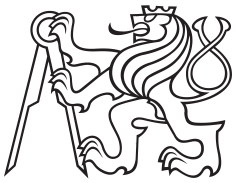
Bakalářská práce



České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická
Katedra počítačů

Cloud native aplikace pro kontrolu plateb

Jiří Pazdera

Školitel: Ing. Martin Komárek
Květen 2021

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Pazdera** Jméno: **Jiří** Osobní číslo: **476054**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Softwarové inženýrství a technologie**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Cloud native aplikace pro kontrolu plateb

Název bakalářské práce anglicky:

Cloud native payment checker application

Pokyny pro vypracování:

Pro potřeby různých ziskových i neziskových organizací (např. sdružení majitelů bytů, členů sportovních klubů, dětských oddílů, fitness center, soukromých škol, ...) vytvořte systém kontrolující bankovní platby (pravidelné i jednorázové).
Klíčové vlastnosti:
1) Evidence plánovaných plateb členů.
2) Stahování informací o platbách minimálně z Fio banky.
3) Párování plánovaných a příchozích plateb.
4) Upomínání opožděných plateb.
5) Dále nastudujte a navrhněte řešení pro:
6) Vracení nespárovaných plateb,
7) Automatický výpočet penále za opožděné platby.
Vývoj provádějte agilním způsobem s využitím cloud native přístupu a mikroservisní architektury.

Seznam doporučené literatury:

1. GARRISON, Justin a Kris NOVA. Cloud Native Infrastructure: Patterns for Scalable Infrastructure and Applications in a Dynamic Environment. Sebastopol, CA: O'Reilly Media, 2017. ISBN 978-1491984307.
2. NEWMAN, Sam. Building microservices. Sebastopol, CA: O'Reilly Media, [2015]. ISBN 978-1491950357.
3. NADAREISHVILI, Nadareishvili, Ronnie MITRA, Matt MCLARTY a Mike AMUNDSEN. Microservice Architecture: Aligning Principles, Practices, and Culture. Sebastopol, CA: O'Reilly Media, 2016. ISBN 978-1491956250.

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Martin Komárek, katedra informační bezpečnosti FIT

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

Datum zadání bakalářské práce: **11.02.2021** Termín odevzdání bakalářské práce: _____

Platnost zadání bakalářské práce: **30.09.2022**

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

III. PŘEVZETÍ ZADÁNÍ

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

. _____
Datum převzetí zadání

_____
Podpis studenta

Poděkování

V první řadě bych chtěl poděkovat firmě Stratox za možnost využívat jejich platformu CodeNOW. Dále bych chtěl poděkovat Tereze Doležalové za poskytnutí odborných konzultací, své kamarádce Zitě Strunecké a své rodině za podporu při studiu.

Prohlášení

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

V Praze, 21. května, 2021

Podpis: .....

I declare that this work is all my own work and I have cited all sources I have used in the bibliography.

Prague, May 21, 2021

Abstrakt

Práce se zaměřuje na implementaci systému, který má usnadnit a částečně automatizovat administraci spojenou s účetnictvím malých spolků. Systém se připojuje do bankovního API Fio banka, a.s. a stahuje si informace o transakcích v bankovním účtu. Systém má mikroservisní architekturu a využívá cloud-native technologie. Práce sestává ze šesti částí. Na začátku se nachází úvod. Druhá část se zabývá analýzou a nastíněním problému, třetí řeší návrh systému jako takového, čtvrtá popisuje popis implementace a problematiku s ní spojenou. Pátou částí je pak kapitola Testování, která obsahuje popis průběhu testování a poslední část je závěr, který shrnuje celou práci.

Klíčová slova: Javascript, Java, React.js, Spring, REST API, Apache Kafka, microservisní architektura, účetní systém

Školitel: Ing. Martin Komárek

Abstract

The thesis focuses on the implementation of a system which should help and automate administration connected with an accounting of small to medium organizations. The system uses the bank API of Fio Banka a.s. for downloading payments. The system is built based on microservice architecture and uses cloud-native technologies. The thesis consists of six chapters. The first one is the introduction. The second chapter focuses on the analysis of the problem, the third one is a description of system design, the fourth one describes the implementation process and the fifth one is a chapter about testing of the system. The last chapter summarizes the whole thesis.

Keywords: Javascript, Java, React.js, Spring, REST API, Apache Kafka, microservice architecture, accountings system

Title translation: Cloud native payment checker application

Obsah

Obrázky

Tabulky

Kapitola 1
Úvod

Mnoho neziskových organizací a spolků řeší problémy s placením členských příspěvků. Účetní těchto spolků často vedou různé evidence svých členů v Excelových tabulkách a musí manuálně dohledávat platby ve výpisu bankovního účtu. Během těchto manuálních činností může docházet k chybám. Zároveň musí účetní jednotlivě oznamovat všem plátcům, že od nich očekává platby případně, že se zpozdili se zaplacením svých příspěvků.

Cílem práce je analyzovat, navrhnout a implementovat systém založený na bázi nativních cloudových technologií a mikroservisní architektury pro použití v účetnictví neziskových organizací.

V posledních deseti letech distribuované systémy opouštějí velké monolitické systémy a zaměřují se spíše na menší,samostatné mikroslužby[1]. Výhodou využití mikroslužeb je především modulárnost a snadná horizontální škálovatelnost. Díky modulárnosti lze dílčí části systému spravovat nezávisle na sobě, což usnadňuje vývoj systému, následné testování a případné změny v architektuře. Horizontální škálovatelnost se vyznačuje přidáním více procesů, mezi kterými se rozdělí příchozí požadavky. Velkým systémům jako je např. Netflix to umožňuje pružně reagovat na změny zatížení jejich aplikace[2].

Cloud native infrastruktura je základním stavebním kamenem pro běh cloud native aplikací[3]. Cloud native aplikace jsou kolekcemi malých nezávislých volně vázaných služeb. Přinášejí byznysové výhody pro systémy, které prosperují z rychlého zakomponování uživatelské zpětné vazby. Cloud native přístup k vývoji aplikací je účinným způsobem jak zrychlit vývoj nových aplikací, optimalizaci stávajících a jejich propojení[4].

V první části dokumentu je vedena analýza a návrh řešení problémů podle předem definovaných funkčních požadavků. Dále je představena implementace návrhu řešení a na kterém jsou proveden vývojové testy.

Kapitola 2
Analýza

V této kapitole rozeberu byznysové požadavky, tedy co by systém měl dělat. Vysvětlím zde jaké objekty z reálného světa systém bude sledovat a jaké jsou mezi nimi vztahy. Všechny požadavky a návrhy vycházely z konzultací se zadavatelem.

2.1 Byznysový model

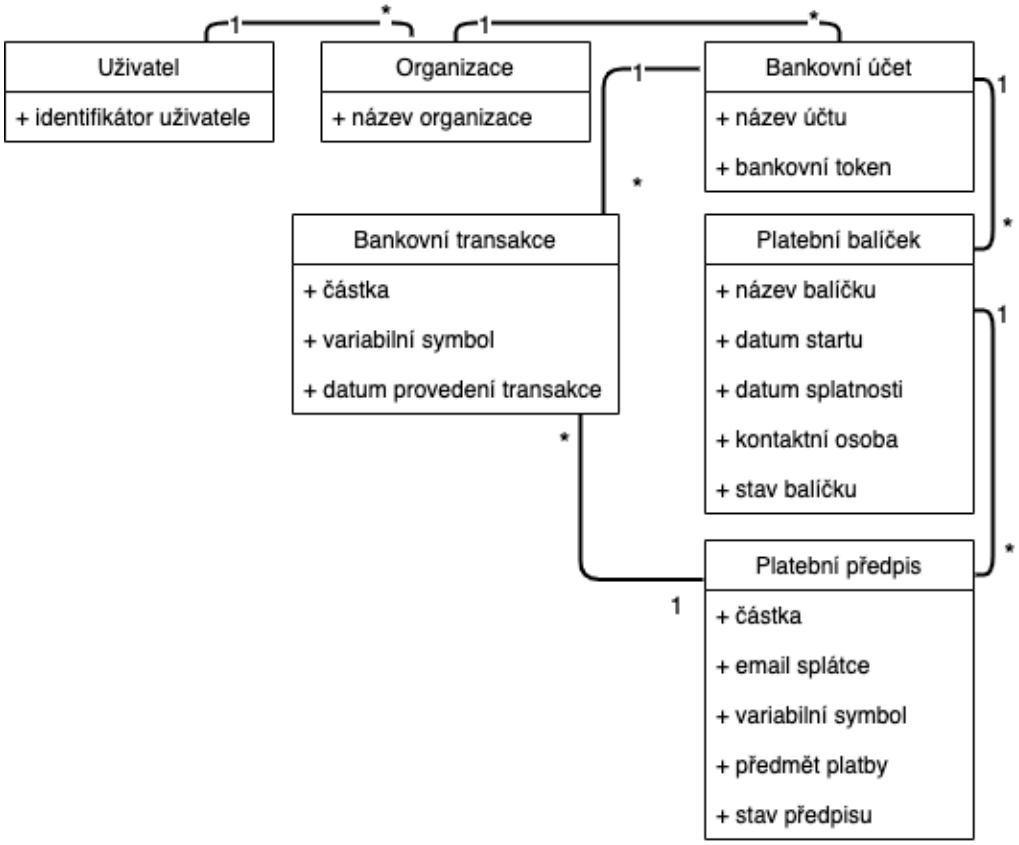
V rámci řešení problému, je třeba rozlišovat různé entity a zjistit jejich vztahy. K tomu může sloužit byznysový model entit, který se nachází na obrázku 2.1.

Organizace. Organizace představuje v systému spolek nebo firmu, například si pod tím lze představit školní družinu, zahrádkářský spolek a nebo řemeslný obchod. Jediným atributem organizace je její název.

Bankovní účet. Jedná se o bankovní účet organizace, se kterým bude systém spojený a ve kterém budou sledovány příchozí platby. Účet je definovaný jménem a bankovním tokenem, který si uživatel vygeneruje u své banky[5]. Systém si díky tomuto tokenu dokáže sám zjistit číslo bankovního účtu, se kterým je token spojený.

Platební předpis. Platební předpis je možné přirovnat k faktuře. Obsahuje e-mailovou adresu splátce, předmět platby, částku platby a variabilní symbol na základě kterého bude platební předpis spojen s bankovní transakcí.Zároveň je třeba držet nějaký stav předpisu, aby bylo jasné, zdali je zaplacený či nikoli. Platební předpisy jsou pak sdružovány do platebních balíčků. Příklady platebních předpisů můžou být např. měsíční členský poplatek, čtvrtletní poplatek za družinu nebo objednávka služby či produktu.

Platební balíček. Je souborem platebních předpisů, který je svázaný s bankovním účtem organizace. Je v něm definovaný datum splatnosti, datum startu, kontaktní osoba a stav balíčku. Datum splatnosti je termín, po kterém budou plátci nezaplacených platebních předpisů odeslány e-mailové notifikace s urgencí o zaplacení.



