FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Aplikace pro správu osobních financí
Student:	Vojtěch Stuchlík
Vedoucí:	Ing. Miroslav Balík, Ph.D.
Studijní program:	Informatika
Studijní obor:	Webové a softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce letního semestru 2018/19

Pokyny pro vypracování

Cílem práce je implementovat server a webového klienta, díky nimž budou uživatelé moci spravovat své příjmy a výdaje.
Systém bude uživateli umožňovat:

1) tvorbu vlastních sekcí útraty (tj. oblast, ve které uživatel utrácel jako např. jídlo, kosmetika, doprava, ap.);
2) měsíční přehled;
3) filtrace položek dle kritérií;
4) podporu jazykových mutací;
5) více peněženek;
6) položky v jiné měně (v případě, že platím v zahraničí kartou);
7) změna položky (update), či přesunutí do jiné peněženky;
8) import/export dat v daném formátu;
9) registrace/přihlášení přes Facebook API;
10) zobrazení zůstatku.

Analyzujte obdobné aplikace, vyberte vhodné technologie, aplikaci vytvořte a otestujte.

Seznam odborné literatury

Dodá vedoucí práce.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

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

V Praze dne 30. prosince 2017

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Bakalářská práce

Aplikace pro správu osobních financí

Vojtěch Stuchlík

Katedra softwarového inženýrství
Vedoucí práce: Ing. Miroslav Balík Ph.D.

3. května 2018

Poděkování

Poděkovat bych chtěl svému vedoucímu bakalářské práce Miroslavu Balíkovi za cenné rady a vedení při práci, své rodině za podporu během studia, ale také všem zúčastněným testerům, díky nimž jsem mohl odhalit slabé místa práce a opravit je.

Prohlášení

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

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. Dále prohlašuji, že jsem s Českým vysokým učením technickým v Praze uzavřel dohodu, na základě níž se ČVUT vzdalo práva na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona. Tato skutečnost nemá vliv na ust. § 47b zákona č. 111/1998 Sb., o vysokých školách, ve znění pozdějších předpisů.

V Praze dne 3. května 2018 .....

Odkaz na tuto práci

Stuchlík, Vojtěch. *Aplikace pro správu osobních financí.* Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2018. Dostupný také z WWW: ⟨`http://utrata.cu.ma`⟩.

Abstrakt

Práce se zaměřuje na návrh a implementaci webové aplikace, která uživatelům umožňuje spravovat své příjmy a výdaje. Práce ukazuje srovnání s obdobnými aplikacemi a dále je popisovaná aplikace rozebrána analyticky, návrhově a v neposlední řadě popisuje implementační proces.

Součástí práce je i testování. A to jak black box testy, tak white box testy.

Klíčová slova webová aplikace, útrata, příjmy, výdaje, zůstatek hotovosti

Abstract

The work focuces on the design and implementation of the web application that allows users to manage their incomes and expenses. The work shows comparison with similar applications and the application is described analytically, design and there is description of the implementation process.

Part of the work is also testing. And it contains both black box testing and white box testing.

Keywords web application, spending, incomes, expense, cash balance

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Mnoha lidem není lhostejné, kolik peněz utratí, proto si vedou své vlastní či rodinné účetní záznamy – především v papírové podobě, v lepším případě v nějakém tabulkovém procesoru jako například Microsoft Excel[1] či LibreOffice[2]. Nevýhoda těchto řešení je především v tom, že jsou přístupné jen na jednom místě a v případě rodinného účetnictví s nimi může v daném okamžiku pracovat pouze jeden člen rodiny. Tento problém některé domácnosti řeší systémem Google sheet[3], což je tabulkový procesor, který je v daném okamžiku přístupný více uživatelům najednou z více míst. Zůstává ale nevýhoda tabulkového procesoru, tj. všechny součty a jiné vzorce si uživatelé musejí sami nastavit.

Elektronických aplikací šitých na míru této problematice se využívá jen výjimečně, protože bývají často placené a zbytečně komplexní s mnoha funkcemi, které odrazují od naučení se dané aplikace. Není tedy překvapivé, že tyto aplikace využívají především mladší uživatelé.

Na základě těchto úsudků vznikla myšlenka vytvoření webové aplikace pro správu osobních financí (dále jen Útrata), která je zde popsána. Uživatelé si budou moci zakládat několik rozpočtových peněženek, v nichž si mohou vytvářet jednotlivé položky příjmů a výdajů. Tyto pak mohou zpětně upravovat či mazat do doby, než si je nenávratně přesunou do archivní sekce.

První část této práce se zaměřuje na srovnání s obdobnými aplikacemi na správu osobních financí a popisuje, v čem tkví výhody této práce. Také jsou analyzovány možnosti nasazení na hostingový server dle požadavků na aplikaci a výběr vhodné technologie.

Další části se věnují aplikaci samotné, a to od návrhu, popisu implementace a nasazení až k testování. Na závěr je uvedeno pár slov o dosažených výsledcích a o možnostech další rozšiřitelnosti.

[1] https://microsoft-excel-2016.en.softonic.com/
[2] https://www.libreoffice.org/download/download/
[3] https://www.google.com/sheets/about/

Kapitola 1

Cíl práce

Cílem práce je nejdříve zanalyzovat současné podobné aplikace řešící stejný problém, podívat se na danou sféru, ve které se pohybujeme, zdokumentovat požadavky. Dále je pak nutné vybrat vhodnou technologii a navrhnout architekturu.

Následně dle požadavků navrhnout, implementovat aplikaci a otestovat ji. Testování bude provedeno jednak automatizovaně, pomocí jednotkových a systémových testů, a jednak testery vybranými z různých věkových skupin. Testeři budou postupovat podle testovacích scénářů, jejichž obsah bude pokrývat funkce aplikace, které jsou nadefinovány v seznamu funkčních požadavků.

Kapitola 2

Analýza

V této kapitole se podíváme na již existující obdobné aplikace a srovnání s touto prací, analyzujeme předpoklady a požadavky. Analýza se zaměřuje na zdokumentování hlavních funkčností aplikace, ale nezatěžuje se detaily, jak budou ony funkce definovány.

Také je potřeba zvolit si technologie, které k tvorbě použijeme a podívat se, kam bude vhodné aplikaci nasadit.

2.1 Obdobné aplikace

Nejdříve proběhlo prozkoumání již fungujících podobných aplikací. Mezi zkoumané aplikace patří eÚčty.cz[4] (webové řešení), což je v Česku nejpoužívanější webové řešení[1], Spendee[5] (řešení pro mobilní telefony) a běžný mobilní klient internetového bankovnictví. Bylo vybráno několik zajímavých funkcionalit, které se staly kritérii pro porovnání.

Existují placené verze Spendee i eÚčty.cz, ale my jsme se zaměřili na bezplatné verze kvůli srovnatelnosti, jelikož aplikace Útrata bude nabízena také kompletně zdarma.

Bylo zjištěno, že několik funkcionalit podporují všechny aplikace. Hlavní výhodou Útraty je však přihlášení přes Facebook[6], což odstiňuje uživatele od manuální registrace, tedy i od potřeby pamatovat si další přihlašovací údaje. Dále potom aplikace disponuje možností překladů, tedy možnost mít aplikaci v několika jazycích.

2.1.1 eÚčty.cz

První zkoumanou aplikací je webová aplikace eÚčty.cz. Je to aplikace určená hlavně pro správu rodinných financí, což znamená, že jeden rozpočet může

[4] https://eucty.cz
[5] https://play.google.com/store/apps/details?id=com.cleevio.spendee&hl=en
[6] https://www.facebook.com

spravovat více uživatelů, kteří jsou přizváni vlastníkem rozpočtu. Tento vlastník dále může všem členům nastavit práva – tedy co vše mohou se sdílenými daty dělat.

Tato aplikace mezi své přednosti dále uvádí statistiky spolu s grafovým podkladem, možnost importu dat z měsíčního výpisu od několika podporovaných bank, možnost několika účtů v různých měnách, vytváření vlastních kategorií, responzivitu pro mobilní zařízení.

To všechno je pro uživatele přístupné zdarma, nicméně aplikace je zbytečně složitá, bylo těžké se v ní vyznat a najít konkrétní funkci.

Velké plus je demo verze – tedy vyzkoušení si základních funkcionalit této aplikace bez potřeby registrování se.

2.1.2 Spendee

Definice 1. *Peněženka je množina položek (příjmů či výdajů), která tyto položky logicky odděluje od ostatních položek. Peněženku lze pojmenovat pro zdůraznění logického oddělení.*

Další je mobilní aplikace Spendee, která slouží pro správu hlavně osobního finančnictví. Bezplatná verze zvládá vše, co uživatel běžně potřebuje, tedy vkládání jednotlivých příjmů a výdajů, nastavování si měsíčního rozpočtu, vytváření si vlastních kategorií, měsíční přehledy. Pokud však chce uživatel více peněženek nebo peněženku s někým sdílet, musí si přikoupit Pro verzi.

Spendee sice nabízí řešení jak pro Android a iOS, tak i webovou beta verzi, nicméně mobilní aplikace ukládají data na lokální úložiště na chytrém zařízení. To znamená, že jediným momentálním způsobem, jak synchronizovat data s webovou aplikaci, je importovat exportovaná data z mobilní aplikace.

Výhoda této aplikace je nedávné propojení s bankou a tedy automatickým importem dat, které musel u eÚčty.cz uživatel udělat ručně.

2.1.3 Mobilní bankovnictví

Jako poslední zkoumanou aplikací je mobilní klient internetového bankovnictví pro mBank[7]. Ten, asi jako každý klient internetového bankovnictví, umožňuje pouze zobrazování položek, kterým uživatel může pouze změnit kategorii. Tedy aplikace také jen pro jednoho uživatele.

Výhoda tohoto řešení je, že uživatel nemusí nic ručně zadávat a data se tvoří jako odraz pohybu financí na uživatelově účtě.

Hlavními nevýhodami tohoto řešení jsou nemožnost v položkách filtrovat dle pro uživatele zajímavých kritérií a uživatel nemá přehled o hotovosti.

[7] https://play.google.com/store/apps/details?id=cz.mbank&hl=en

Tabulka 2.1: Srovnání funkcionalit s konkurenčními aplikacemi

Funkcionalita	Útrata	eÚčty.cz	Spendee	m. bankovnictví
Přihlášení nebo registrace přes Facebook	ANO	NE	NE	NE
Překlady	ANO	NE	NE	NE
Více peněženek	ANO	NE	NE	NE
Tvorba vlastních sekcí	ANO	ANO	ANO	NE
Filtrace dle kritérií	ANO	ANO	NE	NE
Měsíční přehled	ANO	ANO	ANO	ANO
Grafy s přehledem	NE	ANO	ANO	většinou ANO
Propojení s bankou	NE	jen import	NE	ANO
Automatizované opakování výdaje	NE	ANO	ANO	ANO
Sdílení na sociální sítě	NE	NE	ANO	NE
Položka v jiné měně	ANO	NE	ANO	ANO
Změna položky (update)	ANO	ANO	ANO	ANO
Import/export dat	ANO	ANO	jen export	NE

2.1.4 Srovnání funkcionalit

V tabulce 2.1 si srovnáme vybrané zajímavé funkcionality Útraty s výše uvedenými aplikacemi. Z ní je vidět, že Útrata je unikátní propojením s facebookovým účtem.

2.2 Určení požadavků

Abychom byli schopni vytvořit nějaký software, musíme znát požadavky na tento software kladené. Tyto požadavky jsou nám obvykle sděleny zadavatelem přímo, nebo, v případě, že si zadavatel není zcela jist a nemá jasnou představu, se na ně musíme doptat vhodnými otázkami. V našem případě nám byly požadavky zadavatelem sděleny.

Budeme se zabývat funkčními a nefunkčními požadavky. Tyto dvě skupiny se liší hlavně v tom, že požadavky funkční říkají, jaké funkcionality má daný software mít a jaké ne, kdežto nefunkční požadavky kladou na software omezení, která mají dopad na volbu technologií a architekturu.

2.2.1 Funkční požadavky

F1 Přihlášení bude umožněno jak pomocí účtu aplikace, tak pomocí Facebook účtu.

F2 Aplikace bude umožňovat jazykové mutace.

F3 Uživatel si může vytvořit více peněženek.

F4 Uživatel si bude moci vytvořit, jaké sekce útraty chce, a z nich si vybrat jen ty, které chce používat.

F5 Bude umožněna filtrace a řazení položek příjmů i výdajů. Řazení bude umožněno podle názvu, data nebo ceny položky. Filtrovat se pak bude moci skrze rok, kalendářní měsíc, sekci útraty nebo zadané slovo vyskytující se v názvu či popisu položky. Seznam položek se bude aktualizovat automaticky, bez potřeby potvrzovacího tlačítka.

F6 Zobrazení měsíčního přehledu útraty v porovnání s ostatními měsíci. Zahrnut bude i filtr, pro jakou sekci útraty si bude uživatel chtít přehled zobrazit. Porovnání bude ve dvou částech. V první části bude porovnání s měsíci v odpovídajícím období, tj. do data aktuálního dne (když bude například 13. září, tak se bude zobrazovat porovnání útraty položek zaznamenaných v daných měsících pouze po 13. den), v druhé části pak bude porovnání s celými měsíci. V každé z těchto sekcí bude navíc tabulka porovnání aktuálního měsíce s minimem, maximem a průměrem měsíců předešlých.

F7 Podpora pro položky ve více měnách v rámci stejné peněženky.

