



## Assignment of bachelor's thesis

<b>Title:</b>	XpenseTracker Client – client part for personal finance manager
<b>Student:</b>	Amir Qamili
<b>Supervisor:</b>	Ing. Zdeněk Rybola, Ph.D.
<b>Study program:</b>	Informatics
<b>Branch / specialization:</b>	Software Engineering
<b>Department:</b>	Department of Software Engineering
<b>Validity:</b>	until the end of summer 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
-



Bachelor's thesis

# **XPENSETRACKER CLIENT**

**Amir Qamili**

Faculty of Information Technology  
Department of Software Engineering  
Supervisor: Ing. Zdeněk Rybala, Ph.D.  
January 5, 2023

Czech Technical University in Prague  
Faculty of Information Technology

© 2023 Amir Qamili. All rights reserved.

*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, 2023.

# Contents

<b>Acknowledgments</b>	<b>vii</b>
<b>Declaration</b>	<b>viii</b>
<b>Abstract</b>	<b>ix</b>
<b>Acronyms</b>	<b>x</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Motivation and objectives . . . . .	1
1.2 Roadmap . . . . .	1
<b>2 Analysis</b>	<b>3</b>
2.1 Current implementation . . . . .	3
2.1.1 Architecture . . . . .	3
2.1.2 Functionalities . . . . .	4
2.2 XpenseTracker API . . . . .	6
2.3 Existing solutions . . . . .	7
2.3.1 Wallet . . . . .	8
2.3.2 Spendee . . . . .	8
2.3.3 MoneyWiz 3 . . . . .	9
2.3.4 Financisto . . . . .	9
2.4 Requirements . . . . .	9
2.4.1 Functional requirements . . . . .	11
2.4.2 Non-functional requirements . . . . .	14
<b>3 Design</b>	<b>15</b>
3.1 UI . . . . .	15
3.1.1 Patterns . . . . .	16
3.1.2 Wireframes . . . . .	19
3.2 Technologies . . . . .	25
3.2.1 Languages . . . . .	25
3.2.2 Libraries . . . . .	26
3.3 Architecture . . . . .	30
3.3.1 MVC architecture . . . . .	30
3.3.2 Web . . . . .	30
3.3.3 Mobile . . . . .	32
<b>4 Implementation</b>	<b>33</b>
4.1 Project management . . . . .	33
4.2 Tools and libraries . . . . .	33
4.3 Implementation details . . . . .	34
4.3.1 Authentication . . . . .	34
4.3.2 State management . . . . .	35

4.3.3	Data layer . . . . .	35
4.3.4	Views . . . . .	36
4.3.5	Forms . . . . .	41
4.3.6	Charts . . . . .	45
<b>5</b>	<b>Testing</b> . . . . .	<b>47</b>
5.1	Manual testing . . . . .	47
5.2	Usability testing . . . . .	47
5.2.1	Process . . . . .	47
5.2.2	Outcomes . . . . .	48
5.2.3	Conclusions . . . . .	49
<b>6</b>	<b>Conclusions</b> . . . . .	<b>51</b>
<b>A</b>	<b>Installation guide</b> . . . . .	<b>55</b>
A.1	Requirements . . . . .	55
A.2	Web . . . . .	55
A.3	Mobile . . . . .	55
A.3.1	Requirements . . . . .	55
<b>B</b>	<b>Deployment guide</b> . . . . .	<b>57</b>
B.1	Web . . . . .	57
B.2	Mobile . . . . .	57
	<b>Content of enclosed media</b> . . . . .	<b>59</b>

## List of Figures

2.1	Images from the current implementation of XpenseTracker . . . . .	5
2.2	Domain model by Jamaladdin Azizov for XpenseTracker API [2, p. 8] . . . . .	6
2.3	Wallet - Web application . . . . .	8
2.4	Spendee - iOS application . . . . .	9
2.5	MoneyWiz 3 - iOS application . . . . .	10
2.6	Financisto - Android application . . . . .	10
3.1	An example of bad UI, choice of bad color palette and text colors makes it hard to read. Source: <a href="https://www.mockplus.com/blog/post/bad-web-design">https://www.mockplus.com/blog/post/bad-web-design</a> . . .	16
3.2	Navigation examples on web and mobile applications . . . . .	18
3.3	Modal use cases . . . . .	18
3.4	Registration and login view wireframes . . . . .	20
3.5	Web dashboard view . . . . .	21
3.6	Accounts views for web and mobile . . . . .	21
3.7	Records and record creation modal views for web and mobile . . . . .	22
3.8	Categories and templates views . . . . .	23
3.9	Budgets views for web and mobile . . . . .	24
3.10	Projects views for web and mobile . . . . .	24
3.11	Most commonly used frameworks in 2021 according to survey by stackoverflow: .	28
3.12	Comparison of traditional server-side and client-side rendered architectures [26, p. 5, 6] . . . . .	31
3.13	Planned architecture of the client side applications . . . . .	31
4.1	Accounts API slice . . . . .	37
4.2	Login and registration views on the web and mobile application respectively . . .	37
4.3	Dashboard view on the web app . . . . .	38
4.4	Accounts and projects view on the web app . . . . .	38
4.5	Accounts and projects view on the mobile app . . . . .	39
4.6	Categories and templates view on the web . . . . .	39
4.7	Records view on the web and mobile respectively . . . . .	40
4.8	Budgets view on the web and mobile respectively . . . . .	41
4.9	Form implementations on web . . . . .	44
4.10	Transaction splitting implementation . . . . .	45
4.11	Charts as seen on the dashboard view . . . . .	46

## List of Tables

2.1 Comparison table for chosen finance tracking applications on their free plans . . . 11

## List of code listings



*I would like to thank everyone who helped me during my studies and my thesis, specifically my supervisor Ing. Zdeněk Rybala, 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 January 5, 2023 .....

## Abstract

This thesis work intends to create web and mobile clients for finance tracking with a focus on user interface design. To fulfil this goal, software development life cycle methods should be applied accordingly. Meaning analysis of the already existing solutions and definition of the requirements to achieve a successful product. The input from the analysis is used to design user interfaces using best practices in modern user experience patterns with the help of modern technologies. The implementation of web and mobile clients follows the design prototypes. Finalised applications are tested using the usability testing method to receive feedback from real-life users to assess the quality. Testing results verify the achievement of this thesis goal with overall positive feedback from users.

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

## Abstrakt

Cílem této diplomové práce je vytvořit webové a mobilní klienty pro sledování financí se zaměřením na návrh uživatelského rozhraní. Ke splnění tohoto cíle je třeba odpovídajícím způsobem aplikovat metody životního cyklu vývoje softwaru. Což znamená analýzu již existujících řešení a definici požadavků pro dosažení úspěšného produktu. Vstupy z analýzy se použijí k návrhu uživatelského rozhraní s využitím osvědčených postupů moderních vzorů uživatelského prostředí za pomoci moderních technologií. Implementace webových a mobilních klientů se řídí návrhy prototypů. Finalizované aplikace se testují metodou testování použitelnosti, aby se získala zpětná vazba od reálných uživatelů a posoudila se jejich kvalita. Výsledky testování ověřují dosažení cíle této práce s celkově pozitivní zpětnou vazbou od uživatelů.

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

## Acronyms

API	Application Programming Interface
CRUD	Create, Read, Update, Delete
CSS	Cascading Style Sheets
DOM	Document Object Model
DRY	Don't Repeat Yourself
DTO	Data Transfer Object
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
IDE	Integrated Development Environment
MVC	Model View Controller
MVVM	Model View ViewModel
RTK	Redux Toolkit
SPA	Single Page Application
SDLC	Software Development Life Cycle
UI	User Interface
URL	Uniform Resource Locator
UX	User Experience
VCS	Version Control System

# Introduction

Knowledge is power. Simply knowing something can change behaviour. As money is essential in our lives, people want to know where expenses happen. Explicitly speaking, knowing the financial expenses will impact how the money is used.

With the soaring energy and food prices, money has become even more valuable than a couple of years ago. In our day-to-day lives, we purchase many things over time, whether it be a pack of gum, ice cream on a summer day or groceries. Tracking these expenses could help control unnecessary costs and save money for more significant purchases we cannot usually save enough.

Personal finance is keeping a daily record of income and expenses. This process's advantage is controlling our expenses and monitoring cash flow, as we cannot control our salaries. The disadvantage is the extra continuous work required to note down these updates in our wallet daily; this is even harder with the traditional paper bookkeeping way. However, today, managing finances is much easier with the help of mobile applications. When a person makes a payment at the supermarket, they can update their accounts in the application with the latest expense that happened moments ago. Even better, some applications offer direct synchronisation with their current account in the bank without the need for manual updates from the users' side.

## 1.1 Motivation and objectives

In the past, XpenseTracker has been developed with the supervision of Ing. Zdeněk Rybola, PhD, in the subjects of BIE-SP1 and BIE-SP2. I was also part of the team that developed this application, with other members being: Beksultan Baatyrbekov, Ilya Ryabukin, Jamaladdin Azizov, Nilay Baranwal, and Rishabh Malik.

This application is open to improvements, and the motivation of this thesis topic is to make that happen. Specifically, I should address the functionalities missing from the current implementation, focusing on the client side and how the user interface could be improved for a better user experience.

## 1.2 Roadmap

The process will follow the Software Development Life Cycle (SDLC) to achieve the final product in a systematic and disciplined manner. It comprises several clearly defined and distinct work phases to plan for, design, build, test, and deliver information systems[1].

The first step of SDLC is requirement gathering and analysis. Chapter 2 is about the analysis of the current solution and a comparison of the existing solutions is needed to understand the best direction to go with the development. Finally, utilising this analysis, together with my

supervisor Ing. Zdeněk Rybala, PhD, the requirements to be implemented within this thesis will be determined.

The next step of SDLC is the design part of the thesis. Chapter 3 - design, will focus on how the application should look based on the current standards used throughout various applications. It should contain wireframes for better visualisation to the reader. This chapter will also cover the technologies to be used and the reasoning behind selecting them.

In the implementation chapter 4, will help the reader understand how the client side of the application is working. Guide the reader with how components are connected and different libraries are used within the code.

In testing chapter 5, the quality of the implemented applications will be assessed by briefly discussing the actions taken to achieve the most quality product possible.

Finally, the conclusion chapter 6, is for summarising what has been achieved in the scope of the thesis as well as steps taken for the achievement.

## Chapter 2

# Analysis

The analysis is essential in software development for understanding the problem and determining the requirements to solve the problem. Doing it successfully can save a significant amount of time, which in the business world means money. The focus of this chapter is to define the requirements of the work that will be done as a result of the analysis of this chapter. The analysis has multiple steps; with each step, we will better understand what kind of application we aim to achieve.

As the main idea of the thesis is to improve the existing solution of XpenseTracker, it is necessary to understand what the current implementation is capable of, why the changes are needed and how the working solution could be improved.

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 any application fulfils the requirements from the previous analysis of the current implementation of XpenseTracker.

The following step is to analyse the backend side of XpenseTracker, which Jamaladdin Azizov has implemented in his Bachelor Thesis [2]. The client side will be consuming this API, the applications developed at the end of the thesis, and we need to understand the API's state clearly. As we can implement at most the same amount of functionality it provides, the analysis in this step will give the final requirements and features we can 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. The final requirements achieved in this step will guide us on how the UI should look and how to realise the requirements and the goals we have set.

### 2.1 Current implementation

The existing solution was developed in SP1 and SP2 subjects under the supervision of Ing. Zdenek Rybala, PhD, as a result of the collaboration of multiple students: Amir Qamili, Beksultan Baatyrbekov, Ilya Ryabukin, Jamaladdin Azizov, Nilay Baranwal, Rishabh Malik. The goal was to create an application that would allow users to track their expenses.

#### 2.1.1 Architecture

Modern applications apply a separation of concerns (SoC)[3] design pattern, meaning each part deals with a particular problem.[4] This approach has many benefits, such as more freedom of design, deployment or usage. When the concerns are separated, there is a better opportunity for modular upgrades, reuse and independent development. However, this approach has its

associated execution penalty, as separation of concerns is a form of abstraction, which means adding additional code interfaces, which results in more code being executed.

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

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

One of the requirements of this thesis is to have a mobile client that will be consuming the API. Instead of separating the mobile application's codebase, it was decided to separate the two parts of the current implementation.

### 2.1.2 Functionalities

The current implementation is limited by its functionalities. It satisfies the most basic requirements that were planned. Figure 2.1 shows some images from the current implementation of XpenseTracker. Functionalities offered in the existing solution include:

#### Authentication

- Registration allows users to create their accounts.
- The user is authenticated using its credentials to access the web application.

#### Accounts

- The user is able to do CRUD operations for payment accounts.
- Each payment account lists its last ten transactions.
- Each type of payment account has its icon.

#### Categories

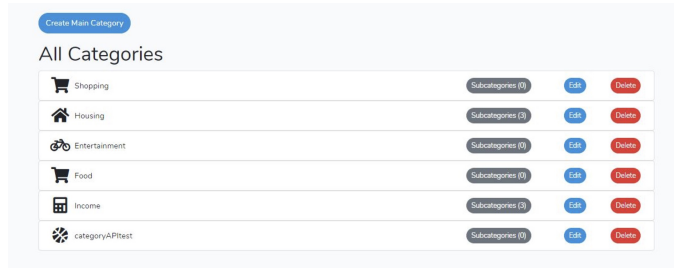
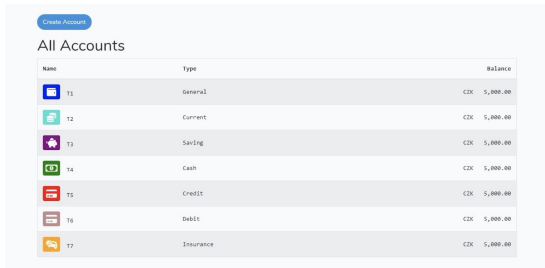
- The user is able to do CRUD operations for categories.
- Each category can have infinite child categories.
- The number of subcategories is shown in the UI.
- Preselected icons represent categories.

#### Transactions

- The user is able to do CRUD operations on transactions.
- UI provides basic filtering by account, category, state, type and date range functionality.
- Basic filtering and pagination allow the user to interact with multiple transactions.
- Transfer type of transaction is supported.

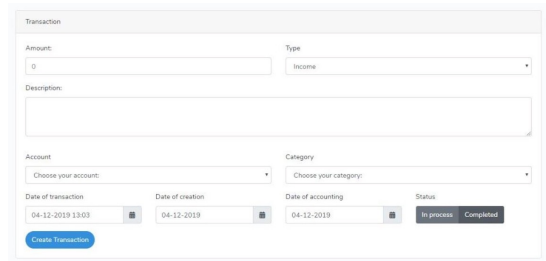
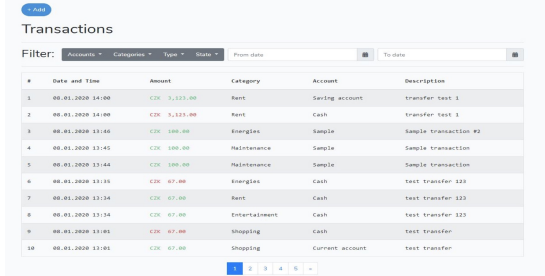
As seen above, the existing solution is just a basic application that enables users to do simple expense tracking and finance management. It is far from being a good candidate for actual users, as it does not offer anything unique to distinguish it from other applications which are much more advanced.