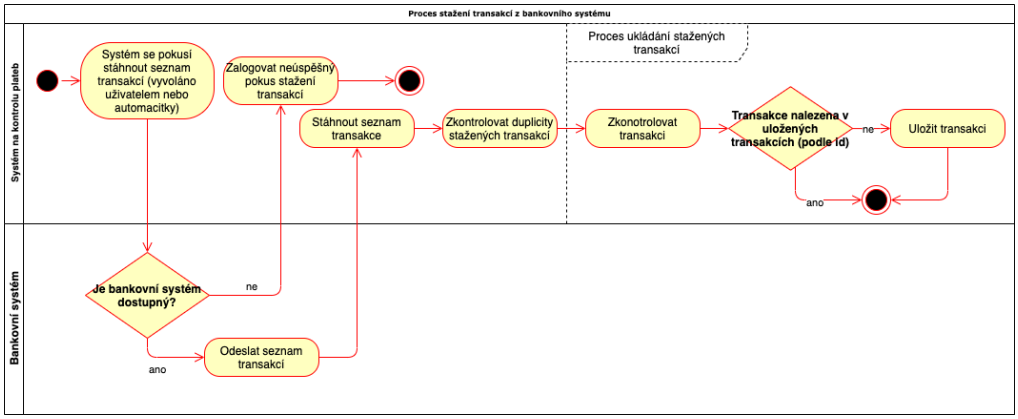
Obrázek 2.1: Byznysový model entit

2.2 Hlavní systémové procesy

Na základě konzultací se zadavatelem bylo definováno několik procesů, které systém musí provádět. Specifikace požadavků byla iterativní. Na obrázcích v následující kapitole jsou finální verze stavových diagramů systémových procesů.

2.2.1 Stažení transakcí z bankovního systému

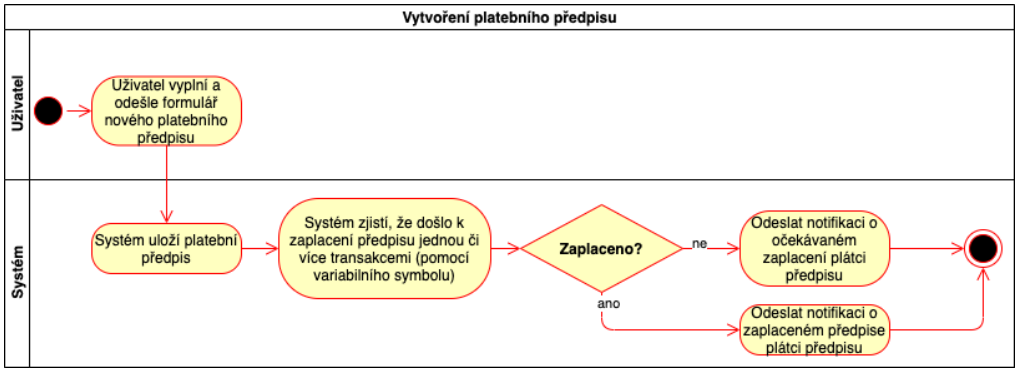
Systém stahuje seznam transakcí (obr. 2.2) z bankovního API. Akce je vyvolávána buď manuálně uživatelem, anebo automaticky jednou za den v 17.00. Stažené transakce jsou následně porovnány s uloženými transakcemi pomocí atributu ID. Když se jedná o novou, ještě neuloženou transakci, systém jí uloží. V opačném případě je transakce odfiltrována. V případě, že transakce neobsahuje variabilní symbol, je automaticky označena jako „nespárovatelná“.



Obrázek 2.2: Procesní diagram stahování transakcí z bankovního systému.

2.2.2 Vytvoření platebního předpisu

Systém umožňuje uživateli vytvářet platební předpisy (obr. 2.3). Po odeslání formuláře k vytvoření platebního předpisu se systém pokusí nově vytvořený platební předpis spárovat s již uloženými transakcemi (viz. 2.2.3). Při úspěšném spárování a zaplacení celého předpisu systém odešle notifikaci o zaplaceném předpise. V opačném případě odešle notifikaci o očekávané platbě, která obsahuje částku, číslo účtu, datum splatnosti a předmět platby.

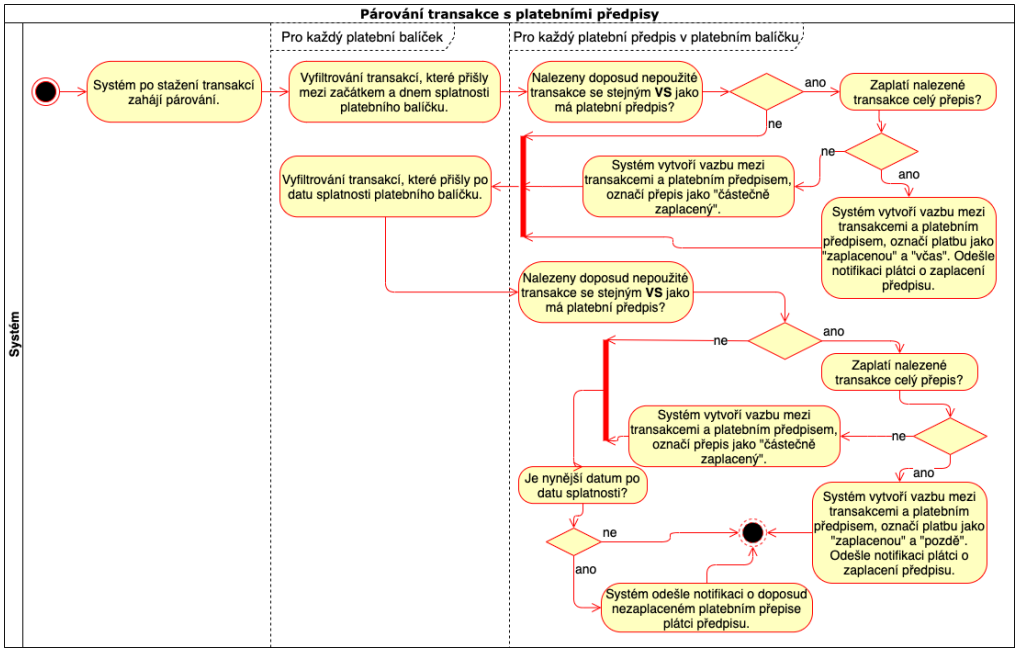


Obrázek 2.3: Procesní diagram vytvoření nového platebního přepisu.

2.2.3 Párování transakcí s platebními předpisy

Systém usnadňuje administraci spojenou s účetnictvím, a to pomocí procesu automatického párování stažených transakcí s platebními předpisy (obr. 2.4). Tento proces se spouští po stažení nových transakcí z bankovního účtu. Systém se pokusí spárovat bankovní transakce s ještě nezaplacenými platebními předpisy z aktivních platebních balíčků. Párování transakce s předpisem se provádí pomocí shodného variabilního symbolu. Nejdříve se provede párování nad transakcemi, které přišly na bankovní účet v časovém intervalu mezi začátkem a dnem splatnosti platebního balíčku. Takto spárované transakce s platebními předpisy jsou v případě plného zaplacení označeny jako

„zaplacené včas“. V případě, že došlo k přeplatku platebního předpisu je platební přepis přeplacený, je označen štítkem jako „přeplacené“. V obou případech je platební předpis vyřazen z předpisů, které se pokusí systém v příštím párovacím procesu spárovat. V případě, že nedojde k úhradě celého platebního předpisu, je označen jako „nedoplacené“. Takovýto předpis není vyřazený z budoucích párovacích procesů. Poté se celý proces spustí znovu, ale mezi párované transakce, budou vybrány transakce, které přišly po datu splatnosti platebního balíčku. Platební předpisy, které zaplacené během tohoto kola párování, budou označeny jako „pozdní“. Každému plátci platebního předpisu, který byl nově označený jako „zaplacené“ nebo „přeplacené“ je odeslána e-emailová notifikace o přijetí platby a zaplacení předpisu. Platební systém umožňuje spárovat několik transakcí s jedním platebním předpisem, tudíž případné nedoplatky by měly být opět automaticky rozpoznány. Systém zároveň každou transakci, kterou spároval s platebním předpisem označí, aby se jí v příštím běhu tohoto procesu nepokoušel znovu spárovat.



Obrázek 2.4: Procesní diagram párování platebních předpisů s bankovními transakcemi.

2.3 Případy užití

Podkapitola Případy užití popisuje chování systému z pohledu uživatele a slouží k bližšímu popsání jednotlivých funkcí. Jedná se o sérii kroků, které vedou ke splnění cíle, přičemž jednotlivé kroky jsou brány jako interakce mezi systémem a aktérem. Případy užití mají i své alternativní scénáře pro situace, kdy dochází k výjimkám.

2.3.1 Registrovat uživatele

Popis: uživatel se registruje do systému
Aktéři: neregistrovaný uživatel
Priorita: must
Hlavní scénář:

1. Uživatel klikne na tlačítko „Login“.
2. Systém zobrazí přihlašovací formulář.
3. Uživatel klikne na tlačítko „Register“.
4. Systém zobrazí registrační formulář.
5. Uživatel vyplní povinná pole a odešle formulář.
6. Systém vyhodnotí registrační formulář a registruje uživatele.
7. Systém uživatele přihlásí a přesměruje ho na domovskou stránku.

2.3.2 Přihlásit uživatele

Popis: uživatel se přihlásí do systému
Aktéři: neregistrovaný uživatel
Priorita: must
Hlavní scénář:

1. Uživatel klikne na tlačítko „Login“.
2. Systém zobrazí přihlašovací formulář.
3. Uživatel vyplní přihlašovací údaje a odešle formulář.
4. Systém zkontroluje přihlašovací údaje, přihlásí uživatele a přesměruje uživatele na domovskou stránku.

Alternativní scénář:

1. Uživatel zadal neplatnou kombinaci uživatelského jména a hesla.
2. Systém uživateli zobrazí chybovou hlášku o chybně vyplněných přihlašovacích údajích.

2.3.3 Odhlásit uživatele

Popis: uživatel se odhlásí ze systému
Aktéři: přihlášený uživatel
Priorita: must
Hlavní scénář:

1. Uživatel klikne na tlačítko „Logout“.
2. Systém uživatele odhlásí ze systému a přesměruje uživatele na domovskou stránku.

2.3.4 Vytvořit organizaci

Popis: uživatel vytvoří organizaci
Aktéři: přihlášený uživatel
Priorita: must
Hlavní scénář:

1. Uživatel na obrazovce se seznamem organizací klikne na tlačítko „+“.
2. Systém zobrazí formulář pro vytvoření organizace.
3. Uživatel formulář vyplní a klikne na tlačíko „Vytvořit“.
4. Systém založí novou organizaci a přesměruje uživatele na seznam bankovních účtů nově založené organizace.

2.3.5 Upravit organizaci

Popis: uživatel vytvoří organizaci
Aktéři: správce organizace
Priorita: must
Hlavní scénář:

1. Uživatel vybere organizaci ze seznamu a klikne na tlačítko „Upravit“.
2. Systém zobrazí formulář s vyplněnými hodnotami upravované organizace.
3. Uživatel změní hodnoty a odešle formulář.
4. Systém zobrazí změnu v seznamu organizací.

2.3.6 Přidat bankovní účet

Popis: uživatel vytvoří organizaci
Aktéři: správce organizace
Priorita: must
Hlavní scénář:

1. Uživatel na obrazovce se seznamem organizací klikne na tlačítko „+“.
2. Systém zobrazí formulář pro založení nového bankovního účtu.
3. Uživatel vyplní formulář.
4. Systém založí nový bankovní účet a přesměruje uživatele na přehled nově založeného bankovního účtu.

2.3.7 Upravit bankovní účet

Popis: uživatel upraví bankovní účet
Aktéři: správce organizace
Priorita: must
Hlavní scénář:

1. Uživatel vybere bankovní účet ze seznamu a klikne na tlačítko „Upravit“.
2. Systém zobrazí formulář upravení bankovního účtu s vyplněnými hodnotami upravovaného bankovního účtu.
3. Uživatel změní hodnoty a odešle formulář.
4. Systém zobrazí změnu v seznamu bankovních účtů.

2.3.8 Změnit bankovní token

Popis: uživatel změní token k bankovnímu účtu
Aktéři: správce organizace
Priorita: could
Hlavní scénář:

1. Uživatel vybere bankovní účet ze seznamu a klikne na tlačítko „Změnit token“.
2. Systém zobrazí formulář pro změnu bankovního tokenu.
3. Uživatel vloží nově vygenerovaný token z bankovního systému.
4. Systém ověří, že token patří ke stejnému bankovnímu účtu, uloží změnu, zavře formulář a zobrazí uživateli zprávu o úspěšné aktualizaci.