F8 Změna hodnot položky a zároveň možnost přesunutí do jiné peněženky.

F9 Aplikace bude umožňovat export uživatelských dat a zároveň i jejich import. Import bude sloužit jak pro obnovu dat, tak pro hromadnou tvorbu nových dat.

2.2.2 Nefunkční požadavky

N1 Server aplikace bude kompatibilní s PHP 5.6 a novější. PHP 5.6 je minimální povolenou verzí z toho důvodu, že je to minimální verze, kde neskončila podpora pro opravu chyb[8].

N2 Klient bude pracovat správně s minimálními verzemi následujících prohlížečů: Google Chrome[9] 64.0.3282.140, Mozilla Firefox[10] 58.0.2

[8] http://php.net/supported-versions.php
[9] https://www.google.com/chrome
[10] https://www.mozilla.cz/stahnout/firefox

N3 Perzistentní úložiště bude databáze MySQL[11]

2.3 Uživatelské účty

V rámci této práce, budeme-li mluvit o uživateli, myslíme tím přihlášeného uživatele bez jakýchkoliv speciálních práv. Toto zjednodušení si můžeme dovolit díky faktu, že aplikace nepodporuje uživatele v různých rolích. Zároveň aplikace neumožňuje nepřihlášenému uživateli cokoliv dělat.

2.4 Případy užití

Modelování případů užití znamená jednak zdokumentovat interakci mezi uživateli a systémem a jednak pokrýt funkční požadavky. Toto se může provést buď textově nebo graficky pomocí jazyka UML (Unified Modeling Language – univerzální grafický jazyk pro modelování systémů). Tyto metody jsou si obsahově ekvivalentní, leč grafická cesta bývá čtenářem rychleji pochopena. Vždy se ale popisuje interakce mezi dvěma stranami: aktérem a případem užití. Aktér může být uživatel, či systém. Případ užití pak popisuje chování systému pro daný požadavek aktéra. V našem případě jsou případy užití popsány textově:

UC01 - Registrovat/přihlásit přes Facebook

Na přihlašovací obrazovce klikne nepřihlášený uživatel na ikonu Facebooku (modré tlačítko s nápisem „facebook“). Systém zkontroluje, zdali není daný uživatel již dříve registrován. V takovém případě uživatele přesměruje na jeho domovskou stránku, tj. přehled peněženek.

V případě, že ještě registrován není, vytvoří mu systém účet a přesměruje jej na jeho domovskou stránku, která bude prázdná. Zobrazovat se bude jen menu a tlačítko na vytvoření nové peněženky.

Jestliže systém z Facebooku nezíská potřebné údaje (jméno, příjmení, jazyk a facebookové identifikační číslo), nikam nepřesměruje, naopak o této skutečnosti nepřihlášeného uživatele informuje a oznámí mu, jaké údaje si bude muset v nastavení Facebooku zpřístupnit.

UC02 - Změnit jazyk

Uživatel se přepne do nastavení účtu a z výběru jazyků si jeden vybere. Po uložení změněných informací systém zkontroluje, je-li jazyk změněn. Pakliže ano, načte aktuální stránku znova již s překlady v novém jazyce.

UC03 - Vytvořit peněženku

Uživatel na své domovské stránce s přehledem peněženek klikne na tlačítko se znakem plus. Systém zareaguje zobrazením vyskakovacího

[11] https://www.mysql.com/

okna (dále jen popup). V popupu vyplní uživatel název nové peněženky a potvrdí tlačítkem.

Jestliže bude jméno prázdné, systém tuto skutečnost detekuje, oznámí to uživateli a čeká na nápravu. V opačném případě se vytvoří nová peněženka a uživatel je přesměrován na její detail. Systém nekontroluje duplicitu jmen peněženek v rámci stejného uživatele, může tedy pro daného uživatele existovat více peněženek stejného jména.

UC04 - Vybrat sekci útraty

Pokud uživateli nedostačují jeho dosavadní vybrané sekce útraty, otevře si osobní nastavení a na příslušném řádku si ze selectboxu vybere sekce, které chce. Poté svůj výběr potvrdí tlačítkem pro uložení osobního nastavení.

UC05 - Vytvořit vlastní sekci útraty

Uživateli v osobním nastavení nějaká sekce ve výběru chybí, proto vyplní její název vedle selectboxu se sekcemi útraty a klikne na tlačítko s ikonou plus. Nová sekce se zobrazí v seznamu a uživatel ji může používat.

Pokud systém zjistí duplicitu v názvu sekce s jinou sekcí útraty, které je uživateli dostupná, je tato skutečnost uživateli oznámena, a uživatel bude muset změnit název sekce, nebo si vybrat již existující.

UC06 - Filtrovat a řadit položky

Uživatel na detailu peněženky, v archivu či v příjmech změní v záhlaví seznamu položek vybraný styl řazení, měsíc, sekci, rok nebo vyplní vyhledávací frázi, systém tuto změnu zaregistruje a zobrazí jen takové položky, které odpovídají danému filtru, a v pořadí, jaké je nastaveno v záhlaví seznamu položek.

V případě, že danému filtru neodpovídá žádná položka, zobrazí systém místo položek větu, že takovému výběru neodpovídá žádná položka. Tato věta bude samozřejmě v jazyce uživatele.

UC07 - Zobrazit měsíční přehled

Uživatel se bude chtít podívat, jak na tom tento měsíc je, proto v menu klikne na ikonku s písmenem „M“. Systém jej přesměruje na stránku s přehledovými tabulkami.

UC08 - Vytvořit položku v jiné měně

Při tvorbě položky klikne uživatel v části formuláře s cenou na text „jiná měna“ a systém zobrazí ve formuláři pod políčkem s cenou další řádek obsahující selectbox s měnami podporovanými systémem a políčko pro zadání kurzu k výchozí měně uživatele. Přednastavená měna je měna uživatelova a kurz je nastaven na 1. Uživatel změní měnu, systém tuto změnu zaznamená, zjistí hodnotu kurzu mezi uživatelovou měnou

a měnou právě vybranou a změní políčko s kurzem na tuto hodnotu. Tento kurz si může uživatel ještě pozměnit a vytvoří položku. Systém zakomponuje položku do seznamu položek s již přepočítanou cenou do uživatelovy měny (tedy $zadanáCena * zadanýKurz$).

UC09 - Změna hodnot položky

Uživatel si chce změnit nějaký údaj položky, která není ještě v archivu, proto u ní klikne na tlačítko s ikonou tužky a systém zobrazí popup s detailem položky. Uživatel pozmění, co potřebuje a klikne na tlačítko „Upravit“. Systém změní položce údaje a znovu uspořádá položky dle aktuálně nastaveného filtru a stylu řazení.

V případě, že jsou změněné hodnoty nepovolené, je o této skutečnosti uživatel obeznámen a systém čeká, až uživatel tuto chybu napraví a znovu se pokusí položku změnit nebo opustí tento formulář.

UC10 - Přesunout položku do jiné peněženky

Uživatel si při tvorbě položky spletl peněženku, ještě ji nearchivoval a nyní ji chce přesunout v rámci peněženek, proto u ní klikne na tlačítko s ikonou tužky, systém zobrazí popup s detailem položky. Uživatel změní v poslední políčku formuláře peněženku a klikne na „Upravit“. Systém přesune položku do nové peněženky a uživateli zobrazí seznam položek již bez této položky.

UC11 - Exportovat uživatelská data

Uživatel si chce stáhnout zálohu svých dat, proto v menu klikne na tlačítko s ikonou šipky směrem dolů, systém vytvoří CSV (Comma-separated values) soubor a vyvolá v prohlížeči metodu stažení tohoto souboru, takže se dle nastavení prohlížeče uživateli nabídne, jestli chce soubor otevřít/stáhnout, nebo jej přímo stáhne.

UC12 - Importovat uživatelská data

Uživatel v menu klikne na tlačítko s ikonou šipky směrem nahoru a systém otevře popup s formulářem. Uživatel klikne na „Vybrat soubor“, systém zavolá metodu v prohlížeči na vybrání souboru z počítače. Uživatel vyhledá a vybere příslušný soubor v podobě, jak jej dříve stáhl, či obohacený o nějaká data, a klikne na tlačítko „Nahrát“. Systém zkontroluje správnost struktury souboru, uloží chybějící data ze souboru, oznámí uživateli úspěch a zavře popup.

Jestliže jsou data v souboru ve špatném tvaru, systém uživatele upozorní, na jakém řádku se chyba vyskytla a čeká, jestli uživatel nahraje nový, opravený soubor, nebo akci zruší.

2.4.1 Pokrytí požadavků případy užití

Následující tabulka 2.2 nastiňuje, jak jednotlivé případy užití pokrývají zadané požadavky. Díky této tabulce můžeme odhalit, jestli jsme nezapomněli na nějaký požadavek, případně jestli jsme nějaký požadavek zbytečně nepokryli více případy užití.

Tabulka 2.2: Pokrytí požadavků případy užití

	F1	F2	F3	F4	F5	F6	F7	F8	F9
UC01	+								
UC02		+							
UC03			+						
UC04				+					
UC05				+					
UC06					+				
UC07						+			
UC08							+		
UC09								+	
UC10								+	
UC11									+
UC12									+

Vidíme, že všechny požadavky jsou pokryty alespoň jedním případem užití. Skutečnost, že jej někdy pokrývá více případů užití nám nevadí, jelikož každý požadavek se typicky rozkládá na více případů užití. Jen je třeba zkontrolovat, zdali se tyto případy užití nepřekrývají – neřeší tutéž část požadavku. Vidíme tedy 3 takovéto překryvy, ale ani v jednom nenastává problém řešení stejné části požadavku vícero případy užití.

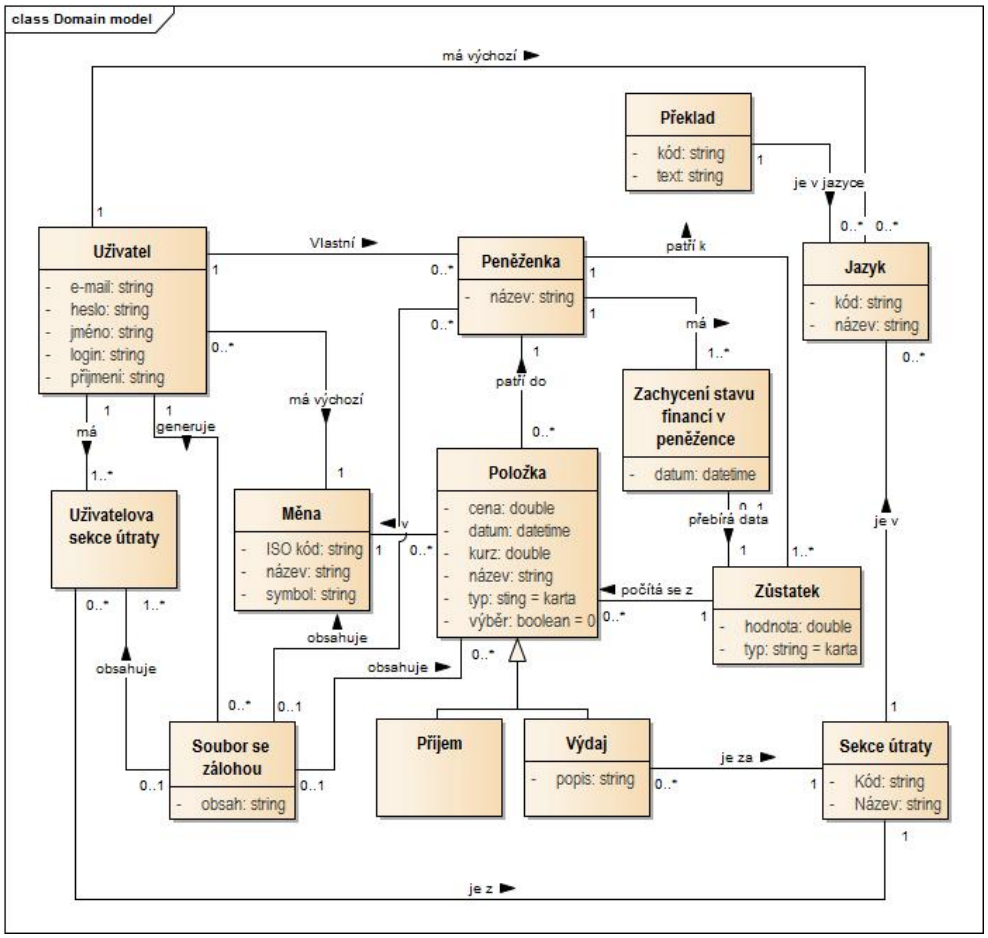
2.5 Doménový model

Zásadním krokem analýzy je proniknutí do problematiky domény. K tomu nám pomáhá doménový model, k jehož vyjádření se v jazyce UML často používá model tříd, kde poté využíváme, že jednotlivé entity mohou mít jak své atributy, tak operace, které je daná entita schopna provádět. Samotné entity pak zpravidla mezi sebou mají relace.

Doménový model, který najdeme na obrázku 2.1, slouží jako podklad pro tvorbu databázového návrhu. Následuje popis entit doménového modelu.

Uživatel
: Přihlášený uživatel používající Útratu.

Peněženka
: Viz. definice 1.



Obrázek 2.1: Doménový model

Jazyk

Tato entita zde je pro zaznamenání, v jakém jazyce chce uživatel zobrazovat překlady.

