FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Webová aplikace pro přehled osobních výdajů
Student:	Vladyslav Volodin
Vedoucí:	Ing. Marek Suchánek
Studijní program:	Informatika
Studijní obor:	Webové a softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce letního semestru 2020/21

Pokyny pro vypracování

Cílem práce je na základě analýzy navrhnout a implementovat webovou aplikaci, která umožní uživatelům tvořit a spravovat přehledy o osobních výdajích v různých kategoriích. Při vývoji aplikace musí být brán důraz na možnosti rozšíření aplikace nezávislými moduly či pluginy. Dále budou uplatněny tradiční metody softwarového inženýrství.

- Analyzujte důkladně vybraná existující řešení pro tvorbu přehledů osobních výdajů, vytvořte doménový konceptuální model a sestavte požadavky na aplikaci.
- Navrhněte webovou aplikaci využívající Spring Framework a PostgreSQL databázi pro přehled osobních výdajů. Architektura aplikace a použité technologie musí umožňovat snadnou rozšiřitelnost. Volbu dalších technologií řádně zdůvodněte.
- Implementujte, zdokumentujte a otestujte aplikaci dle návrhu. V rámci implementace vytvořte alespoň 1 nezávislý modul/plugin pro demonstraci rozšiřitelnosti.
- Zhodnoťte přínosy použití vytvořené aplikace pro uživatele a porovnejte ji s analyzovanými řešeními.

Seznam odborné literatury

Dodá vedoucí práce.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
děkan

V Praze dne 6. listopadu 2019

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Bakalářská práce

Webová aplikace pro přehled osobních výdajů

Vladyslav Volodin

Katedra Softwarového inženýrství
Vedoucí práce: Ing. Marek Suchánek

4. června 2020

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů, zejména skutečnost, že České vysoké učení technické v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona.

V Praze dne 4. června 2020 . . . . . .

Odkaz na tuto práci

Volodin, Vladyslav. *Webová aplikace pro přehled osobních výdajů.* Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2020.

Abstrakt

Tato práce je věnovana návrhu a implementaci webové aplikace pro přehled a analýzu osobních výdajů. Implementace je provedená v jazycích Java s použitím nástroje Spring Framework pro serverovou a JavaScript s ReactJS pro prezentační části. Výstupem práce je prototyp aplikace, která poskytuje uživateli přehled o osobních výdajích, a udává možnost jejích snadného plánování.

Klíčová slova webová aplikace, přehled osobních výdajů, Java, Spring Framework, JavaScript, ReactJS.

Abstract

This work is devoted to design and implementation of web application for overview and analysis of personal expenses. Implementation is done in Java using Spring Framework for server and JavaScript with ReactJS for the presentation parts. The output of the work is the prototype of application that provides the user with an overview of personal expenses, and indicates the possibility of easy spending planning.

Keywords web application, personal expense management, Java, Spring Framework, JavaScript, ReactJS.

Obsah

Seznam obrázků

Seznám výpisů kódu

Úvod

Peníze jsou nezbytnou částí našeho života. Skoro každý den něco kupujeme: buď nějaké hmotné prostředky, nebo platíme za obrovskou množinu služeb. Peníze máme na bankovních účtech, v peněženkách atd. Nikdo z nás není dokonalý, a proto určitě nemůže pamatovat, kolik peněz zaplatil tento měsíc v kině, restauracích, na relaxaci, za předplatné... A kvůli tomu i nemůže svoje výdaje kategorizovat, spravovat, zlepšovat kvalitu svého života a šetřit na tom, co vlastně nepotřebuje. Situace se mnohém zhoršuje, když člověk má více, než jeden bankovní účet a směruje své peníze mezi nimi.

Znát své výdaje, vědět, které patří mezi ty největší je moc důležitým přínosem u vedení osobních financí. Ale ne každý člověk má čas a chuť zapisovat svoje příjmy a výdaje na papír, a pak na konci měsíce je analyzovat. Pomoc v řešeni tohoto problému poskytují různé aplikace, které udělají to mnohem rychleji a efektivněji. Od uživatele se chce jen pravidelně zadávat svoje příjmy a výdaje do programu, někdy provést nějaké počáteční nastavení, když celý ten zbytek aplikace provede sama: výpočte součty, připraví názornou, pochopitelnou pro uživatele analýzu jeho příjmů a výdajů, a velkou množinu jiných užitečných věcí.

Téma jsem si zvolil, neboť problém podle mého názoru nebyl uspokojivě vyřešen a taky protože implementace podobné webové aplikace mi povolí aplikovat znalosti a ty nejzajímavější nástroje a postupy, kterým jsem se na univerzitě naučil.

Výsledek práce bude prospěšný pro ty uživatele, které utrácejí mnoho peněz a pak nepamatují na co, které potřebují šetřit, a kvůli tomu, že nevedou evidenci výdajů – nemohou vědět, jaké kategorie tvoří největší část jejích výdajů.

Kapitola 1

Cíle a struktura práce

Cílem práce je na základě analýzy navrhnout a implementovat webovou aplikaci, která umožní uživatelům tvořit a spravovat přehledy o osobních výdajích v různých kategoriích.

Je nutné analyzovat současné důkladně vybrané existující řešení pro tvorbu přehledů osobních výdajů. Jako součást práce je potřeba vytvořit doménový konceptuální model a sestavit požadavky na aplikaci.

Nezbytnou částí je implementovat, zdokumentovat a otestovat aplikaci, která by podporovala snadnou rozšiřitelnost, dle návrhu.

V rámci implementace se musí vytvořit alespoň 1 nezávislý modul pro demonstraci rozšiřitelnosti.

Jako závěr se musí zhodnotit přínosy použití vytvořené aplikace pro uživatele a porovnat ji s analyzovanými řešeními.

Práce je rozdělená na kapitoly, které v sobě obsahují splnění jednotlivých dílčích cílů práce. Při analýze, návrhu a implementaci se postupuje v souladu s metodami softwarového inženýrství.

První částí práce je „Rešerše problému osobních financí“, kde probírám proč je nutné kontrolu osobních výdajů vést. Následuje ji „Rešerše existujících řešení“, kde probírám již existující aplikace, které podle mého názoru nejlépe řeší ten problém. Pak je část „Analýza a návrh“, kde se zabývám analýzou požadavků, přípravou případů užití, a návrhem architektury aplikace. Poslední část je „Realizace“, kde popisuji implementace serverové a prezentační částí webové aplikace, a jejích propojení.

Kapitola 2

Rešerše problému osobních financí

Peníze jsou především používané k dosažení cílů, ke kterým se každý člověk směruje, a které považuje za ty nejdůležitější. Plyne to i z následujícího tvrzení, které uvádí Luboš Smrčka v jedné ze svých knih o osobních financích:

„Veškerá lidská ekonomická činnost směruje k uspokojení sobeckých potřeb…“ [1, s. 21]

Z toho plyne i to, že potřebujeme mít dostatek finančních prostředku k dosažení svých cílů, protože v současném světě se všechno kupuje za peníze.

Nejsme všichni ekonomové, a očividně nemáme ty nejdůležitější ekonomické znalosti, které potřebujeme k dosažení bohatství. Ale jak se ukazuje – není to ani větší část toho, díky čemu se lidí stávají bohaté [1, s. 11], a jako důsledek úspěšné.

2.1 Výdaje

Důležité je taky upozornit na jednu věc, která značným způsobem ovlivňuje bohatství rodin a jednotlivců – zásadní složkou rozpočtu jsou v první řadě výdaje, a nikoliv příjmy. [1, s. 55]

„Více peněz, většího majetku lze dosáhnout dvěma metodami. Buď soustavným a uspokojují růstem příjmu, zvyšováním výnosu investic a vysokém stálým příjmem. Nebo také minimalizaci vydání a tlačení úrovně výdajů stále a systematicky pod úroveň příjmu. Mohli bychom to nazvat cestou příjmovou a cestou výdajovou.“ [1, s. 56]

Může se stát, že někdo z nás si představuje cestu k bohatství tak, že bude postupně zvyšovat své příjmy, dokud nebude plně spokojený s kvalitou svého života. Ale bohužel ne všichni mají ty ambice a prostředky ke skoro nekonečnému zvýšení svého kapitálu. [1, s. 56]

Proto je moc důležité kromě postupného zvýšení svých příjmů se taky hodně pečlivě zabývat tím, čím jsou osobní výdaje tvořené, a jak jsou sladěny s příjmy. [1, s. 56]

2.2 Potřeby

Každý člověk má v genech chuť nakupovat věcí, které nemá, a není nutné, aby ty opravdu potřeboval nebo měl na to dostatek peněz. [1, s. 58]

„Potřeby jsou nekonečné. Naprosto nekonečné. Pokud je necháme působit, spotřeba může růst bez omezení tak dlouho, dokud existují zdroje k jejímu pokrývání. Na jedné straně právě tohoto žene ekonomiku a celý svět vpřed, na straně druhé to znamená největší ohrožení celého systému.“ [1, s. 58]

Všichni z nás máji své potřeby, které se určitě nějakým způsobem dá roztřídit (někdy to opravdu může být netriviální úkol) do dvou kategorií – důležité a nedůležité. K prvním se například dá odnést takové věci, jako jídlo, pití, bydliště, atd. Ten zbytek je už individuální pro každého. Často je priorita dávána v souladu s Maslowovou pyramidou potřeb 2.1.

Obrázek 2.1: Maslowova pyramida potřeb [2]

Je dobře pro případy, kdy člověk má omezené peněžní prostředky, pečlivě kontrolovat svoje výdaje, a někdy, například dle pyramidy 2.1, uspořádat je dle potřeby – omezovat nějakou z kategorií, aby byla možnost dávat více peněz do jiné důležitější.

2.3 Šetření a Spoření

Jak bylo mnou ukázáno v sekci 2.2, jídlo a pití patří do důležité, a navíc taky nezbytné kategorie, ale i v tom se jednoduše dá ušetřit. Opravdu jeden člověk nepotřebuje 40 housek denně, proto je důležité sledovat své výdaje a šetřit na tom, co je nadbytečné nebo stojí víc, než člověk si může dovolit zaplatit.

„Kritická analýza běžného dne nebo týdne rodiny s lepšími příjmy přináší z hlediska výdajů obvykle velmi šokující zjištění – přinejmenším desetina a spíše ještě větší část našich běžných výdajů je vynaložena naprosto bez smyslu.“ [1, s. 59]

Patří k tomu i veškeré potraviny, které se nestihly spotřebovat a byly následně vyhozeny, nepoužívaný obsah šatníku, elektrotechnika, zahradní stroje, nepoužívané auto, za které se platí pojištění, atd. V zavilosti na tom, co platí pro náš konkretní případ, dá se na tom ušetřit různé, ale většinou docela velké částky peněz. Důležité je mít to pod kontrolou a pravidelně analyzovat situace, které k nadbytečné spotřebě vedou. [1, s. 61]

Všechno, co jsme ušetřily, je dobrým nápadem někam ukládat aby nám něco zůstalo do budoucna. Jedním z nejlepších příkladů místa, kam lze uložit peníze jsou banky, protože je to tradiční, bezpeční a většinou proděleč ný způsob, jak naložit s penězi. [1, s. 78]

„Spoření je ukládání peněz na různorodé bankovní a spořitelní účty. V právem smyslu slova je tím myšleno pravidelné ukládání stále stejných částek po určitou dobu“ [1, s. 77]

Ale pokud chceme, pravidelně ukládat své peníze na tyto účty, potřebujeme je někde vzít. Proto je důležité své výdaje analyzovat, optimalizovat a šetřit na tom, co nepotřebujeme.

2.4 Shrnutí

Hlavní myšlenkou je analyzovat a snižovat výdaje, protože se nedá do nekonečná zvyšovat příjmy. Důležité je rozdělovat potřeby na kategorie, a omezovat ty z nich, které nejsou nezbytné, aby byla možnost investovat do těch nejdůležitějších.

Pokud chceme šetřit, musíme vždy mít aktuální přehled o výdajích, abychom vždy věděli, která z kategorií výdajů spotřebovává nejvíce peněz, a podle toho bychom mohli jí nějak omezovat.

Na základě této kapitoly jako nezbytné věci pro webovou aplikaci považuji:

- možnost vytvářet různé účty,
- třídění výdajů podle kategorií,
- možnost nastavení limit pro jednotlivé kategorie výdajů, aby bylo možné je pečlivě kontrolovat, uspořádávat dle potřeby, a neutrácet v nich více, než je potřeba,

- zobrazení zbývajících peněz na účtech, a součtů výdajů dle kategorií,
- kromě aktuální útraty do kategorie taky zobrazovat pomocí diagramu, jakou část od všech výdajů konkretní kategorie tvoří.

Kapitola 3

Rešerše existujících řešení

Pro analýzu existujících řešeni jsem zvolil následující 3 aplikace, které nejlépe ukazuji to, co já jako obyčejný uživatel očekávám od podobné aplikace (všichni jsou pro mobilní zařízení, protože nic podobného s webovým rozhraním kromě Wallet se mi nepodařilo najít):

1. „Monefy – Money Manager“ [3],
2. „Wallet – Daily Budget & Profit“ [4],
3. „CoinKeeper: budget planner“ [5].

Vybíral jsem ty dle nejdůležitějších podle mého názoru příležitostí:

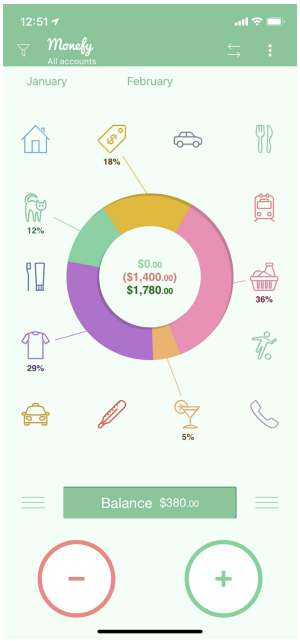
- barevné a minimalistické uživatelské rozhraní,
- třídění výdajů dle kategorií,
- zobrazení součtů peněz utracených v určité kategorii,
- přehledná analýza výdajů,
- jednoduchost použití.

3.1 Rozbor jednotlivých aplikací

3.1.1 Monefy

Na hlavní obrazovce aplikace Monefy na obrázku 3.1 uprostřed vidíme kolečko, rozdělené barvami na fragmenty, symbolizující procentuální část určité kategorie v celkovém seznamu výdajů. Dolů se nachází dva velkých tlačítka „+“ a „−“, jenže za mě tlačítko plus by mělo znamenat „přidat novou transakci“, ale fakticky plus znamená příjem, minus – naopak výdaj.

