FACULTY OF INFORMATION TECHNOLOGY CTU IN PRAGUE

Assignment of bachelor's thesis

Title:	A web application for group expense management
Student:	Quynh Chi Nguyen
Supervisor:	Ing. Josef Pavlíček, Ph.D.
Study program:	Informatics
Branch / specialization:	Web and Software Engineering, specialization Web Engineering
Department:	Department of Software Engineering
Validity:	until the end of summer semester 2022/2023

Instructions

The aim of this thesis is to design, implement and test out a web application for managing group expenses in Java/React. The application allows users to manage group expenses, track and split them automatically evenly or manually. The application will be able to generate statistics of the expenses.

There are already several existing solutions for this matter such as SettleUp, Splitwise. The student analyzes these solutions and makes their own with possible improvements (eg. generating QR codes for payment).

- Research on the existing solutions
- UI/UX design
- Implement the web application (frontend and backend)
- Usability testing and deploying the application

Electronically approved by Ing. Jaroslav Kuchař, Ph.D. on 30 November 2021 in Prague.

Bachelor's thesis

A WEB APPLICATION FOR GROUP EXPENSE MANAGEMENT

Faculty of Information Technology
Department of Software Engineering
Supervisor: Ing. Josef Pavlíček, Ph.D.
June 22, 2022

Contents

List of Figures

List of Tables

List of code listings

I would like to express my gratitude to my supervisor, Ing. Josef Pavlíček, Ph.D, for his support and advisement on my thesis. I also wish to thank the testers for their time and valuable feedback. Special thanks to my family and friends for the support and encouragement they have given me throughout my studies and the writing of this bachelor's thesis. Lastly, I would like to acknowledge my flatmates, with whose sympathy and support in the last few months, I was able to stay motivated and finish this work.

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 June 22, 2022 .....

Abstrakt

Cílem této práce je vytvořit webovou aplikaci umožňující uživatelům rozdělovat výdaje a spravovat dluhy ve skupinách. Na tento problém již existují několik řešení, avšak žádné z nich není dostatečně uspokojující z pohledu kombinace funkcionality a designu. Tato práce vznikla jako snaha vytvořit takovou aplikaci, která bude splňovat podmínky jednoduchého, uživatelsky přívětivého designu, a zároveň bude schopna uživatelům poskytovat především hlavní funkcionality, bez kterých by se aplikace zaměřená na správu a rozdělování skupinových výdajů neobešla.

Práce obsahuje analýzu vybraných existujících řešení, výsledky průzkumu uživatelů, proces UI/UX designu, specifikace požadavků, návrh architektury aplikace, implementace, nasazení, uživatelské testování a na konec diskuzi o možném vylepšení a budoucím vývoji.

Klíčová slova webová aplikace, správa skupinových výdajů, UI/UX design, Java, React

Abstract

The aim of this thesis is to create a web application that allows users to split expenses and manage debts in groups. There are already many existing solutions for this problem; however, none of them is fulfilling enough in terms of functionality and design combination. This thesis was created as an effort to make an application that meets the requirements for a simple, user-friendly design and at the same time is able to provide, first and foremost, basic functionalities of a group expense manager to the users.

This thesis contains analyses of selected existing solutions, user research, UI/UX design processes, requirements specification, application architecture design, implementation, deployment and usability testing, and lastly, further development and improvements.

Keywords web application, group expense management, UI/UX design, Java, React

Acronyms and abbreviations

ID	Identification
QR	Quick Response
CSV	Comma Separated Values
OS	Operating system
GPS	Global Positioning System
UI	User interface
UX	User experience
UC	Use case
CTA	Call to action
HTML	Hypertext Markup Language
CSS	Cascading Styling Sheets
JS	JavaScript
FE	Front end
BE	Back end
API	Application Programming Interface
DBS	Database system
DB	Database
IDE	Integrated Development Environment
JPA	Jakarta Persistence API
ORM	Object-relational mapping
OOP	Object-oriented programming
DTO	Data Transfer Objects
POJO	Plain Old Java Object
HQL	Hibernate Query Language
SQL	Structured Query Language
URL	Uniform Resource Locator
URI	Uniform Resource Identifier
JSON	JavaScript Object Notation
JWT	JSON Web Token
JSX	JavaScript Syntax Extension
XML	Extensible Markup Language
CORS	Cross-Origin Resource Sharing
CLI	Command Line Interface

Introduction

Splitting expenses and resolving debts was never an easy task, and sometimes, it may end up splitting friendships and relationships instead. With today's modern technology, it is not hard to imagine using an application to manage, organize and hence reduce all the stress one might face in such situations.

This thesis aims to create a web application for managing group expenses or debts, allowing users to manage, track, split expenses and settle debts. The application might be used by a wide range of users. It is great for splitting rents and bills between roommates, expenses from a vacation, group trips or events, couples sharing relationships costs, or just loans and debts between friends, relatives…

The topic was chosen for the dissatisfaction with existing solutions as an effort to make a new one with the main, most used features, which will focus on a clear, minimalist, user-friendly design for a better user experience.

The first chapter of this thesis contains analysis of existing solutions and a short comparison of web and mobile applications. Chapter 2 serves as a brief introduction to the world of user experience (UX), bringing the readers closer to the UX design process, specifically the user research stage – the first step to building empathy with the users. In chapter 3, functional and non-functional requirements are specified, along with use case scenarios that describe the user's behavior and interactions with the application. The next chapter is dedicated to UI design, where the visual elements are described and explained. The processes of prototyping and user testing results are also discussed here.

In chapter 5, readers can learn about the development process, along with the used technologies and practices. The following chapter includes the deployment and usability testing. Finally, a discussion about possible improvements and further development in the future is provided in the last chapter.

Chapter 1

Analysis

Splitting expenses and resolving debts are no strange experiences for anyone. Hence it is no surprise that there are already many existing solutions for this problem. This chapter provides further analysis of three selected solutions to find out what they offer, what is good, and what could be improved.

The differences between web and mobile applications are also discussed since it is an interesting question as nowadays, mobile applications are significantly more popular than web applications.

1.1 Existing solutions

Most of the existing solutions provide multi-platform support for both smartphones and the web (or even desktop). There are also applications of other types like banking or social media…that offer features for expense splitting. Because the goal of this thesis is to implement a web application, this analysis will only focus on the web version of applications that target this problem purely. The analysis is based on features presented in the free version of the solutions. However, deficiencies (if there were any) listed in this subchapter apply for both free and paid versions, at least at the time of this thesis's writing.

All the web applications share many similarities. As an application for management, they all offer basic features such as allowing users to create and manage groups and group members, add an expense record, split the expense, and modify it. In the following text, further analysis of selected applications will be covered.

1.1.1 Settle Up

Settle Up[1] is a Czech application that has a simple design and is easy to use. The application claims to be focused on design and user experience, which allows for better orientation.

The design is minimalist and colorful. Users can create and manage groups with different color schemes chosen by themselves. This has a good impact on the users since they can quickly distinguish different groups just by the sight of the colors. The layout is intuitive, it is easy to find out how to perform an action, and there seem to be no unnecessary elements.

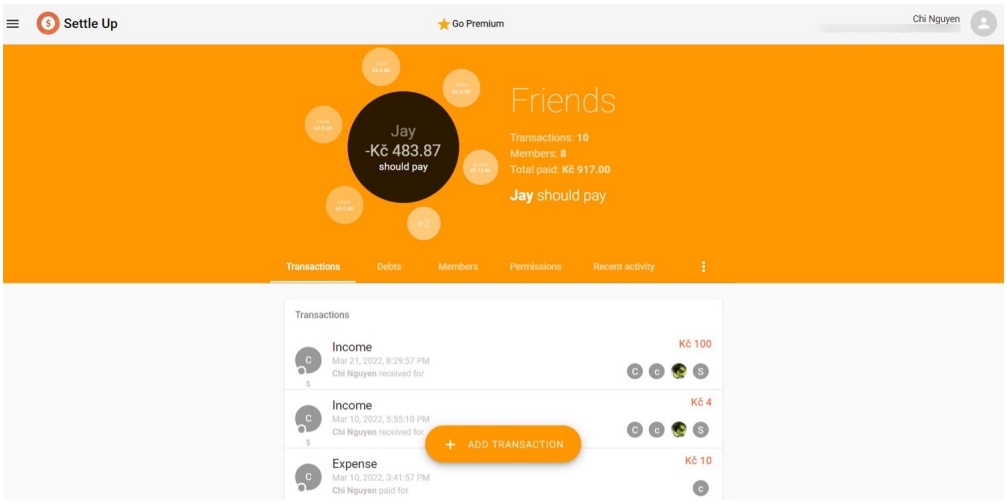

Figure 1.1 Application screenshot – Group view

The web application offers a couple more than just the basic functionalities mentioned above, like debt simplification, which allows the debt to be transferred from one to another, minimizing the number of transfers needed between members. Users can also add a Czech bank account for QR[1] payment code generation, or export a CSV[2] file containing information of all group transactions. *Who pays next* is one of the unique features, where the application suggests who should pay next by simply showing the member who owes the most money in the group on the main page.

The application does not require all the members to have an account in the app, which means the entire management may be taken care of by one person. It is possible to add a member with a registered account who will have access to the group and be able to make changes. Recently, there was a feature added which allows permission setting, so a group owner may set the rights for each registered member (read-only, edit permission…).

However, the application does not allow users to split expenses manually with different settings, which could be considered a significant disadvantage. Users can only split bills equally or by shares, which means the original amount will be divided into even parts and unequally assigned to each member. This, however, is offered in the mobile version, where there are far more features.