Měna

Ve funkčních požadavcích je uvedeno, že položky mohou být v různých měnách. K jejich zachycení slouží entita Měna.

Položka

K zachycení položky z reálného světa slouží atributy `název`, `cena`, `kurz`, `datum`, `typ` a `výběr`. `Typ` může nabývat hodnoty *karta* nebo *hotovost* a `výběr` vypovídá o tom, šlo-li o pouhý výběr z bankomatu. V takovém případě nejde o obyčejný výdaj, nýbrž o přesun peněz z karty na hotovost.

Příjem
Specializace položky značící příjem pro konkrétní peněženku.

Výdaj
Specializace položky značící výdaj pro danou peněženku. U výdaje se uvádí sekce útraty a navíc si u ní uživatel může vyplnit popis.

Sekce útraty
Doménová entita, díky níž lze výdaje logicky seskupovat do skupin podle toho, za co uživatel utrácel.

Uživatelova sekce útraty
Taková sekce útraty, kterou chce uživatel používat.

Soubor se zálohou
Textový soubor obsahující veškerá uživatelova data. Soubor slouží k zálohování dat a jejich následnému obnovení.

Zachycení stavu v peněžence
Tato entita se hodí v případě, kdy si uživatel chce poznamenat, kolik má v danou chvíli v peněžence financí. Uchovává se proto aktuální časová značka a `hodnota`, tj. aktuální finanční obnos v dané peněžence, a navíc se upřesňuje, jestli se zachycuje stav na kartě, nebo v hotovosti.

Zůstatek
Aktuální stav financí v dané peněžence. Obecně vypočítaný jako *příjmy – výdaje*. Zůstatek může být buď na kartě, nebo v hotovosti.

Překlad
Veškerý statický text aplikace má být překladatelný. K tomuto účelu je zde entita Překlad uchovávající daný text pod jeho kódem. Samotný kód označuje, o jaký text se jedná, společně s jazykem definuje konkrétní překlad.

2.6 Technologie

Zvolené technologie nesmí porušovat nefunkční požadavky. Proto jsme se rozhodli, že server bude využívat PHP 5.6.30. Důvodem pro tuto verzi je její stabilita. Verze PHP 5.6.0 byla vydána 28. srpna 2014[2], prošla několika nápravami chyb, až se dopracovala k verzi 5.6.30, která postrádá většinu chyb. Zároveň je dle serveru wptaven.com[3] nejpoužívanější verzí PHP. V úvahu přišla i verze PHP 7.0 a vyšší, ale tyto verze zatím nejsou dostatečně odladěny a mohou obsahovat bezpečnostní rizika.

Dále je vhodné si v daném jazyce zvolit framework, který nám díky svým předprogramovaným funkcionalitám urychlí práci. Dle serveru interval.cz[5]

je v České republice nejoblíbenějším frameworkem Nette[12], který ovšem není v neoblíbenějších deseti světově používaných frameworcích kvůli své neúplné dokumentaci. Nejoblíbenějším mezinárodním frameworkem je Laravel[13], a právě z tohoto důvodu jsme se pro něj rozhodli i my. Zvolili jsme Laravel 5.4[14], který je kompatibilní právě s námi zvolenou verzí jazyka PHP.

Klientská část aplikace, jelikož je webová, využije vestavěného jazyka prohlížeče – Javascriptu[15]. I na tento jazyk existuje celá řada frameworků. Dle serveru hackernoon.com[7] je nejoblíbenějším frameworkem AngularJS[16] od společnosti Google[17]. Jelikož Google vlastní i nejpoužívanější webový prohlížeč[8], bude zaručena kompatibilita i do budoucna pro co nejvíce uživatelů.

Data se budou ukládat do databáze MySQL v souladu s nefunkčními požadavky. Mapování dat z databáze do entitních objektů v aplikaci (ORM – Object-relational Mapping) nabízí také několik frameworků, některé přímo se zaměřující na ORM. Kvůli jednoduchosti ale zůstaneme u jednoho frameworku, tj. Laravelu.

2.7 Vhodný webhosting

Při výběru webhostingu se musíme podívat na nefunkční požadavky, jestli mezi nimi není nějaký, který nám znepříjemní výběr svým specifikem. PHP 5.6 a MySQL ovšem nabízí valná většina webhostingů.

V úvahu připadají 2 kategorie: webhostingy placené a bezplatné. Bezplatné webhostingy mívají většinou omezené funkce, vloženou reklamu, nenabízí podporu či mají časové omezení, což může v budoucnu vadit. Ale občas se najde poskytovatel, jehož omezení jsou přijatelná.

Kupříkladu 000webhost.com[18] nabízí vše námi potřebné i s podporou. Jediné omezení je, že aplikace nebude fungovat jednu hodinu denně. Můžeme si ale sami určit, která hodina to bude. Na závadu by mohlo být, že SSH (Secure shell) službu, díky níž se nám otevírá více možností například při automatizovaném nasazování, nabízí jen v placené verzi.

Další bezplatný webhosting, který splňuje naše nefunkční požadavky, je GoogieHost[19], který nabízí i SSH službu. Jediné, co by se mohlo zdát jako nevýhoda, je doména. Vlastní můžeme mít až doménu 3. řádu.

Z těch placených stojí za zmínku například Wedos[20], který je dle statistik [6] nejpopulárnějším webhostingem. Zároveň se svou cenou velmi blíží

[12] https://nette.org/
[13] https://laravel.com/
[14] https://laravel.com/docs/5.4
[15] https://www.javascript.com/
[16] https://angular.io/guide/quickstart
[17] https://www.google.com/about/
[18] https://www.000webhost.com/
[19] https://googiehost.com/
[20] https://hosting.wedos.com/cs/webhosting.html

webhostingům bezplatným, ovšem ani on nenabízí SSH službu.

Na základně této analýzy byl vybrán webhosting od GoogieHost hlavně kvůli SSH službě a bezplatnosti.

Kapitola 3

Návrh

Tato kapitola se zabývá návrhem aplikace Útrata, jehož cílem je nadefinovat si funkcionalitu, která byla specifikována v předchozí kapitole. V rámci návrhu je třeba řešit zásadní otázky software jako určit architektonický styl či zvolit persistentní úložiště.

V našem případě, kdy jsme zvolili třívrstvou architekturu serveru, je také důležité nadefinovat si, jak bude vypadat výstup prezentační vrstvy. Méně důležité pak může být, jakým způsobem mezi sebou komunikují jednotlivé vrstvy.

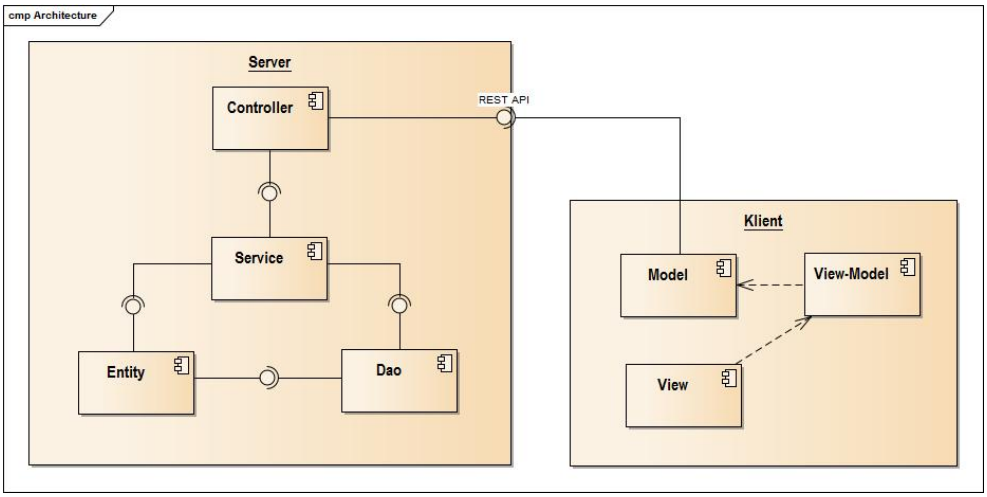
3.1 Architektura aplikace

Návrh architektury odráží nefunkční požadavky a měli bychom být při jeho tvorbě velmi opatrní. Chyby při návrhu architektury mohou způsobit velmi náročné a drahé problémy při udržování systému. Aplikace je rozdělena na dvě oddělené části – server a klient – a je nutné navrhnout architekturu obou částí zvlášť, ale také popsat, jak spolu budou tyto dvě části aplikace komunikovat.

Jak můžeme vidět na obrázku 3.1, obě části spolu komunikují přes aplikační rozhraní (API – Application Programming Interface). Toto API je typu REST (Representational State Transfer), což znamená, že komunikace probíhá pomocí protokolu HTTP (Hypertext Transfer Protocol), který umožňuje různé druhy požadavků. My budeme využívat CRUD (Create Read Update Delete) operace, tedy metody *post*, *get*, *put* a *delete* ze sady metod protokolu HTTP. Data se budou posílat textově ve formátu JSON[21] (JavaScript Object Notation).

Výhoda architektury server-klient je ta, že ke stejnému serveru může přistupovat přes API více klientů z různých platforem a server může mít svůj white list klientů, kterým bude data poskytovat.

[21] https://www.json.org/



Obrázek 3.1: Architektura

3.2 Server

Serverová část bude mít typickou třívrstvou architekturu tvořenou datovou, aplikační a prezentační vrstvou. Výhoda vícevrstvé architektury tkví v tom, že jednotlivé vrstvy můžeme nezávisle vyvíjet či měnit, aniž by to mělo dopad na chod aplikace. Také se díky tomuto konceptu aplikace lépe testuje.

Datová vrstva bude nabízet rozhraní, kterého bude využívat vrstva aplikační. Ta pak bude nabízet rozhraní, kterého bude využívat prezentační vrstva.

3.2.1 Datová vrstva

Tato vrstva se stará o ukládání, mazání a selekci dat z perzistentního úložiště, kterým je databáze MySQL. Na obrázku 3.1 ji můžeme najít jako části `Entity` a `Dao` (Data access object).

Část `Entity` se stará o samotné mapování dat z perzistentního úložiště na objekty a část `Dao` obsahuje metody, které manipulují s databází – tedy ukládání do ní, měnění dat, mazání či vyhledávání v nich.

3.2.2 Aplikační vrstva

Aplikační vrstva (někdy nazývána také business vrstva) obsahuje veškerou logiku aplikace. Na obrázku odpovídá části `Service`. Zodpovídá za správný přenos dat mezi datovou a prezentační vrstvou.

3.2.3 Prezentační vrstva

Hlavní zodpovědnost této vrstvy je přijímat požadavky a na jejich základě volat příslušné metody aplikační vrstvy a zároveň data z aplikační vrstvy

transformovat do srozumitelné podoby, tedy do podoby, kterou API předpokládá.

3.3 Klient

Framework AngularJS využívá návrhový vzor MVVM (Model-View-View Model), přičemž Model bude využívat rozhraní API, které je serverem nabízeno.

Tento framework se zaměřuje na jednostránkové aplikace (SPA – Single Page Application), což umožňuje při změně dat či při vyvolání události uživatelem znovu načíst pouze tu část stránky, v níž byla data změněna. Výhoda takového řešení je, že aktualizace stránky je rychlejší, zbytečně se nepřenášejí data, která nebyla pozměněna, a uživatel je odstíněn od prázdné obrazovky během čekání na všechna data. AngularJS je volitelně typový, objektově orientovaný jazyk s obvyklou syntaxí stylu jazyka C, který se kompiluje do jazyka Javascript, není tedy potřeba znát syntaxi Javascriptu. Je ale možné psát Javascriptový kód přímo do kódu Angularu.

Nevýhodou je neobsáhlá oficiální dokumentace tohoto frameworku. Na druhou stranu, díky faktu, že je to jeden z nejpoužívanějších frameworků pro Javascript, má velké množství tutoriálů a mnoho problémů, na které bychom mohli při vývoji narazit, už před námi někdo řešil a tudíž budou lehce dohledatelné.

Největší výhody tohoto frameworku jsou:

- Two-way-data-binding (obousměrné datové provázání),
- Dependency injection (vkládání závislostí),
- direktivy,
- šablonování,
- testovatelnost[9].

Podrobněji jsou popsány v kapitole Implementace.

3.3.1 Model

Jediná zodpovědnost této části je získat resp. zaslat data na server přes REST API. To znamená, že musí objekty překonvertovat do formátu JSON, který API přijímá, a naopak.

3.3.2 View-Model

Tato část v sobě skrývá veškerou aplikační logiku klienta. Veškeré složité operace, validace, zpracování se dějí zde. Zodpovídá tedy za správnost dat, předávaných části `View`, a musí se také správně rozhodnout, jaká data bude po části `Model` žádat.

3.3.3 View

Úkolem této části je správně zobrazovat data uživateli. V tomto případě to znamená správně označkovat data jazykem HTML (Hypertext Markup Language), aby je prohlížeč mohl správně vykreslit.

