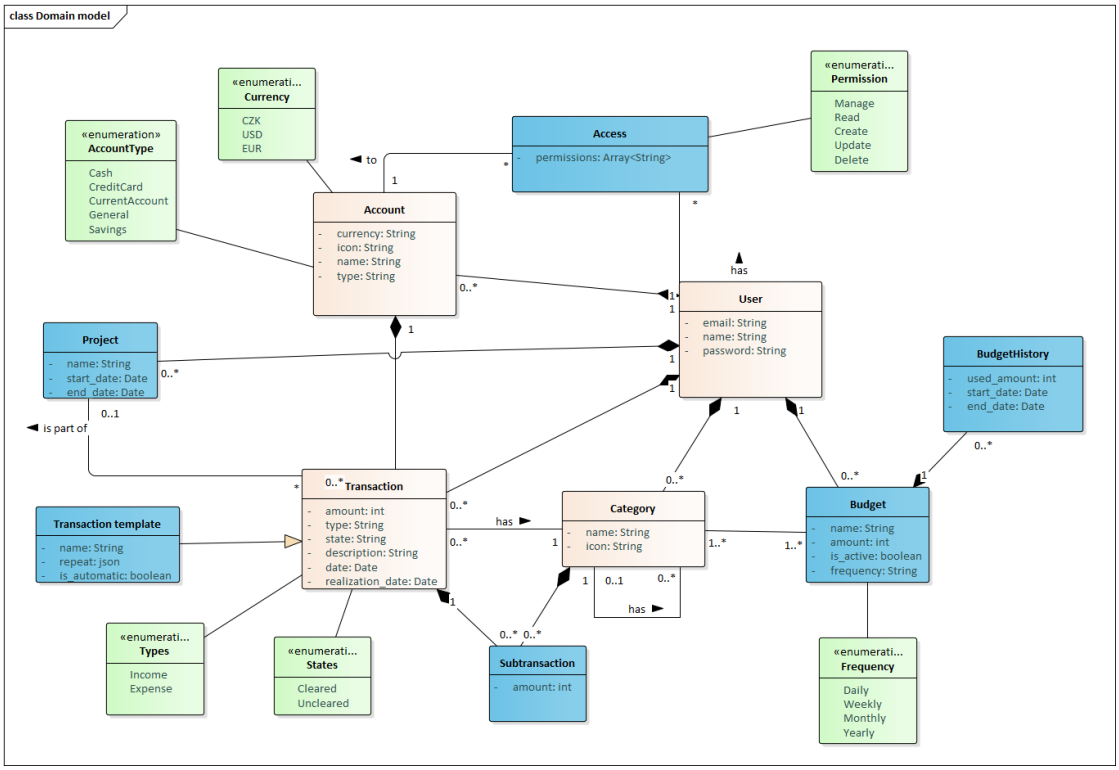

Figure 2.4 Domain model by Jamal Azizov for ExpenseTracker API

Recurring transactions

Repeating transactions are created as a template at the `templates` endpoint. It is possible to create, update and delete templates.

2.4 Application requirements

In this section, the requirements of the web application will be determined. This will ensure, what the web application will and will not be capable of. As the server-side of the web and mobile application has already been implemented, the client side of the application is bound to use the functionalities the API is providing.

Requirements of the application have been decided with the supervision of Ing. Zdenek Rybola, PhD, in the beginning of thesis work and during development some of the requirements have been adjusted accordingly.

2.4.1 Functional requirements

F1 Registration

In the registration screen, a new user enters their data and click submit to validate that there is no other user with same credentials. If there exists another user with same email, then user is notified to enter a different email address for registration. User should be able to navigate to login page in case they already have an account to authenticate and start using application.

- **F2 Login**

  Interface to authorise user with their email and password to allow the usage of features that their account gives access to. If the data provided by user are not correct, user is notified with incorrect data and they should try again. User can navigate to registration page in order to create a new account.

- **F3 Accounts management**

  User will be able to see their payment accounts in order to manage them accordingly. UI should allow the user to create a new account to add records in the next steps. It should be possible user can see balance of their account and update any data of the existing accounts. Listed accounts will allow user to take a look at the records assigned to each account.

- **F4 Categories management**

  Interface to allow user to view default categories and add categories if user chooses to. User should be able to modify or remove categories. Icons and hierarchy of categories should be modifieble. User is allowed to create as many subcategories as desired.

- **F5 Records management**

  UI to allow user to create, update, remove transactions. Created records should be listed in descending order and user should be able to filter and navigate through history of records. For better user interaction with big numbers of records, pagination should be available to group records.

  Filtering should allow user to filter out records based on different fields:

  - Categories
  - Accounts
  - Transaction Type
  - Transaction Date
  - Transaction Amount
  - Projects

- **F6 Budgets management**

  The UI should allow user to manage multiple budgets. Update, create and remove budgets and deactivate them accordingly. User is able to see budget history and see the overall state of the budget for the given period of time.

- **F7 Projects management**

  Application should support UI to manage projects. List of records for each project should be available to user for better visualisation.

- **F9 Repeated transactions**

  User is able to create repeating transactions

- **F10 Share payment accounts**

  According to API documentation user is able to share their payment accounts with other users to manage accounts together. The application should support an interface to allow user easily to share their payment accounts with other users by means of email. It should also allow user to unshare account if it has been shared with other users.

2.4.2 Non-functional requirements

- **N1 Synchronization with server**

  The application must be able to retrieve actual data from server.

- **N2 Supporting different languages**

  Application's should support different languages. That means, it will display information in language chosen by user.

- **N3 Simple and user friendly design**

  The application's design must be simple and intuitive.

Chapter 3
Design

3.1 Technologies

The previous implementation of the XpenseTracker web application was done by using templating engine Blade in Laravel, a PHP framework. As the decision to separate the client side and server side of the application was agreed on, it was necessary to choose more frontend friendly technologies. Also as one of the goals of the thesis is to implement a mobile application together with the web app, it was necessary to find a better way to make it easier to use similar functionalities with the minimum amount of rewriting.

3.1.1 Languages

There are different alternatives that focus mainly on client side of applications. Typescript, Dart and Elm are some of them. Let's take a closer look at what these technologies offer and why they should be chosen for further development.

- **Typescript**
  - Everything that Javascript offers
  - Powerful type system
  - Classes
  - Interfaces
  - IDE Support
  - Generics support
  - Great scalability
  - Biggest community among other candidates
  - Mobile development
- **Dart**
  - Fully object-oriented
  - Easy to learn
  - Mobile development
  - High productivity

- **Elm**
  - No runtime exceptions
  - Great performance
  - Enforced semantic versioning
  - Mobile development not supported

Out of these Typescript and Dart seems to be the best option for web and mobile development. Dart is new and still developing and getting bigger support in the community. Moreover with the personal experience that I had with Javascript ecosystem, it was a clear decision to rewrite the web application using Typescript to make the implementation of the mobile application easier and faster by staying in the same ecosystem, once the web application is finalised.

