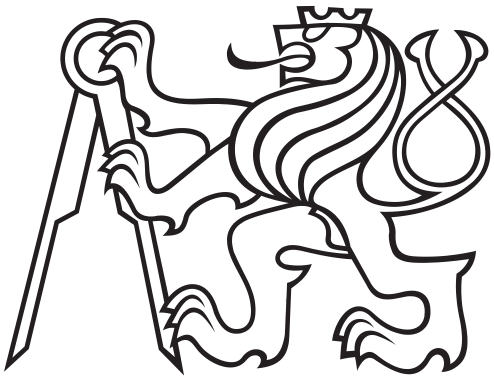
České vysoké učení technické v Praze

Fakulta elektrotechnická
Katedra počítačů



Bakalářská práce

Aplikace pro správu osobních financí

Algis Skriabin

Školitel: Ing. Božena Mannová, Ph.D.

Obor: Softwarové inženýrství a technologie

Leden 2022

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Skriabin** Jméno: **Algis** Osobní číslo: **487011**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Softwarové inženýrství a technologie**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Aplikace pro správu osobních financí

Název bakalářské práce anglicky:

Personal Financial Management Application

Pokyny pro vypracování:

Seznamte se s problematikou osobního finančního řízení a dostupných aplikací, které lze pro tyto činnosti použít. Seznamte se s aplikacemi, které nabízejí banky. Na základě vyhodnocení získaných dat navrhněte a implementujte webovou aplikaci pro osobní finanční řízení. Navrhněte přátelské uživatelské rozhraní. Zaměřte se také na bezpečnost navržené aplikace. Vyberte vhodné prostředky k implementaci, nasazení a testování aplikace. Vyhodnoťte výsledky a navrhněte další funkce nebo další vylepšení.

Seznam doporučené literatury:

[1] Pressmann R. S.: Software Enegineering
[2]
https://www.cnbc.com/guide/personal-finance-101-the-complete-guide-to-managing-your-money/
[3] https://www.thebalance.com/best-personal-finance-software-4171938

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Božena Mannová, Ph.D., kabinet výuky informatiky FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **22.06.2021** Termín odevzdání bakalářské práce: **04.01.2022**

Platnost zadání bakalářské práce: **19.02.2023**

Ing. Božena Mannová, Ph.D.	podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D.
podpis vedoucí(ho) práce		podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

. | Podpis studenta

Datum převzetí zadání

Poděkování

Děkuji vedoucímu Ing. Božena Mannová, Ph.D. za cenné rady a vedení při práci. Také chci poděkovat své rodině a přátelům za podporu.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze, ...

Abstrakt

Cílem práce je navrhnout a implementovat webovou aplikaci, která umožní uživatelům spravovat jejich finance. Nejprve je provedena analýza podobných aplikací s důrazem na jejich výhody a nevýhody. Následující kapitoly jsou věnovány návrhu, implementaci a testování aplikace. Na závěr jsou vyhodnocené výsledky a navržená možná další vylepšení.

Klíčová slova: webová aplikace, finance, správa osobních financí, rozpočet, příjmy, výdaje

Abstract

The aim of the thesis is to design and implement a web application that allows users to manage their finances. First, an analysis of similar applications is performed with emphasis on their advantages and disadvantages. The following chapters are devoted to the design, implementation and tests of the application. At the very end, the results are evaluated and possible further improvements suggested.

Keywords: web application, finance, personal finance management, budget, income, expenses

Obsah

Seznam obrázků

Seznam tabulek

Seznam ukázek kódu

Úvod

Správa osobních financí není na většině středních a vysokých škol vyučována, i když se jedná o velmi užitečnou dovednost. Prostřednictvím pokusů a omylů, správných a nesprávných kroků si každý člověk postupně vyvíjí svůj vlastní finanční algoritmus - někdo začne více utrácet někdo zase šetřit. Podle průzkumu České Bankovní Asociace[1](ČBA) index finanční gramotnosti po letech růstu poklesl na 55 bodů. Průzkum, který pro ČBA realizovala výzkumná agentura Ipsos[2], totiž ukázal, že v porovnání s dobou před pandemií se Češi hůře orientují ve finančních záležitostech, více se zajímají pouze o věci a informace týkající se jejich osoby a situace, a celou řadu informací o finančních produktech považují za zbytečné. Pandemická situace se samozřejmě promítla do uvažovaní nad financemi většiny obyvatel. Češi jsou nyní opatrnější a více uvažují nad každou utracenou korunou. Podle průzkumu chtějí více spořit, plánovat a sledovat své výdaje nebo o penězích jen více přemýšlet, a to zejména mladší generace[1]. Právě ke správnému hospodaření s osobními financemi bude pomáhat webová aplikace MoneyExpert.

Výhodou webové aplikace je to, že není závislá na platformě, vyžaduje jen webový prohlížeč. Zároveň vývoj takové aplikace není nákladný, protože lze například snížit náklady na podpoře a údržbě. Data jsou uchovávána na serveru a jsou přístupná odkudkoliv. Nevýhodou je ale to, že k fungování vyžaduje připojení k internetu, a je zde možné bezpečnostní riziko úniku dat.

Lidé se nerodí jako experti na osobní finance, proto je není možné vinit za občasné nerozvážné nákupy. V důsledku toho je ale snadné ztratit přehled o svých výdajích a mít ke konci měsíce na účtu nedostatek peněz do výplaty. Ale například pomocí plánování rozpočtu lze upřednostňovat potřebné výdaje před nepodstatnými, lépe hospodařit se svými financemi nebo budovat prospěšné návyky. Umožňuje nám tedy dosahovat našich dlouhodobých finančních cílů.

Popsaný problém již řeší řada aplikací; buď jsou však pro běžného uživatele příliš komplikované, vizuálně zastaralé nebo nemají užitečné funkce. Samozřejmě díky široké nabídce produktů se dají najít takové, které jsou dobře vyvinuté a vyhovují většině uživatelů. Od těchto aplikací se pokusím vzít jejich pozitiva a vytvořit pro uživatele přehlednou a užitečnou aplikaci.

Následující kapitoly popisují cíl práce a analyzují podobné aplikace. Další kapitoly jsou věnovány návrhu, implementaci a testování aplikace. Závěrem práce je vyhodnocení výsledků a návrh na další možná vylepšení.

[1] https://cbaonline.cz/
[2] https://www.ipsos.com/cs-cz

Kapitola 1

Cíl práce

Cílem práce je vytvořit přehlednou, jednoduchou a efektivní webovou aplikaci, jejíž cílem je pomoci uživateli pro lepší spravování osobních financí. Cílová skupina jsou všichni jedinci, kteří si chtějí vést záznam o svých financích bez toho, aby se museli připojovat ke svým bankovním účtům. Například lidé ve věku 17–25 let, kteří si přejí mít své finance pod kontrolou a upřednostňují jednoduché minimalistické uživatelské rozhraní před pokročilou funkčností, zároveň třeba pracují a mají stálý měsíční příjem.

Aplikace **MoneyExpert** by měla uživateli přinést jasné informace o přehledu transakcí, je pro mě důležité nezatěžovat uživatele dalšími informacemi a nepřehledným nastavením jako je to často u konkurenčních systémů. Budu se snažit o maximální pohodlí, tak aby ovládání bylo intuitivní.

Aplikace bude uživateli umožňovat vést své finance, kolik a za co utratil, také přiřazovat kategorie ke transakcím, dále vykreslování grafů pro lepší přehled finančního stavu. Také bude možné zakládat účty i v jiných měnách(EUR, CZK), a také bude možnost sledování osobních závazků, plánovaní rozpočtů, a sdílení účtu s dalšími uživateli.

Nejprve bude provedena analýza podobných aplikací, jejich funkcí a jejich nedostatků, na základě které budou určeny funkční i nefunkční požadavky aplikace. Na základě požadavků se určí případy užíti a domain model. Po provedení analýzy se přistoupí k návrhu, kde bude popsaná architektura a vytvořená jasná struktura pro vývoj aplikace. Následně bude provedená samotná implementace. To bude demonstrováno funkční aplikací, jejíž kód bude zveřejněn, a vybranými okomentovanými ukázkami kódu. Pak se přistoupí k testování, které bude provedeno pomocí jednotkových, systémových a také uživatelských testů. Testeři budou postupovat podle testovacího scénáře, který pokrývá obsah funkce aplikace.

Kapitola 2

Analýza

V teto kapitole si popišme již existující řešení, totiž na podobné aplikace pro správu osobních financí. Analýza se zaměřuje na zdokumentování hlavních funkcí aplikace. Pak si zvolíme technologie, které k tvorbě aplikaci použijeme a určíme požadavky.

Rozebíraná řešení jsem volil ze seznamu doporučených finančních aplikací na internetovém článku: Nejlepší finanční aplikace 2021: Appky, které šetří nebo spoří peníze![2]

U vybraných aplikací jsem sledoval následující charakteristiky:

- Cílové platformy
- Kategorizace transakcí
- Přehlednost aplikace
- Statistická data a vykreslování grafů k tomu
- Stanovení rozpočtů
- Možnost vést více účtů v různých měnách
- Připojení s bankou

2.0.1 eÚčty.cz

Česká aplikace pro správu financí eÚčty.cz[1], je určená především pro rodiny, umožni tedy sdílení bankovního účtu mezi dalšími uživateli, kteří vedou společné účetnictví. Uživatel, který rodinu založil (zaregistroval), má právo přidávat a odebírat další uživatele, ostatní uživatelé rodiny mohou měnit pouze své nastavení. Každý uživatel z rodiny může mít různá přístupová práva pro různé účty dle konkrétního nastavení účtu. Příjmy a výdaje se zadávají ručně, transakce na účtech je možné přiřazovat do kategorií. Také máme možnost mít více účtů v různých měnách.

Další služby, které nabízí aplikace:

- Statistika transakcí spolu s grafovým podkladem
Nabízí tyto statistické přehledy: graf stavu na účtech, přehled příjmů a výdajů, přehled dle kategorií, podíl kategorií na výdajích, graf aut

- Import transakcí
Funkce import transakcí umožňuje načíst do služby eÚčty.cz příjmy a výdaje z jiných systémů (např. z internetového bankovnictví) bez nutnosti zdlouhavého opisování výpisů.

- Export do CSV
Vyexportuje kompletní přehled transakcí na všech účtech do formátu csv (Excel).

- Automatizace
Pravidelné transakce umožňují opakované vkládání pravidelných plateb. Transakce, které jsou zadané ve stejných

[1] https://www.eucty.cz/default.aspx

intervalech (každý měsíc, každý týden, ...) a které mají stejnou částku tedy uživatele nemusejí zapisovat ručně, ale můžou systém nastavit tak, aby tyto transakce zapisoval automaticky.

- **Správa hypotéky**

Existují dvě základní možnosti, jak lze hypotéky ve službě eÚčty.cz využívat:
- Sledování existující hypotéky. Umožní sledovat jednotlivé splátky (jak minulé, tak budoucí), zobrazovat si celkové statistiky o hypotéce a zároveň umožní nechat automaticky odečítat splátky z konkrétního účtu.
- Modelování hypotéky, umožní nechat zobrazit nejrůznější nabízené varianty, upravovat jejich parametry a hledat, která hypotéka bude pro uživatele nejvýhodnější.

- **Jednoduché plánování pro jednoduché výpočty**

Umožní zjistit kolik zbude po plánovaném nákupu, ověřit, zda si může dovolit uživatel nějaký výdaj apod.

- **Alarmy**

Upozorní když stav na účtu dosáhne určité částky

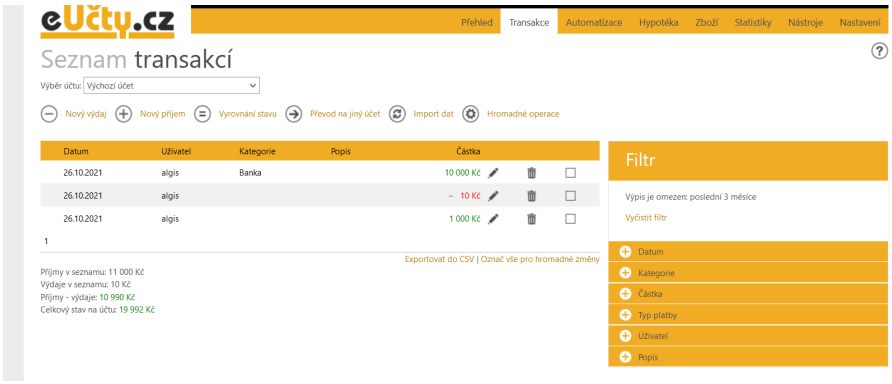
- **Správa auta**

Umožní uživateli si evidovat u auta najeté kilometry a spotřebu pohonných hmot.

- **Sledování energií**

Umožní uživateli sledovaní spotřeby energií ve domácnosti

Informace ohledně funkcionalit jsem čerpal z dokumentace aplikace[10]. Po spuštění aplikace se uživatelům zobrazí tabulka přehledu účtů, která obsahuje seznam všech bankovních účtů uživatele a zůstatky na těchto účtech. Kliknutím na jméno účtu se dostanou na seznam transakcí. Vpravo vedle seznamu transakcí je filtr, pomocí něhož lze omezit seznam jen na požadované transakce. Pod seznamem transakcí je součet příjmů a výdajů aktuálně zobrazených transakcí - ukázkový obrázek 2.1
Celkový dojem: v aplikaci je náročně se orientovat, zároveň je zbytečně komplikovaná a vizuálně zastaralá. Výhodou je to, že je zdarma a splňuje požadavky na svou funkcionalitu.
Hodnocení na Google Play[2]**:** 4.2
Cílové platformy: Android, Microsoft Windows



Obrázek 2.1: Ukázka eÚčty

[2] https://play.google.com/store/apps/details?id=com.las.eucty&hl=cs&gl=US

2.0.2 Spendee

Další aplikace českých vývojářů je Spendee[3]. Aplikace má více než 3 miliony stáhnutí. Hodnocení na Google Play[4] dosahuje 2,9 hvězdiček z 5, ale na App Store[5] dosahuje hodnocení 4,5 z 5. Aplikace má bezplatnou a prémiovou verzi, také je dostupná v češtině i angličtině.

Bezplatná verze:

- Ručně vkládání jednotlivých příjmů a výdajů
- Statistika transakcí spolu s grafovým podkladem
- Transakce je možné přiřazovat do kategorií
- Měsíční přehled
- Možnost stanovení rozpočtů
- Možnost exportu dat do CSV
- Uživatel má k dispozici pouze jednu peněženku a rozpočet

Premium verze (79 Kč/Měsíc):

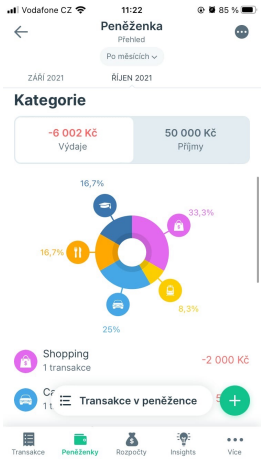
- Propojení Spendee se mobilním bankovnictvím, E-peněženkou (např. PayPal) nebo krypto peněženkou (např. Coinbase) - platby kartou nebo trvalé příkazy pak uživatel nemusí zadávat ručně
- Automatická kategorizace transakcí
- Sdílení účtu s jinými uživateli
- Vedení více účtů v různých měnách
- Neomezené peněženky a rozpočty

Oproti eÚčty má navíc funkci přidání fotografie účtenky nebo zboží do detailu transakce. Další velkou výhodou je možnost propojení s bankou, pak transakce uživatel nemusí zadávat ručně, což výrazně usnadňuje práci. Nevýhodou je to že bezplatná verze je hodně omezená. Zatímco prémioví uživatelé mají neomezený počet peněženek, bezplatný uživatel má k dispozici pouze jednu peněženku. Bez více peněženek uživatel nemá dobrý přehled všech svých transakcí. Informace ohledně funkcionalit jsem čerpal z dokumentace aplikace[5].

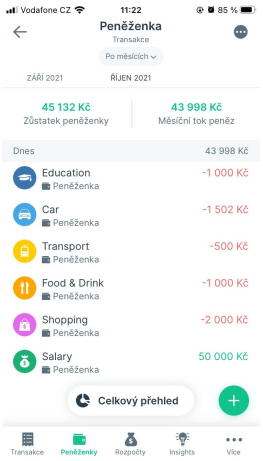
Celkový dojem: Uživatelské rozhraní aplikace je moderní, jasné a jednoduché. Po spuštění aplikace se uživatelům zobrazí obrazovka peněženky, která obsahuje seznam transakcí, kde mohou rychle zkontrolovat aktuální zůstatek a výdaje, také je možnost zobrazit celkový přehled, kde bude znázorněn statistický přehled s grafovým podkladem, viz obrázky 2.2 a 2.3. Navigace v aplikaci je snadná a intuitivní. Celkový dojem odpovídá hodnocení v App store.

Cílové platformy: Android, iOS

[3] https://www.spendee.com/
[4] https://play.google.com/store/apps/details?id=com.cleevio.spendee
[5] https://apps.apple.com/us/app/spendee-budget-money-tracker/id635861140



Obrázek 2.2: Ukázka Spendee - koláčový graf



Obrázek 2.3: Ukázka Spendee - peněženka

2.0.3 Wallet

Wallet[6] je produktem společnosti se sídlem v Praze. Čeští vývojáři BudgetBakers[7] mají na svém kontě s touto aplikací přes 5 milionu stažení na Google Play[8], hodnocení mají 4,7 bodů z 5. Je zdarma, ale některé funkce v aplikaci jsou dostupné jen v prémiové placené verzi. Aplikaci lze použít k běžnému zapisování výdajů a plánování rozpočtů. Umožní kontrolu nad cash flow, a to i v dlouhodobém horizontu. Také je možné propojení s bankou, pomocí které bankovní transakce jsou synchronizovány a následně roztříděny do kategorií. Aktuálně má aplikace napojeno přes 4 000 bank světa. Lze také sdílet bankovní účty s jinými uživateli. Funkcionálně je komplexnější než Spendee a vybavena větším množstvím bezplatných funkcí a statistických nástrojů, které by se perfektně hodily pro zkušené uživatele, ale naopak komplikované pro nové uživatele.

Bezplatná verze nabízí:

- Ručně vkládání jednotlivých příjmů a výdajů
- Statistika transakcí spolu s grafovým podkladem
- Transakce je možné přiřazovat do kategorií
- Měsíční přehled
- Stanovení rozpočtů
- Import/export transakce
- Vedení více účtů v různých měnách
- Uvádění místa provedení transakce na mapě
- Možnost třídění podle toho, zda mi přináší radost nebo ne
- Cashflow trend — spojení vývoje zůstatku a výdajů/příjmů v čase

Premium verze (129 Kč/Měsíc) pak nabízí:

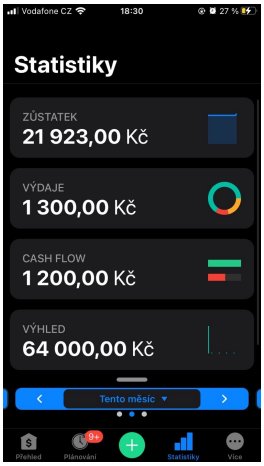
- Propojení bankou - platby kartou nebo trvalé příkazy pak nemusíte zadávat ručně, plná synchronizace
- Sdílení účtu s jinými uživateli
- Neomezený počet účtů

[6] https://web.budgetbakers.com/
[7] https://web.budgetbakers.com/
[8] https://play.google.com/store/apps/details?id=com.droid4you.application.wallet

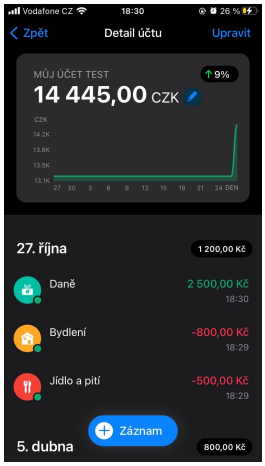
- Pokročilé statistiky a reporty

- Uspořádání podle barevných štítků

Uživatelské rozhraní aplikace je vytvořené podle designových standardů[11] vývoje iOS, což umožňuje přirozenou navigaci a snadné používání. Hlavní nevýhodou je to, že některé obrazovky jsou přetížené informacemi, to by mohlo vyvést uživatele z konceptu. Celkový dojem odpovídá hodnocení aplikace. Ukázka aplikace je na obrázcích 2.4 a 2.5
Cílové platformy: Android, iOS



Obrázek 2.4: Ukázka Wallet - statistika a grafové podklady



Obrázek 2.5: Ukázka Wallet - přehled účtu

2.0.4 George

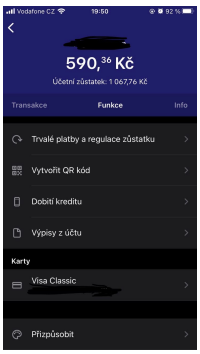
Česká aplikace George[9] od banky Česká spořitelna[10] pro správu osobních účtů, je zdarma a určená pouze pro klienty banky. Také je propojena a synchronizovaná s účty České spořitelny, má přehled o všech transakcích osobního účtu.

Služby, které nabízí:

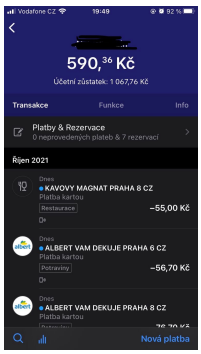
- Přehled všech transakcí
- Správce financí - analýza transakcí na základě kategorií, statistika a grafy
- Převody peněz na jiné účty
- Slevový program - moneyback
- Automatické třídění plateb - George roztřídí výdaje tak, že přesně vím, za co a kolik platím.
- Vlastní bezpečné nastavení - nastavím si vlastní přihlašovací údaje a upravím si vzhled podle sebe.
- Virtuální karta - možnost platby hodinkami či mobilem u obchodníků nebo na internetu.