3.4 Databázový model

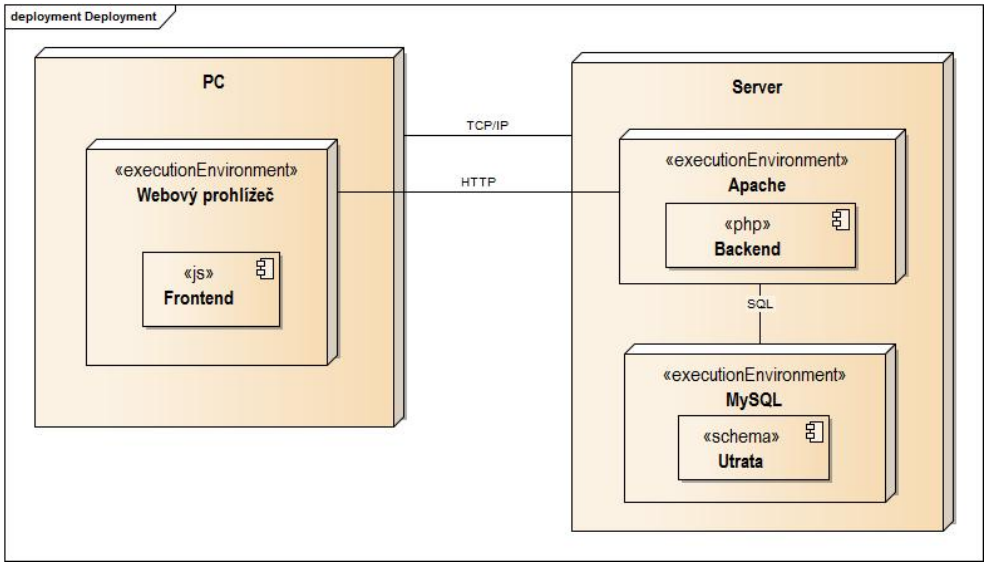
Databázový model na obrázku 3.3 vychází z doménového modelu na obrázku 2.1. Některé entity jsou shodné s třídami z modelu doménového, jako například entita Uživatel se zde reprezentuje jako `member`, některé entity vznikly kompozicí více tříd (Položka, Příjem a Výdaj se mapují na `item`) nebo se zde nenachází vůbec (Zůstatek není potřeba ukládat do perzistentního úložiště, protože je vždy spočítán).

Všechny entity mají prefix `utrata_`, aby nedocházelo ke kolizi tabulek v případě, kdyby ve stejné databázi měla tabulky i jiná aplikace.

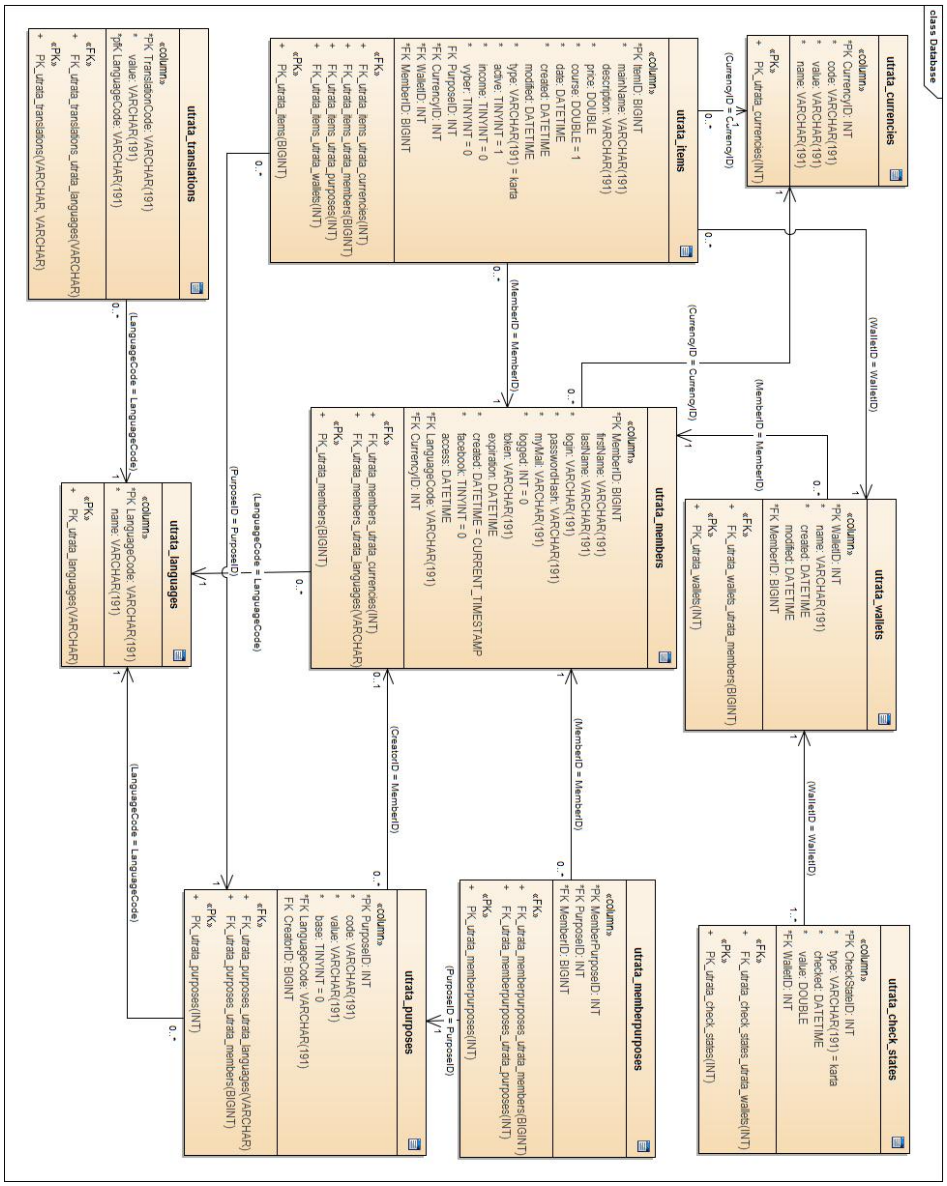
Podle tohoto modelu budeme následně implementovat entitní třídy v datové vrstvě serveru.

3.5 Nasazení

Diagram nasazení zachycuje hardware, na němž bude Útrata nasazena, spolu se softwarem. Tento model mapuje architekturu software na fyzickou architekturu, kde bude tento software spuštěn. Diagram můžeme vidět na obrázku 3.2.



Obrázek 3.2: Model nasazení



Obrázek 3.3: Databázový model

Kapitola 4

Implementace

V této kapitole je rozepsána zvlášť implementace serverové části a části klientské. Pro vývoj obou částí bylo použito IDE (Integrated Development Environment) PhpStorm[22] od JetBrains[23] se studentskou licencí. K verzování byl využit nástroj Git[24] a vzdálený repozitář GitLab[25].

4.1 Server

Při vývoji serverové části aplikace bylo využito serveru Xampp[26], z jehož nabízených částí jsme využili Apache[27], PHP interpreter a MySQL databázi.

Nejdůležitější bylo správně napsat ORM třídy a nakonfigurovat směrování URI (Uniform Resource Identifier) na konkrétní metodu příslušného kontroleru.

4.1.1 Datová vrstva

Datová vrstva, jak bylo znázorněno na obrázku 3.1, sestává z částí Entity a Dao. Část Entity se stará o mapování dat z relační databáze do objektů a část Dao s těmito objekty pak pracuje na nejnižší úrovni, tj. CRUD operace.

Zároveň bylo třeba napsat migrační třídu, která vytvořila strukturu databáze.

4.1.1.1 Vytvoření databáze

Bylo nutné vytvořit migrační třídu, tj. třídu, kde je nakonfigurováno, jak se bude která tabulka jmenovat, jaké bude mít sloupce, co bude její primární

[22] https://www.jetbrains.com/phpstorm/specials/phpstorm/phpstorm.html
[23] https://www.jetbrains.com/
[24] https://git-scm.com/
[25] https://about.gitlab.com/
[26] https://www.apachefriends.org/index.html
[27] https://www.apache.org/

```
1   <?php
2
3   use ...
6
7   class CreateWalletsTable extends Migration
8   {
9       /** Run the migrations. ...*/
14      public function up()
15      {
16          Schema::create('utrata_languages', function(Blueprint $table) {
17              $table->string('LanguageCode')->primary();
18              $table->string('name');
19          });
20      }
21
22      /** Reverse the migrations. ...*/
27      public function down()
28      {
29          Schema::dropIfExists('utrata_languages');
30      }
31  }
```

Obrázek 4.1: Ukázka migrační třídy

popřípadě cizí klíč. Třída musí dědit z třídy `Migration`, aby framework věděl, že jde o migrační třídu. Ukázka takové třídy je na obrázku 4.1, kde jsou dvě metody.

První metoda `up()` se stará o vytvoření (případně upravení) tabulky nadefinovanými sloupci. Proto musíme vyplnit název tabulky a sloupce s příslušnými příznaky. Na ukázce vytváříme tabulku jménem `utrata_languages` se dvěma textovými sloupci `LanguageCode` a `name`, přičemž `LanguageCode` je primárním klíčem tabulky. Další definice cizích klíčů, relací aj. jsou popsány v dokumentaci[28].

Druhá metoda `down()` zodpovídá za destrukci tabulky v databázi.

Jelikož Laravel disponuje CLI (Command-line Interface), dají se tyto metody zavolat z příkazové řádky příkazem `php artisan migrate` resp. `php artisan migrate:fresh`.

4.1.1.2 Část Entity

Ve frameworku Laravel k ORM slouží třída `Model` (známější pro vyhledávání na Internetu a v oficiální dokumentaci jako `Eloquent` dle názvu adresáře, ve kterém se nachází). Pokud chceme vytvořit vlastní mapovací třídu, musí naše nová třída dědit z třídy `Model`, jak je to znázorněno na obrázku 4.2.

Třídě se musí nastavit jméno tabulky, ze které má mapovat. K tomu slouží atribut `$table` na řádku 14. Pokud má tabulka jiný primární klíč než `id`, je nutné ho nadefinovat, jak je to na řádku 16. Jako další důležitý atribut je

[28] https://laravel.com/docs/5.4/migrations

```
10  use Illuminate\Database\Eloquent\Model;
11
12  class Currency extends Model
13  {
14      protected $table = 'utrata_currencies';
15
16      public $primaryKey = 'CurrencyID';
17
18      public $timestamps = false;
19
20
21      /**
22       * @return int
23       */
24      public function getId() {
25          return $this->CurrencyID;
26      }
```

Obrázek 4.2: Ukázka výňatku entitní třídy

```
55  /*###################  DAO ########################*/
56  $app->app->bind('App\Model\Dao\ICheckStateDAO', 'App\Model\Dao\CheckStateDAO');
57  $app->app->bind('App\Model\Dao\ICurrencyDAO', 'App\Model\Dao\CurrencyDAO');
58  $app->app->bind('App\Model\Dao\IItemDAO', 'App\Model\Dao\ItemDAO');
59  $app->app->bind('App\Model\Dao\ILanguageDAO', 'App\Model\Dao\LanguageDAO');
60  $app->app->bind('App\Model\Dao\IMemberDAO', 'App\Model\Dao\MemberDAO');
61  $app->app->bind('App\Model\Dao\IPurposeDAO', 'App\Model\Dao\PurposeDAO');
62  $app->app->bind('App\Model\Dao\ITranslationDAO', 'App\Model\Dao\TranslationDAO');
63  $app->app->bind('App\Model\Dao\IWalletDAO', 'App\Model\Dao\WalletDAO');
64  $app->app->bind('App\Model\Dao\IMemberPurposeDAO', 'App\Model\Dao\MemberPurposeDAO');
```

Obrázek 4.3: Ukázka přiřazení implementace k rozhraní

`$timestamp` (řádek 18), který rozhoduje, jestli se v tabulce mají hledat sloupce `created` a `modified`. V ukázce se mapuje tabulka, která tyto sloupce nemá.

Samotná třída uchovává ve výchozím stavu dle frameworku data z jednotlivých sloupců ve veřejně přístupných atributech, které se jmenují stejně jako sloupce tabulky. Proto bylo v rámci objektového zapouzdření nutné napsat metody (gettery a settery) pro tyto atributy.

4.1.1.3 Část Dao

Každé třídě této části jsme nejprve napsali rozhraní, které říká, jaké metody s jakými parametry a s jakým návratovým kódem bude daná třída mít. Díky tomuto vzoru, kdy v aplikační vrstvě využíváme dependency injection s rozhraními a nikoli s třídami samotnými, jsme schopni snadno vyměnit všechny Dao implementace těchto rozhraní, aniž bychom jakkoliv zasahovali do aplikační vrstvy. Aby ale aplikace věděla, jaká implementace daného rozhraní se má použít, je nutno tuto skutečnost uvést v konfiguračním souboru, jako je to na obrázku 4.3.

```php
class CurrencyService implements ICurrencyService
{
    /** @var ICurrencyDAO */
    protected $currencyDao;

    /**
     * CurrencyService constructor.
     * @param ICurrencyDAO $currencyDAO
     */
    public function __construct(ICurrencyDAO $currencyDAO) {
        $this->currencyDao = $currencyDAO;
    }

    /**
     * @return Currency[]
     * @throws NotFoundException
     */
    public function getCurrencies() {
        $currencies = $this->currencyDao->findAll();
        if (count($currencies) == 0)
            throw new NotFoundException('No Currency found');
        return $currencies;
    }
```

Obrázek 4.4: Ukázka aplikační vrstvy

Předpisy metod v rozhraní jsou většinou CRUD, ale ne vždy byly zapotřebí všechny tyto metody. Například třídě `LanguageDao` stačí metody pro selekci, protože součástí práce není implementovat administrátorské funkce, tj. správa jazyků.

V metodách pak využíváme entitních tříd pro selekci či perzistenci.

4.1.2 Aplikační vrstva

I třídy této vrstvy mají své rozhraní, s nimiž pracuje vrstva prezentační, a tudíž se může vyměnit implementovaná třída bez zásahu v prezentační vrstvě.