Alternativní scénář (krok 3):

1. Uživatel vloží token který je buď neplatný, anebo už je v systému uložený.
2. Systém upozorní uživatele, že nedošlo ke změně tokenu, jelikož se jedná o neplatný token.

2.3.9 Vytvořit platební balíček

Popis: uživatel vytvoří platební balíček
Aktéři: správce organizace
Priorita: must
Hlavní scénář:

1. Uživatel na obrazovce se seznamem platebních balíčků klikne na tlačítko „+“.
2. Systém zobrazí formulář pro založení nového bankovního účtu.
3. Systém založí nový platební balíček a přesměruje uživatele na přehled platebního balíčku.

2.3.10 Upravit platební balíček

Popis: uživatel upraví platební balíček
Aktéři: správce organizace
Priorita: must
Hlavní scénář:

1. Uživatel vybere platební balíček ze seznamu a klikne na tlačítko „Upravit“.
2. Systém zobrazí formulář upravení bankovního účtu s vyplněnými hodnotami upravovaného bankovního účtu.
3. Uživatel změní hodnoty a odešle formulář.
4. Systém zobrazí změnu v seznamu platebních balíčků.

Alternativní scénář (krok 3):

1. Uživatel změnil stav platebního balíčku z „Nový“ na „Aktivní“.
2. Systém pošle e-mailové notifikace o očekávané platbě na všem plátcům zadefinovaným uvnitř platebního balíčku.
3. Systém zobrazí změnu v seznamu platebních balíčků.

2.3.11 Vytvořit platební předpis

Popis: uživatel vytvoří platební balíček
Aktéři: správce organizace
Priorita: must
Hlavní scénář:

1. Uživatel na obrazovce se seznamem platebních předpisů klikne na tlačítko „+“.
2. Systém zobrazí formulář pro založení nového platebního předpisu.
3. Uživatel vyplní formulář.
4. Systém založí nový platební předpis.

Alternativní scénář (krok 4):

1. Uživatel vytváří platební předpis uvnitř „Aktivního“ platebního balíčku.
2. Systém pošle e-mailovou notifikaci plátce založeného platebního předpisu.

2.3.12 Upravit platební předpis

Popis: uživatel upraví platební předpis v dosud neaktivovaném platebním balíčku
Aktéři: správce organizace
Priorita: should
Hlavní scénář:

1. Uživatel vybere platební předpis, který chce upravit a klikne na tlačítko „Upravit“.
2. Systém zobrazí předvyplněný formulář pro úpravu platebního předpisu.
3. Uživatel změní hodnoty a odešle formulář.
4. Systém zobrazí změnu v seznamu platebních předpisů.

2.3.13 Stáhnout platby a spustit párování

Popis: uživatel spustí proces stažení plateb
Aktéři: správce organizace
Priorita: must
Hlavní scénář:

1. Uživatel na obrazovce s bankovními transakcemi klikne na tlačítko „Synchronizovat platby“.
2. Systém zahájí proces stažení bankovních transakcí.
3. Systém po stažení transakcí zobrazí uživateli obnovený seznam bankovních transakcí.

2.3.14 Zobrazit detail transakce

Popis: uživatel zobrazí detail bankovní transakce
Aktéři: správce organizace
Priorita: should
Hlavní scénář:

1. Uživatel na obrazovce s bankovními transakcemi klikne na libovolnou transakci.
2. Systém uživateli zobrazí okno s detailem bankovní transakce.

2.3.15 Vytvořit hromadně platební předpisy z CSV

Popis: uživatel nahraje soubor s předdefinovanými platebními předpisy do platebního balíčku
Aktéři: správce organizace
Priorita: could
Hlavní scénář:

1. Uživatel na obrazovce se seznamem platebních předpisů klikne na tlačítko „Naimportovat z CSV“.
2. Systém vyzve uživatele k vložení souboru v platném formátu (nabídne uživateli šablonu).
3. Uživatel nahraje soubor a systém vytvoří platební předpisy (proces vytvoření kap. 2.2.2).

Alternativní scénář (krok 3):

1. Uživatel vloží neplatné CSV.
2. Systém zobrazí chybu, import neprovede a uživateli opět nabídne CSV šablonu.

2.3.16 Manuálně spárovat transakci s platebním předpisem

Popis: uživatel páruje transakci s platebním předpisem
Aktéři: správce organizace
Priorita: could
Hlavní scénář:

1. Uživatel na obrazovce se seznamem bankovních transakcí vybere transakci a klikne na tlačítko „Spárovat“.
2. Systém zobrazí modální okno, ve kterém uživateli nabídne platební předpisy již aktivních či uzavřených platebních balíčků.
3. Uživatel vybere platební předpis, se kterým chce transakci spárovat.
4. Systém vyhodnotí zaplacení platebního předpisu a nastaví předpisu příslušný stav.

2.3.17 Zkopírovat platební balíček

Popis: uživatel zkopíruje platební balíček
Aktéři: správce organizace
Priorita: could
Hlavní scénář:

1. Uživatel na obrazovce se seznamem platebních balíčků zvolí balíček, který chce zkopírovat a klikne na tlačítko „Vytvořit kopii“.
2. Systém vytvoří kopii platebního balíčku včetně platebních předpisů.
3. Systém nastaví platebnímu balíčku stav „Vytvořen“ a platební předpisy uvnitř balíčku budou mít nastavený výchozí stav „Nezaplaceno“.

2.3.18 Exportovat platební balíček

Popis: uživatel vyexportuje platební předpisy z platebního balíčku
Aktéři: správce organizace
Priorita: could
Hlavní scénář:

1. Uživatel zvolí platební balíček a klikne na tlačítko „Exportovat balíček“.
2. Systém vytvoří tabulkový soubor obsahující platební předpisy zvoleného balíčku, včetně jejich nynějšího stavu. Soubor se uživateli automaticky stáhne jako příloha.

2.3.19 Nastavit uživatelská práva v organizaci

Popis: uživatel nastaví práva v organizaci pro uživatele
Aktéři: správce organizace
Priorita: could
Hlavní scénář:

1. Uživatel vybere organizaci ze seznamu a klikne na tlačítko „Nastavit práva“.
2. Systém zobrazí obrazovku s právy organizace.
3. Uživatel může přidat nové/odebrat stávající uživatele.
4. Systém přidá/odebere uživatele do organizace a zobrazí změnu.

2.3.20 Nastavení penále pozdně zaplacených plateb

Popis: uživatel nastaví penále za opožděné platby
Aktéři: správce organizace
Priorita: could
Hlavní scénář:

1. Uživatel vybere platební balíček ze seznamu a klikne na tlačítko „Konfigurovat penále“.
2. Systém zobrazí obrazovku s konfigurací penále pro platební balíček.
3. Uživatel vyplní hodnoty pro koeficient penále, periodu po které je koeficient aplikován a zaškrtne checkbox „Aplikovat penále“.Uživatel odešle formulář.
4. Systém zapíše změny a začne vypočítávat penále pro pozdě zaplacené platby.

2.4 Bankovní API

V rámci analýzy bylo třeba prověřit, zda bankovní API Fio Banka a.s.[6] dokáže vracet potřebné údaje pro fungování systému. Nejdůležitější funkcí je, aby API vracelo informace o bankovním účtu a následně dokázalo vypsat pohyb na bankovním účtu. Fio Banka nabízí dva způsoby přístupu k jejich API. V obou případech je pro stažení dat o bankovním účtu třeba, aby byl použit token vygenerovaný oprávněnou osobou s přístupovými právy k bankovnímu účtu. Samotný token se generuje v internetovém bankovnictví[5].

První z možností přístupu do bankovního systému je PSD2 API[7], splňující český Open Banking standard (COBS)[8], vydávaný Českou bankovní asociací (ČBA). Tento standard vznikl na základě Zákona č. 370/2017 Sb.[9], který uděluje, mimo jiné, povinnost poskytnout přístup třetím stranám do bankovních systému. Výhodou standardu je, že všechny banky, které ho implementují mají následně stejné API, což znamená jednodušší integraci s bankovními systémy jiných bank než je Fio Banka. Pro možnost využití tohoto API je třeba získat certifikát od společnosti První certifikační autorita, a.s. (I.CA)[10]. Samotné API poskytuje všechny funkce, které jsou třeba k implementaci systému.

Druhou z možností je použít Fio API Bankovnictví[11]. K přístupu k tomuto API není nutné získat certifikát od I.CA, ČNB ani jiné schvalování ze strany banky. API vrací data v různých formátech, z nichž nejdůležitější jsou XML a JSON, které patří mezi nejpoužívanější. Toto API rovněž nabízí všechny potřebné funkce.

```
<AccountStatement>
    <Info>
        <accountId>2111111111</accountId>
        <bankId>2010</bankId>
        <currency>CZK</currency>
        <iban>CZ7920100000002111111111</iban>
        <bic>FIOBCZPPXXX</bic>
        <openingBalance>7356.22</openingBalance>
        <closingBalance>7321.22</closingBalance>
        <dateStart>2012-07-01+02:00</dateStart>
        <dateEnd>2012-07-31+02:00</dateEnd>
        <idFrom>1147608196</idFrom>
        <idTo>1147608197</idTo>
    </Info>
    <TransactionList>
        <Transaction>
            <column_22 id="22" name="ID pohybu">1147608196</
                column_22>
            <column_0 id="0" name="Datum">2012-07-27+02:00</
                column_0>
            <column_1 id="1" name="Objem">-15.00</column_1>
            <column_14 id="14" name="Měna">CZK</column_14>
            <column_2 id="2" name="Protiúčet">2222233333</
                column_2>
            <column_3 id="3" name="Kód banky">2010</column_3>
            <column_12 id="12" name="Název banky">Fio banka, a.s
                .</column_12>
```

```
24          <column_7 id="7" name="Uživatelská identifikace"> </
                column_7>
25          <column_8 id="8" name="Typ">Platba převodem uvnitř
                banky</column_8>
26          <column_9 id="9" name="Provedl">Novák, Jan</column_9
                >
27          <column_25 id="25" name="Komentář">Můj test</
                column_25>
28       </Transaction>
29       ...
30    </TransactionList>
31 </AccountStatement>
```

Listing 2.1: Ukázka XML vráceného z Fio API Bankovnictví

```
1  {
2     "accountStatement":{
3        "info":{
4           "accountId":"2400222222",
5           "bankId":"2010",
6           "currency":"CZK",
7           "iban":"CZ7920100000002400222222",
8           "bic":"FIOBCZPPXXX",
9           "openingBalance":195.00,
10          "closingBalance":195.01,
11          "dateStart":1340661600000,
12          "dateEnd":1341007200000,
13          "idFrom":1148734530,
14          "idTo":1149190193,
15       },
16       "transactionList":{
17          "transaction":[
18             {
19                "column22":{
20                   "value":1148734530,
21                   "name":"ID pohybu",
22                   "id":22
23                },
24                "column0":{
25                   "value":1340661600000,
26                   "name":"Datum",
27                   "id":0
28                },
29                "column1":{
30                   "value":1.00,
31                   "name":"Objem",
32                   "id":1
33                },
34                "column14":{
35                   "value":"CZK",
36                   "name":"Měna",
37                   "id":14
38                },
39                "column2":{
40                   "value":"2900233333",
41                   "name":"Protiúčet",
42                   "id":2
```

```
43                    },
44                    "column3":{
45                        "value":"2010",
46                        "name":"Kód banky",
47                        "id":3
48                    },
49                    "column12":{
50                        "value":"Fio banka, a.s.",
51                        "name":"Název banky",
52                        "id":12
53                    },
54                    ...
55                }
56            ]
57        }
58    }
59 }
```

Listing 2.2: Ukázka JSON vráceného z Fio API Bankovnictví

2.5 Vracení nespárovaných plateb

Pro vracení nespárovaných plateb, by uživatelé do systému museli přidávat bankovní tokeny, které mají práva na zadávání plateb. Takto zadané platby musejí být dodatečně autorizovány pomocí SMS nebo Fio podpisem[11]. Proto lze proces automatizovat jen částečně , což není dostačující. Zároveň s tím by mohlo docházet k nechtěnému spamu vlastníka účtu přes SMS či jiné metody. Držení tokenů, které mají práva na zadávání platebních příkazů, s sebou nese zvýšené riziko v případě, že by došlo k úniku dat. Kvůli těmto skutečnostem byla funkce vracení nespárovaných plateb označena vedoucím práce za nesmyslnou.