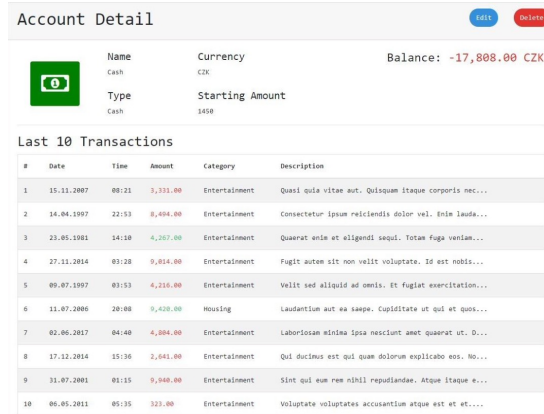
(a) Accounts

(b) Categories



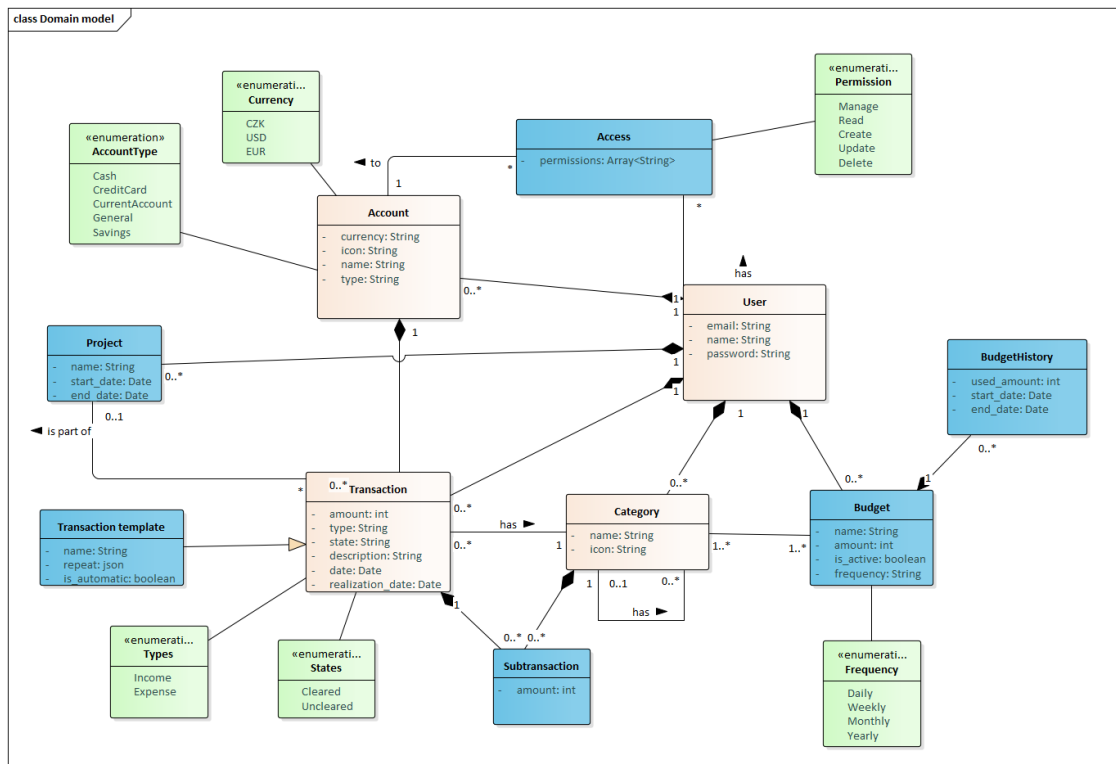
(c) Transactions

(d) Create transaction



(e) Account detail

■ Figure 2.1 Images from the current implementation of XpenseTracker



■ **Figure 2.2** Domain model by Jamaladdin Azizov for XpenseTracker API [2, p. 8]

## 2.2 XpenseTracker API

The reimplemented API has been improved and added new functionalities by Jamaladdin Azizov due to his Bachelor Thesis work.[2] Domain model representation from his work can be seen in Figure 2.2.

As the applications created in this thesis will be consuming this API, it is crucial to analyse it for further definition of requirements on the client side of the applications. The documentation for API is provided at <https://xpensetracker-api.herokuapp.com/api>.

List of functionalities based on endpoints and documentation:

### Authentication

- API provides endpoints for registering and login in.
- As it uses token-based authentication, it is also possible to refresh the token.

### Accounts

- This set of resources provides relevant endpoints to fetch payment accounts, ability to update, delete and add new accounts.
- Retrieve list of transactions for each account.
- It provides endpoints to share an account with other users and update and retrieve the list of users with whom the account has been shared.

## Categories

- Management of default categories for users is possible under this set of endpoints. It is possible to add, update and delete categories accordingly.
- Option to add subcategories with an unlimited hierarchy.

## Transactions

- API provides endpoints to fetch, create, update and delete transactions under this set of resources.
- It is also possible to split a transaction into multiple transactions and fetch, update and delete them individually using the parent transaction id.

## Budgets

- Endpoints under this set of resources provide simple CRUD operations for budgets.
- The ability to close or open budgets.
- Historical data for budgets can be fetched.

## Projects

- Endpoints under this set of resources provide simple CRUD operations for projects.
- Projects can be activated and deactivated accordingly.

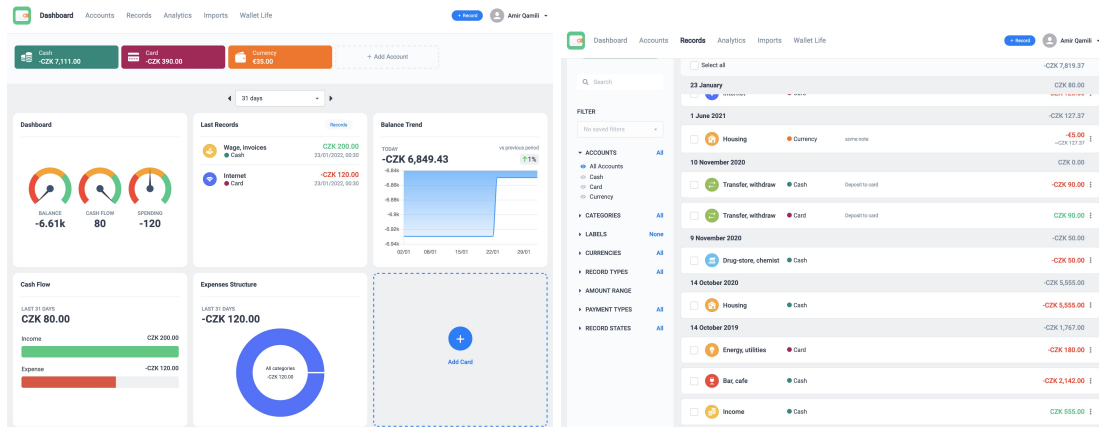
## Recurring transactions

- API provides a set of endpoints that allows CRUD operations for templates.
- Templates are used based on the data set to be either recurring and handled in the backend for the creation of transactions periodically or as a simple template to fill data while creating a transaction.

## 2.3 Existing solutions

The idea of tracking expenses is nothing new, and one of the ways to improve such an existing concept is to review the work that other people have already done. The following will discuss the research on the existing solutions in the market. An analysis of these applications that offer features similar to the goals of this thesis will help us understand where to focus and the essential features that are commonly missing in these applications. It will also help us set the functional requirements for a user-friendly and attractive UI.

When choosing the applications to analyse, I chose the ones previously used by myself or the people around me, including my supervisor, who had excellent ideas for an ideal application based on the problems he faced while using similar applications. This way, we could find solutions to actual problems users might be experiencing and bring new ideas about missing features in previously used applications.



■ Figure 2.3 Wallet - Web application

### 2.3.1 Wallet

It is a Czech product that offers Web, iOS and Android applications.[5] Only several finance tracking applications offer a web UI alongside mobile applications, as only a few people would like to use the web application daily. Having all applications available on all platforms is a big plus for Wallet compared to other options.

Wallet has a good and straightforward UI for essential financing functions such as managing accounts, transactions and categories. However, a maximum of three payment accounts for free users is limited. Categories are set by default on registration, and their modifications are limited; a maximum of two levels of subcategories with exclusion to add more subcategories to the main category. It offers budget functionality to overview expenses on specific periods by selected categories. With friendly and straightforward statistical graphs, it is easy to visualise expenditures on a broader scale. Template functionality makes creating similar repeating recordings easier by creating a blueprint for shared information. One of the missing features, which is valuable, is the ability to split a transaction. This feature is important because people usually end up recording multi-category shopping under one general category, like buying groceries and toiletries in one go.

With the current version, Wallet is one of the most successful applications in the market. Feature-wise quite rich and, from a UX perspective, very user-friendly. On a personal level, it is my favourite application.

### 2.3.2 Spende

Another Czech product developed in Prague in 2013 has Web, Android and iOS applications.[6] However, I could not use the web application multiple times I tried. Some problems with their application, I believe. On visual aspects, the application is also well done and deserves praise for good graphics. User experience is also good in general, and it is clear what you need to click while using the applications. The graphics for statistics related to expenses are also nice. Regardless, Spende is limited in functionalities compared to Wallet and an ideal application I had in mind.

In the free plan, you can only create one payment account, add transactions, and make one budget to track your expenses periodically. Your ability to use the application is quite limited. In Wallet, you are at least allowed to have three payment accounts without the need to pay. It also does not offer any transaction filtering other than a search field. On the other hand, I would expect some of the missing functionalities in Wallet to be available here. Unfortunately, this is



■ Figure 2.4 Spendee - iOS application

the same case in Spendee too.

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

### 2.3.3 MoneyWiz 3

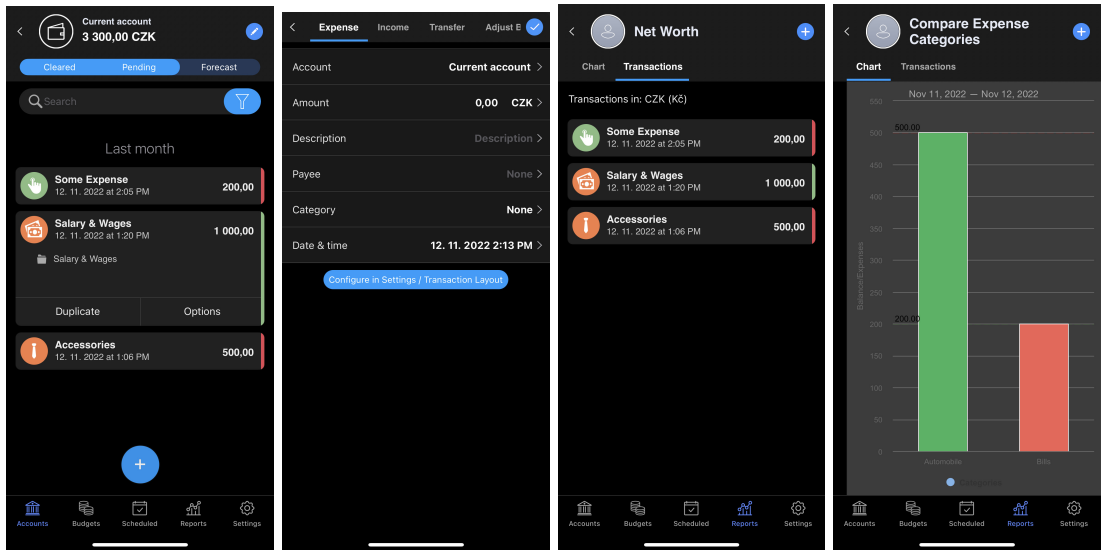
Initially, a macOS application is available on other platforms such as Windows, Android, and iOS.[7] It is rich in features it provides, and the ability to have a desktop application could be interesting for some users. One thing I learned from this application is that having all the features possible in one place can make the usage of the application quite hard. MoneyWiz 3 has almost all the features you could ask for in finance tracking applications, making the application less user-friendly due to the clunkiness of the application; this can be seen in figure 2.3. The user experience could be better. 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 budget. MoneyWiz, made me realise I should be more careful when picking functionalities to implement, as it can make the application unattractive to simple users who only need some key features. Making the balance between making the application as feature-rich and straightforward as possible is a challenging solution.

### 2.3.4 Financisto

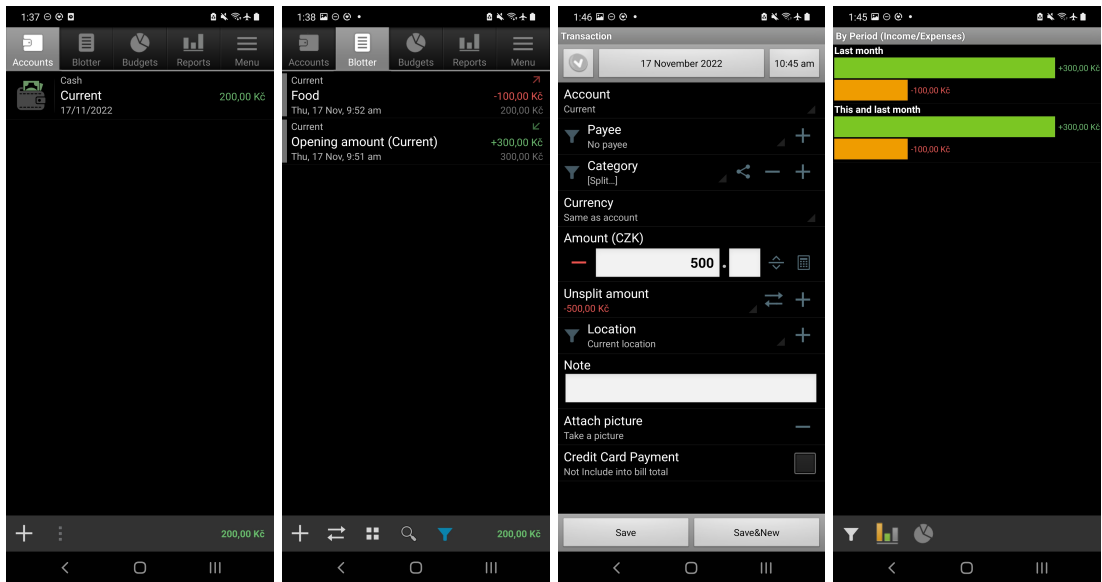
This finance tracking application is available only on Android.[8] The application supports most of the functionalities its competitors offer. On top of that, it has an exciting feature of splitting a transaction that the previous applications do not support. However, it needs to be updated UX and UI-wise. UX-wise, I was unhappy after using the application as many things were unclear, and navigation was placed at the top of the screen, making it hard to use with a single hand.

## 2.4 Requirements

Software requirements describe the features and functions of the target system. They represent the expectations that users have of the software product. These requirements may be explicit or



■ Figure 2.5 MoneyWiz 3 - iOS application



■ Figure 2.6 Financisto - Android application

	Wallet	Spendee	MoneyWiz 3	Financisto	XpenseTracker
Payment accounts	limited	limited	limited	✓	✓
Split transactions	x	x	x	✓	✓
Recurring transactions	✓	✓	✓	x	✓
Budgets	✓	limited	✓	✓	✓
Projects	x	x	x	✓	✓
Templates	✓	x	x	✓	✓
Statistics & Reports	✓	✓	✓	✓	✓
Web client	✓	✓	x	x	✓
iOS/Android client	Both	Both	Both	Android	Both

■ **Table 2.1** Comparison table for chosen finance tracking applications on their free plans

implicit, known or unknown, and expected or unexpected from the client's perspective.

### 2.4.1 Functional requirements

Software requirements define the mandatory functions and features that a software product must have. These requirements represent what the system must do. Functional requirements agreed upon with my supervisor are as follows:

#### F1: Registration

- The system provides the necessary UI for registering to the system.
- Registration fields are set from the API definition.
- Required fields are guided for the user to fulfil.
- The user is notified accordingly with the error message returned from the API on the UI side for inaccurate data.

#### F2: Authentication

- The system provides the UI for the authentication of a registered user.
- In case of incorrect data, the user is notified accordingly with the error returned from API.
- Authentication fields are set from the API definition.