George je odlišná od výše uvedených aplikací tím, že je určená pouze pro klienty České spořitelny. Také aplikace nabízí bezkontaktní platbu, pomocí "virtuální platební karty"- funguje stejně jako fyzická platební karta. Neexistuje v plastové podobě, ale jen v internetovém bankovnictví nebo ve mobilní peněžence[12]. Uživatelské rozhraní pro iOS je příjemné a snadné k používání. Ukázka aplikace je na obrázcích 2.6 a 2.7(z bezpečnostních důvodů jsou skrytá některá data).
Cílové platformy: Android, iOS, Microsoft Windows

[9] https://www.csas.cz/cs/internetove-bankovnictvi/george
[10] https://www.csas.cz/cs/osobni-finance



Obrázek 2.6: Ukázka George - funkce



Obrázek 2.7: Ukázka George - přehled účtu

2.0.5 Tabulka pro srovnání funkcionalit

Následující tabulka ukazuje srovnání poskytovaných funkcí analyzovaných aplikací. Hodnocení bylo provedeno na základě poskytovaných funkcí.
Hodnocení: 1 – výborný; 2 – chvalitebný; 3 – dobrý; 4 – dostatečný; 5 – nedostatečný;

Funkce	**MoneyExpert**	**eÚčty.cz**	**Spendee**	**Wallet**	**George**
Rozpočty	ANO	ANO	ANO	ANO	NE
Více účtů	ANO	ANO	ANO	ANO	ANO
Vytvoření kategorie	ANO	ANO	ANO	ANO	ANO
Filtrace dle kritérií	ANO	ANO	NE	ANO	ANO
Měsíční přehled	ANO	ANO	ANO	ANO	ANO
Grafy s přehledem	ANO	ANO	ANO	ANO	ANO
Propojení s bankou	NE	jen import	ANO	ANO	ANO
Automatizované opakování výdaje	NE	ANO	ANO	ANO	ANO
Přihlášení přes jiné aplikace	NE	NE	ANO	ANO	NE
Účet v jiné měně	ANO	ANO	ANO	ANO	ANO
Import/export dat	NE	ANO	jen export	ANO	NE
Sdílení účtu	ANO	ANO	ANO	ANO	NE
Hodnocení aplikaci	-	4	2	1	3

Tabulka 2.1: Srovnání funkcionalit

Z tabulky mě nejvíce zaujaly funkce:
Rozpočty - umožní naplánovat nějakou akci, takže neztratím více peněz než plánuji.
Více účtů - umožní flexibilitu a kontrolu všech svých finančních účtů.
Kategorie - pomáhá zjistit na jaké kategorie transakcí uživatel ztratí nejvíc peněz.
Grafy s přehledem - užitečný pro analýzu financí.
Vedení účtu v jiné měně - rozšiřuje práci s účtem.
Sdílení účtu - je velmi užitečná pro správu rodinných financi nebo s partnerem.

2.0.6 Závěr

Analyzoval jsem aplikace, které v sobě propojují všechny klíčové funkce, které jsem předpokládal v aplikaci MoneyExpert a zároveň podporují české prostředí. Analyzované aplikace mají většinou stejné funkce, takové jako přehled účtu, stanovení rozpočtů, kategorizace transakcí a uváděni statistiky s grafickým podkladem. Existují aplikace u kterých jsou bezplatné pouze základní funkce. Pro pokročilejší funkce, jako podrobná statistika a sdílení peněženky, si musí uživatel koupit premium verzi.

Jako nejlepší mi přišla aplikace Wallet, protože podporuje všechny funkce a zároveň je snadná na používání. Následně jako 2. je Spendee, také podporuje většinu funkcí a stále je přehledná, ale má hodně omezenou bezplatnou verzi. Pak jako 3. je George, je to skvělá aplikace, snadná a přehledná, ale má omezení v tom, že je určena pouze pro klienty České spořitelny. Nejhorší aplikací se stala eÚčty, kvůli své náročnosti na používání, běžný uživatel se dost těžké orientuje v systému.

Výhodou aplikace MoneyExpert bude její jednoduchost a dostupnost, také časově i finančně nenáročný vývoj. Na rozdíl od analyzovaných aplikací umožní MoneyExpert sledování osobních závazků, plánovaní rozpočtů a bude kompletně zadarmo. Pro to abych dosáhl cíle své práce, bude jako vzor pro MoneyExpert sloužit aplikace Wallet. V aplikaci ale nebude propojení účtů s online bankou, kvůli časové náročnosti, také nebudou implementované funkce jako: automatizace opakujících výdajů, přihlášení přes jiné aplikace, Import/Export dat.

2.1 Určení požadavků

V teto kapitole si určíme specifikaci požadavků na systém. Soustředil jsem se na funkční a nefunkční požadavky. Zdrojem funkčních požadavků bylo to co vyplynulo z analýzy a samotného zadání projektu, také komunikace se vedoucím práce, který vystupuje v roli zákazníka. Nefunkční požadavky zahrnují hlavně použité technologie, cílové platformy a bezpečnost.

2.1.1 Funkční požadavky (FP)

FP-1: Aplikace podporuje přihlášení / registraci

FP-2: Účty

- Systém umožní vytvářet více bankovních účtů a upravovat je
- Systém umožní vést bankovní účty v různých měnách(CZK, EUR)

FP-3: Systém bude pracovat s dvěma typy transakcí - výdaj, příjem.

FP-4: Převod transakci - funguje tak, že se v něm uvádí cílový bankovní účet a zdrojový, pak v účtu, který převod přijal se přidá transakce a ve druhém se odstraní.

FP-5: V systému transakce se budou zadávat ručně, protože nemám přístup do banky a těžké to můžu dělat online.

FP-6: Transakce se budou dělit na kategorie (např. jídlo, ubytování, auto a podobné) - pro lepší přehlednost ztrát - na jaké kategorie transakcí šlo peněz nejvíce.

- Systém umožní vytvářet kategorie a přidávat do ní transakci .

FP-7: Systém bude zobrazovat přehled o bankovních účtech a transakcích.

- Součet všech výdajů a příjmů za zvolené období.
- Aktuální zůstatek na účtu.
- Seznam všech transakcí za zvolené období.

FP-8: Systém umožní sledování svých závazků - uživatel si vytvoří závazek, nastaví mu termín kdy aplikace má ho upozornit a deadline, první termín slouží proto aby aplikace uživatele upozornila, že se blíží tento závazek.

FP-9: Systém umožní vést rodinné finance nebo s partnerem - sdílení účtu.

FP-10: Systém umožní jednoduchý rozpočet, takže neztratím více peněz než plánuji.

FP-11: Systém bude vytvářet jednoduchou statistiku financí a na základě toho vykreslovat grafy - Kolik ztratil za zvolené období a na jaké kategorie transakcí šlo peněz nejvíce, to pomůže uživateli pochopit na co jdou peníze.

2.1.2 Nefunkční požadavky (NP)

NP-1: Back-end musí poskytovat komunikační prostředí přes REST

NP-2: Data jsou posílány pomocí JSON formátu

NP-3: Back-end technologie: JAVA, Spring boot[11], MySQL, postgresql

NP-4: Front-end část: Standardní technologie tedy HTML, CSS, JavaScript, framework Vue.js[12],

framework Vuetify[13]

NP-5: Aplikace je ve vrstevnatý architektuře: Persistentní vrstva, DAO, Services, Komunikační vrstva (REST), UI

NP-7: Systém je zaměřen na desktopový webový prohlížeč - Nutná podpora pro: Google Chrome[14], Mozilla Firefox[15], Microsoft Edge[16]

NP-8: Aplikace poskytne uživatelům bezpečný způsob autentizace aby jejich osobní údaje byly v bezpečí

2.1.3 Analýza technologií

Pro back-end část jsem se rozhodl použit Java Spring boot framework, má výhodu v tom, že je postavený na MVC architektuře, aplikace rozděluje do 3 vrstev a tím je přehledná a jednoduše rozšiřitelná. Také spring boot má zabudovanou dependency injection - způsob, jak si objekt řekne o další objekty, které ke své činnosti potřebuje a je na nich závislý, má spoustu menších knihoven, které nám usnadní práci s objekty a komunikaci s front-end[13]. Také poskytuje flexibilní způsob přizpůsobení Java Beans, konfigurací XML a databázových transakcí, k tomu ještě má výkonné dávkové zpracování a spravuje koncové body REST. Spring Boot konfiguruje vše automaticky; nejsou nutné žádné ruční úpravy a usnadňuje správu závislostí. Další výhodou spring boot je možnost mapování dat do entit objektů ORM (Object-relational Mapping)[21].

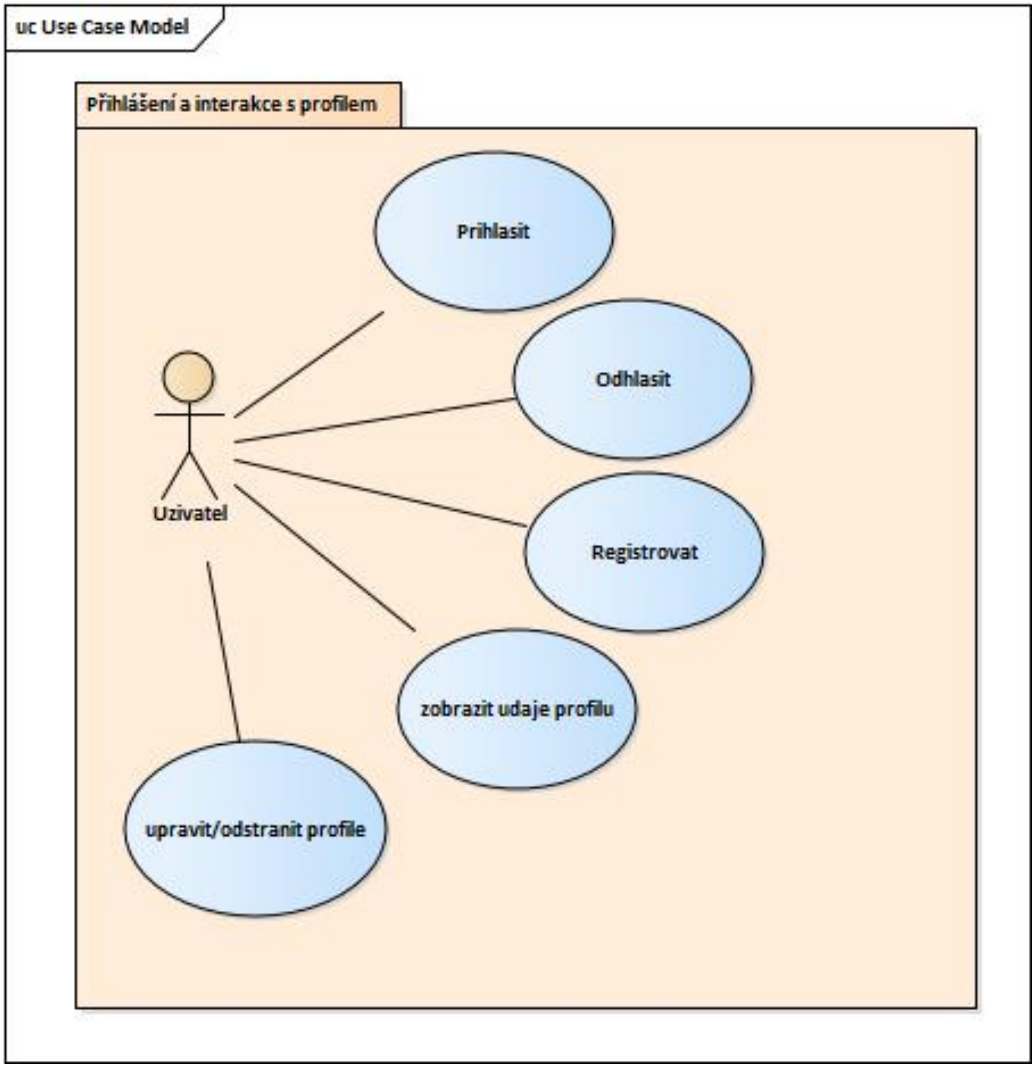
Pro front-end část byl použit framework Vue.js[17] v kombinaci s frameworkem Vuetify[18]. Vue.js je progresivní framework pro vytváření uživatelských rozhraní. Na rozdíl od jiných monolitických frameworků je Vue od základu navržen tak, aby byl postupně přizpůsobitelný. Vue.js je zaměřen na vrstvu ViewModel vzoru MVVM. Propojuje view a model prostřednictvím obousměrných datových vazeb. Také Vue schopen pracovat s SPA (Single-page application - definice je v 4.2), když se používá v kombinaci s moderními nástroji a podpůrnými knihovnami[31]. Vuetify - je kompletní framework uživatelského rozhraní postavený na Vue.js. Poskytuje vývojářům nástroje, které potřebují k vytváření bohatých a zajímavých uživatelských zkušeností. Je vyvinut přesně podle specifikace Material Design[19], přičemž každá součást je pečlivě vytvořena tak, aby byla modulární, responzivní a výkonná[32].

Před nasazením aplikace bylo použito jako datové úložiště MySQL databáze - viz nasazení aplikace 4.3.

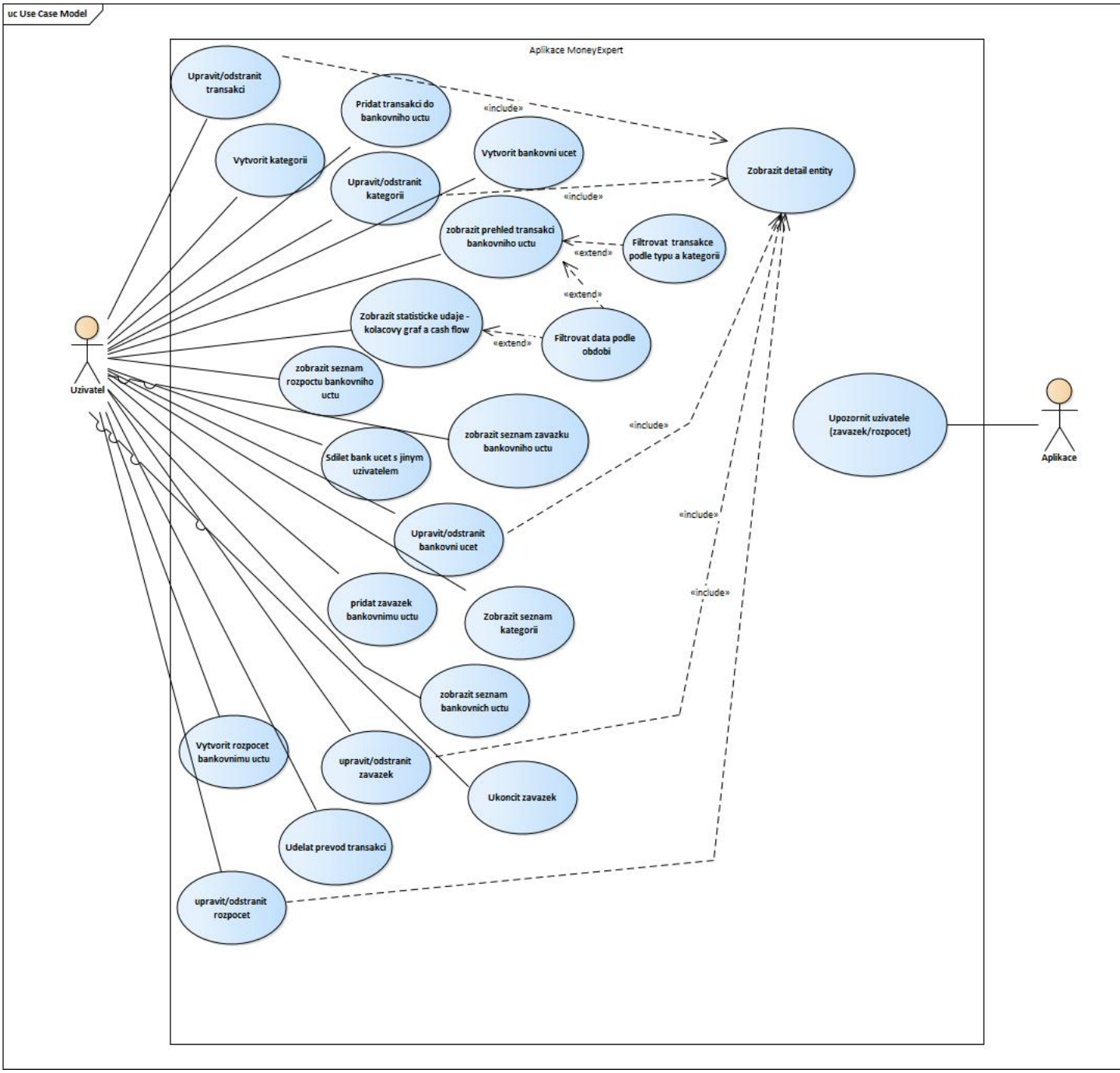
[11] https://spring.io/projects/spring-boot
[12] https://vuejs.org/
[13] https://vuetifyjs.com/en/
[14] https://www.google.com/intl/ru/chrome/
[15] https://www.mozilla.org/ru/firefox/new/
[16] https://www.microsoft.com/en-us/edge
[17] https://vuejs.org/
[18] https://vuetifyjs.com/en/
[19] https://material.io/

2.1.4 Případy užití - Use Case(UC)

Případy užití vyjadřují, kdo bude jakým způsobem používat systém. Cílem modelování případů užití je najít hranice systému. Use Case diagram se skládá z případů užití (use case), dále aktérů (actors) a vztahů mezi nimi. Aktér je ten kdo systém používá a ovlivňuje. Případy užití - to co bude systém umožňovat.
Modelování případů užití, je způsob zachycení pouze funkčních požadavků.[28] Obrázky případů užití jsou znázorněny na 2.8 a 2.9



Obrázek 2.8: Model případů užití



Obrázek 2.9: Model případů užití

UC1: Přihlásit/Registrovat

- uživatel vyplní své přihlašovací údaje
- při registraci uživatel vyplní potřebné údaje - jméno, příjmení, email, username, heslo

UC2: Zobrazit údaje profilu

- zobrazí se údaje uživatelského profilu

UC3: Upravit/odstranit profile

- uživatel si zobrazí údaje profilu (UC2)

- ve formuláři buď upraví zadané hodnoty nebo odstraní uživatelský profile.

UC4: Zobrazit přehled transakcí bankovního účtu

- zobrazí se kategorie, datum, částka pro všechny příjmy a výdaje ve zvoleném období.

UC5: Zobrazit statistické údaje - koláčový graf a cash flow

- hodnoty v koláčovém grafu jsou součty výdajů bankovního účtu v jednotlivých kategoriích za zvolené období.
- zobrazí se také cash flow - součet výdajů a součet příjmů ve zvoleném období.

UC6: Zobrazit seznam bankovních účtů

- zobrazí se název a aktuální zůstatek v měně účtu pro všechny vytvořené a dostupné bankovní účty

UC7: Zobrazit seznam rozpočtů bankovního účtu

- zobrazí se částky rozpočtů pro jednotlivé kategorie

UC8: Zobrazit seznam závazků bankovního účtu

- zobrazí se závazky s atributy: název, deadline, datum upozornění, částka a popis závazku

UC9: Zobrazit seznam kategorii

- zobrazí se výchozí a vytvořené uživatelem kategorie

UC10: Zobrazit detail entity

- zobrazí se detail vybrané entity

UC11: Přidat transakci do bankovního účtu

- uživatel si zobrazí seznam transakci. (UC4)
- zvolí přidat transakci
- ve formuláři vyplní částku transakce, poznámku a kategorii a typ transakci (příjem, výdaj)

UC12: Filtrovat data podle období

- uživatel má možnost filtrovat transakce a statistické údaje podle sledované období

UC13: Filtrovat transakce podle typu a kategorii

- uživatel má možnost filtrovat transakce podle typu a kategorii

UC14: Vytvořit bankovní účet

- uživatel si zobrazí seznam bankovních účtů. (UC6)
- zvolí možnost přidat nový bankovní účet
- ve formuláři napíše název a typ měny účtu(CZK, EUR), vyplní počáteční zůstatek.
- při vytvoření nového bank. účtu se vytvoří startovní transakce, která bude mít částku jako parametr ”zůstatek”, ale pokud ten zůstatek je menší nebo roven 0, tak start. transakce se nepřidá.

UC15: Udělat převod transakci

- uživatel si zobrazí seznam transakce (UC4)
- uživatel si vybere transakci
- zobrazí se detail transakce (UC10)
- v detailu entity vybere možnost udělat převod

- ve zobrazeném formuláři zvolí cílový bankovní účet a stiskne udělat převod
- ve zdrojovém bank. účtu ta transakce se odstraní a ve cílovém účtu se objeví

UC16: Upravit/odstranit bankovní účet

- uživatel si zobrazí seznam bank. účtů (UC6)
- uživatel si vybere bank. účet
- zobrazí se detail učtu (UC10)
- ve formuláři buď upraví informace nebo odstraní účet

UC17: Vytvořit rozpočet bankovnímu účtu

- uživatel si zobrazí seznam rozpočtů. (UC7)
- zvolí přidat rozpočet
- ve formuláři napíše částku, název, sledovanou kategorii, také procento od částky, kdy systém má upozornit uživatele aby nepřesáhl rozpočet

UC18: Sdílet bank. účet s jiným uživatelem

- uživatel si zobrazí bank. účty (UC6)
- vybere ten účet který chce sdílet
- zobrazí se detail učtu (UC10)
- uživatel vybere možnost "sdílet účet"
- uživatel zadá username uživatele s kterým chce sdílet ten bankovní účet
- stisknutím "sdílet účet"přidá vybraného uživatele do vlastníků bank. účtu

UC19: Upravit/odstranit transakci

- uživatel si zobrazí seznam transakce (UC4)
- uživatel si vybere transakci
- zobrazí se detail transakce (UC10)
- ve formuláři buď upraví informace nebo odstraní transakci

UC20: Upravit/odstranit rozpočet

- uživatel si zobrazí seznam rozpočtů (UC7)
- uživatel si vybere rozpočet
- zobrazí se detail rozpočtu (UC10)
- ve formuláři buď upraví zadané hodnoty nebo odstraní rozpočet.

