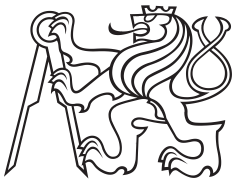
Bakalářská práce



České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická
Katedra počítačů

Mikroservisní architektura v praxi

Jan Bajer

Školitel: Ing. Martin Komárek
Květen 2023

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Bajer** Jméno: **Jan** Osobní číslo: **499353**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Softwarové inženýrství a technologie**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Mikroservisní architektura v praxi

Název bakalářské práce anglicky:

Microservice architecture in practice

Pokyny pro vypracování:

Rozšiřte stávající systém “Kontrola plateb”[1] o nové funkce a implementaci několika vzorů typických pro architekturu mikroslužeb[2].
1. Zprovozněte zadaný systém a nasaďte na Value Stream Delivery Platformu CodeNOW[3].
2. Implementujte architektonický vzor CQRS (Command Query Responsibility Segregation) a s ním spojený vzor Event Sourcing.
3. Rozšiřte systém o API Gateway.
4. Přidejte možnost upomínání opožděných plateb přes SMS.
5. Přidejte možnost placení přes platební bránu.
Práci provádějte iterativním způsobem, průběžně testujte, nasazujte a dokumentujte. Testování provádějte primárně pomocí End-to-End testů.

Seznam doporučené literatury:

[1] Pazdera Jiří. Cloud native aplikace pro kontrolu plateb. B.S. thesis, České vysoké učení technické v Praze. Výpočetní a informační centrum., 2021. Dostupné z: https://dspace.cvut.cz/handle/10467/94732.
[2] Chris Richardson. Microservices patterns: with examples in Java. Manning Publications, 2018. Dostupné z: https://learning.oreilly.com/library/view/microservices-patterns/9781617294549/.
[3] CodeNOW. CodeNOW Documentation, 2022. Dostupné z: https://docs.codenow.com/.

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Martin Komárek kabinet výuky informatiky FEL

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

Datum zadání bakalářské práce: **13.02.2023** Termín odevzdání bakalářské práce: **26.05.2023**

Platnost zadání bakalářské práce: **22.09.2024**

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

Chtěl bych tímto poděkovat svému vedoucímu Ing. Martinu Komárkovi za cenné rady a čas, který mi věnoval při vedení této bakalářské práce. Dále bych chtěl poděkovat své rodině a přátelům, kteří mě podporovali během mého studia.

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, 26. května 2023

Abstrakt

Tato bakalářská práce se zaměřuje na rozšíření existujícího systému "Kontrola plateb" o nové funkce a implementaci vzorů typických pro mikroservisní architekturu. Popisuje zprovoznění zadaného systému a nasazení na Value Stream Delivery Platformu CodeNOW. Poté jsou rozebrány různé API Gateways a implementace vybraného řešení. V další kapitole je řešen úkol rozšíření systému o platební bránu. Následně je popsán úkol přidání možnosti upomínání opožděných plateb přes SMS. Dále jsou vysvětleny vzory CQRS a Event sourcing. Poté jsou popsány vylepšení nad rámec zadání, následně jakým způsobem probíhalo testování a na konci se nachází shrnutí celé práce a budoucnost systému.

Klíčová slova: Java, Spring Boot, Apache Kafka, architektura mikroslužeb, CodeNOW, Keycloak, PostgreSQL, Twilio, Stripe, API Gateway, CQRS, Event sourcing

Školitel: Ing. Martin Komárek

Abstract

This bachelor thesis focuses on extending the existing system "Payment Control" with new features and implementation of patterns typical for microservice architecture. It describes the commissioning of the specified system and deployment on the Value Stream Delivery Platform CodeNOW. Then the various Gateway APIs and the implementation of the selected solution are discussed. In the next chapter, the task of extending the system with a payment gateway is dealt with. Subsequently, the task of adding the ability to remind late payments via SMS is described. Next, the CQRS and Event sourcing patterns are explained. Then the enhancements beyond the assignment are described, followed by how the testing was done and finally a summary of the whole work and the future of the system.

Keywords: Java, Spring Boot, Apache Kafka, microservice architecture, CodeNOW, Keycloak, PostgreSQL, Twilio, Stripe, API Gateway, CQRS, Event sourcing

Title translation: Microservice architecture in practice

Obsah

1 Úvod 1

2 Popis a nasazení původního systému 3

2.1 Výchozí stav aplikace 4
2.2 Zprovoznění aplikace pro kontrolu plateb 4
2.2.1 Zprovoznění frontendu 4
2.2.2 Zprovoznění backendu 5
2.2.3 Nasazení na CodeNOW 6
2.2.4 Shrnutí zprovoznění aplikace . 7
2.3 Použitá architektura a technologie 8
2.3.1 Architektura mikroslužeb 8
2.3.2 React 9
2.3.3 TypeScript 10
2.3.4 Spring Boot 10
2.3.5 Apache Kafka 10
2.3.6 Keycloak 10
2.3.7 PostgreSQL 10
2.3.8 CodeNOW 11
2.3.9 Kubernetes 11
2.3.10 Grafana 11
2.3.11 On-Premise Vs Cloud řešení 12

3 Mikroslužba API Gateway 13

3.1 Funkce API Gateway 13
3.2 Existující řešení 14
3.2.1 Kong 14
3.2.2 Amazon API Gateway 14
3.2.3 Azure Application Gateway . 15
3.2.4 Express Gateway 15
3.2.5 KrakenD 15
3.2.6 Spring Cloud Gateway 15
3.3 Vybrané řešení 15
3.3.1 Implementace Spring Cloud Gateway 15

4 Platební brána 17

4.1 Důvod implementace platební brány 17
4.2 Funkce platebních bran 17
4.3 Existující řešení 18
4.3.1 PayPal 18
4.3.2 Amazon Pay 19
4.3.3 Braintree 19
4.3.4 Skrill 19
4.3.5 Stripe 19
4.4 Vybrané řešení 19
4.4.1 Implementace platební brány Stripe 19

5 Notifikace přes SMS 21

5.1 SMS brána 21
5.2 Existující řešení 22
5.2.1 Telesign 22
5.2.2 Vonage 22
5.2.3 Amazon SNS 22
5.2.4 MessageBird 22
5.2.5 Telnyx 23
5.2.6 Twilio 23
5.3 Vybrané řešení 23
5.3.1 Implementace SMS brány Twilio 23

6 Návrhový vzor CQRS a Event sourcing 25

6.0.1 Co to je CQRS 25
6.0.2 Event sourcing 26
6.0.3 CQRS a Event sourcing 27
6.0.4 CQRS a Event sourcing v tomto systému 27

7 Další vylepšení systému nad rámec zadání 29

7.1 QR kód 29

8 Testování systému 31

8.1 Statické testy 31
8.2 Unit testy 31
8.3 Integrační testy 31
8.4 Smoke testy 32

9 Závěr 33

9.1 Budoucnost systému 34

Literatura 35

A Obsah elektronické přílohy 41

Obrázky

2.1 Diagram architektury systému.[24] 3
2.2 Ukázka z nástroje Grafana.[21] . . 7

6.1 Návrhový vzor CQRS.[15] 26

8.1 Diagram architektury systému po implementaci všech změn a úprav. 32

Tabulky

Kapitola 1

Úvod

Tato bakalářská práce navazuje na předchozí bakalářskou práci "Cloud native aplikace pro kontrolu plateb"[24], kterou v květnu roku 2021 odevzdal Jiří Pazdera.

Systém "Kontrola plateb"byl vytvořen s cílem usnadnit a zefektivnit proces kontroly bankovních plateb pro různé ziskové i neziskové organizace. Díky tomuto systému mohou organizace efektivněji monitorovat příchozí platby, provádět kontrolu jejich správnosti a včas reagovat na případné problémy. Systém zajišťuje, že žádné platby nejsou ztraceny nebo přehlédnuty.