#### F3: Accounts management

- User can see their payment accounts.
- The application provides the interface to create payment accounts.
- Existing payment accounts can be updated using the interface.
- The user can delete payment accounts using the interface provided.
- Required fields for payment accounts are to be set based on the API documentation.
- The user is warned for incorrect data at payment account creation and update.
- It is possible to see the recent transactions for each payment account, with a limitation of 10 records for simplicity.

#### F4: Transactions management

- A view of all of the transactions is provided.
- User can create transactions using the interface in the application.
- User can update transactions using the interface provided.
- User can delete transactions using the interface provided.
- Required fields for transactions are to be set based on the API documentation.
- The user is warned for incorrect data at transaction creation and update.
- Transactions are filterable based on different conditions:
  - Categories
  - Accounts
  - Projects
  - Transaction date
  - Realization date
  - Transaction amount
  - Transaction type: income, expense or transfer
- Transactions should be sortable based on the transaction date and transaction amount.
- By default, transactions are listed in decreasing order of transaction date.
- A single transaction is splittable into multiple transactions using the interface provided.
- Split transactions are distinguishable from regular transactions.
- Pagination functionality allows the user to see a set of transactions on the screen.

#### F5: Categories management

- UI provides a list of categories by default set to the user on registration.
- User can add new categories without any limitation in the hierarchy using the interface provided.
- User can update categories.
- User can delete categories.
- Required fields for categories are to be set based on the API documentation.
- The user is warned for incorrect data at category creation and update.



## F6: Budgets management

- The system provides a UI for budget management.
- User can create new budgets in the provided interface.
- User can update budgets.
- User can delete budgets.
- User can activate and close budgets.
- Required fields for budgets are to be set based on the API documentation.
- The user is warned for incorrect data at budget creation and update.
- The UI provides the historical data for each budget.
- Active and inactive budgets are distinguishable.
- The progress for the budget is visualised.

## F7: Projects management

- The system provides a UI for project management.
- User can create new projects in the provided interface.
- User can update projects.
- User can delete projects.
- User can activate and close projects.
- Required fields for projects are to be set based on the API documentation.
- The user is warned for incorrect data at project creation and update.
- Active and inactive projects are distinguishable.

## F8: Templates management

- The system provides a UI for template management.
- User can create new templates in the provided interface.
- User can update templates.
- User can delete templates.
- Required fields for templates are to be set based on the API documentation.
- The user is warned for incorrect data at template creation and update.
- Repeating and non-repeating templates are distinguishable in the UI.

## F9: Charts & statistics

- Expenses are visualised based on categories.
- The income and expense balance is visualised.
- Data used in charts is filterable by date.

## 2.4.2 Non-functional requirements

A non-functional requirement is a characteristic that defines the way a system should function and the constraints it must adhere to. These requirements are optional and represent additional requirements for the system's operation. Examples of non-functional requirements include application performance and other properties that are expected to be present in the system.

### NF1: Programming language

- Web and mobile applications should be developed using the same language.

### NF2: Support

- The application should be available in modern browsers such as:
  - *Google Chrome*  $\geq 97.0$
  - *Firefox*  $\geq 97.0$
  - *Safari*  $\geq 15.0$
  - *Microsoft Edge*  $\geq 96.0$
- The application should be available on Android devices running Android 11.0 or newer.

### NF3: Synchronization

- The applications should retrieve actual data from the server.

### NF5: Documentation

- The implemented solution should be documented to include an installation, and deployment guide.

After the analysis and requirements, it is time to design the entire system and its elements. The design transforms requirements from the previous step into an implementable state.

To fulfil this step of SDLC, I want to focus on three stages: UI, technologies and architecture. The UI stage to create visual representations of the planned applications. The technologies stage determines the programming languages and libraries to implement the proposed UI. Finally, I will define the architecture for the web and mobile clients in the last step based on the technologies from the previous step.

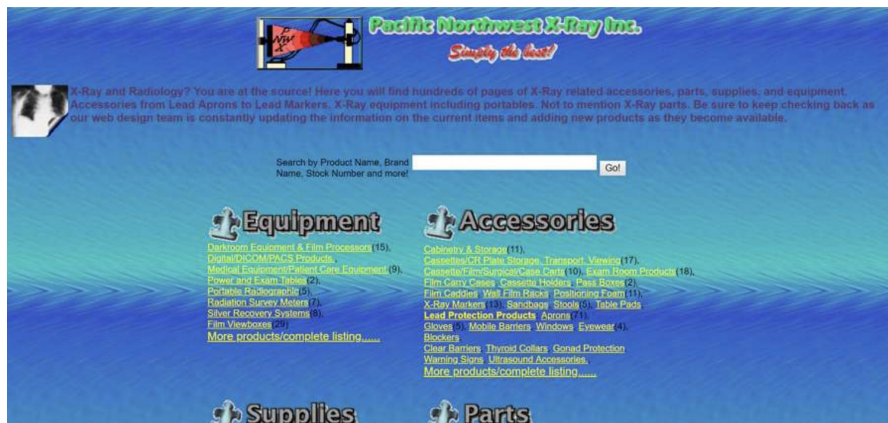
### 3.1 UI

An interface is a bridge between humans and machines that makes communication between 2 sides possible. The UI on our day-to-day lives, such as computers, phones, tablets or game consoles, consists of a front-end, a visually interactive face that communicates with the programmed system, and the back-end. Front-end interfaces are also known as graphical user interfaces (GUIs).[9]

A successful UI design combines good functionality, usability and aesthetics to achieve a successful outcome based on user requirements and expectations. For this reason, UI designs should focus on a user's needs and expectations rather than what a programmer or designer thinks is logical or looks good. Similar to these points, there are several principles mentioned in Laws of UX by Jon Yablonski[10] on which I would like to base the overall design of the UI of the applications:

**Jakob's Law** Jakob Nielsen, a usability expert, introduced this law, describing users' tendency to develop an expectation of design conventions based on their cumulative experience with other websites. With this law, it is pointed out that users value familiarity incredibly. By using familiar products, users can use another product easily as long as they spend minimum energy learning an interface and more time achieving their objectives. However, this principle does not necessarily infer that each website should look the same so users will not have difficulties using them. Instead, it is just a guiding principle pointing out that users use previous experiences to understand new experiences.[10]

**Hick's Law** Psychologists William Edmund Hick and Ray Hyman found that an individual's reaction time increases logarithmically when the number of choices available increases. In other words, people take longer to decide, given more options to choose from. This implies that complex or busy interfaces result in longer decision times for users because they must first process the options available to them and then select the most relevant concerning their



■ **Figure 3.1** An example of bad UI, choice of bad color palette and text colors makes it hard to read. Source: <https://www.mockplus.com/blog/post/bad-web-design>

goal. When an interface is too busy, actions are unclear or difficult to identify, and critical information is hard to find, a considerable amount of brain power is required to see what we are looking for. Simplifying an interface or process helps to reduce mental strain. Still, it must be sure to add contextual clues to help users identify the options available and determine the relevance of the information available to the tasks they wish to perform.[10]

**Aesthetic–Usability Effect** An aesthetically pleasing design can influence usability. Not only does it create a positive emotional response, but it also enhances our cognitive abilities, increases the perception of usability, and extends credibility. In other words, an aesthetically pleasing design creates a positive response in people’s brains. It leads them to believe the design works better, and they are more likely to overlook minor usability issues.[10]

### 3.1.1 Patterns

A pattern is a recurring solution to common problems in UI design. They can be considered models with a structure and can help solve a problem faster rather than building from scratch. For example, the breadcrumbs pattern allows users to follow their steps back. Each pattern has a specific use case, and knowing how and when to use them makes it possible to create efficient and consistent interfaces. They have a wide use case in the digital sphere, such as websites, mobile devices and other software to provide interaction, functionality and intent.[11]

UI patterns correspond to long-term reflections of designers based on experience, who have tried and failed different patterns, like Figure 3.1. Patterns are nothing new and have a long history from the past. It is essential to know the patterns and the reasoning for the result we have today to build successful UIs without the need to reinvent something from scratch. These small reusable UI solutions can build robust, intuitive solutions that resonate with people.[11]

#### 3.1.1.1 Why Patterns?

According to Product Designer Diana MacDonald, there are several reasons to choose patterns for building a cohesive and intuitive UI mentioned in the author’s book [11]. Some of them are:

**Design efficiently** Patterns help design more efficiently as they provide the tools and solutions that were proven successful.[11]

**Consistency and familiarity** Consistent patterns help users to get familiar with the website, similar to what Jakob’s Law mentioned previously.[11]

**Consistency and reuse** Due to their nature, using patterns allows for the reuse of components for both design and code-wise. This allows for a consistent and predictable interface from a user perspective. Reusable code makes it more efficient to have a working and successful product without the need to code features from scratch. This approach is also in place with the software engineering principle, Don't Repeat Yourself (DRY).[11]

**Evidence-based solutions** Patterns have been tried and evaluated in many different ways, and the ones we have today are the ones that were proven to be successful. They have been going through lots of trimming and refinement by designers and UX experts to be the way they are today.[11]

### 3.1.1.2 Common Patterns

There are different UI patterns, and I would like to choose the suitable ones and review their advantages and disadvantages for my use case for developing web and mobile applications.

#### Navigation

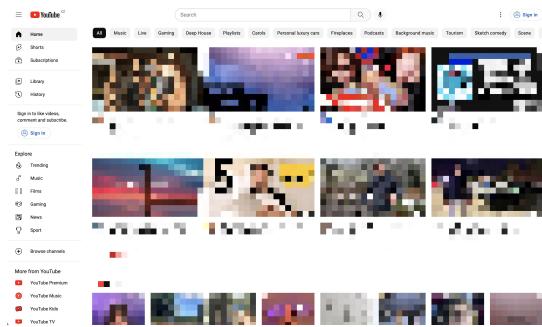
UX-friendly applications have good navigation to allow users to get from point A to point B without too much confusion and easily. Navigation is the map of the application, and the less time users spend on finding which link to click results in pleased users.

**Vertical navigation - Web** The vertical sidebar is a navigation menu component covering a portion of the web view horizontally. Links are present in a tree structure, table or group layout hierarchically. Unlike horizontal navigation, links are usually visible all the time, and there is no need for mouse hover to show the child links. Vertical navigation allows more space to include main navigation links on the first level as opposed to horizontal navigation, which due to its limited space, allows only for a limited number of main navigation links. Vertical navigation can easily be transformed into a mobile-friendly version, as opposed to horizontal navigation, which can not be translated directly due to limited space horizontally on mobile devices.[12][13] In the planned application, I don't expect to need too much horizontal space to fill in content and using vertical navigation is a good choice for the web application.

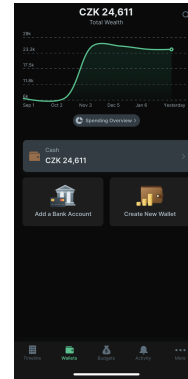
**Tab bar navigation - Mobile** This type of navigation is the most common pattern for navigation on mobile apps. It is positioned on the bottom of the screen and is present on all the screens of mobile. The reason for being the most common solution for navigation comes from the fact that mobile devices are used by a single hand usually, and the bottom of the screen is the most reachable spot by the thumb on the whole mobile device screen. The main disadvantage of the tab bar navigation is that it has limited space for navigation items and can contain only a few of those links.[14] For my use case of the mobile application, tab bar navigation should be enough.

#### Modal

Also known as dialog windows, this component is a pop-out window shown while browsing a website or clicking an action button. When the modal window pops up, the content on the view aside from the modal gets disabled and is not possible to access until the modal is dismissed. This is to catch users' focus on the modal and present critical information requiring user action. Most use cases include reacting to a message, filling out a form, etc. Filling out a form is one of the core requirements of the applications to be developed. Most of the functionality in XpenseTracker is based on user data submitted through the applications' forms.[15] A modal to show the form for creating a transaction is an excellent pattern to allow the user to submit the needed data.

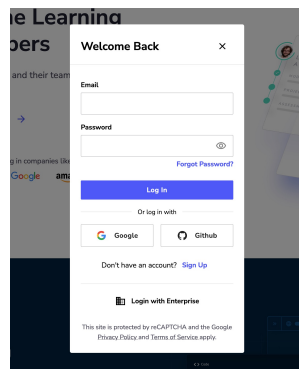


(a) Vertical navigation - Youtube <https://www.youtube.com>

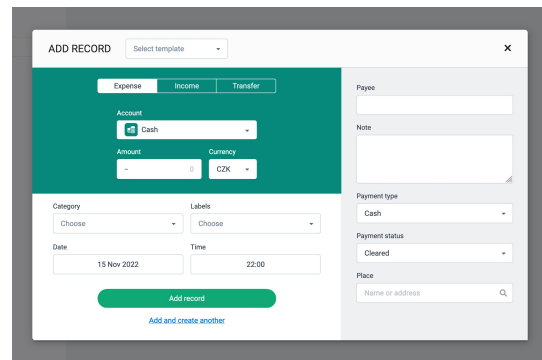


(b) Tab bar navigation on mobile - Spendeo

■ Figure 3.2 Navigation examples on web and mobile applications



(a) Login modal - Educative <https://www.educative.io>



(b) Record creation modal - Wallet

■ Figure 3.3 Modal use cases

## Card

It's a UI design pattern that brings related information inside a container that resembles a playing card. Information gathered in a card should be concise and to the point, without any extra knowledge or long text. Just like physical cards, they should represent the information they contain visually.[16] This pattern helps define budgets and templates as they have information that can be visualised using this UI pattern.

### 3.1.2 Wireframes

The general idea and best practices have been defined previously. Now it's time to apply the theory to the next step of the design. For this, I have decided to create wireframes to illustrate the web and mobile applications. Wireframes are just technical representations of a screen using skeletal lines. They are also known as UI blueprints. Wireframes do not require a specific visual style, they need to be clear and understandable. [17]

I want to review each view planned for both clients and describe the wireframes individually. For the creation of these wireframes, I have used Balsamiq [18], a desktop application to create wireframes.

## Registration and login 3.4

These pages are the entry point to the application. Users can create new accounts using the registration page and, using the login page, can authenticate to use the application. Both pages will contain forms for users to fill in.

## Dashboard 3.5

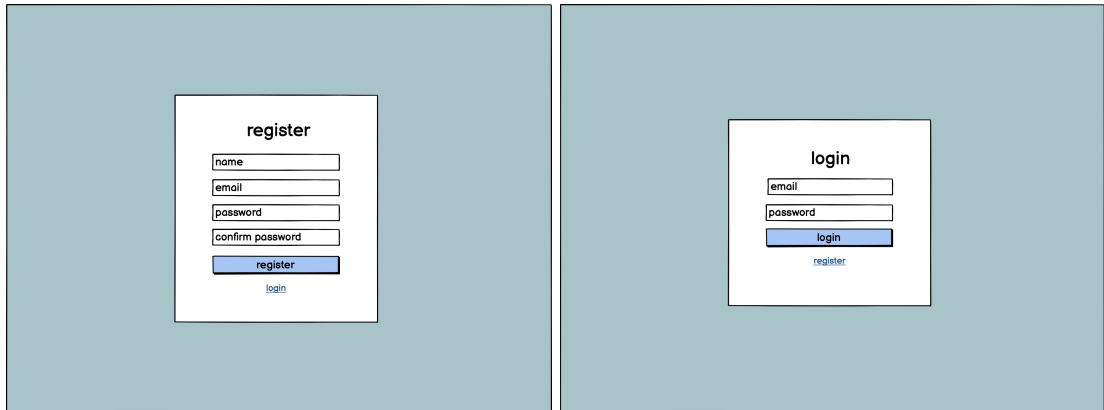
This view is the initial screen shown to the user after successful authentication. In typical applications, you would see the latest resources depending on the application. For this reason, a list of the most recent transactions can be placed in this view for the user to take a look at first glance. Charts and statistical graphs could be placed in this view as part of the requirements. Visual representation of data is a typical way to show to the user on the first page they visit after the authentication.