UC21: Upozornit uživatele (závazek/rozpočet)

- případ závazku: systém upozorní uživatele pokud aktuální datum bude větší nebo roven času notifikace, anebo pokud datum bude větší nebo roven deadlinu
- případ rozpočtu: systém upozorní uživatele pokud součet částek transakcí s určenou kategorii bude větší nebo roven částce rozpočtu, anebo pokud procento součtu částek transakcí s určenou kategorii bude větší nebo rovné procentu od částky rozpočtu kdy má systém upozornit
- notifikace bude vypadat jako ikona oznámení v aplikaci

UC22: Přidat závazek bankovnímu účtu

- uživatel si zobrazí seznam závazků. (UC8)
- zvolí přidat závazek
- ve formuláři vyplní název, deadline, čas kdy má systém upozornit uživatele na závazek, částku a popis závazku

UC23: Upravit/odstranit závazek

- uživatel si zobrazí seznam závazků. (UC8)
- uživatel si vybere závazek
- zobrazí se detail závazku (UC10)
- ve formuláři buď upraví zadané hodnoty nebo odstraní závazek.

UC24: Vytvořit kategorii

- uživatel si zobrazí seznam kategorii (UC9)
- zvolí vytvořit kategorii
- ve formuláři vyplní název

UC25: Upravit/odstranit kategorii

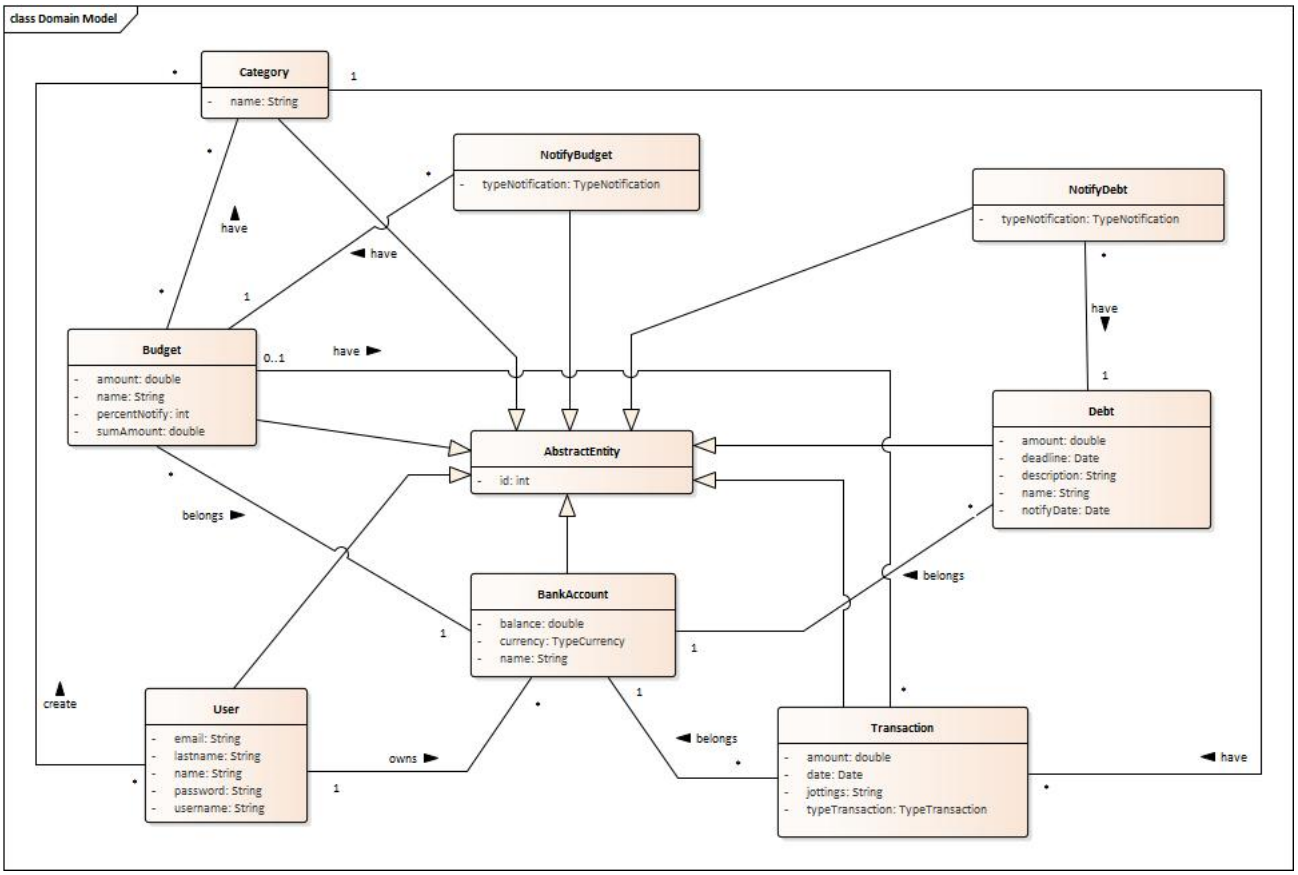
- uživatel si zobrazí seznam kategorii
- uživatel si vybere kategorii
- zobrazí se detail kategorie (UC10)
- ve formuláři buď upraví zadané hodnoty nebo odstraní kategorii.

UC26: Ukončit závazek

- uživatel si zobrazí seznam závazků. (UC8)
- uživatel si vybere závazek
- zobrazí se detail závazku (UC10)
- ve formuláři si vybere možnost ukončit závazek
- stisknutím se objeví upozornění, že při ukončení závazku se odstraní samotný závazek a objeví se transakce typu výdaj s částkou závazku v bank. účtu, ke kterému patří závazek.

2.1.5 Doménový model

Doménový model je strukturovaná vizuální reprezentace vzájemně propojených konceptuálních tříd nebo objektů, které odrážejí skutečný svět. Pomáhá pro evidenci business entit a vizualizuje asociace mezi nimi pomocí notaci Unified Modeling Language(UML).[29] - obrázek Doménový model 2.10



Obrázek 2.10: Doménový Model

AbstractEntity
Používá se pro vygenerovaní Id entit.

User
Uživatel používající Aplikaci. Vlastní bankovní účet a vytváří kategorie, také má možnost sdílet své bankovní účty s jinými uživateli.

BankAccount
Bankovní účet, obsahuje množinu transakcí, závazků a rozpočtů. Má atributy název(name), typ měny (currency) - pro práci s různými měnami, balance - aktuální zůstatek na účtu.

Transaction
Transakce má atributy amount - částka peněz, date, jottings - poznámky a typeTransaction - dva typy příjem, výdaj. Patří do Category a BankAccount. Také může patřit do Budget, ale nemusí.

Category
Kategorie má množinu transakcí a atribut název(name). Pak má ManyToMany relaci s uživatelem - to je tak, protože jsou default kategorie, které se přiřadí automaticky po registraci uživatele. Pak místo toho aby se vytvářely default kategorie po každé co se zaregistruje uživatel, pomocí ManyToMany relaci stačí přidat do tabulky relaci mezi uživatelem a kategorii - existující kategorii a uživatele.

Budget

Rozpočet má určitou kategorii a atributy název(name), částka(amount) - limitní částka pro určitou kategorii, sumAmount - součet částek přijatých transakcí, procento upozorněni (percentNotify) - procento z sumAmount, při kterém uživatel bude upozorněn.

Debt

Závazek patří do bankovního účtu, má atributy amount - částka, notifyDate - od kdy má upozornit uživatele, deadline - kdy má skončit dluh, název(name) závazku a popis(description).

NotifyDebt

Entita, která se vytvoří když se má upozornit Debt - pokud u Debt se nastane notifyDate nebo Deadline. Pak pomocí tyto entity víme všechny Debt, které mají být upozorněné.

NotifyBudget

Entita, která se vytvoří když se má upozornit Budget - pokud u Budget atribut sumAmount překročí amount nebo percentNotify. Pak pomocí tyto entity víme všechny Budget, které mají být upozorněné.

Kapitola 3

Návrh

Následující kapitola popisuje architekturu aplikace a techniky používané k návrhu a sestavení aplikace. Kapitola také zahrnuje databázový model, finální prototypy obrazovek, a zároveň vytváří jasnou strukturu pro nadcházející fázi procesu vývoje softwaru.

3.1 Architektura

Architektura je typu klient-server - struktura distribuované aplikace, která rozděluje úlohu nebo pracovní zátěž mezi poskytovatele služby, nazývané servery, a žadatele o služby nazývané klienti. V architektuře klient-server, když klientský počítač odešle požadavek na data serveru přes internet, server přijme požadovaný proces a doručí požadované datové pakety zpět klientovi.[77]
Výhody architektury klient-server:

- Dokážeme aplikaci snadno udržovat
- Dobře škálovatelná
- Ukládání dat centralizovaně zaručuje, že jsou data aktuální a dostupná
- Všechna data jsou uložena na serveru, který je mnohem bezpečnější než většina klientů. Je snazší nastavit ovládání oprávnění na serveru tak, aby k datům měli přístup pouze klienti s příslušnými přístupovými právy.[77]

Nevýhody:

- Klienti jsou náchylní k virům, pokud jsou přítomni na serveru nebo jsou na server nahráni.
- Servery jsou náchylné k útokům Denial of Service (DOS).
- Bezpečnostní riziko: datové pakety mohou být během přenosu podvrženy nebo pozměněny.
- Bezpečnostní riziko zachycení přihlašovacích údajů nebo jiných užitečných informací uživatele - útoky MITM (Man in the Middle).[77]

Aplikace na straně serveru zpracovává požadavky HTTP, provádí aplikační logiku, získává a aktualizuje data z databáze. Klientská strana poskytuje uživatelské rozhraní, tedy prostředek pro příjem, zobrazování a úpravu údajů zadaných uživatelem, které slouží jako podklad pro požadavek na server. Kromě toho lze klienta nakonfigurovat tak, aby zpracovával podmnožinu dat, aby se snížilo zatížení zdrojů serveru.

Implementace aplikace je rozdělena na serverovou a klientskou část. V frameworku spring boot pro komunikaci mezi klientem a serverem se používá REST API, architektonický styl pro aplikační rozhraní (API), které používá HTTP požadavky pro přístup a použití dat. HTTP umožňuje posílat druhy požadavků jako - GET, PUT, POST a DELETE, které se týkají čtení, aktualizace, vytváření a mazání operací týkajících se zdrojů[78]. Data mezi klientem a serverem jsou posílány v JSON formátu. Kvůli bezpečnostnímu riziku úniku dat, je komunikace zabezpečená zašifrovaným kanálem HTTPS - viz kapitola 4.3. Na obrázku 3.1 je znázorněn Diagram komponent projektu, který popisuje logickou architekturu systému[30].

3.2 Server

V serverové části je použit framework Spring Boot, má vícevrstvou architekturu - Datovou, Aplikační a Prezentační. Výhoda vícevrstvé architektury je tom, že jednotlivé vrstvy můžeme nezávisle vyvíjet, také se aplikace lépe testuje. Datová vrstva je pro interakci s databází. Aplikační vrstva obsahuje byznys logiku. Prezentační vrstva prezentuje uživateli data, která získává z vrstvy aplikační.

3.2.1 Controller

Patří do Prezentační vrstvy - zpracovává požadavky HTTP, překládá JSON parametry do objektů entit, ověřuje požadavek a přenáší jej do aplikační(business) vrstvy, také zároveň data z business vrstvy přeposílá klientovi prostřednictvím HTTP odpovědi v JSON formátu. Komunikace mezi serverem a klientem prochází přes něj, pomocí rozhraní REST API.

3.2.2 Service

Patří od Aplikační vrstvy, zpracovává veškerou business logiku, také je odpovědný za správný přenos dat mezi datovou a prezentační vrstvou. Skládá se ze tříd a metod, které zahrnují výpočty založené na vstupech a uložených datech, ověřování veškerých dat pocházejících z prezentace a přesné zjištění, jakou logiku zdroje dat odeslat, v závislosti na příkazech přijatých z prezentace.

3.2.3 Dao, Entity

Hlavní komponenta pro interakci s databázi Dao patří do datové vrstvy - spravuje spojení se zdrojem dat za účelem získání a uložení dat, také překládá business objekty(Entity) z nebo do řádků databáze. Každé repository(Dao) pracuje s vlastní třídou Entit, které jsou vygenerovány pomocí JPA entit v spring boot. Na obrázku 3.2 je znázorněn databázový model, který sestává z Entit.

3.3 Klient

V klientské části je použit framework Vue.js v kombinaci s frameworkem Vuetify, jsou to progresivní frameworky pro vytváření uživatelských rozhraní. Technicky je Vue.js zaměřen na vrstvu ViewModel vzoru MVVM[1]. Propojuje view a model prostřednictvím obousměrných datových vazeb[60].

3.3.1 View-Model

Aplikační logika klienta. Připravuje data z modelové vrstvy pro zobrazení v UI a na druhé straně reaguje na změny v UI, které propisuje zpět do Modelu. Ve Vue.js je každá instance Vue ViewModel. Jsou vytvořeny pomocí konstruktoru Vue nebo jeho podtříd[60].

3.3.2 Model

Ve Vue.js jsou modely jednoduše prosté objekty JavaScriptu nebo datové objekty. Jakmile je objekt použit jako data uvnitř instance Vue, stává se reaktivním. Můžeme manipulovat s jejich vlastnostmi a instance Vue, které je sledují, budou o změnách informovány. Vue.js dosahuje transparentní reaktivity převodem vlastností datových objektů do ES5 getter/setters. Není potřeba špinavá kontrola, ani nemusíme výslovně signalizovat Vue, aby aktualizoval pohled. Kdykoli se data změní, pohled se aktualizuje[60].

3.3.3 API Service

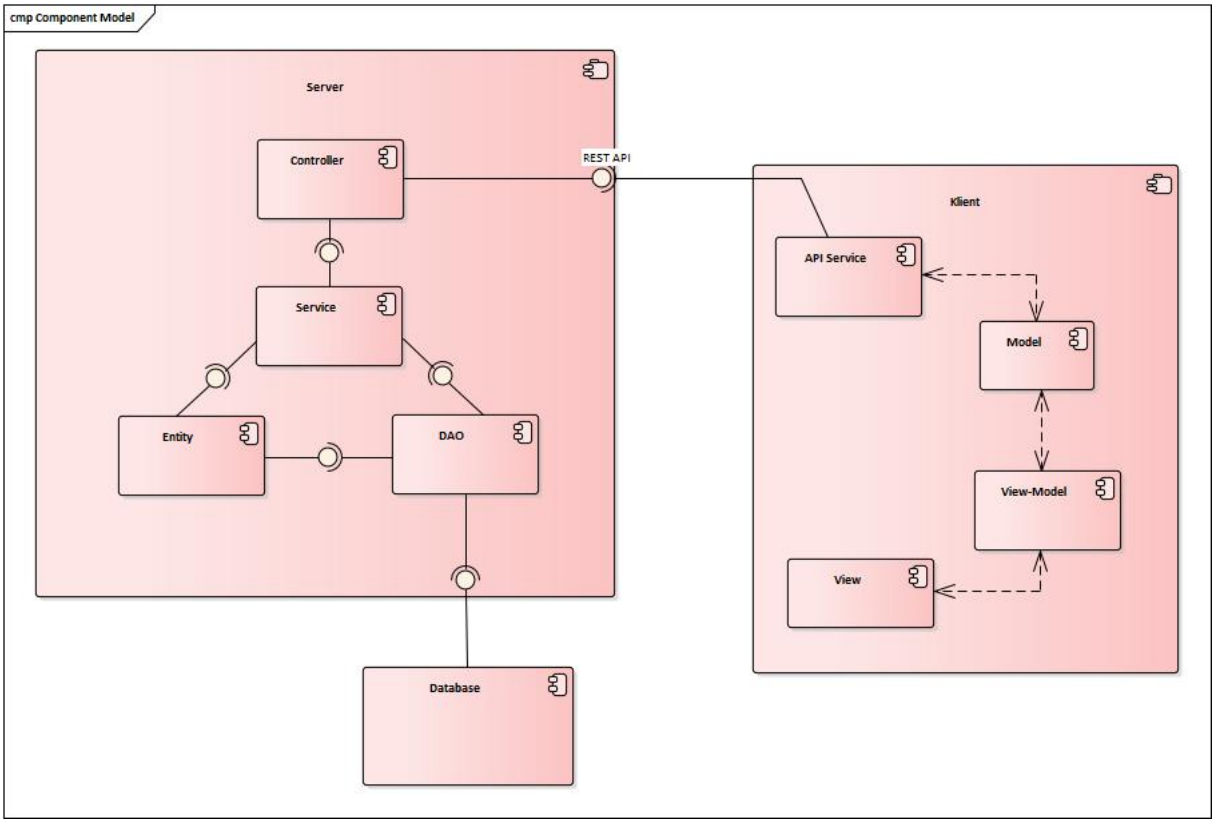
Posílá/přijímá data na/z server přes REST API, pak data vrátí do modelu, který API požadoval.

[1] https://whatis.techtarget.com/definition/Model-View-ViewModel

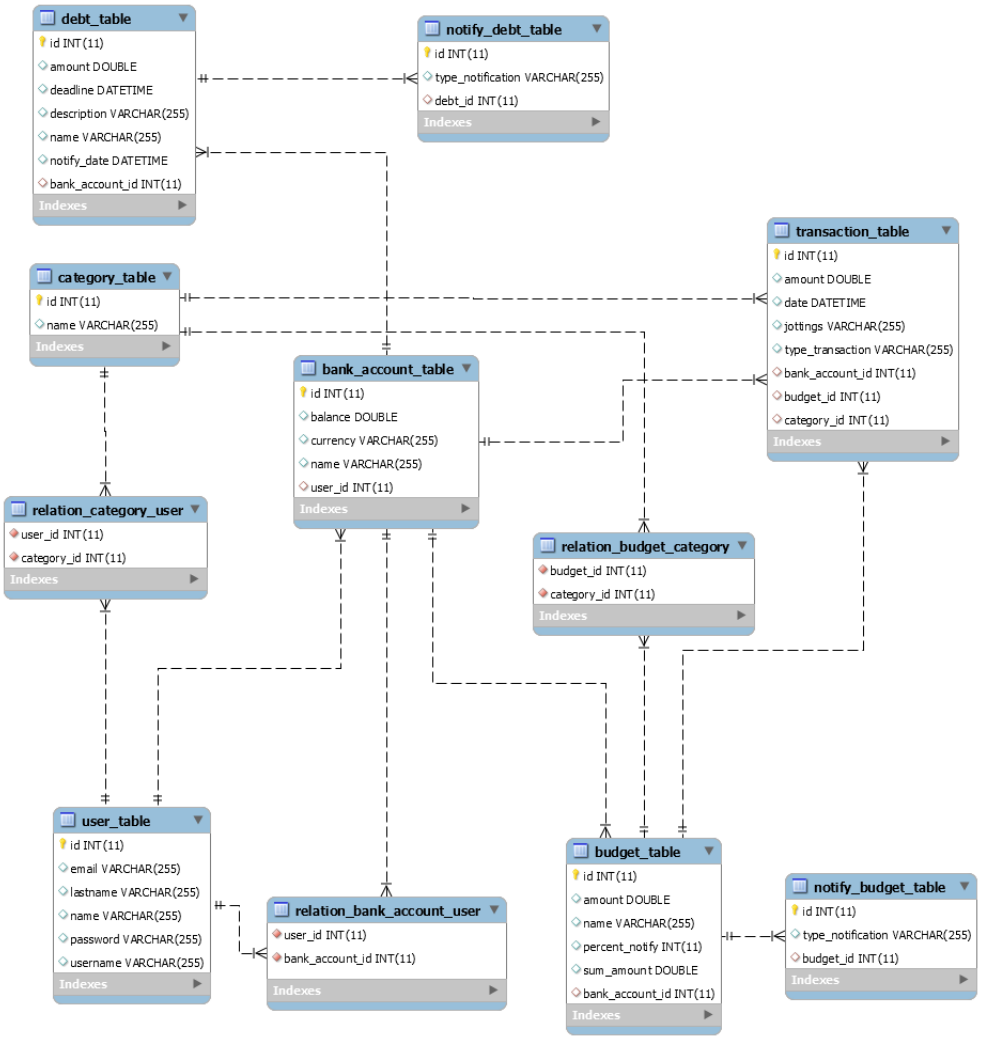
3.3.4 View

Převádí data reprezentovaná modelem do podoby vhodné k interaktivní prezentaci uživateli. Vue.js používá šablony založené na DOM. Každá instance Vue je spojena s odpovídajícím prvkem DOM. Když je instance Vue vytvořena, rekurzivně prochází všemi podřízenými uzly svého kořenového prvku a nastavuje potřebné datové vazby. Poté, co je pohled zkompilován, začne reagovat na změny dat[60].

*Poznámka k odkazu na zdroj informace [60] - je to oficiální dokumentace Vue pro verzi 0.12, ale informace uvedené v kapitole jsou stále aktuální i pro Vue verzi 2.x, která je používaná v projektu.