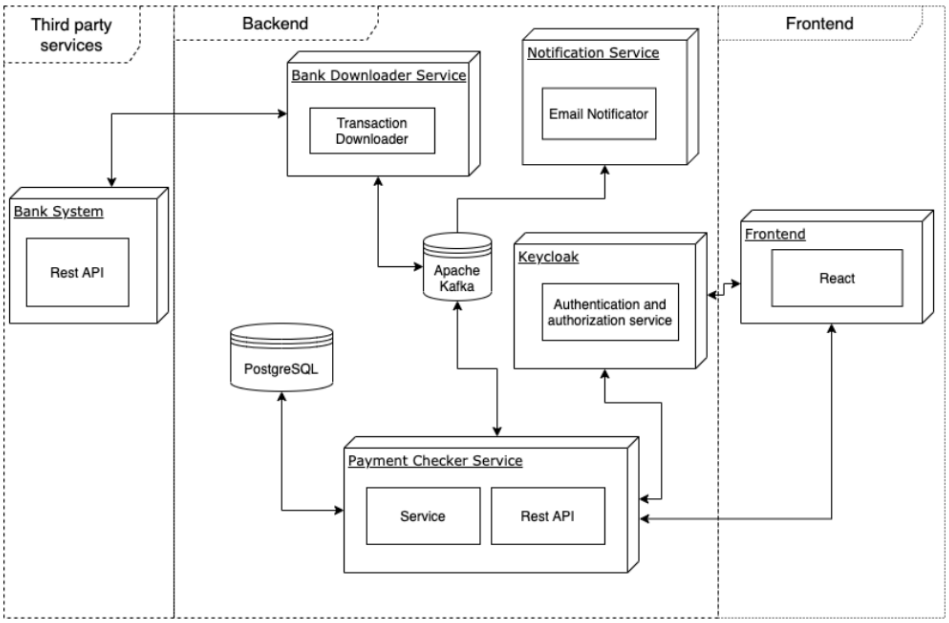
Cílem této práce je rozšířit stávající systém o nové funkce a implementovat několik vzorů typických pro architekturu mikroslužeb. První část práce se věnuje popisu původního systému, jeho zprovoznění a nasazení na platformu CodeNOW. Dále je popsán úkol rozšíření systému o API Gateway a implementace vybraného řešení. V další kapitole je řešen úkol rozšíření systému o platební bránu. Práce dále popisuje úkol přidání možnosti upomínání opožděných plateb prostřednictvím SMS zpráv. Následně je vysvětlen architektonický vzor CQRS (Command Query Responsibility Segregation) a návrhový vzor Event sourcing. Práce dále popisuje vylepšení nad rámec původního zadání, poté jakým způsobem probíhalo testování a v závěru se nachází shrnutí celé bakalářské práce.

Kapitola 2

Popis a nasazení původního systému

V této kapitole rozeberu předchozí bakalářskou práci, která je využita jako startovní bod v této bakalářské práci. Popíši zde nezbytné kroky pro zprovoznění aplikace a problémy se kterými jsem se setkal. Zde uvádím stručný popis předchozí bakalářské práce a obrázek 2.1 architektury:

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.[24]



Obrázek 2.1: Diagram architektury systému.[24]

2.1 Výchozí stav aplikace

Bakalářská práce[24] Jiřího Pazdery byla odevzdána v květnu roku 2021. Jeho aplikace byla funkční a splnila všechny požadavky zadání. Nicméně v posudku mu oponent práce "Ing. Ivo Malý, Ph.D."v kritériu Odborná úroveň vytknul: "Zdrojový kód by měl být lépe okomentovaný, aby se dal dále rozvíjet. Taktéž student detailněji nepopsal způsob kompilace a nasazení celé klient-server aplikace.".[24]

Nasazení této aplikace je popsáno v této kapitole.

2.2 Zprovoznění aplikace pro kontrolu plateb

Tato podkapitola popisuje kroky nezbytné pro zprovoznění aplikace pro kontrolu plateb.[24]

Po stažení lze vidět, že obsahuje čtyři hlavní moduly:

- (1) aca-pazderaa-bank-downloader - zdrojový kód mikroslužby stahující transakce z banky
- (2) aca-pazdera-notificator - zdrojový kód notifikační mikroslužby
- (3) payment-checker-service - zdrojový kód mikroslužby na párování plateb
- (4) pazdejir-frontend - zdrojový kód frontendu

V rámci projektu bylo dohodnuto s mým školitelem Ing. Martinem Komárkem, že aplikace bude zprovozněna a nahrána na platformě CodeNOW. V květnu 2021 byla aplikace Jiřím Pazderou pro kontrolu plateb nasazena na CodeNOW, ale jelikož se CodeNOW stále vyvíjí, tak po určitých změnách přestala tato aplikace fungovat (důvody jsou například: zastaralé dependencies, chybějící konfigurační soubory) a byla z CodeNOW smazána. Od mého školitele mi bylo doporučeno tuto aplikaci nejdříve zprovoznit na svém počítači (localhost) a až poté ji nahrát na platformu CodeNOW. Následně vysvětlím problémy, se kterými jsem se setkal při zprovoznění této aplikace pro kontrolu plateb.

2.2.1 Zprovoznění frontendu

Frontend používá knihovnu React (viz 2.3.2) a programovací jazyk TypeScript (viz 2.3.3).

Potřebné knihovny jsou zapsány v souboru package.json a lze je stáhnout nástrojem npm [35]. Při prvním spuštění frontendu jsem zaznamenal několik chyb, jež ukazovaly na řádky kódu, které jsem předtím nijak neupravoval. Po delším zkoumání jsem zjistil, že chyba byla v souboru package.json. Ten totiž obsahuje použité knihovny a zároveň i verze knihoven. Zde uvádím příklad jakým způsobem byla každá verze knihovny zapsána: "typescript":"^4.1.2". U tohoto příkladu je zápis verze ekvivalentní s: '`>= 4.1.2 < 5.0.0`' [34].

To je rozptyl verzí, který je možný u dané knihovny využít a automaticky se stáhne nejnovější verze z tohoto rozptylu. Při užití znaku stříšky (^) je doporučeno pravidelně prohlížet kód, zda je kompatibilní s nejnovější verzí nebo ne. Od skončení předchozí bakalářské práce a začátkem mé bakalářské práce uplynul více než rok, takže některé knihovny vydaly novější verze a to způsobilo určitou nekompatabilitu. Po odstranění znaku stříšky (^) před určitými verzemi knihoven a tedy použitím starších verzí všechny chyby zmizely.

Dále již stačilo upravit soubor ".env", který obsahuje proměnné jako je například.: URL Keycloaku a URL backendu.

2.2.2 Zprovoznění backendu

Backend této aplikace je tvořen ze tří mikroslužeb a to z mikroslužby stahující transakce z banky, notifikační mikroslužby a mikroslužba na párování plateb. Tyto tři mikroslužby jsou naprogramované v jazyce Java a využívají framework Spring Boot (viz 2.3.4).

Pro fungování aplikace musí mikroslužby spolu komunikovat, a proto je zde využit nástroj Apache Kafka (viz 2.3.5).

Všechny tři mikroslužby používáji tuto technologii, pouze se odlišují v tom, jaké téma (topic) používají. Nejdříve je potřeba zapnout Apache ZooKeeper[19], což je koordinační služba, kterou používá Kafka broker k ukládání a správě metadat o clusteru Kafka a tématech (topics). Dále je potřeba zapnout Kafka broker[6], což je server na kterém běží Kafka a ukládá data (zprávy, topics) pro každou mikroslužbu se správným portem.

Mikroslužba stahující transakce z banky

Tato mikroslužba je napojena na bankovní api od Fio a stahuje transakce ve formátu XML z daného bankovního účtu určeného tokenem, který dostane z Kafky. Následně jsou transakce deserializována, vyfiltrují se aby byly použity pouze odchozí transakce a ty jsou následně uloženy do Kafky.[24] Tento token si uživatel může vytvořit po přihlášení do svého Fio účtu v nastavení API. Tokenu lze nastavit dobu platnosti, pojmenování a různá práva.

Notifikační mikroslužba

Tato mikroslužba slouží k upozorňování uživatelů e-mailem. Z Kafky si načte data komu se e-mail odesílá, předmět, kontaktní osoba a jaký typ zprávy (první notifikace, upomínka před datem, upomínka pozdní, přijatá platba). Mikroslužba využívá knihovnu Thymeleaf, což je šablonový systém pro Javu. [24]

Po zapnutí této mikroslužby je připojena k SMTP (Simple Mail Transfer Protocol serveru od Googlu[53], který se využívá k odesílání e-mailových zpráv. Server od společnosti Google lze vytvořit přihlášením do služby Gmail, kde je možné si tento server vygenerovat a zároveň vygeneruje i heslo k připojení serveru. V notifikační mikroslužbě je poté potřeba v application.yaml nastavit

e-mail od Gmailu, port, protokol a heslo od serveru. Adresa SMTP Google serveru je smtp.gmail.com.

