CZECH TECHNICAL UNIVERSITY IN PRAGUE

Faculty of Electrical Engineering

Aplikace pro správu osobních financí (Backend)

Author: Mikhail Khomutov (khomumik@fel.cvut.cz)

Study program: Otevřená informatika

Branch of study / Specialization: (BPOI318) Software

Thesis supervisor: doc. Ing. Karel Richta, CSc. (katedra počítačů)

May 2023

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Khomutov** Jméno: **Mikhail** Osobní číslo: **495564**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Otevřená informatika**

Specializace: **Software**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Aplikace pro správu osobních financí

Název bakalářské práce anglicky:

The Application for Personal Financal Management

Pokyny pro vypracování:

Action reversibility deals with the problem of finding a sequence of (other) actions that undoes its effects. Several recent works investigate reversibility of deterministic actions. Action reversibility might be useful in fully observable non-deterministic (FOND) planning where one might want to undo undesirable effects of a non-deterministic action. However, the concept of action reversibility from deterministic planning cannot be stragitforwardly adopted to FOND planning as "reverting" actions might also be non-deterministic.

Requirements:

• investigate the concept of non-deterministic action reversibility and identify interesting classes of non-deterministic action reversibility (e.g., strong reversibility - guaranteed to undo action effects, weak reversibility - has a chance to undo action effects, uniform reversibility - the same policy undoes action effects in a set of states)
• theoretically investigate relationships of the proposed classes of non-deterministic action reversibility
• design and develop methods for solving some classes of non-deterministic action reversibility
• evaluate the methods on at least 6 domain benchmarks of FOND planning

Seznam doporučené literatury:

[1] Pressmann R. S.: Software Enegineering,
[2] Vávrů J., Ujbányai M.: Programujeme pro Android, Grada
[3] https://uvero.cz/aplikace-pro-spravu-financi-ktere-stoji-za-vyzkouseni

Jméno a pracoviště vedoucí(ho) bakalářské práce:

doc. Ing. Karel Richta, CSc. **katedra softwarového inženýrství FIT**

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

Datum zadání bakalářské práce: **10.02.2023** Termín odevzdání bakalářské práce: **26.05.2023**

Platnost zadání bakalářské práce: **22.09.2024**

doc. Ing. Karel Richta, CSc.	podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D.

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.

Poděkování

Rád bych poděkoval vedoucí mé práce doc. Ing. Karel Richta, CSc. za poskytovaní možnosti pracovat pod jejím vedením.

Prohlášení

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

V …… datum …… podpis autora

Abstrakt

Hlavním cílem práce je návrh backendové podpory webové aplikace, která umožní uživatelům vest své finance. Prvním cílem dané prací byla analýza podobných aplikaci v internetovém prostoru a určení funkcionality aplikace. Na základě provedené analýzy byli specifikovaný základní případy užití a prvotní návrh. Dále byly definované implementační technologie.

Výstupem této práce je návrh aplikace pro správu osobních financi.

Klíčova slova: správa osobních financi, peníze, rozpočet, backend, webová aplikace

Abstract

The main goal of the work is to design backend for web application, which offers tool to manage finances for users. First aim of this work was analyze of similar applications from the Internet space and determination application’s functionality. Basic use-cases and the initial design were specified based on performed analyze. Implementation technologies were defined also.

The output of the work is to design up backend application for managing finances and gained experience.

Keywords: personal finance management, money, budget, backend, web application

1 OBSAH

2 ÚVOD

Existuje spousta možností, jak spravovat osobní rozpočet: od běžného notebooku až po moderní software. Jedním ze standardních nástrojů pro správu rodinného rozpočtu je však excelovská tabulka nebo ruční vedení rozpočtu v sešitu. Takové metody zaberou spoustu času a pro člověka jsou někdy docela obtížné.

Když se dotkneme tématu technologického pokroku, je třeba poznamenat, že technologie zaujala důležité místo v životě lidstva. Nyní se bez telefonů, tabletů, počítačů, notebooků atd. v životě neobejdete. Mobilní aplikaci pro vedení osobního rozpočtu je možné vždy mít po ruce a kdykoli ji vyplnit a tím zjednodušit i proces výpočtu a evidence výdajů a příjmů. Stane se pohodlnější, jednodušší a rychlejší kontrola příjmů a výdajů, software aplikace vám pomůže vizuálně vidět přehledy o rozpočtu, sledovat aktuální stav účtů.

Využívání takové webové aplikace zvýší finanční gramotnost obyvatel a zefektivní rozdělování prostředků rozpočtu.

3 CÍL PRÁCE

Cílem práce je seznámeni s tématem správy financí v rámci osobního rozpočtu. Pro kvalitní vypracovaní práce zapotřebí:

- analýza podobných aplikací
- analýza technologii, které se budou používat v aplikace
- analýza vedení osobního rozpočtu
- analýza struktury aplikace z pohledu implementace

Druhým cílem práce je získaní drahocenné zkušenosti vlastního vývoje aplikace od začátku do Implementační časti. Ten projekt pomůže proniknout hlouběji do oblasti analytiky aplikace, která součásti komplexního vývoje.

4 VÝZKUM

Samozřejmě k správě osobních financí můžete mít i osobní sešit, kam si ručně zapisujete každou finanční činnost, sestavujete finanční plán na měsíc a podobně. Mnohem jednodušší je však mít po ruce webovou aplikaci finančního účetnictví.

Tato kapitola představuje analýzu dobře známých řešení problému vedení rozpočtu. Mimo jiné bude nutné zajistit požadavky z pohledu možného uživatele pro přesnější sestavení aplikace. Bude provedena analytická analýza, analýza technologii, které použijeme k tvorbě aplikace.

4.1 ANALÝZA ZÁKLADNÍHO VEDENÍ ROZPOČTU

Pokud se příjmy rovnají výdajům, pak jde o vyrovnaný rozpočet. Pokud předpokládané výdaje převyšují příjmy, pak je tento rozpočet deficitní. Rozpočet, ve kterém příjmy převyšují výdaje, bude mít přebytek. Pokud příjmy převyšují výdaje, je nutné z plánů vyloučit zbytečné nákupy, aby byl rozpočet vyrovnaný.

Základní funkce aplikace:

- Zadávání, ukládání a prohlížení informací o příjmech a výdajích;
- Výběr aktuálního zůstatku;
- Vyhledávání transakcí za den a období;
- Analýza informací s grafickým znázorněním dat:
    - Měsíční dynamika výdajů (ne více než měsíc);
    - Náklady podle kategorií (ne více než rok);
- Poměr výdajů a příjmů podle měsíců (ne více než rok);
- Smazání dat za konkrétní den, období nebo úplné vymazání databáze.

Popis programu a pokyny pro uživatele

Osobní rozpočet je váš osobní plán příjmů a výdajů na určité období měsíce, čtvrtletí, roku. Pomocí tohoto opravdu efektivního programu můžete bez námahy kontrolovat osobní příjmy a výdaje, dále je možné kategorizovat výdaje a příjmy do skupin, plánovat rozpočet nebo sledovat stav účtů, sestavovat grafy a tabulky.

4.2 ANALÝZA PODOBNÝCH APLIKACÍ

Pro porovnání byly přijaty takové populární aplikace jako Monefy, Wallet a Spendee. V některých případech mluvíme o aplikacích pro Android, ale to není tak důležité, protože v této části mluvíme jen o analýze. V důsledku této analýzy je nutné získat srovnání funkčnosti pro obecný pohled na konkurenční aplikace.

4.2.1 Monefy[1]

Monefy má atraktivní a intuitivní rozhraní. Všechny výdaje jsou zobrazeny v grafu na hlavní obrazovce – můžete je sledovat za den, týden nebo měsíc v různých kategoriích. Aplikace podporuje více měn.

Obrázek 1 Monefy

Výhody:

- Účtování nákladů a výnosů
- Exportování dat do Excel[2]
- Možnost účtování v požadované měně
- Schopnost přidávat a odebírat kategorie výdajů a příjmů
- Můžete si vybrat období vykazování a sestavit vizuální statistiku výdajů s grafy
- Podporuje více účtů současně

Nevýhody:

- Pro výkaz můžete vybrat pouze časové období určené programem nebo jedno datum
- Problémy se synchronizací s bankovními aplikacemi a dalšími zařízením

4.2.2 Wallet[3]

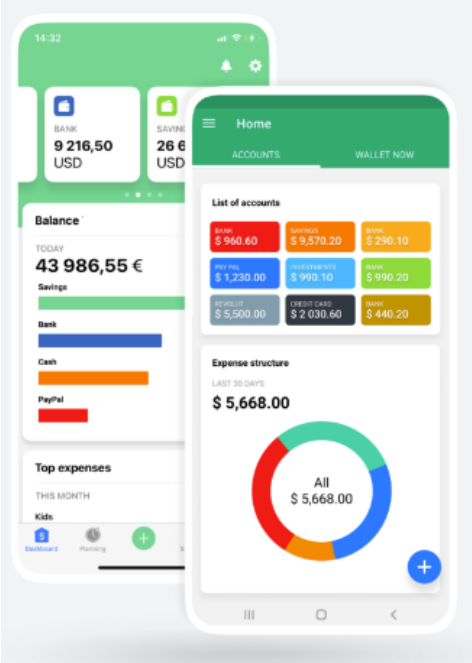
Celá aplikace je velmi přehledná a přidávání výdajů je velmi intuitivní. Stejně jako ostatní aplikace pro správu financí, i tato nabízí funkce v podobě jednotlivých rozpočtů, třídění podle štítků a přehledné grafy.

Výhody:

- Přehledná a dobře vypadající aplikace
- Uspořádá vaše finance prostřednictvím personalizovaných rozpočtů, cílů a kategorií
- Užitečné přehledy výdajů, včetně dost praktických grafů

Nevýhody:

- Bezplatná verze umožňuje pouze „ruční“ sestavování rozpočtu, což je extrémně
- Časově náročné



Obrázek 2 Wallet

[1] https://monefy.me/
[2] https://www.microsoft.com/ru-ru/microsoft-365/excel
[3] https://budgetbakers.com/

4.2.3 Spendee[4]

Spendee je aplikace pro osobní finance a rozpočet, která uživatelům umožňuje sledovat své finance pomocí mobilního telefonu nebo počítače a spravovat své denní výdaje online. Aplikace automaticky synchronizuje vaše výdaje a umožňuje uživatelům sledovat, kam jdou jejich peníze.