## Accounts 3.6

User is able to see their payment accounts in this view. To create a new payment account, a button aligned with the view title opens a modal for filling in the necessary information to create the payment account. Selecting a payment account reveals the most recent records associated with the payment account as the accordion's content.

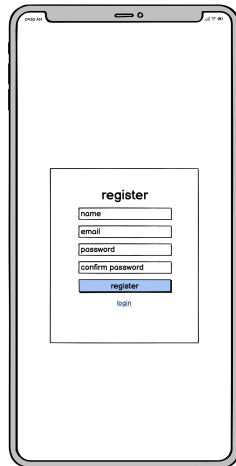
## Records 3.7

Records view is responsible for listing all the transactions from all payment accounts. To create a new record, the button aligned on the right side of the view title opens the record modal. Click on the record will load the modal with record data for an update or deletion. User can apply filtering by using the button adjacent to the record creation button, which will open a menu with filter options. Records with subtransactions are labelled with a plus icon at the beginning; clicking that icon will reveal the subtransactions. Pagination of the records is possible using the buttons at the end of the records labelled with numbers.

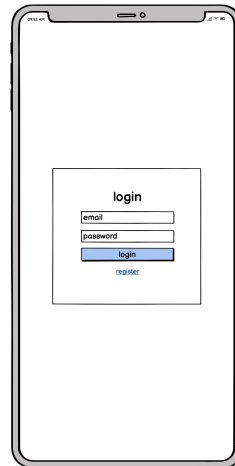


(a) Web registration view

(b) Web login view



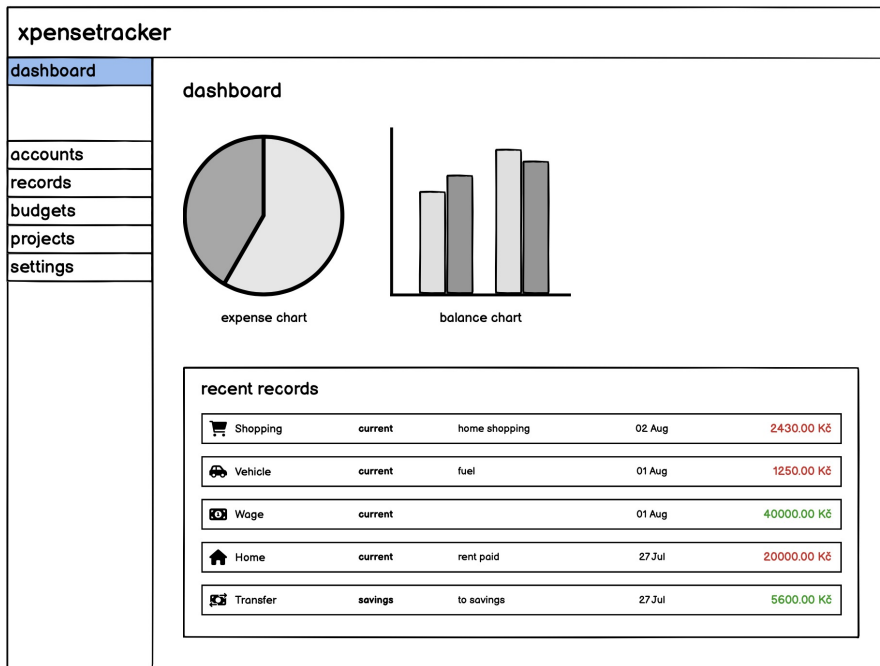
(c) Mobile registration view



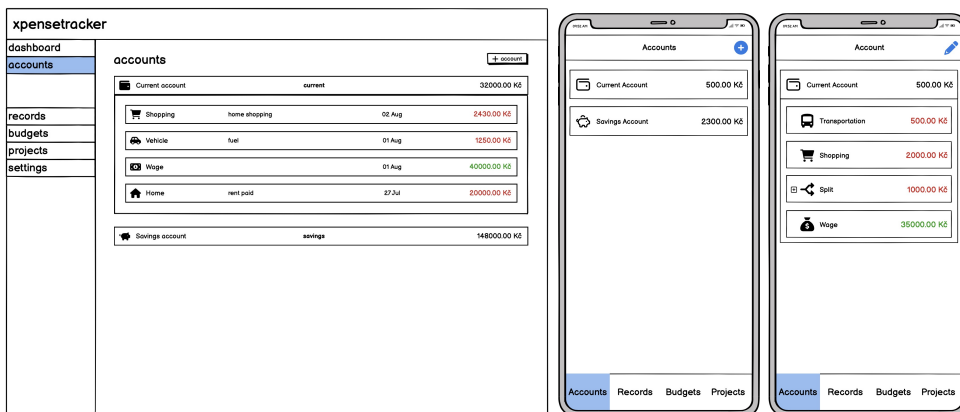
(d) Mobile login view

■ **Figure 3.4** Registration and login view wireframes





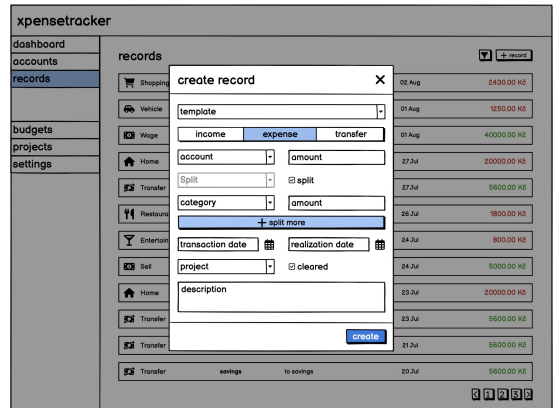
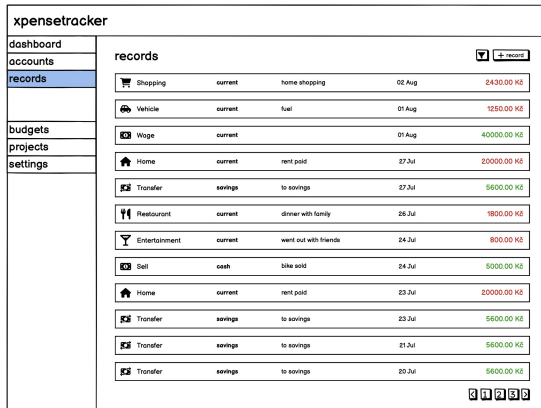
■ Figure 3.5 Web dashboard view



(a) Web accounts view

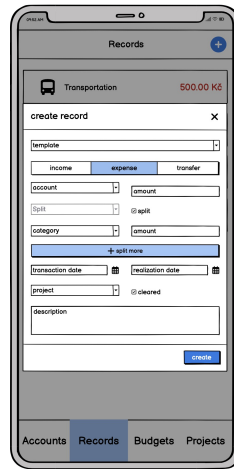
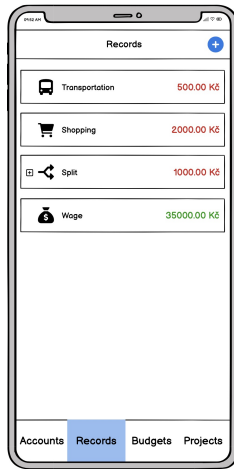
(b) Mobile accounts view

■ Figure 3.6 Accounts views for web and mobile



(a) Web records view

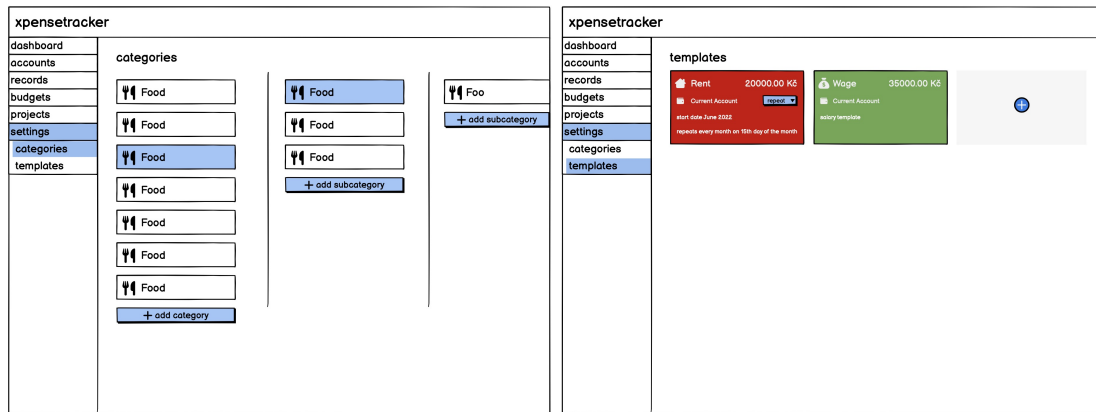
(b) Web record creation modal



(c) Mobile records view

(d) Mobile record creation modal

■ Figure 3.7 Records and record creation modal views for web and mobile



(a) Categories view

(b) Templates view

■ **Figure 3.8** Categories and templates views

### Categories 3.8a

In this view, preset categories will be listed in a horizontal view and clicking on each item reveals the sub-children of that category. As there can be an infinite hierarchy of children, the view will be horizontally scrollable to make space. On the mouse hover over the item, buttons to edit or delete the category show up, allowing the user to modify each category. On edit, input with category name and editable category icon is shown. A button at the end of each list is positioned to add more categories.

### Templates 3.8b

This view is responsible for showing recurring transactions and transaction templates. A card will represent each type, and based on the type of transaction - income or expense, it will have the colour green or red respectively. Cards will have information about the template, such as name, amount, category, account information, and, if applicable, the frequency of recurring transactions. A new template can be created using the card-like button with a plus icon in the middle. Modal view with template creation form opens on click of the button.

### Budgets 3.9

A card-like component will represent each budget on the web and as a list item on the mobile. Each item will have information about the budget, like if it's active, frequency and name. A bar component to visualise the progress within that period is available in the item. If the budget has a history, a small button will be available indicating more information related to the budget. Clicking that button will extend the item and show the budget history. The extended part will contain a bar-like component filled with the historical data for visualisation and controllers to navigate through each historical data.

### Projects 3.10

Similar to payment accounts representation, this view will represent projects in a list-like layout, with each item containing records assigned to the project. Each item will include information such as name, start and end date and status. A new project can be created using the button on the right of the view title. Similarly to payment accounts clicking the project will open the modal with prefilled data for an update or deletion.



(a) Web budgets view

(b) Mobile budgets view

■ Figure 3.9 Budgets views for web and mobile



(a) Web projects view

(b) Mobile projects view

■ Figure 3.10 Projects views for web and mobile

## 3.2 Technologies

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

### 3.2.1 Languages

Different alternatives on the market focus mainly on the 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 I should choose any of them 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 is not supported

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

## 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 perfect choice for full-stack development, saving time and reusing functions and libraries with much less effort to create web, server and mobile applications.

One of the main disadvantages of using JavaScript is that it's a 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 human-based errors.

## TypeScript

Typescript is a free and open-source programming language developed and maintained by Microsoft. It is a strict syntactical superset of JavaScript and adds optional static typing, interfaces and class-based object-oriented programming to the language. The most significant advantage is to enable IDEs to provide a more prosperous 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 allows the machine to analyse the code faster and allows developers to understand the code better and faster. These features enable TypeScript to help developers feel more confident in their code and save considerable time validating that they have not accidentally introduced new bugs into the codebase.

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.

### 3.2.2 Libraries

As the saying goes, "Don't reinvent the wheel", no need to do work that has already been done. Using the packages and libraries that have been verified in the community and have enough support is wise. This would save time for coding problems that have already been resolved. Although, packages and libraries shouldn't just be added to a project just like that by developers. 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
- easy to integrate

As we advance, most of the libraries and tools used in the development are written in Javascript. However, majority of these libraries and tools support Typescript as well. We will learn the core libraries used in developing web and mobile apps. More libraries have been selected, but the following are the most noteworthy.

## React.js

React is a JavaScript front-end library for building UI components. Facebook developed it in 2011, and later in 2013, it was open-sourced. It has one of the most significant communities of developers and a big market share in web development.

React creates user interfaces predictably and efficiently using declarative code. It can be used to develop single-page web and mobile applications or build complex applications with other libraries. We can summarise the reasons behind React's popularity and why it's a good choice in today's technologies:

**Simplicity** It is easy to learn React's API. Its component-based approach, well-defined lifecycle, and use of plain Javascript make React simple to 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 minimising the changes applied to DOM. For this, React uses 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 unique 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 changed.

**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 can be reused throughout the application, dramatically reducing the development time.

**Data flow** React follows a unidirectional data flow. 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.

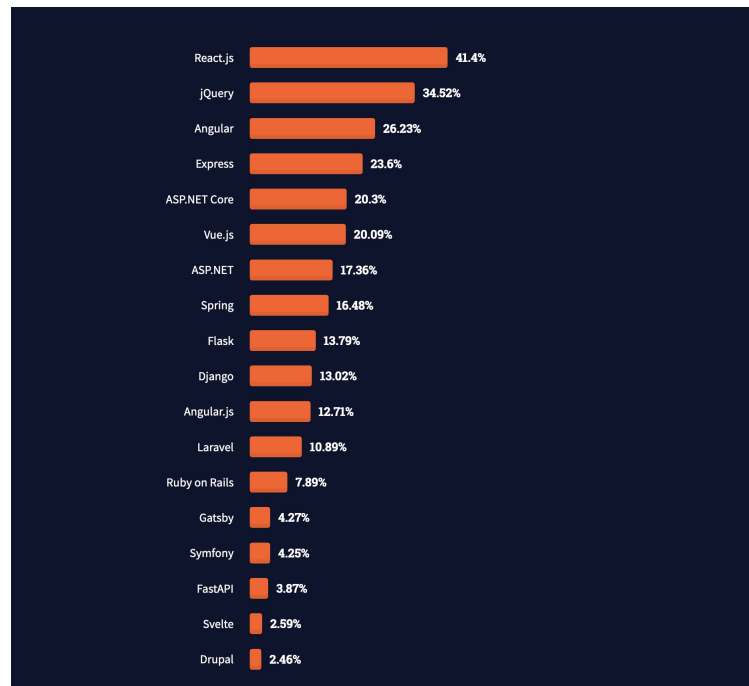
**Testability** React applications are straightforward to test. React views can be treated as state functions, so we can manipulate the state we pass to the React view and look at the output and triggered actions, events, functions, etc. Many libraries are also available to test React applications, which are easy to set up and start testing.

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

Of course, reviewing React's disadvantages is vital to understand better if it is a good candidate for our use case. Drawbacks for React can be summarised as follows:

**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, you would usually need to use a state management library that will allow access to data without the need for 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.



■ **Figure 3.11** Most commonly used frameworks in 2021 according to survey by stackoverflow:  
<https://insights.stackoverflow.com/survey/2021#web-frameworks>

**Memory usage** Virtual DOM was noted as one of React's pros as providing good performance. However, this comes with a price; virtual DOM consumes a high amount of memory, which can cause performance issues for some users.

Other strong and mature alternatives to React at the moment are Angular.js and Vue.js:

**Angular.js** It is older than React and Vue.js. It has been developed by Google and follows an 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. It has great community support.

**Vue.js** The newest out of 3 comparable frameworks. It is easy to learn and has the advantage of two-way data binding. That is an outstanding feature, as many times, one-way data binding comes with issues of data sharing between child and parent components. The community is considerably smaller than React and Angular.

Considering the advantages and disadvantages of React library and other options, React is a good choice due to my personal experience with it. External libraries can resolve the drawbacks of React to complement it. Specifically, the issue with state management can be resolved by using the Redux library. The same can be said for the routing problem.