1.1.2 Splitwise

Splitwise[2] is a very well-known group expense manager for the huge range of features it offers. Aside from the main features, there are many other complementary functions such as expense repeat/reminder, adding friends, recording friend's debt without a group, add comments…. This analysis considers the most-used key features particularly.

The web application has little to no differences in functionality compared to the mobile version. The application allows users to split expenses and reimbursements of many currencies equally, by shares, by exact amounts, or by percentage. The user may itemize the whole expense for better clarity, so each member can see exactly what they are paying for. Debt simplification or multiple payers for one bill are also helpful features offered in *Splitwise*. The application allows users to settle debts by adding a transfer record or directly sending the money through the Paypal payment gateway.

[1] Quick Response
[2] Comma Separated Values

The size of the application is, however, its weakness. The biggest drawback of the application is its confusing, not user-friendly interface. There is too much information packed in one single page which distracts the user's attention and leads to the difficulty of finding out how to perform an action. The web page has a three-column layout with the two side columns being static, showing lists of friends and groups, sharing box, promotion..., and the middle column changing according to the user's interaction. The advantage of this approach is that the user has almost all the information in just one page, hence the layout does not change too much per click, giving a more secure feeling while using the app. However, this layout can be very confusing for new or occasional users.

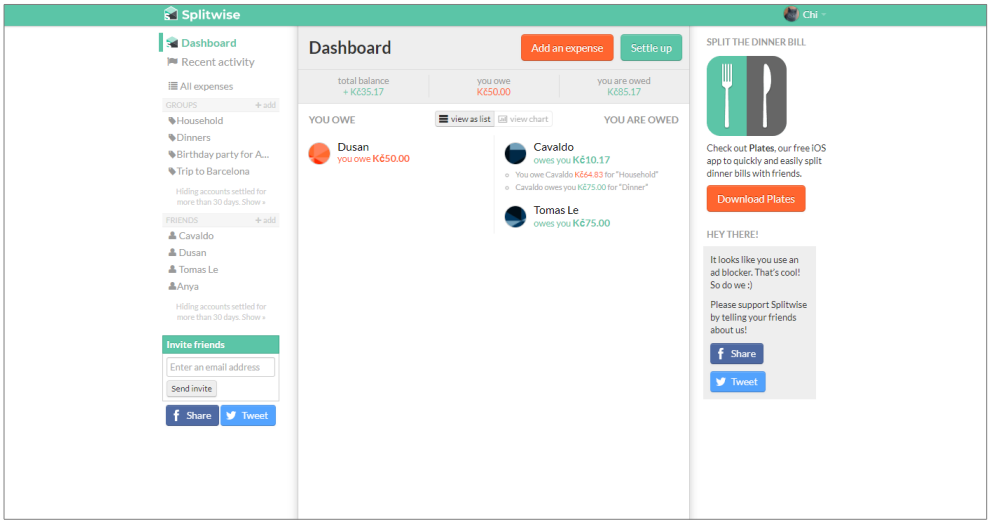

Figure 1.2 Application screenshot – Homepage after login

This problem is *partially* solved by the two buttons on the Homepage, which stand out to draw the user's attention to the most important actions: adding an expense and settling up – a screenshot can be seen in Figure 1.2. These buttons use the main colors of the application: #5BC5A7 Medium Aquamarine (greenish color) and #FF652F Outrageous Orange. They stand out well in either white or gray backgrounds. This color combination is convenient because green may symbolize a safe action or positive information and orange dangerous action or negative information. However, too small font and wrong color choice with low contrast in some places still significantly reduce visibility and overall make the user experience rather bad.

Another cons to be mentioned is the requirement of having an account in *Splitwise* to be present in a group. In order to add a participant to an expense record, users must enter an email address, to which an invitation will be sent. This problem can be solved by adding a member with a random or non-existent email. However, this has several disadvantages: it is time-consuming for a user to perform an extra unnecessary action, it is not very convenient as some people certainly do not want to bother a stranger with a – who knows – matching email, and providing a real email address as a piece of personal information might not be very pleasant to some people.

In conclusion, *Splitwise* might be the best choice in its category for its huge range of features, but overall, the design looks quite outdated and is not too user-friendly.

1.1.3 Sesterce

Sesterce[3] is much less known than the other two discussed above. This application was chosen to analyze in this text mainly because of its simple, modern appearance as an example of an application with a nice design but not fulfilling functionality. It has an intuitive user interface.

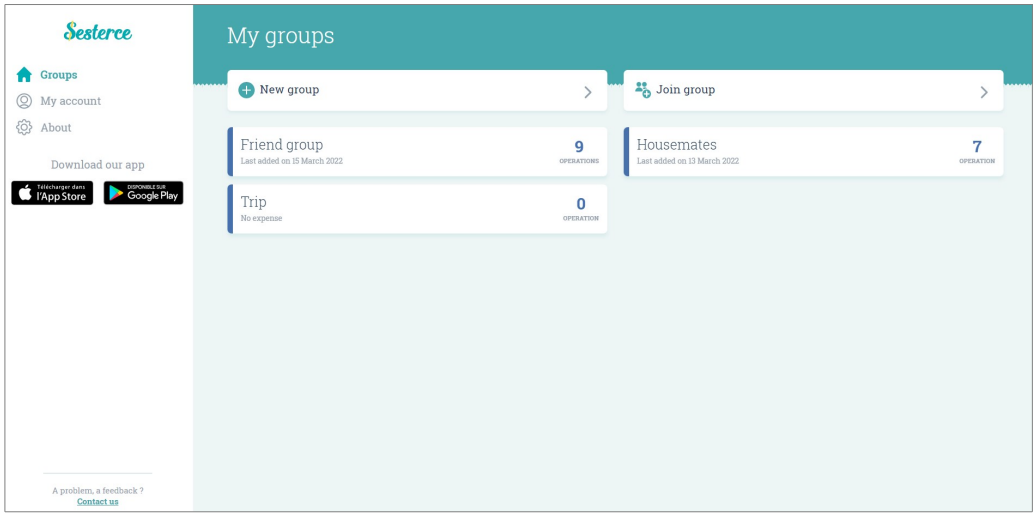

Figure 1.3 Application screenshot – Homepage

Every element is visible with high contrast, and the information is well communicated to the users.

The application does not require users to have an account. It allows users to try the app anonymously and create an account with an email, although there are no login pages for some reason.

Each group has a unique ID[3], hence everyone who has the ID (and password if set) should be able to access the group with all rights to read and edit, which unfortunately means details of an activity log can not be recorded as everyone has the same rights and can be anonymous. Unfortunately, there is possibly a bug where the application returns an error of not matching ID or password in every attempt to join a group. There are also no options to set a new password in case the old one is lost/forgotten. Overall, this application is not well optimized for synchronization and is rather suitable for local use.

Sesterce offers splitting expense/income only by shares or fixed amounts. An expense can be paid by multiple members, and there is also debt simplification. The nice thing is, nonetheless, that this application allows a combination of splitting types, which means users can assign a fixed amount to one person and let the app calculate the amounts based on assigned shares to others.

Overall, the web application feels friendly and usable; some bugs might be removed but for the essential issues probably less convenient to use compared to the other solutions.

1.2 Mobile application vs Web application

With the rapid growth and expansion of mobile phones and tablets, it is no surprise that mobile applications are gaining more and more popularity for their convenience. The questions arise: Why do people bother with web apps when mobile apps are significantly more preferred? Are they not out of date? Is it not better to create a mobile app rather than a web app that should work on different devices and screen sizes?

This subchapter is dedicated to answering the questions mentioned above and to explaining the choice of application type this thesis targets.

[3]Identification

1.2.1 Mobile application

A mobile application is a native application developed for a specific mobile OS[4] such as Google's Android and Apple's iOS. Different programming languages are used for different OS: Android apps mainly use Java or Kotlin, while iOS apps mainly use Objective-C or Swift.[4]

Mobile applications are rightfully favorable for their undeniable convenience. One of the most significant advantages against web apps is their ability to operate offline; thus, they can be faster than web apps, which require constant internet connection. An application is downloaded and installed to the mobile device; hence it can make use of all OS and device functions such as GPS[5], camera, notification…. A mobile application can be found in App Store, where the updates can also be downloaded. These apps must always be approved by the App Stores so they are usually more secure.

However, mobile apps are, logically, more expensive to develop compared to web apps. To create a native mobile application, at least two separate applications should be written (if the goal was to reach as many users as possible), which results in increased development costs. They also require memory storage and sometimes do not support older OS versions.

1.2.2 Web application

A web application runs in a web browser, usually created using HTML[6], CSS[7] and Javascript. They do not need to be downloaded or installed, so they can be used cross-platform with no memory storage required. The same web application can be accessed from various devices (desktop, tablet, mobile phone…, or even smart TV!) just via a web browser.

Compared to mobile apps, web applications are easier to develop and maintain, as updates are done on the server, not on every client's device. Thus, developing one fully responsive web application proves to be much faster than multiple platform-specific ones. A web app can also be launched independently and quickly since no App Store approval is needed. It can be easily accessed, providing the users an option to try it out without downloading. Other advantages of web applications include better view with a larger screen, more space for bigger font, it is more comfortable to input data using a keyboard and a mouse, possible multitasking….

Despite being less popular, web applications are still relevant, widely used, and even favored by some individuals. With an idea in mind, the choice of application type depends on many factors and might not be easy as each type offers different benefits. However, when it comes to simplicity and cost-efficiency, a web application can not disappoint.

[4] Operating system
[5] Global Positioning System
[6] Hypertext Markup Language – a language used to structure web pages
[7] Cascading Styling Sheets – a language used to style web pages

Chapter 2

User experience

When people hear the word "design", they often associate it with visual content creation, arrangement of colors and shapes. Then what exactly is UX[1] design? Unlike UI[2] which *"is essentially a conversation between users and a product to perform tasks that achieve users' goals. […] It is what users see and feel directly when using a product"*[5], UX literally means the overall feeling a user may experience while interacting with the product. The objective of UX design is, therefore, to design an interface that offers users a great experience. A good product benefits from both UI and UX design principles.