3.1.1.1 Javascript

Javascript is one of the most popular languages on the market. It has a big community, numerous frameworks and libraries. It's heavily used in web applications, server applications and mobile development. This makes it a very good choice to use for full-stack development, to save time and reuse functions and libraries with much less effort to make web, server and mobile applications.

One of the main disadvantages of using Javascript is that it's dynamically typed language. This makes it very error-prone, and to avoid those bugs created during development, it requires some extra effort to make the codebase more confident from the human-based errors.

3.1.1.2 Typescript

Typescript is a free and open-source programming language developed and maintained by Microsoft. It is a strict syntactical superset of JavaScript. It adds optional static typing, interfaces and class-based object-oriented programming to the language. The biggest advantage is to enable IDEs to provide a richer environment for detecting common errors during development.

Modern editors, can interpret Typescript code and provide instant feedback to the developer of possible code errors in compile time. Static typing not only allows the machine to interpret the code faster, but it also allows developers to understand the code better and faster. These features allows TypeScript to help developers feel more confident in their code, and save considerable amounts time in validating that they have not accidentally broken the project.

One of the advantages of Typescript is that it supports any Javascript codebase out of the box, meaning any Javascript codebase can be used to compile by Typescript and it can detect potential issues such as undefined data or such that are usually caused by human factor.

3.1.2 Libraries

As the saying goes, "Don't reinvent the wheel", no need to do work that has already been done. It's wise to use the packages and libraries that have been verified in the community and has enough support, as much as possible to leverage the time spent on coding for problems that are resolved already. Although, I believe that packages and libraries shouldn't just be added to a project just like that. It should be considered if the library as a whole:

- mature enough
- adds value to project
- the overall functionality needed is reasonable
- has good documentation

- simple to integrate

Going forward the libraries and tools used in the development, are written in Javascript, however all of these libraries and tools support Typescript.

Following, we will get to know the core libraries used in the development of the web app. There are more libraries used, but the following are just the most important ones.

3.1.2.1 React

React is a JavaScript front-end library for building UI components. It has been developed by Facebook in 2011 and later in 2013 it has been open-sourced. It has one of the biggest communtiy of developers and a big market share when it comes to web-development.

React creates user interfaces in a predictable and efficient way of using declarative code. It can be used to create single page web and mobile applications, or building complex applications if used with other libraries. The reasons behind React's popularity and why it's a good choice in today's technologies can be summarised as:

- **Simplicity**

  It is easy to learn React's API. It's component based approach, well-defined lifecycle and use of plain Javascript is what makes React simple learn and build web and mobile applications. It uses a syntax called JSX, to render UI components. JSX is just a special syntax that looks like HTML, which can be said as HTML inside Javascript. This syntax gets rendered as HTML by React API.

- **Performance**

  Making changes in the DOM, is costly. React overcomes this problem by minimizing the changes applied to DOM. For this React is using Virtual DOM, which makes it fast. It creates an in-memory data structure cache which computes the changes made to components and then updates the browser accordingly if there is a change in value. This allows a special feature that enables the programmer to code as if the whole page is rendered on each change whereas React library only renders components that actually change.

- **Reusable components**

  Components are the building blocks of any React application, and a single app usually consists of multiple components. These components have their logic and controls, and they can be reused throughout the application, which in turn dramatically reduces the application's development time.

- **Data flow**

  React follows a unidirectional data flow. This means that when designing a React app, developers often nest child components within parent components. Since the data flows in a single direction, it becomes easier to debug errors and know where a problem occurs in an application at the moment in question.

- **Native approach**

  React can be used to create mobile applications (React Native). And React is a diehard fan of reusability, meaning extensive code reusability is supported. So at the same time, we can make IOS, Android and Web applications.

- **Testability**

  React applications are very easy to test. React views can be treated as functions of the state, so we can manipulate with the state we pass to the React view and take a look at the output and triggered actions, events, functions, etc. There are also many libraries available to test React applications, which are easy to setup and start testing.

- **Community**

  The popularity of React increased starting 2015. It's one of the most popular repositories in Github. The community is quite well developed as there are many libraries providing customizable components ready to be used directly in your application. This is a big checkmark to one of the requirements mentioned before, when deciding about library to be used in the application.

Of course it's important to review the disadvantages of React in order to get a better undersanding if it is a good candidate for our use case. Drawbacks for React can be summarised as:

- **State management**

  React uses unidirectional data flow, which makes it hard to share data to the last child component from the parent component, if there are too many subcomponents in between that do not need the data. This case is called "Prop Drilling". To avoid this case usually you would need to use some kind of state management library, that will alow to access data without the need of prop drilling.

- **Routing unavailable**

  Routing is not supported out of the box. For this you need to use a different library. Libraries offering similar functionality as React offer this by default.

- **Memory usage**

  Virtual DOM was noted as one of the pros of React as it provides good performance. However this comes with a price, Virtual DOM consumes high amount of memory, which can cause performance issues for some users.

  Other alternatives to React at the moment are Angular.js and Vue.js [2]:

  - **Angular.js**

    It is older than React and Vue.js. It has been developed by Google and follows a MVVM pattern. Just like React follows a one-way data binding approach. Dependencies are provided in modules. It's also a good choice, however it has a steep learning curve. It's mainly used in large-scale applications. Big community support.

  - **Vue.js**

    The newest out of 3 frameworks that are comparable. It is easy to learn and has an important advantage of two-way data binding. That is a very good feature as many times one-way data binding comes with issues of data sharing between child and parent component. Community is considerably smaller than React and Angular.

  Considering the advantages and disadvantages of React library and other options, it is safe to say React is a good choice, due to personal experience with it. Disadvantages of React can be resolved by external libraries to complement it, specifically the issue with state management can be resolved by using Redux library. The same can be said for the routing problem.

3.1.2.2 React Redux

Redux is a state management Javascript library, used mainly with libraries such as React, Angular and Vuejs [3] It is basically an in-app database that handles store and use data during the lifecycle of the application in the browser. As React follows uni-directional data flow, Redux helps to solve the issue and allows any component to get the data without the need to be passed from the parent components.