Mikroslužba na párování plateb

"Hlavní část systému je mikroslužba na párování plateb. 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."[24]

V případě, že je potřeba odeslat e-mail nějakému uživateli, tak uloží tuto informaci do Kafky a tu zpracuje Notifikační mikroslužba. V databázi je osm tabulek a to konkrétně: User, Transaction, Prescribed_Payment, Payment_Package, Pairing, Organization_User_Role, Organization, Bank_Account.

Tato mikroslužba využívá kromě předchozích zmíněných technologií i Keycloak (viz 2.3.6) a PostgreSQL (viz 2.3.7).

Pro fungování Keycloaku je potřeba stáhnout a nainstalovat Keycloak a následně nakonfigurovat Keycloak server. To zahrnuje přidání a nastavení Keycloak realmu, vytvoření a konfiguraci klientů a vytvoření uživatelských účtů. Dále již stačilo v mikroslužbě na párování plateb v konfiguračním souboru application.yaml změnit proměnnou názvu realmu a URL Keycloak serveru.

Pro používání PostgreSQL [39] je potřeba stáhnout a nainstalovat PostgreSQL a zapnout server (například pomocí scriptu). Jakmile server běží, tak se dá ovládat klientskými nástroji jako je pgAdmin. Tento nástroj umožňuje interakci s databází a provádění úkolů mezi které patří vytváření nových databází a tabulek, vkládání dat a další. V mikroslužbě na párování plateb je potřeba v konfiguračním souboru application.yaml změnit proměnné jako je URL databáze, port a přihlašovací údaje.

2.2.3 Nasazení na CodeNOW

V rámci projektu bylo určeno, že aplikace bude zprovozněna a nahrána na platformě CodeNOW (viz 2.3.8).

Na CodeNOW [13] lze vytvořit aplikaci a v ní komponenty, což jsou nezávislé vývojové části. V aplikaci na kontrolu plateb se jedna komponenta rovná jedné mikroslužbě. Každá komponenta aplikace obsahuje zdrojový kód dané aplikace v samostatných GitLab repozitářích. Při vytváření komponenty lze specifikovat technologický stack, například.: Java/Maven/Springboot, JavaScript/React a lze k ním připojit následující služby:[14]

- relační databáze: PostgreSQL, MariaDB
- distribuovaná databáze: CockroachDB
- in-memory databáze: Redis
- cloudové úložiště: MinIO
- platforma pro distribuované streamování událostí: Apache Kafka

- zprostředkovatel zpráv (anglicky message broker): RabbitMQ
- správa identity a přístupu: Keycloak

Platforma CodeNOW nabízí možnost vytvoření instancí těchto služeb aniž by se musely instalovat a nějak složitě konfigurovat. Pro aplikaci pro kontrolu plateb jsou využité služby PostgreSQL, Apache Kafka a Keycloak. Je tedy potřeba vytvořit instance těchto služeb na CodeNOW. Následně se v jednotlivých mikroslužbách a na frontendu musí přidat konfigurační soubor (a nebo upravit použité konfigurační soubory) pro spuštění aplikace v prostředí CodeNOW. To zahrnuje úpravu proměnné URL služeb a případně další související proměnné (například: přihlašovácí údaje do databáze).

Dále se musí do mikroslužeb přidat potřebné dependencies, které jsou nezbytné pro spuštění v CodeNOW. Konkrétně se tyto dependencies musí přidat do souboru pom.xml, který obsahuje informace o projektu a informace o konfiguraci pro Maven k sestavení projektu. Spouštění jednotlivých komponent se provádí v kontejnerech, které jsou správovány nástrojem Kubernetes (viz 2.3.9) a je možné je monitorovat pomocí nástroje Grafana (viz 2.3.10), který je zobrazen na obrázku 2.2.

Obrázek 2.2: Ukázka z nástroje Grafana.[21]

2.2.4 Shrnutí zprovoznění aplikace

Zásadním zjištěním pro mě je, že nasazení systému, který rok nikdo neudržoval a nebyl jsem součástí vývoje, tak bez dokumentace nasazení aplikace může být velice náročné.

Největším problémem byly obecně závislosti (dependencies), které byly nutné někde přidat (viz kapitola 2.2.3) a u některých upravit verzi (viz upravení verzí knihoven v kapitole 2.2.1). Dále bylo potřeba naučit se pracovat s platformou CodeNOW, kam byl systém úspěšně nasazen. Tento proces nasazení, aby vše fungovalo, trval přibližně 50 hodin.

2.3 Použitá architektura a technologie

Tato sekce se zaměřuje na popis použité architektury a technologií, které byly využity v aplikaci Kontrolor plateb.

2.3.1 Architektura mikroslužeb

Architektura mikroslužeb [45] je přístup k návrhu softwaru, ve kterém je aplikace sestavena jako sada malých nezávislých mikroslužeb z nichž je každá zodpovědná za konkrétní úkol nebo funkci v rámci celkové aplikace. Každá mikroslužba je navržena tak, aby byla vysoce modulární s vlastní kódovou základnou (anglicky codebase), databází a dalšími závislostmi. To umožňuje každou mikroslužbu vyvíjet, testovat a nasazovat nezávisle na ostatních mikroslužbách, což znamená, že je lze upravit nebo nahradit aniž by to ovlivnilo zbytek systému.

Výhody architektury mikroslužeb

Architektura mikroslužeb nabízí několik klíčových výhod, které ji činí atraktivní volbou pro mnoho typů softwarových systémů.[45]

- Flexibilita - Mikroslužby jsou nezávisle nasaditelné jednotky kódu, které lze vyvíjet, testovat a nasazovat samostatně. To umožňuje větší flexibilitu v procesu vývoje a usnadňuje provádění změn v konkrétní službě, aniž by to ovlivnilo celý systém. Umožňuje také týmům pracovat na různých službách současně a nasazovat jejich změny nezávisle na sobě.
- Škálovatelnost - Architektura mikroslužeb usnadňuje škálování aplikace při jejím růstu. Vzhledem k tomu, že každá mikroslužba je samostatná a nezávislá, lze ji podle potřeby škálovat nahoru nebo dolů, aniž by to ovlivnilo ostatní mikroslužby. To usnadňuje zvládat změny požadavků a umožňuje aplikaci růst a vyvíjet se v průběhu času. Tento přístup je užitečný zejména pro aplikace, u kterých se očekává, že časem porostou a budou se vyvíjet, protože umožňuje agilnější vývoj a snadnější údržbu. Mikroslužby lze také horizontálně škálovat. To může být užitečné zejména u systémů, které mají některé komponenty využívané hodně a jiné ne, protože to umožňuje škálovat pouze ty služby, které to potřebují.
- Odolnost proti chybě - Systém založený na mikroslužbách může být odolnější vůči selhání, protože chyba v jedné službě může sice způsobit její pád, nicméně toto selhání je izolováno na tuto službu a nezpůsobí pád celé aplikace. To může zvýšit spolehlivost a dostupnost systému jako celku. Narozdíl v monolitické architektuře by takové selhání mohlo způsobit pád celého systému.
- Vývoj a nasazení - Architektura mikroslužeb umožňuje rychlejší vývoj a cykly nasazení. Vzhledem k tomu, že na každé službě lze pracovat nezávisle, lze různé části aplikace vyvíjet a vydávat v různých časech, aniž

by bylo nutné koordinovat změny napříč celým systémem. To usnadňuje provádění změn a přidávání nových funkcí do aplikace a stejně tak i opravování chyb nebo problémů s výkonem.

- Výběr jazyku a technologií - Mikroslužby mohou být napsány v různých programovacích jazycích a mohou využívat různé technologie, což umožňuje flexibilitu při výběru nástrojů a technologií použitých k vytvoření systému. To může usnadnit používání nejlepších nástrojů pro danou práci a výběr technologického stacku. Pokud je potřeba přepsat mikroslužbu do jiného jazyka, tak to není takový problém jako přepis kódu monolitické aplikace, protože každá mikroslužba je relativně malá.