A deeper insight into UX design will be presented below, starting with the design process stages. In the previous chapter, a *preview* of user research was introduced to the readers. Throughout this chapter, the process of getting to know the target users will be applied and explained.

2.1 User-centered design

User-centered design, or UCD, is a design framework in which the designer focuses on the user and considers their feelings, story and emotions. This information can be gathered in the user research phase. UCD process usually consists of five steps:

Define In this phase, designers and stakeholders brainstorm and define the product concept.

Research User research in the form of surveys and interviews is usually conducted in this phase in order to empathize with the end-users and determine what they want, need, and expect from the product. Market research is also included in this phase.

Ideate With a concept and results obtained from user research, it is time for analysis and reevaluation of the product. Requirements are usually defined in this phase.

Design This is the stage when the UI design works are done. A prototype of the final product is created based on the results of the previous stages.

Test Testing with the target user is crucial, especially when it comes to a *user centered design*. In this phase, the prototype is tested with users. After the positive feedback evaluation, the product can finally go into development.[6]

As an iterative design process, these steps can loop over and over until the prototype fulfills the user's needs and expectations.

[1] User experience
[2] User interface

2.2 User research

After the analysis of existing solutions is made, it is time for user research. In order to identify what users need and expect from a group expense manager, a survey was made and sent to different places (friends, family, online forums…).

Out of 89 participants, 87.6 % is made of people in age 20–29. The rest comprises people aged 15–19 and 30–59. The demography shows that more young adults are interested in this type of application. After discussions with older relatives and acquaintances, it is clear that middle-aged people rarely split bills, which corresponds with the answers from people over 40 in the survey.

38 % of participants already use an application to manage debts and split expenses. 73.5 % of them prefer using a mobile app, 5.9 % prefer using a web app, and the rest would use both versions. The frequent reason for using a mobile app is convenience because people are on their phones more often. Also, it is easier to use when the internet connection is (for some places even constantly) bad. On the other hand, some say it is more convenient to use a computer/web app. In the end, it is all about personal preference and opinion. However, the answers match with the fact that mobile applications are the favored choice for most people.

The survey also asked what users like and dislike about the application they are currently using (if they are already an app user) and the features they use the most or find useful. The second question was also asked to people who do not use any applications for debt records or split expenses. These people often use paper and pen to do the work, some use Excel, and others do not share bills.

2.3 Personas

Persona is a fictional character with a realistic portrayal of a target user. Personas are created at the beginning of the design process to empathize with the users and identify required functions for the product. This can be achieved by implementing common behavioral patterns and building a mental model for each type of user which the personas correspond to.[7]

From the results gained through the survey, three personas were created which portray three types of users this application is (or is not) designed for. This helps narrow down the target user group for whom the design should focus.

2.3.1 Persona A

Name: Emma
Age: 23
Gender: Female
Hobbies: Coffee dates with friends, attending music events, traveling
Typical day: Emma is a university student. She wakes up at 8 AM every morning and goes to the university by public transport. Her classes usually start around 9 AM and end in the afternoon. After school, she often studies to keep up with her classes, and in her free time, she meets up with friends. On weekends, she usually works at a shop from morning to late afternoon to earn money to pay for her living. She lives with her friends in a rented flat. Because she shares common household expenses like rent, cleaning tools, food,…with her five flatmates, expense splitting is a part of her daily life. Sometimes, she also needs to split expenses when going out with friends. She uses a smartphone which is constantly out of storage, as she keeps a lot of photos, songs, and many other things on her phone. She prefers using a web browser to access the application. She uses both a computer and a mobile phone.
Short history: Emma is friendly, energetic and enjoys spending her time with others. After high school graduation, she moved to a different city to study at a university. She has rented a flat and has been living there with her friends up to the present. She has an income from

her part-time job, and her parents send her a little money as financial support to help her pay for rent and food every month. She has many friends with whom she loves going out for food or having fun in her free time. As travel lovers, they sometimes manage to go on low-cost trips. Emma does not have much money but still enough to pay for her bills and have a fun student life, thanks to her effort to keep track of her money.

Emma is a typical user of this web application. She often needs to split expenses in different situations like household costs, trip costs, and shared expenses occurred between friends…, so it is no surprise she would make the most out of the application. This is the type of user the application is made for.

2.3.2 Persona B

Name: Ray
Age: 18
Gender: Male
Hobby: Video games, technology, books
Typical day: Ray is a high school student who still lives with his parents who provide for him. His day starts at 7 AM when he wakes up and prepares to go to school. He stays at school until 2 PM. After school, he usually goes home to relax and spend time on his hobbies. He sometimes hangs out with his friends. They usually do activities that do not require spending much money: gaming, skating, playing basketball in a public playground. In the evening, he either plays games or prepares for school.
Short history: As a teenager, Ray is still under his parents' care and does not have to stress out over life. He is carefree but not careless. He is an introvert and prefers staying inside and enjoying his time alone, but he is not a loner as he has many friends with whom he also enjoys spending his spare time. He sometimes works part-time to earn extra money besides pocket money from his parents. He does not usually share expenses with anyone, except for short trips or planned events with friends.

As a person who does not spend money on a regular basis and is utterly carefree regarding money, Ray represents the type of infrequent user who does not necessarily need an application for group expense management but might use it sometimes to split shared expenses with friends. The application should solve his problems, too; hence the design must consider his goals and needs, but he is not the primary user for whom the application is implemented.

2.3.3 Persona C

Name: Norman
Age: 40
Gender: Male
Hobby: Spending time with family, sport, work, business
Typical day: Norman starts his day with a cup of coffee and a quick breakfast before getting to work. He runs a medium-size sports equipment business with three stores in a big city. As a director, he usually has a busy schedule meeting business partners and resolving internal problems. His schedule always ends at 5 PM if there were no planned evening events or dinner meetings. After that, he goes home and spends all his time with his family.
Short history: Norman is a busy and successful man. After high school, he went to a university but dropped out to start his own business. He is an ambitious person who takes his business very seriously, which is why he succeeds. He has a wife and two kids, with whom he spends most of his free time, and a stable career, where he earns more than enough to provide for his family, mortgage, and savings. When he meets up with friends and colleagues, he pays for

himself or sometimes even for others. Overall it is a person who does not have any financial issues and can afford to be very generous to people around him. He does not share bills.

Norman is the type of person who would never use this application. Hence, the design does not have to consider this user.

Chapter 3
Requirements

Before actual UI design and product implementation, it is important to define the application's behavior. In this chapter, all functional and non-functional requirements will be specified, based on which use cases that describe the user's behavior will be described afterward.

3.1 Functional requirements

F1 User account management Users can register by creating an account with their email, display name, and password. These data will be stored in the database, and users can use them to log into the application.

A user account also contains optional data: a photo and a bank account number. The user can modify these data.

F2 Group management Users can create a group after they log in. A group must have a name and default currency. A group's photo and description are optional. These data can be modified by a user who is logged in and is a group member. Each group has a unique ID, which is assigned by the application. Users can see and share this ID but not modify it.

F3 Group members management Users can view the list of members in a group. A member in a group may or may not have an account, meaning users can add a member profile for someone who is not registered in the app. A member profile consists of a name and optional data like a photo and bank account number. Users can modify the list of members or a member profile.

While adding a new member or modifying a member profile, the group's ID is available for users to copy and send to others for access to the group.

F4 Join a group Any user can view an existing group if they have its ID, but to join the group, they must be logged in. When a user joins a group, they can associate themselves with an already existing member profile, taking on all the debt records, or add themselves as a new member with zero debt records.

F5a Group expense management Users can create an expense record within a group. This record contains an amount paid, a description, information about the person who paid, a date when the expense occurred – the actual date by default – and optional notes about the expense. Users may adjust the splitting as they please or leave it to the application, which will split the expense equally between all the group members by default.

F5b Different types of expense splitting The application allows users to input an amount, choose the people involved from the list of group members, and then split the amount between them. There are four types of splitting:

- **Equal** All members involved are assigned with the same amount.
- **Exact amounts** The user can assign exact amounts for each participant.
- **Percentage** Similar to *Exact amounts*, but instead of amounts, the user can assign different percentages of the original amount to others.
- **Shares** Similar to the previous two. The amount is split evenly into several shares, which can be assigned differently to all the members involved.

F6 Member's balance and QR code payment Users can view who owes them money or to whom they are in debt. The application lists all the debts of each member.

F7 Settle up Any member can settle up a debt. When settling up, the application generates a QR code for payment if the member who is going to get paid has a valid bank account set.

F8 Statistics The application generates a graph showing the paying trend of all the members in a group. Users can see who among the members pays the most.

3.2 Non-functional requirements

N1 Web application using Java and React The application must be implemented as a web application using Java and React.

N2 Clear and simple, minimalist design The design must communicate well with the user. Everything must be intuitive, and the user should be able to complete tasks or find a way to complete tasks easily. The design should not confuse or give the users a feeling of clutter but please the users' eyes and give them a comfortable and pleasant experience. Any redundant elements that may distract the user's focus should not be included.

3.3 Use case scenarios

Below are scenarios in which the users interact with the application. All actions start from the "Homepage" if not stated otherwise. The users' behavior – user-flow – is also shortly described.

3.3.1 Unauthenticated user

UC1 Registration

1. User opens the web page.
2. The application displays a Welcome page with a "Sign up'' form.
3. User enters their display name, email address, password and password confirmation.
4. User clicks on the "Create account" button.
5. The application saves all the data and logs the user in. It redirects the user to a blank "My groups" page.