Obrázek 3 Spendee

Výhody:

- Pohodlná správa všech vašich finančních dat, včetně kryptoměny, na jednom zařízení
- Uživatelsky přívětivé rozhraní s barevnými grafy a diagramy, které vizuálně znázorňují a rozpisují vaše utrácení
- Přizpůsobitelné funkce – nastavení upozornění, která vás upozorní, když překročíte rozpočet, nebo měsíční připomenutí k zaplacení účtů
- Možnost připojení více běžných účtů, jako je rodinný tarif, ale pro vaše finance, abyste věděli, kdo za co zaplatil a kolik bylo celkem utraceno

Nevýhody:

- Nelze synchronizovat se všemi bankovními účty: Záleží, zda to finanční instituce Spendee povolí. Pokud je vynechán ten váš, budete muset ručně zadat podrobnosti o každé transakci, kterou jste provedli. To je buď pro nebo proti, v závislosti na tom, jak chcete sledovat své finance.

[4] https://www.spendee.com/

4.2.4 Závěr

Funkcionalita	Monefy	Wallet	Spendee
Nemá reklamu	-	+	+
Vestavěné grafy	+	+	+
Vystavení limitů	-	+	+
Synchronizace s banky	-	+(Premium)	+(Premium)
Rozeznávání účtenek	-	-	-
Vytvořeni kategorií	+(Premium)	+	+
Notifikace	-	+	+
Sdílený přístup (zakládaní účtu)	-	-	+
Export do Excel	+	+	+

Tabulka 1 Porovnávání elektronických peněženek

Po analýze se potvrdila vysoká konkurence v oblasti aplikací této práce. Mnoho společností se snaží propagovat svůj produkt pomocí více funkcí ve svých aplikacích, což je dobré jak pro trh, tak pro spotřebitele. Pro běžného člověka bude většina aplikací stačit bez prémiového účtu (pokud existuje). Tento typ vylepšení ovlivňuje drobné funkce, které jsou spíše specifické pro běžného kupujícího. Díky tomu můžeme konstatovat, že funkce standardní aplikace bez prémiového předplatného (nebo jednorázové platby) jsou nejpoužívanější a nejžádanější.

Díky rozboru bych rád zdůraznil náhradní funkcionalitu, kterou bych v této práci rád viděl:

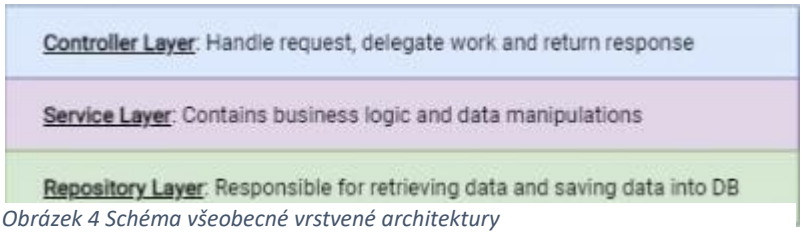
- Vestavěné grafy
- Vytvořeni kategorií
- Notifikace
- Sdílený přístup (zakládaní účtu)
- Export do Excel nebo jiných typů soubor

4.3 ANALÝZA TECHNOLOGIÍ

V této kapitole si povíme o technologiích, se kterými bude aplikace postavena. Jaký programovací jazyk by měl být použit jako základ aplikace? Které programovací prostředí poskytuje nejvíce funkcí nebo je pro programátora přívětivější? Jaké frameworky budou použity? Jakou architekturu zvolit? To jsou otázky, které si zde budeme klást.

4.3.1 Architektura

Vrstvená architektura



Obrázek 4 Schéma všeobecné vrstvené architektury

Vrstvená architektura odděluje funkce frameworku, API a databází do samostatných jednotek, což umožňuje vstupům od uživatelů interagovat s každou jednotkou *nezávisle*. Tento typ architektury často umožňuje programátorům měnit informace v jedné jednotce, aniž by to ovlivnilo ostatní. Programátor může například změnit informace v databázi, aniž by změnil způsob interakce rozhraní API s touto databází.

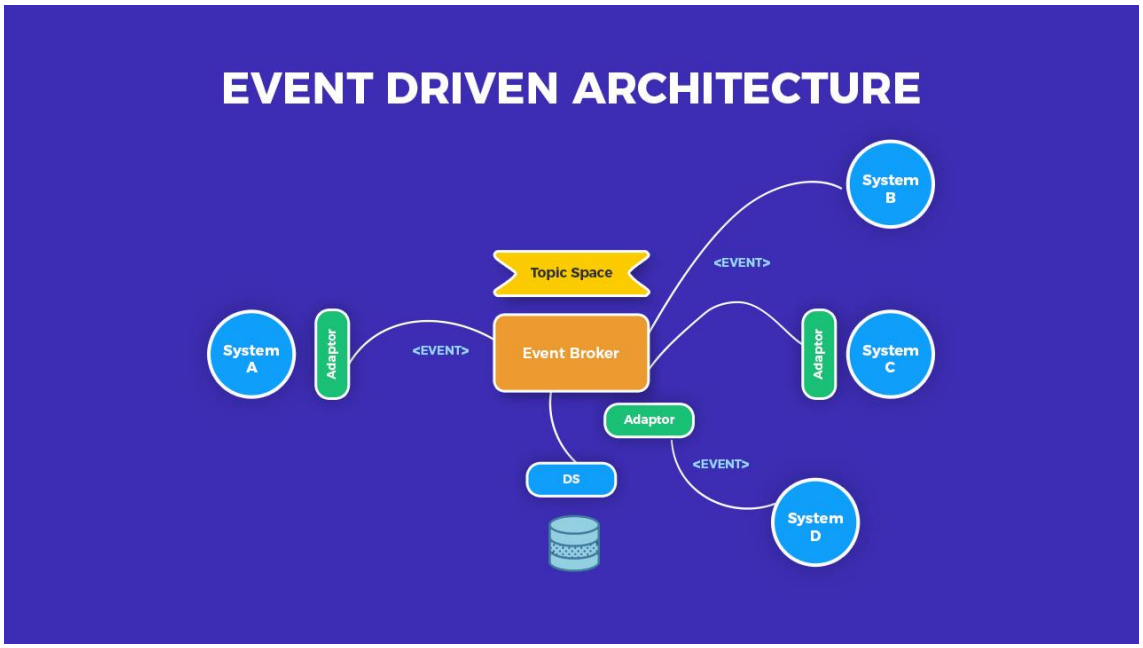
Controller Layer: žádná logika, žádná přímá manipulace s daty. Hlavním cílem kontroléru je zpracovat dotaz, přeposlat dále na servisní vrstvu a vrátit výsledek.

Service Layer: business logika, může pracovat a manipulovat s daty.

Repository Service: CRUD[5], jinak práce s databází

[5] CRUD – Create, Read, Update, Delete. Shrnuje čtyři základní operace nad záznamem v trvalém úložišti (např. v SQL databázi)

Událostmi řízená architektura

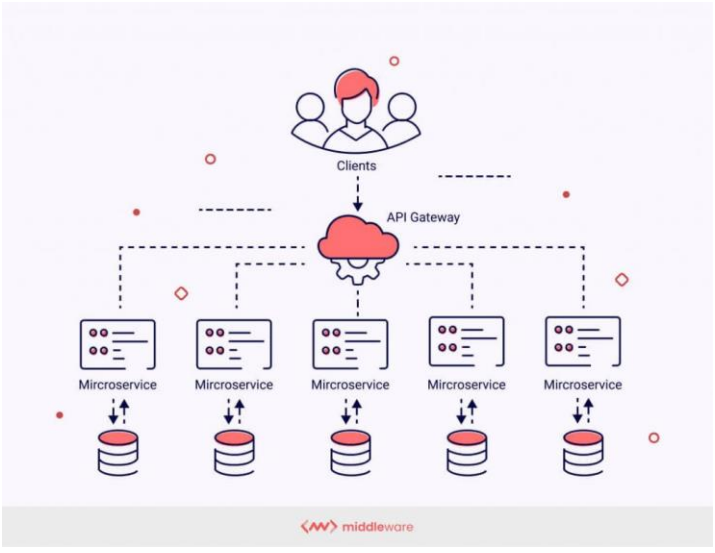


Obrázek 5 Ilustrace událostmi řízené architektury

Událostmi řízená architektura vytváří jednotku, která přijímá všechny uživatelské vstupy a potom je přesměruje na příslušné místo. Když například uživatel dokončí akci na webu, která vyžaduje použití databáze, centrální jednotka pomocí logiky vyvolá data z databáze a poté provede aktualizace na základě akcí uživatele. To umožňuje, aby celý program fungoval jako vzájemně závislá skupina.

Architektura mikroservisů

Mikroservisní architektura zahrnuje rozdělení velkých časti na malé. Jinými slovy, nyní budeme místo několika těžkopádných služeb využívat velké množství malých služeb. Tím sdílíte a zjednodušujete úkol. Aplikace s touto architekturou je snadno škálovatelná, testovatelná a odolná proti chybám.



Obrázek 6 Schéma mikroservisní architektury

4.3.2 Backend

Před deseti lety 9 z 10 backendových aplikací běželo na takzvaném LAMP stacku (Linux, Apache, MySQL and PHP). Ale postupem času se technologie vyvíjela a množila se s neuvěřitelnou rychlostí. Stále vznikalo více nových frameworků pro různé programovací jazyky. někteří samozřejmě konkurence nevydrželi a zmizeli. Nyní však máme obrovský výběr technologií, které zjednodušují vývoj a nabízejí nové zajímavé funkce.

4.3.2.1 Spring Boot

Java Spring Framework (Spring Framework, Duben 2014) je populární open source framework na podnikové úrovni pro vytváření samostatných aplikací na produkční úrovni, které běží na Java Virtual Machine (JVM).

Obrázek 7 Spring Boot emblém

Spring Boot se používá k vytváření samostatných a produkčních aplikací založených na Spring. Spring Boot, původně vydaný v roce 2002, využívá platformu Spring a knihovny třetích stran, aby umožnil vývojářům začít s minimálními problémy. Několik výhod, které nabízí framework Spring Boot:

- Pomáhá vyhnout se složité konfiguraci XML v Springu
- Framework připravený na mikroservisní architekturu
- Pomáhá spravovat endponity[6] REST
- Snadná správa závislostí

4.3.2.2 ASP.NET Core

Obrázek 8 ASP.NET Core emblem

ASP.NET Core, poprvé zveřejněno v roce 2016, je open-source verze ASP.NET. Jedná se o přepracování dřívějších verzí ASP.NET pouze pro Windows. ASP.NET Core, oblíbený framework pro vývoj webových aplikací pro vytváření webových aplikací na platformě .NET, je mnohem rychlejší než většina aktuálně dostupných frameworků. Klíčové výhody ASP.NET Core jsou:

- Aplikace ASP.NET lze spustit v systémech Windows, Linux, macOS a Docker
- Dobrá realizace asynchronního modelu vyplnění kodu – async/await[7]
- ASP.NET MVC – framework pro web vývoj na .NET. Poskytuje velké množství funkcí, např. autorizace a filtry. Kromě toho, framework je navržen tak, aby s toho bulo možné snadno udělat svojí realizací nějakých modulu.