Práce s kategoriemi výdajů a účty pro mě není jednoduchá a názorná. Pro moje případy užití musím zmáčknout příliš hodně tlačítek a otevírat stejně mnoho obrazovek. [3]



Obrázek 3.1: Monefy – Money Manager [3]

Výhody

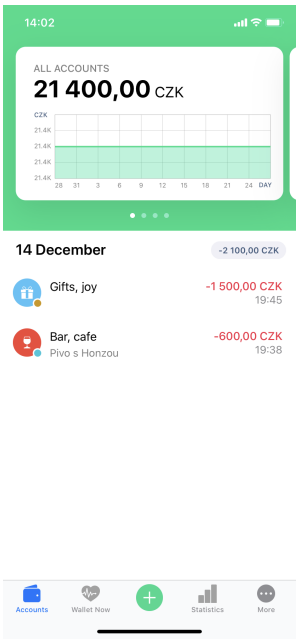
- Názorné zobrazení výdajů pomoci diagramu,
- Synchronizace přes Dropbox,
- Volba měny,
- Možnost přidávat vlastní kategorie výdajů.

Nevýhody

- Těžko pochopitelné a neintuitivní uživatelské rozhraní,
- Diagram výdajů není přesně to, co by mělo zobrazovat vždy,
- Nezobrazuje se pršený počet utracených peněz dle kategorie na hlavni obrazovce.

3.1.2 Wallet

Zakladatelem společnosti BudgetBakers, která vyvinula tuto aplikaci je Jan Müller, který předtím pracoval v Seznam.cz, AVAST a České spořitelně. Wallet je velmi populární aplikace pro kontrolu výdajů. Podporuje rozmanitost různých



Obrázek 3.2: Wallet [4] – hlavní obrazovka

Obrázek 3.3: Wallet [4] – statistika výdajů

měn a pomáhá uživateli přesně naplánovat rozpočet. Je možné ji používat zdarma, nezobrazuje žádnou reklamu, ale stavem na 31.3.2020 má i placené předplatné za 2,99$ měsíčně, které dovoluje mít nekonečně mnoho účtů, nastavit automatické třídění bankovních transakci, pokročilou statistiku, atd.

Je to celkem dobrá aplikace, není tak přehledná, jak následující, ale dá se ji komfortně používat. Na obrázku 3.2 je vidět přehledná minimalistická hlavní obrazovka s seznamem posledních plateb a grafem zůstatku na vybraném účtu nebo součtu všech zůstatků. Dolů je tlačítko „+“, které na rozdíl od aplikace Monefy [3] jednoznačně znamená „přidat novou transakci“.

Na obrázku 3.3 je vidět příklad statistiky, kde kliknutím po segmentům kolečka se zobrazí název kategorie a přesná částka zaplacená v dánem období. [4]

Výhody

- Synchronizace s bankami,
- Pochopitelné uživatelské rozhraní,
- Přehledná statistika,
- Dobrá webová aplikace,
- Volba měny.

Nevýhody

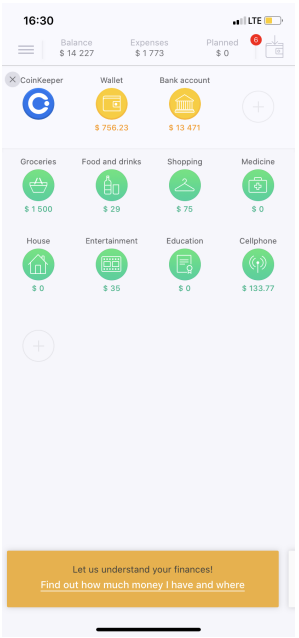
- Na hlavní obrazovce není vidět, kolik bylo celkem utraceno v určitých kategoriích, je potřeba otevírat statistiku,
- Nelze nastavit limit pro účet.

3.1.3 CoinKeeper

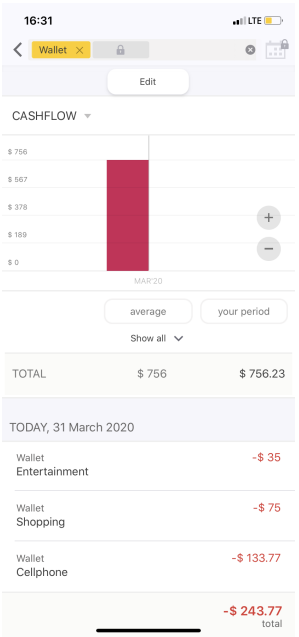
CoinKeeper se objevil v roce 2011 v jednom z oddělení ruské společnosti i-Free a okamžitě upoutal pozornost a stal se jednou z nejstahovanějších aplikací v App Store.

V roce 2013 byla založena společnost Disrapp, která se nyní plně zaměřuje na vývoj CoinKeeper. Podle odhadu Roem.ru 16.03.2015 byla tato aplikace lídrem na trhu osobních finančních služeb v Rusku [6].

Hlavní obrazovka (obrázek 3.4) představuje dva seznamy: účtů a kategorií výdajů. Manipulace se provádí „přetahováním“ mince z účtu do kategorie výdajů a zapisováním přesné částky. Je možnost zobrazit cashflow (obrázek 3.5) a nastavit limity. Dá se ji používat zdarma, ale má rozmanitost funkcí, které jsou přístupné pouze v placené verzí. [5]



Obrázek 3.4: CoinKeeper [5] – hlavní obrazovka



Obrázek 3.5: CoinKeeper [5] – kategorie „Wallet“

Výhody

- Zobrazení součtů v horní části aplikace,
- Zobrazení účtů, kategorii výdajů a limit na hlavni obrazovce,
- Možnost přidáváni vlastních účtů a kategorii výdajů,
- Nastaveni limit,
- Pochopitelné, jednoduché a lehké uživatelské rozhraní a ovládání,
- Jednoduchá editace,
- Volba měny.

Nevýhody

- Větší část funkcionality pouze v placené verzi,
- Bezplatná verze je velmi omezena,
- Není webová aplikace – nelze použít na počítací,
- Analýza jen pomoci cashflow, názornější by byl diagram výdajů dle kategorií.

3.2 Shrnutí existujících řešení

Ze všech probraných aplikací moje požadavky nejlépe splňuje CoinKeeper, osobně používám pravě tuto implementaci. Ale velkými problémy jsou nedostatek webové aplikace nebo počítačového programu a analýza pomocí cashflow místo názorného diagramů, jak je to například u Monefy nebo Wallet.

Při implementaci se budu inspirovat minimalistickým uživatelským rozhraním CoinKeeper a jeho myšlenkou zobrazovat všechny kategorie se součty na hlavní obrazovce, a diagramy u Monefy a Wallet.

Kapitola 4

Analýza a návrh

4.1 Analýza požadavků

Podle autorů „Requirements Engineering: A Roadmap“ [7], Pamela Zave poskytuje jednu z nejjasnějších definic RE:

„Inženýrství požadavků je softwarové inženýrství zaměřené na skutečné cíle, funkce a omezení softwarových systémů. Vztahuje se také na jejich vztahové faktory přesné specifikace chování softwaru, jejich vývoj v průběhu času a napříč rodinami softwaru.“ [8]

4.1.1 Funkční požadavky

„Funkční požadavky popisují chování produktu, výsledku nebo služby, např. formou popsaného procesu nebo interakce s okolím.“ [9] – PM Consulting.

1. Vytvářet peněžní účty – na obrazovce vedle seznamu účtů bude tlačítko, stisknutím kterého se uživatel dostane do dialogu vytvoření nového účtu, bude mít možnost volby názvu a počátečního zůstatku.

2. Editovat peněžní účty – v dialogu, který se zobrazí po rozkliknutí účtu bude tlačítko editace, které umožní změnit všechny údaje, které se zadávali při vytváření.

3. Vytvářet kategorie výdajů – na obrazovce vedle seznamu kategorií výdajů bude tlačítko, stisknutím kterého se uživatel dostane do dialogu vytvoření nové kategorie výdajů, bude mít možnost volby názvu, případně limity.

4. Editovat kategorie výdajů – v dialogu, který se zobrazí po rozkliknutí kategorie výdajů bude tlačítko editace, které umožní změnit všechny údaje, které se zadávali při vytváření.

5. Přidávat nepravidelné příjmy do účtu – v detailu účtu bude tlačítko, stisknutím kterého se uživatel dostane do dialogu přidávání nepravidelného příjmu, kde vyplní částku a volitelně komentář.

6. Zobrazit v pochopitelné formě analýzu výdajů.

7. Přenášet peníze mezi účty – v dialogu, který se zobrazí po rozkliknutí účtu bude tlačítko přenášení, stisknutím kterého se uživatel dostane do dialogu, kde zvolí cílový účet a částku.

8. Přenášet peníze z účtu do kategorie výdajů – v dialogu, který se zobrazí po rozkliknutí kategorie výdajů bude tlačítko přidávání výdaje, stisknutím kterého se uživatel dostane do dialogu, kde zvolí účet a částku.

9. Zobrazit seznam plateb v kategoriích výdajů a v účtech – v dialogu, který se zobrazí po rozkliknutí kategorie výdajů nebo účtu se zobrazí seznam všech souvislých plateb.

10. Zobrazit aktuální součet peněz na všech účtech a součet výdajů v aktuálním měsíci – na hlavní obrazovce se budou zobrazovat všechny součty.

11. Nastavení limity pro platby do kategorie výdajů (slouží jen pro varování, nikoliv pro zakázání platby) – v dialogu, který se zobrazí po rozkliknutí kategorie výdajů bude tlačítko nastavení limity pro platby.

12. Nastavení limity pro výdaje v aktuálním měsíci.

4.1.2 Nefunkční požadavky

„Nefunkční požadavky jsou doplněk funkčních požadavků. Popisují další nezbytné vlastnosti potřebné vzhledem k prostředí a kontextu. Např. se jedná o požadavky na spolehlivost, bezpečnost, výkonost, podporu během provozu atp.“ [10] – PM Consulting.

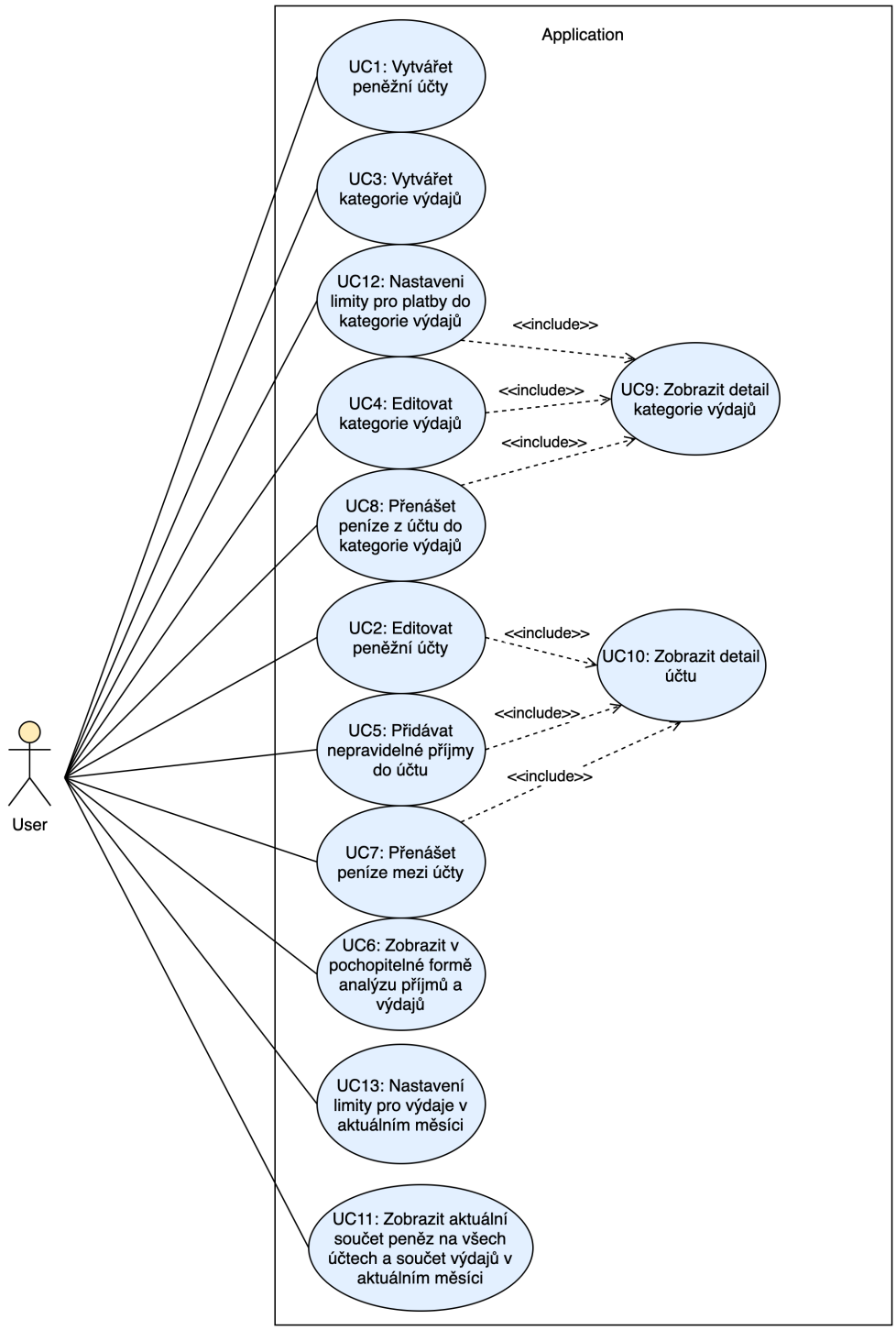
1. Webová aplikace – aplikace bude dostupná standardním webovým prohlížečem přes protokol http.

2. Aplikace musí být spolehlivá a zajišťovat konzistenci dat.

3. Aplikace musí být rozšiřitelná.

4.2 Případy užití

V této kapitole se nachází výčet případů užití (use cases). Počítá se s jednou rolí – Uživatel, který vlastně aplikaci používá. Všichni případy užití plynou z funkčních požadavků, popsaných v kapitole 4.1.1. Každý z nich je doplněn krátkým stručným popisem. Model případů užití je vidět na obrázku 4.1.



Obrázek 4.1: Model případů užití

UC1: Vytvářet peněžní účty

- Uživatel otevře hlavní obrazovku.
- Zde vedle seznamu peněžních účtů stiskne tlačítko „Přidat účet“.
- Vyplní všechny potřebné údaje (název, měnu, zůstatek...).
- Stiskne potvrzující tlačítko.

UC2: Editovat peněžní účty

- Uživatel otevře detail účtu (UC10).
- Stiskne tlačítko editace.
- Vyplní všechny potřebné údaje.
- Potvrdí stisknutím potvrzujícího tlačítka.