UC2 Sign in

1. User opens the web page.
2. The application displays a Welcome page with a "Sign up'' form.

3. User clicks on “Login” button or link under the “Sign up” form.
4. User enters their credentials.
5. User clicks on “Sign in” button.
6. The application verifies the user’s credentials.
7. If the verification succeeds, the application redirects the user to the “My groups” page. If not, an error message is displayed.

UC3 Join a group This UC is described from the point of view of a new user who obtained a link to a group.

1. User opens the link to join a group.
2. The group’s main page is displayed.
3. User clicks on the “Join” button.
4. The application displays a notification informing the user that they need to be logged in to join a group.
5. User clicks on the “Log in” or “Sign up” button.
6. The application redirects user to the “Log in” page or the “Sign up” page according to their choice.

3.3.2 Authenticated user

For a user who is logged in, the Homepage is the “My groups” page where the application lists all groups the user is a member of.

UC4 Create a group

1. User clicks on the “New group” button.
2. The application displays a modal window with a two-page form. The user is automatically added as a member.
3. User enters the group’s name, default currency and other data on the left “New group” window.
4. On the right side, there is a form to add a new member. The user enters a name of the new member. Other information (email, bank account, photo) is optional.
5. User saves the new member by clicking on the “Add” button.
6. The application closes the “New member” form card.
7. Use clicks on the “Create new group” button to save the new group.
8. The application closes the form card and redirects the user to the newly created group’s main page.

UC5 Join a group

1. User clicks on the “Find group” button.
2. The application displays a modal window asking for the group’s ID.
3. User enters the group’s link/ID.
4. User clicks on the “Find group” button.
5. The application shows the group’s main page if the link/ID was entered correctly. On this page, the user can view the group’s name and the expense records, which can give them a brief confirmation that they are at the right place.

6. User clicks on the “Join group” button.
7. The application displays a modal window asking the user to identify themselves.
8. User identifies themselves. Two situations might occur:

   The user already has a member profile The application lists all the members without an associated account in a drop-down select box. The user chooses the profile they wish to associate with and then clicks on the “Join” button.

   The user is a completely new member The user clicks on “I am not here yet”.
9. The application closes the modal window and notifies the user of the successful action.

UC6 View list of members

1. User is on a group’s main page.
2. User clicks on the “Members” button.
3. The application shows a list of members.

UC7 View a member’s balance

1. User is on the “Members” page.
2. User clicks on the corresponding icon on the right side of the member’s profile.
3. The application lists all the debts and loans between the member and others in the group.

UC8 Settle up

1. User is on a member’s balance page.
2. User chooses the loan/debt they want to settle by clicking on the “Settle up” button.
3. If the receiver has a bank account, the application displays a QR code with the payment information. Otherwise it skips to step 5.
4. User clicks on the “Done” button.
5. The application displays a pop-up notification card asking for confirmation.
6. User confirms the action by clicking on the “Settle up” button.
7. The application returns to the member’s balance page, now without the debt record that was settled up.

UC9 Add a member

1. User is on the “Members” page.
2. User clicks on the “Add member” button.
3. The application opens a form.
4. User enters a name for the new member. Other information (bank account, photo) is optional.
5. The group’s ID is available in the form for users to copy and send to others for access to the group.
6. User saves the new member by clicking on the “Add” button.
7. The application closes the form and returns to the “Members” page with a confirmation about the successful action.

UC10 Edit a member’s profile

1. User is on the “Members” page.

2. User clicks on the edit icon on the right side of a member's profile.
3. The application opens a form.
4. User edits the information.
5. User saves the changes by clicking on the "Save" button.
6. The application closes the form and returns to the "Members" page with a confirmation about the successful action.

UC11 Delete a member

1. User is on the "Members" page.
2. User clicks on the edit icon on the right side of a member's profile.
3. The application opens a form.
4. User clicks on "Delete member" CTA[1] link at the end of the form (above "Save" buttons).
5. The application displays an alert dialog.
6. User confirms the action by clicking on the "Delete" button.
7. The application closes the form and returns to the "Members" page with a confirmation about the successful action.

UC12 Add an expense

1. User clicks on the "Add expense" button.
2. The application opens a two-page form. On the left page, there is the expense information and on the right page is the splitting information (splitting type, participants, amounts…).
3. User enters a description and an amount and chooses the payer. The date may be modified. The user may as well add some notes to it.
4. The default splitting type is Equal. If the user wishes to split the expense equally between all the group members, they can ignore the right page.
5. User adds the expense by clicking on the "Add" button.
6. The application closes the form and returns to the group's page with a confirmation about the successful action.

Alternative flow – different splitting type:

5. User chooses a splitting type on the right-side.
6. User can choose the participants by unselecting the members who are not involved.
7. User enters splitting values (percentage, exact amounts or shares).
8. User adds the expense by clicking on "Add" button.
9. The application closes the form and returns to the group's page with a confirmation about the successful action and the expense record added.

UC13 Edit an expense

1. User clicks on the edit icon on the right-side of an expense's record.
2. From here, the user-flow is similar to UC12 Add an expense starting from step 2.

UC14 Delete an expense

1. User clicks on the edit icon on the right-side of an expense's record.

[1] Call To Action

2. The application opens a two-page form. On the left page, there are data of an expense and on the right page is the splitting information (splitting type, participants, amounts…).
3. User clicks on “Delete expense” CTA link at the end of the left page (above “Save” buttons).
4. The application displays an alert dialog.
5. User confirms the action by clicking on the “Delete” button.
6. The application closes the form and returns to the group’s main page with a confirmation about the successful action.

UC15 Log out

1. User navigates to the top-left corner to the User icon.
2. A drop-down menu opens with two options: “Edit account” and “Log out”
3. User clicks on “Sign out” option.
4. The application redirects the user back to Homepage.

Chapter 4
UI design process

4.1 Visual elements

4.1.1 Colors

All of the applications reviewed within the research tend to have similar color choices. Most of the reviewed applications use a white/solid background to highlight the components. As an attempt to create a new and unique application (which also means more enjoyable, fresh, establishing a better user experience), a different approach was chosen. White and purplish-blue (#6201FF - Electric Indigo) colors were used as two primary colors instead of a single colored background.

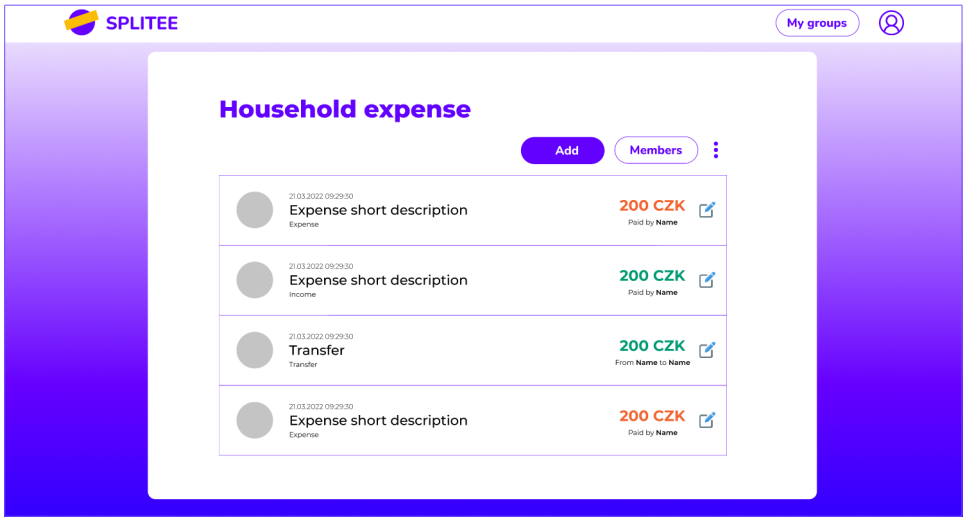

Figure 4.1 Prototype screenshot – group view – created with Figma[8]

White is a neutral color that possesses the ability to make everything pop on it; therefore, it was used as a background for the main scene. A purplish-blue rim was added to underline the white background to prevent the boring layout. This provides more visibility to the main area and narrows the user's attention to the white middle part, where all the action will be performed.

According to color psychology, purple represents wealth and royalty, and blue is associated with trust, stability, and peace.[9] Both colors, being cold colors, induce feelings of comfort and trust within the human mind. The application intends to make users feel calm and secure since

resolving debts can be quite stressful, so a purplish-blue color scheme was established as a suitable choice.

A tiny portion of yellow, green, and orange was used to add some accent to the design. In contrast to purplish-blue, yellow gives the impression of optimism and friendliness. Green and orange are used to emphasize the positive and negative actions and information visually. These accent colors not only make the design more lively and enjoyable, but also add to the functionality.

4.1.2 Typography

The typeface[1] chosen for this project is *Montserrat*, a sans-serif[2] font family found and taken from Google Fonts[11]. In general, sans-serif fonts are easier to read on a screen, whereas serif fonts are easier to read on printing mediums[10].

Using a single typeface gives the layout a simple and unified look. Different fonts (bold, regular, light…) are used to define a page's hierarchy and emphasize important information.

4.1.3 Layout and components

A design might be confusing and give insecure feelings if the layout changes too much or is entirely different from page to page. While making the frames, the priority was to keep the elements as visually similar as possible.