4.3.2.3 Django

Django (Leden 2013) je Python webový framework na vysoké úrovni, který podporuje rychlý vývoj a čistý, pragmatický design. Postará se o většinu problémů s vývojem webu, takže se můžete soustředit na psaní své aplikace, aniž byste museli znovu objevovat kolo. Je to zdarma a open source. Mezi nejvýznamnější výhodami jsou:

Obrázek 9 Django emblem

- Rychlý vývoj
- Na velké projekt

[6] Endpoint je koncový bod komunikačního kanálu. Když API komunikuje s jiným systémem, dotykový bod mezi nimi se nazývá endpoint. V dané práci API endpoint je URL.

[7] Umožňuje vývojářům psát výkonný neblokující kód, což má pozitivní vliv na propustnost serveru.

- Univerzálnost. Může to být redakční systém nebo celá sociální síť. Můžete Django požádat, aby provedl nějaké strojové učení nebo komplikovanou analýzu dat.

4.3.2.4 Závěr

Pro analýzu jsem vybral frameworky různých jazyku pro rozsáhlejší výzkum a rozšíření svých znalosti. Každý z frameworků se ukázal být svým způsobem zajímavým. Pro tuto práci byl vybrán framework Spring Boot kvůli jeho široké distribuci a snadnému použití. Nejnovější je ASP.NET core, co může říct o jeho potenciální době údržby ze strany vývojářů. Django je dobrým příkladem technologie pro rychlý vývoj monolitických aplikací, ale nesplňuje potřeby dané prací, ale je dobrým protikladem.

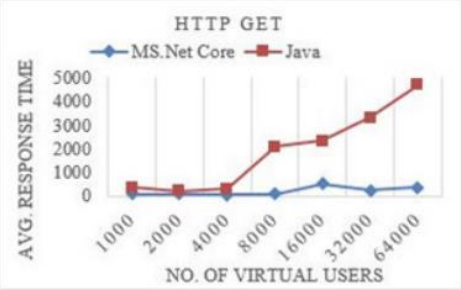
Po srovnání z pohledu uživatele je čas se ponořit do otázky, která je pro všechny znepokojivá – Co je rychlejší?

Testovací konfigurace:

- ***ACP.NET core (MS.NET core):*** ACP.NET core RESTful aplikace, která je implementovaná pomocí Microsoft Visual Studio 2019. Kod je napsán v C# s nalinkovaným frameworkem .NET Core 3.1. SQL databáze je spojena pomocí ADO.NET technology.
- ***Spring Boot:*** Spring Boot Java 2.1.6 RESTful aplikace je implementována pomocí Spring tool suite (pro spojení s databází)

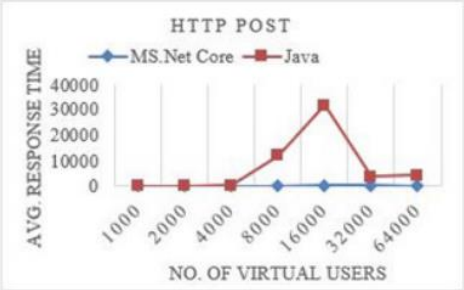
Implementováno 4 CRUD operace.

GET (Read):



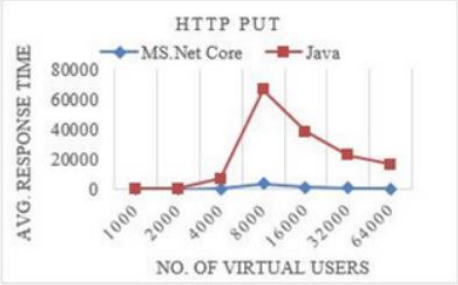
Obrázek 10 Průměrná doba odezvy GET

POST (Create):



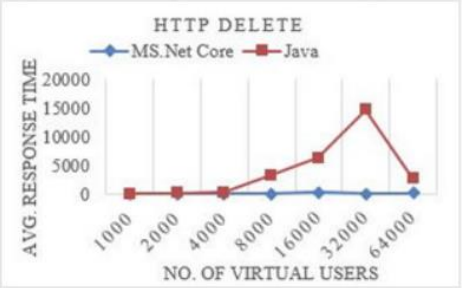
Obrázek 11 Průměrná doba odezvy POST

PUT (Update):



Obrázek 12 Průměrná doba odezvy PUT

DELETE:



Obrázek 13 Průměrná doba odezvy DELETE

Po srovnání bylo zjištěno, že rychlost Spring Boot při simulaci velkého počtu virtuálních uživatelů nevykazovala uspokojivé výsledky. V tomto srovnání zvítězila platforma ASP.NET core

Toto srovnání bylo velmi poučným. Však pro tuto práci byl vybrán Spring Boot pro svou snadnost a přehlednost v učení, připravenost na vrstvenou architekturu a rychlost nasazení aplikací.

4.3.3 Databáze

V této kapitole se budeme zabývat 2 typy databází SQL a NoSQL, porovnáme je a vyvodíme závěry, která z nich je pro tuto práci vhodnější. Každý typ má své klady i zápory. Nejprve určíme definice těchto typů databází, pak pro přehlednost uděláme tabulku.

SQL:

Data jsou uložena v řádcích a tabulkách, které jsou propojeny různými způsoby. Jeden záznam tabulky se může propojit jeden s druhým nebo s mnoha dalšími, nebo mnoho záznamů tabulky může souviset s mnoha záznamy v jiné tabulce. Tyto relační databáze, které nabízejí rychlé ukládání a obnovu dat, dokážou zpracovat velké množství dat a složité SQL dotazy.

NoSQL:

Když lidé používají termín „databáze NoSQL“, obvykle jej používají k označení jakékoli nerelační databáze. Někteří říkají, že výraz „NoSQL“ znamená „není SQL“, zatímco jiní říkají, že znamená „nejen SQL“. Ať tak či jinak, většina souhlasí s tím, že databáze NoSQL jsou databáze, které ukládají data v jiném formátu než relační tabulky.

Srovnání:

Porovnávací kritéria	SQL	NoSQL
Škálování	SQL databáze jsou *vertikálně* škálovány, to znamená, že můžete zvýšit zatíženi[8] DB[9] migrací na větší server (víc CPU, RAM nebo SSD).	NoSQL databáze jsou *horizontálně* škálovatelné, to znamená, že můžete zvýšit zatíženi DB přidáním dodatečného serveru.
Struktura	*Relační a tabulkové způsoby organizace db*: tabulky se sloupci (atributy) a řádky (záznamy)	*Klíč-hodnota db*: hodnoty odkazované pomocí klíčů (obdoba slovníků) *Dokumentové db*: ukládání strukturovaných dokumentů (typicky [B]JSON) *Sloupcové db*: něco mezi dokumenty a SQL – každý řádek může mít jiné sloupce (které mohou tvořit skupiny) *Grafové db*: relace representované grafy

[8] Zatížení databáze (DB load) měří úroveň aktivity ve vaší databázi.

[9] DB (db) - zkrátka pro databáze

Vlastnosti	ACID[10] *Atomicita*: všechny transakce buď projdou nebo spadnou najednou. *Konzistence*: db se musí řídit pravidly, která se ověřuji každý krok. *Izolace*: Současné běžící transakce se nemohou ovlivňovat jeden druhého. *Trvanlivost*: Transakce jsou konečné a ani selhání systému nemůže vrátit kompletní transakcí.	CAP[11] *Konsistence*: Každý požadavek obdrží buď nejnovější výsledek, nebo chybu. *Dostupnost*: Každý požadavek má nechybový výsledek. *Tolerance rozdělení*: Jakékoli zpoždění nebo ztráty mezi uzly nepřeruší provoz systému.
Podpora	SQL má obrovskou komunitu díky dlouhé historii.	NoSQL má menší komunitu, a proto má menší podporu.

Tabulka 2 Porovnávání SQL a NoSQL

Jak je vidět z porovnání, že každý druh databáze má svoje výhody. Vzhledem k tomu, že tématem naší práce je správa osobních financí, je logické předpokládat, že pro analýzu našeho rozpočtu budeme potřebovat různé nástroje, jako jsou grafy a diagramy. Tyto nástroje, jak mi moje zkušenost říká, vyžadují vysoce strukturovaná data. Podle toho jenom zbývá vybrat vhodnou relační databázi.

4.3.3.1 MySQL

Mezi mnoha SQL databází byla vybrána MySQL. Proč je MySQL lepší než jiné databáze?

Snadno se používá a spravuje	Snadno se používá a běží rychle. MySQL server[12] si můžete stáhnout a nainstalovat během několika minut. Kromě toho, MySQL má takové funkce jako MySQL Admin[13], MySQL Workbench[14], nástroje GUI a několik dalších nástrojů, usnadňují konfiguraci a správu.
Výjimečný výkon	Dokáže zpracovávat více dotazů a transakcí s optimální rychlostí a jedinečným ukládáním do mezipaměti.
Je podporováno velkou komunitou	MySQL je bezesporu populární open-source databázový systém s miliony aktivních instalací. V důsledku toho má obrovskou uživatelskou základnu s tisíci fórumů, kde můžete najít řešení pro jakýkoli problém s MySQL.
Je to bezpečné a spolehlivé	Šifrování dat v MySQL chrání neoprávněné prohlížení dat, zatímco podpora SSH a SSL poskytuje bezpečné připojení.

Tabulka 3 Výhody MySQL

[10] ACID – zkratka pro vlastností SQL db: Atomicity (Atomicita), Consistency (Konsistence), Isolation (Isolace) a Durability (Trvanlivost)

[11] CAP – zkratka pro vlastnosti NoSQL db: Consistency (Konsistence), Availability (Dostupnost) a Partition tolerance (Tolerance rozdělení)

[12] https://dev.mysql.com/downloads/mysql/

[13] https://dev.mysql.com/doc/refman/8.0/en/mysqladmin.html

[14] https://www.mysql.com/products/workbench/

5 ANALÝZA

V této kapitole definujeme funkční a nefunkční požadavky. Shromážděme všechny informace, které jsme dříve našli a zpracovali, do rozumnější podoby. A kromě toho budou také definovány základní případy užití.