UC3: Vytvářet kategorie výdajů

- Uživatel otevře hlavní obrazovku.
- Zde vedle seznamu kategorií výdajů stiskne tlačítko „Přidat“.
- Vyplní všechny potřebné údaje (název, měnu...).
- Stiskne potvrzující tlačítko.

UC4: Editovat kategorie výdajů

- Uživatel otevře detail kategorie výdajů (UC9).
- Stiskne tlačítko editace.
- Vyplní všechny potřebné údaje.
- Potvrdí stisknutím potvrzujícího tlačítka.

UC5: Přidávat nepravidelné příjmy do účtu

- Uživatel otevře detail účtu (UC10).
- Stiskne tlačítko přidání příjmu.
- V překrývajícím dialogu zvolí částku.
- Potvrdí stisknutím potvrzujícího tlačítka.

UC6: Zobrazit v pochopitelné formě analýzu výdajů

- Na hlavní obrazovce bude vidět výdajový diagram.

UC7: Přenášet peníze mezi účty

- Uživatel otevře detail účtu (UC10).
- Stiskne tlačítko přidání platby.
- V překrývajícím dialogu zvolí účet a uvede částku.
- Potvrdí stisknutím potvrzujícího tlačítka.

UC8: Přenášet peníze z účtu do kategorie výdajů

- Uživatel otevře detail kategorie výdajů (UC9).
- Stiskne tlačítko přidání platby.
- V překrývajícím dialogu zvolí zdroj peněz – účet, a uvede částku.
- Potvrdí stisknutím potvrzujícího tlačítka.

UC9: Zobrazit detail kategorie výdajů

- Uživatel otevře hlavní obrazovku.
- Stiskne na dlaždici, reprezentující určitou kategorii.
- V překrývajícím dialogu uvidí detail a seznam plateb.

UC10: Zobrazit detail účtu

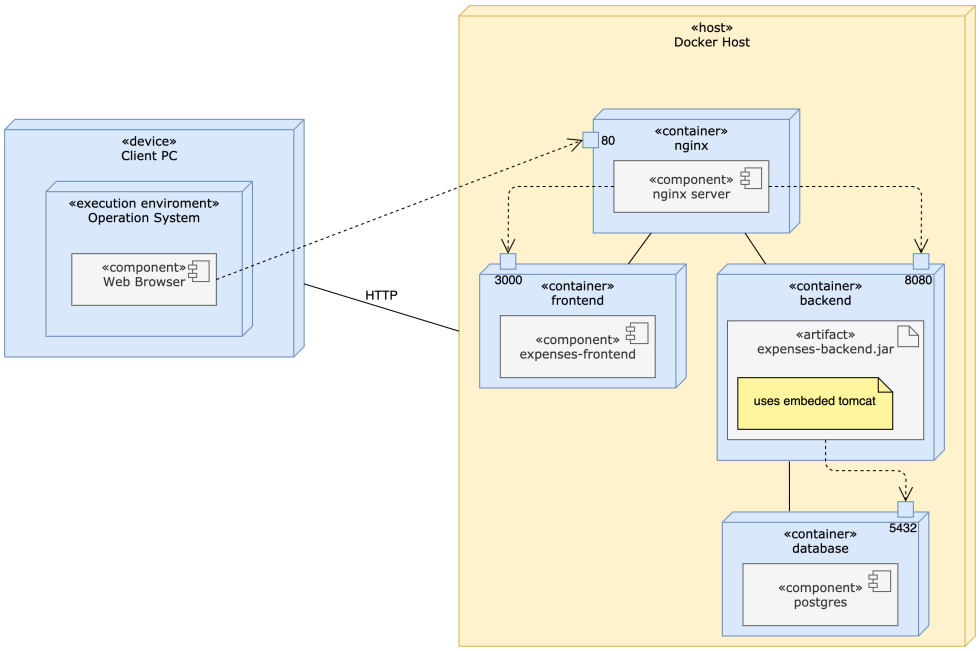
- Uživatel otevře hlavní obrazovku.
- Stiskne na dlaždici, reprezentující určitý účet.
- V překrývajícím dialogu uvidí detail a seznam plateb.

UC11: Zobrazit aktuální součet peněz na všech účtech a součet výdajů v aktuálním měsíci

- Uživatel otevře hlavní obrazovku.
- Nahoře uvidí všechny součty.

UC12: Nastaveni limity pro platby do kategorie výdajů

- Uživatel otevře detail kategorie výdajů (UC9).
- Přejde do stavu editace.
- Upraví limitu.



Obrázek 4.2: Diagram nasazení

- Potvrdí stisknutím tlačítka „Uložit“.

UC13: Nastavení limity pro výdaje v aktuálním měsíci

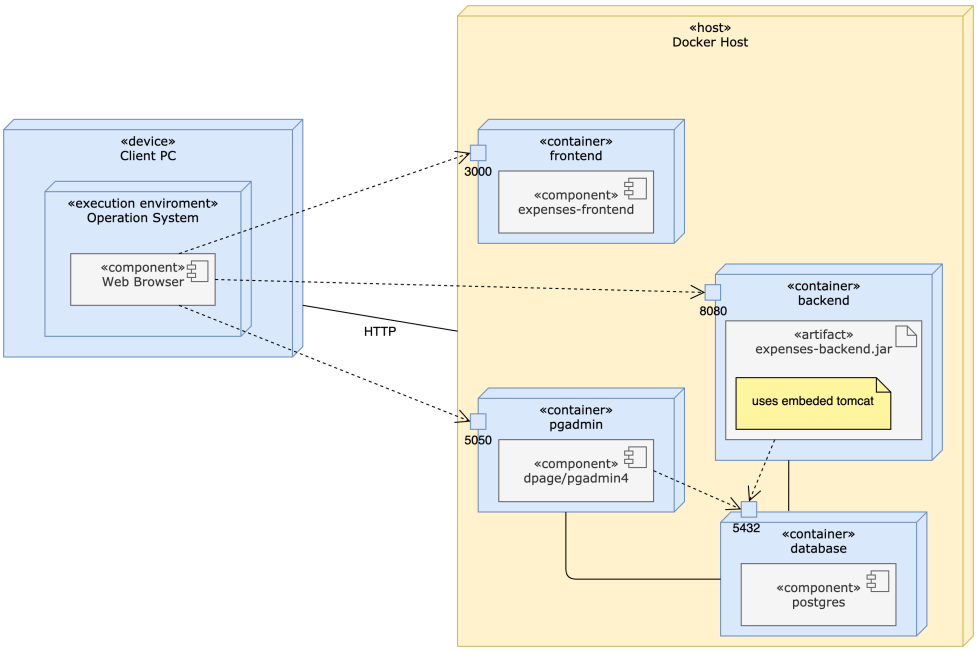
- Uživatel otevře hlavní obrazovku.
- Stiskne tlačítko editace limity u sekce „Plánováno“.
- Nastaví měsíční limitu.
- Potvrdí stisknutím tlačítka „Uložit“.

4.3 Architektura a design

Tato část je především věnovaná popisu návrhu a architektury vyvíjené aplikace. Připravil jsem ty nejdůležitější diagramy, které podle mého názoru jsou nezbytné při vytváření aplikace takového druhu.

4.3.1 Architektura

Protože se zabývám především návrhem webové aplikace, mezi nejdůležitějšími části patří rozhodnuti o způsobu nasazení a architektuře. Tomu se musí dávat největší pozornost, aby byla v budoucnu možnost rychlého nasazení, snadné



Obrázek 4.3: Diagram nasazení – počítač vývojáře

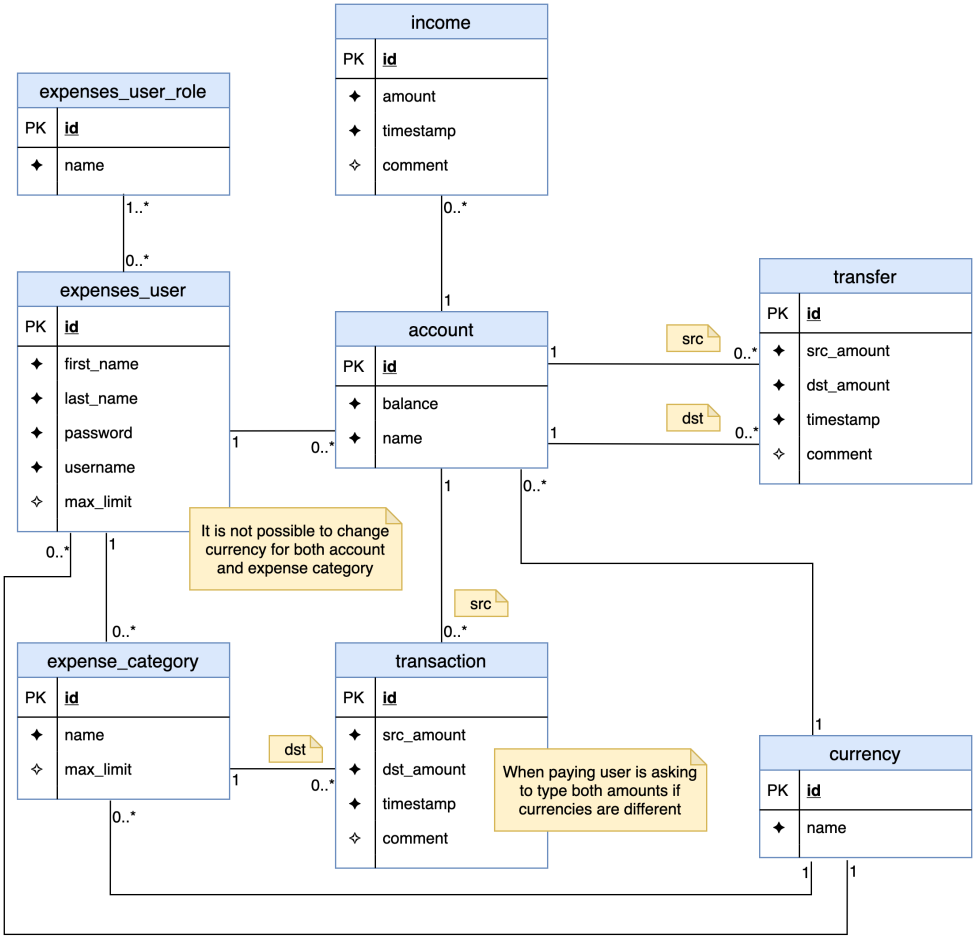
rozšiřitelnosti a škálování při vysokém zatíženi serveru, taky důležitou rolí má bezpečnost aplikace a vysoká dostupnost. Proto jsem zvolil kontejnerovou architekturu, která velmi dobře splňuje tyto požadavky [11], [12], [13].

Jako technologii, poskytující kontejnerovou architekturu jsem zvolil Docker [14] díky jeho popularitě a velkému počtu firem, využívajících ho ve svých projektech [15]. Moje volba taky byla ovlivněna tím, že Docker [14] pomáhá vyvíjet rychle, snadno spolupracovat a při tom zůstává jednoduchým v použití [16].

Aplikace jsem rozhodl rozdělit na serverovou a prezentační části, každou ve svém vlastním kontejneru. Komunikace s první probíhá pomoci RESTful [17] rozhraní přes HTTP protokol. Umožňuje to taky snadnou rozšiřitelnost, například vytváření mobilní aplikace, která by komunikovala se stejným serverem, a díky architektuře nevyžaduje provádění změn v existujících modulech.

Všechno popsáné jsem zobrazil na diagramu nasazení 4.2. Kromě výše zmíněného je tam vidět i databáze ve vlastním kontejneru, a NGINX server [18], který se především využívá jako reverse proxy [19]. Díky tomu může uživatel komunikovat s aplikací na portu 80, skrývá se identita serverů, a navíc se dá nakonfigurovat load balancing, kompresi dat, a SSL šifrování [19].

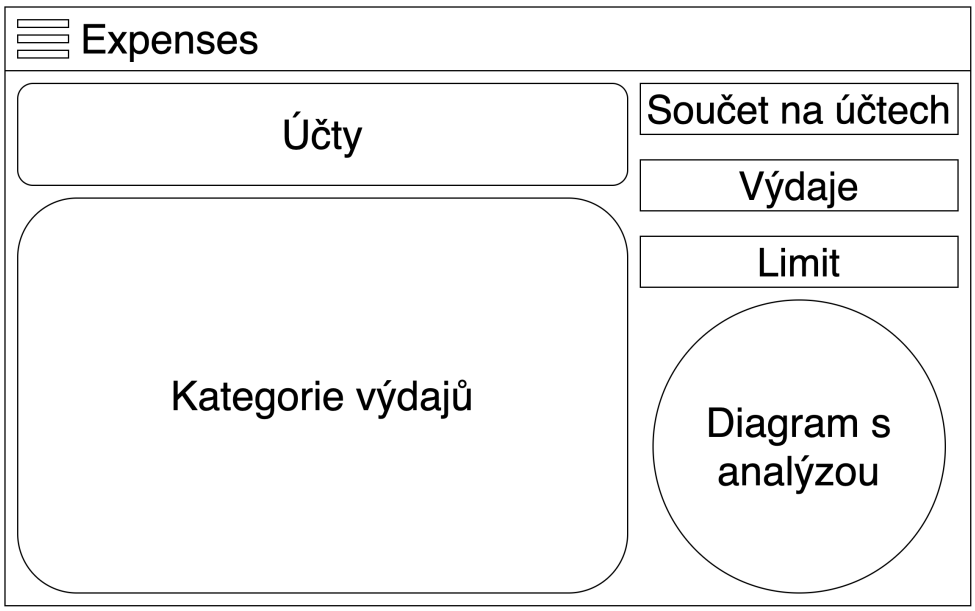
Jenže pro účely vývoje není potřeba mít u sebe reverse-proxy server, proto jsem potřeboval navíc vytvořit development konfiguraci pro vývojáře, kterou je vidět na diagramu 4.3. Kromě odstranění NGINX [18] jsem přidal kontejner s aplikaci pgAdmin [20], která umožňuje jednodušší práci s databází přes GUI.



Obrázek 4.4: Databázový model

4.3.2 Databáze

Podle zadání této práce databází je PostgreSQL [21]. Iniciální schemu: entity a jejích relace jsem popsal na entitně-relačním diagramu 4.4. Pokrývá všechno, co je potřeba k realizací funkčních požadavků 4.1.1 a dobrému fungování a rozšiřování aplikace v budoucnu. Jak je vidět, entity Transaction a Transfer by se daly sjednotit pomocí dědičnosti, protože mají stejné atributy, ale kvůli tomu, že jsou to konceptuálně různé věci, a v průběhu údržby se mohou neustálé a nezávislé měnit, a taky pro účely jednoduché podpory jsem je nechal reprezentovat různými tabulkami.