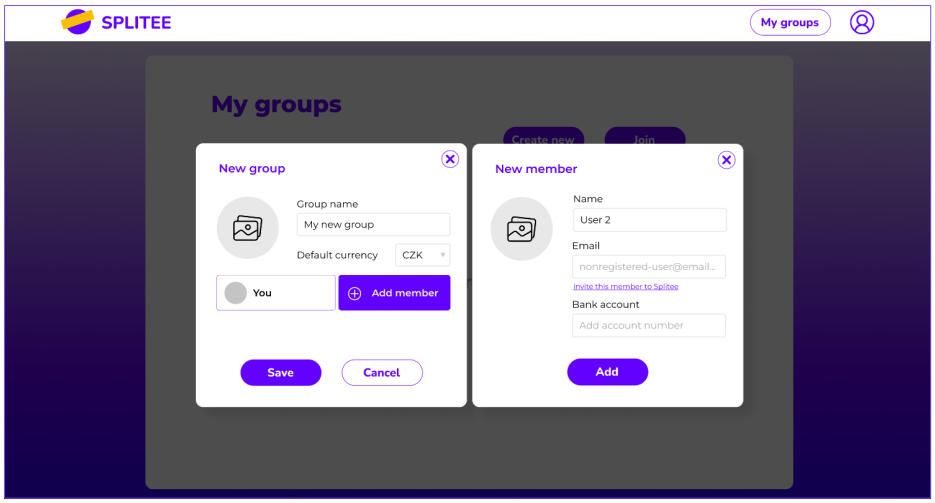

Figure 4.2 Modal for create group action – created with Figma

When a create/edit action is called, a modal shows up with the background slightly hidden just enough for the user to see where they are. This simultaneously helps the modal stand out to draw the user's focus.

4.2 Prototype

A prototype is a presentation of the final product. It is an interactive sample of the application, where others can see how the final application would look in the end. Above that, it also helps

[1] A collection of fonts with related lettering design, for exp. *Montserrat* is a typeface, and *Montserrat Bold* is a font[10]

[2] A font without a tiny lines at the end of characters[10]

for better orientation while implementing the application.

The prototype was created using Figma, a well-known design tool used by many designers. For the first step, all the important pages were listed: Homepage, login/register page, a list of groups, a list of members, and transactions within a group. Wire-frames were then drawn using paper and pen, and finally, interactive frames were created in Figma based on them.

4.3 Testing

Six testers were selected among the survey's correspondents according to the Personas to test the prototype. Three match Persona A's profile, and the others match Persona B's profile. Persona C is the type of user who would not use this application, so there is no reason to get feedback from people of this type.

Along with the scenarios, these questions were asked:

1. How does the design make you feel? Is it simple and intuitive?
2. Are you able to easily complete the task? Is it easy to find everything you need?
3. Is there anything confusing? Do the elements (button, links…) do what you expect?
4. Do you understand what function each element has? (is it an input field? Is it clickable?…)
5. Is there anything you struggled with and what is that?
6. Any other suggestions?

This test gives an insight into how users view the app and how the design affects them. It points out what is wrong and what can be improved so that the final design could make it as convenient for users as possible.

Below are short descriptions of the testers and the tests' results.

4.3.1 Group of testers A

Persona A matching testers fall into the age range of 22–25. They live almost independently (with irregular financial support from their parents) in a shared household where the common costs like rent, cleaning tools, food…are regularly split. They also split money spent for entertainment when going out with friends. Two of them already use a digital group expense manager (*Splitwise*), and the last one uses paper and pen.

4.3.2 Group of testers B

Testers of this category are younger students between 19–23 years old. All of them live with their parents or relatives and are mostly provided by them. They have no financial problems and spend just as much money as they earn from part-time jobs. Splitting money is an *once in a while* occasion for them. They mostly use paper and pen; one of them actually uses a splitting feature available in a digital wallet, and they have never used a group expense manager application before. They rarely need to split expenses because most of the time, they just pay for themselves.

4.3.3 Test results

The testers stated that the design is pretty, simple, and intuitive. All the elements communicate well. The only problem that occurred was caused by the fact that the prototype is not fully interactive as a final programmed application would be. The frames change according to the

scenarios, so sometimes, it was not very clear when they could not click on something. This problem was fixed by individual calls with testers, who could not complete the test because of the confusion.

4.3.4 Conclusion

Based on the feedback, a few changes were noted to fix the design on testers' suggestions that would make the design less confusing, hence increasing the application's usability.

Chapter 5

Application development

The application uses a three-tier architecture, which consists of a presentation tier, logic tier, and data tier (database). In this text, the presentation tier will be referred to as *front end (FE)* and the other two tiers *back end (BE)*.

The front end is the visual part of a web application which users interact with directly. It includes everything users see and experience: text, images, colors, shapes…. During front end development, the prototype made earlier came in handy to create the designed layouts. For this project, ReactJS[12] is used along with Chakra UI[13] – a component library – to achieve the required results.

The back end, on the other hand, is the part that runs in the background and makes sure everything on the front end works fine. It deals with things users do not have direct access to, including application logic, API[1], data storage, and manipulation…. As for required technologies for this project, Spring Boot[14], one of the most popular Java frameworks, was chosen to create the logic tier for its simplicity. The Spring Boot application is connected to a PostgreSQL[15] database, where all the data are stored.

In this chapter, the whole creation process will be covered, starting with the back end and an introduction to the database model, followed by the logic tier development and then the front end development.

5.1 Back end

The development starts with the BE as the foundation for the FE to be built on later. This sub-chapter consists of the database description, configuration, and the process of building the services using Spring Boot.

5.1.1 Database model

This section will describe each entity in the database with all the relations between them. There are several critical points to pay attention to while designing the database model (Figure 5.1):

- A group ”member” does not need an account and can be associated with an account later.
- A group must be able to list all the debt situations between its members.
- An expense can be split between a subset of a group (not all the members are participated)

[1] Application Programming Interface

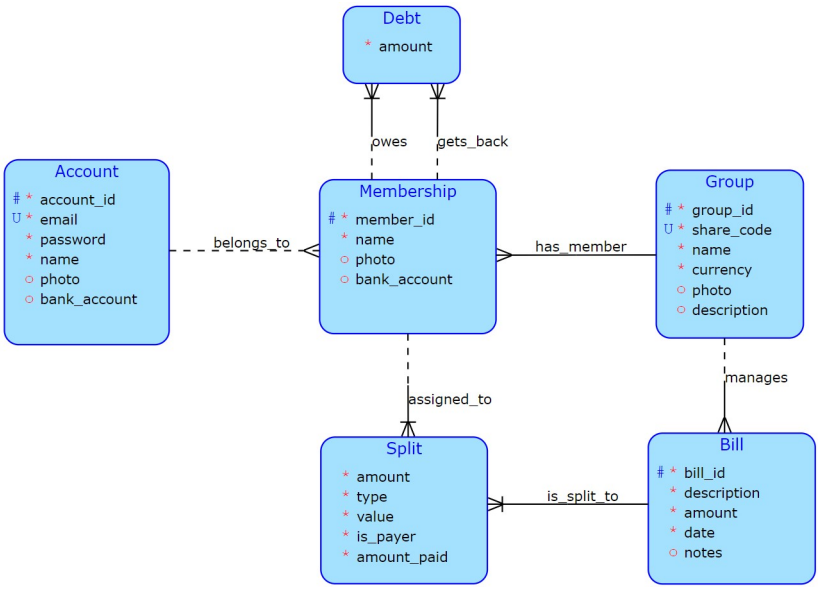

Figure 5.1 Conceptual schema – created on DBS portal[16]

Group represents a group, within which are expenses split

- **group_id** unique identifier of the group. A sequence generator is used to generate and provide the uniqueness of identifiers
- **group_code** a 7 characters code available for users to share
- **name** a short name of the group
- **photo** an URL[2] to the group's photo
- **description** a description to give users more insight about the group

Membership represents each member profile in a group

- **member_id** unique identifier of a membership. A sequence generator is used to generate and provide the uniqueness of identifiers
- **account_id** this attribute contains an account's unique identifier if the profile is associated with a registered account. Otherwise it is `NULL`
- **group_id** ID of the member's group
- **name** a name of the member profile.
- **photo** an URL to the member's photo for better identification
- **bank_account** a bank account to which the debts can be sent to while settling up

Account represents a user account

- **account_id** unique identifier of an account. A sequence generator is used to generate and provide the uniqueness of identifiers
- **email** an email address the account is registered with
- **password** a hashed password
- **name** a name for display

[2]Uniform Resource Locator

photo an URL to the account owner's photo

bank_account a bank account to which the debts can be sent to while settling up

Bill represents an expense to be split within a group

bill_id unique identifier of a bill. A sequence generator is used to generate and provide the uniqueness of identifiers

group_id ID of the group, within which the bill is created/managed

description a short description of what the bill is for (for example: lunch, shopping, trip…)

amount an amount paid for the bill

date a date when the bill was paid

notes other notes toward the bill

Split represents the parts a bill is split into

bill_id ID of a bill the split comes from. This makes a part of the entity's primary key.

member_id ID of a member the split is intended. This makes a part of the entity's primary key.

amount the final amount a member should pay for the bill's participation

type the split's type: *equal, exact amount, percentage* or *shares*

value this attribute is the same as amount if the chosen type is equal or exact. In case of percentage or share, it contains the value of the percentage or the share. For example, if a person should pay 50 % of the bill, the value should be "50". In other case, the person might had 2 meals from total 3, the value should be 2 etc.

is_payer a boolean attribute that informs whether the member is the payer

amount_paid if the member paid for a bill, the amount paid should be stored here

Debt represents a debt record between two group members

member_id_owes ID of a member. This makes a part of the debt's primary key.

member_id_gets_back ID of another member. This makes a part of the debt's primary key.

amount the debt's amount

5.1.1.1 Database tables

Database tables based on the model above were created with a create script in pgAdmin 4[17]. The create script was generated by the DBS portal from the conceptual model. Some minor renaming occurred due to the keyword matching table name. The data types for attributes were also modified to suit the requirements.