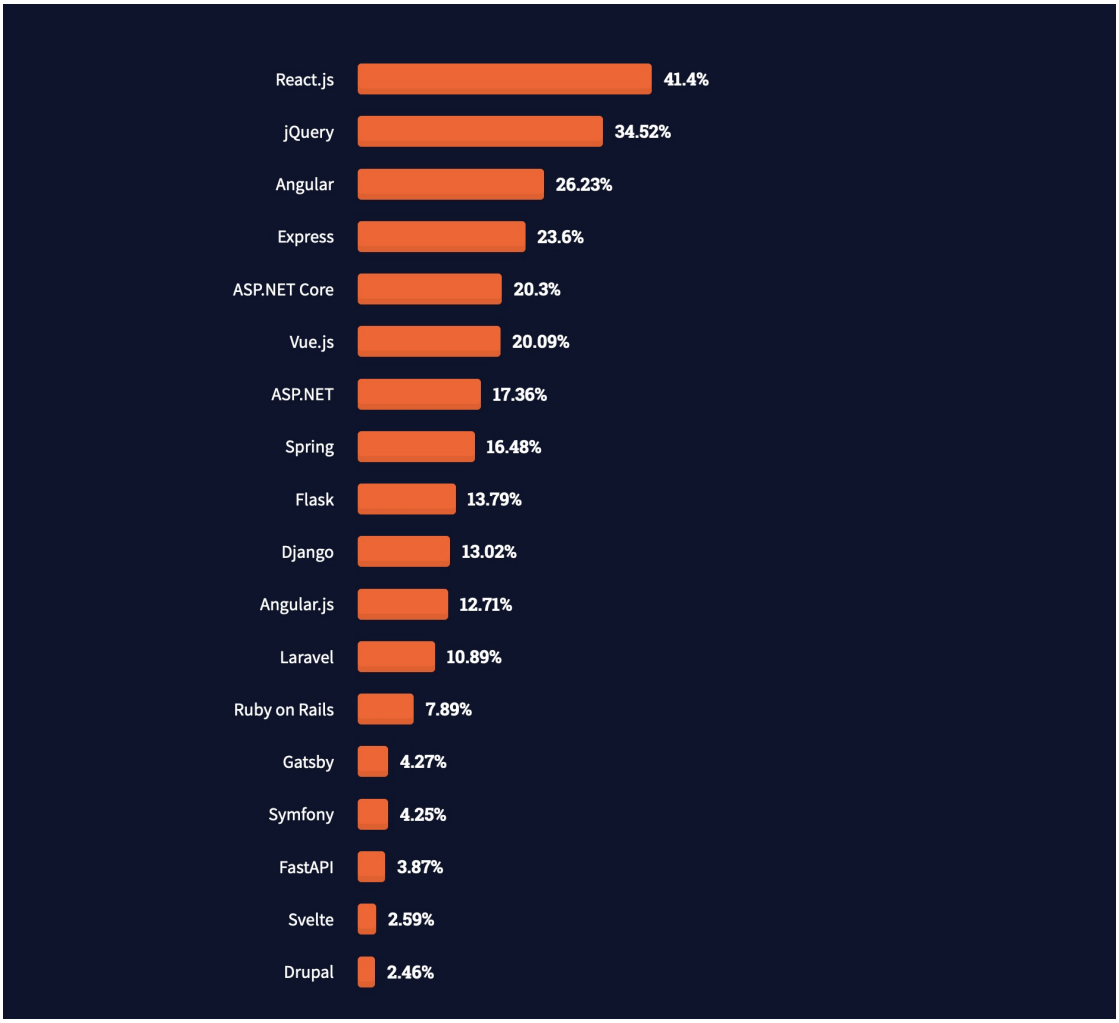

Figure 3.1 Most commonly used frameworks in 2021 according to survey by stackoverflow: `https://insights.stackoverflow.com/survey/2021#web-frameworks`

As stated previously, state management is not the strongest side of React. This is a consequence of React's component based development approach. That makes it hard to pass data from the parent component to the child component at the end of React Component tree, where there are multiple components in between. To achieve this without external library, data has to be passed through each component to the child component until it reaches the final component that utilises it. This causes bad code quality by resulting in unused data as some components are behaving as a bridge to let the data pass through.

Another problem Redux helps resolve is allowing the application to have a central source of data. If data needs to be retrieved or modified, this gets done in one single location. This prevents data to be consistent throughout application.

Of course Redux has it's drawbacks as other libraries. Redux comes with lots of boilerplate code and it can be difficult to grasp a hand of it. It's also not very beginner friendly and if not used correctly can cause unnecessary rerenders.

3.1.2.3 Redux Toolkit

Redux Toolkit is a Javascript library that makes state-management with Redux easier. As mentioned previously, Redux comes with lots of boilerplate code and it is complicated to configure as a start. Redux Toolkit helps to resolve this issues. It is simple to learn and use. Being opinionated makes it a good choice for starters.

3.1.2.4 MUI

MUI is a user interface library [4]. It provides ready to use and customisable React components that follows Google's Material Design [5].

Material Design is a design system, that aims to create a unified user experience across various devices, platforms and input methods. There are 3 principles to Material Design according to Google:

- **Material is the metaphor**

  Material Design is inspired by the physical world and its textures, including how they reflect light and cast shadows. Material surfaces reimagine the mediums of paper and ink.

- **Bold, graphic, intentional**

  Material Design is guided by print design methods — typography, grids, space, scale, color, and imagery — to create hierarchy, meaning, and focus that immerse viewers in the experience.

- **Motion provides meaning**

  Motion focuses attention and maintains continuity through subtle feedback and coherent transitions. As elements appear on screen, they transform and reorganize the environment with interactions generating new transformations.

No need to rewrite components like inputs, buttons and such that are already implemented by the community. Since MUI successfuly follows the Material Design and the variety of different components makes it a good choice to use in the application's overall design.

3.1.2.5 Axios

To communicate with the API, there is a need to make HTTP requests to get data and use all the methods the RESTful API is offering.

Javascript, hence Typescript, offer a native solution, Fetch API. It's an interface that makes the communication mentioned previously. Although there is native solution, this interface is missing some features and has little too much syntax to use it. As a result I decided to use a library called Axios to make data axchange with the API.

Axios[6] is a Javascript library that is used to make XMLHttpRequests from the browser. It makes it easy to communicate with API's. Axios supports Promise API out of the box, whereas native solution Fetch API does not. It provides protection against CSRF and very easy to use with asynchronous requests. Error handling is straightforward in Axios, there is no need to check for network errors as Axios handles it automatically and throws error, opposed to Fetch where developer needs to handle the case if any network has occured. HTTP interception is important if there is a need to change the HTTP request from the application to the server, in cases such as authentication, logging etc. This is possible with Axios and Fetch does not support this by default.

Due to it's simplicity and better error handling, it has been decided to use Axios to communicate with the server.

3.1.2.6 React Table

React Table is library that provides hooks creating tables and datagrids. It is a quite customisable library that is important for this use case, as the web application requires to represent data in table like structure. React Table is customisable in a way that, it does not render any HTML elements by itself, it renders whatever markup is passed to it.

3.1.2.7 React Hook Form

The application that will be build, is going to use forms quite often. It's the essential part of the application to interact with users. It's important to validate data that is passed in the forms. React Hook Form library provides these tasks and makes it easy to create good looking and UX friendly forms.

3.2 User Interface Design

This section is describing the planned user interface designs that will be realised as a result of the next chapter.

3.2.1 Login and registration pages

Login and registration pages are the first steps into the application. They will be presenting some forms with inputs, that users will have to fill in, in order to gain access to the application. The goal is to make the forms as easy as possible to use and make them look as good as possible.

The idea is to have a centralised component, with the required inputs in both pages. User is able to navigate between pages with the links that will be available in the form under the button to submit. In the registration page, user has to verify the password and in case passwords do not match, user should be notified of such error. On form submissions, the button should be disabled to prevent further submissions.