## React Native

There are different alternatives for mobile development. However, the framework choice for mobile development is evident due to choosing React for web application development. The relationship between these two libraries provides the opportunity to reuse components and logic with minor modifications. Most importantly, React Native provides a single codebase for Android and iOS platforms, saving development time by coding once. The state management could be the same in both development projects, based on my experience.



## Redux

Redux[19] is a state management Javascript library used mainly with libraries such as React, Angular and Vuejs. It is an in-app database that handles storing and using data during the application lifecycle in the browser. As React follows unidirectional 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.

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 an 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 behave as a bridge to let the data pass through.

Another problem Redux helps resolve is allowing the application to have a central data source. This gets done in one location if data needs to be retrieved or modified. This prevents data from being inconsistent throughout the application.

Of course, Redux has its drawbacks as other libraries. Redux comes with lots of boilerplate code, and it can be challenging to grasp a hand at it. It's also not beginner-friendly and, if not used correctly, can cause unnecessary renders.

## Redux Toolkit - RTK

Redux Toolkit[20] is a Javascript library that makes state management with Redux easier. As mentioned, Redux comes with lots of boilerplate code, which is complicated to configure and provides opportunities to make mistakes. Redux Toolkit helps to resolve these issues and provides APIs that simplify standard Redux tasks. It is simple to learn and use. Being opinionated makes it a good choice for starters.

## React Router

In the introduction of React, it was mentioned that one of the missing features is routing. Routing is essential for any application with multiple pages containing different content. React Router library exists to resolve this issue for single-page applications (SPAs) by introducing "client-side routing". This type of routing allows the app to update the URL from a link click without making another request for another document from the server. The advantage of this type of routing is that there is no need to wait for the server response and reevaluate the CSS and Javascript assets for the next page. It results in faster UX and more dynamic animations.[21]

## React Navigation

Similar to the case with React, React Native does not have a routing solution out of the box, and contrary to its naming, React Navigation exists to provide a solution for routing and navigation in React Native applications. It exists from the support of the community and is recommended by official documentation of React Navigation [22] as well. The tabbed navigation pattern which was decided in the design chapter for mobiles is provided by this package.[23]

## Ant Design

React encourages a component-based approach to development. A component is a small feature that makes up a piece of the UI, like a button or an input field. Dividing a complex UI into small components is beneficial for development since it provides easier readability and maintainability.

A component can present a text or a complex one, such as a form consisting of different sub-components. Building these types of complex components can be time-consuming, considering the potential amount of time to be spent on testing and making the component extendable. For this reason, component libraries exist. They provide components that are frequently used in applications, such as buttons or a date picker.

Ant Design[24] is a component library that provides easy integration to React applications with many customisable components out of the box. It has some of the core components required in the planned application, such as a Table, DatePicker and TreeSelect, that are challenging to find together in most libraries with functionalities necessary for this application. According to the documentation, these components should fit the application ideally based on the suggested examples and integration guide.

Another reason to choose Ant Design, it's design strategy. Aesthetics is one of the core requirements for a good UI and UX. The components' simple but elegant styling and customizability impressed me personally and fulfilled the imagined application.

## Native Base

Similar to Ant Design, Native Base is a component library for React Native, providing reusable and customisable components for mobile development. The reason for choosing this library instead of a different one is mainly the availability of the components to be used in the mobile app. There are few alternatives as opposed to React libraries.[25]

### 3.3 Architecture

In this section, I will discuss the type of software architecture pattern that I plan to use in the client applications.

#### 3.3.1 MVC architecture

This is one of the oldest and most common architectural design patterns used in software development to distinguish data, logic and presentation. *M* stands for *Model*, *V* for *View* and *C* for *Controller*. These terms shortly can be summarised as follows:

**Model** Usually means data and management of data in an application.

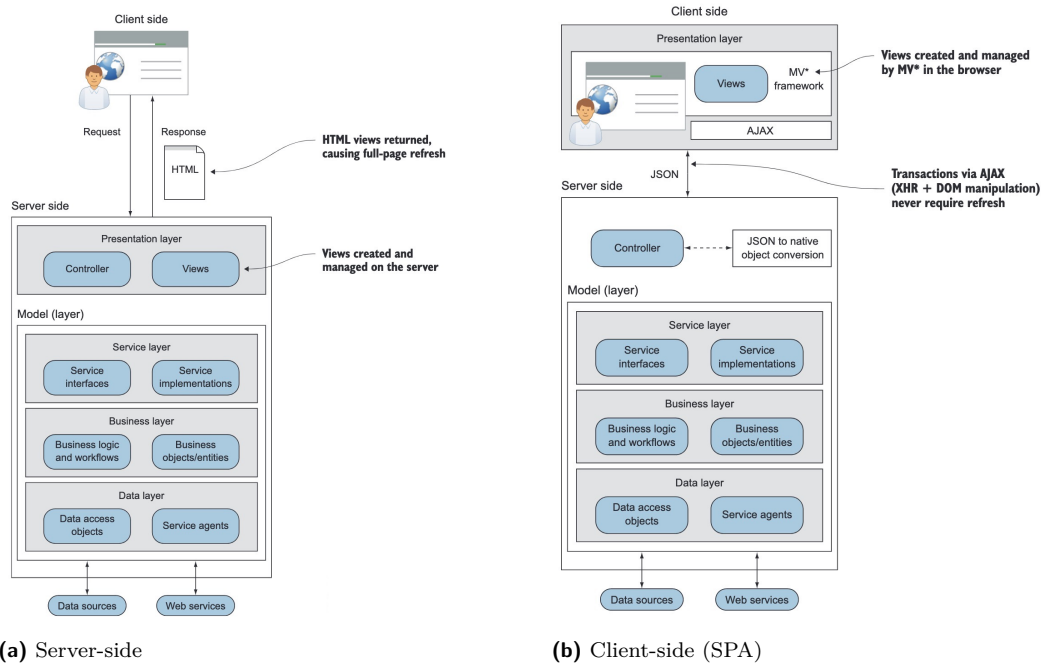
**View** This is the visual part and what the user interacts with in an application. It's the visual representation of the model data.

**Controller** Processes server-side logic and acts like the middleman between View and Model to control the flow of data.

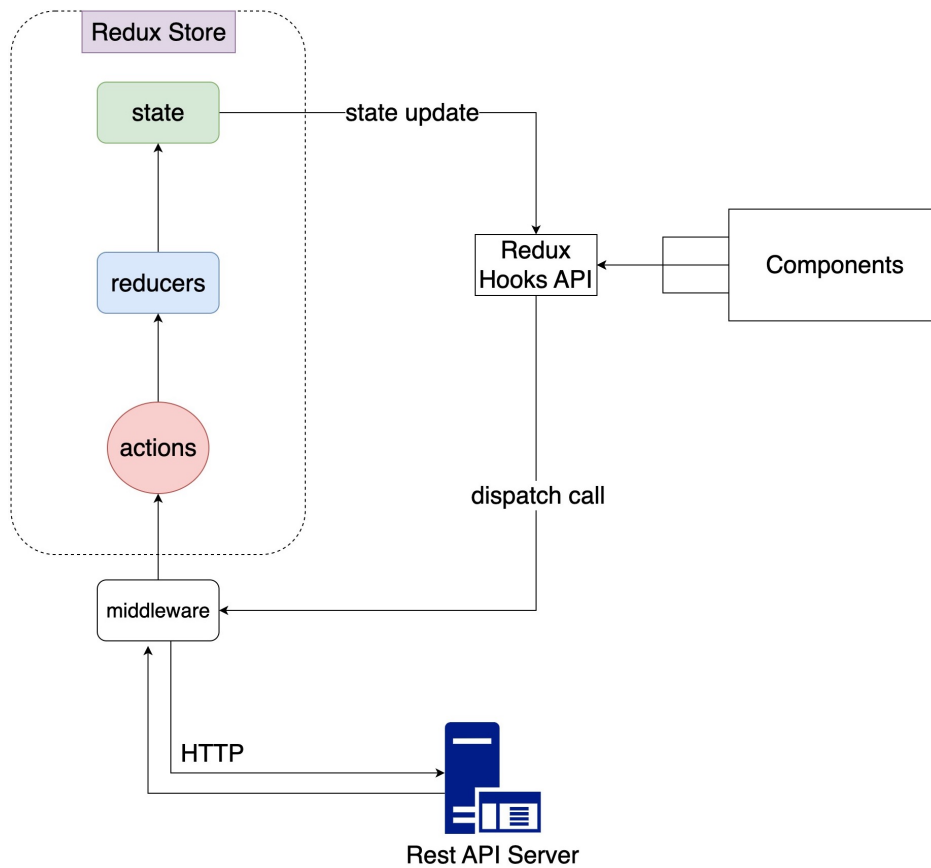
#### 3.3.2 Web

Historically browsers were used to show the HTML received from the server and did not have much influence on the content displayed to the user. Their function was merely like a typical file viewer. Nowadays, with more powerful computers and modern browsers, the idea of having desktop-like applications on the browser is not uncommon. Interacting more with the content and having a faster UI is essential for better UX.

There are two ways to generate the UI on the browser with current technologies: the server-side and client-side (SPA) rendered UIs. In a server-side rendered application, the most traditional and older option, the web page and its content are predefined on the server based on the request and are just displayed on the browser on each refresh. Figure 3.12a demonstrates this



■ **Figure 3.12** Comparison of traditional server-side and client-side rendered architectures [26, p. 5, 6]



■ **Figure 3.13** Planned architecture of the client side applications

architecture. On the other hand, in a client-side rendered application, the application's presentation layer is extracted from the server and handled on the browser level. The browser decides on the content to be shown based on the user interaction and data from the server without the need for a browser refresh. Figure 3.12b demonstrates the client-side architecture.

React, the chosen technology for the web application, promotes the single-page application (SPA) pattern by using independent components that represent the blocks of a complex application. However, in its default form, it does not include all the features typically associated with an SPA. According to its documentation, React is primarily designed for building simple user interfaces that consume data and generate HTML. In this sense, React can be considered the View layer of an MVC architecture.

The addition of Redux for state management makes the application more complex and adds the rest of the 2 layers from MVC to the architecture that React itself misses. Redux contains the state which is the data layer and the logic to handle the management of data by commands received from the View layer. Figure 3.13 demonstrates the planned architecture for the web application.

### 3.3.3 Mobile

The mobile app is similar to the web app and can be considered as a single-page application(SPA). The way the web and mobile applications should communicate with the server is the same. Tools to be used for establishing this connection with the server are the same and safe to say the working principles of React and React Native are pretty much the same with few exceptions that don't prevent using the same architecture used in the web application. Taking into consideration all these similarities, Figure 3.13 can be used as a reference to the planned architectural design of the mobile app.

# Implementation

This chapter is where the design decisions from the last two chapters come into realisation as part of the SDLC. I would like to introduce the steps I've taken to achieve the result planned in the design chapter.

As an introduction, I would like to review how the project management was set up, followed by a discussion of libraries and tools used to support the development and resolve the disadvantages of React and React Native mentioned in the design part. Next, I would like to discuss how features such as authentication, state management and data layer have been implemented. The following details of implementation will be dedicated to UI-specific implementations of the clients.

## 4.1 Project management

For the development of the client applications, I decided to use a VCS for keeping records and the history of development files. Regarding software, I decided to use Git [27] for this purpose. Both clients have their own repositories which are stored on the FIT Gitlab servers, they are accessible using the following URLs:

- Web: <https://gitlab.fit.cvut.cz/qamilami/xpensetracker-frontend>
- Mobile: <https://gitlab.fit.cvut.cz/qamilami/xpensetracker-mobile>

## 4.2 Tools and libraries

As it was mentioned before, the community around the React ecosystem is quite well-developed. There are lots of libraries to use proven by the community to speed up development time, this is quite crucial in my case as I have to implement two different clients and the amount spent for specific features should be as minimal as possible. Of course, I couldn't choose the correct libraries each time to decrease the time spent with the implementation of some functionalities,, this is expected at this level and is a part of the technical debt [28] that most developers and companies experience. Nevertheless, I would like to list the libraries and tools that were used to achieve the finalisation of client applications.

**Expo** Expo is a framework that enables a suite of tools and services designed for React Native development. It is also recommended by React Native documentation [29] for an easier start with the development. Expo is built on top of React Native and there is no need to use xCode or Gradle for building iOS or Android builds respectively, which usually ends with different errors while setting up the environment. Taking into consideration the limited amount of time

to finish the applications, using Expo's building service using the cloud for creating a build is a wise choice. With Expo, it is not necessary to have a physical device for testing either, Expo provides a client app that allows you to run your application through an emulator and test the application without any problem.[30]

As for the disadvantages, it's worth mentioning the unavailability of some native APIs available in React Native. However, these should not affect the decision of using the Expo, because in my case I do not need all of the mentioned native modules.[30]

Taking into consideration the benefits over the disadvantages, I think Expo is a valuable tool to use for mobile development.

**React Hook Form** The application that will be built is going to use forms quite often. It's an essential part of the application to interact with users. It's important to validate the 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.[31]

**Chart.js** For the requirement regarding charts and statistics, it was necessary to find a library that allows the creation of charts for this purpose. Chart.js is a library that provides various chart types with high customizability. It's actively maintained by the community and is the most popular charting library in Github. The ease of integrating with React applications is another bonus to choosing this library for this purpose.[32]

**Vercel** It is a PaaS product, that allows the deployment and hosting of different kinds of applications, specifically built to work well with React applications. Using Vercel is quite easy as the only thing that 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 with little to no configuration. The current implementation is also hosted on Vercel and can be accessed at <https://xpensetracker.vercel.app>. Vercel was used to create a demo website for allowing my supervisor and other users for testing the web application during and after the finalisation of development.[33]

## 4.3 Implementation details

### 4.3.1 Authentication

Authentication is the key to interacting with the server and differentiating user data from each other. According to API documentation, authentication is of a token-based type. An access token is sent from the server to the client upon receiving user credentials, which is just a digitally encoded user signature. Using this token, the user authenticates their identity to access specific resources. Alongside the access token, the server also sends another token called the refresh token. The refresh token is a credential artifact similar to the access token used to obtain a new access token when it becomes invalid, as they have an expiration date, without asking the user for their credentials. This type of authentication is widespread and simple to implement on the client and server sides. However, it has the disadvantage of not being secure enough, but for the sake of this thesis, it should be enough for internal usage.

In theory, implementation for token-based authentication on the client side is relatively simple. The access token received from the server, using the necessary credentials, needs to be used for each of the following requests regarding user-specific resources. For this reason, the token needs to be stored and easily accessible when it is necessary to authorise the user. In the browser, the `Window` interface representing the window containing the DOM document has the `localStorage` property that makes it possible to store data. For mobile, a similar solution is `AsyncStorage`, a key-value storage system like `localStorage`. When the user submits the login

form, on a successful response from the server, data is stored under the `user` object in either `AsyncStorage` or `localStorage`.

The access token is used whenever it is needed for the requests to the server by retrieving it using the custom hook `useToken`. `useToken` contains some functions that help retrieve the token from the storage type and check if it's valid or not. The `PrivateRoute` component uses the authentication state from the `useToken` hook, which is a custom Route component to allow certain pages to be visible for authenticated users and redirect non-authenticated users to the Login page. For mobile, it is a similar case with the `HomeScreen` component deciding which navigation stack to show to the user.

Refreshing token happens at the query level in the data layer. Whenever a query fails due to an error from the server with an authentication error, the same query first refreshes the access token by grabbing the `refresh_token` from the storage and then notifies the state with the new access token. `useToken` hook listens to these changes and updates the new access token in the storage as well for further usage.