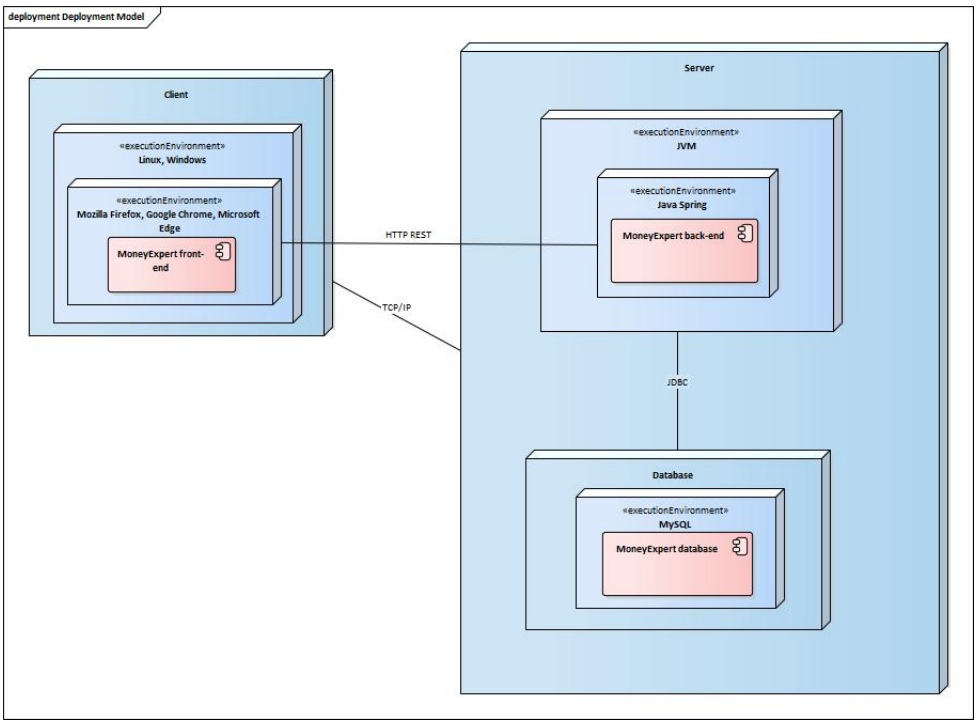
Obrázek 3.1: Diagram komponent



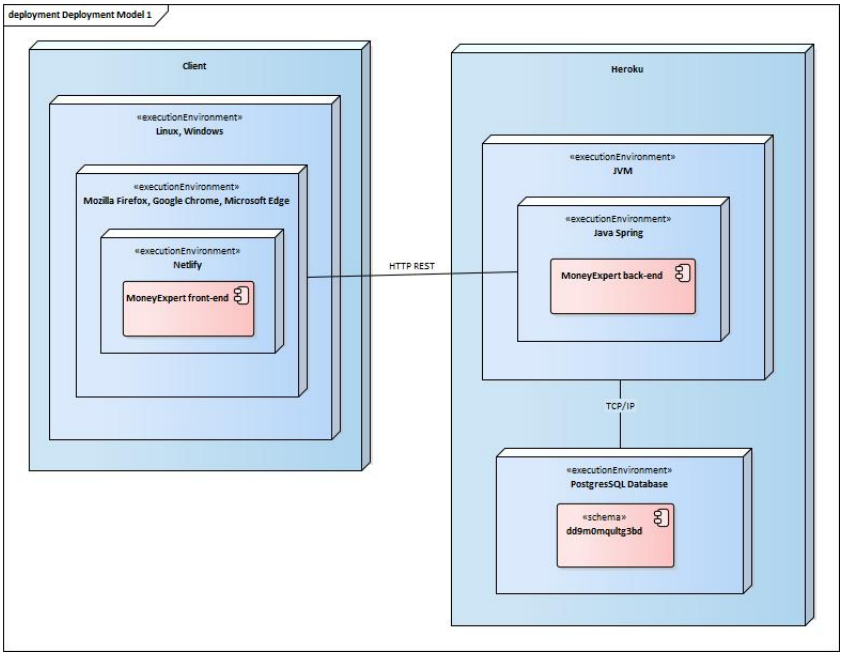
Obrázek 3.2: Databázový model

3.4 Diagram nasazení

Diagram nasazení zobrazuje hardwarové a softwarové komponenty a jejich vzájemné vztahy, ukazuje, kde přesně je která softwarová komponenta nasazena[62]. Na prvním obrázku 3.3 je diagram nasazení v lokálním prostředí a na druhém 3.4 je diagram v cloud platform - viz nasazení v cloud platform 4.3. Poznámka: diagram nasazení v cloud platform není přesný, je spíše orientační.



Obrázek 3.3: Diagram nasazení - lokální prostředí

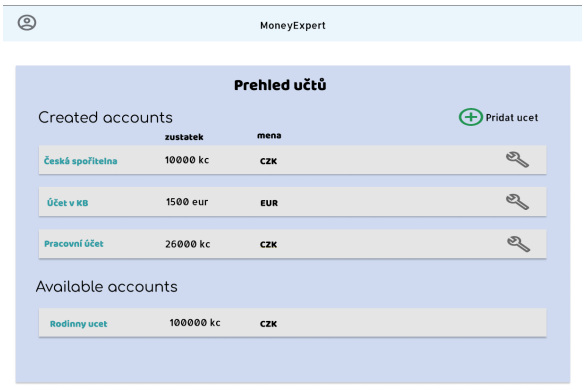


Obrázek 3.4: Diagram nasazení - cloud platform

3.5 Design

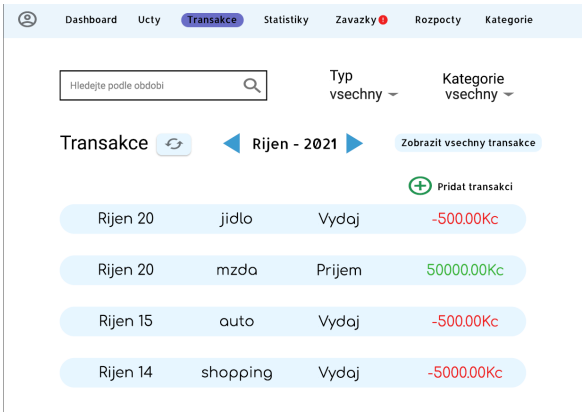
Následující kapitola zahrnuje finální prototypy obrazovek. Prototyp je vyjádřením záměru designu. Prototypování umožňuje prezentovat design a vidět ho v akci. Prototyp je simulace toho, jak bude hotový produkt fungovat. Umožňuje otestovat použitelnost a proveditelnost designu aplikace.

Návrh designu byl udělán díky získaným zkušenostem z oboru SIT. Dále jsou obrazovky High-fidelity prototypu[42], které byly vytvořeny pomocí nástroje Figma[2]:

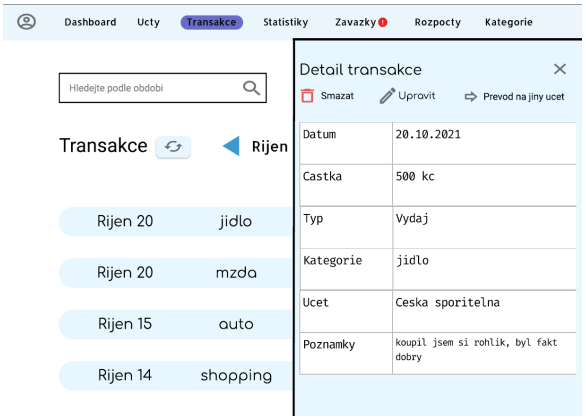


Obrázek 3.5: Přehled účtů

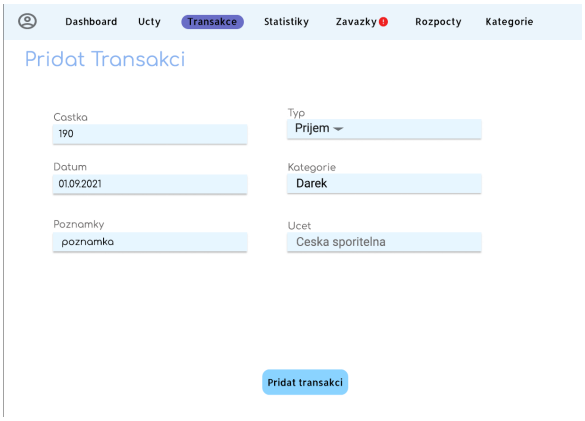
Obrázek 3.6: Dashboard



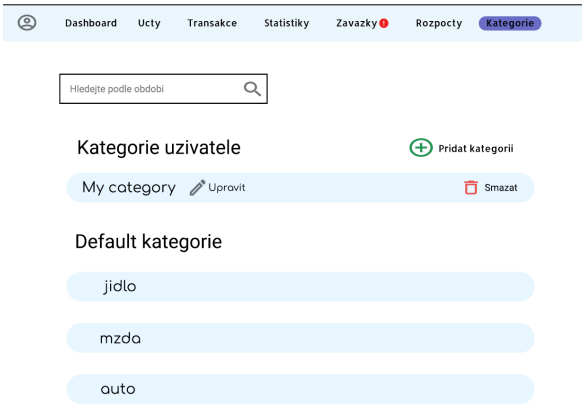
Obrázek 3.7: Seznam transakcí



Obrázek 3.8: Detail transakce

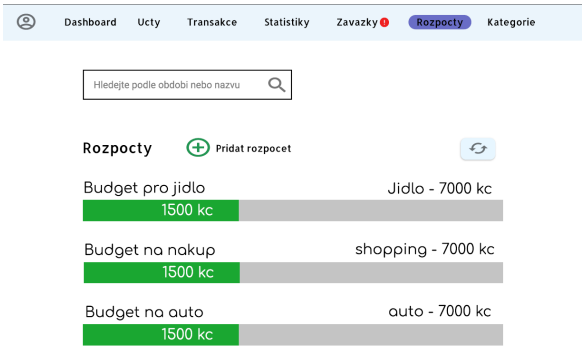


Obrázek 3.9: Přidání transakce

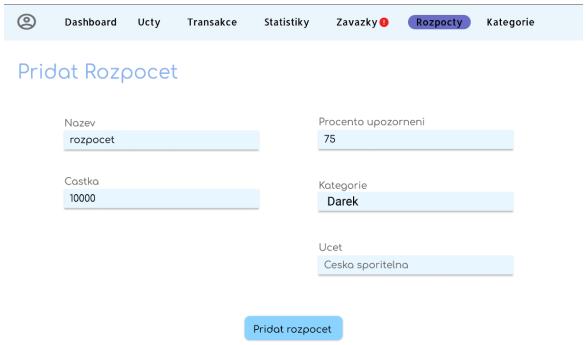


Obrázek 3.10: Seznam kategorií

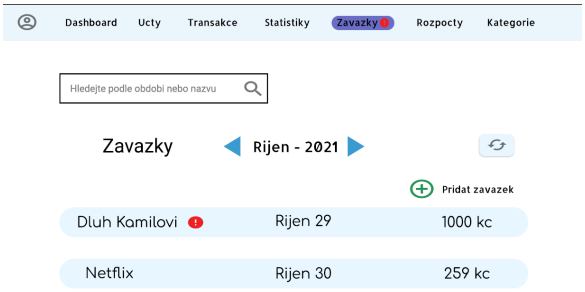
[2] https://www.figma.com



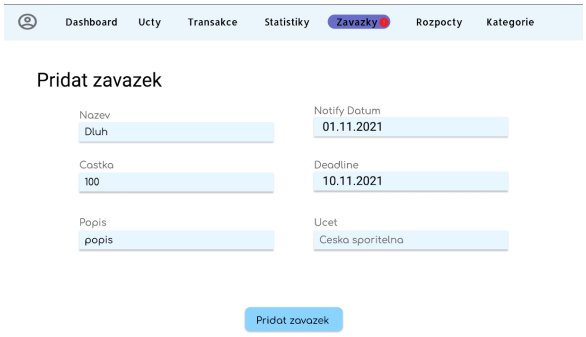
Obrázek 3.11: Seznam rozpočtů



Obrázek 3.12: Přidání rozpočtu



Obrázek 3.13: Seznam závazků



Obrázek 3.14: Přidání závazku

Pro navigaci slouží horní navigační menu, v případě upozornění se objeví u položek "Závazky" nebo "Rozpočty" ikona oznámení.
Dashboard souží pro navigaci a poskytování základních informace o bankovním účtu.
Obrazovka statistiky obsahuje dva grafy:

- koláčový - zobrazuje součty výdajů v určitém období na určité kategorie.
- cash flow - zobrazuje součet výdajů a příjmů v určitém období.

Obrazovka 3.5 obsahuje seznam vytvořených a dostupných bankovních účtů. Každá položka v seznamu má název, zůstatek a měnu bankovního účtu. Uživatel má právo upravovat/odstraňovat pouze vytvořené bankovní účty.
Obrazovka 3.7 obsahuje transakce bankovního účtu v určitém období. Uživatel může filtrovat transakce podle typu a kategorii, také vyhledávat podle období. Pak má možnost zobrazení všech transakcí bez omezení na období.
Obrazovka 3.8 zobrazuje detail vybrané transakce a možnosti editování, odstranění a převodu na jiný bank. účet.
Obrazovka 3.10 obsahuje kategorie uživatele a výchozí kategorie. Uživatel má právo upravovat/odstraňovat pouze vytvořené kategorie, také má možnost vyhledávat podle názvu.
Obrazovka 3.11 obsahuje rozpočty bankovního účtu. Každá položka v seznamu má kategorii, název a částku rozpočtu. Při upozornění se objeví ikona oznámení a položka se obarví červenou barvou. Uživatel může vyhledávat podle názvu.
Obrazovka 3.13 obsahuje závazky bankovního účtu. Každá položka v seznamu má název, deadline a částku závazku. Při upozornění se objeví ikona oznámení. Uživatel může filtrovat závazky podle blížících deadlinů.
*Celý prototyp aplikace je pod odkazem[3].

[3] https://www.figma.com/file/ONBvf9JUxCMUSfhFadfnTR/MoneyExpert?node-id=0

Kapitola 4

Implementace

V této kapitole je rozepsána implementace serverové a klientské části - pro vývoj byl použit nástroj IntelliJ IDEA[1] a verzovácí nástroj git[2].

4.1 Serverová část - back-end

V serverové části byl použit framework Spring Boot, který má vícevrstvou architekturu - Datovou, Aplikační a Prezentační. Pro lokální vývoj aplikace byl použit Xampp[3], který umožňuje otestovat práci na vlastním počítači bez jakéhokoli přístupu na internet. Aby to bylo co nejjednodušší, je mnoho důležitých funkcí zabezpečení ve výchozím nastavení zakázáno. Databáze se vygeneruje pomocí JPA entit a realizuje koncepci ORM. **Zdrojový kod** je k dispozici v **main** větvi na github[4].

4.1.1 Datová vrstva

Hlavní komponenta pro interakci s databází je Dao (data access object) - spravuje spojení se zdrojem dat za účelem získání a uložení dat, také překládá business objekty(Entity) z nebo do řádků databáze. Každé repository(Dao) pracuje s vlastní třídou Entit, které jsou vygenerovány pomocí JPA entit v spring boot. Na obrázku 3.2 je znázorněn databázový model, který sestává z Entit.

4.1.2 Entity

Ve frameworku Spring boot slouží pro realizaci koncepce ORM třídy z balíku model. Pro vytvoření nové Entity musíme přidat třídu do balíku model a přidat speciální anotace, k tomu také musí dědit z třídy AbstractEntity, jak je to ukázáno na obrázku.

Ke třídě se nastaví název tabulky, ke které má mapovat, k tomu slouží anotace @Table(name = "..."). Pak další anotace @Entity představuje tabulku uloženou v databázi. Každá instance entity představuje řádek v tabulce[20]. Každá entita JPA musí mít primární klíč, který ji jednoznačně identifikuje. Anotace @Id v AbstractEntity třídě definuje primární klíč a zároveň anotace @GeneratedValue generuje primární klíče. Další anotace @MappedSuperclass[22] znamená, že dědičnost je evidentní pouze ve třídě, ale nikoli v modelu entity.

Pak je ve třídě anotace @NamedQueries[23] - staticky definovaný dotaz s předdefinovaným nezměnitelným řetězcem dotazu. Použití pojmenovaných dotazů namísto dynamických dotazů zlepšuje organizaci kódu oddělením řetězců dotazů JPQL od kódu Java. Vynucuje také použití parametrů dotazu místo dynamického vkládání literálů do řetězce dotazů a vede k efektivnějším dotazům.

Pak jsou atributy entity, pomocí anotace @Column můžeme zmínit podrobnosti o sloupci v tabulce. Pro relace mezi entity jsou použité anotace jako @ManyToMany, @OneToMany, @ManyToOne, @OneToOne. Pro čtení a aktualizaci hodnoty atributu slouží gettery a settery. Setter aktualizuje hodnotu proměnné, zatímco getter čte hodnotu proměnné.

[1] https://www.jetbrains.com/idea/
[2] https://git-scm.com/
[3] https://www.apachefriends.org/index.html
[4] https://github.com/algiss00/MoneyExpert_BcPrace

```
@Table(name = "bankAccount_table")
@Entity
@NamedQueries({
        @NamedQuery(name = "BankAccount.getAll", query = "SELECT b FROM BankAccount b"),
        @NamedQuery(name = "BankAccount.getByNameCreated", query = "SELECT b
        FROM BankAccount b " +
                "where b.name = :name and b.creator.id = :uid")
})
public class BankAccount extends AbstractEntity {
    @Column
    private String name;

    @Enumerated(EnumType.STRING)
    private TypeCurrency currency;

    @Column
    private Double balance;

    @ManyToOne
    @JoinColumn(name = "user_id")
    @JsonIgnore
    private User creator;

    @ManyToMany(mappedBy = "availableBankAccounts")
    @JsonIgnore
    private List<User> owners;

    @OneToMany(mappedBy = "bankAccount", cascade = CascadeType.ALL)
    @JsonIgnore
    private List<Transaction> transactions;

    @OneToMany(mappedBy = "bankAccount", cascade = CascadeType.ALL)
    @JsonIgnore
    private List<Budget> budgets;

    @OneToMany(mappedBy = "bankAccount", cascade = CascadeType.ALL)
    @JsonIgnore
    private List<Debt> debts;
```

Ukázka kódu 4.1: Ukázka entity BankAccount

4.1.3 Dao

Třídy dao části, také zvané Repository, jsou v balíku dao - spravují spojení se zdrojem dat a překládají business objekty(Entity) z nebo do řádků databáze. Pro vytvoření Repository musíme přidat třídu do balíku dao a zase, jako u Entit, přidat speciální anotace, a musí dědit z abstraktní třídy AbstractDao, který má v sobě atribut entity manager s anotaci @PersistenceContext.

Instance EntityManager je spojena s persistence context. Persistence context je sada instancí entit, ve kterých pro jakoukoli trvalou identitu entity existuje jedinečná instance entity. V rámci persistence contextu jsou spravovány instance entit a jejich životní cyklus. Rozhraní EntityManager API se používá k vytváření a odebírání instancí trvalých entit, k vyhledávání entit podle jejich primárního klíče a k dotazování nad entitami.[19]

Třída repostitoy má anotaci @Transactional - pro zpracování databázových transakcí[27]. A anotaci @Repository - je specializací anotace @Component, která naznačuje, že anotovaná třída je „Úložiště“, které lze použít jako mechanismus pro zapouzdření chování při ukládání, načítání a vyhledávání, který emuluje kolekci objektů.[24]

V samotné třídě repository jsou implementované metody, nejprve, ty co jsou z abstarctDao - CRUD* metody, pak nějaké speciální dotazy, třeba jak to je ukázáno na 4.2 metoda getByNameAvailableBankAcc.

```
@Repository
@Transactional
public class BankAccountDao extends AbstractDao<BankAccount> {

    BankAccountDao(EntityManager em) {
        super(em);
    }

    @Override
```

```
    public BankAccount find(int id) {
        return em.find(BankAccount.class, id);
    }

    /**
     * get available BankAccount by name
     *
     * @param name - name of BankAcc
     * @param uid  - userId
     * @return
     */
    public List<BankAccount> getByNameAvailableBankAcc(String name, int uid) {
        try {
            return em.createNativeQuery("SELECT acc.name, acc.id,
            acc.balance, acc.currency, acc.user_id " +
                            "FROM bank_account_table as acc inner JOIN
                            relation_bank_account_user as relation " +
                            "ON relation.bank_account_id = acc.id" +
                            " where relation.user_id = :userId and acc.name = :name ",
                    BankAccount.class)
                    .setParameter("userId", uid)
                    .setParameter("name", name)
                    .getResultList();
        } catch (Exception ex) {
            ex.printStackTrace();
            return Collections.emptyList();
        }
    }
```

Ukázka kódu 4.2: Ukázka BankAccountDao

*CRUD - zkrátka od create, read, update, delete, jedná se o životní cyklus datových entit.

4.1.4 Aplikační vrstva (Business layer)

Aplikační vrstva zpracovává veškerou business logiku - je odpovědná za správný přenos dat mezi datovou a prezentační vrstvou. Skládá se ze service objektů, které zahrnují výpočty založené na vstupech a uložených datech, ověřování veškerých dat pocházejících z prezentační vrstvy a přesné zjištění, jakou logiku zdroje dat odeslat, v závislosti na příkazech přijatých z prezentační vrstvy.

Třídy aplikační vrstvy jsou v balíku service - každá service třída dědí z abstraktní třídy AbstractServiceHelper, která v sobě má sdílenou logiku mezi všemi service, jako třeba často používanou metodu getAuthenticatedUser(), která vrací autentizovaného uživatele. Sdílení metod pomáhá pro redukci stejného kódu - ukázka rodičovské třídy na 4.3.

Například TransactionService potřebuje vědět do jakého bankovního účtu se přidá, také musím zkontrolovat pokud ten bankovní účet vůbec existuje. Proto mám sdílenou metodu getByIdBankAccount, která zkusí najít bankAccount z datové vrstvy pomocí dao třídy a pak ještě zkontroluje pokud uživatel je oprávněn do něj přistupovat pomocí metody isUserOwnerOfBankAccount.