5.1 POŽADAVKY

Funkční požadavky definuji, co produkt má dělat, jaké má vlastnosti a funkce. Nefunkční požadavky popisují obecné vlastnosti systému. Jsou také známé jako atributy kvality.

5.1.1 Funkční

FP-1: Registrace účtu aplikace.
FP-2: Přihlášeni pomocí účtu aplikace.
FP-3: Vytváření vlastních peněženek.
FP-4: Tvorba výdajů/příjmů a jejich přiřazení ke kategorii.
FP-5: Tvorba kategorií výdajů/příjmů a jejich druhů.
FP-6: Podpora více druhů měn pro peněženky.
FP-7: Přehled statistiky a grafu.
FP-8: Přehled výdajů nebo příjmů podle kategorií.
FP-9: Export uživatelských dat do Excel
FP-10: Filtrace výdajů podle velikosti hodnoty výdaje

5.1.2 Nefunkční

NP-1: Backend: Spring Boot, Java, MySQL
NP-2: Mikroservisní architektura
NP-3: REST API
NP-4: Repository šablona pro Data Access Layer pomocí JPA
NP-5: Cache
NP-6: Prostředí: Intelij Idea Ultimate Edition

5.2 PŘÍPADY UŽITÍ

V této kapitole se podíváme na případy užití aplikace. Největší výhodou diagramu případů použití je, že pomáhá vývojářům softwaru a podnikům navrhovat procesy z pohledu uživatele. Výsledkem je, že systém funguje efektivněji a slouží cílům uživatele. Případy užití je možné definovat, jak textově, tak i graficky pomocí UML[15]. Případ užití je vždy iniciován účastníkem. Účastníkem může být cokoliv/kdokoliv, co/kdo systém používá nebo ovlivňuje.

V našem případě jsou případy užití popsány textově:

UC 1 - Registrace uživatele v systému

1. Systém zobrazí přihlašovací formulář a 2 tlačítka: „Přihlásit se“, „Registrovat se“. Formulář bude potřebovat login a heslo.
2. Uživatel klikne na tlačítko „Registrovat se“.
3. Systém zobrazí formulář pro registraci a tlačítko „Zaregistrovat se“
4. Uživatel vyplní formulář (login, jméno, příjmení, heslo) a klikne na tlačítko „Zaregistrovat se“

[15] UML – Unified Modeling Language, univerzální graficky jazyk pro modelovaní systémů

5. Systém uloží data nového uživatele do dbá a zobrazí přihlašovací formulář.

UC 2 - Přihlášeni uživatele do systému

1. Systém zobrazí přihlašovací formulář. Formulář bude potřebovat login a heslo.
2. Uživatel vyplní formulář a klikne na tlačítko „Přihlásit se“
3. Systém ověří správnost formuláře a posle dotaz, jestli existuje uživatel s takovým loginem a heslem. Pokud Ano – přesměruje na hlavní stránku aplikace

UC 3 - Vytvořit peněženku

1. Systém zobrazí stránku s existujícími peněženky
2. Uživatel klikne na tlačítko „+“, co znamená přidat peněženku
3. Systém zobrazí formulář pro vytvořeni nové peněženky (název, měna, počáteční zůstatek peněženky)
4. Uživatel vyplní formulář a klikne na tlačítko „Vytvořit“
5. Systém ověří správnost formuláře. Pokud všechno bude v pořádku uloží novou peněženku do db a zobrazí stránku s existujícími peněženky

UC 4 - Vytvořit kategorie pro výdaj/příjem

1. Systém zobrazí stránku s existujícími kategoriemi
2. Uživatel klikne na tlačítko „+“, co znamená přidat kategorii
3. Systém zobrazí formulář pro vytvořeni nové kategorie (název)
4. Uživatel vyplní formulář a klikne na tlačítko „Vytvořit“
5. Systém ověří duplicitu nazvu kategorie. Pokud všechno bude v pořádku systém přidá novou kategorii do db a zobrazí stránku s existujícími kategoriemi

UC 5 - Zobrazit statistiku

1. Systém zobrazí hlavni stránku s velkým kruhovým diagramem se statistikou

UC 6 - Zobrazit přehled výdajů a přijmu

1. Systém zobrazí hlavni stránku aplikace
2. Uživatel klikne na tlačítko „Přehled výdajů a příjmů“
3. Systém zobrazí stránku se všemi výdaje a příjmy tohoto měsíce (raženo podle data)

UC 7 - Filtrovat přehled výdajů a přijmu velikosti hodnoty výdaje

1. Systém zobrazí stránku se všemi výdaje a příjmy tohoto měsíce (raženo podle data)
2. Uživatel stiskne tlačítko „Řadit podle velikosti hodnoty výdaje“
3. Systém zobrazí stránku se všemi výdaje a příjmy tohoto měsíce (raženo podle velikosti hodnoty výdaje)

UC 8 - Zobrazit přehled výdajů nebo příjmů podle kategorií

1. Systém zobrazí hlavni stránku aplikace
2. Uživatel klikne na tlačítko „Kategorie“
3. Systém zobrazí okno s seznamem kategorií
4. Uživatel vybere kategorií a klikne na tlačítko této kategorii
5. Systém zobrazí vybranou kategorii a v datech kategorií bude spočítaná hodnota podle výdajů nebo příjmů, závisí na tom jakého typu je kategorie

UC 9 - Exportovat uživatelská data

1. Systém zobrazí hlavni stránku aplikace
2. Uživatel stiskne tlačítko „Exportovat do Excel“
3. Systém vytvoří .xlsx[16] soubor a vyvolá v prohlížecí metodu staženi souboru

UC 10 - Vytvořit položku výdaje nebo přijmu

1. Systém zobrazí hlavni stránku aplikace

[16] .xlsx – výchozí formát souboru založený na XML pro Excel 2010 a Excel 2007

2. Uživatel klikne na velké tlačítko „+“ pro příjem nebo „-“ pro výdaj
3. Systém zobrazí formulář pro přidáni přijmu nebo výdaje (částka, kategorie, poznámka)
4. Uživatel vyplní formulář a klikne na tlačítko „Přidat“
5. Systém ověří správnost formuláře. Pokud všechno je v pořádku, přidá novy výdaj do db a přesměruje na hlavní stránku aplikace

5.2.1 Pokrytí požadavků případy užití

Pro přehlednost vyplnění funkčních požadavku můžeme se podívat na následující tabulku pokryti. Díky tabulce můžeme vidět, jestli případy užití pokrývají funkční požadavky.

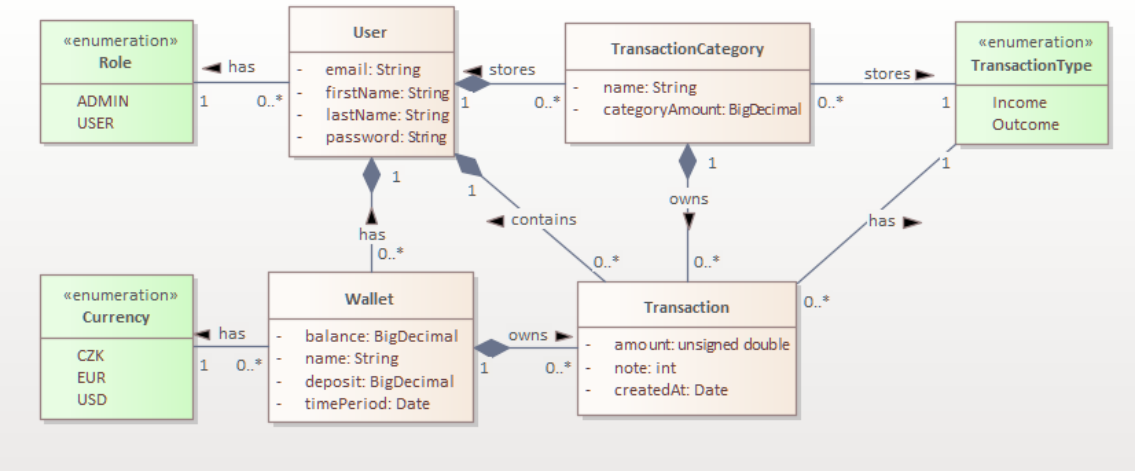
	FP-1	FP-2	FP-3	FP-4	FP-5	FP-6	FP-7	FP-8	FP-9	FP-10
UC 1	+									
UC 2		+								
UC 3			+			+				
UC 4					+					
UC 5							+			
UC 6										+
UC 7										+
UC 8								+		
UC 9									+	
UC 10				+						

Tabulka 4 Pokrytí požadavků případy užití

Je vidět že všechny požadavky jsou pokryty alespoň jedním případem užití. Některé funkční požadavky jsou vyplněny několika případy užití. Tato tabulka nám může říct, jak to že jsme udělali víc případu užití než potřebujeme, tak i to že jsme nějaký zapomněli udělat.

5.3 DOMÉNOVÝ MODEL

Jedním z nejdůležitějších kroků analýzy projektu je vytvořeni doménového modelu. Pro vytváření modelu použijeme jazyk UML, přesněji Class Diagram. Doménový model je strukturovaná vizuální reprezentace vzájemně propojených konceptů nebo objektů reálného světa, která zahrnuje slovní zásobu, klíčové koncepty, chování a vztahy všech svých entit.



Obrázek 14 Ukázka doménového modelu aplikace teto práce

User

Uživatel (User) je osoba, která něco používá nebo provozuje.

Role

Role je entita v systému, zahrnující různá oprávnění a odpovědnosti uživatele.

Wallet

Peněženka (Wallet) je entita, která obsahuje všechny výdaje a příjmy, a také počáteční zůstatek. Na základě těchto informací se vytvoří obecný přehled o vašich penězích, jinými slovy, kolik vám zbývá.

Currency

Měna (Currency) je entita, která popisuje měnu celé peněženky (Wallet) .

Transaction

Transakce (Transaction) je entita, která popisuje náklady a výnosy uživatele.

Transaction Category

Transakční kategorie (Transaction Category) je entita, která ukládá a popisuje druhy výdajů a příjmů.