Obrázek 4.5: Wireframe hlavní obrazovky

4.3.3 Grafické rozhraní

Pro případ této webové aplikace podle mě je vystačující pouze model hlavní obrazovky. Má v sobě většinu funkcionalit a ten zbytek se bude provádět pomocí překrývajících dialogů. Na obrázku 4.5 je vidět wireframe hlavní obrazovky, která obsahuje dvě sekce: jedna s jednotlivými účty a jejich zůstatky, a druha s kategoriemi výdajů a aktuální útratou v nich. Vpravo jsou tři sekce se součtem zůstatků na účtech, součtem všech výdajů, a plánovaným limitem. V pravé sekci dolů je diagram s analýzou, na kterém je vidět, do jaké kategorie v aktuálním měsíci zaplatil uživatel nejvíce peněz.

Kapitola 5

Realizace

5.1 Použité technologie

5.1.1 Spring Framework

Spring usnadňuje vytváření enterprise aplikací Java. Poskytuje vše, co vývojář potřebuje pro používání jazyka Java v enterprise prostředí, s podporou Groovy a Kotlin jako alternativních jazyků v JVM, a s flexibilitou pro vytváření mnoha druhů architektur v závislosti na potřebách aplikace. Od verze frameworku 5.1 vyžaduje JDK 8+ (Java SE 8+) a poskytuje podporu JDK 11 LTS „out of the box“ [22].

Spring Framework je rozdělen do modulů, a aplikace si mohou vybrat ty, které potřebují. Srdcem jsou moduly kontejneru jádra, včetně konfiguračního modelu a mechanismu dependency injection [22].

5.1.2 Spring Boot

Konfigurace Springu pro enterprise aplikace se kvůli obtížné konfiguraci závislosti stala velmi zdlouhavou a náchylnou k chybám. To platí zejména pro aplikace, které také používají několik knihoven třetích stran. Proto autoři Springu se rozhodli poskytnout vývojářům nástroje, které automatizují proces konfigurace a urychlují proces vytváření a nasazení Springových aplikací, souhrnně nazývané Spring Boot. [23]

Důležité vlastnosti:

- snadné řízení závislosti – aby se urychlil proces správy závislostí, Spring Boot implicitně balí nezbytné knihovny třetích stran pro každý typ aplikace založené na Springu a poskytuje je vývojářům prostřednictvím tzv. startovacích balíčků,

- automatická konfigurace – po zvolení vhodného startovacího balíčku se Spring Boot pokusí automaticky nakonfigurovat aplikaci na základě přidaných závislostí a parametrů v souboru application.properties,
- nativní podpora aplikačního serveru – kontejneru servletů.

5.1.3 Spring Security

Spring Security je Java/JavaEE framework, který poskytuje mechanismy pro vytváření autentizačních a autorizačních systémů, jakož i další bezpečnostní funkce pro podnikové aplikace vytvořené pomocí Spring Framework. Projekt zahájil Ben Alex na konci roku 2003 pod názvem „Acegi Security“, první vydání vyšlo v roce 2004. Následně byl projekt absorbován společností Spring a stal se jeho oficiálním pomocným projektem. Poprvé byl veřejně představen pod novým názvem Spring Security 2.0.0 v dubnu 2008.

Spring Security je framework, který se zaměřuje na poskytování autentizace a autorizace Java aplikacím. Stejně jako všechny Spring projekty je skutečná síla Spring Security nalezena v tom, jak snadno může být rozšířený, aby splňoval požadavky vývojového týmu. [24]

Důležité vlastnosti:

- komplexní a rozšiřitelná podpora pro autentizaci i autorizaci,
- ochrana proti útokům, jako je session fixation, clickjacking, cross site request forgery atd,
- integrace Servlet API,
- volitelná integrace se Spring Web MVC.

5.1.4 Spring Data

Hlavním cílem Spring Data je poskytnout známý a konzistentní springový programovací model pro přístup k datům a přitom si zachovat zvláštní vlastnosti základního datového úložiště. [25]

Usnadňuje použití technologií přístupu k datům, relačním a nerelačním databázím, map-reduce frameworkům a datovým službám založených na cloudu. Toto je zastřešující projekt, obsahující mnoho dílčích pod-projektů, které jsou specifické pro danou databázi. Projekty jsou vyvíjeny ve spolupráci s mnoha společnostmi a vývojáři, kteří stojí za těmito vzrušujícími technologiemi. [25]

Spring Data je další vhodný mechanismus pro interakci s databázovými entitami, jejich organizaci v úložišti, získávání dat, změnu. V mnoha případech bude stačit prohlásit rozhraní a metodu v něm, bez implementace, kterou technologie připraví sama pomoci reflexe.

Důležité vlastnosti:

- výkonné úložiště a vlastní abstrakce mapování objektů,
- dynamické odvozování dotazů z názvů metod úložiště,
- podpora transparentního auditu (vytvořeno, naposledy změněno),
- možnost integrace vlastního kódu úložiště.

5.1.5 React

React.js je open-source knihovna pro vytváření front-endu v programovacím jazyku JavaScript, která se vyvíjí a je podporovaná společností Facebook a jednotlivými vývojáři. [26]

React se dá použit k vývoji single-page webových a taky mobilních (pomocí React Native) aplikací. Vytvořil ho Jordan Valke – programátor z Facebooku. První edice byla zveřejněna 29. května 2013.

Důležité vlastnosti:

- propojením JavaScriptu a HTML v JSX se docílí snadná pochopitelnost komponenty,
- možnost server-side renderingu,
- virtual DOM.

5.1.6 Ostatní

Pro vytváření unit-testu serverové části aplikace jsem použil Javovskou knihovnu jUnit5 [27] spolu s Mockito [28] pro mockování závislostí, aby se v rámci unit testování nemusel vytvářet Spring kontext.

Protože jsem vytvářel RESTful backend rozehrání, musel jsem implementovat stateless způsob autorizace. Zvolil jsem JWT tokeny, a logiku jejích generace jsem implementoval pomocí knihovny jjwt [29].

Sestavením backendu a řízením závislostí se zabývá nastroj Maven [30], což umožňuje snadno a rychle přidávat nové knihovny a řídit stahování .jar závislostí v souboru pom.xml.

Pro implementaci uživatelského rozhraní jsem použil knihovnu material-ui [31], která se řídí pravidly vyvaření Material Design [32] aplikací od Googlu. Jde o knihovnu pro React.js, která navíc má i své ikony pro využití v rámci webové aplikace.

Pro vytváření a nasazení všech kontejnerů aplikace jsem použil nástroj docker-compose, který je součástí Dockeru [14]. Díky tomu se dá popsat všechny kontejnery a jejích vlastnosti v jednom konfiguračním souboru, a spouštět je pomocí jednoho příkazu, což umožňuje rychle nasazení, konfiguraci, a vývoj.

5.2 Kontejnery

V této sekci popisuji jednotlivé kontejnery a způsob jejích konfigurace pomoci nástroje docker-compose.

```
1  version: '3'
2  services:
3    database:
4      ...
5    backend:
6      ...
7    frontend:
8      ...
9    nginx:
10     ...
11
12 volumes:
13   database-data:
```

Listing 5.1: Struktura souboru docker-compose.yml

Na listingu 5.1 je vidět konfigurační soubor docker-compose.yml, pomoci kterého se vytváří a konfigurují všechny kontejnery aplikace. Na začátku se musí uvést verze formátu compose souboru, jinak bude zvolena zastaralá verze 1 [33].

Pak (řádky 2 až 10) následuje výčet služeb – jednotlivých kontejnerů, které naše aplikace potřebuje. Konfigurace každého kontejneru bude probrána dále, proto je na listingu opuštěna.

Na konci souboru mám definici jednoho volume, který využívá kontejner databáze pro ukládání dat (listing 5.2). Díky tomu se zajišťuje persistence. [34]

5.2.1 Databáze

```
1    database:
2      env_file:
3        - ./database/database.env
4      volumes:
5        - database-data:/var/lib/postgresql/data
6      expose:
7        - "5432"
8      build:
9          context: ./database
```

Listing 5.2: Docker compose – konfigurace databáze

Na listingu 5.2 je podrobně vidět konfigurace kontejneru s databází. Na začátku předávám proměnné prostředí, které jsou definované v souboru *database.env*. Pak obsah *database-data* se mapuje na */var/lib/postgresql/data* v kontejneru, a poskytuje se port pro připojení spojených kontejnerů.

Na konci se nastavuje build context – cesta do adresáře, který obsahuje Dockerfile pro sestavení kontejneru.

```
FROM postgres
ADD init.sql /docker-entrypoint-initdb.d/
```

Listing 5.3: Dockerfile kontejneru databáze

Na listingu 5.3 je důležitý pouze druhý řádek, kde se přenáší soubor *init.sql*, který se nachází v projektu ve složce *database* do složky */docker-entrypoint-initdb.d/* uvnitř kontejneru, díky čemu se při prvním spuštění aplikace provede inicializace databáze pomoci tohoto create-scriptu.

Nastavení defaultní role při vytvoření uživatele je řešeno přímo v databázi pomoci triggeru na listingu 5.4. Spouští se ten po každém insertu do tabulky *expenses_user*, a pokud bude uživatel mít méně, než jednu roli, automaticky mu bude přidělená role „USER“.

```
CREATE FUNCTION user_roles_count (id bigint)
RETURNS BIGINT AS
$$
select count(*)
from expenses_user_role eur
  left join relation_user_role rel on eur.id = rel.role_id
  left join expenses_user eu on eur.id = rel.user_id
where eur.id = $1
$$ LANGUAGE SQL IMMUTABLE;

create or replace function fill_user_role_if_needed()
  returns trigger as
$BODY$
begin
  if (user_roles_count(new.id) < 1)
  then
    insert into relation_user_role(user_id, role_id)
    values (
      new.id,
      (select id from expenses_user_role where name = 'USER')
    );
  end if;
return new;
end;
$BODY$
  LANGUAGE plpgsql VOLATILE;

create trigger fill_user_role_if_needed
  after insert on "expenses_user"
  for each row
  execute procedure fill_user_role_if_needed();
```

Listing 5.4: Databázový trigger, který nastavuje uživatelskou rolí

5.2.2 Backend

```
  backend:
    restart: always
    build: ./expenses-backend
    working_dir: /expenses-backend
```

```
    links:
      - database
    depends_on:
      - database
    volumes:
      - ./expenses-backend:/expenses-backend
      - ./.m2:/root/.m2
    expose:
      - "8080"
    command: mvn clean spring-boot:run
```

Listing 5.5: Docker compose – konfigurace backendu

Podobným způsobem je konfigurován i kontejner backendu – listing 5.5. Rozdíl je v tom, že při každém spuštění se spustí příkaz *mvn clean spring-boot:run*, který provede build a následné spuštění Spring Boot aplikace backendu. Kontejner závisí na databázi, což znamená, že nejdřív se musí spustit ona. Taky má link na databáze, což povoluje aplikaci s ní komunikovat pomoci url *database:5432*.

5.2.3 Frontend

Konfigurace na listingu 5.6 je velmi podobna předchozím. Frontend aplikace závísí na backendu, ale nemá na ten žádný link, proto se jako proměnná prostředí musí předat REACT_APP_API_URL, což je vlastně adresa backendu. V našem případě na adrese *http://localhost* běží nginx reverse-proxy server, který zajistí přesměrování požadavků na */api* do backend kontejneru.

```
frontend:
  build:
    context: ./expenses-frontend
  volumes:
    - './expenses-frontend:/expenses-frontend'
    - '/expenses-frontend/node_modules'
  expose:
    - "3000"
  environment:
    REACT_APP_API_URL: http://localhost/api
  stdin_open: true
  depends_on:
    - backend

nginx:
```

Listing 5.6: Docker compose – konfigurace frontendu

5.2.4 Nginx

```
    volumes:
      - ./nginx.conf:/etc/nginx/nginx.conf
    ports:
      - 80:80
```

```
      - 443:443
    depends_on:
      - frontend
      - backend
    links:
      - frontend
      - backend

volumes:
```

Listing 5.7: Docker compose – konfigurace nginx

Konfigurace tohoto kontejneru je uvedená na listingu 5.7, a je moc jednoduchá. Závisí na frontend a backend kontejnerech, a navíc ma na ně linky. Mapuje porty hostu 80 a 443 (aktuálně není využit) na odpovídající porty uvnitř kontejneru. Tato konfigurace ani nepotřebuje Dockerfile díky řádku *image: nginx*. Nakonfiguruje se ten defaultně sám.

Důležitým je soubor s konfiguraci nginx serveru, který je uveden na listingu 5.8, který kromě všeho obsahuje i lokaci pro websockety, která je na listingu opuštěna. Konfiguruje se v něm to, že server musí poslouchat na portu 80, a to, že všechny požadavky, které dostane s lokací /api přesměruje do backendu, zbývající – do frontendu.

```
events {}

http {
  server {
    listen 80;

    location /sockjs-node {
      ...
    }

    location /api {
      proxy_pass http://backend:8080;
      rewrite ^/api/(.*)$ /$1 break;
    }

    location / {
      proxy_pass http://frontend:3000;
      rewrite ^/(.*)$ /$1 break;
    }
  }
}
```

Listing 5.8: Soubor nginx.conf

5.2.5 Rozdíly vývojové konfigurace

```
version: '3'
services:
  database:
    ...
```

```
5   pgadmin:
6     ...
7   backend:
8     ...
9   frontend:
10    ...
11
12 volumes:
13   database-data:
14   pgadmin:
```

Listing 5.9: Struktura souboru docker-compose.dev.yml

Vývojová konfigurace na listingu 5.9 neobsahuje nginx kontejner, a navíc má konfiguraci pgadmin. Na konci se taky definuje tam pro něj i jmenovaný volume.

```
1     restart: always
2     build: ./expenses-backend
3     working_dir: /expenses-backend
4     links:
5       - database
6     depends_on:
7       - database
8     volumes:
9       - ./expenses-backend:/expenses-backend
10      - ./.m2:/root/.m2
11    ports:
12      - "8080:8080"
13      - "5005:5005"
14    command: mvn clean spring-boot:run -Dspring-boot.run.
    jvmArguments="-Xdebug -Xrunjdwp:transport=dt_socket,server=y,
    suspend=y,address=5005"
```

Listing 5.10: Docker compose pro vývoj – konfigurace backendu

U konfigurace backendu (listing 5.10) teď probíhá mapování portu hostu 8080 na stejný port uvnitř kontejneru, což umožňuje komunikaci s aplikací backendu. Stejným způsobem se mapuje i port 5005, který se používá pro přípojení debuggeru. Změnil se i příkaz pro spuštění, do kterého přibyly argumenty, umožňující debugování.