Nevýhody architektury mikroslužeb

I když architektura mikroslužeb nabízí mnoho výhod, není bez problémů a kompromisů. Mezi nevýhody patří: [45]

- Větši komplexita - Architektura mikroslužeb přináší do systému další složitost, protože vyžaduje dekompozici systému na menší nezávislé komponenty. To může ztížit pochopení a údržbu systému.
- Problémy při vývoji a nasazení - Vývoj a nasazení mikroslužeb může být složitější než u monolitických systémů, protože každá služba představuje samostatnou kódovou základnu a musí být nasazena a spravována nezávisle.
- Problémy s komunikací - Aby mikroslužby fungovaly jako kompletní systém, musí spolu komunikovat. To může být náročné, protože to vyžaduje pečlivý návrh rozhraní API a rozhraní mezi službami, aby se zajistila jejich spolehlivost, efektivita a bezpečnost.
- Větší náročnost na zdroje - Architektura mikroslužeb může vyžadovat více zdrojů (například paměť a procesor), protože každá služba běží ve vlastním procesu a může vyžadovat vlastní server nebo kontejner. To může vést ke zvýšení nákladů na infrastrukturu a může to také ztížit optimalizaci využití zdrojů.
- Ladění (anglicky debugging) a testování - Ladění a testování systému založeného na mikroslužbách může být náročnější, protože vyžaduje sledování interakcí mezi více službami a identifikaci hlavní příčiny problémů. To může být časově náročné a může vyžadovat specializované nástroje a procesy.

2.3.2 React

React [41] je populární JavaScriptová knihovna pro vytváření uživatelských rozhraní a často se používá pro tvorbu jednostránkových (single-page) aplikací. Jednou z klíčových vlastností Reactu je schopnost vytvářet opakovaně použitelné komponenty uživatelského rozhraní. Komponenty jsou samostatné

jednotky, které představují část uživatelského rozhraní. Mohou přijímat vstupy (props) a vracet React prvek (element), který popisuje, jak má uživatelské rozhraní vypadat.

2.3.3 TypeScript

TypeScript [64] je programovací jazyk, který je nadstavbou JavaScriptu a rozšiřuje jej o funkce jako jsou třídy a rozhraní, které v JavaScriptu nejsou a mohou usnadnit psaní objektově orientovaného kódu.

2.3.4 Spring Boot

Spring Boot [49] je framework určený pro programovací jazyk Java a jedná se o nadstavbu na framework Spring. Implementuje tedy veškeré funkcionality Spring frameworku a navíc má vestavěný webový server Tomcat, takže se může aplikace spustit jako samostatná jednotka aniž by se musela nastavovat konfigurace aplikačního serveru. Spring Boot automaticky zvládne mnoho konfigurací, které jsou pro aplikaci Spring nutné, včetně nastavení kontextu aplikace, konfigurace DataSource a nastavení připojení k databázi.

2.3.5 Apache Kafka

Apache Kafka [6] je open-source platforma pro distribuované streamování události, která se používá jako systém pro zasílání zpráv (anglicky message broker), kde producenti posílají zprávy do topics (témat) a konzumenti je přijímají. Kafka ukládá záznamy na disk, což znamená, že jsou trvalé i v případě, že dojde k výpadku clusteru Kafka nebo že producent či konzument přestane fungovat. Kafka také umožňuje horizontální škálování rozdělením dat v tématu na více oddílů (anglicky partions), které lze distribuovat na více serverů.

2.3.6 Keycloak

Keycloak [28] je systém, který se běžně používá k zabezpečení aplikací a služeb. Poskytuje funkce, jako je jednotné přihlášení, správa uživatelů. Podporuje několik standardních protokolů a poskytuje podporu pro OpenID Connect, OAuth 2.0 a SAML. V této mikroslužbě plní účel jako autentizační server, který umožňuje uživatelům se zaregistrovat a přihlásit.

2.3.7 PostgreSQL

PostgreSQL (zkráceně Postgres) je relační databázový systém s otevřeným zdrojovým kódem. Využívá a rozšiřuje jazyk SQL, který se používá k ukládání a manipulaci s daty. PostgreSQL běží na všech hlavních operačních systémech včetně Linuxu a Windows. Splňuje a je kompatibilní s ACID (atomicita, konzistence, izolovanost, trvalost) a podporuje velké množství datových typů. Dokonce je možné definovat i vlastní datový typ.[38]

2.3.8 CodeNOW

CodeNOW [12] je cloudová platforma, která poskytuje řadu nástrojů a služeb pro automatizaci procesu sestavování, testování a nasazování softwaru, což týmům umožňuje rychleji a spolehlivěji dodávat aktualizace a změny softwaru a služeb. Poskytuje funkce jako je kontinuální intergace (anglicky continuous integration) a nasazení, revize kódu, řízení projektů a integruje se s řadou nástrojů a služeb třetích stran.

- Revize kódu - CodeNOW poskytuje nástroje pro provádění revizí kódu a spolupráci na změnách kódu, čímž pomáhá týmům zlepšit kvalitu a udržovatelnost jejich kódové základny. Jedním z těchto nástrojů je SonarCube [48], který je využit při statickém testování (viz 8.1).
- Řízení projektů - CodeNOW poskytuje nástroje pro správu projektů pro sledování a řízení projektů vývoje softwaru, včetně funkcí, jako je sledování problémů a agilní plánování projektů.
- Integrace třetích stran - CodeNOW se integruje s řadou nástrojů a služeb třetích stran, včetně populárních systémů pro správu verzí, systémů pro sledování problémů a testovacích nástrojů.

2.3.9 Kubernetes

Kubernetes [43] je open-source systém pro automatické nasazení, škálování a orchestraci kontejnerů. Kontejnery jsou seskupovány do podů, které jsou základními provozními jednotkami v Kubernetes a tyto pody jsou škálovány na požadovaný stav. Nabízí funkce jako je automatické balení komponent do kontejnerů, samoopravování a horizontální škálování. Kubernetes automaticky restartuje kontejnery, které selhaly. V Kubernetes clusteru se stroje, na kterých běží aplikace nazývají uzly (anglicky nodes). Uzly jsou spravovány řídicí rovinou (anglicky control plane), která se skládá ze sady procesů běžících na jednom nebo více strojích. Tyto procesy jsou zodpovědné za udržování požadovaného stavu clusteru, například počtu replik určité aplikace, které mají být spuštěny. Kubernetes používá deklarativní model konfigurace, což znamená, že zadáte požadovaný stav aplikace a řídicí rovina zajistí, aby cluster tomuto stavu odpovídal, což usnadňuje automatizaci nasazení a správy aplikací.

2.3.10 Grafana

Grafana[42] je open-source nástroj pro vizualizaci a monitorování dat (viz obrázek 2.1), který umožňuje vytvářet, zkoumat a sdílet ovládací panely (anglicky dashboards). Tyto ovládací panely lze vytvářet a přizpůsobovat tak, aby zobrazovaly data, která jsou pro nás ta nejdůležitější. Dokáže zobrazovat data z různých zdrojů, například ze souborů protokolů, databází a systémů pro monitorování aplikací. Nástroj Grafana [21] dokáže vám nebo vašemu týmu zasílat upozornění, když je dosaženo určitých mezních hodnot nebo pokud se vyskytnou jiné problémy, kterým je třeba věnovat pozornost.

2.3.11 On-Premise Vs Cloud řešení

On-Premise a Cloud software [66] jsou dva uplně odlišné modely licencování a dodání. [46]

- ON-PREMISE - Jedná se o lokálně nainstalovaný software na vašich počítačích či serverech. Většinou se platí jednorázový licenční poplatek. Správa, údržba a bezpečnost tohoto řešení je na vás. Máte ale úplnou kontrolu nad systémem a můžete k němu přistupovat i bez internetového připojení.
- CLOUD - Software as a Service (Saas) je software, který je hostován centrálně a licence je v podobě předplatného. Údržba a bezpečnost systému je na poskytovateli, ale k užívání je potřeba internetové připojení.

Kapitola 3

Mikroslužba API Gateway

Tato kapitola se zabývá návrhovým vzorem API Gateway neboli API brána. Jsou zde vysvětleny hlavní a podpůrné funkce tohoto vzoru (viz 3.1), rozebrány některé existující řešení (viz 3.2) a následně představení řešení (viz 3.3) pro tento systém.