3.2.2 Home page

After successful login, home page is the main view user will be seeing. This page should reflect a structure of a dashboard. Centralised component, that will contain all the necessary information for user to interact with. Dropdown button at the top right corner will allow user to sign-out of their account.

Dashboard will contain a sidebar like navigation on the left side, to allow user to navigate through different views. The content of the views will be represented on the right side of the sidebar.

There will be 5 main views, user is able to see:

- Accounts
- Records
- Budgets
- Projects
- Settings

3.2.2.1 Accounts view

This view will be default view shown to user after login. It will have data regarding user's payment accounts used in the application. Accounts will be listed in order of creation. Accounts will have basic information like; name, type and balance, in each row they are represented. Each account will be expandable to see some of the transactions assigned to the payment account. Transactions in the expanded view will be represented in a table like structure to give brief information about the transaction.

A button on the top right of the view allows user to open the account creation modal. This modal will come up on the view in a form like structure, with inputs required to create an account. Submission of this form will start the process of sending required data for account creation to the server. After successful creation, user will be able to see the newly created account in the list.

In the expanded state of the account accordion, user will be able to see edit and delete icons, that will trigger editing and deleting of account accordingly. Editing will open the account modal, that will have data pre-filled for further edit.

3.2.2.2 Records view

This view is responsible for showing all transactions in an descending order of transaction date by default, in a table like structure. Each row will contain descriptive information about the specific transaction. At the header of table, labels for columns of the rows will be present to give information about the type of data. Rows of transactions will be paginated to group them for better visualisation. It will be possible to change pagination size at the bottom right corner of the view. Filtering is also important in this view, this functionality is possible by either using the columns in the table header or the button on top right view that opens a dropdown with different filtering options.

On the top right corner a button to open the transaction creation modal is placed. This modal will have inputs required for creation of a transaction, based on the type of transaction: expense, income, transfer; a button group at the top will set the inputs accordingly. Submission of this form will start process of sending data to server for creation of a transaction.

On click of a row, the modal will be loaded with transaction data pre-filled and in case user decides to update or delete the transaction, initial button for transaction creation is replaced by 2 buttons for update and delete of transaction.

3.2.2.3 Budgets view

In this view it is possible to see the budgets user has. Budgets will be represented like cards, that could be expanded to see the history of the budget and a bar like component will represent the percentage of money used according to set budget. Initial view of cards should contain basic information about the budget usage and dates.

To create a budget a button on top right corner opens the budget creation modal, that has relevant inputs. Budget modal has a submission button to trigger budget creation process.

Budget card should contain a small button to close or open budget accordingly. This button will represent the state of budget: open or closed. On the budget card hover, 2 more buttons should be visible to edit or delete budgets accordingly.

3.2.2.4 Projects view

Projects view is similar to accounts view, a list of projects with basic information about it, such as status, dates and name. Each list can be expanded to see the transactions assigned to the project. In the expanded state 2 icons representing edit and delete buttons, trigger editing and deleting of project accordingly.

Button on top right corner activates project modal for creation of projects. Submission of this modal triggers process of project creation and newly created project is added to the list of projects.

3.2.2.5 Settings view

This view will have a tab like button groups, which will open subviews. Categories will be listed in 1 of the tabs in a tree like structure.

Each category on hover shows 2 more buttons for editing and deleting of category. On click of category a subgroup on the right lists the subcategories. User can create categories by clicking the text like buttons at the end of each column, that opens a new input with confirmation button. Category icons can be updated in this view as well.

Chapter 4
Implementation

This chapter covers the implementation of the web and mobile applications and their deployment and user guide.

Development has been divided into 2 projects for frontend and mobile applications. For both implementations Git was used to version control the source code. The source code for both projects are stored on FIT Gitlab servers and also on my personal Github account to integrate it with Vercel - PaaS for CI/CD.

For the development of source code I have used Visual Studio Code[7]. It is an open-source code editor developed by Microsoft, with lots of extensions to be configurable according to users need.

4.1 Web application

This section will focus on the implementation of the web(frontend) application. Get a closer overview at how; authentication is done, components are implemented and data is stored and managed.

4.1.1 Project structure

Diagram 4.1 is a simple representation of the project structure.

```
xpensetracker-frontend
├── node_modules
├── src
│   ├── assets
│   ├── components
│   ├── pages
│   ├── services
│   ├── shared
│   │   ├── helpers
│   │   ├── hooks
│   │   ├── lib
│   │   ├── types
│   │   ├── constants.ts
│   │   ├── enums.ts
│   │   └── routes.ts
│   ├── store
│   ├── app.component.tsx
│   └── index.tsx
└── package.json
```

Figure 4.1 Web application source code structure

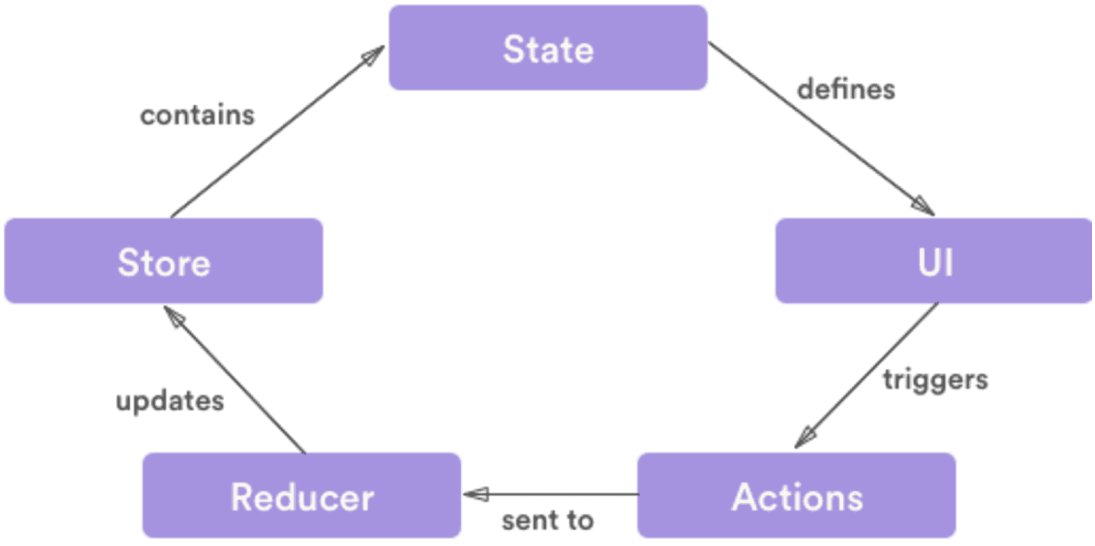

Figure 4.2 Redux state management representation (8)

4.1.2 State management

To implement the state management, Redux with Redux Toolkit has been used. Redux Toolkit provides helper functions to make usage of Redux simpler and use less amount of boilerplate code Redux requires to be setup. There are 3 important terms when using Redux with React:

- **Action**

  Javascript object with required type that informs reducer what happened to the state, it is not responsible to modify the state. Type of action is a string that describes the action and optional properties are information that are needed to update the state. An action is dispatched and then reducer handle the update of the state.