U konfigurace frontendu místo expose probíhá mapování portu hostu 3000 podobně, jak je to u backendu s portem 8080. Hodnota proměnné prostředí REACT_APP_API_URL je *http://localhost:8080*.

5.3 Backend

Díky Spring Boot není potřeba skoro nic konfigurovat. Většina věcí je pro většinu aplikací stejná, proto je důležité popsat specifické věci, aby Spring mohl ty nakonfigurovat sám. Popisuje se to v souboru *application.properties*, obsah kterého je na listingu 5.11. V tomto případě Spring potřebuje pouze konfigurací datasource a hibernate. Řádky 1 až 7 jsou mnou použité pro logiku generace a validace tokenů.

```
jwt.secret=themostsecretkeyonfitcvut
# 1 hour
jwt.token.access.validity=3600
# 7 days
jwt.token.refresh.validity=604800
jwt.token.access.prefix=access
jwt.token.refresh.prefix=refresh

# db
spring.datasource.url=jdbc:postgresql://database:5432/expenses
spring.datasource.username=admin
spring.datasource.password=1234
spring.datasource.driver-class-name=org.postgresql.Driver

spring.jpa.properties.hibernate.hbm2ddl.auto=none
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.
    PostgreSQLDialect
spring.jpa.show-sql=true
```

Listing 5.11: Soubor application.properies

Přidání autentizace probíhá přímo před její kontrolou filtrem Spring Security, provádí se to v třídě AuthTokenFilter na listingu 5.12.

```
@Component
public class AuthTokenFilter extends GenericFilterBean {
  private final TokenProvider tokenProvider;

  public AuthTokenFilter(@Qualifier("jwtTokenProvider")
    TokenProvider tokenProvider) {
    this.tokenProvider = tokenProvider;
  }

  @Override
  public void doFilter(ServletRequest request, ServletResponse
    response, FilterChain filterChain) throws IOException,
    ServletException {
    String token = tokenProvider.resolveToken((HttpServletRequest
    ) request);
    if (token != null && tokenProvider.validateAccessToken(token)
    ) {
      Authentication authentication = tokenProvider.
    getAuthentication(token);
      SecurityContextHolder.getContext().setAuthentication(
    authentication);
    }
    filterChain.doFilter(request, response);
  }
}
```

Listing 5.12: Filtr zajišťující kontrolu tokenu

Zmíněný filtr zkontroluje token, a přidá autentizaci. Pokud token není, pokračuje se dál a záleží na konfiguraci Srping Security, zda je autentizace

potřebná. V tomto případě je povolen přístup k *auth* endpointům a měnám bez autentizace (listing 5.13).

```
@Override
protected void configure(HttpSecurity http) throws Exception {
  http.csrf().disable()
      .authorizeRequests().mvcMatchers("/auth/sign-in", "/auth/
  register", "/auth/refresh", "/currencies").permitAll()
      .anyRequest().authenticated()
      .and()
      .exceptionHandling().authenticationEntryPoint(
  authenticationEntryPoint)
      .and()
      .sessionManagement().sessionCreationPolicy(
  SessionCreationPolicy.STATELESS);

  http.addFilterBefore(authenticationFilter,
  UsernamePasswordAuthenticationFilter.class);
  http.addFilterBefore(exceptionHandlerFilter, AuthTokenFilter.
  class);
  http.addFilterBefore(corsFilter, ChannelProcessingFilter.
  class);
}
```

Listing 5.13: Konfigurační metoda třídy SecurityConfig.java

Pak DispatcherServlet podle URL rozhoduje, který controller má ten požadavek zpracovat. Při spuštění aplikace díky anotaci *@ComponentScan*, která je součástí *@SpringBootApplication*, Spring vyhledá v aktuálním balíčku a všech potomcích Spring beany, které pozná podle anotací. Díky anotaci *@RequestMapping* se dá ukázat url, kterou controller zpracovává, jak je to vidět na listingu 5.14.

```
@RestController
@RequestMapping("/incomes")
public class IncomeController {
  private final IncomeService incomeService;

  public IncomeController(IncomeService incomeService) {
    this.incomeService = incomeService;
  }

  @PostMapping("/create")
  public IncomeDTO create(@RequestBody IncomeDTO dto) {
    return incomeService.create(dto);
  }
}
```

Listing 5.14: Controller pro příjmy

V tomto případě IncomeService je rozhraní, které implementuje beana GeneralIncomeService, a při inicializaci aplikace Spring provede injection této beany pomoci konstruktoru.

IncomeController umí zpracovat POST požadavek na url */create*, a deleguje toto zpracování servisní třídě (listing 5.15).

```
@Service
public class GeneralIncomeService implements IncomeService {
  private final IncomeRepository incomeRepository;
  private final AccountRepository accountRepository;

  public GeneralIncomeService(IncomeRepository incomeRepository,
                              AccountRepository accountRepository
    ) {
    this.incomeRepository = incomeRepository;
    this.accountRepository = accountRepository;
  }

  ...

  @Override
  @Transactional
  public IncomeDTO create(IncomeDTO dto) {
    Account account = accountRepository.findById(dto.getAccountId
    ()).orElseThrow(
        () -> new IllegalArgumentException("No account for id " +
     dto.getAccountId() + " found.")
    );
    if (!getCurrentUser().getUsername().equals(account.getUser().
    getUsername())) {
      throw new AccessDeniedException("You don't own the account.
    ");
    }
    account.setBalance(account.getBalance().add(dto.getAmount()))
    ;
    return IncomeDTO.from(incomeRepository.save(mapIncome(dto)));
  }

  private Income mapIncome(IncomeDTO dto) {
    Income income = new Income();
    income.setAmount(dto.getAmount());
    income.setComment(dto.getComment());
    income.setTimestamp(dto.getTimestamp());
    income.setAccount(accountRepository.findById(dto.getAccountId
    ()).orElseThrow(
        () -> new IllegalArgumentException("Account not found.")
    ));
    return income;
  }
}
```

Listing 5.15: Servisní třída pro příjmy

Je to vytváření příjmu pro určitý účet. Na začátku se ten vždy vyhledá, a zkontroluje se, zda uživatel, který tuto metodu volá, vlastní tento účet, jinak bude mu přístup zakázán. Pak se zvětší součet peněz na účtu, a uloží se příjem, který bude následně vracen uživateli zpět.

Management transakcí je ve Springu řešen pomocí AOP. Stejná metoda create na listingu 5.15 je označená jako *@Transactional*, což znamená, že kód uvnitř metody bude proveden v rámci jedné transakce, takže pokud selže uložení příjmu, nebudou uloženy i změny, tykající se součtu peněz na účtu.

5.3.1 Testování

Pro testování složitějších částí (ty, které kromě delegování volání metod taky provádí nějaké složitější operace) backendové aplikace jsem vytvořil unit testy.

Na listingu 5.16 je příklad testovací třídy, která se zabývá testováním funkčnosti beany JwtTokenProvider. Práce s mocky uvnitř testovacího kódu je umožněna díky anotaci *@ExtendWith(MockitoExtension.class)*. Na začátku se definuje konstanta TEST_USERNAME, která se pak využívá uvnitř testovacích metod pokud je potřeba pracovat s nějakým uživatelem. Kvůli tomu, že je beana JwtTokenProvider závislá na UserDetailsService, musí se vytvořit mock pro poslední. Testovací metody jsou v tomto listingu opuštěny.

```
@ExtendWith(MockitoExtension.class)
@RunWith(JUnitPlatform.class)
class JwtTokenProviderTest {
  private final String TEST_USERNAME = "TestUsername";

  @Mock
  private UserDetailsService userDetailsService;

  @InjectMocks
  private JwtTokenProvider tokenProvider;

  ...
}
```

Listing 5.16: Testovací třída pro beanu JwtTokenProvider

Na listingu 5.17 je vidět příklad testovací metody *getAuthentication*, která testuje vracení providerem správné autentizace podle tokenu. Je tam i pomocná metoda *prepareUserDetails* společná pro celou třídu.

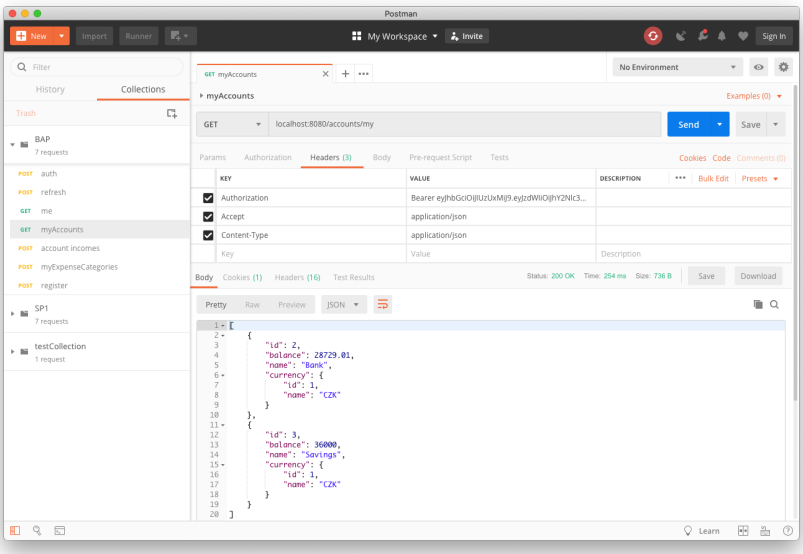
```
...

private UserDetails prepareUserDetails() {
  User user = new User();
  user.setUsername(TEST_USERNAME);
  user.setPassword("");
  user.setRoles(Collections.emptySet());
  return ExpensesUserDetails.from(user);
}

...

@Test
void getAuthentication() {
  setTokenProviderValueFields();
```

```
16    when(userDetailsService.loadUserByUsername(TEST_USERNAME)).
        thenReturn(prepareUserDetails());
17
18    String accessToken = tokenProvider.generateAccessToken(
        prepareUserDetails());
19    Authentication authentication = tokenProvider.getAuthentication
        (accessToken);
20    assertEquals(TEST_USERNAME, ((ExpensesUserDetails)
        authentication.getPrincipal()).getUsername());
21  }
22
23  ...
```

Listing 5.17: Příklad testovací metody třídy JwtTokenProviderTest

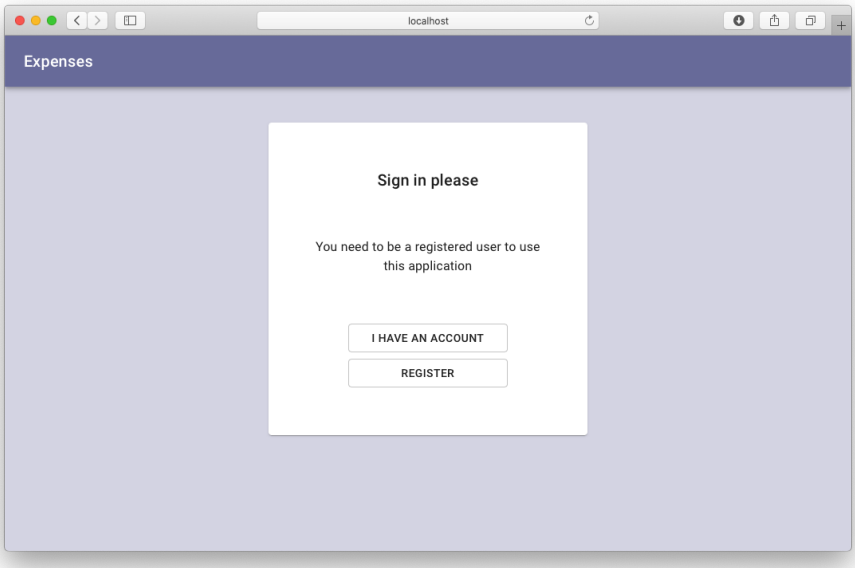
Na začátku testu se zavolá pomocná metoda *setTokenProviderValueFields*, která je kvůli své jednoduchosti opuštěna. Zabývá se nastavováním polí, která by měl na základe konfigurace nastavit Spring. Dělá to pomoci reflexe.

Dál se nastavuje chování mocku beany userDetailsService, která je pak použitá uvnitř provideru. Generuje se token pro testového uživatele, a volá se pak metoda getAuthentication třídy JwtTokenProvider, podle výsledku volání které se kontroluje správnost vracené autentizace.



Obrázek 5.1: Postman [35]

Funkčnost API jsem ověřoval pomocí nástroje Postman [35]. Na obrázku 5.1 je vidět příklad volání endpointu */accounts/my* v dev konfiguraci. Spolu s autorizační hlavičkou posilám požadavek, který vrátí v JSON formátu seznam



Obrázek 5.2: První obrazovka pro nepřihlášeného uživatele

všech účtu uživatele, kterému byl token vydán. Pokud token není validní – bude vracen popis chyby.

5.4 Frontend

Z toho důvodu, že aplikace vyžaduje přihlášení, pokud nebude uživatel přihlášený a zkusí otevřít aplikaci – bude přesměrován do autentizační stránky, kterou je vidět na obrázku 5.2.

Navigace a přesměrování nepřihlášených uživatelů probíhá pomoci knihovny react-router-dom. Na začátku jsem nadefinoval komponentu ProtectedRoute, kterou je vidět na listingu 5.18, která zkontroluje, zda uživatel má tokeny v lokálním úložišti prohlížeče, a na základě toho buď vykreslí příslušnou komponentu, nebo přesměruje uživatele na stránku autentizace.

```
1 const ProtectedRoute = ({ component: Component, ...rest }) => {
2   const isAuthenticated = getTokens() !== null;
3   return <Route
4     {...rest}
5     render={ () => {
6       return isAuthenticated ? <Component /> : <Redirect to="/
    auth" />
7     }}
8   />;
```

```
}
```

Listing 5.18: Komponenta ProtectedRoute

Na listingu 5.19 je vidět hlavní komponentu aplikace, která definuje navigaci. MenuBar je součástí všech stránek, proto je nezávislý na cestě. Všechny nevalidní cesty jsou automaticky přesměrované na hlavní obrazovku, která se vykresluje pomoci popsané výše komponenty ProtectedRoute, což zakazuje vstup nepřihlášeným uživatelům.

```
function App() {
  const [isAuthenticated, setAuthenticated] = useState(getTokens
    () !== null);
  return (
    <div>
      <MenuBar isAuthenticated={isAuthenticated} />
      <Router>
        <Switch>
          <ProtectedRoute path="/" exact component={MainPage} />
          {!isAuthenticated &&
            <Route path="/auth" component={AuthPage} />
          }
          {!isAuthenticated &&
            <Route path="/login" render={props => <LoginPage {...
    props} setAuthenticated={setAuthenticated} />} />
          }
          {!isAuthenticated &&
            <Route path="/register" render={props => <
    RegistrationPage {...props} setAuthenticated={
    setAuthenticated} />} />
          }
          <Redirect from="*" to="/" />
        </Switch>
      </Router>
    </div>
  );
}

export default App;
```

Listing 5.19: Navigace aplikace

Po stisknutí tlačítka „I have an account“, bude uživatel přesměrován do obrazovky přihlášení na obrázku 5.3, kde vyplní svoje přihlašovací údaje a následně bude přesměrován do hlavní obrazovky aplikace. Jenže pokud zadá své údaje špatně, zobrazí se mu dialog s touto informací.