2.6 Výpočet penále opožděných plateb

Výpočet penále lze provést pomocí koeficientu a periody přičítání koeficientu. Původní částka je vynásobena koeficientem a výsledek je pravidelně přičítán k dlužné částce každých x dní, kde x je perioda přičítání koeficientu.

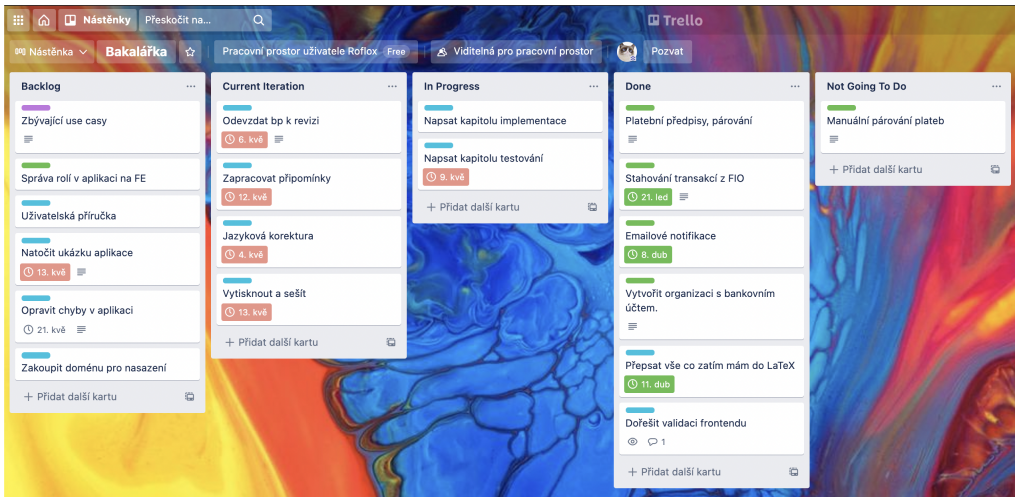
2.7 Práce na projektu

S vedoucím bakalářské práce jsme se shodli na postupu práce pomocí iterací. Každá iterace byla jeden týden dlouhá a byla ukončena nehledě na stav úkolů obsažených v iteraci. Každá iterace byla ukončená konzultací, při které byl představen aktuální stav projektu a plán na další iteraci. Zároveň jsem vždy dostal zpětnou vazbu ke změnám v aplikaci, a na základě toho byly případně změněny samotné úkoly na následující iteraci.

K organizaci práce na projektu jsem používal aplikaci Trello[12] (obr. 2.5), která umožňuje vytvářet jednotlivé úkoly a vizuálně je zobrazit na

Kanban tabuli. Kanban tabule se používá k rozdělení úkolů do sloupců, které reprezentují jejich aktuální stav. V mém případě, jsem měl úkoly rozdělené do těchto sloupců:

- Backlog - seznam všech úkolů, na kterých ještě nebyla provedena žádná práce
- Current iteration - úkoly, které se budou realizovat v nynější iteraci
- In Progress - úkoly, na kterých se aktuálně pracuje
- Done - hotové úkoly
- Not Going To Do - úkoly, které nebudou realizovány



Obrázek 2.5: Kanban tabule v Trello

Kapitola 3
Návrh systému

V této kapitole se věnuji návrhu samotného systému. Představím technologie, které byly zvažovány, a zároveň vysvětlím samotnou architekturu systému a problémy, které bylo třeba ještě před samotnou implementací řešit.

3.1 Zvažované technologie

Ze zadání vyplývá, že systém má být postavený na cloud native technologiích, za použití mikroservisní architektury. Toto zadání značně omezuje technologie, které byly zvažovány během analytické fáze projektu.

3.1.1 Platforma

Vzhledem k tomu, že při využívaní cloud native přístupu je kladen důraz na to, kde jsou aplikace nasazeny, bude samotný systém nasazený v některém z cloudový prostředí. Na vybírat lze z více možností, mezi nejznámější se řadí Microsoft Azure[13], Amazon Web Services (AWS)[14], Google Cloud[15]. V rámci tohoto projektu mi bylo nabídnuto také prostředí CodeNOW[16]. V následujících kapitolách nastíním klíčové vlastnosti zmiňovaných prostředí. Všechna vypsaná prostředí nabízí hostování služeb běžících v kontejnerech za pomocí nástroje Docker či případně orchestračního nástroje Kubernetes. Hlavní rozdíl mezi zmiňovanými platformami je cena. Jako výslednou možnost jsem vybral platformu CodeNOW, jelikož mi byla zpřístupněna bezplatně mým školitelem.

CodeNOW

Platforma je softwarovou továrnou umožňující celý životní cyklus vývoje aplikace od jejího sestavení, testovaní, sledování výstupů (logů) až po její nasazení. Samotná platforma při tvorbě aplikace umožňuje vytváření komponent. Komponenty jsou pak v naší terminologii ekvivalentem mikroslužeb. Technologie, které platforma CodeNOW nabízela při analýze technologií, jsou Java/Spring Boot, Java/Micronaut a JavaScript/React[17].

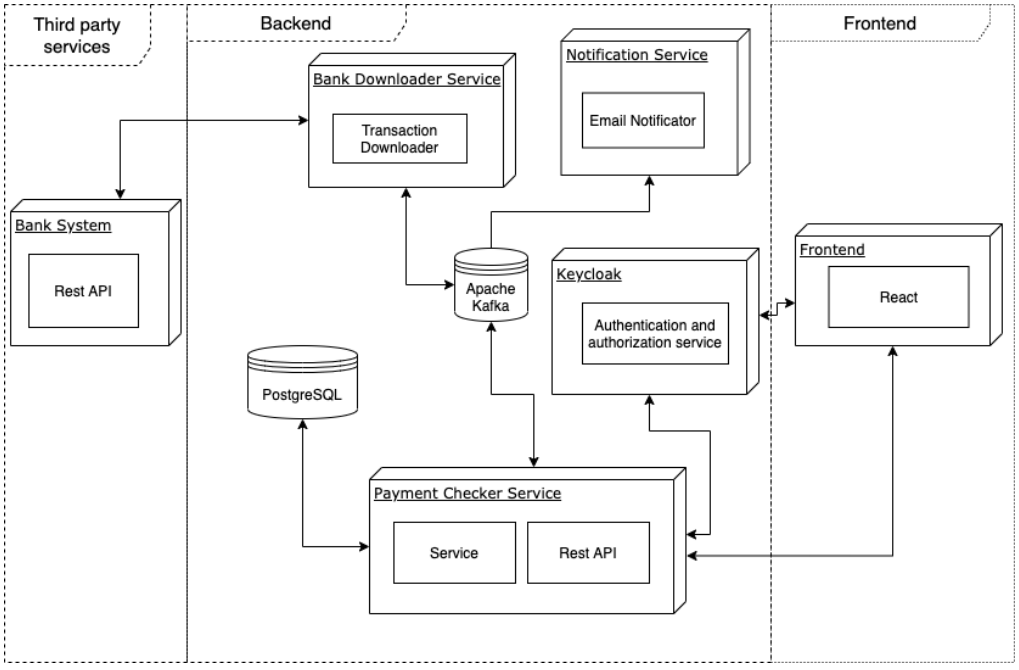
Po vytvoření komponenty je vytvořen i GitLab repozitář, ve kterém jsou uložené zdrojové kódy a konfigurační soubory mikroslužby.

CodeNOW nabízí možnost vytvářet instance systémů jako je Apache Kafka, CocroachDB, Keycloak, PostgreSQL, RabbitMQ a Redis. Tyto systémy lze následně administrovat přímo z uživatelského rozhraní, a následně je spojit se samotnými aplikacemi.

3.2 Architektura

Požadavkem zadání je, aby systém byl postavený na mikroservisní architektuře. Mikroservisní architektura je jednou z možností, jak se v dnešní době vyvíjejí rozsáhlé systémy. Jedním z důvodů, proč jsou mikroslužby na vzestupu, je jejich modulárnost. Tím, že jsou jednotlivé části systému od sebe odděleny, je možné mikroslužby měnit, a v případě správného postupu i bez dopadu na zbytek systému. Dalším z důvodů je výborná škálovatelnost. V případě, že jedna část systému nezvládá zpracovávat požadavky, které řeší, je možné vytvořit instanci téže mikroslužby a připojit jí do zbytku systému, a tím rozložit požadavky mezi více instancí téže mikroslužby[18].

Samotný návrh systému jsem vypracoval na základě konzultací s odborníky z firmy Stratox. Systém je rozdělen na dvě hlavní části: frontend a backend (obr. 3.1).



Obrázek 3.1: Diagram architektury systému

3.2.1 Frontend

Frontend je součástí prezentační vrstvy aplikace, která je spuštěna převážně na straně klienta. Prezentační vrstva se stará o zobrazení uživatelského rozhraní (GUI), zpracování událostí, vyvolaných interakcí uživatele s GUI, a

o komunikaci se serverovou částí aplikace. Mezi komunikaci se řadí odesílání dat zadaných uživatelem a následné přijetí a zpracování dat odeslaných ze serveru.

První možností, kterou jsem při návrhu systému zvažoval, bylo tzv. Server Side Rendering. Jde o techniku, která se v dnešní době přestává používat. Při načtení webové stránky si klient (prohlížeč) vyžádá celou webovou stránku, která již obsahuje veškerá data a logiku. V případě, že se přesunete na jinou část webu, si Váš prohlížeč opět vyžádá novou webovou stránku, a to i v situaci, že se z webu např. odebere jen jediný odkaz. Důsledkem přílišného vytížení serveru může být pak systém zpomalený.

Druhou možností je použít např. knihovnu React[19]. React je nástroj pro tvorbu Single Page Aplikací (SPA). Tento typ aplikace si při prvním načtení stránky stáhne celou aplikaci, a následně pak přepisuje aktuální vzhled stránky s pomocí dat z backendu[20]. Klient pak při procházení stránek v aplikaci nenačítá celou webovou stránku znovu[21], ale aplikace přepisuje pouze změny, a tím je značná část výpočetního výkonu přenesena ze serveru na klienta.

Jako hlavní frontendovou technologii jsem vybral knihovnu React, která je spojená s programovacím jazykem JavaScript potažmo TypeScript. Důvodem tohoto výběru je předchozí zkušenost s technologií a současně vysoká popularita dané technologie v posledních letech. Díky tomu má nyní JavaScript velkou komunitu, ve které se v případě jakýkoliv problémů mohu poradit o řešení. Zároveň existuje velká řada jiných knihoven, které mohu použít, a nebudu muset vše programovat sám.