### 4.3.2 State management

Redux Toolkit is built around Redux core and provides helper functions and packages to make using Redux simpler and use less boilerplate code Redux requires to be set up. Redux has three essential terms:

**Action** Javascript object with required type that informs reducer what happened to the state.

It is not responsible for modifying the state. The type of action is a string that describes the action, and optional properties are the information needed to update the state. An action is dispatched, and the reducer handles the state's update.

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

**Store** This is the central source containing the application's global state.

In the traditional way of setting up Redux, you needed to write the actions and reducers individually, which was a lot of repeated code. Redux Toolkit provides a `createSlice` API, it's named this way as it's concerned with a single piece of the state. `createSlice` automatically generates all the action creator functions and generates action type strings internally based on the reducer's names by just writing the reducer logic. Another important feature is that `createSlice` uses the Immer library internally to write immutable reducer functions, which is a requirement as per the definition of a reducer, they need to be pure functions and return an immutably updated state. Finally, these individual slices are combined in a single store to create the global state.

### 4.3.3 Data layer

Usually, during the development of client-side applications, it is necessary to use mock data to simulate the response from the server side. This pattern is proper when the development of client-side and server-side happen in parallel. However, in the context of this thesis, this was not necessary as the server side had already been implemented, and there was a working API to interact with. For this reason, it was a good idea to implement the data layer of the applications as the next step, as this logic, in theory, should be used by both web and mobile clients.

Client-side applications, in general, consume the data received from the server by making temporary manipulations to represent it in the correct state to the user. This data must be up to date whenever a resource has been modified. For example, if the user deletes a payment

account, it should not be available to the user as an option when a new record is being created. RTK Query can resolve issues like that and provide fetching server data.

RTK Query is an optional library included with Redux Toolkit, built on top of Redux Toolkit. According to the documentation of Redux Toolkit, "RTK Query is a powerful data fetching and caching tool. It is designed to simplify common cases for loading data in a web application, eliminating the need to hand-write data fetching & caching logic yourself.". It provides several APIs. Most important to mention are the following:

**createApi** This is the core API of the RTK Query functionality. It allows for defining a set of endpoints on how the data will be fetched and transformed from specific server endpoints.

**fetchBaseQuery** It is a factory function implemented around the `fetch` API that aims to simplify HTTP requests and generate data fetching methods compatible with RTK Query's `baseQuery` configuration option.

The communication configuration with the server is done by setting the server URL and HTTP headers for request authentication using `fetchBaseQuery`. It is also possible to set up the token refresh logic mentioned in the previous section by setting up some logic to check if the request succeeded and, if not, make a new request for refreshing the access token. Once the new token is retrieved, the previously failing request can be executed again.

For each type of resource in the application, `createApi` is used to create an API slice for fetching the data from the server, example for such API slice for accounts can be seen in Figure 4.1. This API slice returns custom hooks that can be used in the components for executing the request. Once the hook is called, RTK Query will automatically fetch data on the component mount and re-fetch if parameters change. The hook returns an object containing the data and state of the request. This data can be used in the component directly.

#### 4.3.4 Views

This section is dedicated to the implementation of different views of the applications.

##### Login and Register

These views are what users see when they open the application either on the browser or on the mobile. They can fill in the form on the Login view if they have a user account or simply create one using the form in the Register view. The forms are simple and contain few inputs required from the user, placed in a card-like element centred in the view. Navigation between two views is possible using the text-type buttons at the bottom of the cards. Images from the views can be seen for both clients in Figure 4.2.

##### Dashboard

This view is created to show the user a summary of the latest 10 transactions and a graphical representation of some transactions by categories and types. The button on the top right of the view allows for filtering transactions used in the graphs by a date range.

##### Accounts and Projects

The data type for these resources is very similar, for this reason, I decided to make them look the same.

**Web - Figure 4.4** Each item in a list is represented by an accordion element which is expandable to show the transactions attached to it. On mouse hover on the accordion, an icon button



```

import { api, providesList } from 'store/service';
import { IAccount, IAccountDTO } from 'shared/types';

const accountApi = api.injectEndpoints({
  endpoints: (builder) => ({
    getAccounts: builder.query<IAccount[] | [], void>({
      query: () => ({ url: '/accounts' }),
      providesTags: (result) => providesList(result, 'Accounts'),
    }),

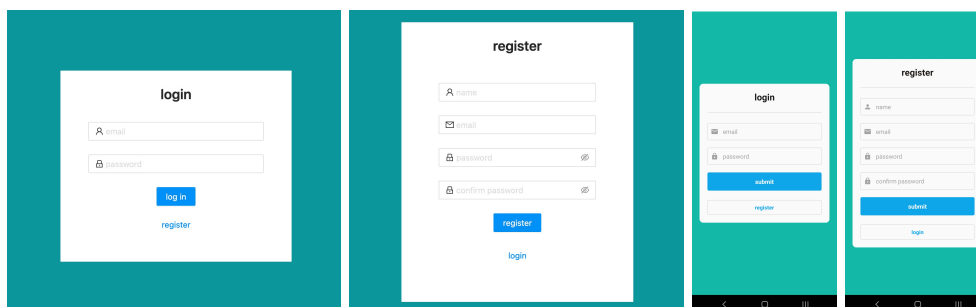
    addAccount: builder.mutation<IAccount, IAccountDTO>({
      query: (body) => ({
        url: '/accounts',
        method: 'POST',
        body,
      }),
      invalidatesTags: [{ type: 'Accounts', id: 'LIST' }],
    }),

    updateAccount: builder.mutation<IAccount, Partial<IAccount>>({
      query: (data) => {
        const { id, ...body } = data;
        return {
          url: `/accounts/${id}`,
          method: 'PUT',
          body,
        };
      },
      invalidatesTags: (result, error, { id }) => [
        { type: 'Accounts', id },
        { type: 'Transactions', id: 'LIST' },
        { type: 'Templates', id: 'LIST' },
        { type: 'Projects', id: 'LIST' },
      ],
    }),
  }),
  overrideExisting: false,
});

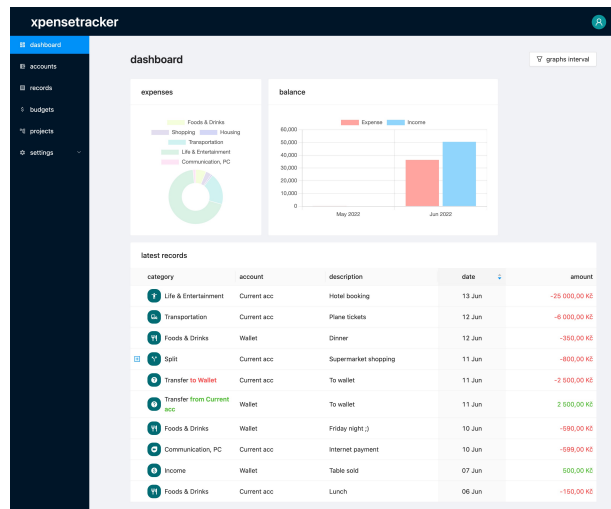
export const { useGetAccountsQuery, useAddAccountMutation, useUpdateAccountMutation } = accountApi;

```

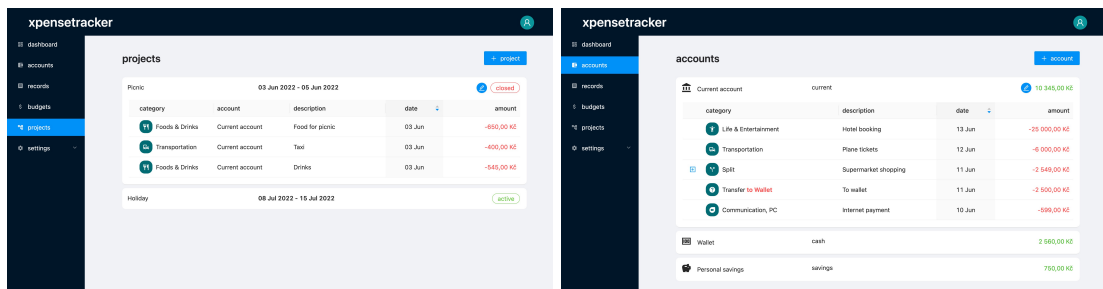
■ **Figure 4.1** Accounts API slice



■ **Figure 4.2** Login and registration views on the web and mobile application respectively



■ **Figure 4.3** Dashboard view on the web app



■ **Figure 4.4** Accounts and projects view on the web app

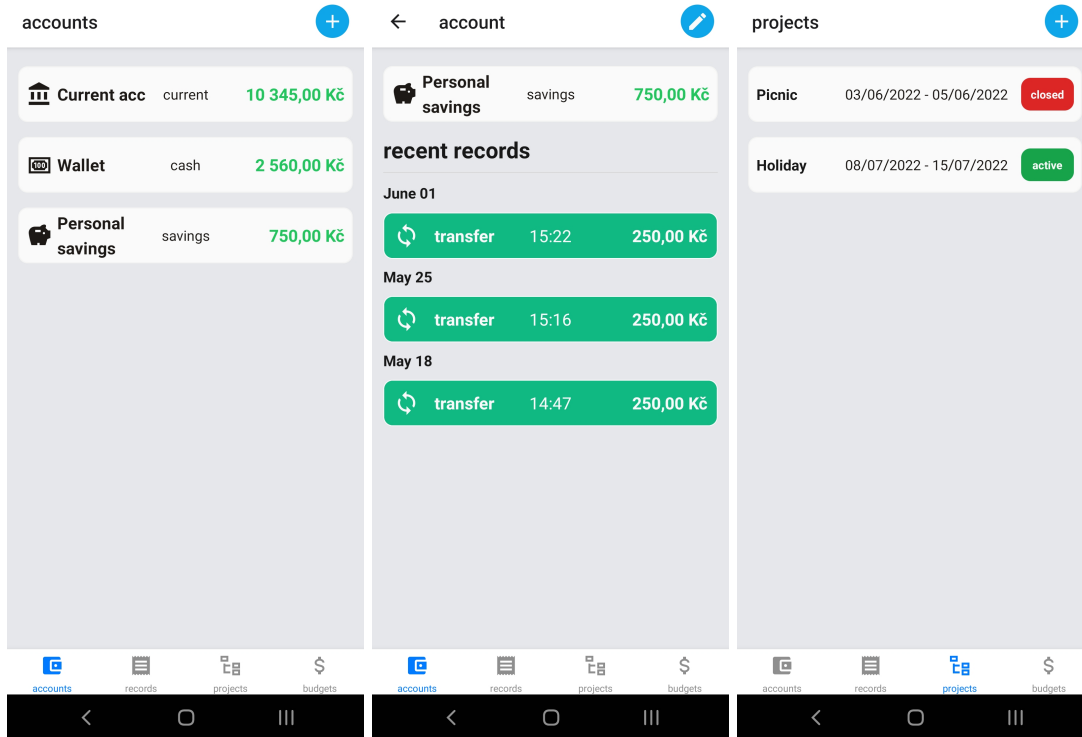
for editing the resource is displayed. This button activates the modal with the prefilled form. To create a new resource the button on the top right is used to activate the modal with the form.

**Mobile - Figure 4.5** The view shows a list of pressable items with information about the resource. On the press of the element, the user is navigated to a new view which contains detailed information about a singular item. In this view transactions attached to the resource are listed. To edit or delete the resource a button on the top right view can be pressed to show the modal, in which the necessary action can be completed. Same as in the web version, in the main view, a button with the plus icon on the top right can be pressed to create a new resource using the form.

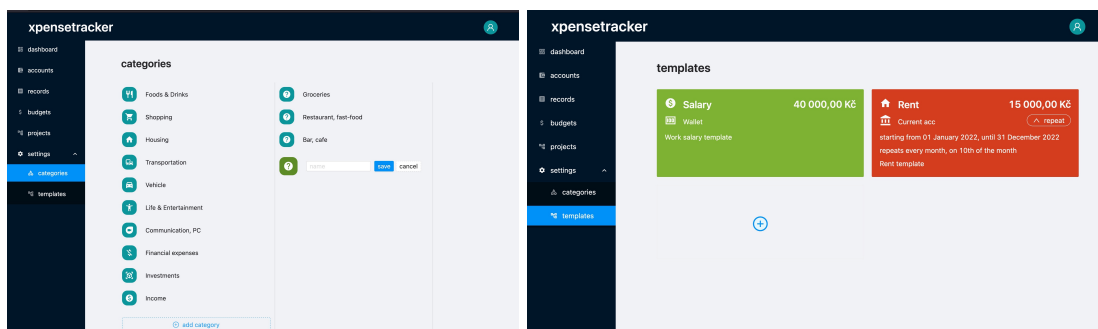
## Records

In this view, all the transactions created by the user can be managed. Filtering and pagination are implemented to allow for better navigation through transactions.

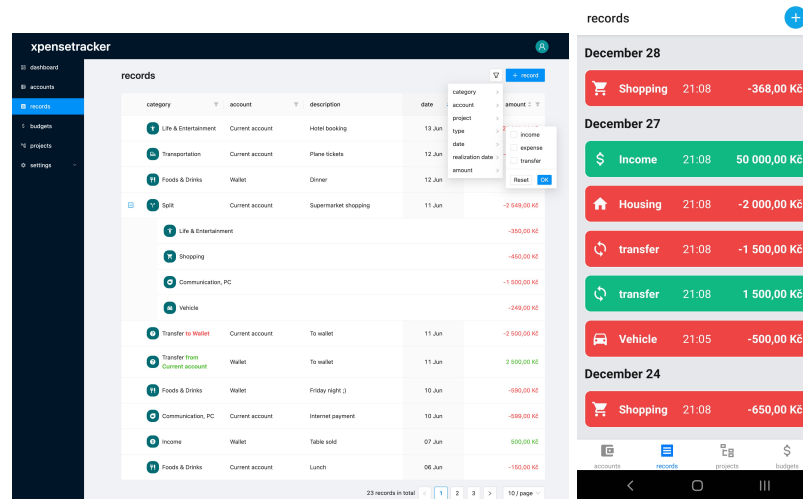
**Web - Figure 4.7** On the web, transactions are shown in a table-like view. The transactions are by default listed by the transaction date and using the columns on the header this sorting can be changed as well as filtering of transactions can be applied. Each item in the table can be clicked to display the modal prefilled with data from the clicked item. To create a new



■ Figure 4.5 Accounts and projects view on the mobile app



■ Figure 4.6 Categories and templates view on the web



■ **Figure 4.7** Records view on the web and mobile respectively

transaction the button on the top right can be used to show the creation modal. The main filtering button is placed next to the create button with a filter icon, there are various filtering options such as category, account, project, transaction date, realization date, transaction type and amount. The pagination is placed at the end of the table of transactions.

**Mobile - Figure 4.7** Transactions are listed as individual items in a scrolling view. Pressing on the item shows the modal with prefilled data, except the split transactions navigate to a new view with details of the subtransactions.

## Budgets

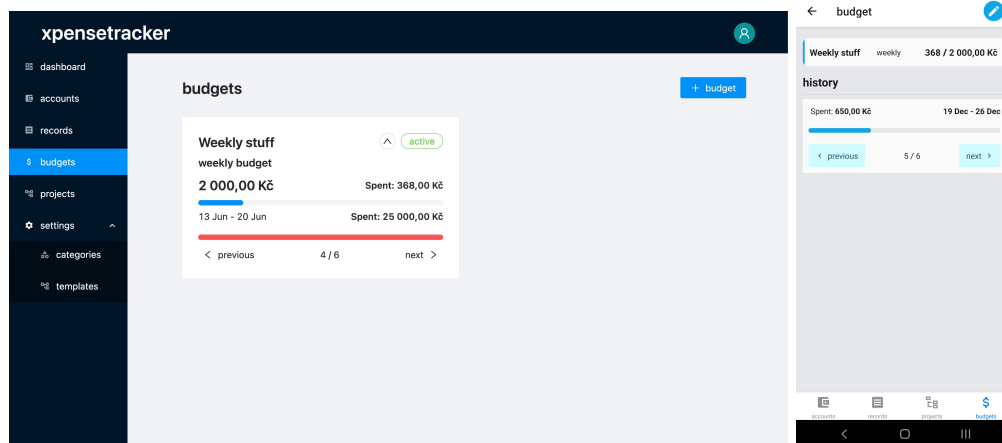
Management of budgets is handled in these views on the web and mobile apps.