3.1 Funkce API Gateway

API Gateway má smysl implementovat pro většinu systému založených na mikroslužbách. [45] Hlavní odpovědností API Gateway je směřování požadavků do mikroslužeb a agregace požadavků. [8] Dále má i několik okrajových funkcí zahrnující autentizaci, autorizaci, omezení rychlosti požadavků, mezipaměť odpovědí a sběr metrik. Zde jsou zmíněné funkce rozepsány do většího detailu: [23]

- Směřování požadavků (request routing) - API Gateway poskytuje jediný vstupní bod do systému pro definovanou skupinu mikroslužeb. Všechny API požadavky od externích klientů jdou nejprve do API Gateway, která přesměruje jejich požadavky na odpovídající koncové body interních mikroslužeb. To usnadňuje správu a škálování aplikace. Může se tím zjednodušit architektura jelikož je poskytnut jediný vstupní bod pro klientské požadavky. Funkce směřování požadavků je zároveň i potenciální nevýhoda, jelikož se z API Gateway stává jediný bod selhání. Pokud dojde k výpadku API Gateway, může to ovlivnit dostupnost celé aplikace.

- Agregace požadavků (API composition) - V rámci API Gateway je možné sdružit více klientských požadavků, které cílí na více interních mikroslužeb do jednoho požadavku klienta. Získaná data jsou odeslaná zpět do klientské aplikace. To může mít za následek nižší latenci odpovědi a celkově vyšší výkon aplikace.

- Autentizace - Ověření totožnosti klienta, který zadává požadavek.

- Autorizace - Ověření, zda je klient oprávněn provádět danou operaci.

- Omezení rychlosti (rate limiting) - Omezení počtu požadavků za konkrétní čas od konkrétního klienta a nebo od všech klientů, aby se zabránilo přetížení systému a jeho případnému pádu.
- Mezipaměť (cache) - Ukládání odpovědí do mezipaměti, aby se snížil počet požadavků na služby.
- Sběr metrik - API Gateway může poskytovat různé metriky a protokoly, které lze použít ke sledování výkonu a stavu aplikace. Tyto metriky mohou zahrnovat například počet požadavků, chybovost a dobu odezvy.

3.2 Existující řešení

Existuje několik existujících API Gateways. [8] Na výběr je mnoho možností, ať už jde o samostatné nástroje nebo funkce začleněné do širší platformy pro správu API. Výběr té správné API Gateway závisí na: [27]

- 1) složitosti nasazení
- 2) On-Premise vs Cloud (vysvětlení v sekci 2.3.11)
- 3) nabízených funkcí (různé API Gateways nabízí různé funkce)
- 4) peněžních nákladech (většina nástrojů nabízí free verzi, ale cena se zvyšuje s nabízenými funkcemi)

Existuje několik řešení a následující text popisuje některá z populárních variant.

3.2.1 Kong

Kong [29] je open source API Gateway s velkou komunitou vývojářů, kteří se podílejí na jeho vývoji a podpoře. Tato API Gateway je populární pro svou škálovatelnost, rozšiřitelnost a nabízené funkce zahrnující směřování požadavků, autentizaci, omezení rychlosti požadavků a mnoho dalších. Nabízí cloud i on-premise řešení.

3.2.2 Amazon API Gateway

Amazon API Gateway [3] je plně spravovaná služba, která vývojářům usnadňuje vytváření, publikování, údržbu, monitorování a zabezpečení rozhraní API v libovolném rozsahu. Integruje se s dalšími službami AWS, podporuje rozhraní API RESTful i WebSocket a umožňuje nám obousměrnou komunikaci v reálném čase. Pokud jsou mikroslužby nebo rozhraní API již hostovány na AWS, má smysl integrovat je s Amazon API Gateway. Nabízí pouze cloud řešení.

3.2.3 Azure Application Gateway

Azure Application Gateway [1] nabízí komplexní správu rozhraní API v cloudu i na lokálu poskytované službou Microsoft Azure. Poskytuje funkce jako je směřování požadavků, ukládání do mezipaměti, omezování, analýza a integruje se s dalšími službami Azure. Nabízí pouze cloud řešení.

3.2.4 Express Gateway

Express Gateway [18] je postavena na systému Express.js. Express Gateway je soubor komponent, které jsou deklarativně postaveny kolem Express, aby vyhovovaly případu použití API Gateway. Síla Express Gateway spočívá ve využití bohatého ekosystému kolem middlewaru Express. Je jednoduchý, rychlý a nabízí všechny základní funkce. Nabízí cloud i on-premise řešení.

3.2.5 KrakenD

KrakenD [30] je open source API Gateway. Její základní funkcí je vytvoření rozhraní API, které funguje jako agregátor mnoha mikroslužeb do jediného koncového bodu a automaticky provádí: agregaci, směřování požadavků, filtrování, autentizaci, omezení rychlosti požadavků a další. Nabízí deklarativní způsob vytváření koncových bodů. Nabízí cloud i on-premise řešení.

3.2.6 Spring Cloud Gateway

Jedná se o open-source API Gateway postavenou nad Spring Framework 6, Spring Boot 3 a Project Reactor. [50] Je navržena s cílem poskytnout jednoduchý, ale efektivní způsob směřování požadavků na rozhraní API a mikroslužby. Tato API Gateway nabízí širokou škálu funkcí, včetně bezpečnosti, monitorování metrik, dynamického směřování požadavků, filtrování požadavků, omezení rychlosti požadavků a jističe (circuit breaking). Také podporuje několik protokolů jako je HTTP, WebSockets a TCP.

3.3 Vybrané řešení

Na základě provedené rešerše bylo rozhodnuto využít Spring Cloud Gateway (viz sekce 3.2.6) jako preferované řešení pro API Gateway v systému Kontrolor plateb. Tato volba je v souladu s celkovou architekturou systému, která je založena na programovacím jazyce Java a frameworku Spring Boot(viz 2.3.4). Výběrem Spring Cloud Gateway zajistíme bezproblémovou integraci s již existujícími komponentami systému, jelikož se stane další mikroslužbou postavenou na frameworku Spring Boot v rámci tohoto systému.

3.3.1 Implementace Spring Cloud Gateway

Základním pilířem pro vytvoření Spring Cloud Gateway [50] je vytvořit Spring Boot projekt do kterého se přidá "spring-cloud-starter-gateway"závislost[50].

Po přidání této závislosti záleží už jen na nás, jaké funkce chceme, aby naše API Gateway zvládala. Výhodou tedy je, že je velice flexibilní a můžeme si ji upravit podle svých potřeb. Nevýhodou je, že je potřeba si tuto API Gateway naprogramovat samostatně, nicméně tato knihovna má velice dobrou dokumentaci. [50]

V tomto projektu má naše API Gateway na starost směřování požadavků do určité mikroslužby. Dále zabezpečuje mikroslužby s JWT (JSON Web Token) [9] získaný ze služby Keycloak (viz 2.3.6), který se používá k ověření uživatele rozhraní API a k povolení autorizace OAuth 2.0 pro všechna rozhraní API chráněná protokolem OAuth pomocí OpenID Connect.

V souboru application.yaml jsou zapsány informace o možných cestách, na které lze směřovat požadavky (na které mikroslužby a jejich REST API), a také URL Keycloaku, ze kterého se získává token JWT. Při spuštění mikroslužby API Gateway jsou tyto informace z application.yaml inicializovány ve třídě SecurityConfig.

Kapitola 4
Platební brána

Tato kapitola se zabývá rozšířením systému o možnost placení přes platební bránu. Jsou zde rozepsány existující řešení a implementace vybraného řešení.

4.1 Důvod implementace platební brány

Původní záměr důvodu implementace platební brány bylo zjednodušení odeslání peněz pro plátce. Nyní se nejdříve vytvoří platební předpis a ten je odeslán e-mailem plátci s potřebnými informacemi ke platbě jako je bankovní účet, částka a variabilní symbol. Plátce musí tyto informace pro zaplacení ručně opsat. Vylepšením mělo být přidání do tohoto e-mailu odkaz, který je přesměruje na stránku aplikace Kontrolor plateb, kde budou tyto údaje předvyplněny. Poté by plátci stačilo vyplnit kreditní kartu a zaplatit. Při samotné implementaci bylo zjištěno, že to není možné a platba přes platební bránu Stripe se pošle na určitý Stripe účet. Tento účet má pro komunikaci vlastní veřejný a tajný klíč (viz 4.4.1). Odtamtud by se peníze museli přeposlat na specifický bankovní účet i s určitým variabilním symbolem, což by nešlo zautomatizovat. Platební brána byla implementovaná a po dohodě se školitelem má prozatimní funkci zaslání peněžní podpory (donate) na aplikaci Kontrolor plateb. Jelikož původní nápad pro zjednodušení plateb nebyl zrealizován, tak bylo vymyšleno jiné vylepšení a to posílání v e-mailu i QR kód. Po naskenování tohoto QR kódu v bankovní aplikaci načte údaje ke platbě a není tedy potřeba údaje ručně přepisovat (viz 7.1).