3.2.2 Backend

Backend obstarává hlavní část logiky celého systému. Jeho úkolem je validovat a zpracovat data přicházející přes vystavené REST API[22]. Zpracovaná data pak následně ukládá do databáze. Tato část systému bude dle 3.2 rozdělena na tři mikroslužby. Mikroslužby mezi sebou musí komunikovat, aby si byly schopné předávat potřebná data. Jedno z řešení, které připadá v úvahu, je komunikace pomocí REST API, tento způsob by byl jistě funkční, ale byl by špatně škálovatelný a měl by určité implementační problémy. Jedním z těchto problémů by bylo vymyšlení a následná implementace komunikačního protokolu. Jelikož by při komunikaci nebyla zaručená žádná forma uložení těchto zpráv, které si mezi sebou mikroslužby posílají. V případě, že by jedna ze služeb byla nedostupná, docházelo by ke ztrátě požadavků, které nebyly doručeny. Z tohoto důvodu bylo určeno, že při komunikaci mikroslužeb bude použita služba Apache Kafka.

Notifikační mikroslužba

Tato část systému (viz. obr. 3.1 Notification Service) bude sloužit k upozorňování uživatelů. Tato mikroslužba bude připojená k SMTP serveru a bude odesílat notifikace plátcům platebních předpisů v případě, kdy se vytvoří nový platební předpis, mají zpoždění se splacením platebního předpisu, anebo v případě, že systém zaeviduje úhradu platebního předpisu. Bude přijímat

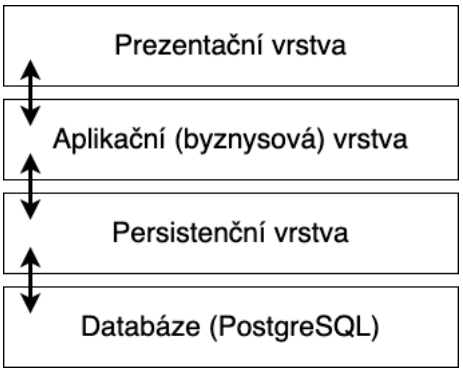
informace od mirkoslužby na párování plateb. Jmenovitě emailovou šablonu, kterou má využít a data, kterými má šablonu naplnit.

Mikroslužba stahující transakce z banky

Služba (viz. obr. 3.1 Bank Downloader Service), která v systému zajišťuje middleware mezi bankovním API a mikroslužbou na párování plateb. Od párovací mikroslužby dostává pouze token k bankovnímu účtu, a pak už jen stahuje výpis z bankovního účtu, přičemž automaticky odfiltruje veškeré odchozí transakce z bankovního účtu. V případě, že by přibyla podpora i jiné banky než Fio, bylo by to realizováno vytvořením nové mikroslužby, která by fungovala z byznysového hlediska stejně jako tato služba.

Mikroslužba na párování plateb

Hlavní část systému je mikroslužba na párování plateb (viz. obr. 3.1 Payment Checker Service). Je to jediná mikroslužba, která ukládá nějaká data, a tudíž i jediná služba, která potřebuje být připojená k databázi. Je to též ta část systému, která pomocí REST API komunikuje s klientem. Architektura samotné mikroslužby se nazývá vícevrstvá architektura[23](obr. 3.2). První vrstvou mikroslužby je vrstva persistentní, starající se o ukládání a přístup k datům systému. Druhou vrstvou je aplikační vrstva, obstarávající veškerou logiku systému jako je např. párování plateb, komunikace s ostatními mikroslužbami, anebo zpracování dat. Tato vrstva je též zodpovědná za pravidelné systémové úlohy jako je např. kontrola pozdních plátců platebních předpisů. Poslední vrstvou je prezentační vrstva, která vystavuje již dříve zmiňované REST API, které využívá frontend aplikace.



Obrázek 3.2: Vrstvená architektura

3.2.3 Služba správy uživatelů a správy přístupu

Jako autentizační a autorizační službu jsem vybral službu Keycloak firmy JBoss. Nabízí možnost správy uživatelů a řízení přístupu a jejich rolí. Implementuje standard OAuth[24]. V serverové části je možné pomocí této služby určovat práva jednotlivých uživatelů, a pro klientskou stranu slouží

služba jako autentizační server, přes který se uživatelé přihlašují, registrují a případně upravují svůj uživatelský účet. Služba jako autentizační metodu používá technologii *json web token* dále jen JWT[25].

3.2.4 Databáze

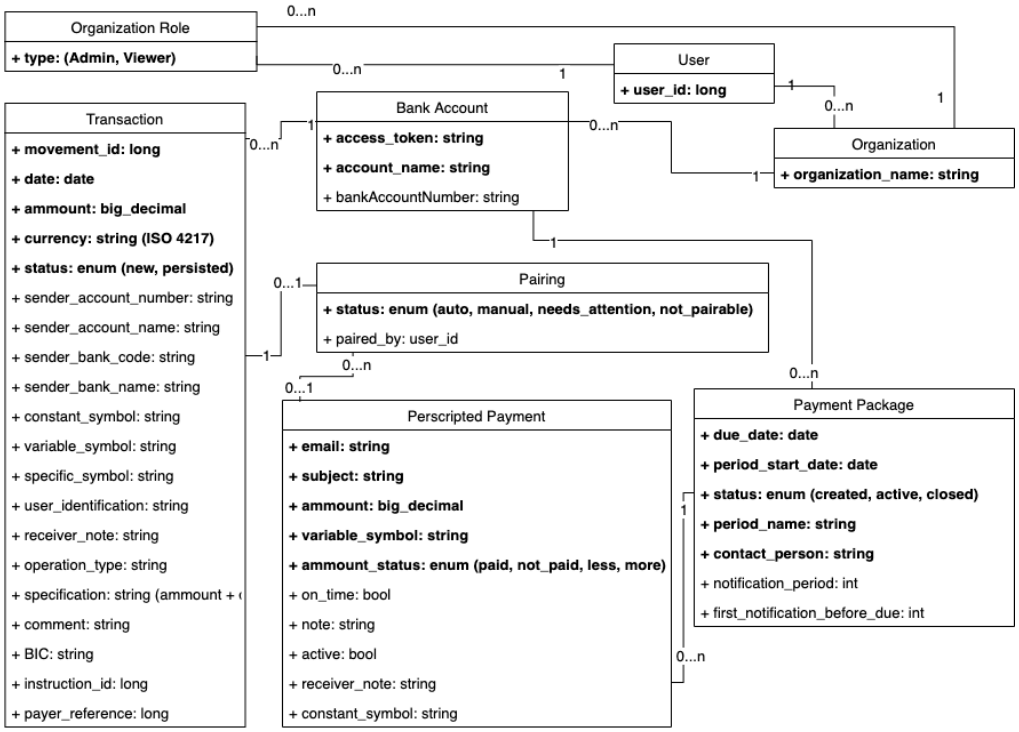
Mikroslužba na párování plateb (kap. 3.2.2) využívá databázi k ukládání dat. Platforma CodeNOW podporuje SQL databázi RabbitMQ a PostgreSQL. Jako databázový systém byl zvolen PostgreSQL[26]. Jedná se o opensource projekt, který je využívám velkou řadou firem jako např Uber, Netflix, Instagram, Spotify a další[27].

3.2.5 Apache Kafka

Pro zjednodušení komunikace mezi mikroslužbami byla využita služba Apache Kafka. Kafka umožňuje zprávy jednotlivých mikroslužeb ukládat do topiků (témat)[28] jako jednotlivé eventy (události)[29]. U nástroje Kafka se služby dělí na producers (producenty) a consumers (konzumenty). Producenti vytváří do logů nové eventy, které jsou konzumovány konzumenty. V případě, že se konzument připojí k topiku, automaticky si stáhne eventy, které přibyly od doby, kdy byl naposledy připojený.

3.3 Třídy

Jak bylo již popsáno v kapitole 2.1 nyní si představíme zbývající třídy reprezentující byznysovou logiku aplikace. Oproti zmiňované kapitole zde přibyla třída reprezentující párování mezi transakcemi a platebními předpisy. Viz obr. 3.3.



Obrázek 3.3: Diagram tříd reprezentovaných v systému

Kapitola 4
Implementace

Tato kapitola se věnuje popisu implementace samotného systému. Všechny části systému byly vyvíjeny pomocí komerčního editoru Intellij Idea od společnosti JetBrains. Zdrojové kódy každé mikroslužby se nachází ve vlastním Git repozitáři, a samotné sestavení mikroslužeb probíhá přímo v platformě CodeNOW. Samotné mikroslužby se pak spouštějí v kontejnerech[30] opět na platformě CodeNOW. Všechny kontejnery jsou spravovány pomocí nástroje Kubernetes[31].

Kubernetes v tomto případě řeší vytváření nových a ničení již běžících kontejnerů. V případě, že se služba běžící uvnitř kontejneru zasekne nebo je ukončena, Kubernetes vadný kontejner ukončí a vytvoří nový kontejner s novou instancí mikroslužby a připojí jí do zbytku systému.

4.1 Frontend

Frontend aplikace je celý vytvořený v programovacím jazyce Typescript ve verzi 4.1.2 za pomocí knihovny React[19] ve verzi 17.0.1.

4.1.1 TypeScript

TypeScript je opensource projekt od společnosti Microsoft[32], který rozšiřuje programovací jazyk JavaScript. Přináší možnost statického tipování a další atributy jako např. *interface*, které známe např. z Java. Kód samotného TypeScriptu se pak kompiluje přímo do JavaScriptu. Díky TypeScriptu, jsem mohl využívat právě výhody staticky tipovaného jazyka i při vyvíjení webového grafického rozhraní, jako např. našeptávání vývojového prostředí a objevení chyb již při kompilaci. Zároveň při použití TypeScriptu není problém využívat knihovny psané jen pro JavaScript.

4.1.2 Knihovny

Pro vytvoření této části aplikace jsem využil několik knihoven, ze kterých jsem vybral ty nejdůležitější. V následujících podkapitolách krátce popíši k čemu dané knihovny slouží, a jak jsem je použil. Všechny knihovny, které

jsem použil jsou volně dostupné z `https://www.npmjs.com/`. Ke stahování knihoven jsem používal nástroj npm[33].

React

Nejdůležitější využívanou knihovnou, kterou jsem používal při vytváření frontendu, je React. React mi umožnil užití komponent, které jsem si sám vytvořil, případně jsem využil některé z jiných knihoven. Jednou z důležitých částí Reactu, kterou jsem používal byl React Hooks[34], které jsou součástí Reactu od verze 16.8.0. Díky tomu jsem mohl u mnou vytvořených komponent nastavovat jejich vlastní stav, a to bez použití jiných knihoven jako je Redux[35].

Axios

Knihovna Axios je HTTP Client založený na JavaScrpit promise[36]. V prohlížeči je založený na technologii *XMLHttpRequests*, která je v mém případě využívaná ke komunikaci s backendem systému. Knihovna nabízí i možnost uložení autentizačního JWT, který je posílaný spolu s daty při každém volání cílového URL.

React Bootstrap

Tato knihovna[37] mi poskytla možnost používání již předem vytvořených grafických komponent (např. tlačítka). Komponenty obsahují jednotný vzhled a mají již předem vytvořenou část logiky, která je s grafickým prvkem spojená. V mém případě jsem nemusel používat skoro žádné vlastní CSS.

Yup

Yup[38] je knihovna, pomocí které vývojář zadefinuje schéma objektu. Používá se při validaci uživatelských vstupů, které jsou pak následně porovnávány s vytvořeným schématem. Schéma obsahuje sérii pravidel, které musí vstupy splňovat. V případě, že nejsou pravidla splněna, Yup vrací k porušeným pravidlům přednastavené chybové hlášky. Například při nevyplnění povinného pole vrátí hlášku: „Hodnota je povinná.“.

jsonwebtoken

Tato knihovna[39] byla použita k práci se samotném JWT. V JWT lze uchovávat data o uživateli, a proto byla tato knihovna využita na získání těchto dat. Šlo zejména o uživatelské jméno a uživatelský identifikátor UUID.

React Router

Knihovna[40] starající se o vykreslování React komponent v závislosti na URL adrese uvnitř aplikace. Při vytváření routeru se definuje komponenta

Switch, a do ní se už jen definují *Route*, které obsahují pod jakou URL se nacházejí, a jakou komponentu mají vykreslit.