Transaction Type

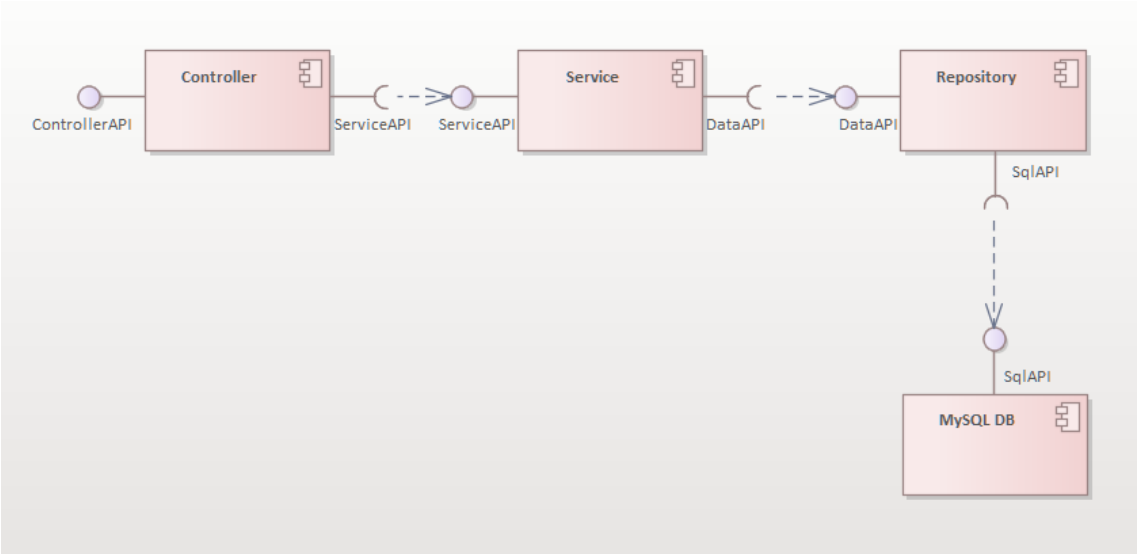
Druh transakce (Transaction Type) je entita, která popisuje jaké transakce mohou být v této kategorií.

6 NÁVRH

V této kapitole si povíme o návrhu aplikaci dane práci, konkrétně o tom, co budou jednotlivé komponenty aplikace představovat. Naším cílem je navrhnout strukturu aplikace, která splňuje aspekty předchozích kapitol.

6.1 ARCHITEKTURA

Aplikace má vrstvenou architekturu, což nám říká, že existují různé vrstvy pro různé potřeby. Pro tuto práci byly vybrány 3 vrstvy: Controller, Service a Repository. Každá z těchto vrstev má svou vlastní jedinečnou funkci. V takové architektuře leží jedna vrstva na druhé a každá vrstva může interagovat pouze s vrstvou nad nebo pod, ale nemůže vrstvu žádným způsobem přeskakovat. Například horní vrstva (třetí) nemůže nijak interagovat se spodní (první).



Obrázek 15 Ukázka Component diagramu aplikace teto práce

6.2 SERVER

Server je prostě počítač, který naslouchá příchozím požadavkům. Ačkoli existují stroje vyrobené a optimalizované pro tento konkrétní účel, každý počítač, který je připojen k síti, může fungovat jako server. Ve skutečnosti budeme při vývoji aplikací používat jako server svůj vlastní počítač. Server naší aplikace bude mít 3-vrsvovou architekturu, jak bylo řečeno drive v kapitole Architektura

6.3 KLIENT

Klienti jsou cokoli/kdokoli, co/kdo odesílá požadavky na backend. Často se jedná o prohlížeče, které požadují kód HTML[17] a JavaScript[18], který vykonají, aby zobrazily webové stránky koncovému uživateli. Existuje však mnoho různých druhů klientů: mohou to být mobilní aplikace, aplikace běžící na jiném serveru nebo dokonce chytré zařízení s podporou webu.

[17] HTML je zkratka pro HyperText Markup Language. Jedná se o standardní značkovací jazyk pro tvorbu webových stránek.

[18] JavaScript je skriptovací jazyk, který umožňuje vytvářet dynamicky aktualizovaný obsah, ovládat multimédia, animovat obrázky.

6.4 CONTROLLER

Controller je vrstva, která kontroluje, zda má uživatel přístup k danému zdroji, což zajistí ochranu vašich dat. Controller také určuje vrácená data na základě očekávaného využití a blokuje nežádoucí volání API, stejně jako požadavky a odpovědi, pokud data neodpovídají pravidlům Controllera.

6.5 SERVICE

Servisy zpracovávají obchodní logiku, což znamená, že jsou zodpovědné za transformaci dat, provádění dalších akcí (např. vyžádání dalších dat Repository nebo jiné servisy pro zpracování nějaké logiky). Pokud tedy potřebujeme odeslat něco, služba je zodpovědná za získání dat jako parametrů, jejich formátování do jiné podoby a jejich odeslání prostřednictvím Controllera nebo uložení prostřednictvím Repositiry.

6.6 REPOSITORY

Repository vrstva odpovídá za komunikaci s databázi. Zde, například můžeme umístit svoje SQL dotazy nebo operace ORM[19]. Ve většině případů tato vrstva má nejmenší implementaci kvůli velkému poctu frameworku a specifikací jazyků.

6.7 MYSQL DB

Databáze je organizovaný soubor strukturovaných informací nebo dat, obvykle uložených elektronicky v počítačovém systému. V našem případě je database MySQL, která je modelována v řádcích a sloupcích v řadě tabulek.

[19] ORM – je zkratka pro Object Relationship Mapping. Je to technika používaná při vytváření „mostu“ mezi objektově orientovanými programy a ve většině případů relačními databázemi. Neznámým příkladem ORM v Javě je Hibernate

7 IMPLEMENTACE

Implementace backendu aplikace je klíčovou částí vývoje, která zajišťuje správné fungování a provoz systému. Tato kapitola se zaměřuje na detailní popis postupu, nástrojů a technologií, které byly použity při implementaci backendu. Odkaz na implementaci backendu aplikace pro správu osobních financí: https://gitlab.fel.cvut.cz/khomumik/fin_backend

7.1 REPOSITORY – DATOVÁ VRSTVA

Datová vrstva je klíčovým prvkem backendové aplikace, který se zabývá správou a manipulací s daty. V této kapitole se zaměříme na implementaci datové vrstvy pomocí JPA Repository poskytovaného Spring Frameworkem. JPA Repository je výkonný nástroj, který usnadňuje práci s relačními databázemi a poskytuje jednoduché rozhraní pro komunikaci s daty.

7.1.1 Úvod do JPA Repository

JPA (Java Persistence API) je standardní Java API pro perzistentní ukládání a načítání objektů z relačních databází. JPA Repository je součástí Spring Data, který přidává další vrstvu abstrakce nad JPA, aby usnadnil práci s daty. Repository poskytuje jednoduché metody pro vytváření, čtení, aktualizaci a mazání (CRUD) datových entit - viz [34].

7.1.2 Návrh datového modelu

Před implementací datové vrstvy je nutné navrhnout datový model aplikace. Identifikujeme klíčové entity, které budou reprezentovat naše data, a vytvoříme odpovídající databázové tabulky. Pro každou entitu definujeme příslušné atributy a jejich vztahy s ostatními entitami - viz 5.3.

7.1.3 Implementace JPA Repository

Pro každou entitu vytvoříme rozhraní JPA Repository, které rozšiřuje základní rozhraní poskytované Springem. Rozhraní definuje metody pro základní operace CRUD, jako je vytváření, čtení, aktualizace a mazání entit. Spring Data JPA automaticky implementuje tyto metody na základě názvů metod a konvencí.

TransactionDao.java

```
1   package com.example.fin.dao;
2
3   import ...
9
    16 usages  Khomutov, Mikhail
10  public interface TransactionDao extends JpaRepository<Transaction, Long> {
        1 usage  Khomutov, Mikhail
11      Optional<List<Transaction>> findAllByUserId(Long userId, Sort sort);
        1 usage  Khomutov, Mikhail
12      Optional<List<Transaction>> findAllByUserIdAndWalletId(Long userId, Long walletId);
        3 usages  Khomutov, Mikhail
13      Optional<List<Transaction>> findAllByTransactionCategoryId(Long transactionCategoryId, Sort sort);
        2 usages  Khomutov, Mikhail
14      Optional<Transaction> findByIdAndUserId(Long id, Long userId);
15  }
16
```

Obrázek 16 Příklad implementaci datové vrstvy pro Transakční entitu

JPA Repository také poskytuje možnosti pro dotazování nad daty. Pomocí speciálních klíčových slov a výrazů můžeme definovat složitější dotazy a filtrovat data na základě různých kritérií. Spring Data JPA převádí tyto dotazy na SQL dotazy a provádí je nad databází - viz [34]. V našem případě pro potřeby aplikace byly vytvořeny další dotazy, které jsou ukázaný na obrázku výše.

7.2 Model – Entity – ORM[20]

Entity jsou objekty, které reprezentují konkrétní datové struktury v aplikaci. Tyto struktury odpovídají databázovým tabulkám, a proto se mohou ukládat a načítat z relačních databází. Entity obsahují atributy, které přesně popisují data, která reprezentují, a mohou obsahovat také metody pro manipulaci s těmito daty.

7.2.1 Implementace

Implementace entit s využitím Hibernate - viz [35], Project Lombok - viz [18], Spring Frameworku - viz 4.3.2.1 a JPA. Pro implementaci entit v naší aplikaci využijeme kombinaci několika technologií, které usnadní a zefektivní proces vývoje. Hibernate je populární nástroj pro objektově-relační mapování (ORM), který zajišťuje převod entit na databázové tabulky a zpět. Project Lombok je knihovna, která pomocí anotací usnadňuje tvorbu entit tím, že generuje kód za běhu. Spring Framework poskytuje podporu pro správu entit, transakce a další klíčové funkcionality. JPA (Java Persistence API) je standardní Java API pro perzistentní ukládání a načítání objektů z relačních databází.

```
Khomutov, Mikhail
@Entity
@Data
@Builder
@NoArgsConstructor
@AllArgsConstructor
public class Wallet {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @NotNull
    private String name;

    @NotNull
    private BigDecimal deposit; // first deposit on account

    @Transient
    private BigDecimal balance = BigDecimal.ZERO; // calculated balance taking deposit and transactions

    @Column(nullable = false)
    private LocalDate timePeriod;

    @NotNull
    @Enumerated(EnumType.STRING)
    @Column(columnDefinition = "ENUM('CZK', 'EUR', 'USD')")
    private Currency currency;

    @OneToMany(mappedBy = "wallet", orphanRemoval = true, fetch = FetchType.LAZY)
    private Set<Transaction> transactions = new HashSet<>();

    @NotNull
    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "user_id", nullable = false)
    private User user;
}
```

Obrázek 17 Příklad implementaci entity databázového modelu Wallet

[20] ORM (Object-Relational Mapping) je technika, která umožňuje mapování objektového modelu aplikace na relační model uložený v relační databázi. Pomocí ORM můžeme pracovat s objekty a jejich atributy namísto přímého manipulování s tabulkami a sloupci v databázi.