Pokud uživatel rozhodne o registraci, stiskne tlačítko registrace na obrazovce 5.3 a bude přesměrován do obrazovky 5.4, kde vyplní svoje údaje, a pokud uživatelské jméno bude volné – rovnou se provede přihlášení a přesměrování do hlavní obrazovky.

Očividně všechno zmíněné vyžaduje správnou práci s tokeny, jejích umístěni do hlavičky požadavků a včasné obnovení. Tuto práci jsem rozhodl rozdělit

do jednodušších části: unifikovaného odeslání požadavků, a wrapperu, který se bude zabývat prácí s tokeny. Na listingu 5.20 je vidět metodu, která zajišťuje odesílání POST požadavků a vracení promise json odpovědí.

```
const postRequest = async (uri, headers, body) => {
  const promise = new Promise((res, rej) => {
    fetch(uri, {
      method: 'POST',
      headers: {
        'Accept': 'application/json',
        'Content-Type': 'application/json',
        ...headers
      },
      body: JSON.stringify(body)
    }).then(async response => {
      res({
        data: await response.json()
      });
    }).catch(e => {
      res({
        data: {
          error: e
        }
      });
    });
  });
  return promise;
};
```

Listing 5.20: Metoda pro odesláni POST požadavků

Zmíněnou metodu pak použije wrapper na listingu 5.21, který předá ji autorizační hlavičku, ale předtím v případě potřeby provede obnovení tokenů.

```
const postRequest = async (endpoint, body) => {
  return await refreshTokensIfNeededBeforeExecution(() => {
    return RestClient.postRequest(API_URL + endpoint,
    getAuthHeader(), body);
  });
};
```

Listing 5.21: Metoda-wrapper pro odesláni POST požadavků

Na obrázku 5.5 je vidět hlavní obrazovku. Vlevo má seznamy účtů a kategorií výdajů s možností vytvářeni stisknutím tlačítka „+“. Vpravo je součet všech peněz na účtech, celkové měsíční výdaje, a plánovaná částka výdajů s tlačítkem, stisknutím kterého se otevře dialog nastavení měsíční limity, jak je vidět na obrázku 5.6. Vpravo dole je diagram výdajů podle kategorií, a když příslušný sektor diagramu bude pod kurzorem myši, objeví se nápověda s její názvem.

Na obrázku 5.7 je vidět dialog vytváření nového účtu, kam je potřeba vyplnit údaje, tykající se účtu. Až uživatel bude mít alespoň jeden vytvořený účet, bude mít možnost zobrazit jeho detail (obrázek 5.8) stisknutím příslušné

dlaždice, která ho reprezentuje. Dialog detailu účtu nejen zobrazuje pohyby a nezbytné údaje, charakterizující účet, ale i povoluje přechod do stavu editace (obrázek 5.9), vytváření převodů a příjmů pomocí odpovídajících dialogů na obrázcích 5.10 a 5.11.

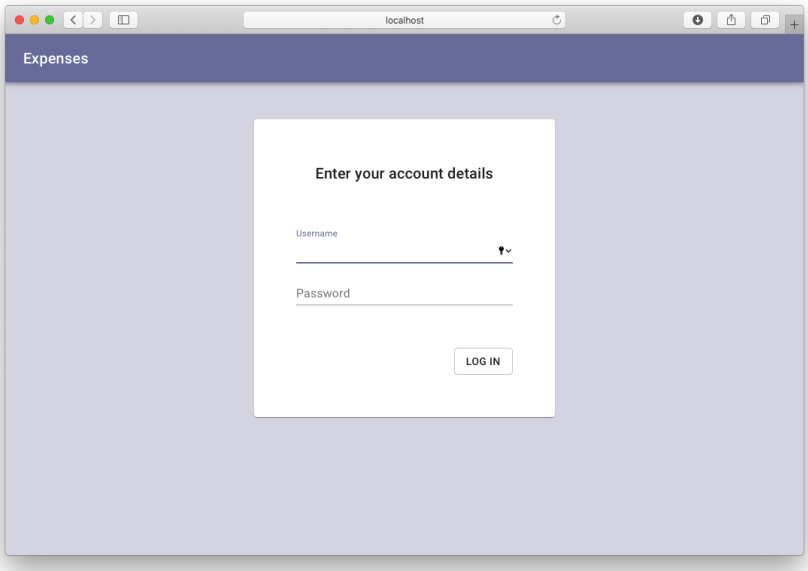
Dialog vytváření kategorie výdajů (obrázek 5.12) je velmi podobný takovému u účtu, jen místo kreditu povoluje vyplnit volitelnou limitu. Stejně když uživatel už nějakou kategorii vytvořenou má, může zobrazit její detail (obrázek 5.13), kde jsou vidět všechny výdaje, které do této kategorie patří. Je taky tlačítko editace, stisknutím kterého se dialog překresli do tvaru na obrázku 5.14, čímž povolí editaci kategorie. Stisknutím tlačítka „Add transaction“ se otevře dialog na obrázku 5.15, pomoci kterého se po vyplnění všech údajů a potvrzení vytvoří nový výdaj. Pokud po přidaní transakce bude překročena limita kategorie – bude uživateli zobrazeno varování na obrázku 5.16 a transakce bude vytvořena pouze, pokud se stiskne tlačítko „OK“.

5.5 Mobilní webová aplikace

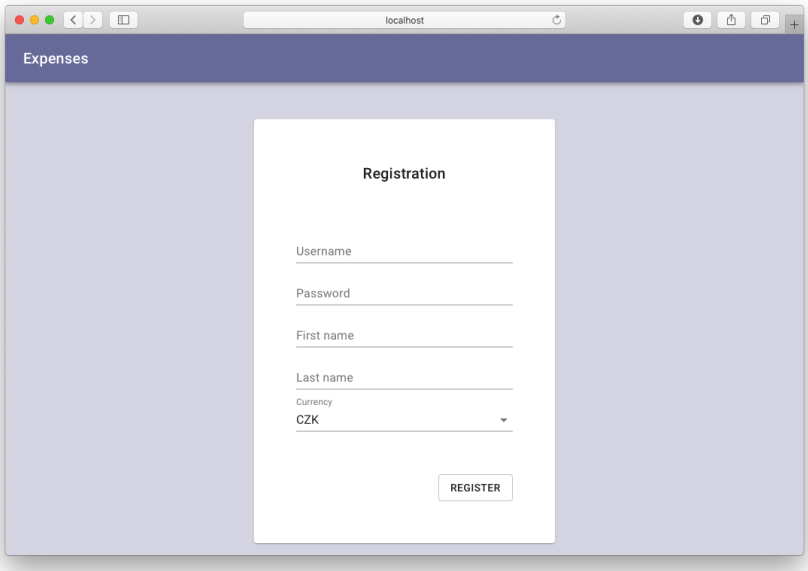
Pro demonstraci rozšiřitelnosti jsem vytvořil jednoduchou read only mobilní webovou aplikaci, která povoluje uživateli se přihlásit a uvidět svoje účty a součet peněz, které na nich má. Obě dvě obrazovky jsou vidět na obrázcích 5.17 a 5.18. Je to nezávislá od frontendu aplikace, která se ani nenachází v kontejneru. Ví pouze to, že na adrese *localhost/api* může komunikovat se serverem, který zpracuje všechny její požadavky. Je přístupna na adrese *localhost:3000*.

5.6 Dokumentace

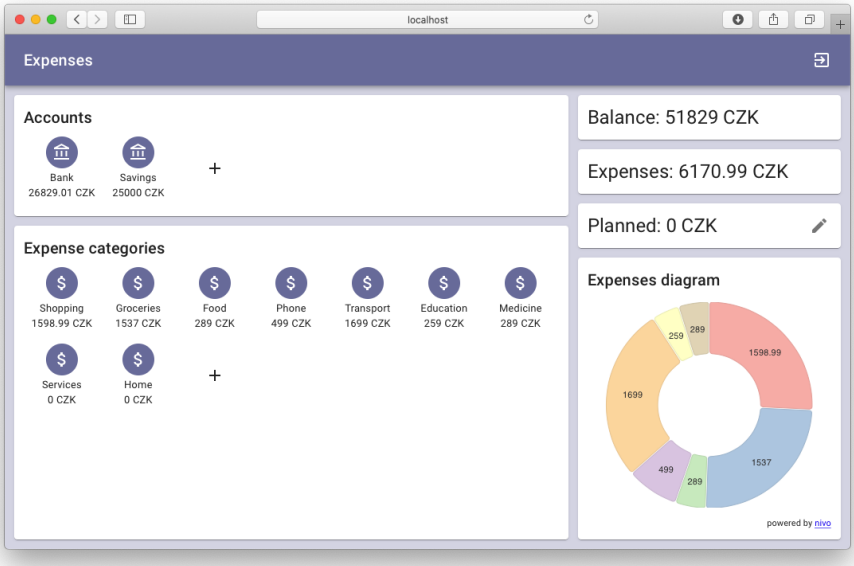
V sekci „Frontend“ jsem dostatečně popsal funkčnosti aplikace, a jak se má používat. Ale kromě toho je taky potřeba zdokumentovat postup pro instalaci a rozhraní backendu. Oba dva pdf dokumenty jsou vytvořené z markdown souborů, jsou umístěny ve složce */docs*, a jsou součástí příloh této práce. Soubor *manual.pdf* popisuje požadavky, instrukce pro spuštění a lokaci jednotlivých částí. Soubor *api.pdf* popisuje služby backendu a způsob, jak se k nim má přistupovat.



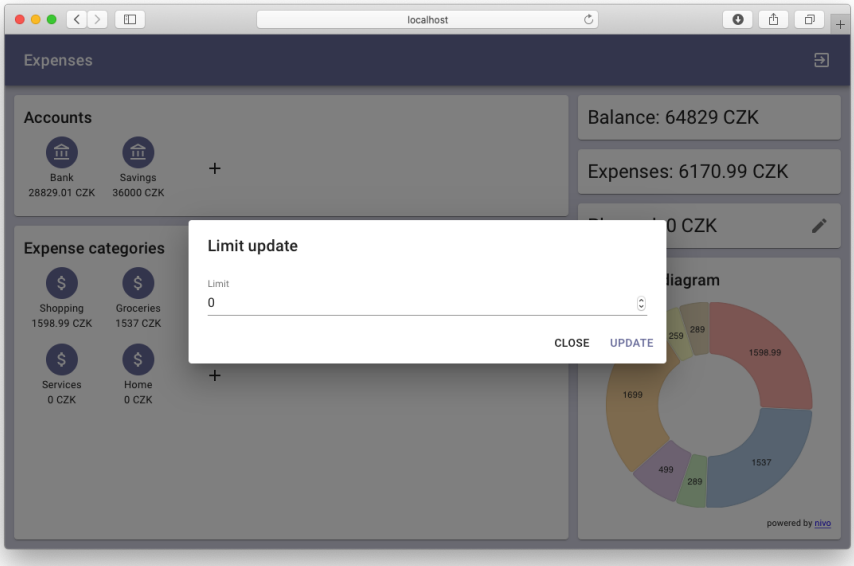
Obrázek 5.3: Obrazovka přihlášení



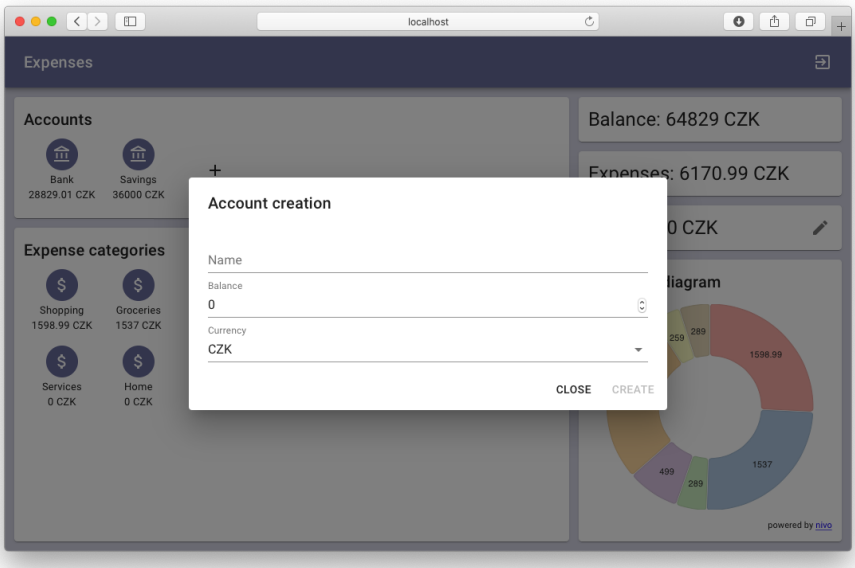
Obrázek 5.4: Obrazovka registrace



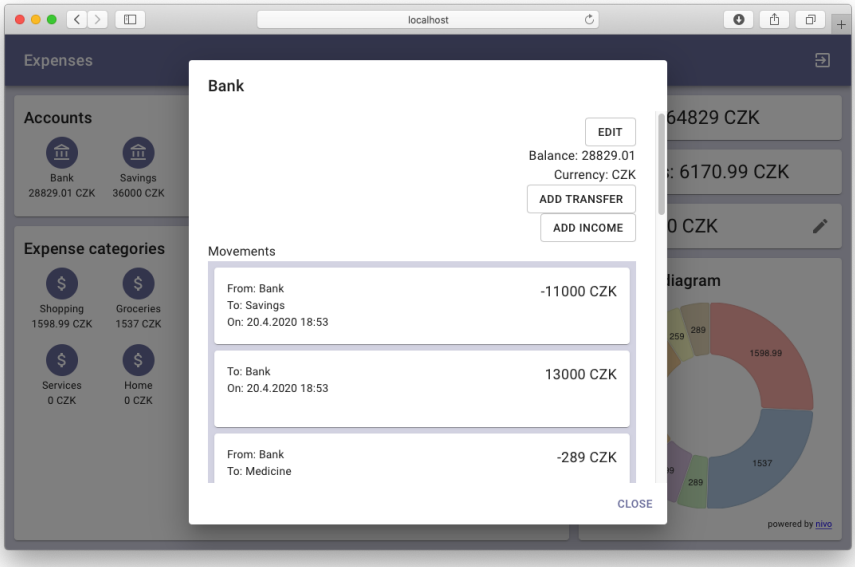
Obrázek 5.5: Hlavní obrazovka



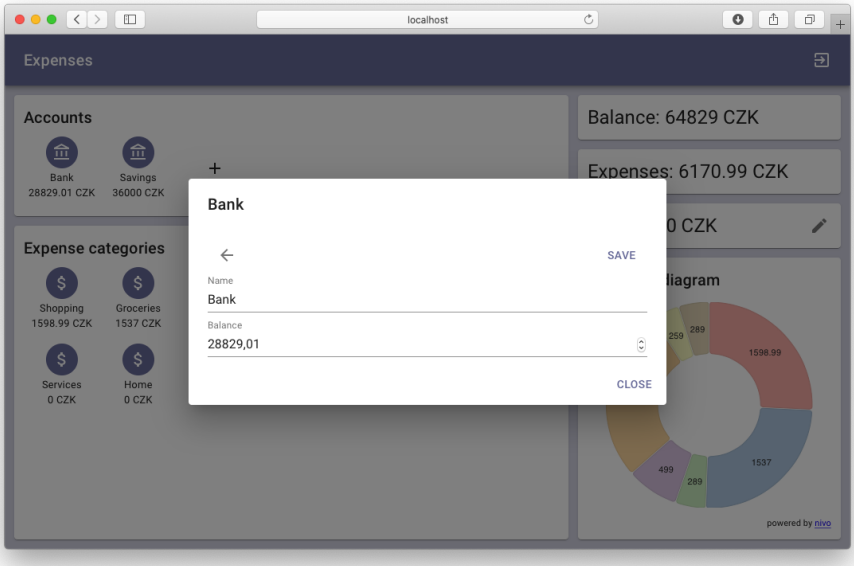
Obrázek 5.6: Dialog nastavení měsíční limity