4.1.3 Struktura aplikace

Zdrojové kódy aplikace jsou rozděleny na několik částí, které popíšu v následujících bodech.

- soubor *App.tsx* - hlavní komponenta, obsahující React Router
- soubor *store.tsx* - komponenta pracující s local storage, která ukládá JWT a následně ho poskytuje celé aplikaci
- soubor *types* - obsahuje rozhraní objektů, které jsou předávané mezi komponenty
- soubor *shemas* - obsahuje definice Yup objektů
- adresář *pages* - obsahuje komponenty vykreslované přímo React Routerem
- adresář *components* - obsahuje komponenty, jako modální okna, vlastní tlačítka nebo šablonu tabulek
- adresář *dto* - obsahuje data transafer objects (DTOs), které definují rozhraní objektů, které jsou přijímané a odesílané na backend
- adresář *utils* - adresář obsahující pomocné funkce

4.1.4 Přihlašování

Přihlašovací mechanismus, který jsem implementoval, komunikuje přímo se službou Keycloak. Uživatel je při kliknutí na přihlašovací tlačítko přesměrován na stránku Keycloak, která obsahuje v URL i adresu, na kterou má po úspěšném přihlášení uživatele přesměrovat. Současně s přesměrováním je prohlížeči předán i URL parametr nazvaný „code“. Na přesměrované stránce je pak následně parametr *code* odeslaný zpět na službu Keycloak, která následně vrátí JWT, který se uloží pomocí *store.tsx* a uživatel je přesměrován opět na domovskou stránku.

4.2 Backend

Backend aplikace se dělí na tři hlavní mikroslužby (kap. 3.2.2). Všechny tři mikroslužby jsou naprogramované za pomocí frameworku Spring Boot[41]. Spring Boot jsem zvolil z důvodu předchozích zkušeností s daným frameworkem. V případě použití Micronaut by implementace probíhala obdobně. Tento framework obsahuje zabudovaný web server Apache Tomcat[42]. Sestavování nových verzí mikroslužeb a stahování závislých balíčků probíhá pomocí nástroje Maven[43]. Konfigurace ke každé mikroslužbě se načítá ze souboru *application.yaml*. Každá z nich obsahuje číslo portu aplikace, detaily konfigurace pro popojení k Apache Kafka a další.

4.2.1 Spring a Spring Boot

Spring je soubor knihoven nejčastěji používaný s programovacím jazykem Java. Je zaměřený převážně na výstavbu backendových systémů. Jedná se o nástroj, který je používaný velkými firmami (např. Netflix) a obsahuje nástroje, které usnadňují vývoj velkých i malých systémů. Knihovny obsažené ve Springu jsou prověřené a umožňují využívání tzv. dependency injection. Vnořování závislostí (dependency injection) je návrhový vzor, při kterém jedna komponenta má vytvořené závislosti (dependencies) na jinou komponentu. Díky tomu může závislá komponenta využívat metody vnořené komponenty, aniž by na ní při sestavování musela mít referenci.

Spring aplikace se typicky spouští uvnitř webového serveru Apache Tomcat. Spring Boot pak už při sestavování zdrojových kódů rovnou obsahuje i Apache Tomcat a výsledná sestavená aplikace je pak už jednoduše spustitelná pomocí Java. Zároveň Spring Boot obsahuje i některé již předdefinované komponenty Springu, kterými se ale v mém případě nebylo třeba zaobírat.

4.2.2 Apache Kafka

Všechny tři mikroslužby využívají Apache Kafka jako komunikační nástroj. Každá z nich ukládá do svého topiku data, která se snaží předat ostatním mikroslužbám. Knihovna, která je pro práci s Apache Kafka, se nazývá *spring-kafka*.

Kafka Producenti ukládají data do Kafky jako serializované JSON objekty pomocí *JsonSerializer*. Aby mikroslužba mohla být připojená ke Kafce, jako consumer, bylo třeba vytvořit konfiguraci, která říká jakým způsobem mají deserializovat data z Kafky. Pro deserializaci dat z Kafky jsem použil *JsonDeserializer*.

4.2.3 Notifikační mikroslužba

Notifikační mikroslužba je implementovaná jako služba využívající SMTP[44] server. Služba se při zapnutí připojí k SMTP serveru a Apache Kafka pouze jako konzument. Z Kafka načítá data, která musejí obsahovat předmět emailu, typ emailové notifikace (první notifikace, upomínka před datem, upomínka pozdní, přijatá platba) a kontaktní osoba pro případ, že by uživatel chtěl někoho kontaktovat. Aby nebyly emailové šablony uloženy přímo ve zdrojovém kódu aplikace, byla pro tento účel použita knihovna Thymeleaf[45]. Thymeleaf je Java šablonový systém využívaný na vykreslování (render) HTML dokumentů. Sytém na základě typu emailové notifikace načte pomocí Thymeleaf jednu ze čtyř HTML šablon. Služba následně vyplní HTML šablonu daty přijatými z DTO a vloží je do MimeMultipart zprávy, kterou pak pomocí SMTP serveru odešle adresátovi. Mikroslužba dále už nic neukládá.

Konfigurace mail serveru se načítá z konfiguračního souboru *application.yaml*.

4.2.4 Mikroslužba stahující transakce z banky

Tato mikroslužba slouží ke stahování a zpracovávání seznamu transakcí a informací o bankovním účtu. Služba při požadavku na stažení informací o bankovním účtu zavolá API banky. API je volané pomocí knihovny *spring-boot-starter-webflux*. Požadavek na bankovní systém je vytvářen přes třídu *RestTemplate* z právě zmiňované knihovny. Tato třída dokáže přijatá data deserializovat do objektu definovaného pomocí třídy. Při vyvolání požadavku na bankovní API se volá URL ve tvaru *https://www.fio.cz/ib_api/rest/periods/{bankovni-token}/{datum-od}/{datum-do}/transactions.xml*. Data stahuje ve formátu XML, jelikož JSON formát, který vrací Fio API[11] by byl složitější na deserializaci (listing 2.2).

Ve zdrojových kódech této mikroslužby se dále nachází definice třídy, do které jsou data stažená z bankovního API deserializována. Tyto operace jsou prováděny pomocí knihovny *jaxb-api*, která umožňuje u atributů tříd přidávat anotace. Tyto anotace pak pomáhají při deserializaci, kde určují které atributy tříd jsou mapovány na jaké elementy uvnitř staženého XML.

Mikroslužba po úspěšném stažení dat vyfiltruje ze seznamu všechny odchozí transakce. Výsledný seznam pak uloží do Kafky.

4.2.5 Mikroslužba na párování plateb

Hlavní mikroslužba, která se dělí na několik logických vrstev (kap. 3.2.2), je zodpovědná za většinu logiky systému. Dále popíšu důležité části této mikroslužby.

Persistenční vrstva

Tato vrstva je zodpovědná za persistenci dat a následné poskytování dat aplikační vrstvě. K práci s databází se v jazyce Java nejčastěji používá jeden ze dvou způsobů. Prvním z nich je použití JDBC (Java Database Connection), který spočívá v psaní vlastních SQL dotazů a následném spouštění těch dotazů přímo ve zdrojových kódech. Druhým častějším způsobem je použití JPA (Java Persistance API). JPA slouží ke zjednodušení práce s databází. Díky JPA lze vytvořit třídy, které reprezentují jednotlivé databázové tabulky a zacházet s daty jako s normálními Java objekty. JPA se řadí mezi ORM nástroje, nicméně se nejedná o implementaci, ale pouze o specifikaci. Konkrétní impelementací JPA, kterou jsem použil je Hibernate *spring-boot-starter-data-jpa*[46], která je součástí Spring Boot frameworku. Hibernate a většina ostatních JPA implementací používá pro své fungování JDBC.

V rámci Hibernate jsem si vytvořil modelovou třídu do balíčku *cz.pazdera.b6b36pro.paymentcheckerservice.model* pro každou entitu, kterou jsem chtěl ukládat do databáze (obr. 4.1).

Dále jsem si v rámci prezenční vrstvy vytvořil *CrudRepository* pro každou entitu, kterou jsem chtěl ukládat do databáze. *CrudRepository* je součástí *spring-boot-starter-data-jpa*. Na místo vytváření Repozitáře vytvářejí automaticky implementace CRUD operací jako je vytváření (create), čtení (read),

aktualizace (update), smazání (delete) pouze z Java *interface*. Do repozitáře lze přidat další metody pro konkrétnější selektování záznamů z databáze. Takto vytvořené metody, jsou pak automaticky naimplementovány v případě, že jsou správně pojmenované podle jmen atributů modelových tříd.

Hibernate zároveň vytváří tabulky z modelových objektů přímo v databázi, díky čemuž není třeba psát vlastní SQL skripty na vytvoření databázových tabulek.

Aplikační vrstva

Aplikační nebo též servisní vrstva obsahuje veškerou logiku celé služby. Tato služba využívá CRUD repozitáře z persistenční vrstvy k ukládání, zapisování a mazání dat z databáze. Pro každou entitu, se kterou systém pracuje jsem vytvořil service třídu. Nyní rozeberu ty nejdůležitější service třídy, které neobsahují jen CRUD operace.

BankAccountService. Tato třída obsahuje mimo jiné i metodu, která automaticky vyvolává proces synchronizace plateb s bankovním systémem.

EmailNotificationService. Třída, která obsahuje logiku pro zapisování eventu do Kafka, který je pak následně přečtený notifikační mikroslužbou (kap. 4.2.3).

TransactionService. TransactionService obashuje logiku párování bankovních transakcí s platebními předpisy. Zároveň tato třída vyvolává automatický proces párování a řeší ukládání nově stažených transakcí z Kafka do databáze.

Prezentační vrstva

Prezentační vrstva vystavuje REST API celé mikroslužby. API je rozdělené na několik zdrojů (resources). Ve Spring Frameworku se REST API impelemtuje pomocí kontrolerů (Controllers). Při implementaci jsem vytvořil následující kontrolery: *BankAccountController*, *OrganizationController*, *PaymentPackageController*, *PrescribedPaymentController* a *TransactionController*. Metody zmiňovaných tříd jsou pak mapovány pomocí anotací z balíčku *spring-boot-starter-webflux* na příslušné HTTP metody jako je POST, GET, PUT a DELETE. Tyto metody jsou zároveň zabezpečeny pomocí knihovny *spring-boot-starter-security* tak, aby volání REST metody bylo umožněno jen přihlášeným uživatelům.

Součástí této vrstvy je i *RestExceptionHandler*, který slouží k zachycení vyvolaných výjimek (exceptions) při běhu kódu. Při zachycení očekávané výjimky pak dokáže handler automaticky vracet stavové kódy (400 Bad Request, 403 Unauthorized, 404 Not Found atd.) odpovídající typu odchycené výjimky.

Poslední, ale neméně důležitou částí, jsou definice tříd přijímaných na REST endpointech. Třídy jsou oanotovány pomocí anotací z knihovny *validation-api*, díky kterým jsou automaticky ověřeny určité vlastnosti přijatých objektů (např. datové typy, minimální hodnoty, délky textových řetězců atd.).

Spring Security

Součástí mikrosužby je zároveň i Spring Security. Jak bylo řečeno v předchozí kapitole slouží k zabezpečení REST endpointů, které mikroslužba vystavuje. Pomocí této knihovny se tato služba připojuje přímo na Keycloak, který pak řeší autentizaci a autorizaci uživatelů. Aby vše fungovalo, musel jsem vytvořit třídu implementující *KeycloakWebSecurityConfigurerAdapter* z knihovny *keycloak-spring-boot-starter*[24]. Tato třída vyhodnocuje postupně každý požadavek, který přijde na REST endpoint pomocí předem definovaných filtrů, které jsou v základní konfiguraci Spring Bootu. Aby se uživatelský identifikátor dostal i do databáze, je do seznamu filtrů přidán filtr. Tento filtr při nově příchozím požadavku ověří, zdali je uživatelský identifikátor již v databázi, a případně do databáze identifikátor zapíše.