- **Reducer**

  Pure functions that takes the current value of state, performs the operations on it as instructed by the action, then outputs the new value of the state.

- **Store**

  Is the central source of states.

In the application there is a central store and each one of the entities from the domain model are a child store for better management of the state. In addition to those, there is also the auth store to hold basic state regarding authentication and ux store to store state related to component behaviour. All the child stores are combined in the main store using `combineReducers` that takes child stores as arguments to form the central store.

In each individual store the slice for the entity is created by `createSlice` which accepts an object of reducer functions, a slice name, and an initial state value, and automatically generates a slice reducer with corresponding action creators and action types.

Code listing 4.1 Redux Toolkit slice example for accounts

```
import {
  createSlice,
  isPending,
  isRejected,
  PayloadAction,
} from "@reduxjs/toolkit";
import { getRequest } from "shared/lib/api";
import customAsyncThunk from "store/config/custom-async-thunk";
import { IAccount } from "shared/types";
interface IAccountState {
  data: IAccount[] | [];
  loading: boolean;
  error: any;
}
const initialState: IAccountState = {
  data: [],
  loading: false,
  error: null,
};
export const getAccounts = customAsyncThunk(
  "accounts/getAll",
  async (_: void, { dispatch }) => {
    const { data } = await getRequest(`/accounts`);
    return data;
  }
);
const accountsSlice = createSlice({
  name: "accounts",
  initialState,
  reducers: {},
  extraReducers(builder) {
    builder
      .addCase(
        getAccounts.fulfilled,
        (state: IAccountState, { payload }:
          PayloadAction<IAccount[]>) => {
            state.data = payload;
            state.loading = false;
            state.error = null;
        }
      )
      .addMatcher(isPending, (state: IAccountState) => {
        state.loading = true;
        state.error = null;
      })
      .addMatcher(
        isRejected,
        (state: IAccountState, { payload }: PayloadAction<any>) => {
          state.loading = false;
          state.error = payload;
        }
      );
  },
});
export default accountsSlice.reducer;
```

To handle asynchronous calls to the API, there is a special function named `createAsyncThunk` provided by Redux Toolkit, that accepts a Redux action type string and a callback function that should return a promise. It generates promise lifecycle action types based on the action type prefix that you pass in, and returns a thunk action creator that will run the promise callback and dispatch the lifecycle actions based on the returned promise. To use this function with Typescript, I created a custom generic function called `customAsyncThunk`, that generates the relevant asynchronous action creator. As opposed to the usual behaviour, Redux Toolkit does not generate reducers automatically for `createAsyncThunk`. These needs to be handled manually under the `extraReducers` callback function inside the `createSlice` API. The thunk action creator function will have plain action creators for the pending, fulfilled, and rejected cases attached as nested fields that makes it easy to manipulate the UI for cases where a response from the API is expected.

State from the store is retrieved using the hooks exported from `shared/hooks/use-store.ts` which are custom hooks that using `useSelector` hook from Redux, retrieves specific state data. To access store in the components, it's just necessary to use the selector accordingly. Returned object contains `data`, `error`, `loading` and in some cases there can be more information stored in the object.

To dispatch action creators from components, it's required to use the `useDispatch` hook exported from `react-redux` library. `useDispatch` returns a reference to the dispatch function from the Redux store. By passing an action creator function to the returned reference from `useDispatch` it's possible to dispatch actions from the store.

4.1.3 Data storage and persist

In the default behaviour, on browser reload data in the Redux store is lost and it is reset to the the initial state of the store and the process of fetching user related data such as accounts, transactions and categories needs to follow. To prevent this data needs to be cached in the browser and store should be restored from the cached data. To solve this problem I used redux-persist library. Using this library it's also possible to keep the user logged in, for the scenarios where they just close the browser and no need to log the user back in, each time they reopen the browser.

Redux-persist offers different types of storage: session storage, local storage or async storage. By default this is set to local storage, which I decided to keep it as it is. The setup is straightforward: a Javascript object with `key` and `storage` values is passed to the `persistReducer` interface imported from redux-persist package together with the reducers that needs to be persisted. Returned `persistedReducer` are passed to the `configureStore` interface from redux-toolkit in the reducers key. Then the returned store from `configureStore` is passed to `persistStore` interface which is then used to wrap the parent component in the application.

Access and refresh tokens received from the API after successful login are stored as a browser cookie, to be used for requests that require authentication. Tokens are saved as an object named `user` in the local storage that is handled by `token.service.ts`. This service is responsible to save the tokens and when there is a need to refresh the access token, returns the refresh token from the object saved in the local storage.

4.1.4 Authentication

Almost all of the endpoints provided by the API, it is required to have access to relevant resource. To be able to use the resources correctly the token received in the login process is saved in the local storage and then it is set as `Bearer ${token}` in the Authorization header of the axios instance that is used as client. For the cases where the access token expires, there are interceptors set in the `interceptors.service.ts`. The interceptors service sets the Authorization headers for request using the token retrieved from the token service in case it is not set and if the

response fails it makes a request to the API to refresh the token and then uses the new access token in the axios client and updates the token in the browser cookie by calling the token service `updateLocalAccessToken` method. It also sets the retry header to true to retry the request that previously failed.

4.1.5 Components

React application is based on components, everything seen in the browser can be defined as a component. Components in a simple way get some data as props and depending on the props, they render HTML elements on the page. There are 2 types of components according to React handbook: stateful and stateless componenets. The difference is that the stateful components have their internal state using the `useState` hook. It is preffered to have as many stateless components as possible. The reason is that, they are easier to manage and less time spent for debugging.

Components are divided into 2 main folders: `pages` and `components`. Components in the pages folder are just components for individual pages of entities and the settings page which contain categories, thus they are separated from the components folder to make their purpose more clear. Components in the components folder represent individual components that are rendered in various places and pages.

Navigation in between pages is done with `react-router-dom` library. Based on the paths that are passed to the `Route` component imported from the library, relevant page is shown. For protected pages that only authenticated users can access use a custom component called `PrivateRoute` for routing. This component is checking if there is a token in the local storage and checks if this token is valid. If valid user is allowed to access the pages that were requested, if not valid user is redirected to login page.

Component pages access the relevant data from the store using selector hooks defined in `shared/hooks/use-store.ts` and present data accordingly. Different pages use different components to show data to the user:

- Accounts - Individual accounts are using `Accordion` component imported from `mui` library, to visualise data. Each of the accordions have `CustomTable` component that shows limited number of transactions.
- Records - Transactions are listed in a table structure using custom HOC `CustomTable` that receives props from `React Table` hook and fills up the data into JSX elements accordingly. Filters and pagination
- Budgets - Card components from `mui` represent each budget. Cards have a `LinearProgress` component that represents percentage of used amount and more information such as date and spent amount. Each card can be expanded to see history using stepper component.
- Projects - Accordions represent each project. Expanded accordion shows `CustomTable` component with data of transactions.
- Settings - `TabList` component handles subviews in settings. Currently only categories are visible, which are rendered by `List` component.
- Login and registration - `Card` component wraps controlled inputs from `React Hook Form` library.