**Web - Figure 4.8** Budgets are represented by card-like elements with various information about the budget. The card has a bar-like element that indicates the progress of the budget in terms of the used amount over the set amount. The state of the budget is represented by the chip-like button, this button can be used to activate or deactivate the budget after a confirmation pop-up. The historical data of the budget can be accessed by expanding the card using the button shown on mouse hover. The expanded part contains a slider-like component to show each period with a progress bar. Navigation is done using the text buttons on each side of the expanded part.

**Mobile - Figure 4.8** The implementation is similar to the web with small differences. The budgets are represented as a list of items and the progress of the budget is indicated with a filled background. On press of an item, the detailed view of the budget is shown. In this view a similar sliding element allows the user to navigate through the history of the budget with a progress bar included as in the web app.

## Categories and Templates

These resources are not meant to be used too often for this reason they are placed under the navigation group Settings. Their purpose is to make the usage of other resources easier. There



■ **Figure 4.8** Budgets view on the web and mobile respectively

is a set of categories set by default to each user and in general, it is not expected that users will make modifications frequently aside from a couple of tweaks according to their preference. As for templates they are meant to make the creation of transactions either automated or much simpler when done manually.

**Categories - Figure 4.6** Categories on the same level are placed horizontally in a stacking way. The order of categories solely depends on the order of creation. By default, an icon is assigned to each main category. To edit a specific category, it's enough to click the button shown on mouse hover. Clicking the category opens a new list of subcategories to the right of the category selected. Each category by definition can have multiple subcategories, this is why I decided to make categories expand horizontally rather than vertically. At the end of each list, a button is placed to allow the user to create new categories.

**Templates - Figure 4.6** Templates are represented by card-like elements as they contain more information for the user than other resources. Since there can be 2 types of templates: income and expense, the card background is represented by green or red respectively. For repeating templates, the information for frequency and interval can be revealed using the button placed under the amount of the template named **repeat**, this will expand the card vertically and show the necessary information. To edit or delete the template the edit button shown on the mouse hover can be used to activate the modal. For the creation of a new template, the button with the plus icon can be used for the action.

### 4.3.5 Forms

Forms are the essential components in client applications that communicate with the server for data exchange just like a mailbox we use in our daily lives. This is not different in my case, where for each resource type there should be a gateway to a specific API endpoint with a specific HTTP request. There are several forms to be created based on the requirements defined in the analysis chapter. Each form is responsible for creating, updating and deleting a specific resource.

**Account** Data needed from the user for this resource is only limited to:

- name - textual input field
- starting balance - numeric input field
- account type - dropdown menu with a limited number of options

- currency - dropdown menu with a limited number of options

The user is also able to update any of the fields in the case of an update of the resource, as the API allows it.

**Record** The form for this resource is the most complex one due to the variations of the data required from the user. The DTO for this resource differs mainly based on the record type and these types have some input fields in common:

- account - dropdown menu with existing payment accounts as options
- amount - numerical input field
- transaction and realization date - date and time pickers
- description - multi line textual input field
- cleared - checkbox input field to specify if the record has been completed and allows realization date input field to be filled.

Aside from the above, income and expense type of records have the following in common:

- category - dropdown menu with categories as options in hierarchical order
- template - dropdown menu with the pre-created non-repeating templates as options

The expense type record in addition to the above has the following inputs:

- project - dropdown menu with existing active projects as options
- split - a checkbox to enable the splitting of a transaction and show the split list
- split list - this is a group of input fields visible only if the splitting checkbox is checked that consists of a list of inputs grouped in pairs containing a category and amount field, to represent each splitting, and at the end of this list, a button to append a pair to the end of the list is positioned

The transfer type of record contains the following in addition to the common input fields mentioned at the beginning:

- to account - a dropdown menu with the list of available payment accounts as options

**Category** This is another form with a limited input fields:

- icon - a limited number of icons to choose from a modal pop up
- name - textual input field

**Budget** Form for budget creation, updating and deleting.

- name - textual input field
- categories - multi-select dropdown menu with categories as options in hierarchical order
- amount - numerical input field
- frequency - a dropdown menu with a list of available frequencies as options

**Project** Form for project creation, updating and deleting.

- name - textual input field
- starting date - date picker
- end date - date picker

**Template** This form allows user to create 2 types of templates: repeating and non-repeating. The repeating templates which refer to repeating transactions are handled by the server and no other action is required from the user after their creation. Non-repeating templates, as the name suggests are used to automatically fill the form data while creating a new record, to provide faster creation.

- name - textual input field
- amount - numerical input field
- category - dropdown menu with categories as options in hierarchical order
- account - dropdown menu with existing payment accounts as options
- project - dropdown menu with existing active projects as options
- description - multi line textual input field
- repeat - checkbox to enable a repeating template

If the repeating checkbox is checked, a group of input fields is shown to the user to specify the repeating for the template:

- repeat frequency - radio-tab-like input field with specific options: daily, weekly, monthly and yearly as per the API definition
- every - numerical input field to specify the interval of the frequency
- from and until date - date picker range to specify start and end date of repeating

All the above forms except the category form, are displayed in a modal with the trigger of a button. Category form is not shown in a modal because manipulations with categories require multiple actions and having a modal show up each time is not very UX-friendly also it made more sense to reveal it as a placeholder input group for quick updates or creation. The modal component used for wrapping the forms is a component that is provided by Ant Design and Native Base libraries respectively for the web and mobile.

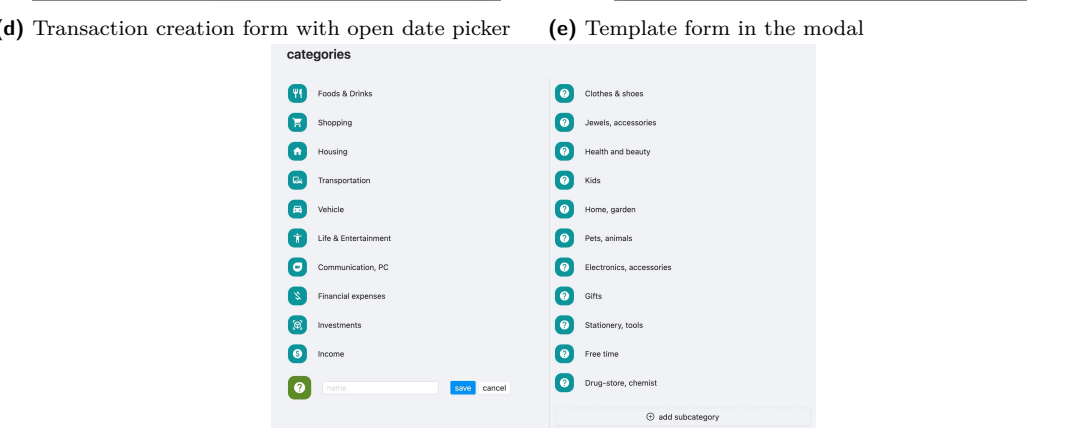
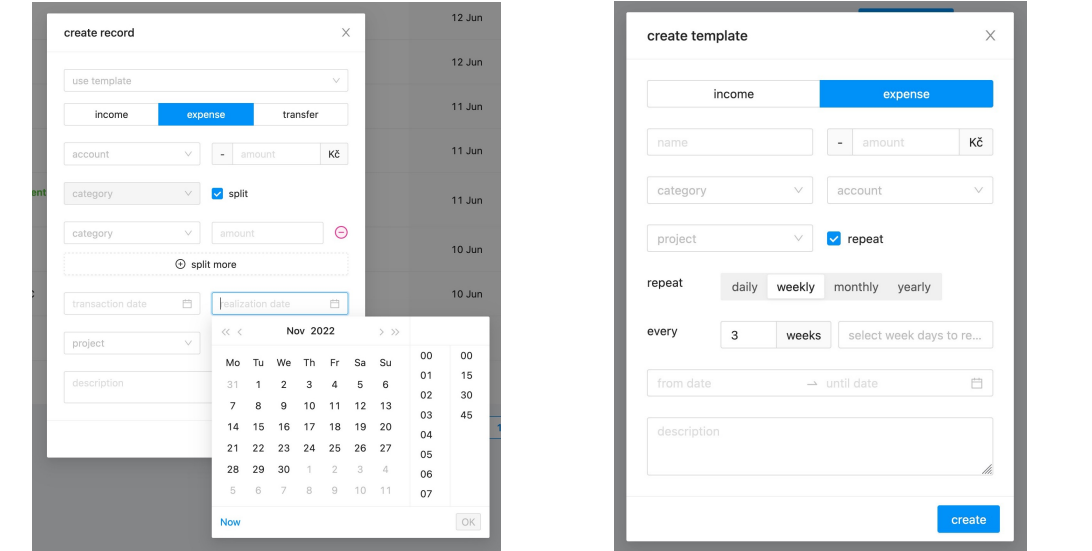
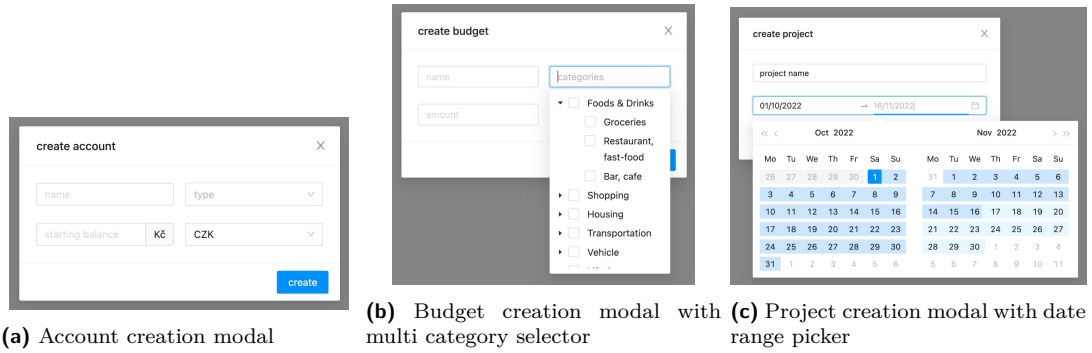
Forms require some implementation of validations and specific logic for showing different input types based on what the user needs to achieve. In the web app, Ant Design library provides custom hooks to be used with its form component for easier handling of the form data and different validation scenarios. For the mobile app, React Hook Form was used to achieve a similar result as in the web app, as the Native Base library does not provide specific form manipulations needed.

## Transaction splitting

Transaction or record splitting is one of the core functionalities that differentiate XpenseTracker from other applications. It is simply a functionality to allow the user to split a bigger expense type of transaction into smaller transactions for better categorisation and representation.

The splitting is enabled by checking the optional checkbox that displays a group of inputs to the user to start splitting. Inputs are paired as an item consisting of a category dropdown menu and a numerical input field for the amount of the subtransaction. Each of the inputs needs to be filled to continue with splitting. To add another split item, the button at the end of the list triggers a validation sequence to check if the fields are filled correctly: if validation passes it appends another item to the list of split items, if it does not pass it warns the user with a message at the end of the list. An update to the amount of a split item updates the total amount of the transaction automatically. If the total amount of split items is not equal to the total amount of the transaction, a warning message is shown to the user at the end of the list. Any split item can be removed from the list by using the button at the right side of the row.

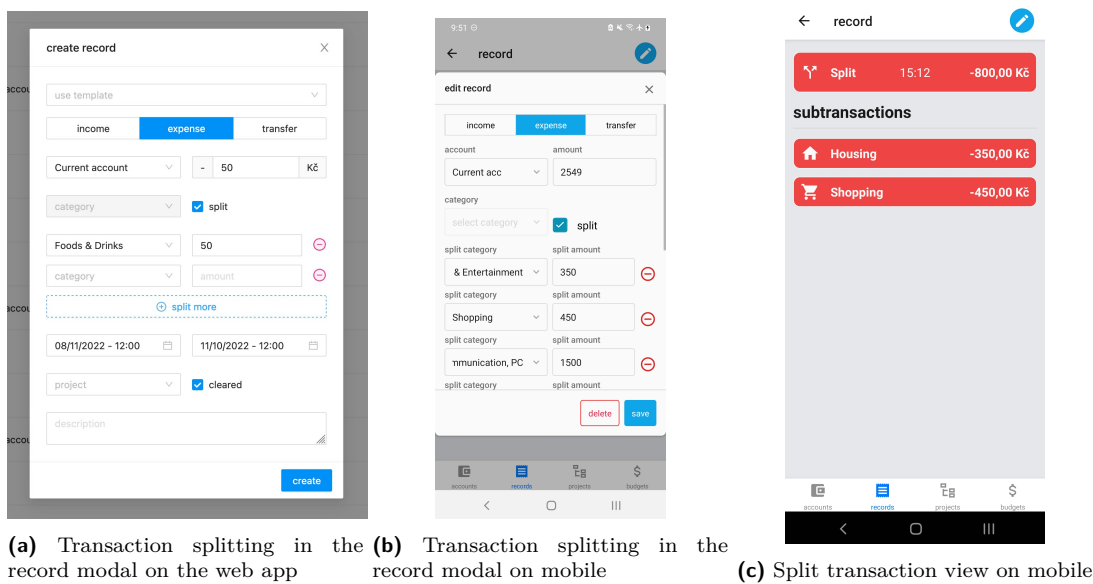
The main category for a transaction that is being split, is automatically set as **Split**, a special category not shown to the user. This category allows distinguishing these special transactions



(a) Account creation modal (b) Budget creation modal with multi category selector (c) Project creation modal with date range picker (d) Transaction creation form with open date picker (e) Template form in the modal (f) Category form as in place form type in the category list

■ Figure 4.9 Form implementations on web





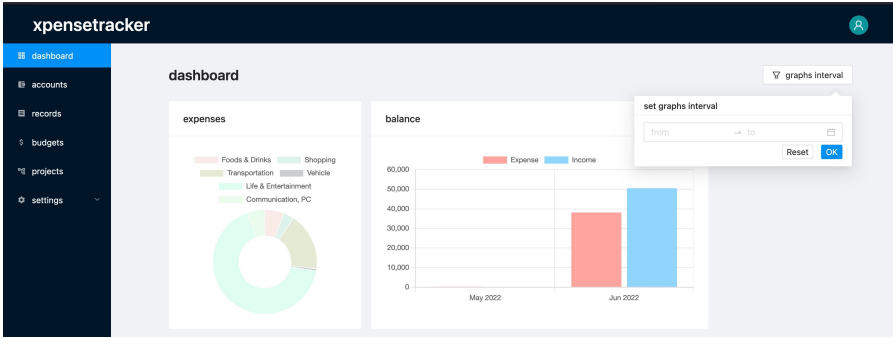
**Figure 4.10** Transaction splitting implementation

from others when it comes to listing all the transactions in the records view. A transaction with a **Split** category gets an icon appended at the beginning of the row to specify a collapsible/-expandable row to display subtransactions, this behaviour is slightly different on mobile due to the limited space: a split transaction shows the subtransactions in a separate view on the press of the transaction.


### 4.3.6 Charts

One of the functional requirements defined is the ability to visualise specific transactions through the help of charts and graphs. To fulfil this feature, I used a 3rd party package `Chart.js`, that helps with creating graphs. As it was decided in the design chapter, the graphs will be placed on the **Dashboard** view. For simplicity reasons, I decided to use pie and bar charts. Pie chart for showing how the expenses look based on main categories and bar chart to show expense and income transactions on a chosen date interval.