Typicky zde existují třídy pojmenované podle databázových tabulek a pracují s příslušnou Dao třídou, jako je to na obrázku 4.4, kdy `CurrencyService`, která implementuje rozhraní `ICurrencyService`, má vloženou závislost na `ICurrencyDao`. Každá třída v této vrstvě má typicky vloženou jednu Dao třídu, ale může mít vloženy další třídy z aplikační vrstvy. Kupříkladu třída `CsvService`, která implementuje rozhraní `IFileService`, je závislá na celé řadě dalších tříd z této vrstvy, protože potřebuje data o položkách, peněženkách, uživateli a jeho sekcích útraty apod.

V samotných metodách pak probíhají kontroly vstupních dat a následně všechny složité operace jako verifikace hesla uživatele, spočítání zůstatku na kartě v dané peněžence, upravení sloupce všech položek splňujících daný filtr, zkontrolování duplicity položky, naformátování uživatelských dat do formátu CSV apod.

4.1.3 Prezentační vrstva

Zde se nachází veškeré kontrolery (Controllers), jejichž starostí je přijímat data z API, konvertovat do objektů a poskytnout nižší vrstvě, a naopak data z nižší vrstvy konvertovat do formátu předpokládaného API. I pro tento účel využijeme framework Laravel. Stačí, aby námi napsané kontrolery dědily z třídy `Controller`. My jsme si však vytvořili ještě abstraktní třídu `AbstractController`, která dědí z frameworkového kontroleru a všechny námi napsané kontrolery dědí až z této abstraktní třídy. Dědičnost je tedy zachována a v této abstraktní třídě jsme si navíc vynutili závislost na `MemberService`, což nám umožní z jakéhokoliv kontroleru zjistit přes tuto třídu, jestli je uživatel přihlášen (obrázek 4.5, řádek 95).

Framework nám velmi usnadňuje i práci s požadavky (Request) na server. Každá metoda kontroleru může mít jako první parametr `Request`, který pak obsahuje veškeré informace o požadavku, nejčastěji jsme však využívali obsah s daty (obrázek 4.5, řádek 97). `AbstractController` ovšem využívá hlavičku požadavku, ve kterém je uveden uživatelův token, na základě něhož se rozhoduje, jestli je uživatel přihlášen.

Konverze objektů do formátu JSON, který je požadován API, je díky frameworku také velmi jednoduchá. Metoda vrátí odpověď na požadavek (Response), která má při vytváření dva nepovinné parametry:

- data určená ke konverzi jako pole,
- HTTP stavový kód[29].

Výchozí stav, pokud se neuvede ani jeden parametr, je prázdný obsah (blank) a statový kód 204 (No Content). Tato volba se využívá jen v případě, kdy klient neočekává žádnou odpověď. Na obrázku 4.5 vidíme příklad úspěšné odpovědi (řádek 110) a několik odpovědí informujících o chybě (řádky 103, 105 a 107).

4.1.4 Komunikace mezi vrstvami

Komunikace mezi datovou a aplikační vrstvou je celkem prostá. V případě, že se akce v datové vrstvě provede správně, tak se vrací ona entita, se kterou se pracuje. Pokud dojde k neúspěchu, je návratovou hodnotou $NULL$.

Komunikace mezi aplikační a prezentační vrstvou je o něco složitější. Při úspěchu se, stejně jako u komunikace datová vrstva – aplikační vrstva, předává daná entita či jiná data, ale při chybě budou vyhazovány výjimky. Tyto výjimky budou v prezentační vrstvě zachycovány a podle typu výjimky budou předány příslušné chybové zprávy spolu s HTTP kódem do API, jak je znázorněno na obrázku 4.5.

[29] https://www.interval.cz/clanky/stavove-kody-a-hlaseni-v-odpovedi-protokolu-http/

```
90      /** @param Request $req ...*/
94      public function create(Request $req) {
95          $this->assumeLogged($req);
96          $member = $this->loggedUser($req);
97          $data = $req->get('item');
98
99          try {
100             $item = $this->itemService->createItem($member, $data);
101             $formatted = $this->itemService->format($item);
102         } catch (AlreadyExistException $e) {
103             return Response::create(['error' => $e->getMessage()], Response::HTTP_CONFLICT);
104         } catch (NotFoundException $e) {
105             return Response::create(['error' => $e->getMessage()], Response::HTTP_NOT_FOUND);
106         } catch (BadRequestHttpException $e) {
107             return Response::create(['error' => $e->getMessage()], Response::HTTP_BAD_REQUEST);
108         }
109
110         return Response::create(['success' => $formatted], Response::HTTP_CREATED);
111     }
```

Obrázek 4.5: Ukázka prezentační vrstvy

```
47  /*##################-  Item  -#####################*/
48  Route::get('/items/wallet/{walletId}', 'ItemController@getWalletItems');
49  Route::get('/items/statistics', 'ItemController@statistics');
50  Route::get('/item', 'ItemController@get');
51  Route::post('/item', 'ItemController@create');
52  Route::put('/item', 'ItemController@update');
53  Route::put('/item/check/{id}', 'ItemController@check');
54  Route::put('/items/check', 'ItemController@checkAll');
55  Route::delete('/item/{id}', 'ItemController@delete');
```

Obrázek 4.6: Ukázka nastavení mapování URI na konkrétní metodu

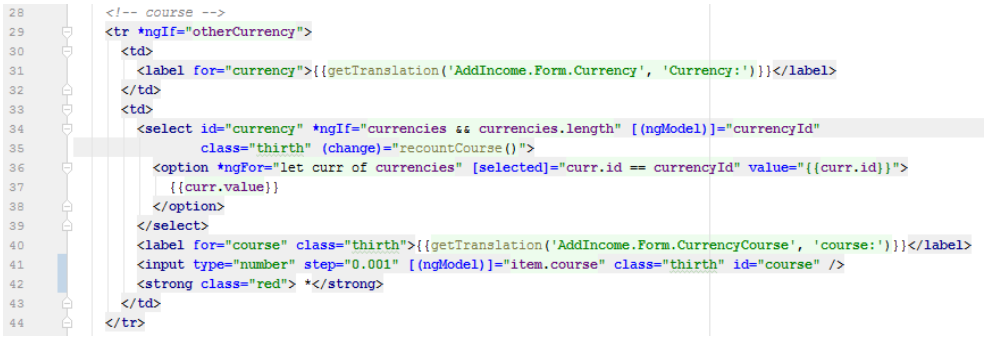
4.1.5 Mapování URI na metodu v kontroleru

Bylo nutné nakonfigurovat, při jakém URI se má volat jaká metoda. K tomu slouží konfigurační soubor frameworku `routes/web.php`, který je na obrázku 4.6. Uvádí se HTTP metoda, URI a poté zavináčem oddělen kontroler a jeho metoda.

4.2 Klient

Klientská část funguje jako SPA (Single-page application) a každá obrazovka se vždy skládá z jednotlivých komponent. Například úvodní stránka po přihlášení se skládá z komponenty menu a komponenty seznam peněženek. Je využito návrhového vzoru MVVM tvořeného ze tří propojených částí, jak je to znázorněno na obrázku 3.1.

Výhoda Angularu je, že má CLI. Sestavení aplikace za účelem vývoje se dělá příkazem `ng serve`, který zkompiluje zdrojové kódy do Javascriptu a aplikace je dostupná na URL *http://localhost:4200*. Při uložení změn ve zdrojovém kódu se aplikace automaticky překompiluje a v prohlížeči se znovu načte.



Obrázek 4.7: Ukázka direktiv a vázání dat v klientovi

4.2.1 Komponenty

Komponenta je část aplikace zodpovídající za určitou část funkcionality a z komponent se staví celá aplikace. Komponenta se vytváří nejlépe přes CLI příkazem `ng generate component X`, který automaticky vytvoří všechny potřebné soubory a zavede komponentu do seznamu komponent v konfiguračním souboru.

Základní komponenta celého klienta je `app.component` sestávající, jako každá komponenta, ze čtyř souborů: `app.component.html`, `app.component.css`, `app.component.ts` a `app.component.spec.ts`.

Soubory `.html` a `.css` tvoří část View, soubor `.ts` zastupuje část View-Model a soubor `spec.ts` slouží k testování komponenty.

Popíšeme si tyto soubory na nějaké obecné komponentě `X`:

> **`X.component.html`** je šablona, do které se značkují data do HTML jazyka. Lze zde využívat jak čistého HTML, tak také direktiv či data-binding frameworku AngularJS.
>
> Direktivy jsou speciální atributy HTML tagů, jako je například `*ngIf` (obrázek 4.7, řádek 29), díky níž se zobrazí řádek tabulky (tag `tr`) pouze v případě, že je splněna podmínka v `*ngIf` – tedy že nabývá proměnná `otherCurrency` hodnotu *true*. Nebo direktiva `*ngFor` (obrázek 4.7, řádek 36), která nám umožňuje iterovat kolekcí a pracovat tak postupně s každým prvkem samostatně. Na ukázce to znamená, že se pro každý prvek kolekce vykreslí HTML tag `select`, který bude mít vyplněné atributy podle hodnot jednotlivých prvků kolekce. Vedle direktiv předdefinovaných frameworkem si můžeme tvořit vlastní direktivy.
>
> Two-way-data-binding je další velká výhoda Angularu, která nám umožňuje přistupovat z šablony k proměnným ve View-Model a zároveň se po změně proměnné v šabloně data aktualizují. Na obrázku 4.7 můžeme tuto techniku vidět na řádku 34 (`[(ngModel)]="currencyId"`),

```
11    @Component({"selector": 'app-wallet-content'...})
16    export class WalletContentComponent implements OnInit {
17      @ViewChild(Ng2PopupComponent) popup: Ng2PopupComponent;
18
19      @Input() wallet: Wallet;
20      items: Item[] = [];
21      thisComponent: any = this;
22      filter: {order: string, asc: string} = {order: '', asc: ''};
23
24      constructor(
25          private route: ActivatedRoute,
26          private walletService: WalletService,
27          private translationService: TranslationService,
28          private itemService: ItemService,
29      ) {}
30
31      ngOnInit(): void {
32        this.getWallet();
33        this.initializeItems();
34      }
35
36      getTranslation(code: string, defaultCode: string = ""): string {
37        return this.translationService.getTranlation(code, defaultCode);
38      }
```

Obrázek 4.8: Ukázka View-Model klienta

kdy se hodnota `currencyId` bude měnit v závislosti na vybrané možnosti ze select–listu.

Na řádku 35 se také nachází event-binding (`(change)="func()"`), díky němuž se zavolá funkce `func()` ve View-Model při změně tohoto HTML tagu.

`X.component.css` je soubor, kde se definují CSS (Cascading Style Sheets – kaskádové styly) k dané komponentě. Spolu s šablonou tvoří část View.

`X.component.ts` odpovídá části View-Model a obsahuje většinu logiky klienta. Využívá dependency-injection, jak můžeme vidět na obrázku 4.8 v konstruktoru, a dále definuje funkce a proměnné, které jsou přístupné z šablony. Konstruktor slouží převážně na dependency-injection, na ostatní inicializační procesy

`X.component.spec.ts` je soubor, který slouží k jednotkovému otestování dané komponenty.

Výhoda konceptu komponent je, že jednotlivé komponenty mohou mít v sobě další komponenty. Tohoto jsme využili při implementaci základní komponenty aplikace `app.component`, kdy jsme do šablony vložili pouze frameworkovou komponentu `<router-outlet>`, která je zodpovědná za mapování komponent na jednotlivé URI, jak to můžeme vidět na obrázku 4.9.

```
11  const routes: Routes = [
12    { path: 'login', component: LoginComponent },
13    { path: 'dashboard', component: DashboardComponent },
14    { path: 'wallet/:id', component: WalletContentComponent },
15    { path: 'wallet/:id/seeOld', component: OldItemsComponent },
16    { path: 'wallet/:id/incomes', component: IncomesComponent },
17    { path: 'wallet/:id/preview', component: MonthlyPreviewComponent },
18    { path: '', redirectTo: '/login', pathMatch: 'full' },
19  ];
20
21  @NgModule({
22    imports: [ RouterModule.forRoot(routes) ],
23    exports: [ RouterModule ]
24  })
25  export class AppRoutingModule { }
```

Obrázek 4.9: Ukázka mapování URI klienta

4.2.2 Části klienta

Již jsme si zmínili, že šablona `X.component.html` a `X.component.css` tvoří část View a že `X.component.ts` odpovídá části View-Model. Zbývá ještě popsat část Model.

Třídy v této části se nazývají servisy a podle toho se i jmenují. Servisy mají za úkol komunikovat se serverem a předávat získaná data překonvertovaná z JSONu do objektů. Jelikož Angular funguje asynchronně, nevrací servisy přímo překonvertované objekty, ale jen příslib (promise) objektů, aby aplikace nečekala zbytečně dlouho na odpověď od serveru. Ve výsledku to znamená, že servis vrátí jen příslib, a až dostane odpověď od serveru, pošle i data. Pokud trvá komunikace mezi serverem a klientem dlouho, uživateli se zdá, že se jednotlivé části stránky postupně donačítají.

Servisy se vytvářejí v CLI příkazem `ng generate service X`, který vytvoří servis, příslušný `spec.ts` soubor sloužící k testování servisu a zavede servis do seznamu servisů v konfiguračním souboru `app.module.ts`.