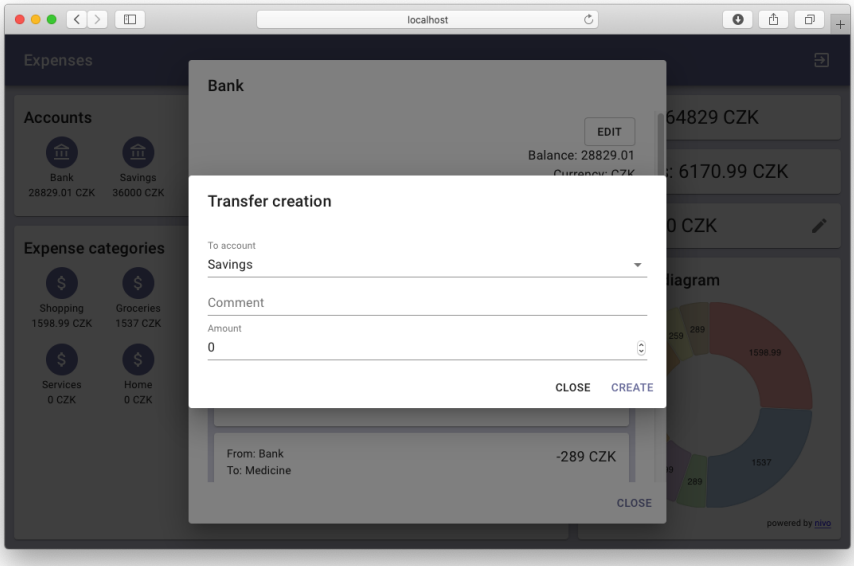
Obrázek 5.7: Vytváření účtu



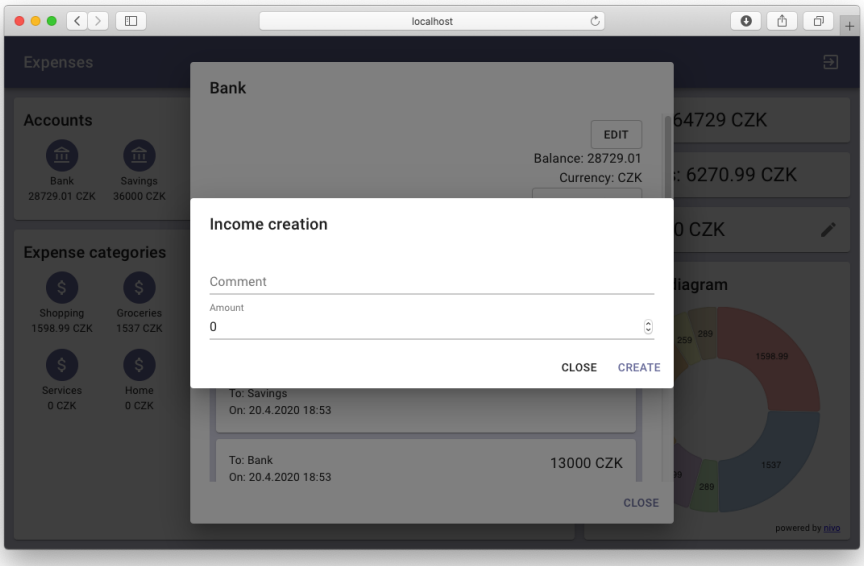
Obrázek 5.8: Detail účtu



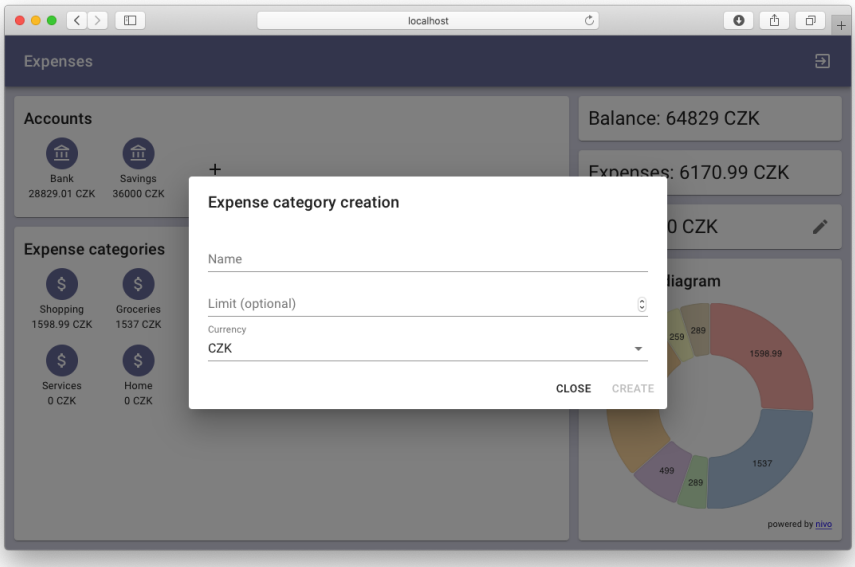
Obrázek 5.9: Editace účtu



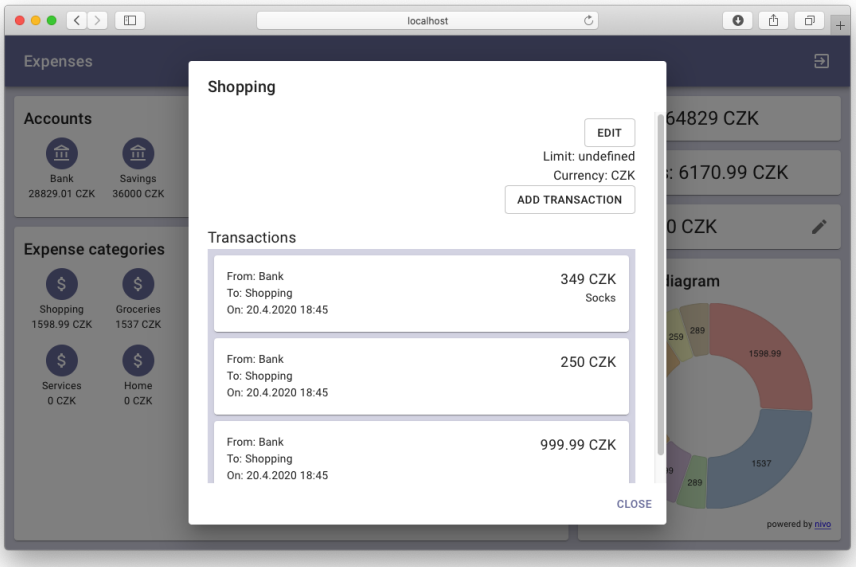
Obrázek 5.10: Vytváření převodu mezi účty



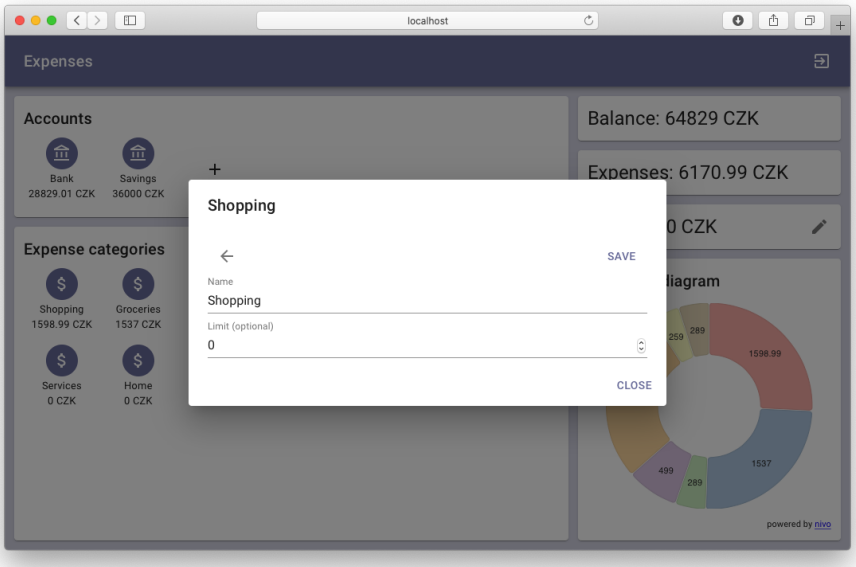
Obrázek 5.11: Vytváření příjmu



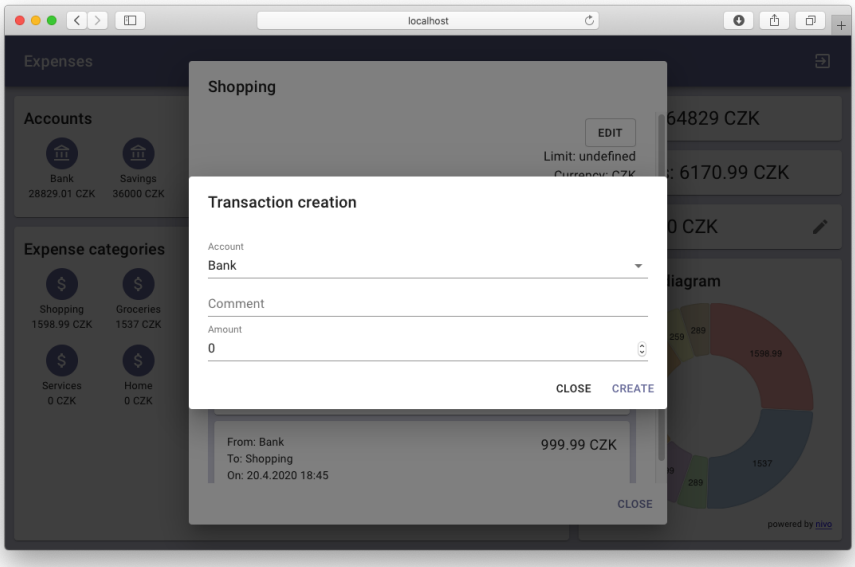
Obrázek 5.12: Vytváření kategorie výdajů



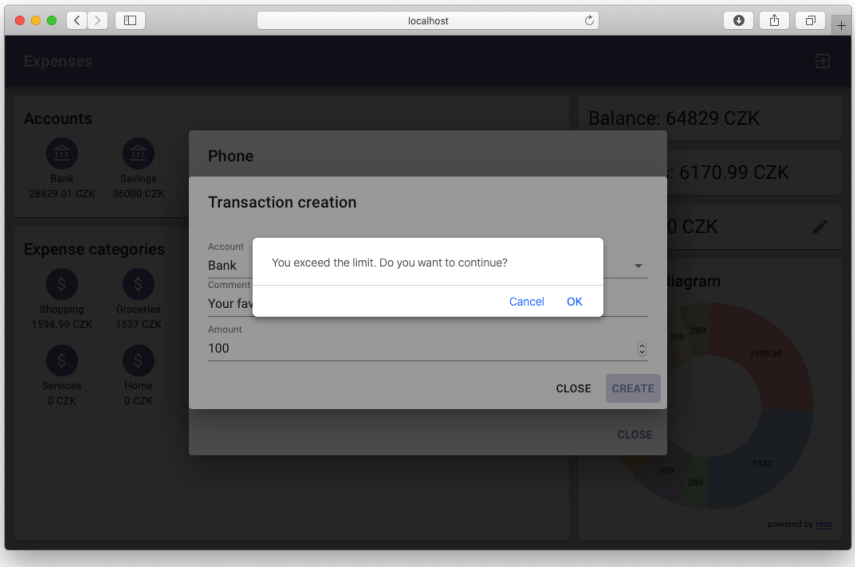
Obrázek 5.13: Detail kategorie výdajů



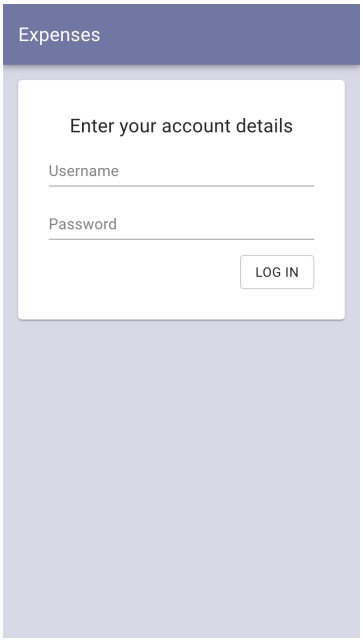
Obrázek 5.14: Editace kategorie výdajů



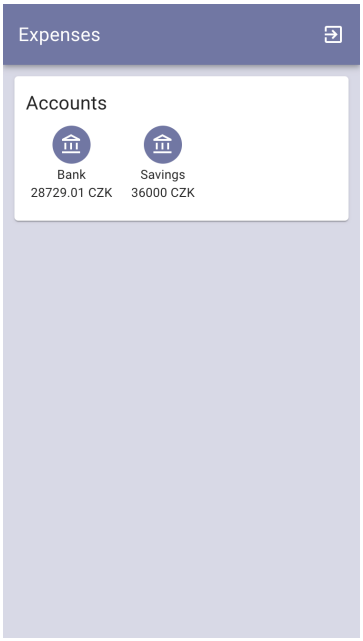
Obrázek 5.15: Vytváření transakce



Obrázek 5.16: Upozornění o překročení limity kategorie



Obrázek 5.17: Mobilní aplikace: Obrazovka přihlášení



Obrázek 5.18: Mobilní aplikace: Hlavní obrazovka

Kapitola 6

Zhodnocení

6.1 Zhodnocení aplikace

Mezi největší klady moje aplikace patři snadná rozšiřitelnost a unifikace vývoje pro různé platformy. Není nutné se zabývat návrhem a vývojem aplikací pro iOS a Andorid zvlášť. Zabírá to hodně času a peněz, proto ne každá aplikace takového druhu podporuje všechny platformy.