4.2 Funkce platebních bran

Platební brána [17] je technologie zpracování transakcí, která zachycuje platby kartou, ukládá a přenáší informace o kartě od zákazníka k obchodníkovi. Poté sdílí oznámení o přijetí nebo odmítnutí platby zpět k zákazníkovi. V podstatě funguje jako prostředník mezi zákazníkem a zpracovatelem platby.

Existují dva hlavní typy platebních bran, hostované a integrované.[22]

- Hostované platební brány přesměrovávají zákazníky z webu k dokončení platby. Výhodou pro obchodníky je, že platební brána zajišťuje bezpeč-

nost transakcí a ochranu údajů zákazníka. Nicméně, někteří zákazníci mohou mít výhrady, protože platební brána odvádí zákazníky z webu.

- Integrované platební brány využívají rozhraní API platební brány, čímž umožňují, že platební brána je součástí webového rozhraní. Zákazníci nemusí být pro dokončení transakce přesměrováni jinam. Web se tak stává zodpovědným za bezpečnost transakcí a ochranu údajů o zákazníkovi.

Mezi klíčové vlastnosti platebních bran patří:[22]

- Bezpečnost - Jednou z nejdůležitějších funkcí, které je třeba u platební brány zohlednit, je zabezpečení. Měly by být v souladu s předpisy jako je PCI-DSS a mít pokročilé bezpečností funkce (např. 3D Secure)[7]. Tyto typy systémů pomáhají chránit údaje zákazníků před potenciálními hackery nebo podvodníky.
- Náročnost integrace - Některé platební brány lze na webu implementovat snadněji než jiné. Pro prodejce bez technických znalostí to může být důležité.
- Známost značky - Některé z platebních bran jsou online nakupujícím dobře známé. To může být výhodou z hlediska důvěry zákazníků.
- Podpora plateb - Je důležité ověřit, zda bude platební brána schopna přijímat platby v požadované měně a ve vybraných zemích.
- Náklady - Náklady spojené s používáním platební brány se dělí na tři typy: zřizovací poplatek, měsíční poplatek a transakční poplatek. Je tedy potřeba vybrat nejvýhodnější variantu.
- Předplatné [67] - Pokud požadujete od zákazníka opakovanou fakturaci je potřeba aby platební brána tuto funkci podporovala.

4.3 Existující řešení

Existuje mnoho platebních bran a některé z nich jsou popsána v této podkapitole.

4.3.1 PayPal

PayPal [37] je jednou z nejznámějších platebních bran a nabízí možnost hostované i integrované platební brány. PayPal platební bránu je možné nastavit zdarma, nicméně si PayPal účtuje poplatek za každou transakci. Služba PayPal je oblíbená a mnoho nakupujících používá účet PayPal, tudíž této platební bráně budou zákazníci věřit. Další výhodou je, že platební brána PayPal podporuje většinu bank a měn. Nabízí dodatečné funkce jako jsou další služby ochrany proti podvodům, opakované vyúčtovaní a ověření kupujícího.

4.3.2 Amazon Pay

Amazon Pay [2] je platební brána určená pro obchodníky a zákazníky Amazonu. Amazon si účtuje poplatky za transakce a nabízí automatické platby. Zákazníci se mohou přihlásit a zaplatit pomocí svých stávajících účtů Amazon s uloženými platebními údaji a adresou. To může zvýšit důvěryhodnost a snížit množství práce, kterou uživatelé potřebují k vykonání platby.

4.3.3 Braintree

Braintree [11] byla v roce 2013 [10] koupena společností PayPal. Nabízí obchodní nástroje pro budování globálních podniků, přijímání plateb a umožnění obchodování jejich uživatelům. Účtuje si poplatky za transakce. Má nástroje a funkce, které pomáhají škálovat podnikání jako je opakovaná fakturace. Tato platební brána podporuje několik typů platebních karet a dokonce i PayPal transakce.

4.3.4 Skrill

Skrill [47] je jednou z nejznámějších platebních bran. Má dobrou ochranu proti podvodům a je nastaven tak, aby fungoval po celém světě ve většině měnách. Je jednoduchá na nastavení a proto je vhodná pro začátečníky v oblasti elektronického obchodování. Účtuje si poplatky za transakce.

4.3.5 Stripe

Stripe [52] je cloudová platební brána, která nabízí možnost hostované i integrované platební brány. Podporuje širokou škálu bank a měn, což zajišťuje globální dostupnost. Jednou z klíčových funkcí pro obchodníky je možnost přizpůsobení platební pokladny podle jejich potřeb. Stripe si účtuje poplatky za transakce. Důležitou vlastností Stripe je také otevřené rozhraní API, které usnadňuje integraci této platební brány do existujícího systému.

4.4 Vybrané řešení

Na základě rešerše bylo jako vhodné řešení vybrána platební brána Stripe (viz sekce 4.3.5). Snadná integrace byla jedním z klíčových faktorů při výběru. Dokumentace je přehledná a obsahuje příklady kódu, což usnadňuje implementaci. Stripe nabízí podporu pro široké spektrum kreditních karet a zároveň klade velký důraz na bezpečnost.

4.4.1 Implementace platební brány Stripe

V této části popíšu jak jsem integroval platební bránu Stripe do systému Kontrolor plateb. Nejdříve bylo potřeba si založit účet na platformě Stripe. Tento účet obsahuje veřejný a tajný klíč, který je následně potřeba pro komunikaci s účtem Stripe. Implementace se dělí na na backend a frontend.

Backend

Tato platební brána byla implementována tak, že je to další nová mikroslužba vytvořená ve frameworku Spring Boot. Tajný klíč je potřeba v mikroslužbě Stripe pro vytvoření "PaymentIntent"[51], což se používá ke sledování platby od vytvoření až po zaplacení. Pokud dojde ke změně tohoto "PaymentIntent", například změna ceny nebo měny, tak je aktualizována. Také je potřeba přidat závislosti (dependency) pro používání Stripe objektů a jejich metod.

Frontend

Na frontendu bylo nejdříve potřeba přidat potřebné závislosti (dependency). Pro pracování se Stripe API je potřeba přidat do konfiguračního souboru veřejný klíč pro možnost komunikace se Stripe účtem. Poté jsem vytvořil stránku Donate, kde při načítání se kontaktuje na backendu mikroslužba platební brána a vytvoří se "PaymentIntent", což je nová platba. Uživatel si poté vybere možnost jak velkou částku chce zaplatit a po vyplnění údajů karty je možné platbu potvrdit. Údaje jsou při odeslání zvalidovány a podle toho jak platba dopadla se zobrazí uživateli zpráva o výsledku.

Kapitola 5
Notifikace přes SMS

Tato kapitola se věnuje přídání možnosti notifikace uživatelů přes SMS. V podkapitolách jsou některá existující řešení rozebrána a následně je popsané řešení.

5.1 SMS brána

Poskytovatel SMS brány umožňuje počítači odesílat a přijímat textové zprávy do mobilních telefonů a dalších systémů podporující SMS prostřednictvím globální telekomunikační sítě.[60] Funguje prostřednictvím protokolu SMPP (Short Message Peer-to-Peer). [26] Brána SMS přeloží odeslanou zprávu a připraví ji tak, aby byla kompatibilní pro doručení po síti a mohla se dostat k příjemci.

Při výběru nejvhodnější SMS brány je dobré zvážit:[20]

- Kvalita a rychlost doručení[20] - Patří to mezi jedny z nejdůležitějších parametrů. Je dobré si vybrat bránu, která se dokáže připojit k co největšímu počtu mobilních sítí po celém světě. Také je důležité aby jste měli jistotu doručení zprávy a rychlost doručení byla ve vteřinách místo hodin.

- Náklady - Cena za SMS je samozřejmě také velmi důležitá a u různých poskytovatelů SMS bran se liší. Vyplatí se také zjistit zda k používání SMS platformy není nutné předplatné.

- Bezpečnost[25] - Je důležité aby poskytovatel posílal zprávy přes ověřené trasy, které chrání zákazníky před spamem a nechtěnými zprávami.