The database was then connected to the Spring Boot app via IntelliJ IDEA[18] – an IDE used to develop the application.

5.1.2 Logic tier

The back end is built with Maven[19], a tool for easier application building. Maven works based on Project Object Model (POM). This model describes the project, including dependencies on external libraries using an XML structure.[20] Following is the project's directory structure. POM can be found in the root of the project as `pom.xml`.

```
src
├── main
│   └── java
│       ├── controller ..... API controllers
│       │   └── dto ..... DTO classes – transferring objects
│       ├── entity ..... entity classes representing tables in the DB
│       ├── repository ..... repository interfaces which communicate with the DB
│       ├── security ..... security configuration
│       └── service ..... service classes where the logic operations are implemented
│   ...
├── pom.xml ..... Project Object Model – a model used to describe the whole project
└── ...
```

5.1.2.1 Spring Boot

The logic layer is implemented with Java using Spring Boot framework. Spring Boot is an extension of the Spring framework – an open-source Java-based framework. Spring supports Dependency Injection and many other modules like Spring Data for database manipulation or Spring Security, Spring Test…, makes it easy to create Java applications.[21]

The main feature of Spring Boot, which gives it an advantage over Spring, is *Autoconfiguration*. Spring Boot automatically configures a Spring application based on the dependencies, which can speed up and simplify development since no manual configurations are needed.

A basic project structure with Maven build specification was generated by the Spring Initializr[22].

5.1.2.2 Spring JPA & ORM

JPA is an acronym for Java Persistence API (from 2019 Jakarta Persistence API). It is a Java standard that allows object-relational mapping (ORM). ORM is a programming technique used for automatic conversion of data between a relational database and an object-oriented program (OOP) language. The essential element of JPA is an `entity` or an entity class, which is a class representing a table in the relational database. Each instance of this class corresponds to a row in the table.[23]

Spring JPA is a module which allows easy implementation of JPA based repositories. Each entity class has a `repository` interface that acts as a data access layer, which is annotated with `@Repository` annotation. This interface encapsulates operations and transactions with the database. By extending from Spring's `JpaRepository`, the most important methods such as `save()`, `findBy()`, and `deleteBy()` are already available to use, which reduces the boilerplate codes for developers. Custom queries can be added using the `@Query` annotation. Below is an example:

```
@Query(value = "SELECT g.memberships from group g where g.code = :code")
Collection<Membership> findAllMembersByCode(String code);
```

Code listing 1 An example of a custom query

The query listed above is written with HQL[3] – an object-oriented version of SQL[4].

[3] Hibernate Query Language
[4] Structured Query Language

5.1.2.3 API

The communication between the FE and BE is provided using HTTP – Hypertext Transfer Protocol. In the BE, there are controller classes that implement HTTP methods like `GET`, `POST`, `PUT`, `PATCH`, `DELETE`.

GET is used to retrieve the resource.

POST is to create new resource.

PUT is used to modify the resource.

PATCH is used to modify the resource. The difference between `PUT` and `PATCH` is that `PUT` updates the entire resource if it exists or creates new if it does not exist. With `PATCH`, on the other hand, it is possible to modify the resource partially.

DELETE is used to delete the resource.

Each class dealing with an entity in the database has its URI. The class's methods are appropriately annotated. They return `ResponseEntity`, objects representing the whole HTTP response.

```
@RequestMapping(value = "/api/groups")
public class GroupController {
...
    @PostMapping
    public ResponseEntity create(@RequestBody GroupDTO dto) {
    ...
    }

    @GetMapping("/{code}")
    public ResponseEntity findByCode (@PathVariable String code) {
    ...
    }
...
}
```

Code listing 2 Annotated controller class with URI mapping

Requests may or may not contain a body. DTO classes are used to represent the resource contained in the request body and will be described in the following section.

5.1.2.4 DTO - Data Transfer Objects

DTO are objects that *"carry data between processes in order to reduce the number of method calls"*.[24] They are also used to decouple the domain models from the front end. Basically, the objects that are transferred between the front end and API controllers are DTO. In the back end, there are methods to convert DTO to entity objects, which are later persisted to the database. Entity objects can also be converted to DTO to be sent to the front end.

5.1.2.5 Debt calculation

In order to keep track of debts between every pair of members in the group, there are two approaches to consider.

Let A be the set of a group's members and x, y two members of the set.

Cartesian square of A – this results in a table of $|A|^2$ cells. Each cell with coordinate $[x, y]$ contains a positive number of the amount x owes y. The whole debt situation between two members x and y can be represented by two cells $[x, y]$ and $[y, x]$, with at most one of them containing a number bigger than 0 at a time. When x is in debt to y, the amount would be saved in cell $[x, y]$; otherwise, when y is in debt to x, the amount would be saved in cell $[y, x]$.

Whenever a new debt occurs, it would require finding both cells and recalculating the debt. This can be done in four primitive steps:

1. Find the correct cell and add the new debt to it
2. If both cells are not 0, determine which cell has the bigger value
3. Recalculate the debt of the cell with the bigger value by subtracting the smaller amount from the bigger one
4. Nullify the other cell.

This approach is simple to implement but requires a lot of space. Imagine having a huge group of hundreds, the table would have the size of tens to thousands of thousands of cells. And that is a lot.

A set of unique pairs – in this set, $[x, y]$ and $[y, x]$ are treated the same as the "debt situation between x and y". However, in the database, only one of them is saved as a key, which can contain a negative value.

Let $[x, y]$ be the key saved in the database. When the value of this key is positive, that would mean x owes y money; otherwise, if the value is negative, it results in y owes x.

This approach consumes half the memory required from the first approach. It comes with a price, which is a more complex algorithm for finding or updating a debt. The program has to compare how the pair is saved in the database and how the debt data came in, determine sides (who owes whom), and choose the operation (addition or subtraction) to be performed with the original value based on who is in debt and who gets the money back. It is also more complicated to display these data in the UI.

The price is, however, not too high considering how much space would have been saved. Therefore, this approach was chosen to implement the debts.

5.1.3 Authentication & authorization

Authentication is the process when a system verifies a user's identity. This process requires secret information or information that is difficult to access. Usually, encoded passwords are used for this purpose.

When a user signs up, they must provide a unique email and a password. The application verifies the email's uniqueness, encodes the password, and then saves this information into the database. The password is encoded using `BCrypt`.

To log into the application, the user uses the email and password used during registration. The application verifies these credentials and returns a JWT (JSON[5] Web Token) in case of a successful login. This token is saved into the user's local storage, and every time a request is made, it must contain this token in an "Authorization" header to access protected resources. The application checks the JWT, gets the user's info and authorizes it.

Authorization is a process when a system verifies a user's authority and permission to access the resources. Usually, authorization is provided based on the user's role, which might be `USER`, `ADMIN`, or something else. In this application, authorization is based on the groups a user is a part of.

[5] JavaScript Object Notation

An unauthenticated user can see a group's main page containing a list of expenses. This page is public for users with a shared link or ID to see how the application looks beyond the sign up and login pages. Users who want to join the group and access other features will be requested to log in.

An authenticated user has full access to all the groups they are a part of. When accessing a group that they are not a member of, the same rights as an unauthenticated user are applied.

Authorization is also *partially* solved by routing in the FE. The routing protects some specified routes from a group of users, basically unauthenticated and authenticated. When a user accesses a page that is not intended for them, the application redirects them to the "Homepage", which is "Sign up" page for unauthenticated and "My groups" for authenticated. Routing will be discussed in section 5.2.3.

5.2 Front end

Front end development consists of creating the UI, working with the visual elements, and communicating with the BE. As for a dynamic web app that this project aims for, there is a lot of work with Javascript (JS) to get, manipulate and display the correct data. The JS packages are installed and managed using `npm`[25].

5.2.1 ReactJS

ReactJS or React is an open-source JavaScript library for building user interfaces developed and maintained by Facebook (now Meta Inc.)[12]. It is declarative and component-based. The application comprises components that independently manage their `states` and communicate with each other via `props`.

JSX or JavaScript Syntax Extension, as the name suggests, is an XML[6]-like extension to the JS language syntax. React components are usually written using JSX. It is a way of combining markup and display logic in one unit called "component". JSX takes an *HTML-looking* expression and turns it into a function call, making it possible to use the full power of JS code that looks just like markup and is easier to follow.

```
const listGroups = groups.map( group => {
    return (
        <Box key={group.code} as='button' layerStyle='item'
        onClick={() => navigate(`/group/${group.code}`)}>
            <Avatar size='lg'
            name={group.name}
            src={group.photo}
            />{' '}
            <Heading variant='cardName'>{group.name}</Heading>
        </Box>
    )
})
```

Code listing 3 An example JSX code

[6]Extensible Markup Languag

5.2.2 Chakra UI

Many options were available for choosing the approach to UI implementation. Using a component library was chosen to avoid building the UI from scratch and save time on writing excessive pure CSS. After researching existing React component libraries, Chakra UI was chosen to serve the purpose. A fairly nice, claimed-to-be unbiased comparison can be found on Chakra UI's pages[26].

Despite being relatively new, with version 1.0.0 released in 2020, Chakra UI has received many positive reviews and rapidly gained popularity. It offers a beautiful default theme with many out-of-the-box components, which they call "building blocks" needed to build a React application.[13] Prop-based approach makes it easy to override the components' styles simply by passing arguments as props. On top of that, high customizability and extensibility are crucial for building applications with custom designs. It is also very easy to learn and use, making it a suitable choice for this project.

However, the application's UI was not built solely with Chakra UI, as it still has shortcomings and missing components, given it is a relatively young library. An addition of elements from Ant Design, icons in particular[27], were used to complement the limited choices of icons Chakra UI offers. It also does not offer a date picker component at the time of this thesis' writing, and for that purpose `react-datepicker`[28] was used.