4.1.5.1 CategoryDropdown

This is a custom made component, that aims to present categories in a tree like structure as a selector for choosing a category in various cases. Meaning on the selector view, only the

categories that have same parent or no parent should be visible. There was no ready solution from my research that would fulfill the requirement, thus it had to be created from scratch.

To create this component, as a first step, data that would be passed to the component needed to be transformed into correct format. API was returning an array of categories, without any hierarchy that would make the consumption easier. Thus, once the data was received from API, it was reordered through the helper functions to create the tree structure from them, where in the first level there would be only categories without any parent id set. This transformed structure was then set in the store for further use.

The component receives some props from the component that renders it and starting from the first level of the items, it loops through each item and creates the menu's for that level. This way the HTML elements are created but they are hidden unless the menu currently being generated is set to active by the parent component that renders the selector. If the current level is set to active, then with the help of css that level is visible to the user.

4.1.5.2 Modal

Modal is a component used for form representation, that contain inputs, selectors or date pickers according to the type of modal. Modal wrapper contains the header part and the submission/edit-delete buttons depending on why the modal has been activated. Content of it is passed from the page it has been opened from. The content of the modal is set by the content that is passed inside the wrapper modal at the component of render. For this case modals have been divided by their type of content as `RecordModal`, `AccountModal`, `BudgetModal` and `ProjectModal`. Common inputs for modals are located in `src/components/modal/inputs`. These are custom inputs using `React Hook Form`, that can be controlled and they are ready to be used by the modals for various cases.

Modals except the `RecordModal` are just simple forms, with couple of inputs. However `RecordModal` is quite complex, as technically it is responsible for 3 types of forms: income, expense and transfer. I choose to do so, as I was inspired by Wallet's modal for transaction creation, I decided to follow a similar way and combined them into 1.

On this modal, it's possible to switch between types of transaction by the button group placed at the top of modal. The switch, changes the set type of transaction and shows the correct inputs that are needed. Which I think is quite user friendly, although improvements are always possible.

Some features of this modal that worth pointing out:

- When switching between income and expense transaction, the sign of the amount input is reflected accordingly.
- Inputs are filtered out and hidden, if the selected transaction type does not require those inputs for form submission.
- If the transaction type is transfer, the two account selectors are dependent of each other. Namely the account selected on the left selector, cannot be selected in the right selector. Giving user clear choices on how to make the transfer.
- In the expense type of transaction, user is able to split the transaction into multiple sub-transactions. This reveals a row with a category and amount fields. User can add as many subtransactions, no specific limit provided. The only condition to add a new row of sub-transaction is to fill all the rows of subtransactions by setting the amound and choosing a category.
- While splitting transactions, the total amount and subtransaction amounts update according to user interaction. Meaning if the total amount set in the beginning is less than the total amount of subtransactions, total amount of parent transaction updates while user enters a value above the total amount.

- If user removes a subtransaction and tries to submit the form with the total amount exceeding the amount of subtransactions, user is warned with a message of left amount to be used for splitting.

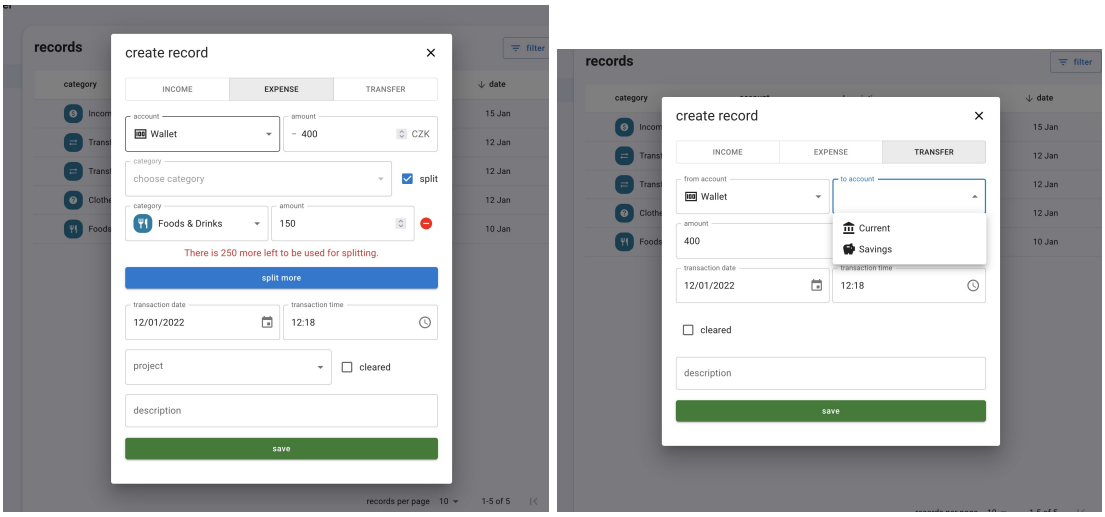

Figure 4.3 Transaction Modal

Handling of buttons are passed from the main page view, to modals based on the choice of action. If the modal is in creation mode, there is a single button at the bottom of the modal to submit the form, to create the relevant record. If modal is in edit mode, data is prefilled to the inputs and there are 2 buttons at the bottom of modal: delete and save. That trigger processes of delete and update of record accordingly.

4.1.5.3 CustomTable

The purpose of this component is to present mainly transactions in a table like structure, that will handle pagination and filtering of the rows. It's an important feature as most of the applications do not have a good filtering option for transactions.

This component took a fair bit of time to get into realisation, due to poor choice of libraries in the process, which were not supporting filtering in a way that me and my supervisor expected it would work.

The first component `DataGrid` from `MUI` library used for table representation, was limited in the functionalities it offered regarding filtering in multiple columns and as we decided to have an external filter with a button to show types of filters for user to use. This wasn't possible as it only supported filters from the columns visible and the only way to make it work was to create my own filtering, which was not my goal from the start as I wanted to use already made solutions.

The second try was with `Table` component from `antd` library. This was potentially a solution to our case, but once I started working on it I realised it's not possible to transform the data passed to the component as it was possible with the previous solution. So this was also not a good choice in the end.

The third choice was using `React Table`. It was the perfect solution as it didn't require too much of modification of the data passed to the hook it provides and is quite customisable in many ways. Supports both of the types of filtering required in the application and subtransactions could be flattened if a filter is applied to see the individual subtransaction on the table.

The implementation is quite straightforward; data and the types of filters that will be presented in the table are passed to the hook `useTable` provided by `react-table` and the returned

object contains all the filters, filter setting function, pagination setting function and the data to be displayed. This object is then passed to `CustomTable` component which handles render of rows according to the given format of HTML.