The implementation is fairly straightforward with little configuration needed. According to the documentation of the library, each chart type requires the data to be shaped in a specific format to be able to render the chart correctly. For this, each chart component has a function that parses the raw data passed as a prop to the component and returns the data ready to be consumed by the component imported from the library. The filter button on the top right corner of the dashboard view can be used to set the date range for the data used in the charts. The resulting components on the dashboard view can be seen in Figure 4.11.



■ Figure 4.11 Charts as seen on the dashboard view



## Chapter 5

# Testing

This chapter describes the testing strategies applied to verify the quality of the implemented software.

### 5.1 Manual testing

Manual testing is the most common type of testing in software development because it doesn't require extra effort and resources to perform. The downside of this type of testing becomes more obvious as the application grows and new features are added, as with each new feature the time required to test increases dramatically and it becomes more error-prone to perform repeated actions. On the other hand, automated testing is the exact opposite of manual testing, it takes time to implement in the beginning but it pays off in the late stages of the development. Taking into consideration the scope of the thesis and its requirements, I decided to focus mainly on manual testing and add automated tests only if there will be an opportunity later on.

The process followed for manual testing was simple. Every time a feature was implemented, I reproduced all the steps leading to that feature to verify that the implementation is working as expected. If there were bugs found during testing, the scenario leading to the bug would be added to the list of steps of testing for verification after it was fixed.

My supervisor also helped me with testing to verify that the applications were working as he would expect and he reported any bug or unexpected behaviour for resolving.

### 5.2 Usability testing

This is a type of non-functional software testing method, executed manually by real-life users to verify the entire application. Users perform specific tasks created by the facilitator that observe the users while they complete the requested tests. Usability testing is done by real-life users who are the potential candidates to use the application in the end once it is released to the public. Results obtained through testing help to identify issues with the user interface design, discover opportunities for improvement, and understand the behaviour and preferences of the intended audience.[34]

#### 5.2.1 Process

Users were asked to complete the following tasks mainly categorised into 2, web and mobile application-related tasks. Web application related tasks were performed on the demo website

and the Android phone provided with the mobile app installed was used for mobile related tasks. Tasks for usability testing were the following:

- Web application testing:
  1. Open the web application on <https://xpensetracker.vercel.app> and create a user account by registering.
  2. Create at least 2 payment accounts.
  3. Create a budget with any number of categories.
  4. Create a non-repeating template.
  5. Create a project.
  6. Create some transactions including these scenarios:
    - Create a transaction of income type.
    - Create a transaction of expense type.
    - Create a transaction with description filled in.
    - Split a transaction.
    - Create a transaction using a template.
    - Delete a transaction.
  7. Use filtering options to filter out transactions created in the previous step.
  8. Investigate the charts in dashboard view.
- Mobile application testing:
  1. Open the mobile application and use the same account from the web application to login.
  2. Add more transactions including these scenarios:
    - Create a transaction of type transfer.
    - Create a transaction with a project assigned.
    - Create a transaction using a template.
    - Split a transaction.
  3. Investigate budgets progress in budgets.
  4. Investigate progress in payment accounts.

Once the tasks were completed by the users, I asked the following questions to conclude the testing:

1. How would you describe your overall experience with the applications?
2. How did you find the interface of applications?
3. What do you think could be changed or improved in any of the applications?

## 5.2.2 Outcomes

### User 1:

1. "Nothing to complain, good enough for this level in my opinion."
2. "I believe it was a standard UI, something I was used to."
3. "The mobile application was a little bit slower than the web app, so probably that can be improved."

### User 2:

1. "In general, I can give positive feedback even though I don't have much experience with such types of applications. However, I can say that the web app was easier and clearer to use, I can't say the same for the mobile app."
2. "I liked the web app more than the mobile app, mainly because of the looks and performance being better."
3. "Charts could be improved or new ones can be added."

### User 3:

1. "In the past, I used to use Wallet and found the ability to split transactions to be a very useful feature in XpenseTracker. While I still enjoyed using the application and appreciate the features it offers that are not available in Wallet, there are still aspects of the mobile app that could be improved. As someone who primarily uses mobile apps, I believe it could be improved more."
2. "Not many complaints about the web app and as long as the mobile app catches up with the features provided by the web app."
3. "I wasn't satisfied with the management of resources in the mobile app, possibly because I was used to using Wallet. Also, the need to open the modal for the deletion of transactions is not very UX friendly, swiping the item to show delete buttons would be better in my opinion."

### User 4:

1. "I found the web application to be more feature rich and have a better UI than the mobile app, which I would expect the opposite in today's standards. Nevertheless, I can say they were not complicated to use and seemed to have relatively good UI."
2. "The interface on the web app was easy to follow, however, the mobile app has room for improvement. Such as replacing modals and better performance."
3. "Definitely, the mobile app needs more work to be on par with the web app. Can't think of much to improve on the web app."

### User 5:

1. "Overall good experience with the apps and accomplishes what they intend to do."
2. "The web app was clearly easier to use compared to the mobile app. In general, the interface was pleasant enough."
3. "A notification system for letting the user know of the outcome of certain actions can add value in my opinion."

## 5.2.3 Conclusions

Users generally had positive feedback, but some expressed disappointment with the quality of the mobile application, expecting it to be on par with the web application. This was understandable given the high standards of mobile apps today. However, the lower quality of the mobile app was due to the decision to prioritize the web app, as agreed upon with my supervisor.

The testing also yielded suggestions for improvement, including the implementation of a notification system to inform users of the outcome of their actions, which would enhance the

user experience. Another suggestion was to allow users to interact with resources using swiping, which would streamline certain tasks by eliminating the need to open a modal.



found in other similar applications. It can be confidently stated that the XpenseTracker clients are a viable option in the market.

## Future improvements

The testing results indicated that users were generally unhappy with the speed at which form submissions were processed in both the web and mobile clients. In addition, users of the mobile app were dissatisfied with the way forms were displayed in a modal on the same screen and the overall performance of the user interface.

To enhance the overall performance of server communication, it will be necessary to make changes primarily on the server side, which should be discussed with the creator of the application. Based on user feedback about the forms in the mobile app, they could be placed on their own screen to provide more space for use rather than being displayed in a modal with limited space. Additionally, improving the way data is retrieved from the server can significantly improve the rendering of items and reduce the load on the processor, thereby improving the performance of the mobile user interface.



# Bibliography

1. WIKIPEDIA CONTRIBUTORS. *Systems development life cycle* — *Wikipedia, The Free Encyclopedia* [[https://en.wikipedia.org/w/index.php?title=Systems\\_development\\_life\\_cycle&oldid=1117629194](https://en.wikipedia.org/w/index.php?title=Systems_development_life_cycle&oldid=1117629194)]. 2022. [Online; accessed 8-October-2022].
2. AZIZOV, Jamaladdin. *ExpenseTracker - personal finance manager*. 2021. Bachelor's Thesis. Czech Technical University.
3. WIKIPEDIA CONTRIBUTORS. *Separation of concerns* — *Wikipedia, The Free Encyclopedia* [[https://en.wikipedia.org/w/index.php?title=Separation\\_of\\_concerns&oldid=1118745808](https://en.wikipedia.org/w/index.php?title=Separation_of_concerns&oldid=1118745808)]. 2022. [Online; accessed 12-October-2022].
4. MICROSOFT. *Architectural principles* [<https://learn.microsoft.com/en-us/dotnet/architecture/modern-web-apps-azure/architectural-principles>]. 2022. [Online; accessed 21-December-2022].
5. BUDGETBAKERS. *Wallet* [<https://budgetbakers.com>]. 2022. [Online; accessed 10-October-2022].
6. SPENDEE. *Spendee* [<https://www.spendee.com>]. 2022. [Online; accessed 10-October-2022].
7. SILVERWIZ LLC. *MoneyWiz 3* [<https://apps.apple.com/us/app/moneywiz-3-personal-finance/id1004710719>]. 2021. [Online; accessed 10-October-2022].
8. BLUECAT. *financisto* [<https://play.google.com/store/apps/details?id=com.bluecatsoftware.financisto>]. 2022. [Online; accessed 10-October-2022].
9. DAVE WOOD. *Interface Design. An Introduction to Visual Communication in UI Design*. Bloomsbury Publishing Plc, 2014.
10. JON YABLONSKI. *Laws of UX*. O'Reilly Media, Inc., 2020.
11. DIANA MACDONALD. *Practical UI Patterns for Design Systems: Fast-Track Interaction Design for a Seamless User Experience*. Apress, 2019.
12. SNOW HU. *Vertical Navigation Menu - Everything You Should Know* [<https://www.mockplus.com/blog/post/vertical-navigation-menu>]. 2021. [Online; accessed 08-December-2022].
13. PAGE LAUBHEIMER. *Left-Side Vertical Navigation on Desktop: Scalable, Responsive, and Easy to Scan* [<https://www.nngroup.com/articles/vertical-nav/>]. 2021. [Online; accessed 08-December-2022].
14. RALUCA BUDIU. *Basic Patterns for Mobile Navigation: A Primer* [<https://www.nngroup.com/articles/mobile-navigation-patterns>]. 2015. [Online; accessed 08-December-2022].

15. SNOW HU. *Modal Web Design Guide & Inspiration for Designers* [<https://www.mockplus.com/blog/post/modal-design-inspiration>]. 2021. [Online; accessed 08-December-2022].
16. PAGE LAUBHEIMER. *Cards: UI-Component Definition* [<https://www.nngroup.com/articles/cards-component/>]. 2016. [Online; accessed 08-December-2022].
17. PABLO PEREA, PAU GINER. *UX Design for Mobile*. Packt Publishing, 2017.
18. BALSAMIQ. *Balsamiq* [<https://balsamiq.com>]. 2022. [Online; accessed 13-November-2022].
19. REDUXJS. *Redux* [<https://github.com/reduxjs/redux>]. 2022. [Online; accessed 09-December-2022].
20. REDUXJS. *Redux Toolkit* [<https://github.com/reduxjs/redux-toolkit>]. 2022. [Online; accessed 09-December-2022].
21. REACT ROUTER. *Client Side Routing - React Router* [<https://github.com/remix-run/react-router/blob/main/docs/start/overview.md>]. 2022. [Online; accessed 12-October-2022].
22. REACT NATIVE. *Navigating Between Screens* [<https://reactnative.dev/docs/navigation>]. 2022. [Online; accessed 09-December-2022].
23. REACT NAVIGATION. *React Navigation* [<https://reactnavigation.org/>]. 2022. [Online; accessed 09-December-2022].
24. ANT-DESIGN. *Ant Design* [<https://github.com/ant-design/ant-design>]. 2022. [Online; accessed 09-December-2022].
25. GEEKYANTS. *Universal Components for React & React Native* [<https://github.com/GeekyAnts/nativebase>]. 2022. [Online; accessed 09-December-2022].
26. EMMIT A. SCOTT, JR. *SPA Design and Architecture*. Manning Publications Co., 2016.
27. GIT. *Git -distributed-is-the-new-centralized* [<https://git-scm.com/>]. 2022. [Online; accessed 13-November-2022].
28. WIKIPEDIA CONTRIBUTORS. *Technical debt — Wikipedia, The Free Encyclopedia* [[https://en.wikipedia.org/w/index.php?title=Technical\\_debt&oldid=1122868481](https://en.wikipedia.org/w/index.php?title=Technical_debt&oldid=1122868481)]. 2022. [Online; accessed 15-December-2022].
29. REACT NATIVE. *Setting up the development environment* [<https://reactnative.dev/docs/environment-setup>]. 2022. [Online; accessed 09-December-2022].
30. EXPO. *What is Expo* [<https://docs.expo.dev/introduction/expo/>]. 2022. [Online; accessed 09-December-2022].
31. REACT HOOK FORM. *React Hook Form* [<https://github.com/react-hook-form/react-hook-form>]. 2022. [Online; accessed 09-December-2022].
32. REACTCHARTJS. *react-chartjs-2* [<https://github.com/reactchartjs/react-chartjs-2>]. 2022. [Online; accessed 09-December-2022].
33. VERCEL. *How to Deploy a React Site with Vercel* [<https://vercel.com/guides/deploying-react-with-vercel>]. 2022. [Online; accessed 09-December-2022].
34. KATE MORAN. *Usability Testing 101* [<https://www.nngroup.com/articles/usability-testing-101/>]. 2019. [Online; accessed 31-December-2022].

# Appendix A

## Installation guide

This guide outlines the requirements and steps for installing the XpenseTracker client application, to be used for further support and development.

### A.1 Requirements

The following tools are required to be installed on your machine before going forward with the setup of both web and mobile development:

- Node.js v16.17.0 or higher - Required to install package manager.
- npm v8.15.0 or higher - Required to install application dependencies.

### A.2 Web

1. Run the following command in the web application source code directory `src/app/web` to copy content from `.env.example` to `.env` to load the environment variables that will be used in the application: `cp .env.example .env`
2. The following command needs to be executed to install project dependencies: `npm install`
3. Next, run `npm start` to start the local development server.
4. The local development server can be accessed from the browser at: `http://localhost:3000`
5. To create a build after making changes to the code, run `npm run build`. This command will generate a `dist` folder containing the deployment ready application.

### A.3 Mobile

#### A.3.1 Requirements

There are some additional requirements to setup the development environment locally:

- watchman - Required for macOS and Linux users, instructions for installation are present in <https://facebook.github.io/watchman/docs/install#buildinstall>

- Expo Go - Required to run the application on a mobile device without the need for an emulator:
  - Android: <https://play.google.com/store/apps/details?id=host.exp.exponent&pli=1>
  - iOS: <https://apps.apple.com/app/expo-go/id982107779>

If the above requirements are fulfilled it's possible to continue with the installation of mobile application locally.

1. Run the following command in the mobile application source code directory `src/app/mobile` to copy content from `.env.example` to `.env` to load the environment variables that will be used in the application: `cp .env.example .env`
2. The following command needs to be executed to install project dependencies: `npm install`
3. Next, run `npm start` to start the local development server.
4. Follow the instructions on the command line to open the mobile application on the Expo Go client.

## Appendix B

# Deployment guide

This guide explains the process of creating packages that can be easily deployed or installed.

### B.1 Web

1. If you wish to create a fresh build and use it for deployment then please follow the instructions in the installation guide.
2. To deploy the web application on a production server all you need to do is to extract the compressed package inside the public folder of your server.
3. You can access the application using a browser at URL `http://yourserver.url/`.

### B.2 Mobile

1. If you wish to create fresh builds aside from the one included in the attached media of this thesis, it is necessary to follow extra steps:
  - a. Follow the instructions in the installation guide to setup your local environment to continue with creation of the executable.
  - b. Create a user account at `https://expo.dev/`.
  - c. Run the following command in the project folder: `npm install -g eas-cli`
  - d. Login to EAS Build by running in the project folder: `eas login`
  - e. Run `eas build -p android --profile preview` to start the process of building.
  - f. Follow the instructions on the command line to download your executable.
2. To install the application on a mobile device, simply copy the `.apk` executable in the attached media of the thesis or the one downloaded in the previous step to your Android device and open it for installation. Once the application is installed on your device simply open it to access the XpenseTracker mobile application.



## Content of enclosed media

dist.....	Distributables
├─ XpenseTracker-v1.0.0.apk .....	Android installation package
├─ XpenseTracker-v1.0.0.zip.....	Web installation package
src.....	Source codes directory
├─ app.....	Source codes for applications
│   ├─ mobile.....	Source code of the mobile application
│   └─ web.....	Source code of the web application
└─ thesis.....	Source code of the thesis
text.....	Text work
├─ deployment.pdf .....	Deployment guide in PDF format
├─ installation.pdf .....	Installation guide in PDF format
└─ thesis.pdf.....	Thesis in PDF format