Samotná Service třída, má anotaci @Service, která se používá u tříd, které poskytují business logiku. Spring kontext tyto třídy automaticky detekuje, když se použije konfigurace založená na anotacích a skenování cesty ke třídě[26], také anotace @Transactional - pro zpracování databázových transakcí[27].

```
@Transactional
abstract class AbstractServiceHelper {
    protected final UserDao userDao;
    protected final BankAccountDao bankAccountDao;
    protected final TransactionDao transactionDao;
    protected final BudgetDao budgetDao;
    protected final DebtDao debtDao;
    protected final CategoryDao categoryDao;
    protected final NotifyBudgetDao notifyBudgetDao;
    protected final NotifyDebtDao notifyDebtDao;

    private static final Logger log = Logger.getLogger(AbstractServiceHelper.class.getName());

    public AbstractServiceHelper(UserDao userDao,
    BankAccountDao bankAccountDao,
```

```
    TransactionDao transactionDao,
    BudgetDao budgetDao,
    DebtDao debtDao,
    CategoryDao categoryDao,
    NotifyBudgetDao notifyBudgetDao,
    NotifyDebtDao notifyDebtDao) {
        this.userDao = userDao;
        this.bankAccountDao = bankAccountDao;
        this.transactionDao = transactionDao;
        this.budgetDao = budgetDao;
        this.debtDao = debtDao;
        this.categoryDao = categoryDao;
        this.notifyBudgetDao = notifyBudgetDao;
        this.notifyDebtDao = notifyDebtDao;
    }

    /**
     * return Authenticated user.
     *
     * @return
     * @throws NotAuthenticatedClient
     */
    public User getAuthenticatedUser() throws NotAuthenticatedClient {
        try {
            User user = userDao.find(SecurityUtils.getCurrentUser().getId());
            if (user == null) {
                throw new UserNotFoundException();
            }
            return user;
        } catch (Exception ex) {
            ex.printStackTrace();
            throw new NotAuthenticatedClient();
        }
    }
```

Ukázka kódu 4.3: Ukázka AbstractServiceHelper

4.1.5 Zajímavé business logiky z service tříd, které stojí za zmínku.

Metoda **bankAccountLogic** v TransactionService, která slouží pro konzistentnost aktuálního zůstatku v bankovním účtu, také kontroluje pokud přidávaná transakce nepatří k nějakému rozpočtu, a pokud patří tak se zavolá metoda budgetLogic. Metoda se volá po každé co se vytvoří transakce v bankovním účtu. Nejlepší popis jakým způsobem funguje metoda bude procesní diagram 4.1. Diagram je zjednodušen tak, že není tam ukázaná autentizace a autorizace uživatele, ale děje se to na Service straně, pomocí sdílených metod v rodičovské třídě. V diagramu předpokládáme, že uživatel je přihlášený, to jest má autentizační token a má vytvořený bankovní účet. (Bohužel procesní diagram je špatně vidět ale můžete si zkusit zvětšit obrázek při čtení dokumentu v pdf formátu.)

Metody **checkNotifyDates** a **checkDeadlineDates** v DebtService - hledají debts(závazky) u kterých se blíží termín, kdy má aplikace upozornit uživatele, a přidá je do tabulky NotifyDebt. Funguje tak, že pomocí anotaci @Scheduled(cron = "0 0 */6 * * * ") tyhle metody se volají každé 6 hodin(nastavení času se dělá přes speciální cron syntax[41]). Pro hledaní závazků používám SQL dotaz, který porovnává aktuální datum a datum kdy má upozornit. V případě checkNotifyDates, hledám jen ty závazky, u kterých atribut notifyDate je menší nebo roven aktuálnímu datu a zároveň atribute deadline je vetší než aktuální datum, tzn. že dotaz vyhledá jen ty závazky, u kterých se nastal notifyDate. Stejný princip u případu checkDeadlineDates, vyhledá jen ty, u kterých se nastal deadline. Pak všechny závazky, co se mají upozornit, přidávají se do tabulky NotifyDebt. Následně v klientské části, když uživatel se přihlásí do aplikace a přejde do komponenty Dashboard, bude udělán dotaz na NotifyDebt tabulku, pak pokud odpověď bude obsahovat závazky, aplikace vytvoří ikonu oznámení - viz 4.2.8.

Metoda **budgetLogic** v TransactionService - slouží pro sledování stavu Budget(rozpočet), volá se po každé co se přidá transakce typu EXPENSE do bankovního účtu. Funguje tak, že při přidání transakce typu EXPENSE, metoda začne hledat rozpočet ke kterému by mohla patřit transakce, hledá ten rozpočet podle kategorie transakce. Například máme bankovní účet a má rozpočet na kategorii "Jidlo". Přidáváme EXPENSE transakci s kategorií "Jidlo", zavolá se metoda a v ní se najde rozpočet podle kategorie transakce. Přičteme ke atributu

rozpočtu sumAmount(součet částek všech transakcí, které patří do rozpočtu) částku transakci, která se přidala. Následně kontrolujeme pokud součet částek transakcí v rozpočtu (sumAmount) je vetší než částka(Amount) rozpočtu

- pokud ano: přidávám rozpočet do tabulky NotifyBudget - v ní jsou všechny rozpočty, které se mají upozornit.

- pokud ne: kontrolujeme atribut sumAmount, nedošel-li ke určitému procentu od částky rozpočtu(procento se určuje v atributu percentNotify - upozorňuje uživatele kdy procento sumAmount od amount bude vetší nebo roven percentNotif)

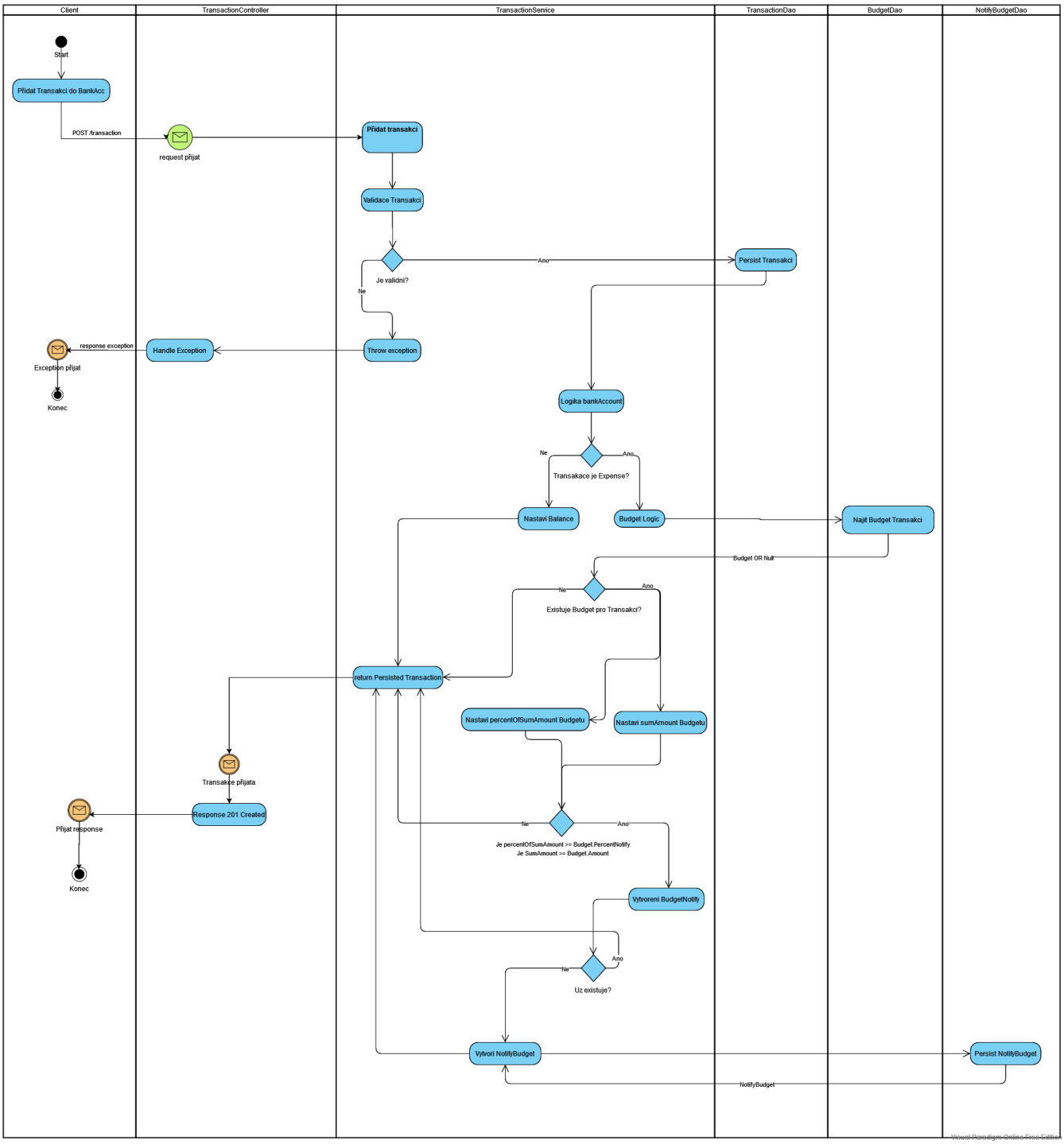
- pokud ani jedno z nich - nic se neděje.

V klientské části to funguje tak, že po každé přidání transakce se udělá dotaz na tabulku NotifyBudget a vyhledá rozpočty uživatele, které se mají upozornit.

Metody **transferTransaction** a **currencyConvertLogic** v TransactionService - první metoda slouží pro převod transakci z jednoho bankovního účtu na jiný v rámci atributů availableBankAccounts a createdBankAccounts v entitě User. Druhá pro správnou konverzi částek transakcí při převodu mezi účty s různými měnami(v tomto projektu máme omezení jen na CZK a EUR měny). Konverze částek se dělá pomocí pevných kurzů, částka se násobí aktuálním kurzem na rok 2021 - 1 euro = 25.32czk, 1 czk = 0.039eur. Před ostrým provozem aplikace bylo by možné předělat na volání API pro získávání aktuálních kurzů.

Sdílení účtu - entita User má atribut availableBankAccounts, jsou tam všechny bankovní účty, které mu sdílely uživatele, tzn. uživatel má k nim přístup. Nejsou tam ale bankovní účty, co uživatel vytvořil, proto je další atribut createdBankAccounts. Sdílení účtu se dělá v metodě addNewOwner v BankAccountService, přidávat jiné uživatele do bank. účtu má právo jen tvůrce, také je zakázáno přidávat již existující uživatele. Vlastník bank. účtu má právo přidávat do něj transakce, rozpočty a závazky, ale nemá právo na odstranění a editovaní údajů bank. účtu - pouze tvůrce má právo na odstranění vlastníka a editovaní údajů bank. účtu.

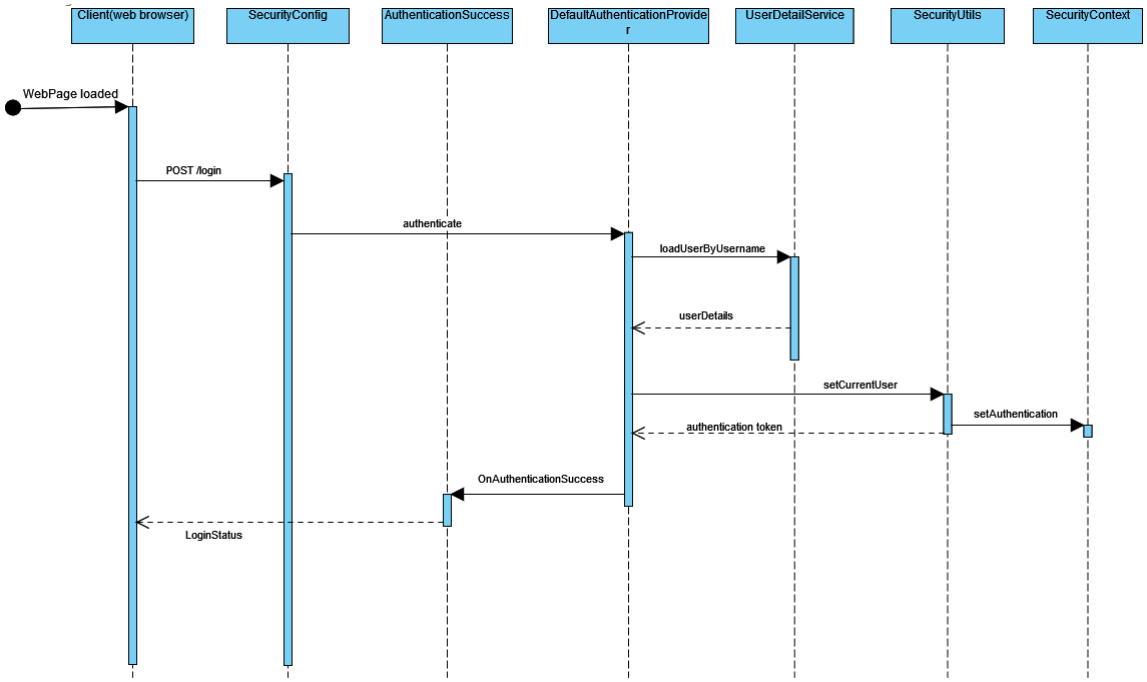
Editace Transakce a vliv na ostatní entity - entita Transaction je v relaci s BankAccount, Budget a Category, tzn. že editace transakce, ovlivní uvedené entity. Uživatel má možnost editovat základní údaje transakce (amount, date, jottings) pomocí metody updateBasic() v TransactionService. Editování atributu amount ovlivní zůstatek bank. účtu a sumAmount rozpočtu ke kterému by mohla patřit transakce. Dále je metoda updateTransactionType(), která slouží pro editaci typu transakci, to zase má vliv na bank. účet a rozpočet - protože pokud změním typ z výdaje na příjem, tak se změní zůstatek bank. účtu, a zároveň sumAmount rozpočtu, protože budu muset odebrat transakci z rozpočtu, pokud k nějakému patřila. Další metodou je updateCategoryTransaction() pro editaci kategorie transakce. Editace kategorie ovlivní Budget a Category, ke které patřila transakce. Pokud transakce patřila k rozpočtu, tak musím z něj ji odebrat, potom zkontrolovat jestli nemusím přidat transakci k rozpočtu, která má editovanou kategorii transakce. Zároveň Category to ovlivní tak, že transakci u ní odeberu a přidám k nové kategorii, tím pádem se změní statistické údaje.



Obrázek 4.1: Procesní diagram bankAccountLogic

4.1.6 Autentizace

Popis autentizace:
1. Klient udělá POST požadavek /login na serverovou část projektu s paramtery username a password.
2. SecurityConfig ten požadavek přijme a zavolá funkci authenticate v autentizačním provideru.
3. Autentizační provider najde uživatele podle username v databázi a zavolá setCurrentUser v SecurityUtils.
4. SecurityUtils nastaví autentizaci a vrátí autentizační token.
5. Zavolá se callback onAuthenticationSuccess, který pošle JSON response jako entitu LoginStatus klientovi, také do hlaviček http odpovědi se přidá atribut set-cookie.



Obrázek 4.2: Sekvenční diagram Autentizace

4.1.7 SecurityConfig

Metoda configure() ve třídě SecurityConfig definuje, které cesty URL by měly být zabezpečeny a které nikoli, také řeší CSRF a CORS útoky. CORS - Cross-Origin Resource Sharing je mechanismus založený na HTTP hlavičce, který umožňuje serveru označit jakýkoli origin (doménu, schéma nebo port) jiný než jeho vlastní, ze kterého by měl prohlížeč povolit načítání zdrojů.[40] V metodě corsConfigurationSource() jsou nastavené povolené originy - http://localhost:8081, http://localhost:8080, https://money-expert.netlify.app. Localhost originy byly použité při lokálním testování. Po nasazení, origin front-endu se změnil na money-expert.netlify.app.

Informace ohledně TODO komentáře, nad příkazem csrf().disable() - CSRF je zkratka pro Cross-Site Request Forgery. Jedná se o útok, který nutí koncového uživatele provádět nežádoucí akce na webové aplikaci, ve které je aktuálně autentizován[81]. Je disable pro jednoduchost komunikace mezi klientem a serverem, aby šlo dobře testovat endpointy v lokálním prostředí. Ukázka SecurityConfig je na 4.4

```java
@Bean
    CorsConfigurationSource corsConfigurationSource() {
        CorsConfiguration configuration = new CorsConfiguration();
        // deployed front-end origin
        configuration.addAllowedOrigin("https://money-expert.netlify.app");
        // origin for test
        configuration.addAllowedOrigin("http://localhost:8080");
        // origin for test
        configuration.addAllowedOrigin("http://localhost:8081");
        configuration.setAllowedMethods(Arrays.asList("GET", "PUT", "POST",
        "PATCH", "DELETE", "OPTIONS"));
        configuration.addAllowedHeader("Content-Type");
        configuration.setAllowCredentials(true);
        UrlBasedCorsConfigurationSource source = new UrlBasedCorsConfigurationSource();
        source.registerCorsConfiguration("/**", configuration);
        return source;
    }

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http
                .cors().and()
                .authorizeRequests().anyRequest().permitAll()
                .and().exceptionHandling()
                .authenticationEntryPoint(new HttpStatusEntryPoint(HttpStatus.UNAUTHORIZED))
                .and().headers().frameOptions().disable()
                .and().authenticationProvider(authenticationProvider)
                // todo disable for tests, but must be enabled in production
                .csrf().disable()
                .formLogin().successHandler(authenticationSuccessHandler)
                .failureHandler(authenticationFailureHandler)
                .loginProcessingUrl(SecurityConstants.SECURITY_CHECK_URI)
                .usernameParameter(SecurityConstants.USERNAME_PARAM)
                .passwordParameter(SecurityConstants.PASSWORD_PARAM)
                .and()
                .logout().invalidateHttpSession(true).deleteCookies(COOKIES_TO_DESTROY)
                .logoutUrl(SecurityConstants.LOGOUT_URI)
                .logoutSuccessHandler(logoutSuccessHandler)
                .and().sessionManagement().maximumSessions(1);
    }
```

Ukázka kódu 4.4: Ukázka SecurityConfig

4.1.8 Prezentační vrstva

Prezentační vrstva zpracovává požadavky HTTP, překládá JSON parametry do objektů entit, ověřuje požadavek a přenáší jej do business(aplikační) vrstvy, také zároveň data z business vrstvy přeposílá klientovi prostřednictvím HTTP odpovědi v JSON formátu.

K tyto vrstvě patří všechny třídy Controllery, které jsou v balíku controller - komunikace mezi serverem a klientem prochází přes něj, pomocí rozhraní REST API. Každá Controller třída má anotaci @RestController a @RequestMapping("/...") - první anotace říká, že ta komponenta je Controller, tzn. zpracovává HTTP požadavky pomocí REST API, pak druhá anotace je určena k mapování HTTP požadavků na metody Controller třídy.

Controller třída má zapouzdřenou Service trídu určité Entity - tzn. že každý Controller je odpovědný za business logiku určité entity. Například BankAccountController má zapouzdřený BankAccountService, to jest odpovídá za business logiku BankAccount Entity. Každá metoda má anotaci mapování jako např. metoda getAllTransactionsFromBankAcc má anotaci @GetMapping(value = "/transactions/{accId}", produces = ...), která znamená, že metoda odpovídá na HTTP požadavky s metodou GET na cestě .../transactions/{accId} - vrátí vše transakce z určitého bankovního účtu.

Seznam endpoitnů je v souboru API.txt v back-end části projektu. Ukázka Controller třídy je na 4.5

```java
@RestController
@RequestMapping("/bank-account")
public class BankAccountController {
    private final BankAccountService bankAccountService;
    private static final Logger log = Logger.getLogger(BankAccountController.class.getName());
```

```
6
7       public BankAccountController(BankAccountService bankAccountService) {
8           this.bankAccountService = bankAccountService;
9       }
10
11      @GetMapping(value = "/{id}", produces = MediaType.APPLICATION_JSON_VALUE)
12      ResponseEntity<?> getAccountById(@PathVariable int id) throws Exception {
13          BankAccount bankAccount = bankAccountService.getByIdBankAccount(id);
14          return new ResponseEntity<>(bankAccount, HttpStatus.OK);
15      }
```

Ukázka kódu 4.5: Ukázka Controller třídy

4.1.9 Komunikace mezi vrstvami

Komunikace mezi Datovou aplikační vrstvou prochází tak, že pokud se v datové vrstvě akce provede úspěšně, tak vrátí entitu aplikační vrstvě, která s ní potom dál pracuje. Pokud akce nebude úspěšná tak vrátí null nebo prázdný list entit, ale také jsou případy kdy může vychodit výjimku.

Výjimky chytají Controllery pomocí metody handleExceptions s anotaci @ExceptionHandler. Při výjimkách controller vrátí uživateli HTTP odpověď s chybovým kódem 400 a zprávou chyby.

Komunikace mezi aplikační a prezentační vrstvou se děje tak, že při úspěchu, aplikační vrstva vrátí prezentační vrstvě data, která požadovala. V opačném případě vychodí výjimku, kterou chytne a zpracuje prezentační vrstva jak bylo popsáno výše. Příklad metody handleExceptions v Controller třídách je na 4.6

```
1  @ExceptionHandler({
2              TransactionNotFoundException.class,
3              BankAccountNotFoundException.class,
4              UserNotFoundException.class,
5              BudgetNotFoundException.class,
6              CategoryNotFoundException.class,
7              DebtNotFoundException.class,
8              NotAuthenticatedClient.class,
9              NotifyBudgetNotFoundException.class,
10             NotifyDebtNotFoundException.class,
11             NotValidDataException.class,
12             Exception.class})
13     void handleExceptions(HttpServletResponse response, Exception exception)
14             throws IOException {
15         log.info(() ->
16                 "REST /account... returned error: " + exception.getMessage());
17         response.sendError(HttpStatus.BAD_REQUEST.value(), exception.getMessage());
18     }
```

Ukázka kódu 4.6: Ukázka handleExceptions

4.2 Klientská část - front-end

V klientské části byl použit framework Vue.js v kombinaci s frameworkem Vuetify, jsou to progresivní frameworky pro vytváření uživatelských rozhraní. **Zdrojový kod** je k dispozici v **front-end** větvi na github[5].

Webová aplikace skládá z komponent a je typu SPA(Single-page application), tzn. že je to v podstatě pouze jedna webová stránka, která neustále komunikuje s uživatelem a dynamicky přepisuje aktuální stránku, spíše než načítání celých nových stránek ze serveru. Cílem jsou rychlejší přechody, díky kterým bude web vypadat více jako nativní aplikace. Zároveň Vue aplikace mají ochranu proti XSS[6] - Vue dokumentace s vysvětlením bezpečnostních rizik je pod odkazem[7].

[5] https://github.com/algiss00/MoneyExpert_BcPrace
[6] https://portswigger.net/web-security/cross-site-scripting
[7] https://vuejs.org/v2/guide/security.html

4.2.1 Struktura Vue aplikace

Pro dobrou představu aplikace je potřeba rozepsat jeho strukturu. Tady je rozepsán obsah nejdůležitější složky - src, která obsahuje jádro celé aplikace Vue:

- main.js - je vstupní bod do aplikace. Tento soubor inicializuje aplikaci Vue. Také je místem, kde se registrují globální komponenty nebo další knihovny Vue.
- views - složka s komponentami.
- api - složka s API požadavky do endpointů serverové části
- router - složka s mapováním cest komponent
- store - složka, kde jsou sdílené proměnné

Následující kapitoly popisují jednotlivé části struktury projektu.