Returned filters and filter functions can be used by the inputs in the `Filter` component that contain filters of different kind. Applied filters are reflected by the `useTable` hook to the returned filtered data.

4.1.6 Tools

Some other tools and libraries used in the implementation are:

- **vercel**

  It is a PaaS product, that allows deployment and hosting of different kinds of applications, specifically built to work well with React applications. Using Vercel is quite easy as the only things needs to be done is to create the project in Vercel admin and just push the code changes and everything gets deployed and served for free. Current implementation is also hosted on vercel at `https://xpensetracker.vercel.app` [9]

- **create-react-app**

  A library that makes creation of react project easy by just running a single command [10].

- **redux-persist**

  Library that allows to persist Redux store in the local storage of the browser [11].

- **date-fns**

  It is a date library that provides simple and easy ways to manipulate Javascript dates in the browser [12].

- **antd**

  A library that contains a set of high quality components and demos for building interactive user interfaces [13].

- **history**

  Library that allows to easily manage session history anywhere JavaScript runs [14].

- **clsx**

  A utility for constructing `className` strings for JSX elements conditionally [15].

- **husky**

  Library that allows developer to lint commit messages, run tests, lint code etc. when using git commit or push [16].

- **jsonwebtoken**

  An implementation of JSON Web Tokens in Javascript, that allows to handle jwt tokens [17].

- **redux-logger**

  A Javascript library that allows logging Redux actions for easier debugging [18].

4.2 Mobile application implementation

In the beginning of the thesis together with my supervisor, it was agreed that the main priority would be web application, as that one was decided to be redesigned and implemented from scratch with new technologies. According to the plan, once the web application is in a robust and bug free state, the implementation for the mobile application could start. I had an estimation that by end of November 2021, web application would be finalised and for the rest of the time I would have enough time to implement a small prototype of the mobile application. However due to unforseen problems during implementation of filters, splitting transactions and bugs on the API side that needed to be resolved. Those problems and bugs slowed down the development more and this resulted with not being on time with the set goals. Due to this mobile application has not been implemented.

Chapter 5
Conclusions

The goal of this Bachelor Thesis was to create web and mobile applications based on the API created by Jamal Azizov's Bachelor Thesis work. To implement the functionalities the API is providing based on the research and the requirements that were set in the analysis chapter.

The theoretical parts of the work has been fulfilled successfuly. Namely, analysis of current solutions and requirements based on the analysis, design of the solution planned and finally the implementation of the proposed solution.

However the practical parts were not completed fully as per the assignment requirements. Missing parts in the implementation include recurring transactions, sharing payment account and mobile application implementation. The reasoning behind that is the lack of time with such a big scope of requirements and lack of planning ahead regarding technologies used were the main reasons for not being able to fulfill the practical part requirements fully.

The latest version of the web application can be accessed on `https://xpensetracker.vercel.app`.

5.1 Further improvements

First thing to improve would be to implement the missing functionalities that were proposed, however not finalised in this work. A mobile application is a must in todays standards for an application that requires many on-the-go updates. Supporting currency exchanges could be a good addition. Also localisation of the application might attract more users worldwide.