V tomto příkladu je entita Wallet definována pomocí anotací @Entity, které označují, že tato třída je entitou a odpovídá tabulce "wallet" v databázi. Anotace @Id a @GeneratedValue slouží k definici primárního klíče a jeho automatickému generování.

Atributy třídy jsou mapovány na sloupce v databázi přímo i pomocí anotace @Column. Anotace @NotNull slouží k označení, že daný atribut nebo parametr nemůže mít hodnotu null. Pokud je hodnota null, validace selže a bude vyvolána výjimka nebo jiná chybová zpráva. Anotace @JoinColumn umožňuje upřesnit, jaký sloupec v cílové entitě bude sloužit jako cizí klíč pro vazbu na zdrojovou entitu. @Transient se používá k označení atributu entity, který by neměl být persistentní, tj. neukládá se do databáze. @ManyToOne anotace se obvykle používá v rámci entity, která představuje "strana mnoha" ve vztahu, v našem případě Wallet může vlastnit jednoho User a User naopak může mít 0 a více Wallet.

7.3 Aplikační vrstva – Servisní vrstvá

Servisní vrstva je klíčovým komponentem architektury aplikace, která zajišťuje zpracování obchodní logiky a koordinaci mezi různými částmi systému. V této kapitole se budeme zabývat implementací servisní vrstvy pomocí Spring Frameworku.

7.3.1 Úvod do servisní vrstvy

Servisní vrstva je vrstva mezi prezentační (Controller) a datovou vrstvou (Repository). Jejím hlavním úkolem je zpracování obchodní logiky aplikace, jako je validace dat, výpočty, provádění operací nad daty a koordinace komunikace mezi různými komponentami systému. Servisní vrstva slouží k oddělení obchodní logiky od prezentační a datové vrstvy, což zajišťuje modulárnost, znovupoužitelnost a testovatelnost aplikace.

7.3.2 Implementace servisní vrstvy

Spring Framework poskytuje robustní podporu pro implementaci servisní vrstvy. Jeho IoC (Inversion of Control) viz [36] kontejner umožňuje správu a vytváření závislostí mezi komponentami. Anotace jako @Service a @Autowired usnadňují vytváření a používání servisních tříd.

```
    Khomutov, Mikhail
19  @Service
20  @Slf4j
21  @RequiredArgsConstructor
22  public class WalletServiceImpl implements WalletService {
23
24      private final WalletDao walletDao;
25      private final UserService userService;
26      private final TransactionService transactionService;
27
        1 usage   Khomutov, Mikhail
28      @Override
29      public List<Wallet> getAllWallets(Sort sort) {
30          var userId = userService.getCurrentUser().getId();
31          var wallets = walletDao.findAllByUserId(userId, sort)
32                  .orElseThrow(() -> new ResourceNotFoundException("Wallets with given id were not found"));
33          wallets.forEach(this::actualizeWallet);
34          return wallets;
35      }
36
        1 usage   Khomutov, Mikhail
37      @Override
38      public Wallet getOneWallet(Long walletId) {
39          var userId = userService.getCurrentUser().getId();
40          var walletFromDb = walletDao.findByIdAndUserId(walletId, userId)
41                  .orElseThrow(() -> new ResourceNotFoundException("Wallet with given id was not found"));
42          actualizeWallet(walletFromDb);
43          return walletFromDb;
44      }
```

Obrázek 18 Ukázka servisy pro Wallet

Při implementaci servisní vrstvy je důležité dodržovat principy správného návrhu a rozdělit funkčnost do jednotlivých servisních tříd, které mají jasně definovanou odpovědnost. Servisní třídy by měly být nezávislé na konkrétní implementaci datové vrstvy a měly by pracovat s rozhraním datové vrstvy (repository) pomocí injektování závislostí.

V tomto příkladu je implementována třída WalletServiceImpl, která implementuje rozhraní WalletService. Tato třída je označena anotací @Service, čímž je označena jako servisní komponenta. V konstruktoru jsou injektovány závislosti na WalletDao, UserService a TransactionService, které jsou potřebné pro manipulaci s daty peněženek. Anotace @Slf4j je součástí projektu Lombok a slouží k automatickému generování logovacího objektu pro třídu, ve které je použita. @RequireArgsConstructor vygeneruje konstruktor, který přijímá jako parametry všechny atributy třídy, které jsou označeny anotacemi @NonNull nebo jsou final. Je to také anotace projektu Lombok.

Na obrázku je vidět část kodu pro peněženky. Tyto metody zahrnují logiku aplikace. Např. metoda *getOneWallet()* implementuje omezení přístupu ke zdroji Wallet, konkrétně neumožňuje uživateli získat peněženku někoho jiného z databáze. Následuje aktualizace atributů peněženky, které nejsou uloženy v databázi.

```
4 usages  Khomutov, Mikhail
84  @   private void actualizeWallet(Wallet wallet) { wallet.setBalance(calculateBalance(wallet)); }
87
88
        1 usage  Khomutov, Mikhail
89  @   private BigDecimal calculateBalance(Wallet wallet) {
90          var transactions = transactionService.getAllTransactionsFromWallet(wallet.getId());
91          var result = transactions.stream() Stream<Transaction>
92                  .map(transaction -> transaction.isIncome() ? transaction.getAmount() : transaction.getAmount().negate()) Stream<BigDecimal>
93                  .reduce(BigDecimal::add) Optional<BigDecimal>
94                  .orElse(BigDecimal.ZERO);
95          return wallet.getDeposit().add(result);
96      }
97  }
98
```

Obrázek 19 Příklad počítání zůstatků u peněženky

V tomto případě se jedná o aktualizace zůstatků, které jsou považovány za součet transakcí. To se provádí proto, aby bylo snadné aktualizovat některé atributy, které závisí na jiných objektech.

7.4 PREZENTAČNÍ VRSTVA

Prezentační vrstva se stará o komunikaci s uživatelem a prezentaci dat. V této kapitole se zaměříme na návrh a implementaci prezentační vrstvy v naší aplikaci s využitím Spring Frameworku a anotace @RestController.

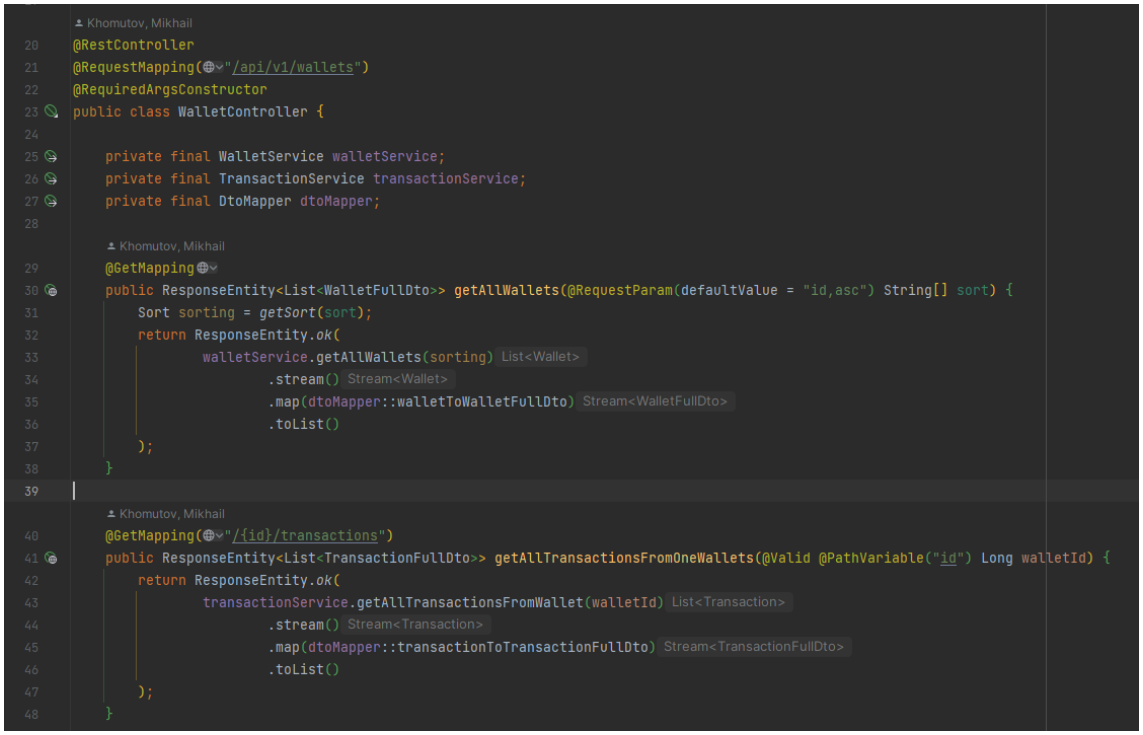
7.4.1 Implementace prezentační vrstvy

V naší implementaci prezentační vrstvy využijeme anotaci @RestController, která kombinuje anotace @Controller a @ResponseBody. Tímto způsobem se zajišťuje automatická serializace objektů na formát odpovědi (např. JSON[21]) a umožňuje snadnou implementaci RESTful rozhraní.

Prezentační vrstva jsou uši a ústa naší aplikace. A proto potřebujeme označit kde můžeme poslouchat dotazy od klientské časti aplikace. Pro tyto účely potřebujeme anotace @RequestMapping() a další @GetMapping, @PostMappingm, @PatchMapping, @DeleteMapping a další, které jsou rozšířením funkcionality. Anotace @RequestMapping se

[21] JSON (JavaScript Object Notation) je formát datového výměnného formátu, který se často používá pro přenos dat mezi klientem a serverem ve webových aplikacích.

používá na úrovni třídy, specifikuje základní cestu, která se bude používat pro všechny metody v rámci této třídy. Další anotace GET, POST atd. specifikují konkrétní URL cestu a HTTP metodu, kterou metoda obslouží.



Obrázek 20 Ukázka třídy WalletController

V tomto příkladu metoda getAllWallets je mapována na GET požadavek s URL cestou /api/v1/wallets. Metoda getAllTransactionsFromOneWallets je mapována na GET požadavek s URL cestou /api/wallets/{id}/transactions, kde {id} je proměnná, která bude nahrazena konkrétním ID peněženky.

V metodě getAllWallets zahrnutá jedná z tak zvaných “feature”[22]. Je to možnost třídění odpovědí controllera podle specifického atributu entity Wallet. V tomto příkladu je metoda označena anotací @GetMapping a URL cestou /wallets. Parametr “sort” je specifikován v anotaci @RequestParam("sort") a odpovídá názvu parametru ve vstupní metodě. Pokud byl ve vstupním HTTP požadavku předán parametr sort (např. “?sort=name”), bude tato hodnota předána do metody. „defaultValue“ specifikuje výchozí hodnotu pokud URL neobsahuje parametr sort.