4.2.3 Sestavení aplikace

Aby vše správně fungovalo, musíme do konfiguračního souboru `app.module.ts` uvést veškeré komponenty a servisy, které chceme používat. Komponenty do pole `declarations` a servisy do pole `providers`.

4.2.4 Propojení s třetí stranou

Aplikace využívá třetí strany ve dvou případech: při získávání kurzu mezi měnami a při registraci či přihlášení pomocí Facebooku.

U získávání kurzu se volá externí URI $http://api.fixer.io/latest?base=EUR\&to=CZK$, které vrací potřebná data. V případě této URI je to aktuální kurz z měny EUR na měnu CZK ve formátu JSON.

```
1   image: php:5.6
2
3   cache:
4     paths:
5       - vendor/
6
7   stages:
8     - test
9     - deploy
10
11  test:
12    before_scr...: <10 items>
23    stage: test
24    script:
25      - ./vendor/bin/phpunit --configuration phpunit.xml
26
27  deploy:
28    before_script:
29      - apt-get update
30      - apt-get install -y ftp
31    stage: deploy
32    script:
33      - chmod +x CI-scripts/transfer.sh
34      - ./CI-scripts/transfer.sh
```

Obrázek 4.10: Ukázka konfigurace CI

Pro přihlášení přes Facebook bylo třeba stáhnout si facebookSDK[30] (Software development kit) a pomocí dependency injection naimportovat FacebookService a ten používat jako každý jiný servis, tj. volat metody, které vrací příslib na data.

4.3 Nasazení

Webhosting, který uváděl, že poskytuje SSH přístup, ve skutečnosti SSH službu nemá, ale umožňuje přesun souborů pomocí FTP (File Transport Protocol). Využili jsme tedy tohoto protokolu k automatizaci nasazování zdrojových kódů na server. Automatizaci zaštiťuje GitLab, který podporuje CI (Continuous Integration). Bylo třeba CI nakonfigutovat, což se dělá v souboru `gitlab-ci.yml`, který se musí nacházet v kořenovém adresáři repozitáře a je ve formátu YAML (Ain't Markup Language). CI se poté spouští při každém push na GitLab.

Na obrázku 4.10 je konfigurační soubor pro CI serverové části. GitLab pracuje s Dockerem[31], který si stáhne vhodný image pro daný projekt (řádek 1). Pokud náš projekt využívá externích knihoven, je vhodné je nestahovat při každém sestavení (build), ale uchovávat je v cache (řádek 3 – 5). Následně se

[30] https://www.npmjs.com/package/ng2-facebook-sdk
[31] https://www.docker.com/

uvede, jaké akce (jobs) se mají v rámci nestavení provést. V našem případě je to otestovat projekt a následně nakopírovat zdrojové soubory na server. Každá akce může mít také nadefinované, co se má provést, než se začne se samotnou akcí, jako například stažení potřebných programů. Na řádku 30 například instalujeme FTP pro přenos souborů a samotná akce spustí jen soubor `CI-scripts/transfer.sh`, jenž se postará o přenos souborů.

Akce se provádí v pořadí, v jakém jsou uvedeny (řádky 8 – 9). Pokud bude mít nějaká akce návratový kód různý od nuly, další akce se přeskočí. To znamená, že pokud testy selžou, zdrojový kód se nenakopíruje na server.

Kapitola 5

Testování

Každá aplikace má být řádně otestována, aby se ověřila její kvalita. Některé testy slouží do budoucna jako regresní – tedy aby se po rozšíření aplikace ověřilo, že stará část funguje stále stejně dobře.

Proto se v této kapitole podíváme na testy jednak automatizované, které budou později sloužit jako již zmiňované testy regresní, a jednak testování za pomoci testerů.

5.1 Automatizované testy

Na server bylo použito white-box testing (jednotkové testy) i black-box testing (systémové testy). Díky CLI Laravelu se obě skupiny testů spouštějí z příkazové řádky za pomoci příkazu:

```
./vendor/bin/phpunit --configuration phpunit.xml
```

Do konfiguračního souboru `phpunit.xml` se uvádí například typ prostředí – tedy testing – nebo jaká databáze se bude používat, protože testování by mělo mít svou vlastní databázi, případně úplně jiné perzistentní úložiště, a kde se v adresářové struktuře nacházejí testy.

Klient byl testován pouze jednotkovými testy. Jelikož i Angular disponuje CLI, testy se spouštějí příkazem:

```
ng test --watch=false
```

Testování ve výchozím nastavení probíhá tak, že vývojář mění kód a testy se automaticky spouští při každé uložené změně kódu. Pro zabránění tomuto chování je zde uveden přepínač `--watch=false`, díky němuž se testy provedou jen jednou a příkaz skončí.

5.1.1 Jednotkové testy serveru

Jednotkové testy se zaměřují na malou část kódu. Typicky jde o jeden test pro jednu metodu nějaké třídy. Laravel umožňuje udělat každou třídu testo-

```
35  protected function setUp() {
36      parent::setUp();
37      $this->purposeService = new PurposeService(
38          new FakePurposeDAO(),
39          new FakeLanguageService(),
40          new FakeMemberPurposeService()
41      );
42      $this->member = (new FakeMemberService())->getMember('vojta');
43      $this->member2 = (new FakeMemberService())->getMember('jožka');
44  }
45
46  public function testGetLanguagePurposes() {
47      $purposes = $this->purposeService->getLanguagePurposes($this->languageCode);
48
49      $this->assertEquals(2, count($purposes));
50      foreach ($purposes as $purpose)
51          $this->assertEquals($this->languageCode, $purpose->getLanguage()->getCode());
52  }
```

Obrázek 5.1: Ukázka testovací třídy jednotkového testu

vací, pokud dědí ze třídy `TestCase`. Za testovací metodu takové třídy se bere metoda, jejíž jméno začíná prefixem `test`, nebo má anotaci `@test`.

Jelikož veškeré složité procesy a operace jsou umístěné v aplikační vrstvě, byla jednotkovými testy testována pouze tato vrstva.

Díky konceptu dependency injection rozhraní a ne přímo implementace tříd jsme vytvořili mock třídy, které tato rozhraní implementují, a vložili je do konstruktoru testovaných tříd aplikační vrstvy (obrázek 5.1, řádek 37 – 41). V takové situaci jsme vždy věděli, jaká data budou mock třídy vracet, a tedy i výsledek, jaký má testovaná třída produkovat.

Třída `TestCase` nám nabízí mnoho testovacích metod, které můžeme díky dědičnosti využívat. Nejčastěji jsme využili metod `assertEquals`, která má dva parametry: předpokládaná hodnota a testovaná hodnota. Použití této metody je vidět na obrázku 5.1, řádek 49, kde předpokládáme, že pole bude velikosti 2.

Díky tomuto testování byly odhaleny a opraveny následující nedostatky software:

- špatná podmínka při vytváření položky,
- neošetření vlastníka peněženky při selekci položek v peněžence,
- několik neodchycených výjimek.

5.1.2 Systémové testy serveru

Systémové testy se zaměřují na konkrétní funkci aplikace, aniž by věděly cokoliv o vnitřní struktuře aplikace. V našem případě jsme serveru přes URI posílali jednotlivé dotazy a testovali jsme, jaké posílá odpovědi. Jelikož systémové testy pracují i s databází, bylo nutné vytvořit databázi pro účely tes-

```
25    public function testGetWalletBadId() {
26        $resource = $this->get('/wallet/asd', ['Authorization' => $this->memberVojtaToken]);
27        $resource->assertStatus(400);
28        $resource->assertJson(['error' => 'WalletService: Not INTEGER or smaller than 1.']);
29    }
```

Obrázek 5.2: Ukázka systémového testu

```
13    protected function setUp() {
14        parent::setUp();
15        Artisan::call('migrate:refresh', ['--seed' => '']);
16        Artisan::call('db:seed', ['--database' => 'sqlite']);
17    }
```

Obrázek 5.3: Ukázka metody připravující databázi

tování. Nejjednodušší databáze, kterou lze frameworkem použít, je SQLite[32], pro kterou není potřeba žádného externího serveru. Data i se strukturou tabulek jsou uložena v jediném souboru na souborovém systému. Proto jsme tuto databázi k testovacím účelům využili. Zapotřebí bylo jen nakonfigurovat v souboru `phpunit.xml`, že perzistentní úložiště bude SQLite. Jednoduchosti této databáze jsme využili i v testování na GitLabu.

Stejně jako jednotkové testy, tak i třídy systémových testů dědí ze třídy `TestCase`. Pro systémové testy jsme využili také zděděných metod, ale tentokrát to byly metody určené přímo k volání REST (Representational State Transfer) API. Na obrázku 5.2, řádku 26 je vidět volání metodou *get* obsah peněženky o nečíselném identifikátoru, proto je očekávaná chybová hláška.

Jelikož se testovací třídy mohou spouštět v náhodném pořadí, bylo nutné zaručit, že před každým testem bude databáze ve stejném stavu. Proto všechny systémové testovací třídy mají ve své metodě `setUp()` (obrázek 5.3) příkazy, které smažou data v databázi a vloží vždy stejná data. Tato metoda se volá vždy před spuštěním každého testu.

Systémové testy žádnou chybu neodhalily.

5.1.3 Jednotkové testy klienta

Jak jsme si již zmínili v předešlé kapitole, k jednotkovým testům slouží `spec.ts` soubory. Stejně jako jsme u serveru vkládali servisům mock třídy implementující stejné rozhraní, jako servis očekával v konstruktoru, tak jsme i u klienta využili podobného principu. Rozdíl je v Angularu ten, že mock třídy nemusí implementovat stejné rozhraní, ale stačí, když budou mít stejné metody, tj. stejné názvy, návratové hodnoty a počty a typy parametrů. Nicméně bylo nutné uvést, jaká mock třída se má použít místo originální třídy (obrázek 5.4, řádky 18–20).

[32] https://www.sqlite.org

```
describe('NoteService', () => {
  beforeEach(() => {
    TestBed.configureTestingModule({
      imports: [HttpClientTestingModule],
      providers: [
        NoteService,
        {provide: CookieService, useClass: FakeCookieService},
        {provide: RoutingService, useClass: FakeRoutingService},
        {provide: LogService, useClass: FakeLogService},
      ],
    });
  });

  it('should be created', inject([NoteService], (service: NoteService) => {
    expect(service).toBeTruthy();
  }));

  it('some test', inject([HttpTestingController, NoteService],
      (httpMock: HttpTestingController, service: NoteService) => {
    const mockNotes = { purposes: [...] };
    service.getLanguageNotes('CZK').subscribe(res => {
      expect(res).toEqual(mockNotes);
      console.log('everything ok');
    });
  }));
});
```

Obrázek 5.4: Ukázka jednotkového testu klienta

Následuje seznam funkcí se shodným názvem `it()`, což jsou jednotlivé testy. Funkce má 3 parametry: `expectation` (textový název vypovídající, co test dělá), `assertion` (funkce, jejíž tělo obsahuje samotné testy) a `timeout` (číslo udávající čas v milisekundách pro simulaci prodlevy asynchronní komunikace. Uvedeno být nemusí a výchozí hodnota je 0).

První takový test vždy testuje, jestli se příslušná testovaná třída vytvořila, a jmenuje se „*should be created*“.

5.2 Testování s testery

Byl vytvořen testovací scénář pokrývající svým obsahem většinu funkčních požadavků na aplikaci, dostupný na *https://goo.gl/forms/IpiyWtpBPKH6ziz03*. Vynecháno bylo testování změny jazyka, protože pro některé lidi by mohlo být složité se na stránce vyznat v jiném jazyce. Registrace proběhla přes Facebook účet, který aplikaci poskytl informaci, jaký jazyk uživatel používá. Dále se netestoval měsíční přehled, protože by bylo pro testery časově velmi náročné vytvořit datový podklad pro otestování tohoto funkčního požadavku. Oba tyto požadavky byly ale otestovány vývojářem a pracují správně. Předmětem tohoto testování byla především kontrola správné funkčnosti, proto nebyl UI předmětem testování.

Z tohoto testování vyplynulo, že by se k aplikaci hodil prvotní průvodce (wizard), o kterém je možno uvažovat do budoucího rozšiřování aplikace. Dále jsme přišli na to, že některé překlady byly nepřesné či nekonzistentní v rámci celé aplikace, a ty jsme opravili. Objevily se i stížnosti na postupné donačítání stránky způsobené asynchronním chováním Javascriptu. Dále už jen testeři navrhovali, jak by se jim více líbilo grafické zpracování některých prvků.

Jako pozitivum testeři shledali propojení s Facebookem, přehlednost aplikace, široké možnosti filtrování položek nebo aktuální stav financí na kartě resp. v hotovosti, který se aktualizuje s každou změnou v příjmech či výdajích bez nutnosti aktualizovat stránku.

Závěr

Cílem práce bylo analyzovat dosavadní podobné aplikace, doménu, zdokumentovat požadavky, vybrat vhodnou technologii a navrhnout architekturu. Toto všechno bylo splněno. Útrata spojuje důležité vlastnosti současných podobných aplikací řešících stejný problém a navíc nabízí přihlašování resp. registraci přes Facebook. Z technologií byl pro serverovou část vybrán framework Laravel postavený na PHP 5.6, pro klientskou část AngularJS, který se kompiluje do Javascriptu. Požadavky byly pokryty případy užití, které se následně staly předmětem implementace. Byla promyšlena a navrhnuta databázová struktura, architektura software i model nasazení. Implementovat jsme měli aplikaci, která bude umožňovat správu osobních financí, a otestovat ji. Aplikace byla implementována a byly vytvořeny jednotkové a systémové testy a aplikace byla otestována i vybranými uživateli z různých věkových skupin.