Appendix A

Web application's screenshots

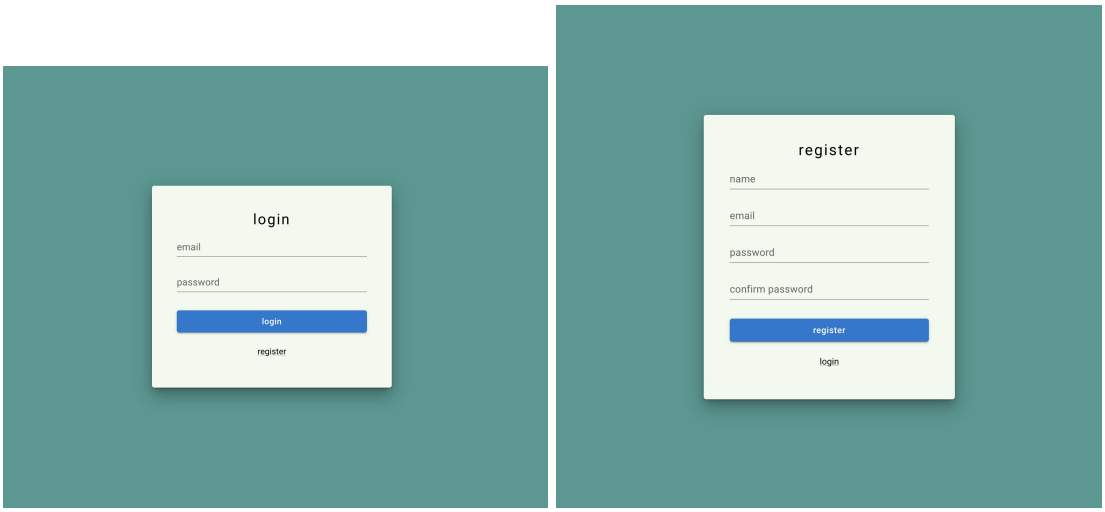

Figure A.1 Login and register view

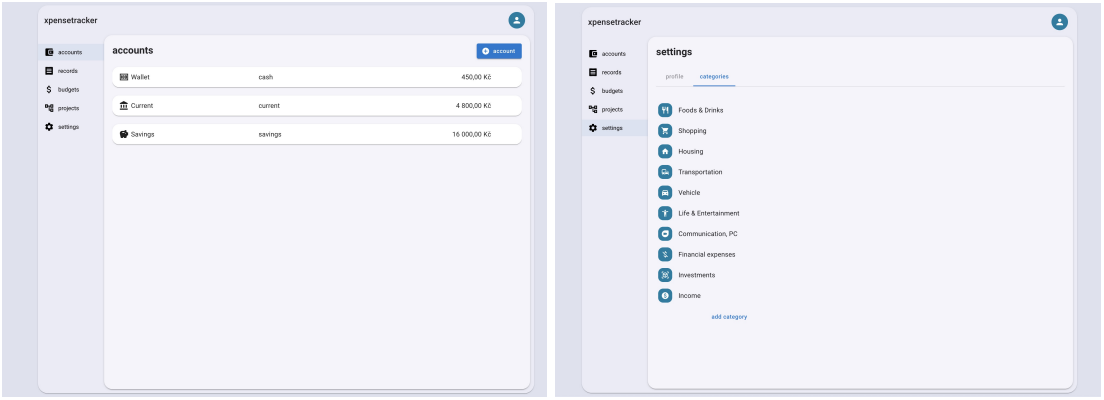

Figure A.2 Accounts and categories view

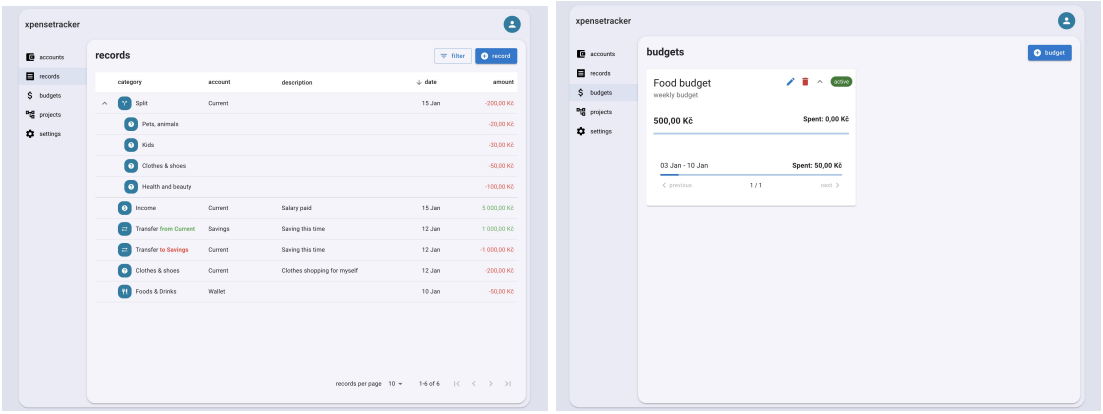

Figure A.3 Records and budgets view

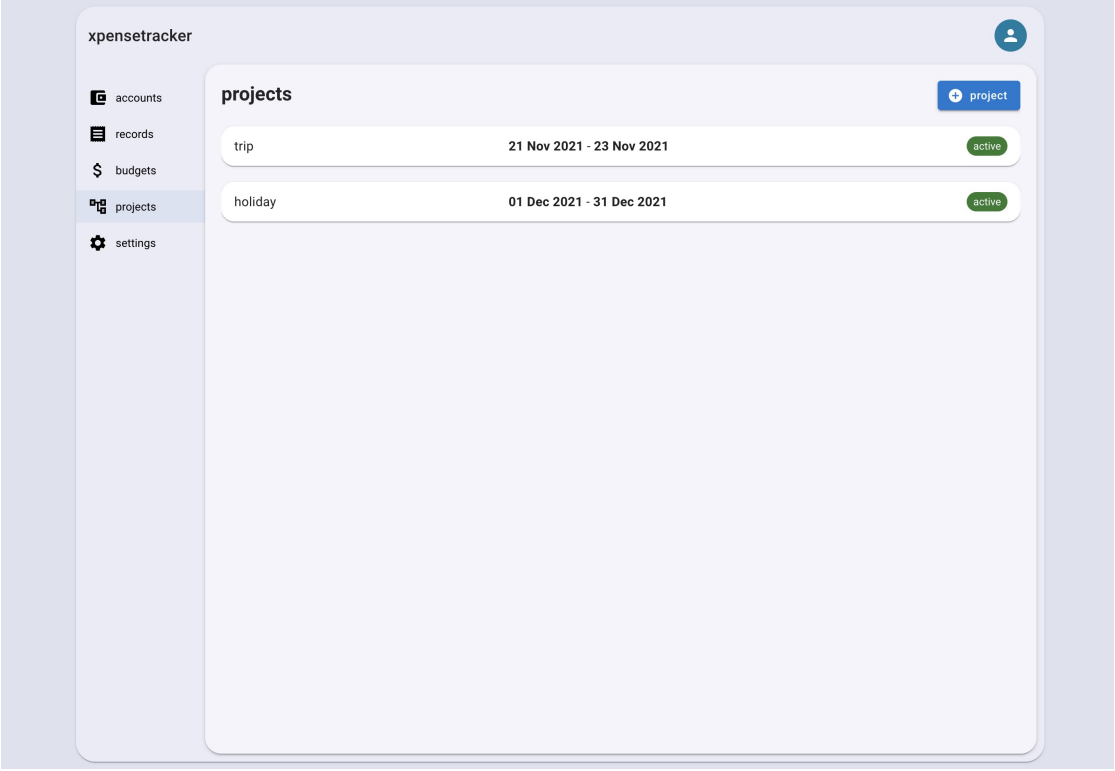

Figure A.4 Projects view

Bibliography

1. *Separation of Concerns [online].* [N.d.]. Available also from: https://en.wikipedia.org/wiki/Separation_of_concerns. [cit. 2022-01-28].
2. *React vs Angular vs Vue: Which framework to choose in 2022 [online].* [N.d.]. Available also from: https://2muchcoffee.com/blog/react-vs-angular-vs-vue-which-framework-to-choose-in-2022/. [cit. 2022-01-28].
3. DAN ABRAMOV AND THE REDUX DOCUMENTATION AUTHORS. *React Redux [online].* [N.d.]. Available also from: https://react-redux.js.org/. [cit. 2022-01-21].
4. *MUI [online].* [N.d.]. Available also from: https://mui.com/. [cit. 2022-01-28].
5. *Google Material Design [online].* [N.d.]. Available also from: https://material.io/design/introduction. [cit. 2022-01-28].
6. MATT ZABRISKIE, John Jakob "Jake" Sarjeant. *Axios [online].* [N.d.]. Available also from: https://axios-http.com/. [cit. 2022-01-28].
7. *Visual Studio Code [online].* [N.d.]. Available also from: https://code.visualstudio.com/. [cit. 2022-01-28].
8. *Redux state management [online].* [N.d.]. Available also from: https://www.loginradius.com/blog/async/react-state-management. [cit. 2022-01-28].
9. *vercel [online].* [N.d.]. Available also from: https://vercel.com. [cit. 2022-01-28].
10. *create-react-app [online].* [N.d.]. Available also from: https://create-react-app.dev/. [cit. 2022-01-28].
11. *redux-persist [online].* [N.d.]. Available also from: https://www.npmjs.com/package/redux-persist. [cit. 2022-01-28].
12. *date-fns [online].* [N.d.]. Available also from: https://www.npmjs.com/package/date-fns. [cit. 2022-01-28].
13. *antd [online].* [N.d.]. Available also from: https://ant.design. [cit. 2022-01-28].
14. *history [online].* [N.d.]. Available also from: https://www.npmjs.com/package/history. [cit. 2022-01-28].
15. *clsx [online].* [N.d.]. Available also from: https://www.npmjs.com/package/clsx. [cit. 2022-01-28].
16. *husky [online].* [N.d.]. Available also from: https://typicode.github.io/husky/#/. [cit. 2022-01-28].
17. *jsonwebtoken [online].* [N.d.]. Available also from: https://www.npmjs.com/package/jsonwebtoken. [cit. 2022-01-28].

18. *redux-logger [online].* [N.d.]. Available also from: `https://www.npmjs.com/package/redux-logger`. [cit. 2022-01-28].

Content of enclosed media

- `readme.txt` ..... Description of contents of enclosed media
- `src` ..... Source codes
  - `web-app` ..... Source code for web application
  - `thesis` ..... Source code of the thesis in LaTeX format
- `text` ..... Text work
  - `thesis.pdf` ..... Thesis in PDF format