4.2.2 Komponenty - Vue Single File Components (SFC)

Komponenty jsou stránky s aplikační logikou. Z komponent se staví celá webová aplikace. To jsou soubory v formátu .vue, definují se jako Single File Components(SFC) a nacházejí se v složce views, tento speciální formát, umožňuje zapouzdřit šablonu, logiku a styl komponenty Vue do jediného souboru[44]. SFC je přirozené rozšíření klasického tria HTML, CSS a JavaScript: ukázka upravené Login.vue komponenty je na 4.7

```
<template>
   //...
        <v-card class="elevation-12">
            <v-card-text>
                <v-form
                        ref="form"
                        v-model="valid"
                        lazy-validation>
                    <v-text-field
                            id="usernameLogin"
                            label="username"
                            v-model="usernameLogin"
                            :rules="usernameRules"
                            hide-details="auto"
                            required
                    />
                 // ... other fields
                 </v-form>
            </v-card-text>
            <v-card-actions>
                <v-btn color="#e7f6ff" to="/signup"
                class="m2-position">Registrace</v-btn>
            </v-card-actions>
            // ... a td.
</template>

<script>
// import api methods
import {login, getUserByUsername} from "../api";

    export default {
        name: 'login',
        data: () => ({
            usernameLogin: "",
            passwordLogin: "",
            // ... a td.
        }),
        methods: {
             async login(event) {
                // ... login method
            }
        },
</script>
```

```
44
45 <style>
46     .m2-position {
47         position: relative;
48         left: 10px;
49         top: 20px;
50     }
51
52     .m3-position {
53         position: relative;
54         bottom: 33px;
55         right: 10px;
56     }
57 </style>
```

Ukázka kódu 4.7: Ukázka komponenty

Každý soubor *.vue se skládá ze tří typů bloků: <template>, <script> a <style>:

- Sekce <script> je standardní modul JavaScriptu. Měl by exportovat definici komponenty Vue jako svůj výchozí export.
- Sekce <template> definuje šablonu komponenty - vzhled stránky.
- Sekce <style> definuje CSS přidružené ke komponentě.

Vue SFC je formát souboru specifický pro framework a musí být předem zkompilován pomocí @vue/compiler-sfc[8] do standardního JavaScriptu a CSS. Zkompilovaný SFC je standardní modul JavaScript (ES) – tzn., že můžeme importovat SFC jako modul, což je použito při mapování cest komponent v router/index.js 4.8.

Sekce <template> je vybudovaná pomocí frameworku Vuetify, jsou tam použity speciální vuetify tágy typu <v-...>, např. <v-btn>, označující tlačítko.
Zdrojem informace o SFC je [44]. Všechny komponenty aplikace se nacházejí v složce views.

4.2.3 Mapování cest komponent

Máme vytvořenou SPA aplikaci pomocí Vue. Jak bylo uvedeno výše, SPA aplikace dynamicky přepisuje aktuální stránku a k tomu nám slouží Vue Router[45]. Vue Router pomáhá propojit URL/historii prohlížeče a komponenty Vue, což umožňuje určitým cestám vykreslit jakýkoli pohled, který je s ním spojen. Mapování cest a komponenty, které se mají zobrazovat na určitých cestách, jsou určeny v souboru router/index.js - ukázka souboru je na 4.8.

```
1  import Vue from 'vue'
2  // import vue router
3  import VueRouter from 'vue-router'
4
5  // import komponent
6  import Banks from '../views/bankAcc/Banks.vue'
7  import Login from '../views/Login.vue'
8  import SignUp from '../views/SignUp.vue'
9  //... (dalsi komponenty)
10
11 Vue.use(VueRouter)
12
13 // mapovani komponent na cesty
14 const routes = [
15     {
16         path: '/banks',
17         name: 'Banks',
18         component: Banks
19     },
20     {
21         path: '/banks/addBankAcc',
22         name: 'AddBankAcc',
23         component: AddBankAcc
24     },
```

[8] https://github.com/vuejs/vue-next/tree/master/packages/compiler-sfc

```
    //... dalsi cesty
]
// vytvoreni instance routeru
const router = new VueRouter({
    routes,
    mode: 'history'
})

// exportovani
export default router
```

Ukázka kódu 4.8: Ukázka router/index.js

4.2.4 Komunikace mezi klientskou a serverovou části

Na vytvoření požadavků do REST API serverové části slouží soubor api/index.js. Obsahuje metody na volání jednotlivých endpointů serverové části.

Požadavky jsou posílany pomocí knihovny axios - HTTP klient založený na slibech(promise)[46]. Například na ukázce 4.9, v metodě getCategoryByName(...) pomocí axios je posílán asynchronní HTTP GET požadavek na endpoint serveru, který vrátí kategorie uživatele podle názvu. Do parametru konfigurace požadavku je posílán jako "params:"název kategorii, zároveň je nastavený "withCredentials: true"- umožní posílání cookie, který se nastavuje při přihlášení. Pokud axios vychodí výjimku, tak aplikace vypíše do logu webu chybnou zprávu a vrátí null.

```
import axios from "axios";

/**
 * This file contains the requests to the back-end part
 */

const url = "https://money-expert-bc.herokuapp.com";

/**
 * get users category by name
 * @param name
 * @returns {Promise<null|T>}
 */
export async function getCategoryByName(name) {
    try {
        let result = await axios.get(`${url}/category/user-by-name`, {
            params: {
                name: name,
            },
            withCredentials: true
        })
        return result.data
    } catch (error) {
        if (error.response) {
            console.log(error.response.data.message);
        }
        return null
    }
}
// ...dalsi metody
```

Ukázka kódu 4.9: Ukázka api/index.js

4.2.5 Správa stavu aplikace

Pro správu stavu(state) aplikace byla použita knihovna Vuex[47], která slouží jako centralizované úložiště pro všechny součásti v aplikaci s pravidly, která zajišťují, že stav lze mutovat pouze předvídatelným způsobem.

Soubor store/store.js je v podstatě kontejner, který uchovává stav aplikace. Existují dvě věci, které odlišují Vuex úložiště(store) od obyčejného globálního objektu:

- Úložiště Vuex jsou reaktivní. Když z něj Vue komponenty získají stav, budou se reaktivně a efektivně aktualizovat, pokud se stav úložiště změní.

- Nelze přímo změnit stav úložiště. Jediným způsobem, jak změnit stav úložiště, je explicitní provedení mutací. To zajišťuje, že každá změna stavu zanechá sledovatelný záznam[48].

Na ukázce 4.10 v state jsou reaktivní sdílené proměnné a v mutations jsou metody pro změny stavu proměnných. Například v komponentě Login.vue při úspěšném přihlášení se volá mutation metoda setUser, která nastaví aktuálně přihlášeného uživatele - ukázka kódu je na 4.11. Zároveň je v ukázce vidět jak se nastavuje snackbarError, při neúspěšném přihlášení. Nejdříve se nastaví chybový text pro uživatele a potom se objeví samotný snackbar nastavením proměnné na true.

```
import Vuex from 'vuex'
import Vue from 'vue'

Vue.use(Vuex)

/**
 * this js file contains shared data
 */

const store = new Vuex.Store({
    state: {
        user: null,
        snackbar: false,
        snackbarError: false,
        //...dalsi objekty
    },
    mutations: {
        setUser(state, user) {
            state.user = user
        },
        setSnackbar(state, status) {
            state.snackbar = status
        },
        setSnackbarError(state, status) {
            state.snackbarError = status
        },
       //...dalsi metody
    }
})
export default store
```

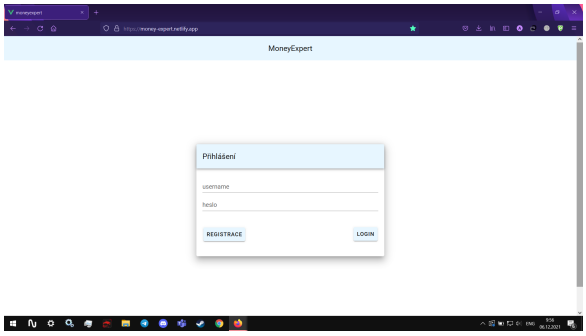
Ukázka kódu 4.10: Ukázka store/store.js

```
    methods: {
            async login(event) {
                if (!this.$refs.form.validate()) {
                    event.preventDefault()
                    return
                }
                this.loading = true
                let result = await login(this.usernameLogin, this.passwordLogin)
                if (result.loggedIn === true
                && result.success === true
                && result.username === this.usernameLogin) {
                    let user = await getUserByUsername(this.usernameLogin)
                    // set actual user to store
                    this.$store.commit("setUser", user)
                    await this.$router.push('/banks').catch(() => {
                    })
                } else {
                    if (result.errorMessage) {
                        // set error text
                        this.$store.commit("setSnackbarText", result.errorMessage)
                        // view snackbar
                        this.$store.commit("setSnackbarError", true)
                    }
                }
                this.loading = false
            }
        },
```

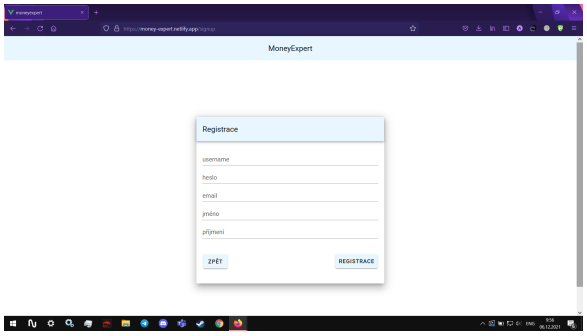
Ukázka kódu 4.11: Ukázka Login.vue

4.2.6 Přehled obrazovek klientské části

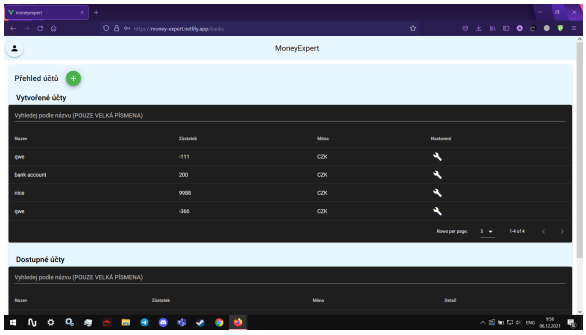
Kapitola zahrnuje základní obrazovky webové aplikace MoneyExpert. Design byl udělán podle prototypu aplikace, viz kapitola 3.0.3.



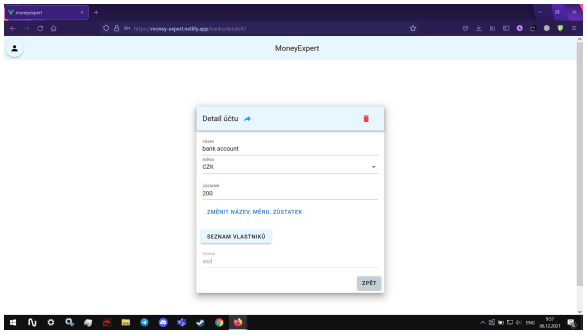
Obrázek 4.3: Přihlášeni



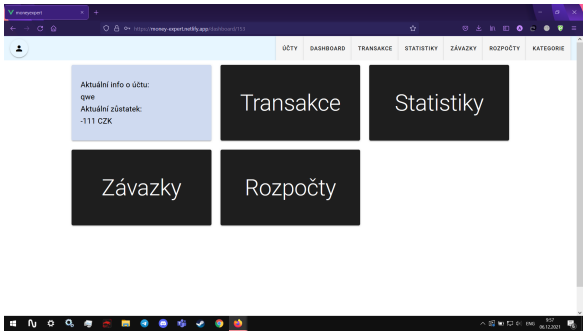
Obrázek 4.4: Registrace



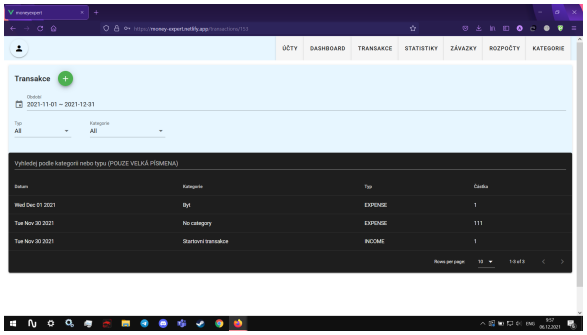
Obrázek 4.5: Bankovní účty



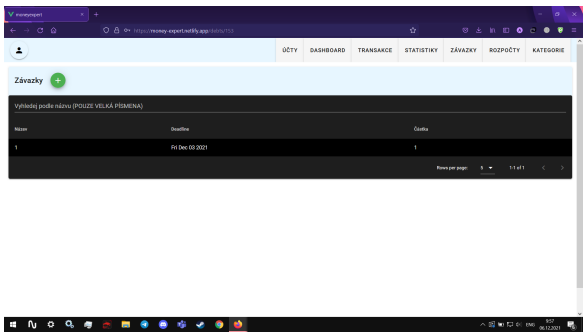
Obrázek 4.6: Detail bankovní účet



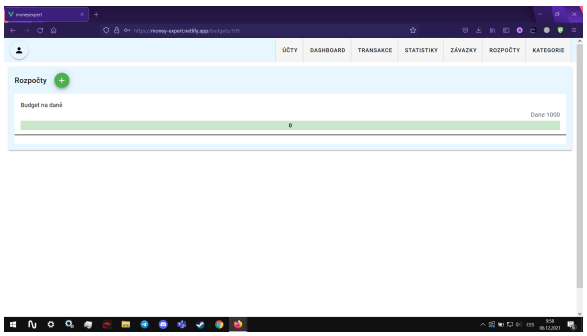
Obrázek 4.7: Dashboard



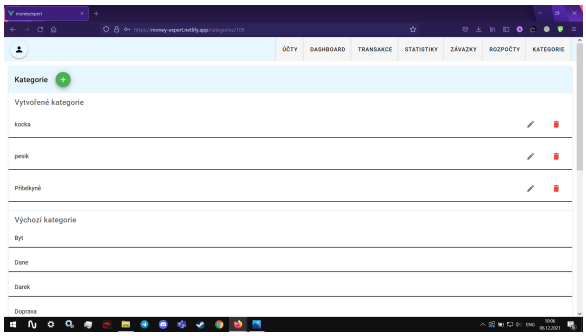
Obrázek 4.8: Transakce



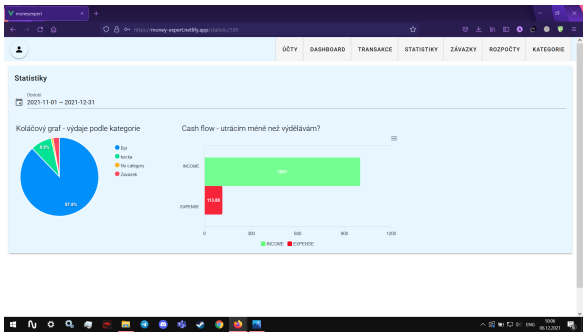
Obrázek 4.9: Závazky



Obrázek 4.10: Rozpočty



Obrázek 4.11: Kategorie



Obrázek 4.12: Statistika

Rozdíl výsledných obrazovek od prototypu:
Obrazovka Dashboard 4.7 - neodpovídá designu prototypu, ale svou funkci pro navigaci splňuje, bude grafické zlepšen v budoucnu.
Obrazovky na detaily entit (transakce, bank. účet, závazek, rozpočet) jsou jako na 4.6. V prototypu detail entity se zobrazuje jako side sheet[9]. Ve výsledné obrazovce to bylo uděláno jinak než v prototypu, jednak kvůli jednoduchosti a jednak rozdělení komponent pomáhá udržovat přehledný kód.
U obrazovek se seznamem dat je možnost vyhledávání podle určitých parametrů ale u rozpočtů a kategorií není. Vyhledávání je implementováno pomocí Vuetify komponenty <v-data-table>, mezi její funkce patří řazení, vyhledávání, stránkování, úpravy obsahu a výběr řádků[63].
U obrazovek 4.12 a 4.8 je možnost filtrování podle období. Je implementováno pomocí Vuetify komponenty <v-date-picker> - je plně funkční komponenta pro výběr data, která uživatelům umožňuje vybrat datum nebo rozsah dat[64].
Poznámka: znázorněné obrazovky jsou před opravami, které vznikly po uživatelském testování viz 5.3

4.2.7 Statistické grafy

Ukázka komponenty Statistic.vue odpovídající statistickým grafům je na obrázku 4.12. Framework Vue umožňuje jednoduchou integraci knihoven, právě na vykreslování grafů byla použita knihovna ApexChart[10].

V souboru main.js, je definovaná globální komponenta VueApexCharts s názvem "apexchart", která je obalová komponenta pro ApexCharts[57]. Globální komponenta je přidaná speciálním tágem <apexchart> - ukázka je na 4.12.

```
1            // globalni komponenta ApexCharts - kolacovy graf
2          <apexchart type="pie" width="380" :options="chartOptions"
3                                  :series="sumOfExpensesCategoryBetweenDate"/>
```

Ukázka kódu 4.12: Ukázka apexchart Statistic.vue

Jak to je vidět na ukázce tág má 4 parametry:

- type - typ grafu, na ukázce je koláčový graf

[9] https://material.io/components/sheets-side
[10] https://apexcharts.com/

- width - šířka grafu
- :options - konfigurace grafu
- :series - data, ze kterých je sestaven graf

Přejdeme na JavaScript část 4.13.

```
export default {
    name: "Statistic",
    data: () => {
            return {
            // ... dalsi data

             // :series
             sumOfExpensesCategoryBetweenDate: [],
             // :options
                chartOptions: {
                // nazvy polozek
                    labels: [],
                    chart: {
                        width: 380,
                        type: 'pie',
                    },
                    responsive: [{
                        breakpoint: 480,
                        options: {
                            chart: {
                                width: 200
                            },
                            legend: {
                                position: 'bottom'
                            }
                        }
                    }]
                },
            }
    },
    methods: {
     // ...metody
    },
    async mounted() {
        // ...dalsi kod

         let categories = await getAllUsersCategories()
         // ...dalsi kod

         // get sum of expenses for category
            for (let i = 0; i < categories.length; i++) {
                // posilani pozadavku na endpoint
                let sumExpenseCategory = await getSumOfExpenseForCategoryBetweenDate(
                this.bankIdStatistic,  from, to, categories[i].id)
                if (sumExpenseCategory == null) {
                    return
                }
                if (sumExpenseCategory !== 0) {
                    // pridani castky transakci na urcitou kategorii
                    this.sumOfExpensesCategoryBetweenDate.push(sumExpenseCategory)
                    // pridani nazvu polozky v grafu
                    this.chartOptions.labels.push(categories[i].name)
                }
            }

            // get sum of expenses and incomes from bankAcc between date
            let sumExpense = await getSumOfExpenseBetweenDate(this.bankIdStatistic, from, to)
            let sumIncome = await getSumOfIncomeBetweenDate(this.bankIdStatistic, from, to)

            // ...dalsi kod
    }
}
```

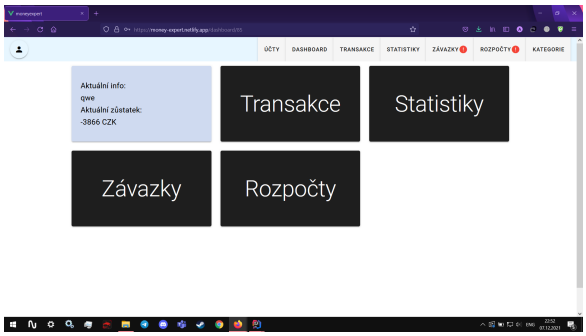
Ukázka kódu 4.13: Ukázka JavaScript Statistic.vue

V části data se definují parametry pro apexchart, pak v části mounted()* se nastaví data na vykreslování grafů. Pro koláčový graf(výdaje podle kategorie) data se načítají procházením všech kategorií uživatele, dále na každou kategorii se posílá požadavek do endpointu na počítání částek transakcí v určitém období. Pro graf cash flow se posílá požadavek do endpointu na počítaní částek výdajů a příjmů v určitém období. Je dobrý znázornit, že posílaní požadavků pro jednotlivé kategorie uživatele není dobré řešení, protože to hodně zatěžuje systém, je potřeba to vyřešit před ostrým provozem. Bohužel na ten problém jsem přišel až po nasazení aplikace.
*Poznámka k části mounted() - je to část životního cyklu Vue aplikace, volá se když je komponenta přidána do DOM. Nejčastěji se používá k odeslání požadavku HTTP na načtení dat, která pak komponenta vykreslí[58]

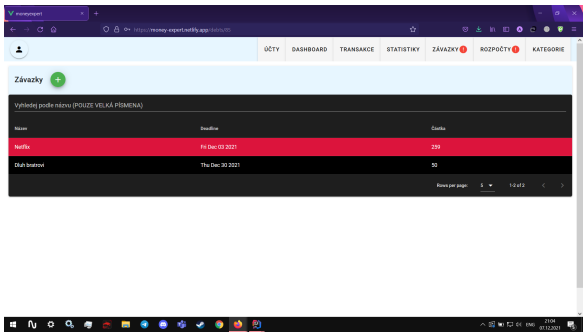
4.2.8 Upozornění uživatele

V aplikaci MoneyExpert je implementovaná funkce na upozornění uživatele, jak to bylo uvedeno v případu užití UC21 - viz 2.1.4. Po přechodu do komponenty dashboard se posílá požadavek na endpoint, který vrátí závazky a rozpočty, které mají být upozorněné v bankovním účtu.