Aplikace je dle testerů dobře využitelná a jejich hodnocení je 3,8 na škále 0–5 (0 znamená nepoužitelné).

Pro aplikaci by bylo v budoucnu vhodné implementovat administrátorskou sekci. Tato sekce by byla využita pro správu překladů, tj. přidat překlady pro další jazyky, jelikož nyní je aplikace přeložena jen do češtiny a angličtiny. Dále by administrátoři mohli nastavit hojně používané sekce útraty v daném jazyce jako výchozí. Výchozí sekce útraty jsou nyní pouze *jídlo* a *ostatní* v češtině a jejich ekvivalenty v angličtině a v případě změny jazyka uživatelem se mu tyto výchozí vyberou jako jím používané. Také se uvažuje o propojení s bankovním účtem, aby byl uživatel odstíněn od manuálního vkládání položek při pohybu financí na kartě, kterých je v porovnání s hotovostí většina. V neposlední řadě v dalším rozvoji aplikace je plánován průvodce prvním použitím aplikace, aby měl nový uživatel povědomí o funkcích aplikace. Toto vylepšení přišlo v úvahu na základě zpětné vazby od testerů. Za úvahu v budoucnu stojí změna webhostingu, který by umožňoval vzdálený přístup k databázi. Ten je důležitý při automatickém nasazování změn, které se dotknou databázového návrhu. Jelikož se aplikace špatně ovládá na mobilních zařízeních, bylo by také vhodné vytvořit klienta pro Android případně pro iOS zařízení.

Literatura

[1] Živě.cz; Anketa: *Které službě pro správu financí dáváte přednost?*; [online]; 2016, [cit. 2018-03-03]; Dostupné z: https://www.zive.cz/clanky/nejlepsi-aplikace-pro-spravu-osobnich-i-rodinnych-financi/sc-3-a-181550/default.aspx

[2] The PHP Group; *Unsupported Historical Releases*; [online]; 2014, [cit. 2018-03-03]; Dostupné z: http://php.net/releases/#5.6.0

[3] GOODING Sarah; *PHP 5.6 Is Now the Most Widely Used PHP Version*; [online]; 2017-03-23, [cit. 2018-03-05]; Dostupné z: https://wptavern.com/php-5-6-is-now-the-most-widely-used-php-version

[4] Google, Inc.; *Browser support*; [online]; 2018, [cit. 2018-03-05]; Dostupné z: https://angular.io/guide/browser-support

[5] POKORNÝ Jan; *10 nejlepších PHP frameworků pro vývojáře*; [online]; 2015-10-29, [cit. 2018-03-12]; Dostupné z: https://www.interval.cz/clanky/10-nejlepsich-php-frameworku-pro-vyvojare/

[6] PorovnejHosting.cz; *Nejpopulárnější webhosting*; [online]; 2018, [cit. 2018-03-12]; Dostupné z: https://porovnejhosting.cz/

[7] KOROTYA Eugeniya; *5 Best JavaScript Frameworks in 2017*; [online]; 2017-01-19, [cit. 2018-03-17]; Dostupné z: https://hackernoon.com/5-best-javascript-frameworks-in-2017-7a63b3870282

[8] jecas.cz; *Nejpoužívanější prohlížeče*; [online]; 2016-03-23, [cit. 2018-03-17]; Dostupné z: http://jecas.cz/statistiky-prohlizecu#nejpouzivanejsi

[9] POKHAREL Yubraj; *Advantages/Disadvantages of Angular JS*; [online]; 2015-01-13, [cit. 2018-03-17]; Dostupné z: http://how-to-angular.blogspot.cz/2015/01/advantagesdisadvantages-of-angular-js.html

Příloha A

Instalační příručka

Tato sekce popisuje softwarové předpoklady a kroky nutné ke zprovoznění aplikace Útrata na vlastním serveru. Po provedení všech kroků bude aplikace dostupná na URL *http://<vas-server>/utrata*.

A.1 Předpoklady

Pro běh aplikace je vyžadováno:

- Apache webserver,
- MySQL 5.0 či novější,
- PHP 5.6,
- webový prohlížeč Google Chrome či Firefox v nejnovější verzi.

A.2 Instalace

1. Z přiložené paměťové karty si zkopírujte celý obsah složky `release/utrata/` do webového adresáře Vašeho serveru.
2. Pokud jste složku neumístili do kořenového adresáře Vašeho serveru, je nutné uvést absolutní cestu z kořenového adresáře Vašeho serveru k aplikaci na dvě místa:
   - v souboru `main.bundle.js`, vyhledejte řádek s kódem `Params.JS_URL_BASE = '/';` a uveďte zde za lomítko cestu k aplikaci,
   - v souboru `index.html` na řádku s kódem `<base href="/">` uveďte za lomítko cestou k aplikaci.

3. Vyrobte novou databázi pro aplikaci Útrata v kódování UTF-8. Výchozí jméno je `utrata`.

4. Spusťte v databázi skript `structure.sql`, který se nachází v přiložené paměťové kartě v adresáři `release`. Ten vyrobí jednotlivé tabulky.

5. Nyní v databázi spusťte skript `min.sql`, nacházející se v témže adresáři, který do databáze nahraje základní data potřebná pro používání aplikace.

6. Aktualizujte v konfiguračním souboru `api/.env` připojení k databázi, tj. host, uživatel a heslo.

7. Propojte aplikaci na Vašem serveru s Facebookem.

   - Na adrese *https://developers.facebook.com/* si vytvořte Aplikaci (*MyApps* → *AddNewApp*).
   - V levém menu vyberte *FacebookLogin* → *Quickstart*, vyberte, že chcete webovou aplikaci, a zde vyplňte URL Vašeho serveru.
   - Najdete Vaše AppID (v záhlaví hned pod menu).
   - V souboru `main.bundle.js` najděte řádek s kódem `appId: 'xxxx',` a nahraďte hodnotu této proměnné Vaším Facebook AppID.
   - Udělejte Vaši Facebook aplikaci veřejnou, tj. vyplňte „Privacy Policy URL“ (*Settings* → *Basic* → *PrivacyPolicyURL*) a přepněte status z vývojového stavu na veřejný stav (přepínač v záhlaví hned pod menu).

8. Otevřete si webovou aplikaci Útrata v prohlížeči tím, že zadáte adresu *http://<vas-server>/utrata*.

9. Nyní se můžete přihlásit pomocí testovacího účtu *guest* s heslem *guest*, nebo pomocí Vašeho Facebook účtu.

Příloha B

Uživatelská příručka

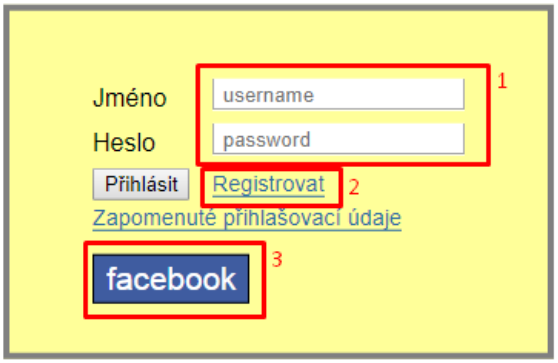
Tato kapitola popisuje způsob, jak používat nainstalovanou aplikaci Útrata.

B.1 Registrace

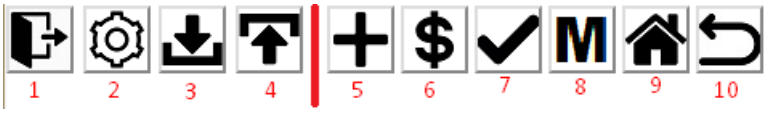
Na úvodní obrazovce se můžete registrovat dvěma způsoby:

- kliknete na nápis „Registrovat“ (obrázek B.1; 2), vyplníte formulář, který se zobrazí v popupu, a potvrdíte, nebo
- prostřednictvím Facebook účtu tak, že kliknete na tlačítko s nápisem „facebook“ (obrázek B.1; 3).

V obou případech Vás systém přesměruje na domovskou stránku Útraty, tj. přehled Vašich peněženek, který aktuálně neobsahuje zatím žádnou peněženku.



Obrázek B.1: Přihlašovací obrazovka



Obrázek B.2: Menu

B.2 Přihlášení

Na úvodní obrazovce se můžete přihlásit dvěma způsoby:

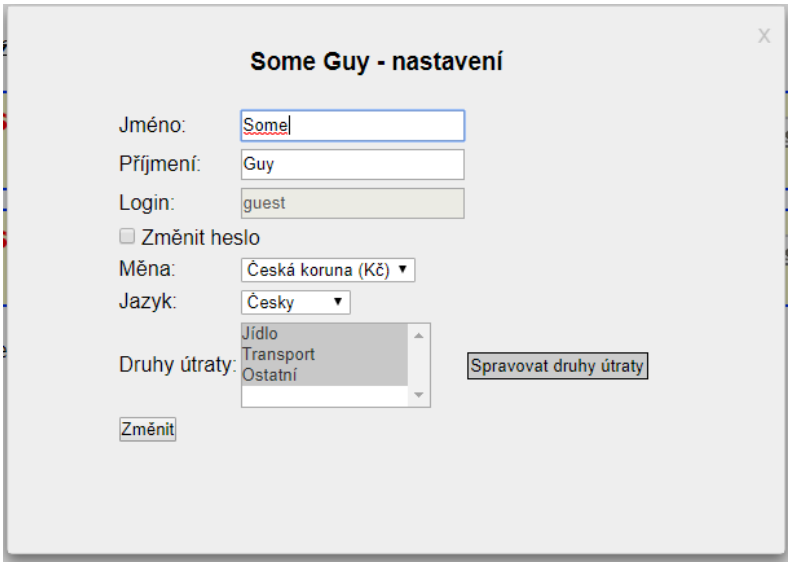
- pokud jste se dříve registroval přes Facebook účet, kliknete na tlačítko s nápisem „facebook“ (obrázek B.1; 3) a tím se přihlásíte, nebo
- vyplňte přihlašovací údaje (obrázek B.1; 1) a klikněte na tlačítko s nápisem „Přihlásit“.

V obou případech Vás systém přesměruje na domovskou stránku Útraty, tj. přehled Vašich peněženek.

B.3 Menu a navigace v aplikaci

V horní části aplikace se nachází menu sestávající ze dvou částí, které jsou na obrázku B.2 odděleny červenou svislou čarou. Levá část (tedy tlačítka s čísly 1–4) se zobrazují na každé obrazovce aplikace. Pravá část slouží k navigaci v peněžence, proto se zobrazují až když si zobrazíte nějakou peněženku. Následuje popis jednotlivých tlačítek, jak jsou očíslovány na obrázku B.2:

1. Tlačítko sloužící k odhlášení uživatele.
2. Nastavení uživatele (blíže popsáno níže).
3. Tlačítko pro stažení zálohy uživatelových dat ve formáte CSV.
4. Tlačítko pro nahrání dříve stažené zálohy dat.
5. Tlačítko pro přidání položky do peněženky (Příjmu či výdaje. Záleží, v jaké kategorii se aktuálně nacházíte).
6. Zobrazení příjmů peněženky.
7. Zobrazení archivu peněženky.
8. Zobrazení měsíčního přehledu (blíže popsáno níže).
9. Tlačítko, které Vás přesměruje na domovskou stránku s přehledem peněženek.
10. Tlačítko „Zpět“.



Obrázek B.3: Nastavení uživatele

Aplikace Útrata umožňuje navigaci také pomocí následujících klávesových zkratek:

Ctrl+B Stejné jako tlačítko „Zpět“ (Back).

Ctrl+H Přesměruje na domovskou stránku (Home).

Ctrl+I Zobrazí příjmy peněženky (Incomes).

Ctrl+M Zobrazí měsíční přehled peněženky (Monthly preview).

Ctrl+O Zobrazí archiv peněženky (Old items).

Ctrl+, Zobrazí popup pro přidání nové položky.

Ctrl+L Odhlásí uživatele (Logout).

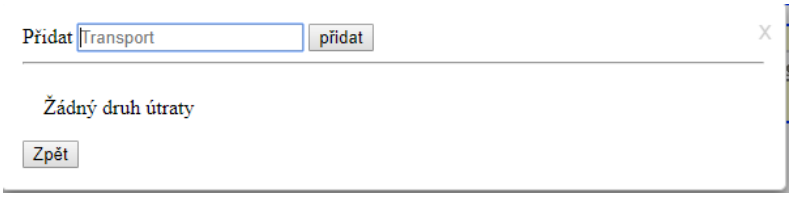
Ctrl+S Zobrazí nastavení uživatele (Settings).

Ctrl+U Vyscrolluje stránku nahoru (Up).

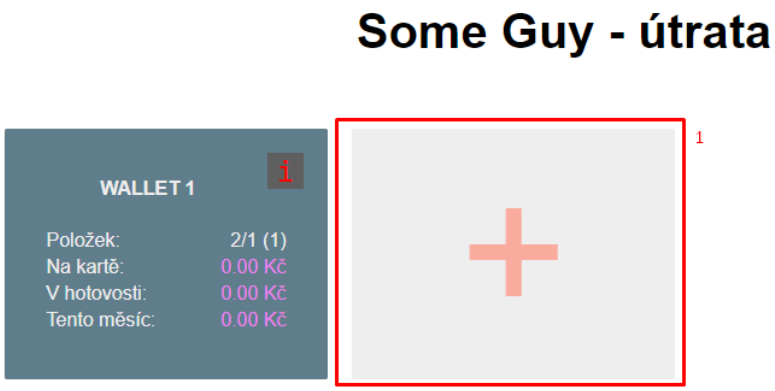
Esc Zavře popup.