7.4.2 Dto objekty - přepravka dat

Jak jste mohli všimnout v Controlleru, který jsem popisoval výše v těle obou metod je neznámý “dtoMapper”. Je to DTO (Data Transfer Object, dále jenom DTO) je objekt, který reprezentuje strukturu dat a slouží k přenosu dat mezi různými komponentami systému. Použití DTO pomáhá oddělit doménovou logiku od prezentační vrstvy a zvyšuje modularitu a flexibilitu systému

[22] Termín "feature" se používá k označení specifické funkcionality nebo schopnosti, kterou aplikace poskytuje uživatelům.

Pro mapovaní Dto objektu existuje openSource[23] framework[24] v Javě – MapStruct - viz [37]. Je navržen tak, aby zjednodušil a automatizoval proces mapování mezi objekty různých typů, zejména mezi entitními objekty a objekty DTO (Data Transfer Object). Jeho hlavním cílem je minimalizovat ruční psaní rutinního a opakujícího se kódu pro mapování.

```
1   package com.example.fin.mapper;
2
3   import ...
10
    9 usages  1 implementation  Khomutov, Mikhail
11  @Mapper(componentModel = "spring")
12  public interface DtoMapper {
        4 usages  1 implementation  Khomutov, Mikhail
13      WalletFullDto walletToWalletFullDto(Wallet wallet);
        1 usage  1 implementation  Khomutov, Mikhail
14      Wallet walletPostRequestDtoToWallet(WalletPostRequestDto request);
        6 usages  1 implementation  Khomutov, Mikhail
15      @Mapping(target = "transactionCategoryId", source = "transaction.transactionCategory.id")
16      @Mapping(target = "transactionCategoryName", source = "transaction.transactionCategory.name")
17      TransactionFullDto transactionToTransactionFullDto(Transaction transaction);
        4 usages  1 implementation  Khomutov, Mikhail
18      TransactionCategoryFullDto transactionCategoryToTransactionCategoryFullDto(TransactionCategory transactionCategory);
        1 usage  1 implementation  Khomutov, Mikhail
19      TransactionCategory transactionPostRequestDtoToTransactionCategory(TransactionCategoryPostRequestDto request);
        2 usages  1 implementation  Khomutov, Mikhail
20      UserFullDto userToUserFullDto(User user);
21  }
22
```

Obrázek 21 Ukázka implementací DTO mapperu pomocí MapStruct

Implementace je velmi jednoduchá, potřebujete vytvořit mapovací rozhraní, ve kterém budete definovat metody pro mapování mezi objekty. Např. metoda *transactionToTransactionFullDto(...) viz obrázek výše*, mapuje objekt "Transaction" na „TransactionFullDto". Parametrem metody je zdroj mapování a výsledkem je DTO objekt. V tomto příkladu bylo pomocí anotace @Mapping specifikováno, že při mapování z objektu "Transaction" na „TransactionFullDto" se má hodnota atributu "id" a " name" zkopírovat do atributu „transactionCategoryId" a „transactionCategoryName".

7.5 AUTENTIFIKACE

Autentifikace je důležitou součástí vývoje moderních webových aplikací, která umožňuje ověřování a identifikaci uživatelů. Zaměřil jsem se na implementaci autentifikace v aplikaci s využitím JWT (JSON Web Token) – viz [38]. JWT je otevřený standard pro výměnu informací mezi stranami ve formátu JSON. Jedná se o kompaktní a samo výstražný způsob zabezpečení, který umožňuje přenášet údaje o uživateli mezi klientem a serverem pomocí podepsaného tokenu. JWT obsahuje informace o uživateli a jeho oprávněních.

7.5.1 Implementace

V rámci Spring Framework jsem konfiguroval bezpečnostní nastavení, které definuje pravidla pro přístup k různým částem aplikace. Zde jsem určil, které cesty vyžadují autentifikaci a autorizaci.

V aplikaci je to realizováno tím, že existují 2 koncové body, pro které není vyžadována autentifikace, a to POST "/api/v1/auth/register" a POST "/api/v1/auth/authenticate". První slouží k registraci nových uživatelů a vydání tokenu a druhá slouží k autentifikace již existujícího uživatelského účtu, který zároveň vydá nový token pro přístup do systému

[23] Open Source Software je software, jehož zdrojové kódy jsou volně k dispozici. Dobrovolní vývojáři mohou zdrojové kódy studovat a modifikovat.
[24] Framework je softwarová struktura pro podporu programování, vývoje a organizaci jiných softwarových projektů

```java
Khomutov, Mikhail
@Component
@RequiredArgsConstructor
public class JwtAuthenticationFilter extends OncePerRequestFilter {

    private final JwtService jwtService;
    private final UserDetailsService userDetailsService;

    Khomutov, Mikhail
    @Override
    protected void doFilterInternal(
            @NonNull HttpServletRequest request,
            @NonNull HttpServletResponse response,
            @NonNull FilterChain filterChain
    ) throws ServletException, IOException {
        final String authHeader = request.getHeader( name: "Authorization");
        final String jwt;
        final String userEmail;
        if (authHeader == null || !authHeader.startsWith("Bearer ")) {
            filterChain.doFilter(request, response);
            return;
        }
        jwt = authHeader.substring( beginIndex: 7);
        userEmail = jwtService.extractUsername(jwt);
        if (userEmail != null && SecurityContextHolder.getContext().getAuthentication() == null) {
            UserDetails userDetails = this.userDetailsService.loadUserByUsername(userEmail);
            if (jwtService.isTokenValid(jwt, userDetails)) {
                var authToken = new UsernamePasswordAuthenticationToken(
                        userDetails,
                        credentials: null,
                        userDetails.getAuthorities()
                );
                authToken.setDetails(new WebAuthenticationDetailsSource().buildDetails(request));
                SecurityContextHolder.getContext().setAuthentication(authToken);
            }
        }
        filterChain.doFilter(request, response);
    }
}
```

Obrázek 22 Ukázka implementaci filtru pro autentifikaci

Vrstvy Controller a Service už známé a chtěl bych vám představit implementací filtru, který je důležitým prvkem pro bezpečnost aplikace – viz obrázek na další stránce. Filtr se používá k ověřování identity uživatele na základě JWT tokenu ve webových aplikacích. Tento filtr se integruje do aplikačního kontextu a zajišťuje proces autentifikace a autorizace uživatelů při přístupu k chráněným zdrojům, která jsou nastavené v konfiguračním souboru.

Autentifikační filtr extrahuje z JWT tokenu identifikační údaje uživatele, jako email. Na základě extrahovaných identifikačních údajů filtr provede autentifikaci uživatele pomocí porovnání hesla s uloženou hodnotou v databázi. Po úspěšné autentifikaci filtr vytvoří autentifikovaný kontext uživatele, který je dostupný pro další části aplikace. Po vytvoření autentifikovaného kontextu filtr předá požadavek ke zpracování dalším komponentám aplikace, jako jsou řadiče (controllers) nebo další filtry. Tyto komponenty mohou využívat autentifikovaný kontext k rozhodování o přístupu k chráněným prostředkům.

8 TESTOVÁNÍ

Testování je nedílnou součástí vývoje softwarových aplikací, které pomáhá ověřit jejich správnou funkčnost, identifikovat chyby a zajistit kvalitu výsledného produktu. V rámci této práce jsem provedl testování aplikace, která je postavena na Spring, a využil jsem nástroje jako JUnit 5 – viz [31] pro psaní testů a provedení integračních testů na API.

8.1 INTEGRAČNÍ TESTY

Integrační testování je testování, které zahrnuje více než jednu vrstvu aplikace. V souladu s tím byly pro každý koncový bod připraveny testy, které zohledňovaly vlivy component na výsledky testu.

Pro provedení integračních testů byl použit nástroj MockMvc[25], který je součástí Spring Framework. MockMvc umožňuje emulovat HTTP požadavky na vaše API a provádět testy na různých vrstvách vaší aplikace, včetně prezentační vrstvy. S MockMvc můžete otestovat správné chování vašich kontrolérů, zpracování požadavků a ověřit odpovědi.

```
138
        9 usages   Khomutov, Mikhail
139     protected ResultActions performGet(String url, String token) throws Exception {
140         url = BASE_URL + url;
141         return mockMvc.perform(get(url)
142                         .header( name: "Authorization",  ...values: "Bearer " + token)
143                         .contentType(MediaType.APPLICATION_JSON))
144                 .andExpect(content().contentTypeCompatibleWith(MediaType.APPLICATION_JSON));
145     }
146
        3 usages   Khomutov, Mikhail
147     protected ResultActions performPost(String url, Object payload, String token) throws Exception {
148         String content = objectWriter.writeValueAsString(payload);
149         url = BASE_URL + url;
150         return mockMvc.perform(post(url)
151                 .header( name: "Authorization",  ...values: "Bearer " + token)
152                 .contentType(MediaType.APPLICATION_JSON)
153                 .content(content));
154     }
```

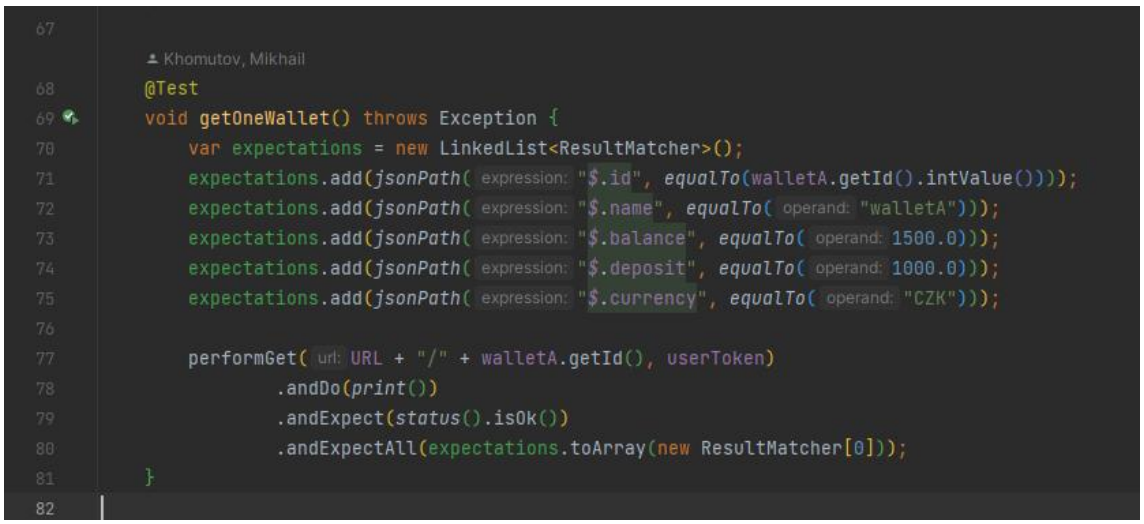
Obrázek 23 Ukázka implementaci testovací třídy IntegrationTest

Na obrázku výše můžete vidět, jak bylo implementováno testování API pomocí mockMvc. Tato část kódu naznačuje, že autentizace nezasahuje do integračního testování a může být spuštěna paralelně, čímž se test komplikuje a zlepšuje se tak jeho výsledky z hlediska pokrytí testemy.