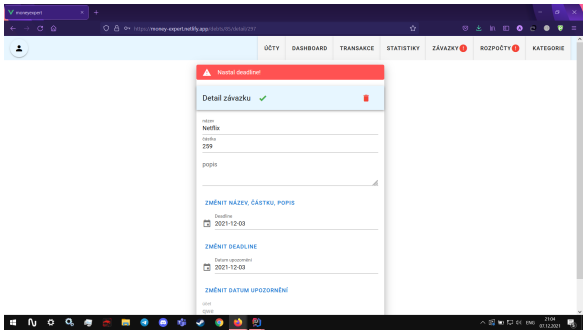
Upozornění se objeví jako ikona oznámení v navigačním menu(obrázek 4.13), a také v seznamech rozpočtů a závazků - položky, které mají být upozorněné obarví se červenou barvou. V případě závazků - text upozornění bude v detailu položky, je tam také možnost ukončení závazku. Ukončení závazku má následky, které budou vysvětleny v dialog boxu při stisknutí na tlačítko ukončení(obrázek 4.16). V případě rozpočtů - text upozornění je na samotné položce, jak je to ukázáno na obrázku 4.17.



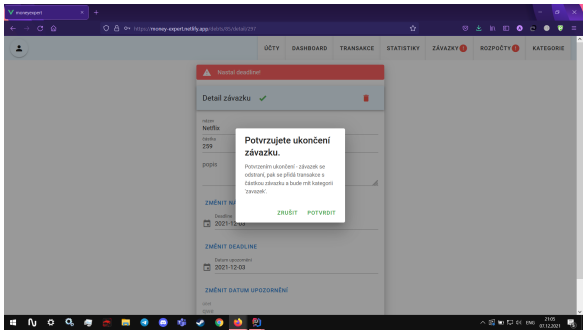
Obrázek 4.13: Ikony oznámení v navigačním menu



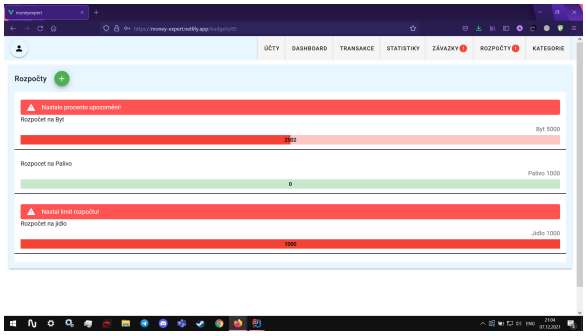
Obrázek 4.14: Upozornění závazků



Obrázek 4.15: Detail závazku



Obrázek 4.16: Ukončení závazku



Obrázek 4.17: Upozornění rozpočtů

Zobrazení ikony oznámení se dělá pomocí Vue Directives - jsou to speciální atributy s prefixem "v-". Očekává se, že hodnoty atributů Directives budou jedním JavaScriptovým výrazem. Úkolem Directives je reaktivně aplikovat vedlejší efekty na DOM, když se změní hodnota jejího výrazu[61]. Na ukázce 4.14 je použit Directives "v-if=...", funguje tak, že pokud je výraz true tak se zobrazí element v DOM, v opačném případě se schová.

```
<v-alert
       dense
       type="error"
       v-if=" isBudgetInPercentNotification(item) === true
           && isBudgetInAmountNotification(item) === false">
       // pokud rozpocet ma byt upozornen pouze
       // podle typu BUDGET_PERCENT, tak se zobrazi hlaska
   Nastalo procento upozorneni!
</v-alert>
```

Ukázka kódu 4.14: Ukázka ikony oznámení Budget.vue

4.2.9 Návod na použití

Návod na použití je na videu pod odkazem[11].

4.3 Nasazení

Aplikace je rozdělená na dvě části - serverovou a klientskou. Každá část je nasazená na Cloud Platform[12]. Serverová část je nasazená na Heroku[13] - je Cloud platforma PaaS[14](platform as a service). Používá se Heroku k nasazení, správě a škálování moderních aplikací. Platforma je flexibilní a snadno použitelná[51]. Heroku používá model kontejneru ke spouštění a škálování všech aplikací Heroku. Kontejnery používané v Heroku se nazývají „dynos". Dynos jsou izolované, virtualizované Linux[15] kontejnery, které jsou navrženy tak, aby spouštěly kód na základě příkazu zadaného uživatelem[56]. Pro bezplatnou verzi Heroku je omezení práce Dynos - po 30 minutách žádné aktivity usne. Heroku pomáhá vytvářet, a provozovat aplikace výhradně v cloud. Zaměřuje se na back-end vývoj a umožňuje nasazení webových projektů. Mezi hlavní funkce patří předkonfigurované kontejnery Dynos a plně spravované runtime prostředí. Výhodou jsou škálovatelnost a bezpečnostní protokoly.
Popis nasazení serverové části na Heroku:

Prvním krokem bylo hostování databáze, před nasazením se používala lokální MySQL databáze, pomocí nástroje Xampp[16]. Pro hostování databáze byla stáhnutá bezplatná varianta doplňku Heroku Postgres[17] - je spravovaná SQL databázová služba poskytovaná přímo Heroku. K databázi Heroku Postgres lze přistupovat z jakéhokoli jazyka s ovladačem PostgreSQL, včetně všech jazyků oficiálně podporovaných Heroku[52]. Bezplatná varianta "Hobby Dev"má omezení na počet řádků v databázi(max 10000) a úložiště je 1 GB, další omezení jsou uvedeny v dokumentaci Heroku Postgres[18].

Dále serverová část byla přenastavená na připojení k hostované databázi, pak nasazená na heroku. Nasazení proběhlo velmi jednoduché, heroku má možnost integrace s github aplikace a umožňuje automatické nasazení[53], tzn. že heroku sestaví a nasadí všechna push do určité větve (v našem případě do větve main, kde se nachází serverová část). Serverová část je nasazená na adrese https://money-expert-bc.herokuapp.com, jsou tam posílány požadavky na jednotlivé endpointy z klientské části, viz 4.2.4.

Klientská část je nasazená na Netlify[19] - je cloud platforma pro webhosting, která se zaměřuje na front-end, mezi hlavní funkce patří integrace s github a bezplatný HTTPS na všech stránkách, včetně automatického vytváření a obnovování certifikátů. Netlify zjednodušuje proces nasazení a hostování webových stránek[54]. Jak bylo uvedeno Netlify jako Heroku umožňuje integraci s github a také automatické nasazení, pro klientskou část je nastvená větv front-end[55]. Přezkoumat klientskou část aplikace MoneyExpert lze na https://money-expert.netlify.app/.

[11] https://drive.google.com/file/d/1mzc9iYk05Z_5xT3XyrZQDd52yMRFCCiE/view?usp=sharing
[12] https://www.cloudbolt.io/what-is-a-cloud-platform/
[13] https://www.heroku.com
[14] https://searchcloudcomputing.techtarget.com/definition/Platform-as-a-Service-PaaS
[15] https://www.linux.org/
[16] https://www.apachefriends.org/index.html
[17] https://elements.heroku.com/addons/heroku-postgresql
[18] https://devcenter.heroku.com/articles/heroku-postgres-plans
[19] https://www.netlify.com/

Po nasazení aplikace jsem narazil na problém s cookie v prohlížecí Google Chrome. Problém byl v tom, že od verze 80 Google Chrome byl spuštěn s novým výchozím nastavením pro atribut SameSite cookie. Tyto změny ovlivňují sledování souborů cookie třetích stran*[65].

Atribut SameSite informuje prohlížeče, kdy a jak spouštět soubory cookie v situacích první nebo třetí strany. SameSite se používá v různých prohlížečích k identifikaci, zda povolit nebo nepovolit přístup k souboru cookie. Hodnoty pro atribut SameSite - strict, lax, none:

- lax - umožňuje odesílání/přístup pouze cookies první strany*
- strict - je podmnožinou lax a nespustí se, pokud je příchozí odkaz z externího webu
- none - signalizuje, že data cookie mohou být sdílena s třetími stranami*/externími stránkami (pro reklamu, vložený obsah atd.)

Pokud dříve nebyl SameSite nastaven, ve výchozím nastavení byl nastaven na none – bylo povoleno sdílení třetí stranou. Nyní, pokud není nastaven atribut SameSite, výchozím nastavením prohlížeče Chrome je lax - nastavení první strany. Konkrétně Chrome nyní zpracovává všechny soubory cookie bez nového "SameSite=None; Secure"atributu jako SameSite=Lax, který je omezuje na kontexty první strany. Tato úprava byla udělána se záměrem na ochranu osobních údajů[65].

Serverová a klientská část jsou nasazeny na různých doménách, proto při přihlášeni, serverová část posílá cookie do třetí strany(klientská část). Problém byl v tom, že nebyl nastavený atribut SameSite při posílaní cookie a Google Chrome přidával do atributu výchozí hodnotu lax. Problém byl vyřešen, nastavením atributu "SameSite=None; Secure"při posílání cookie. Cookie s atributem Secure je odeslán na server pouze se zašifrovaným požadavkem přes protokol HTTPS. Nezabezpečené weby (s http: v adrese URL) nemohou nastavit soubory cookie s atributem Secure. Atribut Secure je požadován, pokud je SameSite=None[67]. Konfigurace cookie je ve třídě LegacyCookieProcessorConfiguration a souboru application.properties serverové části. Aktuálně aplikace funguje na prohlížečích Google Chrome, Mozilla FireFox a Microsoft Edge.

*Cookie první strany (First-party) ukládá doména (webová stránka), kterou přímo navštěvujete. Umožňují vlastníkům webových stránek shromažďovat analytická data, pamatovat si jazyková nastavení a provádět další užitečné funkce, které pomáhají poskytovat dobré uživatelské prostředí[66].
*Cookie třetích stran (Third-party) jsou vytvářeny jinými doménami než doménami, které přímo navštěvujete[66].

Kapitola 5

Testování

Tato kapitola popisuje proces testování aplikace prováděného během implementace a po něm. Testování je rozděleno do tří typů testů – jednotkové testy, systémové testy, a uživatelské testování. Následující části popisují každý test, jak se provádí a jaké jsou výsledky.

5.1 Jednotkové testy - Unit tests

Unit test je typ testování aplikace, při kterém se testují jednotky nebo komponenty aplikace. Účelem je ověřit, že každá jednotka kódu funguje podle očekávání. Jednotkové testy izolují část kódu a ověřují její správnost. Unit neboli jednotka může být funkce, metoda, model nebo objekt[59]. Jednotkové testy jsou v serverové části projektu, byly testovány dao a service vrstvy. Celkově jednotkových testů je 139.

5.1.1 Dao testy

Testy dao vrstvy se nacházejí v balíku "dao"složky "test"back-end projektu. Pro testování jako databáze byla použitá "H2 DB"databáze v paměti a mock data ze souboru test-data.sql. H2 DB eliminuje potřebu konfigurace a spouštění skutečné databáze pro testovací účely[68]. V dao vrstvě byly testovány základní CRUD funkce dao tříd. Každé dao testy mají anotace:

- @ExtendWith(SpringExtension.class) - integruje Spring TestContext Framework do programovacího modelu Jupiter JUnit 5[69].
- @DataJpaTest - zaměřuje se pouze na komponenty JPA. Tato anotace zakáže plnou automatickou konfiguraci a místo toho použije pouze konfiguraci relevantní pro testy JPA. Vyhledává třídy @Entity a konfiguruje úložiště Spring Data JPA anotované anotací @Repository[70].
- @AutoConfigureTestDatabase - používá se ke konfiguraci testovací databáze, která se má používat namísto aplikací definovaného nebo automaticky konfigurovaného DataSource[71].
- @ComponentScan - umožňuje skenování komponent v aplikaci Spring[72].
- @Sql - deklarativní způsob naplnění testovací databáze[73].
- Testované entity mají anotace @Autowired - vkládá závislost, souvisí se spring dependency injection*[74].

Na ukázce 5.1 je znázorněn test pro třídu BankAccountDao. Metoda findBankAccount() testuje funkci find() dao vrstvy entity BankAccount.

*Dependency injection - technika při které jeden objekt dodává závislost jiného objektu. Závislost je objekt, který lze použit. Injekce je předání závislosti.

```java
@ExtendWith(SpringExtension.class)
@DataJpaTest
@AutoConfigureTestDatabase(connection = EmbeddedDatabaseConnection.H2)
@ComponentScan(basePackageClasses = MoneyExpertApplication.class)
@Sql("classpath:test-data.sql")
public class BankAccountDaoTest {
    @Autowired
    private BankAccountDao bankAccountDao;

    @Test
    public void findBankAccount() {
        BankAccount bankAccount = bankAccountDao.find(15);
        assertNotNull(bankAccount);
        assertEquals("CSOB Ucet", bankAccount.getName());
        assertEquals(TypeCurrency.CZK, bankAccount.getCurrency());
        assertEquals(1000, bankAccount.getBalance(), 0.0);
    }
```

Ukázka kódu 5.1: Ukázka jednotkového testu

5.1.2 Service testy

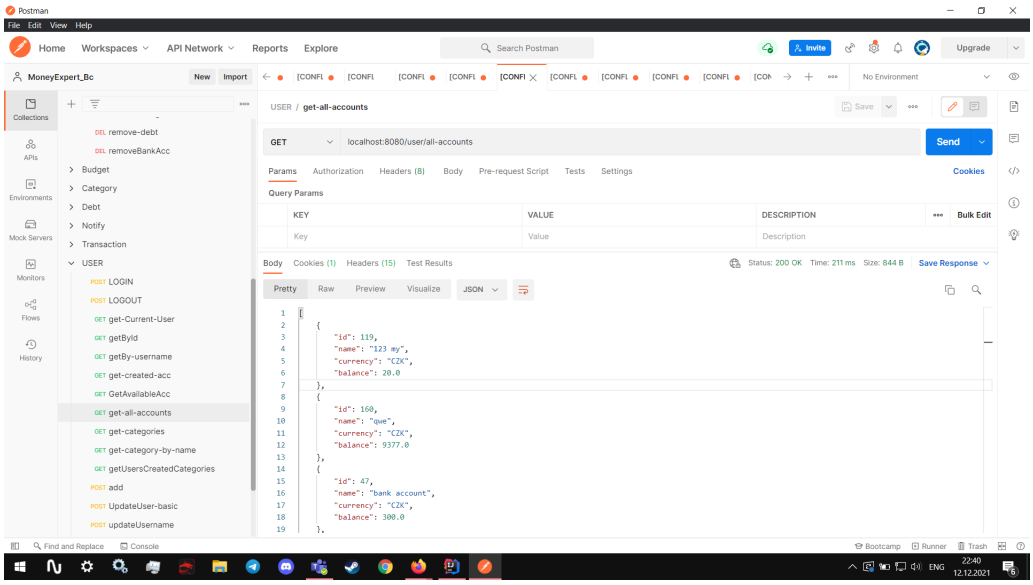
Pro service vrstvu byly použité mock objekty - vrací předefinovaná data, konfigurovaná pro daný test a zároveň umožňují registrovat interakci s objektem. Na ukázce 5.2 je testovaná UserService třída. Před každým testem se volá setUp() metoda, která vytváří testovanou třídu znova, v tyto metodě se vytváří mock objekty pro konstruktor UserService třídy. Například v testu vytváření uživatele, testujeme funkci persist(), po volání funkce musíme si jenom ověřit, zda funkce proběhla podle očekávaní. V našem případě musela se zavolat metoda persist() u dao objektu userDao, nás nezajímá pokud user se přidal do databáze. Pomocí metody verify() ověříme, zda očekávaná metoda mock objektu byla volaná jedenkrát a měla jako parametr očekávaný objekt(user).
Stejným principem byly napsáné ostatní service testy. Testy se nachazejí v balíku "service"složky "test"back-end projektu.

```java
@ExtendWith(SpringExtension.class)
@ComponentScan(basePackageClasses = MoneyExpertApplication.class)
public class UserServiceTest {

    private UserService userService;
    private final PasswordEncoder encoder = PasswordEncoderFactories
    .createDelegatingPasswordEncoder();
    private UserDao userDao;

    @BeforeEach
    public void setUp() {
        userDao = mock(UserDao.class);
        BankAccountDao bankAccountDao = mock(BankAccountDao.class);
        TransactionDao transactionDao = mock(TransactionDao.class);
        BudgetDao budgetDao = mock(BudgetDao.class);
        DebtDao debtDao = mock(DebtDao.class);
        CategoryDao categoryDao = mock(CategoryDao.class);
        NotifyBudgetDao notifyBudgetDao = mock(NotifyBudgetDao.class);
        NotifyDebtDao notifyDebtDao = mock(NotifyDebtDao.class);

        userService = new UserService(userDao, bankAccountDao, transactionDao,
                budgetDao, debtDao, categoryDao,
                notifyBudgetDao, notifyDebtDao, encoder);
    }

    @Test
    public void persist_mockTest_success() throws Exception {
        User user = Generator.generateDefaultUser();
        userService.persist(user);
        verify(userDao, times(1)).persist(user);
    }
```

Ukázka kódu 5.2: Ukázka mock testu

5.2 Systémové testy

Během těchto testů je aplikace ověřována jako funkční celek. Tyto testy jsou používány v pozdějších fázích vývoje. Ověřují aplikaci z pohledu zákazníka. Podle připravených scénářů se simulují různé kroky, které v praxi mohou nastat. Obvykle probíhají v několika kolech. Nalezené chyby jsou opraveny a v dalších kolech jsou tyto opravy opět otestovány[75]. V našem případě byly testované endpointy serverové části v lokálním prostředí během vývoje. Tyto testy byly udělané manuálně pomocí Postman[1] - nástroj pro testování API, jedná se o klienta HTTP, který testuje požadavky HTTP v grafickém uživatelském rozhraní, pomocí kterého získáváme různé typy odpovědí, které následně ověříme. Testy pokrývaly obsah funkce aplikace. Pak po testování samotné serverové části, aplikace byla otestovaná jako celek, totiž byla manuálně otestována klientská část spolu se serverovou. V budoucnu předpokládám napsat automatizované systémové testy.



Obrázek 5.1: Ukázka Postman - testování endpointu

5.3 Uživatelské testování

Uživatelské testování nebo také testování použitelnosti je metoda používaná k hodnocení toho, jak je snadné použití webové stránky. Testy probíhají se skutečnými uživateli, aby se zjistilo, jak web je „použitelný“ nebo „intuitivní“ a jak je snadné pro uživatele dosáhnout svých cílů.[76].

Pro testování byli osloveni uživatele projevující zájem o používání aplikace pro správu osobních financí. Před testováním s každým testerem byly sdíleny základní funkce aplikace. Testeři postupovali podle testovacího scénáře 5.3.1 - jsou to úkoly pro každodenní používání aplikace. Tyto úkoly by měly navíc ověřit splnění funkčních požadavků. Testování probíhalo samostatně a každý tester musel přemýšlet nahlas, pro sledování jejich chování a myšlenek.

5.3.1 Testovací scénář

Tady jsou rozepsané jednotlivé úkoly a předpokládané chovaní testera. Tester má k dispozici pouze úkoly, nikoli jejich popis nebo očekávané použití, aby nebyla omezena svoboda volby testera a používaní aplikace bylo více přirozené.

1. **Zaregistrujte se**

Tester vyplní registrační formulář.

[1] https://www.postman.com/

2. **Přihlaste se**

Tester se přihlásí.

3. **Vytvořte bankovní účet**

Tester vytvoří nový bankovní účet a stisknutím na něj přejde do obrazovky dashboard.

4. **Přidejte pár transakcí**

Tester přejde do seznamu transakcí vybraného bank. účtu a vytvoří několik transakcí s různými částkami a kategoriemi.

5. **Upravte libovolnou transakci**

Tester vybere nějakou transakci a upraví její název, kategorii a částku.

6. **Vyfiltrujte transakce podle období/typu/kategorii**

V seznamu transakcí tester vyfiltruje je podle typu, kategorii a období

7. **Vytvořte závazek, kategorii a rozpočet**

Tester se zorientuje v aplikaci a vytvoří závazek, kategorii, rozpočet.

8. **Vytvořte transakci, která bude patřit do vytvořeného rozpočtu**

Tester vytvoří transakci s kategorií, patřící k vytvořenému rozpočtu.

9. **Zkontrolujte stav rozpočtů**

Tester přejde do seznamu rozpočtů a všimne si, že se stav rozpočtu změnil.

10. **Prozkoumejte detaily bankovního účtu, transakce, závazku a rozpočtu**

Tester projde vytvořené entity a prozkoumá jejich detaily.

11. **Ukončete libovolný závazek**

Tester ukonči vytvořený závazek.

12. **Zkontrolujte změny po ukončení závazku**

Tester si všimne změny po ukončení závazku.

13. **Podívejte se do statistiky**

Tester přejde do sekce statistiky a pochopí ukázané statistické údaje.

14. **Vytvořte nový bankovní účet a udělejte do něho převod transakce**

Tester vytvoří nový bank. účet, přejde do seznamu transakcí, vytvořeného před tím bank. účtu a zvládne udělat převod vybrané transakce do nového bank. účtu.

15. **Zkontrolujte správnost převodu transakce**

Po převodu transakce, tester přejde do seznamu transakcí bank. účtu, kam byl udělán převod, a zkontroluje pokud tam je převedená transakce.

16. **Vyhledejte libovolný vytvořený bankovní účet podle názvu**

V seznamu bank. účtů, tester vyhledá podle názvu libovolný vytvořený bank. účet.

17. **Sdílejte libovolný bankovní účet s uživatelem "algis"**

Tester otevře detail vybraného bank. účtu a zvládne ho sdílet s uživatelem algis.

18. **Zkontrolujte pokud uživatel algis je v seznamu vlastníků**

V detailu bank. účtu, tester zvládne otevřít seznam vlastníků a ověřit, zda je tam uživatel algis.

19. **Odhlaste se**

Tester se odhlásí z aplikace.

5.3.2 Výsledek testování

Testování použitelnosti se zúčastnili čtyři testeři ve věku 20, 24, 23, 41. Jeden z nich měl zkušenosti s používáním podobné aplikace, zatímco zbytek byli uživatelé, kteří se chtěli s takovou aplikací zapojit.