- Integrace[26] - Je potřeba aby SMS brána šla integrovat s vybraným systémem a že se snadno používá. Nejlepší je vybrat si platformu, která nabízí rozhraní REST API. Jedná se o specifický typ rozhraní API, který je v oboru široce uznávaný a používaný. Poskytuje rychlejší komunikaci, vynikající výkon, bezpečnostní funkce a je škálovatelný.

5.2 Existující řešení

Existuje několik řešení a následující text popisuje některá z populárních variant.

5.2.1 Telesign

Platební brána Telesign [54] umožňuje podnikům odesílat SMS zprávy uživatelům po celém světě a to s funkcemi jako je dvoufaktorové ověřování a zobrazení informace o doručení v reálném čase.[16] Kromě toho umožňuje zasílání zpráv SMS, MMS, RCS, WhatsApp a Viber pomocí jediného rozhraní API pro vícekanálovou komunikaci. Cena SMS brány Telesign závisí na jejím využití, přičemž za každou SMS na číslo v České republice je účtována částka 1,23 Kč a nejsou vyžadovány žádné další poplatky.[55]

5.2.2 Vonage

Společnost Vonage, dříve známá jako Nexmo,[16] je jedním z předních poskytovalů komunikačních rozhraní API. Vonage kromě SMS má i další rozhraní API pro hlasové zprávy a video. Cena SMS brány Vonage závisí na jejím využití, přičemž za každou SMS na číslo v České republice je účtována částka 1,24 Kč a nejsou vyžadovány žádné další poplatky.[65]

5.2.3 Amazon SNS

Amazon SNS (Simple Notification Service) [4] je plně spravovaná služba pro zasílání zpráv poskytovaná společností Amazon Web Services (AWS). Díky tomu můžete mít nativní integraci s nejrůznějšími zdroji a cíli událostí AWS. Umožňuje posílat zprávy nebo oznámení velkému počtu účastníků nebo koncových bodů, včetně e-mailu a SMS. Cena za Amazon SNS závisí na jejím využití, přičemž za každou SMS na číslo v České republice je účtována částka 1,41 Kč a nejsou vyžadovány žádné další poplatky.[5]

5.2.4 MessageBird

Tomuto poskytovateli SMS API používá několik známých společností po celém světě včetně Google, WhatsApp, Facebook.[16] Rozhraní SMS API poskytované společností MessageBird je vysoce robustní, které umožňuje automatizované komunikační operace prostřednictvím kódů. Kromě základního rozhraní SMS API poskytuje také řešení pro ověřování mobilních čísel a převod e-mailů na SMS. Cena SMS brány MessageBird závisí na jejím využití, přičemž za každou SMS na číslo v České republice je účtována částka 1,15 Kč a nejsou vyžadovány žádné další poplatky.[31]

5.2.5 Telnyx

SMS brána Telnyx [57] poskytuje robustní a programovatelnou službu SMS API, které důvěřují přední společnosti včetně Slack, HP a Cisco. [16] Tato SMS brána poskytuje spoustu místních a bezplatných čísel. SMS brána umožňuje odesílání vysoce přizpůsobitelných textových zpráv prostřednictvím programovatelného rozhraní API, podporuje více programovacích jazyků, generuje zprávy o doručení a další. Cena SMS brány Telnyx závisí na jejím využití, přičemž za každou SMS na číslo v České republice je účtována částka 2,99 Kč a nejsou vyžadovány žádné další poplatky.[56]

5.2.6 Twilio

Společnost Twilio [63] je nejznámějším [16] poskytovatelem SMS brány a nabízí sadu dalších souvisejících služeb a funkcí, které rozšiřují možnosti SMS. Kromě SMS má Twilio další různá rozhraní API pro hlasové zprávy, e-mail a video. Twilio je vysoce spolehlivý a má silné celosvětové zastoupení. Cena SMS brány Twilio závisí na jejím využití, přičemž za každou SMS na číslo v České republice je účtována částka 1,22 Kč a nejsou vyžadovány žádné další poplatky.[61]

5.3 Vybrané řešení

Na základě rešerše bylo jako vhodné řešení vybrána SMS brána Twilio (viz sekce 5.2.6). Hlavním paramater pro výběr byla dobrá integrace a jelikož je Twilio velice populární, tak existuje i rozsáhlá dokumentace jak propojit SMS bránu Twilio s jiným systémem přes REST API.

5.3.1 Implementace SMS brány Twilio

Nejdříve je potřeba založit si účet u společnosti Twilio. Každý účet má SID (String Identifier)[62] a autentizační token[59]. To se využívá ke komunikaci s Twilio přes REST APIs.

Jelikož je SMS druh notifikace uživatele a v systému Kontrolor plateb je již Notifikační mikroslužba(viz 2.2.2), která umí odesílat e-maily, tak je to vhodné místo na rozšíření této mikroslužby o možnost odesílat i SMS. Notifikační mikroslužba má technologický stack: Java/Maven/Springboot, tudíž i tato SMS brána bude používat tyto technologie. Nejdříve je potřeba přidat Twilio dependency (závislosti) do souboru pom.xml , díky které je možné využívat knihoven Twilio. Při startu mikroslužby se nejdříve inicializuje konfigurace Twilio, která se skládá z SID a autentizačního tokenu. Byl také přidán REST kontroler, který má v sobě API na poslání SMS zprávy na zvolené číslo. Toto REST API je provoláváno na frontendu, kde má uživatel možnost odeslat SMS zprávu. V případě, že číslo je neplatné a nebo se zpráva nepovede doručit, tak je uživatel na chybu upozorněn.

Kapitola 6

Návrhový vzor CQRS a Event sourcing

Tato kapitola se věnuje architektonickému vzoru CQRS (Command Query Responsibility Segregation) a návrhovému vzoru Event sourcing. V této kapitole jsou zmíněné vzory vysvětleny a také proč jsou implementovány často dohromady.

6.0.1 Co to je CQRS

CQRS (Command Query Responsibility Segregation) je vzor, který odděluje operace zápisu (commands) a čtení (query) do samostatných modelů z nichž každý má vlastní odpovědnost (viz obrázek 6.1). V tradičních architekturách se k dotazování a aktualizaci databáze používá stejný datový model. To je jednoduché a dobře to funguje pro základní operace CRUD, ale v robustním systému se tento přístup může stát nepřehledným. Oddělením modelů čtení a zápisu si CQRS klade za cíl zjednodušit návrh aplikace a zlepšit její škálovatelnost, výkonnost a udržovatelnost. [32]

Model zápisu (write model) systému CQRS je zodpovědný za zpracování příkazů (commands), které mění stav systému. Příkazy vytvářejí (create), aktualizují (update) a nebo odstraňují (delete) entity v systému. Model zápisu se modeluje tak, aby databáze byla co nejspolehlivější. Může také zahrnovat obchodní pravidla, validační logiku a správu transakcí pro zajištění konzistentních dat v systému.

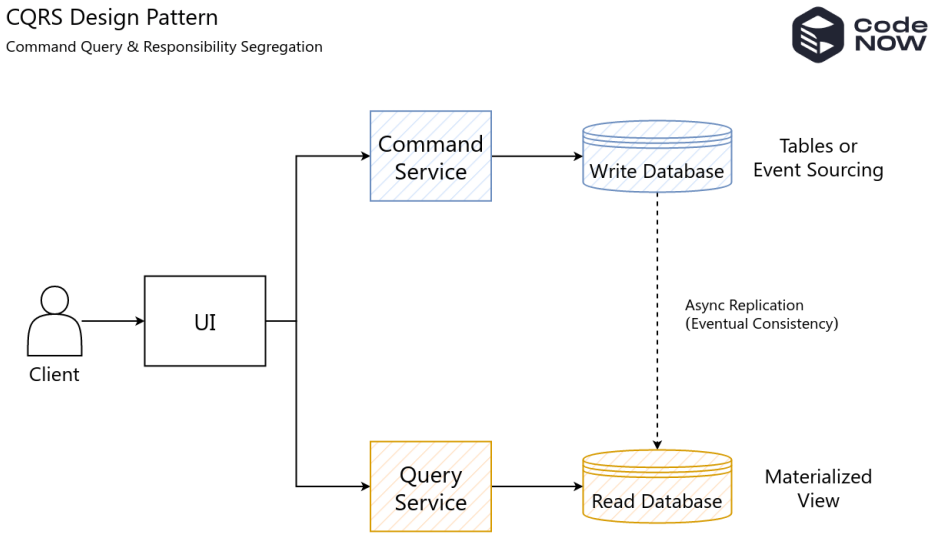
Model na čtení (read model) systému CQRS je zodpovědný za zpracování dotazů (query), které databázy nijak neupravují. Pouze čtou data z modelu na čtení. Tento model je navržen tak, aby optimalizoval výkonnost a škálovatelnost čtení. Data zde proto mohou být uložena v jiném formátu než v modelu pro zápis. Pro optimalizaci ke čtení se může použít například ukládání dat do mezipaměti (cache). Důležitou vlastností je, že pro jeden model zápisu lze vytvořit více modelu čtení.