B.4 Nastavení uživatele

Po kliknutí v menu na ikonu ozubeného kola (obrázek B.2; 2) nebo stisknutí klávesové zkratky `Ctrl+S` se zobrazí popup (obrázek B.3), kde si můžete změnit Vaše jméno, příjmení, heslo, měnu a jazyk. Také si zde můžete vybrat,



Obrázek B.4: Správa sekcí útraty



Obrázek B.5: Vytvořit peněženku

jaké sekce útraty chcete používat. Tento výběr uskutečníte tak, že stisknete klávesu `Ctrl` a zakliknete si sekce útraty.

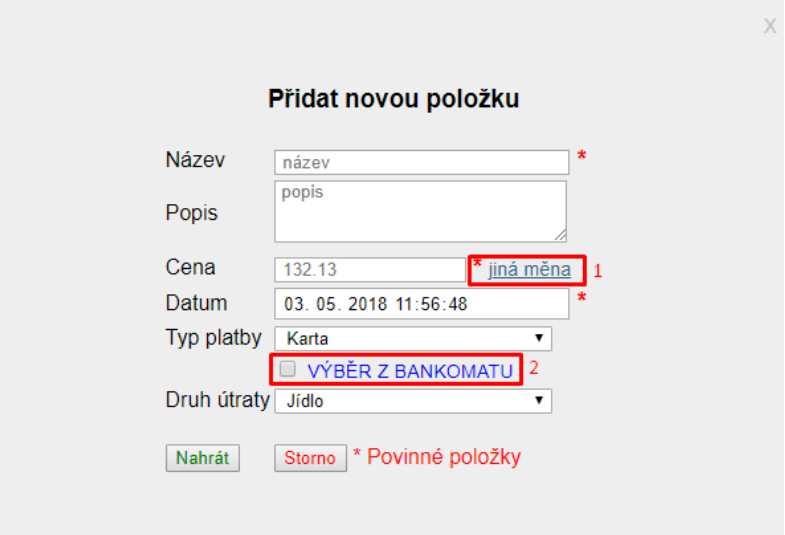
Pokud byste si chtěli vytvořit nějakou Vaši novou sekci útraty, klikněte na tlačítko „Spravovat sekce útraty“. Nyní se Vám v popupu zobrazují Vámi vytvořené sekce útraty (obrázek B.4). Nad tímto seznamem vyplníte název sekce útraty a kliknete na tlačítko „přidat“. Tím se sekce útraty vytvoří a zároveň se nastaví jako Vámi používaná.

B.5 Tvorba peněženky

Jestliže se nenacházíte na domovské stránce aplikace Útrata, klikněte v horním menu na ikonu domečku. Nyní klikněte na tlačítko se znakem „+“ (obrázek B.5; 1), čímž se Vám otevře popup pro vytvoření nové peněženky. Vyplňte název nové peněženky. Potvrdíte kliknutím na tlačítko s nápisem „Vytvořit“ nebo stisknutím klávesy enter.

B.6 Přidání položky

Pro přidání položky (ať už příjmu či výdaje – podle toho, v jaké sekci se nacházíte) slouží tlačítko s ikonou plus („+“), které se nachází v menu, nebo



Obrázek B.6: Vytvořit položku

Obrázek B.7: Položka v seznamu

klávesová zkratka `Ctrl+,`. Zobrazí popup, kam vyplňte potřebné údaje a potvrdíte kliknutím na tlačítko „Vytvořit“.

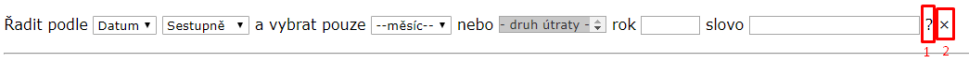
V případě, že chcete přidat položku odpovídající výběru z bankomatu, musíte ji přidat jako výdaj a ve formuláři zatrhnout příznak „VÝBĚR Z BANKOMATU“ (obrázek B.6; 2).

Pokud byste chtěli přidat položku v jiné měně, klikněte na modře zvýrazněný text „jiná měna“ (obrázek B.6; 1). Ve formuláři se zobrazí další řádek, kde vyberete měnu. Automaticky se vyplní kurz, který pak můžete ještě upravit.

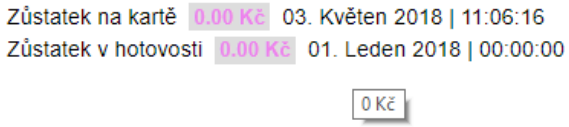
B.7 Úprava a smazání položky

Každá přidaný výdaj, nenachází-li se v archivu, má v pravé části tři tlačítka (obrázek B.7; 1, 2 a 3).

Kliknutím na tlačítko 1 se zobrazí stejný popup jako při tvorbě výdaje s tím rozdílem, že jsou zde vyplněné hodnoty tohoto výdaje a navíc je zde výběrovník pro přesunutí výdaje do jiné peněženky. Kliknutím na tlačítko 2 přesunete výdaj do archivu a kliknutím na tlačítko 3 výdaj smažete.



Obrázek B.8: Formulář pro filtraci položek



Obrázek B.9: Stav financí

Položkám v příjmech chybí tlačítko 2, protože neexistuje archiv příjmů, a položky archivu mají pouze tlačítko 3 pro smazání.

B.8 Filtrace

Pro filtraci a řazení příjmů i výdajů slouží formulář nad seznamem položek (obrázek B.8). Pro filtraci můžete vybrat měsíc, sekce útraty nebo vyplnit vyhledávací výraz. Vyhledávací výraz se řídí pravidly, které jsou dostupné po najetí kurzoru na otazník (obrázek B.8; 1).

Pro seřazení vyberte kritérium, dle kterého se má řadit (první dva výběrníky). Zrušit filtr a nastavit řazení na výchozí hodnoty lze kliknutím na křížek (obrázek B.8; 2).

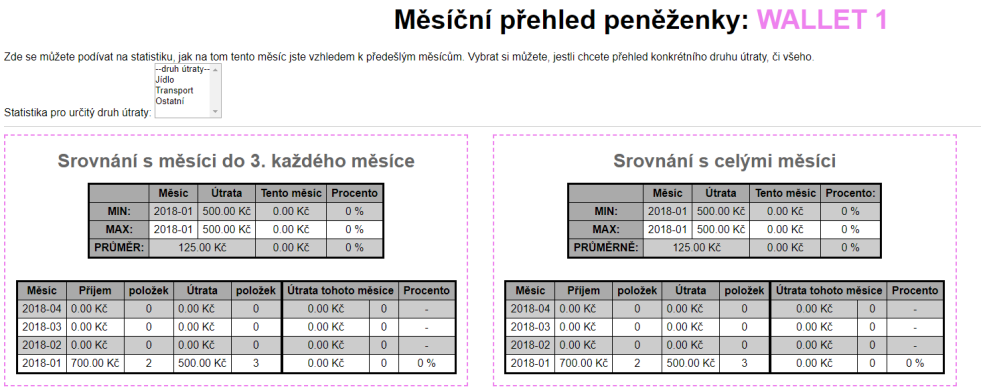
Pro filtrování ani řazení není nutné výběr potvrzovat, seznam položek se aktualizuje automaticky po změně ve filtračním formuláři.

B.9 Stav financí

V pravém horním rohu obrazovky detailu peněženky se nachází Vás stav financí (obrázek B.9), který sestává ze dvou řádků: řádek pro stav na kartě a řádek pro stav v hotovosti. V šedém obdélníku se zobrazuje Váš finanční stav a dále je uvedeno datum, kdy jste si stav naposledy zkontroloval. Po najetí kurzoru na datum se zobrazí, jaký byl finanční stav v příslušném datu. Toto datum lze aktualizovat na dnešní (společně s aktuálním finančním stavem) kliknutím na aktuální stav financí v šedém obdélníku.

B.10 Měsíční přehled

Po kliknutí na ikonu „M“ v menu se zobrazí měsíční přehled (obrázek B.10), který je tvořen dvěma částmi ohraničenými růžovou přerušovanou čarou. První část srovnává výdaje aktuálního měsíce s měsíci předešlými, ale to jen v od-



Obrázek B.10: Měsíční přehled

```
Some;Guy;guest;0;0;some@mail.com;;2018-05-03 11:04:52;CZK;1
3
1;jidlo;Jídlo;1;CZK;
2;transport;Transport;1;CZK;
3;ostatni;Ostatní;1;CZK;
1
1;wallet 1
5
1;Expense 1;Description;300;1;2018-01-01 00:00:00;2018-01-01 00:00:00;karta;1;0;0;0;1;CZK;1
2;Expense 2;Description;100;1;2018-01-01 00:00:00;2018-01-01 00:00:00;hotovost;1;0;0;0;1;CZK;1
3;Expense 3;Description;100;1;2018-01-01 00:00:00;2018-01-01 00:00:00;hotovost;0;0;0;0;1;CZK;1
4;Income 1;;500;1;2018-01-01 00:00:00;2018-01-01 00:00:00;karta;1;1;0;0;;CZK;1
5;ATM pick 1;;200;1;2018-01-01 00:00:00;2018-01-01 00:00:00;hotovost;1;1;1;0;;CZK;1
2
2;1;karta;2018-05-03 11:06:16;0
1;1;hotovost;2018-01-01 00:00:00;0
```

Obrázek B.11: Ukázka souboru zálohy dat

povídajícím období, tj. do data aktuálního dne v měsíci (když bude například 13. září, tak se zobrazuje porovnání útraty položek zaznamenaných v daných měsících pouze po 13. den). V druhé části je pak porovnání s celými měsíci.

B.11 Záloha dat

Po kliknutí na ikonu šipky dolů v menu (obrázek B.2; 3) se Vám stáhne soubor ve formátu CSV s Vašimi daty, který může vypadat jako na obrázku B.11. Tento soubor obsahuje jednotlivě po řádcích:

- informace o uživateli,
- počet Vámi vybraných sekcí útraty,
- jednotlivé sekce útraty,
- počet Vašich peněženek,

- jednotlivé peněženky,
- počet Vašich položek,
- jednotlivé položky,
- počet stavů financí (pro každou peněženku 2),
- jednotlivé stavy financí.

Pokud byste chtěli upravit soubor pro hromadný import dat, můžete přidat řádky s položkami nebo peněženkami. V takovém případě ale musíte **aktualizovat řádek s počtem položek resp. peněženek**. Pokud navíc přidáváte peněženky, pamatujte, že každá peněženka musí mít **dva stavy financí** (jeden pro kartu, druhý pro hotovost).

Struktura řádku s peněženkou je ve tvaru:

- `ID_peněženky`,
- `název_peněženky`.

Struktura řádku se stavem financí je ve tvaru:

- `ID_stavu_financí`,
- `ID_peněženky`,
- `typ_stavu_financí` (*hotovost* nebo *karta*),
- `datum_zaznamenání_stavu`,
- `hodnota_stavu`.

Konečně řádky s položkami jsou ve tvaru:

- `ID_položky`,
- `název`,
- `popis`,
- `cena`,
- `kurz`,
- `datum_položky`,
- `datum_vytvoření_položky`,
- `typ_položky` (*hotovost* nebo *karta*),
- 0 – aktivní, 1 – archivovaná,

- `jde_li_o_příjem` (0 nebo 1),
- `jde_li_o_výběr` (0 nebo 1),
- `0`,
- `ID_sekce_útraty` (v případě příjmu a výběru ponechte prázdné),
- `kód_měny`,
- `ID_peněženky`.

Pokud si budete chtít obnovit data ze zálohy nebo provést hromadný import dat, klikněte a ikonu šipky nahoru (obrázek B.2; 4). Zobrazí se popup, vyberte soubor s daty a klikněte na tlačítko „Nahrát“. Kdyby byla v souboru data ve špatném formátu, zobrazí se červená chybová hláška s popisem chyby. V opačném případě se zobrazí zelená hláška informující o úspěchu.

Příloha C

Seznam použitých zkratek

API Application Programming Interface

CI Continuous Integration

CLI Command-line Interface

CRUD Create Read Update Delete

CSS Cascading Style Sheets

CSV Comma-separated values

Dao Data access object

FTP File Transport Protocol

HTML Hypertext Markup Language

HTTP Hypertext Transfer Protocol

IDE Integrated Development Environment

JSON JavaScript Object Notation

MVVM Model-View-View Model

ORM Object-relational mapping

REST Representational State Transfer

SDK Software development kit

SPA Single-page application

SSH Secure shell

UI User Interface

UML Unified Modeling Language

URI Uniform Resource Identifier

URL Uniform Resource Locator

YAML Ain't Markup Language

Příloha D

Obsah přiloženého média

- `readme.txt` ... stručný popis obsahu CD
- `devGuide.pdf` ... příručka pro vývojáře
- `release` ... adresář se spustitelnou formou implementace
- `doc` ... dokumentace
  - `backend` ... HTML dokumentace serveru
  - `frontend` ... HTML dokumentace klienta
- `src`
  - `impl` ... zdrojové kódy implementace
  - `thesis` ... zdrojová forma práce ve formátu LaTeX
- `text` ... text práce
  - `thesis.pdf` ... text práce ve formátu PDF