Cílem uživatelského testovaní bylo ověřit zda je aplikace dobře použitelná a splňuje své funkční požadavky. Pro hodnocení použitelnosti vyplňoval každý tester speciální dotazník[2], který byl vytvořen metodou System Usability Scale (SUS), která poskytuje spolehlivý nástroj pro měření použitelnosti. Skládá se z 10 položkového dotazníku s pěti možnostmi odpovědi pro respondenty; od Silně souhlasím do Silně nesouhlasím. Původně ji vytvořil John Brooke v roce 1986 a umožňuje hodnotit širokou škálu produktů a služeb, včetně hardwaru, softwaru, mobilních zařízení, webových stránek a aplikací[79]. Výsledné průměrné hodnocení aplikace je 71.8, což je podle tabulky hodnocení SUS známka B - dobře použitelná.

Tabulka hodnocení SUS[80]:

SUS Score	Letter Grade	Adjective Rating
Above 80.3	A	Exellent
Between 68 and 80.3	B	Good
68	C	OK
Between 51 and 67	D	Poor
Below 51	F	Awful

Tabulka 5.1: Hodnocení SUS

Díky dotazníku a zpětné vazbě testerů se aplikace ukázala jako použitelná a jasná, funkční požadavky byly také splněny. Testeři sice byli schopni úspěšně dokončit všechny úkoly, ale během testování se objevilo několik problémů.
Níže jsou uvedeny problémy:

- Aplikace neumožňovala vytvářet bankovní účty s negativním zůstatkem. To by se hodilo uživatelům, kteří mají dluhy na účtu, proto to bylo opraveno.
- Vyhledávání podle názvu bylo možné pouze s velkými písmeny, což pro uživatele nebylo intuitivní; proto to bylo opraveno na možnost vyhledávaní s malými písmeny.
- Při znovu načtení stránky se objevila chyba 404, testera to zmátlo. Problém byl opraven tak, že v aktuální verzi při znovu načtení, uživatele převede do startovní obrazovky. Jak bylo uvedeno v kapitole výše, klientská část je typu SPA, proto se předpokládá, že uživatel moc nepotřebuje znovu načtení stránky.
- Při převodu transakce se ukazoval bank. účet, ke kterému již patří. Převod do stejného účtu je zbytečný, proto takový bank. účet byl odstraněn ze seznamu dostupných bank. účtů.
- Nebylo jasné, jak zobrazit detaily bankovního účtu. Problém byl v nesprávném názvu sloupce na prozkoumaní detailů v seznamu bank. účtů. Sloupec byl přejmenován z "Nastavení"na "Detail".
- Problém editování - testeři si nevšímali tlačítka na změny dat. Proto byla editovací tlačítka opravená na víc výrazná.
- Testeři poznamenali, že některé ikony byly umístěné neintuitivně a navíc nebylo moc jasné, jaké mají funkce. Proto byly ikony přemístěné na nové pozice a také se přidaly názvy funkcí.
- Při změně kategorie u transakce uživatel musel potvrdit akci, aby věděl jaké budou následky. Testerům se to zdálo zbytečné, proto bylo potvrzení změny kategorie odstraněno.
- Testeři poznamenali, že chybí tlačítko zpět do startovní obrazovky, což se obvykle dělá přes stisknutí na název aplikace. Proto byl přidán název aplikace a funguje jako tlačítko do startovní obrazovky.

Celkově se testerům zdála navigace v aplikaci intuitivní. Obrazovky obsahovaly stejné prvky uživatelského rozhraní na různých místech se stejnou funkčností, jako např. ikona pro přidání objektů, díky čemuž bylo používaní jednoduché.

[2] https://docs.google.com/forms/d/e/1FAIpQLSfkTLpAuDa8TJZuAHMPmCc1m842ZcG6t2_V-zl6gNeQQl9Y1A/viewform

Kapitola 6

Závěr a budoucí vylepšení

Cílem práce bylo navrhnout a implementovat webovou aplikaci pro správu osobních financí. Nejprve byla provedena analýza existujících řešení, konkrétně eÚčty, Spendee, Wallet, George. Na základě této analýzy byly určeny funkční a nefunkční požadavky. Dále byly určeny případy užití a doménový model. Po analýze následoval návrh aplikace, který popisoval klient-server architekturu projektu a také představil diagram nasazení a design aplikace. Dále byla provedená implementace serverové a klientské části. V serverové části byl použit spring boot a v klientské části Vue. S frameworkem spring boot jsem měl zkušenosti z oboru SIT, proto nebyly problémy s použitím. S frameworkem Vue to byla moje první zkušenost, je jednoduchý na používaní a má pěknou dokumentaci, což velmi pomáhalo porozumět struktuře Vue projektu. Po implementaci byla aplikace nasazená na cloud platformy, serverová část na Heroku, klientská část na Netlify. Také byl v kapitole nasazení popsán problém s cookie v aktuální verzi Google Chrome. V průběhu implementace byla aplikace testována a ve finální části vývoje bylo provedeno testování použitelnosti. Podle zpětné vazby testerů se aplikace ukázala jako dobře použitelná.

Výsledkem práce je plně funkční prototyp webové aplikace splňující definované cíle a požadavky.

6.1 Budoucí vylepšení

Přestože aplikace poskytuje všechny základní funkce pro správu osobních financí, existuje řada nápadů na její vylepšení. Tyto nápady je třeba vzít v úvahu před nasazením aplikace do produkce.

- Zlepšit rozpočty - současná verze aplikace umožňuje vytvářet rozpočty pouze na neomezenou dobu, tzn. že aplikace sleduje všechny transakce v celém období. Budoucí verze aplikace by mohly poskytovat více možností, jako jsou denní, týdenní, měsíční nebo roční rozpočty.
- Rozšířit statistiku - přidat další statistické grafy, např. pro srovnání cash flow s minulým období.
- Umožnit přihlášení přes Google nebo Facebook.
- Rozšířit na offline verzi - ukládání uživatelských dat do lokálního úložiště by snížilo využití síťových dat uživatelem a zvýšilo výkon aplikace. Také by umožnila vytvářet transakce offline.
- Umožnit připojení na online bankovnictví - pro lepší přehled by bylo možné napojit bankovní účty na online bankovnictví uživatele.
- Vytvořit mobilní verzi aplikace - současná verze je zaměřená pouze na desktopové prohlížeče. Mobilní verze by poskytla uživatelům větší volnost při používání aplikace a tím by zároveň přilákala nové uživatele.
- Přidat překlady pro další jazyky.
- Rozšířit používané měny - současná verze podporuje jen CZK a EUR.
- Rozšířit práci se sdílenými účty, třeba přidat informace o tom, kdo je tvůrcem transakce.
- Zlepšit notifikace - aplikace by mohla automaticky odesílat upozornění uživatelům na email.

Literatura

[1] ČBA: Průzkum ČBA: Finanční gramotnost Čechů 2021 [online]. [cit. 2021-12-25]. Dostupné z: https://cbaonline.cz/cesi-a-financni-gramotnost-2021

[2] Porovnej24.cz: Nejlepší finanční aplikace 2021: Appky, které šetří nebo spoří peníze! [online]. [cit. 2021-10-26]. Dostupné z: https://www.porovnej24.cz/clanky/moderni-aplikace-efektivni-sporeni-penez

[3] eÚčty.cz [online]. [cit. 2021-10-26]. Dostupné z: https://www.eucty.cz

[4] Google: GooglePlay [online]. [cit. 2021-10-26]. Dostupné z: https://play.google.com/store

[5] CLEEVIO S.R.O.: Spendee Budget & Money Tracker [software]. [cit. 2021-10-26]. Dostupné z: https://apps.apple.com/app/spendee/id635861140

[6] BudgetBakers s.r.o.: Wallet [software]. [cit. 2021-10-26]. Dostupné z: https://budgetbakers.com/

[7] Česká spořitelna, a. s.: George [software]. [cit. 2021-10-26]. Dostupné z: https://www.csas.cz/cs/internetove-bankovnictvi/george

[8] Česká spořitelna, a. s.: Oficiální web [online]. [cit. 2021-10-26]. Dostupné z: https://www.csas.cz/cs/osobni-finance

[9] Apple Inc.: AppStore - Apple [online]. [cit. 2021-10-26]. Dostupné z: https://www.apple.com/app-store/

[10] eÚčty.cz: Nápověda ke službě [online]. [cit. 2021-10-26]. Dostupné z: https://www.eucty.cz/hp-help-content.aspx

[11] Apple Inc.: Human interface guidelines [online]. [cit. 2021-10-27]. Dostupné z: https://developer.apple.com/design/human-interface-guidelines/ios/overview/themes/

[12] Česká spořitelna, a. s.: Co je virtuální platební karta? [online]. [cit. 2021-10-27]. Dostupné z: https://www.csas.cz/cs/caste-dotazy/co-je-virtualni-platebni-karta

[13] itnetwork.cz: Lekce 1 - Úvod do Spring Boot frameworku pro Javu [online]. [cit. 2021-6-5]. Dostupné z: https://www.itnetwork.cz/java/spring-boot/uvod-do-spring-boot-frameworku-pro-javu

[14] VMware, Inc.: Spring Boot [online]. [cit. 2021-6-5]. Dostupné z: https://spring.io/projects/spring-boot

[15] Evan You: Vue.js [online]. [cit. 2021-6-5]. Dostupné z: https://vuejs.org/

[16] Vuetify: Vuetify - A material Design Framework for Vue.js [online]. [cit. 2021-6-5]. Dostupné z: https://vuetifyjs.com/en/

[17] Google: Google Chrome [software]. [cit. 2021-6-5]. Dostupné z: https://www.google.com/intl/en/chrome/

[18] mozilla.org: Firefox Browser [software]. [cit. 2021-6-5]. Dostupné z: https://www.mozilla.org/en-US/firefox/new/

[19] Baeldung: JPA/Hibernate Persistence Context [online]. [cit. 2021-10-14]. Dostupné z: https://www.baeldung.com/jpa-hibernate-persistence-context

[20] Baeldung: BALASUBRAMANIAM, Vivek. Defining JPA Entities [online]. [cit. 2021-10-17]. Dostupné z: https://www.baeldung.com/jpa-entities

[21] VMware, Inc.: Object Relational Mapping (ORM) Data Access [online]. [cit. 2021-6-5]. Dostupné z: https://docs.spring.io/spring-framework/docs/3.0.0.RELEASE/reference/html/orm.html

[22] Baeldung: Hibernate Inheritance Mapping [online]. [cit. 2021-10-17]. Dostupné z: https://www.baeldung.com/hibernate-inheritance

[23] ObjectDB Software: JPA Named Queries [online]. [cit. 2021-10-17]. Dostupné z: https://www.objectdb.com/java/jpa/query/named

[24] HowToDoInJava: @Repository annotation in Spring Boot [online]. [cit. 2021-10-17]. Dostupné z: https://howtodoinjava.com/spring-boot/repository-annotation/

[25] VMware, Inc.: Securing a Web Application [online]. [cit. 2021-10-20]. Dostupné z: https://spring.io/guides/gs/securing-web/

[26] JournalDev: Spring @Service Annotation [online]. [cit. 2021-10-19]. Dostupné z: https://www.journaldev.com/21435/spring-service-annotation

[27] Baeldung: PARASCHIV, Eugen. Transactions with Spring and JPA [online]. [cit. 2021-10-19]. Dostupné z: https://www.baeldung.com/transaction-configuration-with-jpa-and-spring

[28] Sběr a modelování požadavků B6B36SMP: Přednáška 4 a 5 - Případy užití [online]. [cit. 2021-02-09]. Dostupné z: https://moodle.fel.cvut.cz/course/view.php?id=3878 ČVUT FEL SIT.

[29] Sběr a modelování požadavků B6B36SMP: Přednáška 2 - Modelování business procesů a entit pomocí UML - předmět B6B36SMP [online]. [cit. 2021-02-18]. Dostupné z: https://moodle.fel.cvut.cz/course/view.php?id=3878 ČVUT FEL SIT

[30] Sběr a modelování požadavků B6B36SMP: Přednáška 9 - UML diagramy sekvencí a komponent [online]. [cit. 2021-02-28]. Dostupné z: https://moodle.fel.cvut.cz/course/view.php?id=3878 ČVUT FEL SIT

[31] Evan You: Introduction - Vue.js [online]. [cit. 2021-10-23]. Dostupné z: https://vuejs.org/v2/guide/index.html

[32] Vuetify: Why you should be using Vuetify [online]. [cit. 2021-10-23]. Dostupné z: https://vuetifyjs.com/en/introduction/why-vuetify/

[33] Google: Material design [online]. [cit. 2021-10-23]. Dostupné z: https://material.io/

[34] S. PRESSMAN, Roger a Bruce MAXIM. Software Engineering. ISBN 978-0078022128.

[35] CNBC LLC.: FRIED, Carla. Personal Finance 101: The complete guide to managing your money [online]. [cit. 2021-11-03]. Dostupné z: https://www.cnbc.com/guide/personal-finance-101-the-complete-guide-to-managing-your-money/

[36] the balance: IRBY, LaToya. Best Personal Finance Software Options [online]. [cit. 2021-11-03]. Dostupné z: https://www.thebalance.com/best-personal-finance-software-4171938

[37] JetBrains s.r.o.: Intelij IDEA [software]. [cit. 2021-11-03]. Dostupné z: https://www.jetbrains.com/idea/

[38] Git [online]. [cit. 2021-11-03]. Dostupné z: https://git-scm.com/

[39] Postman, Inc.: Postman API platform [online]. [cit. 2021-11-03]. Dostupné z: https://www.postman.com/

[40] MDN Web Docs: Cross-Origin Resource Sharing (CORS) [online]. [cit. 2021-11-06]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS

[41] Micro Focus: Understanding Cron Syntax in the Job Scheduler [online]. [cit. 2021-11-06]. Dostupné z: https://www.netiq.com/documentation/cloud-manager-2-5/ncm-reference/data/bexyssf.html

[42] Adobe: BABICH, Nick. Prototyping 101: The Difference between Low-Fidelity and High-Fidelity Prototypes and When to Use Each [online]. [cit. 2021-11-07]. Dostupné z: https://blog.adobe.com/en/publish/2017/11/29/prototyping-difference-low-fidelity-high-fidelity-prototypes-use.html#gs.f5bcmb

[43] Figma [online]. [cit. 2021-11-07]. Dostupné z: https://www.figma.com

[44] Evan You: Single File Components [online]. [cit. 2021-12-01]. Dostupné z: https://v3.vuejs.org/guide/single-file-component.html#introduction

[45] Evan You, Eduardo San Martin Morote: Vue router [online]. [cit. 2021-12-02]. Dostupné z: https://router.vuejs.org/guide/#html

[46] John Jakob "Jake"Sarjeant.: Axios Docs [online]. [cit. 2021-12-02]. Dostupné z: https://axios-http.com/docs/intro

[47] What is Vuex? [online]. [cit. 2021-12-03]. Dostupné z: https://vuex.vuejs.org/#what-is-a-state-management-pattern

[48] Vuex - The Simplest Store [online]. [cit. 2021-12-03]. Dostupné z: https://vuex.vuejs.org/guide/#the-simplest-store

[49] PortSwigger Ltd.: Cross-site scripting [online]. [cit. 2021-12-03]. Dostupné z: https://portswigger.net/web-security/cross-site-scripting

[50] Vue - Security [online]. [cit. 2021-12-03]. Dostupné z: https://vuejs.org/v2/guide/security.html

[51] Salesforce.com: About Heroku [online]. [cit. 2021-12-04]. Dostupné z: https://www.heroku.com/about

[52] Salesforce.com: Heroku Postgres [online]. [cit. 2021-12-04]. Dostupné z: https://devcenter.heroku.com/articles/heroku-postgresql#understanding-heroku-postgres-plans

[53] Salesforce.com: GitHub Integration (Heroku GitHub Deploys) [online]. [cit. 2021-12-04]. Dostupné z: https://devcenter.heroku.com/articles/github-integration

[54] Agility CMS 2021: VARTY, Joel. What is Netlify and What are its Benefits? [online]. [cit. 2021-12-05]. Dostupné z: https://agilitycms.com/resources/posts/what-is-netlify-and-why-should-you-care-as-an-editor

[55] Netlify, Inc.: Netlify docs [online]. [cit. 2021-12-05]. Dostupné z: https://docs.netlify.com/

[56] Salesforce.com: Heroku Dynos [online]. [cit. 2021-12-05]. Dostupné z: https://www.heroku.com/dynos

[57] ApexCharts.: Vue - ApexChart [online]. [cit. 2021-12-06]. Dostupné z: https://apexcharts.com/docs/vue-charts/

[58] Evan You: The Vue Instance Lifecycle Hooks [online]. [cit. 2021-12-06]. Dostupné z: https://vuejs.org/v2/guide/instance.html#Instance-Lifecycle-Hooks

[59] Testování softwaru B6B36TS1: Přednášky 6 - 8 - Jednotkové testování [online]. [cit. 2021-12-07]. Dostupné z: https://moodle.fel.cvut.cz/course/view.php?id=5974 ČVUT FEL SIT

[60] 2015 Evan You: Vue.js version 0.12 [online]. [cit. 2021-12-08]. Dostupné z: https://012.vuejs.org/guide/

[61] Evan You: Vue.js Directives [online]. [cit. 2021-12-08]. Dostupné z: https://vuejs.org/v2/guide/syntax.html#Directives

[62] Microsoft 2021: MICROSOFT 365 TEAM. Jednoduchý návod k UML diagramům a modelování databází [online]. [cit. 2021-12-08]. Dostupné z: https://www.microsoft.com/cs-cz/microsoft-365/business-insights-ideas/resources/guide-to-uml-diagramming-and-database-modeling

[63] Vuetify Date tables [online]. [cit. 2021-12-08]. Dostupné z: https://vuetifyjs.com/en/components/data-tables/

[64] Vuetify Date pickers [online]. [cit. 2021-12-08]. Dostupné z: https://vuetifyjs.com/en/components/date-pickers/

[65] Kevel: SHUPTRINE, Chris. SameSite Cookie Attribute: What It Is And Why It Matters [online]. [cit. 2021-12-11]. Dostupné z: https://www.kevel.co/blog/chrome-samesite/

[66] Clearcode Services S.A.: WLOSIK, Michal a Michael SWEENEY. What's the Difference Between First-Party and Third-Party Cookies? [online]. [cit. 2021-12-11]. Dostupné z: https://clearcode.cc/blog/difference-between-first-party-third-party-cookies/#first-vs-third

[67] MDN Web Docs: Using HTTP cookies [online]. [cit. 2021-12-11]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies

[68] baeldung: Testing in Spring Boot [online]. [cit. 2021-12-11]. Dostupné z: https://www.baeldung.com/spring-boot-testing

[69] Spring Framework: Class SpringExtension [online]. [cit. 2021-12-11]. Dostupné z: https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/test/context/junit/jupiter/SpringExtension.html#SpringExtension--

[70] HowToDoInJava: Spring Boot – @DataJpaTest [online]. [cit. 2021-12-12]. Dostupné z: https://howtodoinjava.com/spring-boot2/testing/datajpatest-annotation/

[71] Annotation Type AutoConfigureTestDatabase [online]. [cit. 2021-12-12]. Dostupné z: https://docs.spring.io/spring-boot/docs/current/api/org/springframework/boot/test/autoconfigure/jdbc/AutoConfigureTestDatabase.html

[72] Jan Bodnar: Spring @ComponentScan tutorial [online]. [cit. 2021-12-12]. Dostupné z: https://zetcode.com/spring/componentscan/

[73] baeldung: Quick Guide on Loading Initial Data with Spring Boot [online]. [cit. 2021-12-12]. Dostupné z: https://www.baeldung.com/spring-boot-data-sql-and-schema-sql

[74] baeldung: Wiring in Spring: @Autowired, @Resource and @Inject [online]. [cit. 2021-12-12]. Dostupné z: https://www.baeldung.com/spring-annotations-resource-inject-autowire

[75] Testování softwaru: HLAVA, Tomáš. Fáze a úrovně provádění testů [online]. [cit. 2021-12-12]. Dostupné z: http://testovanisoftwaru.cz/tag/systemove-testovani/

[76] Experience UX: What is usability testing? [online]. [cit. 2021-12-19]. Dostupné z: https://www.experienceux.co.uk/faqs/what-is-usability-testing/

[77] GeeksforGeeks: SYEDMODASSIRALI. Client-Server Model [online]. [cit. 2021-12-13]. Dostupné z: https://www.geeksforgeeks.org/client-server-model/

[78] TechTarget: GILLIS, Alexandr S. REST API (RESTful API) [online]. [cit. 2021-12-13]. Dostupné z: https://searchapparchitecture.techtarget.com/definition/RESTful-API

[79] Usability.gov: System Usability Scale (SUS) [online]. [cit. 2021-12-16]. Dostupné z: https://www.usability.gov/how-to-and-tools/methods/system-usability-scale.html

[80] Principy tvorby mobilních aplikací B6B39PDA: Usability testing in mobile environment - System Usability Scale [online]. [cit. 2021-12-16]. Dostupné z: https://moodle.fel.cvut.cz/course/view.php?id=5827 ČVUT FEL SIT

[81] Synopsys, Inc.: Cross-Site Request Forgery [online]. [cit. 2021-12-22]. Dostupné z: https://www.synopsys.com/glossary/what-is-csrf.html

Příloha A

Seznam použitých zkratek

ČBA Česká bankovní asociace

CSV Comma-separated values

REST Representational state transfer

JSON JavaScript Object Notation

HTML HyperText Markup Language

CSS Cascading Style Sheets

DAO Data Access Object

MVC Model–view–controller

XML Extensible Markup Language

ORM Object-Relational Mapping

MVVM Model-View-ViewModel

SPA Single-page application

UML Unified Modeling Language

DOS Denial of Service

MITM Man in the Middle

HTTP HyperText Transfer Protocol

API Application programming interface

HTTPS HyperText Transfer Protocol Secure

JPA Java Persistence API

UI User Interface

ES5 ECMAScript

DOM Document Object Model

SIT Softwarové inženýrství a technologie - program FEL ČVUT

JPQL Java Persistence Query Language

CRUD Create, read, update and delete

SQL Structured Query Language

CORS Cross-origin resource sharing

URL Uniform Resource Locator

CSRF Cross-site request forgery

XSS Cross-Site Scripting

SFC Single File Components

UC Use Case

PaaS Platform as a Service

SUS System Usability Scale