Obrázek níže již používá Mockmvc pro svůj účel. Vzhledem k tomu, že metoda performGet vrací ResultActions[26], můžeme pohodlně otestovat výstupy tohoto api požadavku ve formátu JSON. Rád bych také zdůraznil použití propojeného seznamu pro účely testování. Přidáním listu do metody andExpectAll, což je kontrola platnosti, kde můžeme testovat platnost json objektů, test ohlásí všechny chyby najednou, pokud nějaké existují. Např, pokud “name” a “balance” najednou budou mít špatné hodnoty, tak test ukáže, že tyto ě hodnoty byly špatné.

[25] MockMvc je nástroj, který umožňuje emulovat HTTP požadavky a získávat odpovědi, aniž by bylo nutné spouštět vaši aplikaci v reálném běhovém prostředí.

[26] ResultActions je objektem, který poskytuje pohodlné metody pro provádění kontrol a extrahování informací z výsledku požadavku HTTP v řadiči během testování integrace. Je součástí MockMvc



Obrázek 24 Ukázka implementaci integračního testu WalletIntegrationTest

8.2 VÝSLEDKY TESTOVÁNÍ

Pomocí integračního testování byly identifikovány některé nedostatky aplikace, které byly opraveny. Aplikační pokrytí testy není 100%. Do budoucna však může hrát velkou roli jako upozornění na problémy v systému při implementaci následné funkcionality. Kromě testování integračního byla připravená postman kolekce [26] pro ruční testování.

9 ZÁVĚR

Výsledkem této práce jsou obrovské zkušenosti s analytickou a implementační prací na aplikaci „Správa osobních financí“, což je ovšem výhodou pro začínajícího softwarového programátora.

V důsledku samostatný práce byly splněny tyto úkoly:

1. Byl nastudován proces vedení osobního rozpočtu;
2. Byla provedena srovnávací analýza stávajících mobilních aplikací pro správu rodinného rozpočtu.
3. Byla provedena srovnávací analýza komponent aplikace
4. Byl proveden konceptuální návrh aplikace
5. Byl implementován backend aplikace
6. Byly vytvořený integrační testy pokrývající základní testovací scénáře pro endpointy

Základní funkčnost aplikace je implementovaná kromě jednoho připadá užití a to je “Export uživatelských dat do Excel”

Použití této aplikace zautomatizuje práci s vedením osobních financi, výrazně zkrátí čas strávený evidencí výdajů a příjmů, usnadní proces analýzy a plánování a zvýší celkovou efektivitu rozdělování rozpočtu.

Odkaz na implementaci backendu aplikace pro správu osobních financí:
https://gitlab.fel.cvut.cz/khomumik/fin_backend

10 LITERATURA

[1] STÁTNÍ HUMANITÁRNÍ PEDAGOGICKÁ UNIVERZITA PERM; *Závěrečná kvalifikační práce vývoj mobilní aplikace "ŘÍZENÍ RODINNÉHO (DOMÁCÍHO) ROZPOČTU";* Dostupné z: https://vkr.pspu.ru/uploads/180/Sannikova_vkr.pdf, [online]; -.-.2017

[2] Vesti.ua: *„Proč potřebujete aplikace pro finanční účetnictví a jak je používat“*; Dostupné z: https://vesti.ua/opinions/zachem-nuzhny-prilozheniya-dlya-ucheta-finansov-i-kak-imi-polzovatsya, [online]; 11.03.2021

[3] VC.ru*; Jak a proč jsem napsal kontrolu svých výdajů*; Dostupné z: https://vc.ru/tribuna/59422-kak-i-zachem-ya-napisal-svoy-kontrol-rashodov, [online]; 25.02.2019

[4] Prezentace na téma “Aplikace pro správu osobních financí”; Dostupné z: https://ppt-online.org/370456, [online]

[5] Kellton.com; *8 Backend Web Development Frameworks in 2022*; Dostupné z: https://www.kellton.com/kellton-tech-blog/top-8-backend-web-development-frameworks-in-2022, [online]; 31.03.2022

[6] Habr.com; *Web. Historie jedné technologie*; Dostupné z: https://habr.com/ru/company/geekbrains/blog/277957/, [online]; 25.02.2016

[7] IBM.com; *What is Java Spring Boot?*; Dostupné z: https://www.ibm.com/topics/java-spring-boot, [online];

[8] Dev.to; *Backend: Layered Architecture*; Dostupné z: https://dev.to/blindkai/backend-layered-architecture-514h, [online]; 23.01.2022

[9] Dotnet.microsoft.com; *What is ASP.NET Core?*; Dostupné z: https://dotnet.microsoft.com/en-us/learn/aspnet/what-is-aspnet-core, [online];

[10] Dou.ua; *“Výhody a nevýhody .NET: rychlý rozvoj, vysoká rozšířenost a průměrné platy”*; Dostupné z: https://dou.ua/lenta/articles/pros-and-cons-of-dotnet/?hl=ru, [online]; 9.02.2022

[11] Virtual Conference on Engineering, Science and Technology (ViCEST) 2020*; A Performance Comparison of RESTful Applications Implemented in Spring Boot Java and MS.NET Core*; Dostupné z: https://iopscience.iop.org/article/10.1088/1742-6596/1933/1/012041/pdf, [online]; 2020

[12] SimilarTech.com; *Asp .Net VS Django*; Dostupné z: https://www.similartech.com/compare/asp-net-vs-django, [online];

[13] IBM.com; *SQL vs. NoSQL Databases: What's the Difference?*; Dostupné z: https://www.ibm.com/cloud/blog/sql-vs-nosql, [online]; 12.06.2022

[14] Katedra informatiky PřF UJEP v Ústí nad Labem, 2020; *NoSQL databázové systémy*; Dostupné z: https://ki.ujep.cz/opory/Aplikovana_Informatika/Bc/NoSQL_databazove_systemy.pdf, [online]; 2020

[15] GeekForGeeks.org; *Difference between SQL and NoSQL*; Dostupné z: https://www.geeksforgeeks.org/difference-between-sql-and-nosql/, [online]; 15.11.2022

[16] CVUT FEL, Softwarové inženýrství; *UML diagramy případů užití*; Dostupné z: https://moodle.fel.cvut.cz/pluginfile.php/350314/course/section/61830/P%C5%99edn%C3%A1%C5%A1ka3_PripadyUzitiKomplet_SIN_ZS_2020.pdf, [online];

[17] Spring.io; *Accessing data with MySQL*; Dostupné z: https://spring.io/guides/gs/accessing-data-mysql/, [online];

[18] ProjectLombok.org; Project Lombok; Dostupné z: https://projectlombok.org/, [online]

[19] Spring.io; *Securing a Web Application*; Dostupné z: https://spring.io/guides/gs/securing-web/, [online];
[20] MavenRepository.com; Maven Repository ; Dostupné z: https://mvnrepository.com/, [online];
[21] Baeldung.com; *Quick Guide to MapStruct*; Dostupné z: https://www.baeldung.com/mapstruct, [online]; 17, 2023
[22] Baeldung.com; *The Spring @Controller and @RestController Annotations*; Dostupné z: https://www.baeldung.com/spring-controller-vs-restcontroller, [online];
[23] Adminer.org; *Adminer*; Dostupné z: https://www.adminer.org/, [online];
[24] Medium.com; *Problems with Hibernate One-To-Many (And Their Solutions)*; Dostupné z: https://medium.com/interleap/problems-with-hibernate-one-to-many-and-their-solutions-8f32af216b95, [online]; 16.03.2022
[25] GeeksForGeeks.org; *Spring Hibernate Configuration and Create a Table in Database*; Dostupné z: https://www.geeksforgeeks.org/spring-hibernate-configuration-and-create-a-table-in-database/, [online]; 30.11.2022
[26] Postman.com; *Postman*; Dostupné z: https://www.postman.com/, [online];
[27] JetBrains.com; *Gradle*; Dostupné z: https://www.jetbrains.com/help/idea/gradle.html#project_create_gradle, [online];
[28] Gradle.org; *Gradle Releases*; Dostupné z: https://gradle.org/releases/, [online];
[29] HowToDoInJava.com; *Spring Boot Pagination and Sorting Example*; Dostupné z: https://howtodoinjava.com/spring-boot2/pagination-sorting-example/, [online]; 12.12.2022
[30] Guru99.com; *UML Association Vs. Aggregation Vs. Composition [EXAMPLE]*; Dostupné z: https://www.guru99.com/association-aggregation-composition-difference.html#uml-composition, [online];
[31] Junit.org; *Running Tests*; Dostupné z: https://junit.org/junit5/docs/current/user-guide/#running-tests-build, [online];
[32] Baeldung.com; *Testing in Spring Boot*; Dostupné z: https://www.baeldung.com/spring-boot-testing, [online];
[33] Linux.org forum; *Kaskádové mazání propojených subjektů*; Dostupné z: https://www.linux.org.ru/forum/development/14841599, [online];
[34] Spring Data JPA. In: *Spring Framework Reference Documentation*. [online]. [cit. 24. 5. 2023]. Dostupné z: https://docs.spring.io/spring-data/jpa/docs/current/reference/html/#repositories
[35] Hibernate.org; *Hibernate ORM*; Dostupné z: https://hibernate.org/orm/, [online];
[36] Baeldung.com; *Intro to Inversion of Control and Dependency Injection with Spring*; Dostupné z: https://www.baeldung.com/inversion-control-and-dependency-injection-in-spring, [online]; 18.03.2023
[37] MapStruct.org; *MapStruct Documentation*; Dostupné z: https://mapstruct.org/documentation/, [online];
[38] JWT.io; Json Web Token; Dostupné z: https://jwt.io/, [online];
[39] Docs.Spring.io ; *Spring Security Reference Documentation*; Dostupné z: https://docs.spring.io/spring-security/site/docs/current/reference/html5/, [online]

11 TABULKY

12 OBRÁZKY A DIAGRAMY