5.2.3 Routing

React Router[29] was used to resolve routing. The configuration can be found in `index.js`. Redirection is also described and configured in this file. The application redirects all unauthenticated users to "Sign up" page if they try to access protected routes. The "Sign up" and "Login pages", on the other hand, are hidden from authenticated users. When trying to access this page, the user will be redirected to "My groups" page.

5.2.4 Forms

Forms are essential for every web application that require user interaction, especially when user inputs determine how the application behaves. React Hook Form[30], a popular form library for React, is used to simplify work with forms. Form inputs are wrapped in `<form>` tag and can be validated with the use of props like `required`, `minLength`, `pattern`, and more.

```
<form onSubmit={handleSubmit(signup)}>
...
    <Input id='email' type='email' placeholder='your@email.com'
           {...register('email', {
               required: 'This is required',
               pattern: {
                   value: /^[A-Z0-9._%+-]+@[A-Z0-9.-]+\.[A-Z]{2,}$/i,
                   message: "Invalid email address"
               }
           })} />
...
</form>
```

Code listing 4 An example of an input field with simple email validation

5.2.5 Fetch API

To communicate with the BE, Fetch API is used. The Fetch API is built into most browsers so there is no installation required. It provides an interface to access and manipulate requests and responses, allows for asynchronous request of a resource using `fetch()` method. This method has two parameters – a URL and the data consists of properties like `method`, `headers`, stringified `body`…. It returns a promise of `Response` object.

A promise represents the eventual completion of an asynchronous operation. There are three states, in which a promise can be: *pending*, *fulfilled*, and *rejected*. *Pending* is the initial state, where the operation is not yet completed. When the operation completes successfully, the state changes to *fulfilled* with a value that can be then converted to JSON with `json()` method. In case of a *rejected* promise, it is possible to use `catch()` to get the error which caused the operation's failure.

```
fetch(url)
    .then((response) => {
        if(!response.ok)
            throw Error('Something went wrong')
        return response.json()
    })
    .then(data => {
        setData(data)
    })
    .catch(error => {
    })
```

Code listing 5 An example of using `fetch()` function

5.2.6 Image upload

For better distinction of groups and members and to make the layout less monotonous, users can add a photo to a group and other members, as well as to their own account. Cloud storage is suitable for this purpose. Images from users can be uploaded directly to the cloud, and only a link to the image is saved in the database and displayed on a page just by a simple `img` tag with `src` attribute.

Cloudinary is a cloud platform focusing on image and video management.[31] When a user uploads an image in the application, it is uploaded directly to the cloud by a `POST` request, which returns a URL of the image. Cloudinary also provides support for image transformation. Resizing feature was used to reduce the size of the file to prevent using up the limit provided for the free tier.

5.2.7 Expense split

The core of this application is made of the splitting functionality. Users can split an expense in four different ways: equally, by exact amount, by percentage, or by shares. The bill's amount can be split among a subset of the group's members, which means not every member has to be involved, which also applies to the payer. In order to achieve a dynamic display for users to see the splitting process precisely, the mechanism was implemented on the FE. The final results are then sent to BE to persist to the database.

5.2.7.1 Equal split

This is the default splitting approach, where the original amount is equally divided among the participants. The application checks for the number of participants (not every member in a group has to participate in an expense) and then simply divides the amount by this number and assigns the final amount to each member.

5.2.7.2 Exact split

This type comes in handy when users want to assign a particular amount to each participant. The assignment can be accomplished by filling the exact numbers into the input fields. The values in these fields have to add up to the amount of the bill.

5.2.7.3 Split by percentage

Similar to exact splitting, users can assign the percentage of the bill's amount to each participant. The application will calculate and display the amount based on the user's input. The percentages must add up to 100 %.

5.2.7.4 Split by shares

Splitting by shares is the most challenging split type since the total number of shares is variable, leading to a constant recalculation of the resulting amount for each participant. To implement this, a total amount of shares must be stored and updated every time a user inputs a new value.

The equation for resulting amount calculation is: $billAmount * (input/allShares)$

5.2.8 QR code generator

QR code is a two-dimensional type of bar code that is used to store information in an encoded format. In contrast with traditional bar codes, which can be scanned from paper using a laser scanner, QR codes can be conveniently read by any device with a camera and compatible software from both paper and screens, which explains its popularity in recent years.[32] QR codes can be used for many purposes, one of which is for easy payment, when information like the amount of transaction and its recipient is encoded into the black and white square pattern.

The application should generate a QR payment code to make settling debts easy. During the research phase, it became clear that this problem can hardly be solved universally since each country's banking/payment system has its own standards and not every banking app supports QR payment. Although such an idea is attractive, solving this problem is ways out of the scope of this thesis. The final decision is to implement QR codes for domestic payments within the Czech Republic only. While settling debts, the application displays a QR code if the recipient's member profile contains a valid Czech bank account. By scanning the code in a banking app (almost all banking apps in Czech support QR payment[33]), a filled payment order will be generated, so the user does not have to type all the details by themselves. They simply complete it by payment confirmation.

This feature was implemented by use of QR Platba, a standard *"...created with a simple vision: To simplify payment from these devices*[7] *and make life easier for people by not having to rewrite payment details manually."*[33] A QR code is created by an API call in the image's `src` attribute.

[7] Referring to mobile devices

5.2.9 Statistic generator

The application is able to generate a simple doughnut chart featuring members who pays the most in a group. ChartJS[34] was used for this purpose along with `react-chartjs-2`[35]. ChartJS is a popular JS library for creating charts, which offers many built-in chart types and `react-chartjs-2` acts as a wrapper, offering these charts in components to make it even easier to create charts in React.

To provide data necessary for visualization, an addition method was implemented in the BE's controller class to generate an array of data sorted by the total amount spent by each member in a group. Displaying the chart is as simple as including a component to the page.

Chapter 6

Deployment and usability testing

6.1 Deployment

The last part of this project, before the actual testing with users, is to make the application available for use. The BE was deployed during the early phase for easier FE development. There was a problem with CORS[1] after the BE was deployed and used directly while developing FE. CORS is a mechanism that supports secure cross-origin communication. It allows a server (BE) to give permission to load resources to any origin (domain, scheme, or port). This problem was solved by adding `@CrossOrigin` annotation to API services in the BE.

```
@CrossOrigin
@RestController
@RequestMapping(value = "/api/groups")
public class GroupController {
...
```

Code listing 6 Use of the annotation that enables cross-origin requests

6.1.1 Heroku

Heroku is a cloud platform that enables developers to build, run and operate applications in the cloud.[36] It was chosen from personal experience with the service in the past. Heroku supports deployment using Heroku Git, Container Registry (both methods using Heroku CLI[2] or GitHub[37]. GitHub was chosen to serve the purpose of easier manipulation solely via Heroku's UI.

A duplicate git repository of the app's BE was added to GitHub because the main source code is versioned initially using the school's GitLab[38]. A PostgreSQL database was set up and connected using Heroku CLI. The tables were created with a `create script` during this process.

Despite facing some troubles with configuration during initial deployment, the BE was successfully deployed and is now running smoothly with new commits automatically deployed. A bigger problem arose during FE deployment. Although the build and deployment processes succeeded, the web app constantly crashes due to troubles with memory. The problem was caused by the memory limitation of Heroku's free tier, leading to the need to find an alternative solution

[1] Cross-Origin Resource Sharing
[2] Command Line Interface

to deploy the FE. That chosen solution was Firebase.

6.1.2 Firebase

Firebase was founded in 2011 as a chat API. It was acquired by Google in 2014 and is now a very well-known app development platform offering many BE services on top of Google Cloud[39]. For this project, Firebase Hosting is used to host the FE application.

The deployment went without any problems. The process is packed into just a few steps, such as installing Firebase tools with `npm install`, initiating and connecting to a project created in Firebase's UI, and deploying by running `firebase deploy` in the command line. This results in a smoothly operating web application which can be found here: `https://splitee-893f5.web.app/`.

6.2 Usability testing

The same testers from section 4.3 were contacted again with a request to test out the application, which has been already deployed and running on the above address. For reminder: three testers are of type Persona A, who often need to split shared expenses, and the other three are of type Persona B, who do splitting rarely or occasionally. Some of them have used or are still using an app for group expense management.

A scenario that consists of all operations a user could interact with in the application was sent to the testers. Along with those, three questions were asked:

1. How does using the app make you feel? (it is intuitive, I can easily complete the tasks…)?
2. Is there anything you find confusing or struggled with? What was that?
3. Any other suggestions?

Surprisingly, most of the testers stated that finding the balance / settling up page was a little confusing, despite no complaints about this problem being mentioned during the prototype testing. Some bugs were also found during testing. These bugs, including bugs where a newly joined member could not leave the group, or there were no debts changed after a new expense was added…, were immediately removed. There were also some suggestions on improving the app, which will be discussed in the last chapter.

Other than that, there were no problems. All testers agreed on the friendly app usability, the pleasant design, being neither distracting nor confusing, and the properly working functions.

Chapter 7

Future improvements

Like any other product, there is always room for improvement, especially for a completely new application, where the development is only at the very start. Despite being usable, there is still a lot to add to it.

7.1 Better user experience

Because UX is all about small details that make interaction with the application more enjoyable and practical, it is worth to keep searching for those small details and implementing them.

Responsiveness The application's responsiveness is another point to be improved. As mentioned in chapter 1, the expansion of mobile phone users is undeniable. To increase user engagement, it is inevitable for a web application to be usable even on mobile phones. Despite not being the goal of this thesis, the application was intended to be responsive for usage on different devices. However, there were many more significant problems needed to be solved throughout the development, so the app's responsiveness remains a subject for future development.