4.3 Shrnutí implementace

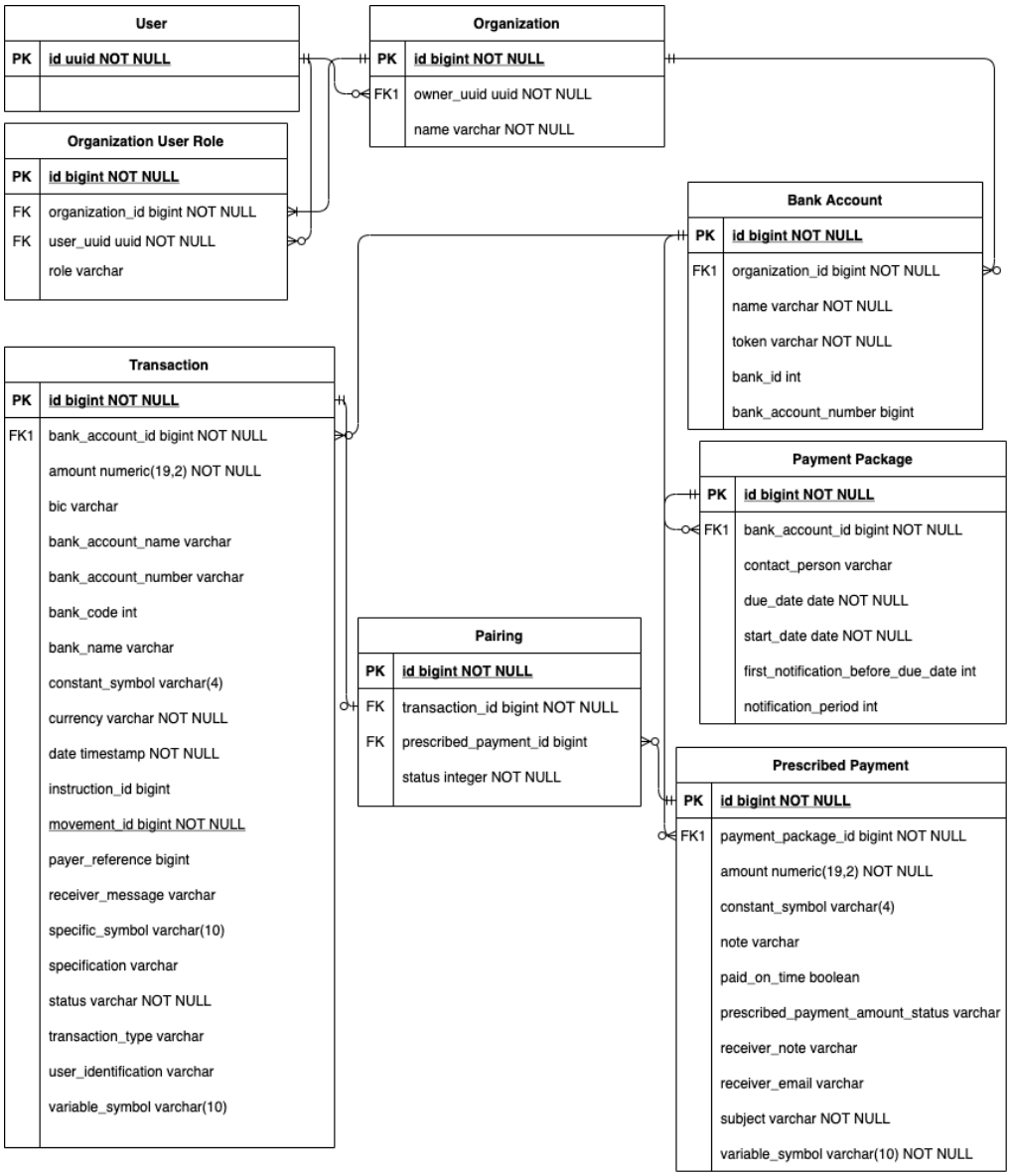
Výsledkem samotné implementace je systém, který je připravený k pilotnímu provozu. Systém je schopný přinášet zákazníkovi určitou hodnotu a zvládá provádět úkony, ke kterým byl navržený.

4.3.1 Frontend

Při implementování této části systému jsem se naučil pracovat s novými technologiemi jako je React, se kterým jsem dříve přišel do styku jen okrajově. Zároveň jsem lépe pochopil sílu SPA. Při implementaci jsem narazil na několik nejasností, jako např. jak aktualizovat seznam v případě, kdy uživatel provede aktualizaci prvku ze seznamu, anebo přidá nový prvek do seznamu. Jelikož jsem s technologií *React Hooks* pracoval poprvé, nebyl jsem si jistý, jestli nevytvářím anti patterny. V rámci omezeného času jsem se problémem více nezabýval a myslím si, že je zde prostor pro zlepšení.

4.3.2 Backend

V této části systému jsem si vyzkoušel práci s message brokerem Apache Kafka, díky kterému jsem byl schopný postavit systém na mikroservisní architekuře. Zároveň jsem si u notifikační mikroslužby vyzkoušel práci s emailovým klientem a s nástrojem Thymeleaf. U mirkoslužby na kontrolu plateb jsem si zároveň nahlédl více do detailu jak funguje Spring Security a zjistil, jak lze připojit vlastní backendové služby ke službě Keycloak. Kdybych měl více času, vyřešil bych zároveň šifrování bankovních tokenů, aby při případném úniku databázových dat, nebyly bankovní tokeny odhaleny v nezašifrované podobě.



Obrázek 4.1: Vrstvená architektura

Kapitola 5
Testování

V následující kapitole představím jakým způsobem byl systém testován.

5.1 Metody testování

S ohledem na to, že byl projekt veden agilním způsobem, byla každá nová verze nasazována přímo do vývojového prostředí na platformě CodeNOW. Toto prostředí je shodné s produkčním prostředím. Testování probíhalo čtyřmi způsoby rozepsanými v následujících podkapitolách.

5.1.1 Smoke testy

Testování funkcionality probíhalo v rámci iterativního vývoje, kdy na konci každé iterace byly prováděny smoke testy. V rámci smoke testů byly vyzkoušeny základní funkce systému a zpětně ověřeny mnou i mým školitelem.

5.1.2 Testování API - Postman

Pomocí nástroje Postman[47] jsem prováděl testování REST API, které vystavuje mikroslužba na párování plateb. V nástroji jsem si vytvořil kolekci, kterou jsem rozdělil přesně podle počtu vystavěných metod samotného API. Do každé metody jsem pak následně zkoušel zadávat jak validní, tak nevalidní vstupní data a testoval jsem vrácené odpovědi a případně opravoval chyby, které se vyskytly při samotném testování. Kolekce viz příloha A.

5.1.3 Jednotkové testy

Jednotkové testování (unit testing) je jedním z nejběžnějších způsobů, jak automaticky otestovat fungování a správnost chování části systému[48]. Běžným nástrojem pro tento druh testování je framework JUnit[49].

5.1.4 UX testování

S ohledem na aktuální kovidová opatření probíhalo UX testování v minimalistické podobě. Aplikace byla předložena specialistce na UX Tereze Doležalové

ze společnosti Unicorn Systems a.s., která aplikaci vyzkoušela a identifikovala nedostatky v designovém návrhu.

Prvním z nedostatků bylo nepoužití UI paternu *Tab* na obrazovce s přehledem platebních balíčků a seznamu transakcí. Dalším nedostatkem bylo nezobrazení přihlašovacího tlačítka na titulní stránce pro nepřihlášeného uživatele. Všechny nedostatky jsem opravil a opravy vydal v následující verzi.

5.2 Testovací scénáře automatických testů

Testování probíhalo vždy kontinuálně při nových verzích systému. Grafický design frontendu byl vždy konzultován se školitelem práce. Zároveň vždy při vytváření nové verze mikroslužby na kontrolu plateb byly spouštěny jednotkové testy. Testy musely být vždy úspěšně dokončeny, jinak by nedošlo k sestavení nové verze mikroslužby.

5.2.1 Scénáře jednotkových testů

Jednotkové testy, které jsem navrhnul pro mikroslužbu na párování plateb byly zaměřeny především na samotný proces párování plateb (viz kap. 2.2.3) s bankovními transakcemi. Tato část systému byla vyhodnocena jako kritická z několika důvodů. Prvním důvodem je, že samotný proces je poměrně složitý a obsahuje velký počet rozhodovacích bloků. Druhým důvodem je, že celý systém byl navržený aby prováděl tuto činnost automaticky, a proto je důležité zaručit, že tato část systému bude fungovat. Třetím důvodem je, že na rozdíl od zbytku mikroslužby, se nejedná jen o CRUD operace.

Jako testovací strategii jsem zvolil metodu, kdy pro předem připravená vstupní data byly kontrolovány finální stavy transakcí a platebních předpisů po dokončení procesu párování. Další možnou strategií, kterou jsem z časových důvodů pouze zvažoval, ale neimplementoval, je testování procesu pomocí procesního testování. Testování tohoto typu slouží k otestování byznys procesů.

5.3 Závěr testování

Během testování sytému byly odhaleny nedostatky především ve frontendu aplikace. Jedná se hlavně o zobrazení aplikace na mobilních zařízeních. Zobrazené grafické prvky se při zobrazení na malém display různě překrývají, anebo nejsou vidět vůbec. Do příštích verzí frontendu je třeba upravit kaskádové styly (CSS) tak, aby bylo zobrazení na mobilních telefonech stejně přehledné jako je v normální desktopové verzi.

Většina uživatelů, kterým byla aplikace předvedena si stěžovala na nemožnost připojení bankovního účtu jiných bank než je jenom Fio Banka a.s. a dále na nemožnost editace platebních příkazů. Zbylé připomínky byly shodné s případy užití s nižší prioritou, které nebyly implementovány. Konkrétně případy užití z kap. 2.3.12, 2.3.15 a 2.3.16.

Kapitola 6

Závěr

Cíl vytvořit cloud native aplikaci pro kontrolu plateb byl splněn. Všechny body ze zadání byly splněny. Vzniklý systém umožňuje uživatelům vytvořit si svojí vlastní organizaci a připojit jí k existujícím bankovním účtům u Fio Banka a.s. Pro každý bankovní účet může majitel organizace vytvářet platební balíčky, a uvnitř nich jednotlivé platební předpisy. Systém pak dokáže samostatně stahovat transakce z bankovního účtu, kontrolovat stavy platebních předpisů a odesílat emailové notifikace s odpovídajícími stavy platebních předpisů. Při založení nového platebního předpisu v již aktivovaném platebním balíčku odesílá notifikaci plátci s informacemi o platbě, která je po něm vyžadovaná. Při zaplacení odesílá notifikaci o přijaté platbě, a v případě, že nepřijde platba na bankovní účet do data splatnosti platby, je uživatel upozorněn připomínkovou notifikací. V backendu systému jsou zároveň připravené funkce pro správu přístupových práv k jednotlivým organizacím.

Zároveň jsem zjistil, že pro vracení nespárovaných plateb je vždy třeba uživatelská autorizace, a tudíž tento proces lze automatizovat pouze částečně, což nedává pro tento projekt smysl.

6.1 Budoucnost systému

V budoucí verzi systému je potřeba vyřešit šifrování přístupových tokenů k bankovním účtům, jelikož momentálně jsou ukládány jako obyčený textový řetězec. Jednou z variant by mohlo být použití asymetrické šifry (např. RSA), která by již při odeslání zašifrovala token pomocí veřejného klíče. Takto zašifrovaný token by byl následně uložen do databáze a byl předáván zašifrovaný do mikroslužby stahující transakce z banky. Tato mikroslužba by pak měla jako jediná privátní klíč, kterým by bankovní tokeny dešifrovala. Toto řešení by stále nebylo ideálním, ale rozhodně by se tak dalo zabránit případným únikům nezašifrovaných tokenů, jelikož klíč od zašifrování by byl na jiném místě než samotný token.