Důležitou vlastností je taky snadné nasazování a spolehlivost dodávek. Pokud dodávka v pohodě funguje na počítači vývojáře, bude fungovat i na produkčním virtuálním stroji. V případě, že by se potřebovalo škálovaní v okamžik, kdy se bude tato aplikace nejvíce používat, da se ji skoro bez žádných nákladů přenést do cloudu, a nastavit horizontální škálovaní frontend a backend kontejnerů.

6.2 Možná zlepšení

Teoreticky může nastat případ, kdy nebude stačit škálovaní frontendu a backendu, bude potřeba škálovat databázi. Bohužel horizontální škálovaní databáze PostgreSQL je náročné, když vertikální je docela drahé. Proto by bylo jako možné zlepšení dobré migrovat data do nějaké dokumentové databáze, která umožní poměrně levné horizontální škálovaní.

Jedním z prvních zlepšení podle mého názoru musí být offline režim, což umožní udělat progresivní webovou aplikaci s možností instalace na mobil nebo počítač. Navíc aplikace v tomto režimu by měla ukládat všechna data na zařízení a provádět pravidelnou synchronizaci. Umožní to pohodlné použití i v místech s horším internetovým přípojením.

Aplikace aktuálně má jenom jeden diagram na hlavní obrazovce, který zobrazuje poměr různých kategorií výdajů. Bylo by vhodné zvětšit množství diagramů pro účely analýzy nejen výdajů, ale i příjmů, převodů peněz mezi účty nebo taky přidat diagramy, rozšiřující porozumění osobních financí uživatelem.

Taky by bylo vhodné povolit uživatelům volit ikony účtů a kategorií výdajů. Dalším zlepšením by mohla být možnost volby jazyka aplikace.

6.3 Srovnání s existujícími řešeními

V této sekci se zabývám srovnáním výsledku moje práce s existujícími řešeními, které jsem probíral v kapitole 3. Při realizaci své aplikace jsem se snažil u nich převzít ty nejlepší funkčnosti a vyhnout se nevhodným podle mého názoru věcem, které jenom komplikují použití.

6.3.1 Monefy

U aplikace Monefy jsem se inspiroval především diagramem výdajů, což umožňuje snadnou jejich kontrolu. Na rozdíl od této implementace, není diagram v moji omezen počtem kategorií – nemůže nastat situace, kdy všechny kategorie nevejdou do místa kolem diagramu.

Vyhnul jsem se nejasnosti v tlačítkách „+“ a „–“. Taky zobrazení seznamů účtů a kategorií výdajů na hlavní obrazovce je podle mě mnohém názornější, než je to řešeno u Monefy.

6.3.2 Wallet

Na rozdíl od aplikace Wallet, moje implementace má diagram výdajů na hlavní obrazovce, což redukuje počet kroků pro přístup k této důležité funkčnosti.

U této aplikace je důraz hlavní obrazovky kladěn na jednotlivé výdaje, součty a změnu zůstatků na účtech. Moje implementace na hlavní obrazovce najednou zobrazuje více, než jeden účet, udává přehled o osobních výdajích dle kategorií pomoci diagramů, a taky ikon s informacemi v seznamu, díky čemu podle mě umožňuje snadnější plánování osobních výdajů.

6.3.3 CoinKeeper

U aplikace CoinKeeper jsem převzal myšlenku zobrazení seznamů účtů a kategorií výdajů na hlavní obrazovce. Místo nevhodné podle mě analýzy pomoci cashflow, která je použitá v této implementaci, moje aplikace disponuje diagramem výdajů podle kategorií, což dělá jejich management názornější.

Taky důležité pro mě je to, že moje implementace má webovou verzi pro počítač.

Závěr

Cílem práce bylo na základě analýzy navrhnout a implementovat webovou aplikaci, která umožní uživatelům tvořit a spravovat přehledy o osobních výdajích v různých kategoriích. Taky bylo nutné vytvořit doménový konceptuální model, a sestavit funkční a nefunkční požadavky. Aplikace musela podporovat rozšiřitelnost a měla být otestovaná.

Analyzoval jsem problém spravování osobních financí, vytvořil jsem entitně-relační diagram, který popisuje doménu, a sestavil jsem požadavky na aplikaci. Program díky RESTful rozhraní podporuje snadnou rozšiřitelnost, což jsem ukázal vytvořením jednoduché read only mobilní webové verze aplikace. Pro účely testování jsem vytvářel unit testy složitějších částí kódu, a testoval jsem RESTful API pomoci nástroje Postman.

Na rozdíl od analyzovaných aplikací Monefy a CoinKeeper, moje realizace poskytuje uživatelům možnost spravovat své výdaje na počítači. Ale díky tomu, že se jedná o webovou aplikaci, taky díky existenci progresivních webových aplikací, a zvolené architektuře, umožňuje se rychlý návrh a vývoj univerzální aplikace pro mobily. V porovnání s aplikací Wallet, moje implementace poskytuje přehlednější analýzu výdajů a navíc má jednodušší uživatelské rozhraní.

Mezi přínosy moje aplikace taky patří to, že nevyžaduje instalaci, a díky tomu se umožňuje jednodušší aktualizace verzí bez závislosti na platformě. Díky kontejnerové architektuře se umožňuje snadná rozšiřitelnost, jednodušší nasazení, delší doba přístupnosti a větší výkonnost (díky jednoduššímu škálovaní).

Dalším rozšířením by mohla být implementace nativní mobilní aplikace pro iOS a Android, offline řežím, nebo přesun informace o platbách do MongoDB.

Bibliografie

1. SMRČKA, Luboš. *Osobní a rodinné finance: (svět rodinných financí - jak spořit a rozmnožovat majetek)*. 1. vyd. Praha: Professional Publishing, 2007. ISBN 978-80-86946-41-2.
2. MASLOW, Abraham Harold. *O psychologii bytí*. 1. vyd. Praha: Portál (vydavatelství), 2014. ISBN 978-80-262-0618-7.
3. AIMBITY AS. *Monefy - Handy personal finance management tool for Android and iOS, verze 1.3* [soft.]. 2020. Dostupné také z: `http://www.monefy.me/`. [přístup 28. března 2020].
4. BUDGETBAKERS S.R.O. *Wallet - Daily Budget & Profit, verze 2.10.4* [soft.]. 2016. Dostupné také z: `https://apps.apple.com/cz/app/wallet-daily-budget-profit/id1032467659`. [přístup 28. března 2020].
5. DIZRAPP OOO. *CoinKeeper: budget planner, verze 2.1.15* [soft.]. 2018. Dostupné také z: `https://apps.apple.com/cz/app/coinkeeper-budget-planner/id849747345`. [přístup 28. března 2020].
6. ROEM.RU. V Rusku zůstává Excel hlavním hráčem na tomto trhu (překlad autora). *Roem.ru* [online]. 2015 [cit. 2020-03-28]. Dostupné také z: `https://roem.ru/16-03-2015/188255/pfm-ru-2015/`.
7. NUSEIBEH, Bashar; EASTERBROOK, Steve. Requirements Engineering: A Roadmap. In: *Proceedings of the Conference on The Future of Software Engineering*. Limerick, Ireland: Association for Computing Machinery, 2000, s. 35–46. ICSE '00. ISBN 1581132530. Dostupné z DOI: `10.1145/336512.336523`.
8. ZAVE, Pamela. Classification of Research Efforts in Requirements Engineering. *ACM Comput. Surv.* 1997, roč. 29, s. 315–321. Dostupné z DOI: `10.1109/ISRE.1995.512563`.

9. PM CONSULTING. Funkční požadavky. *PM Consulting* [online]. 2020 [cit. 2020-04-05]. Dostupné také z: https://www.pmconsulting.cz/slovnikovy-pojem/funkcni-pozadavky/.

10. PM CONSULTING. Nefunkční požadavky. *PM Consulting* [online]. 2020 [cit. 2020-04-05]. Dostupné také z: https://www.pmconsulting.cz/slovnikovy-pojem/nefunkcni-pozadavky/.

11. NETAPP. What Are Containers? *NetApp Knowledge Center* [online]. 2020 [cit. 2020-04-30]. Dostupné také z: https://www.netapp.com/us/info/what-are-containers.aspx.

12. PABLO IORIO. Container based Architectures I/III: Technical advantages. *Medium.com* [online]. 2017 [cit. 2020-04-30]. Dostupné také z: https://medium.com/@pablo.iorio/container-based-architecture-i-iii-technical-advantages-7176195456c5.

13. LARRY LUDLOW. Containers: Pros, Cons and How to Mitigate Risk. *Container Journal* [online]. 2020 [cit. 2020-04-30]. Dostupné také z: https://containerjournal.com/topics/container-ecosystems/containers-pros-cons-and-how-to-mitigate-risk/.

14. DOCKER, INC. *Docker* [soft.]. 2013. Dostupné také z: https://www.docker.com/. [přístup 30. dubna 2020].

15. IBM CLOUD EDUCATION. Docker. *IBM* [online]. 2020 [cit. 2020-04-30]. Dostupné také z: https://www.ibm.com/cloud/learn/docker#toc-why-use-do-OyDVNwwD.

16. DOCKER. Developers bring their ideas to life with Docker. *Docker* [online]. 2020 [cit. 2020-04-30]. Dostupné také z: https://www.docker.com/why-docker.

17. FIELDING, Roy Thomas. *REST: Architectural Styles and the Design of Network-based Software Architectures*. 2000. Dostupné také z: http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm. Doctoral dissertation. University of California, Irvine.

18. IGOR SYSOEV, NGINX, INC. *Nginx* [soft.]. 2004. Dostupné také z: https://www.nginx.com/. [přístup 30. dubna 2020].

19. NGINX. What Is a Reverse Proxy Server? *NGINX Glossary* [online]. 2020 [cit. 2020-04-30]. Dostupné také z: https://www.nginx.com/resources/glossary/reverse-proxy-server/.

20. THE PGADMIN DEVELOPMENT TEAM. *pgAdmin* [soft.]. 2002. Dostupné také z: https://www.pgadmin.org/. [přístup 30. dubna 2020].

21. POSTGRESQL GLOBAL DEVELOPMENT GROUP. *PostgreSQL* [soft.]. 1996. Dostupné také z: https://www.postgresql.org/. [přístup 30. dubna 2020].

22. PIVOTAL SOFTWARE. Spring Framework Overview. *Spring Docs* [online]. 2020 [cit. 2020-05-03]. Dostupné také z: `https://docs.spring.io/spring/docs/current/spring-framework-reference/overview.html`.

23. PIVOTAL SOFTWARE. Spring Boot. *Spring Projects* [online]. 2020 [cit. 2020-05-03]. Dostupné také z: `https://spring.io/projects/spring-boot`.

24. PIVOTAL SOFTWARE. Spring Security. *Spring Projects* [online]. 2020 [cit. 2020-05-03]. Dostupné také z: `https://spring.io/projects/spring-security`.

25. PIVOTAL SOFTWARE. Spring Data. *Spring Projects* [online]. 2020 [cit. 2020-05-03]. Dostupné také z: `https://spring.io/projects/spring-data`.

26. JORDAN VALKE. *React* [soft.]. 2013. Dostupné také z: `https://reactjs.org`. [přístup 3. května 2020].

27. THE JUNIT TEAM. JUnit 5. *JUnit Pages* [online]. 2020 [cit. 2020-05-03]. Dostupné také z: `https://junit.org/junit5/`.

28. SZCZEPAN FABER AND FRIENDS. Mockito. *Mockito site* [online]. 2020 [cit. 2020-05-03]. Dostupné také z: `https://site.mockito.org`.

29. JWTK. Java JWT: JSON Web Token for Java and Android. *GitHub* [online]. 2020 [cit. 2020-05-03]. Dostupné také z: `https://github.com/jwtk/jjwt`.

30. APACHE SOFTWARE FOUNDATION. *Maven* [soft.]. 2004. Dostupné také z: `https://maven.apache.org`. [přístup 3. května 2020].

31. MATERIAL-UI. Material-UI. *material-ui* [online]. 2014 [cit. 2020-05-03]. Dostupné také z: `https://material-ui.com`.

32. GOOGLE. Material Design. *material.io* [online]. 2014 [cit. 2020-05-03]. Dostupné také z: `https://material.io/design`.

33. DOCKER. Compose file versions and upgrading. *docker docs* [online]. 2020 [cit. 2020-05-19]. Dostupné také z: `https://docs.docker.com/compose/compose-file/compose-versioning/`.

34. DOCKER. Use volumes. *docker docs* [online]. 2020 [cit. 2020-05-19]. Dostupné také z: `https://docs.docker.com/storage/volumes/`.

35. ABHINAV ASTHANA. *Postman* [soft.]. 2012. Dostupné také z: `https://www.postman.com`. [přístup 3. května 2020].

Příloha A

Seznam použitých zkratek

AOP Aspect Oriented Programming

API Application Programming Interface

GUI Graphical User Interface

JJWT Java JWT

JSON JavaScript Object Notation

JWT JSON Web Token

SSL Secure Sockets Layer

HTTP HyperText Transfer Protocol

REST Representational State Transfer

RE Requirements Engineering

UC Use Case

URL Uniform Resource Locator

Příloha **B**

Obsah přiložené SD karty

- `readme.txt` ..... stručný popis obsahu SD karty
- `docs` ..... adresář s dokumentací
  - `api.pdf` ..... dokumentace rozhraní backendu
  - `manual.pdf` ..... dokumentace spuštění aplikace
- `src`
  - `impl` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text`
  - `thesis.pdf` ..... text práce ve formátu PDF