Minor layout changes Some minor layout changes can have a significant impact on UX. One of those might be the order of the members in the list. The user's member profile could be placed on top of the member list. On the split form, it is worth reconsidering changes for a better display of larger amounts. Moving the group code copy button to the group's main page might be another good idea.

Major layout changes As suggested by the testers, the UI can be changed so that the Debts page might be easier to find. It appears users would expect to find this feature on the group's main page or somewhere on the expense list.

Other improvements Other improvements like a stronger password validation with different types of characters or old password confirmation when changing the password would be necessary if the application was about to be released for public use. Email verification after registration would also be helpful for better security.

Splitting process The splitting process can be improved by displaying the remaining amount after users enter a splitting value (be it an amount, percentage, or shares). A better idea would be to equally split the remaining between the other participants who do not yet have an assigned amount. For example, when a user assigns, let it be, 20 % out of the bill's amount to a participant, the remaining 80 % might be automatically split equally for the

others. Assigning the amount for everyone without the user's need to calculate it themselves would immensely increase the user experience.

7.2 App management

The current application is hosted separately on Heroku (BE) and Firebase (FE) due to the troubles with deployment described in the previous chapter. For better app management, it might be a good idea to migrate the BE to Firebase so that both parts of the application can be easily managed on one platform. Firebase also offers cloud storage, so moving image storage from Cloudinary to Firebase would also be an option.

7.3 Extra features

There are many more features a group expense management can offer that their implementation would exceed the scope of this thesis. Below are some features people find helpful according to the survey conducted at the beginning of the work:

Bill itemization Users can separate the items on one bill so that everyone can see exactly who is paying for which items

Multi-payers More people can pay for one expense/bill

Currency conversion Helpful for the situation where users pay in a different currency

Repeatable expenses Rent or other periodic expenses

Payment reminder For people who often forget to pay or people who are too shy to ask for money back

The more ambitious idea is to combine this group expense manager with a personal expense manager. There are many applications designed for splitting expenses and even more focusing on personal financial management. However, there is no combination of those two[1]. Managing personal financial situation and debts from or toward others in one single application would be very convenient.

[1] Or at least there is none that the author has heard of.

Conclusion

The main purpose of this bachelor's thesis was to design and create a web application for group expense management. As a result, such an application was created, deployed, and tested.

During the process of creation, many UI/UX design principles were applied. Aiming for a user-centered design, a UX case study was provided, which went from building empathy with the users by conducting user research, identifying target users, creating a prototype, and testing with real users. The application was then realized according to the research and design, using a three-tier architecture: presentation, logic, and data tier. Java's framework Spring Boot was used for the back end, React JS for the front end, and PostgreSQL for the database. An API was implemented for the presentation and the logic tier communication, and JPA was used to connect the logic tier to the database.

The application allows users to create and manage different groups, within which the expenses can be split equally, by exact amounts, by percentage, or by shares. Multiple people may manage a group as long as they are logged in and connected to the group. At the start of the project, a QR code generator for general bank transfers was intended to be added, which would help users pay more quickly. During the process, it became clear that QR payment, or contactless payment in general, is not a simple problem to be solved solely in this bachelor's thesis. As a result, the in-app QR payment code generator is implemented to work for Czech bank accounts. However, there are many other solutions for easy payment for users from other countries, so the application might be improved by considering those solutions.

The application's development is only at the beginning, and in the meantime, it offers just a couple of core features a group expense manager should be able to execute. That leaves a lot of space for improvement as so many interesting features can be added as discussed in the last chapter.

..

Bibliography

1. STEP UP LABS, Inc. *Settle Up* [online] [visited on 2022-03-15]. Available from: `https://settleup.io/`.
2. SPLITWISE, Inc. *Splitwise* [online] [visited on 2022-03-15]. Available from: `https://www.splitwise.com/`.
3. DAVEAU, Takeshi; MARIE, Johan; KOCKEN, Stan. *Sesterce* [online] [visited on 2022-03-15]. Available from: `https://sesterce.io/`.
4. JOBE, William. Native Apps vs. Mobile Web Apps. *International Journal of Interactive Mobile Technologies*. 2013, vol. 7, no. 4.
5. MCKAY, Everett N. *UI is communication: how to design intuitive, user centered interfaces by focusing on effective communication*. Elsevier, Morgan Kaufman, 2013. ISBN 978-0-12-396980-4.
6. GOEL, Gourisha; TANWAR, Poonam; SHARMA, Shweta. UI-UX Design Using User Centred Design (UCD) Method. 2022, pp. 1–8. Available from DOI: `10.1109/ICCCI54379.2022.9740997`.
7. PAVLÍČEK, Josef; COLLECTIVE. *The Cookbook for Interaction Design and Human Computer Interaction* [online]. Available also from: `https://docs.google.com/presentation/d/1nbLjgEX5mS6kl_cRx6CeKuhd-fzz-kyYn_j03vMLkH4/edit?usp=sharing`.
8. FIGMA, Inc. *Figma* [online] [visited on 2022-04-03]. Available from: `https://www.figma.com/`.
9. CHIJIIWA, Hideaki. *Color harmony: a guide to creative color combinations*. Rockport Pub, 1987.
10. OSBORN, Tracy; KEITH, Jeremy. *Hello web design: design fundamentals and shortcuts for non-designers*. San Francisco: No Starch Press, 2021. ISBN 978-1-7185-0138-6.
11. GOOGLE INC. *GoogleFont* [online] [visited on 2022-05-02]. Available from: `https://fonts.google.com/`.
12. META PLATFORMS INC. *ReactJS* [online]. 2013 [visited on 2022-05-25]. Available from: `https://reactjs.org/`.
13. ADEBAYO, Segun. *Data Transfer Object* [online]. 2020 [visited on 2022-06-20]. Available from: `https://chakra-ui.com/`.
14. VMWARE, INC. *Spring Boot* [online] [visited on 2022-05-02]. Available from: `https://spring.io/projects/spring-boot`.
15. *PostgreSQL* [online] [visited on 2022-05-25]. Available from: `https://www.postgresql.org/`.

16. FACULTY OF INFORMATION TECHNOLOGY, Czech Technical University in Prague. *DBS portal* [online] [visited on 2022-05-02]. Available from: https://dbs.fit.cvut.cz/.
17. *pgAdmin 4* [online] [visited on 2022-05-25]. Available from: https://www.pgadmin.org/.
18. JETBRAINS S.R.O. *IntelliJ IDEA* [online]. 2001 [visited on 2022-05-25]. Available from: https://www.jetbrains.com/idea/.
19. FOUNDATION, The Apache Software. *Apache Maven* [online]. 2019 [visited on 2022-06-08]. Available from: https://maven.apache.org/.
20. GLAZAR, Filip. *Maven and Spring framework* [lecture]. Prague: Faculty of Information Technology, Czech Technical University in Prague, 2019.
21. VMWARE, INC. *Spring* [online] [visited on 2022-06-08]. Available from: https://spring.io/.
22. VMWARE, INC. *Spring Initializr* [online] [visited on 2022-06-08]. Available from: https://start.spring.io/.
23. PAVLÍČEK, Josef. *ORM - Objektově relační mapování* [lecture]. Prague: Faculty of Information Technology, Czech Technical University in Prague, 2019.
24. FOWLER, Martin. *Data Transfer Object* [online] [visited on 2022-06-08]. Available from: https://martinfowler.com/eaaCatalog/dataTransferObject.html.
25. *npm* [online] [visited on 2022-06-20]. Available from: https://www.npmjs.com/.
26. ADEBAYO, Segun. Comparison. *Chakra UI Documentation* [online] [visited on 2022-06-20]. Available from: https://chakra-ui.com/guides/comparison.
27. ANT UED. *Ant Design* [online] [visited on 2022-06-20]. Available from: https://ant.design/components/icon/.
28. HACKERONE INC. *React Datepicker* [online]. 2014 [visited on 2022-06-20]. Available from: https://reactdatepicker.com/.
29. REMIX. *React Router* [online] [visited on 2022-06-20]. Available from: https://reactrouter.com/.
30. *React Hook Form* [online] [visited on 2022-06-20]. Available from: https://react-hook-form.com/.
31. *Cloudinary* [online] [visited on 2022-06-20]. Available from: https://cloudinary.com/.
32. The State of QR in 2021. *Blue Bite* [online] [visited on 2022-06-20]. Available from: https://www.bluebite.com/nfc/qr-code-usage-statistics.
33. *QR Platba* [online] [visited on 2022-06-20]. Available from: https://qr-platba.cz/. Translation provided by the thesis's author.
34. *ChartJS* [online] [visited on 2022-06-20]. Available from: https://www.chartjs.org/.
35. *react-chartjs-2* [online] [visited on 2022-06-20]. Available from: https://react-chartjs-2.js.org/.
36. *Heroku* [online]. 2007 [visited on 2022-06-20]. Available from: https://www.heroku.com/.
37. *GitHub* [online] [visited on 2022-06-20]. Available from: https://github.com/.
38. *GitLab* [online] [visited on 2022-06-20]. Available from: https://gitlab.fit.cvut.cz/.
39. GOOGLE INC. *Firebase* [online] [visited on 2022-06-20]. Available from: https://firebase.google.com/.

Content of enclosed CD

- `readme.md` ..... short description of the CD content
- `src`
  - `impl`
    - `be` ..... source code of the back end app
    - `fe` ..... source code of the front end app
    - `create_script.txt` ..... a create script for the database
  - `prototype.pdf` ..... the app's prototype in PDF
  - `thesis` ..... source code of the thesis in LaTeX
- `text`
  - `thesis.pdf` ..... the thesis in PDF