Předmětem dalších prací by bylo upravení systému tak, aby byla více uživatelsky použitelná i na mobilních zařízeních a zařízení přístupu do PSD2 API, díky kterému by pak bylo snadné využívat i bankovní systémy jiné než Fio Banka a.s.

Mezi další případná rozšíření je třeba neopomenout zbývající neimplemen-

tované případy užití, které jsou schopné přinášet cílovým uživatelům další hodnotu. Hlavním kandidátem pro další verzi je správa přístupů do organizací a možnost nastavení koeficientu pro výpočet penále pozdně zaplacených plateb, o který by byla nezaplacená částka vždy v pravidelných intervalech navýšena.

Kvůli časové náročnosti nebyly řádně provedeny uživatelské testy, které by mohly přinést další užitečné informace a zpětnou vazbu k fungování systému. Zároveň by bylo pro budoucí růst aplikace vhodné vytvoření více automatických testů, které by částečně zaručovaly že nové verze systému neohrozí již dříve fungující části kódu. V rámci dalšího testování bych doporučil jeden z nástrojů HotJar a nebo SmartLook, které by poskytovaly informace o interakcích uživatelů se systémem.

Literatura

[1] NEWMAN, Sam. *Building microservices*. Sebastopol, CA: O'Reilly Media, [2015]. ISBN 978-1491950357.

[2] NADAREISHVILI, Nadareishvili, Ronnie MITRA, Matt MCLARTY a Mike AMUNDSEN. *Microservice Architecture: Aligning Principles, Practices, and Culture*. Sebastopol, CA: O'Reilly Media, 2016. ISBN 978-1491956250.

[3] GARRISON, Justin a Kris NOVA. *Cloud Native Infrastructure: Patterns for Scalable Infrastructure and Applications in a Dynamic Environment*. Sebastopol: O'Reilly Media, 2017. ISBN 978-1491984307.

[4] *Understanding cloud-native apps* [cit. 19.05.2021]. Dostupné z: `https://www.redhat.com/en/topics/cloud-native-apps`

[5] fiobanka, 2019, Fio Internetbanking - API bankovnictví, Youtube [online] [cit. 2021-01-09]. Dostupné z: `https://youtu.be/-CwSJUQfowg`

[6] Bankovní služby: API Bankovnictví. Fio Banka [online]. [cit. 2021-01-09]. Dostupné z: `https://www.fio.cz/bankovni-sluzby/api-bankovnictvi`

[7] Otevřené bankovnictví | Fio banka. *PSD2 Vývojářské rozhraní*[online].[cit. 25.02.2021]. Dostupné z: `https://developers.fio.cz/index.html`

[8] Český standard pro Open banking | Česká bankovní asociace. *Úvod | Česká bankovní asociace* [online]. Copyright © [cit. 18.05.2021]. Dostupné z: `https://cbaonline.cz/cesky-standard-pro-open-banking`

[9] *Zákon č. 370/2017 Sb. o platebním styku*. In: ASPI [právní informační systém]. Praha: Wolters Kluwer ČR [vid. 2017-10-11].

[10] *I.CA: Kvalifikované certifikáty pro PSD2* [online]. Praha: První certifikační autorita, - [cit. 2021-3-25]. Dostupné z: `https://www.ica.cz/kvalifikovany-certifikat-pro-psd2`

[11] *FIO API BANKOVNICTVÍ: Verze 1.6.28* [online]. Praha: Fio Banka, 2020, 7. 10. 2020 [cit. 2021-03-23]. Dostupné z: `https://www.fio.cz/docs/cz/API_Bankovnictvi.pdf`

[12] Trello. *About | What is Trello?* [online]. Copyright © 2021 Atlassian [cit. 15.03.2021]. Dostupné z: https://trello.com/about

[13] Microsoft Azure. *Cloudové výpočetní služby*[online]. Copyright © 2021 Microsoft [cit. 15.03.2021]. Dostupné z: `https://azure.microsoft.com/cs-cz/`

[14] Amazon Web Services (AWS) - Cloud Computing Services. *Amazon Web Services (AWS) - Cloud Computing Services* [online]. Copyright © 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved. [cit. 18.4.2021]. Dostupné z: `https://aws.amazon.com/`

[15] Google Cloud. *Cloud Computing Services* [online]. Dostupné z: `https://cloud.google.com/`

[16] *CodeNOW.com*[online]. Prague: Stratox Cloud Native, c2020 [cit. 2021-03-21]. Dostupné z: `https://docs.codenow.com/`

[17] Coding Examples | CodeNOW Documentation. *Hello from CodeNOW Documentation | CodeNOW Documentation* [online]. Copyright © 2021 CodeNOW.com. All rights reserved. [cit. 25.02.2021]. Dostupné z: `https://docs.codenow.com/docs/coding-examples`

[18] DRAGONI, Nicola; LANESE, Ivan; LARSEN, Stephan Thordal. *Microservices: How to make your application scale.* In: [s.l.]: [s.n.], 2017. Dostupné z: `https://hal.inria.fr/hal-01636132/file/microservices-make-application.pdf`

[19] *React: A JavaScript library for building user interfaces* [online]. Merlo Park (California): Facebook, c2021 [cit. 2021-4-28]. Dostupné z: `https://reactjs.org/`

[20] Flanagan, David, *JavaScript - The Definitive Guide*, 5th ed., O'Reilly, Sebastopol, CA, 2006, p.497

[21] *Fixing the Back Button: SPA Behavior using Location Hash.* Falafel Software Blog. Retrieved January 18, 2016.

[22] *What is a REST API?.* [online]. Copyright ©2021 Red Hat, Inc. [cit. 18.05.2021]. Dostupné z: `https://www.redhat.com/en/topics/api/what-is-a-rest-api`

[23] 1. Layered Architecture - Software Architecture Patterns [Book]. *O'Reilly Media - Technology and Business Training* [online]. Copyright © 2021, O [cit. 18.4.2021]. Dostupné z: `https://www.oreilly.com/library/view/software-architecture-patterns/9781491971437/ch01.html`

[24] Keycloak. *Keycloak* [online]. Dostupné z: `https://www.keycloak.org/getting-started`

[25] JSON Web Token Introduction - jwt.io. *JSON Web Tokens - jwt.io* [online]. Dostupné z: `https://jwt.io/introduction`

[26] PostgreSQL: About. *PostgreSQL: The world's most advanced open source database* [online]. Copyright © 1996 [cit. 18.02.2021]. Dostupné z: https://www.postgresql.org/about/

[27] *Which Major Companies Use PostgreSQL? What Do They Use It for? | LearnSQL.com . SQL online courses - learn with us | LearnSQL.com* [online]. Copyright ©2016 [cit. 17.05.2021]. Dostupné z: https://learnsql.com/blog/companies-that-use-postgresql-in-business/

[28] Použití nástroje Apache Kafka v aplikacích založených na mikroslužbách. Root.cz: A distributed streaming platform [online]. [cit. 2021-01-09]. Dostupné z: https://www.root.cz/clanky/pouziti-nastroje-apache-kafka-v-aplikacich-zalozenych-na-mikrosluzbach/k01

[29] INTRODUCTION. APACHE kafka: A distributed streaming platform [online]. [cit. 2021-01-09]. Dostupné z: https://kafka.apache.org/intro

[30] What is a Container? Docker [online]. [cit. 2021-01-09]. Dostupné z: https://www.docker.com/resources/what-container

[31] Co je Kubernetes? Microsoft Azure [online]. [cit. 2021-01-09]. Dostupné z: https://azure.microsoft.com/cs-cz/topic/what-is-kubernetes/

[32] TypeScript: Typed JavaScript at Any Scale.. *TypeScript: Typed JavaScript at Any Scale.* [online]. Copyright © 2012 [cit. 18.4.2021]. Dostupné z: https://www.typescriptlang.org/

[33] npm Docs. *npm Docs* [online]. Dostupné z: https://docs.npmjs.com/

[34] Introducing Hooks – React. *React – A JavaScript library for building user interfaces* [online]. Copyright © 2021 Facebook Inc. [cit. 18.4.2021]. Dostupné z: https://reactjs.org/docs/hooks-intro.html

[35] Getting Started with Redux | Redux. Redux - *A predictable state container for JavaScript apps. | Redux* [online]. Copyright © 2015 [cit. 18.4.2021]. Dostupné z: https://redux.js.org/introduction/getting-started

[36] GitHub - axios/axios: Promise based HTTP client for the browser and node.js. *GitHub: Where the world builds software · GitHub* [online]. Copyright © 2021 GitHub, Inc. [cit. 18.4.2021]. Dostupné z: https://github.com/axios/axios

[37] React-Bootstrap · React-Bootstrap Documentation. *React-Bootstrap · React-Bootstrap Documentation* [online]. Dostupné z: https://react-bootstrap.github.io/getting-started/introduction/

[38] GitHub. *GitHub - jquense/yup: Dead simple Object schema validation. GitHub: Where the world builds software* [online]. Copyright © 2021 GitHub, Inc. [cit. 15.03.2021]. Dostupné z: https://github.com/jquense/yup

[39] npm. *jsonwebtoken* [online]. Dostupné z: https://www.npmjs.com/package/jsonwebtoken

[40] React Router: Declarative Routing for React.js. *React Router: Declarative Routing for React.js* [online]. Dostupné z: https://reactrouter.com/core/guides/philosophy

[41] Spring. *Spring Boot* [online]. Copyright © [cit. 18.4.2021]. Dostupné z: https://spring.io/projects/spring-boot

[42] Apache Tomcat®. *Apache Tomcat® - Welcome!*[online]. Copyright © 1999 [cit. 15.03.2021]. Dostupné z: http://tomcat.apache.org/

[43] Maven – Welcome to Apache Maven. *Maven – Welcome to Apache Maven* [online]. Dostupné z: https://maven.apache.org/

[44] IETF Datatracker. *Simple Mail Transfer Protocol* [online]. Dostupné z: https://datatracker.ietf.org/doc/html/rfc5321

[45] Thymeleaf. *Thymeleaf* [online]. Copyright © The Thymeleaf Team [cit. 18.4.2021]. Dostupné z: https://www.thymeleaf.org/

[46] *Spring Data JPA. Spring | Home* [online]. Copyright © [cit. 24.04.2021]. Dostupné z: https://spring.io/projects/spring-data-jpa

[47] Postman Learning Center. *Home | Postman Learning Center* [online]. Copyright © [cit. 18.4.2021]. Dostupné z: https://learning.postman.com/docs/getting-started/introduction/

[48] Tutorialspoint. *Unit Testing* [online]. Copyright © Copyright 2021. All Rights Reserved. [cit. 18.4.2021]. Dostupné z: https://www.tutorialspoint.com/software_testing_dictionary/unit_testing.htm

[49] JUnit 5. *About* [online]. Copyright © 2021 The JUnit Team [cit. 18.4.2021]. Dostupné z: https://junit.org/junit5/

Příloha A

Přiložené soubory

Odevzdávané soubory:

`frontend.zip` - zdrojové kódy frontendu (kap. 4.1)

`notificator.zip` - zdrojové kódy notifikační mikroslužby (kap.4.2.3)

`bank_downloader_service.zip` - zdrojové kódy mikroslužby na stahování bankovních transakcí (kap. 4.2.4)

`payment_checker_service.zip` - zdrojové kódy mikroslužby na párování plateb (kap. 4.2.5)

`postman_collection.json` - kolekce testů do nástroje Postman (kap. 5.1.2)