V systému CQRS mohou modely čtení a zápisu používat různé mechanismy ukládání dat. Například model zápisu může používat relační databázi, zatímco model čtení může používat databázi NoSQL nebo specializované úložiště dat optimalizované pro přístup k dotazům.

K synchronizaci modelů čtení a zápisu používá systém CQRS architekturu řízenou událostmi. Model zápisu generuje události vždy, když je zpracován pří-

kaz a aktualizován stav systému. Model čtení se k těmto událostem přihlašuje a podle nich aktualizuje svá data.[45]

Základní myšlenka CQRS je jednoduchá, ale nutnost vypořádat se s informacemi, které mohou mít více reprezentací může vést ke složitějšímu návrhu aplikace, zejména pokud zahrnuje vzor Event sourcing. Model čtení musí být synchronizován často, aby se v něm odrážely změny v modelu zápisu. Jinak by se mohlo stát, že uživatel zadá požadavek na data, která nebyla aktualizována dostatečně rychle.[32]



Obrázek 6.1: Návrhový vzor CQRS.[15]

6.0.2 Event sourcing

Event sourcing je návrhový vzor, který se zaměřuje na ukládání událostí (events) jako hlavního zdroje stavu aplikace. Namísto uchovávání aktuálního stavu objektu nebo entity se všechny změny objektu nebo entity zaznamenávají jako série událostí (events) v pořadí v jakém nastaly. [45] Stav objektu nebo entity se pak určuje přehráváním těchto událostí v posloupnosti. Tento přístup poskytuje historii všech změn provedených v objektu nebo entitě a tím může zlepšit auditovatelnost tím, že poskytuje podrobnou historii všech událostí ke kterým došlo. Také na rozdíl od CRUD modelu nabízí možnost rekonstruovat stav aplikace v libovolném okamžiku bez dalších podpůrných mechanismů.

Při použití Event sourcing neexistují destruktivní operace update nebo delete. Události jsou neměnné a lze je ukládat pomocí operace spočívající v pouhém připojení.

Jedním z problémů při získávání informací o událostech je, že rekonstrukce aktuálního stavu subjektu na základě velkého počtu událostí může trvat dlouho. Existuje proto několik strategií, které se používají k optimalizaci jako je například používání snapshotů. Namísto přehrávání všech události pokaždé, když je potřeba znát aktuální stav entity, se mohou pravidelně pořizovat

snapshoty stavu entity. Když je potřeba zjistit aktuální stav entity, tak je možné začít s nejnovějším snapshotem a poté přehrát pouze události, které od tohoto snapshotu nastaly. [33]

6.0.3 CQRS a Event sourcing

CQRS a Event sourcing se často používají společně, protože se v několika ohledech doplňují. CQRS poskytuje způsob, jak oddělit zpracování příkazů (commands) a dotazů (queries) v systému, zatímco Event Sourcing poskytuje způsob, jak zachytit každou změnu v systému jako posloupnost událostí. [32]

Jednou z hlavních výhod kombinace CQRS a Event sourcing je lepší konzistence dat. Event sourcing zachycuje každou změnu v systému jako samostatnou událost a poskytuje úplný záznam všech změn ke kterým došlo. Použitím CQRS k oddělení příkazů a dotazů mohou vývojáři zajistit, že dotazy budou vždy volány z konzistentního pohledu na data.

Další výhodou kombinace CQRS a Event Sourcing je lepší škálovatelnost. Oddělením příkazů a dotazů umožňuje CQRS různým složkám systému škálovat nezávisle. Zachycením každé změny v systému jako posloupnosti událostí poskytuje Event sourcing mechanismus pro přehrávání událostí v případě selhání systému nebo jiných problémů, což může pomoci zlepšit odolnost proti chybám.

CQRS a Event sourcing přinášejí do systému další komplexitu, která může ztížit jeho vývoj, protože zahrnuje návrh a vytvoření samostatných modelů pro operace čtení a zápisu a také vývoj mechanismů pro uchovávání událostí a rekonstrukci stavu systému.

Protože systémy využívající Event sourcing jsou navrženy tak, aby rekonstruovaly aktuální stav systému přehráním všech událostí od počátku času, může dojít k určitému zpoždění mezi okamžikem, kdy dojde k operaci zápisu a okamžikem, kdy se tato změna projeví v modelu čtení. Toto zpoždění může mít za následek případnou nekonzistenci, kdy přečtený model nemusí odrážet nejaktuálnější stav systému.

6.0.4 CQRS a Event sourcing v tomto systému

Závěrem lze říci, že ačkoli CQRS a Event sourcing mají při vývoji moderního softwaru řadu výhod, existuje také několik potenciálních nevýhod, které je třeba vzít v úvahu při rozhodování, zda tyto vzory v konkrétním systému implementovat, či nikoli.

Po pečlivém zvážení výhod a nevýhod CQRS a Event sourcing ve vyvíjeném systému bylo po prodiskutování s mým školitelem rozhodnuto, že zvýšená komplexita těchto vzorů převažuje nad jejich potenciálními výhodami a proto se CQRS a Event sourcing do tohoto systému nebude implementovat. Tento systém je totiž relativně jednoduchý bez nějaké složité byznysové logiky a použití tradiční architektury CRUD (create, read, update, delete) je plně dostačující.

V případě, že by byla potřeba optimalizovat tento systém, tak by se nabízelo využít ukládání dat do mezipaměti (cache). Využít by se mohl například Redis

[44], který je také podporován platformou CodeNOW (viz 2.2.3). Při použití jako mezipaměť může Redis výrazně zvýšit výkon systému tím, že zkrátí dobu potřebnou k přístupu k často používaným datům. Redis ukládá data do paměti, což umožňuje rychlé operace čtení a zápisu. Při požadavku Redis zkontroluje, zda již existuje v mezipaměti. Pokud ano, data se vrátí z mezipaměti, čímž se ušetří čas a zdroje, které by byly potřeba k načtení dat z pomalejšího datového úložiště, například databáze. Redis také poskytuje funkce, jako je expirace, která umožňuje automatické odstranění dat z mezipaměti po uplynutí nastavené doby. To pomáhá zabránit tomu, aby mezipaměť byla zastaralá a spotřebovávala příliš mnoho paměti. [44]

Kapitola 7

Další vylepšení systému nad rámec zadání

V této kapitole budou popsána vylepšení, které nebyly definovány v zadání bakalářské práce.

7.1 QR kód

Uživatelé jsou automaticky upozorňováni na platby přes e-mail. V e-mailu se nachází tyto informace:

- Předmět platby
- Číslo bankovního účtu
- Částka
- Variabilní symbol
- Datum splatnosti
- E-mail na koho se obrátit v případě dotazů

Pro zaplacení musel uživatel přepsat číslo bankovní účtku, částku, variabilní symbol a předmět platby. Pro zjednodušení platby je nyní do e-mailu přidán i QR kód, který po naskenování mobilem tyto parametry platby vyplní v bankovní aplikaci. To může ušetřit uživateli nějaký čas s opisováním údajů a hlavně eliminuje chybu napsání některého údaje špatně. Tento QR kód je vytvořen za pomocí knihovny ZXing ("Zebra Crossing") [36] v notifikační mikroslužbě.

Kapitola 8

Testování systému

V této kapitole představím jakým způsobem byly rozšíření systému testovány. Pro testování aplikace byly zvoleny 4 typy testů: statické testy, unit testy, integrační testy a smoke testy. Tato bakalářská práce byla vedena agilním způsobem a každá nová verze systému byla nasazována přímo na platformě CodeNOW (viz 2.3.8).

8.1 Statické testy

Statické testy slouží k analýze kódu před spuštěním kódu. Statické testy zahrnují zkoumání zdrojového kódu s cílem identifikovat potenciální problémy jako jsou chyby v kódování (bad practices), bezpečnostní zranitelnosti a duplikace kódu. Pro tyto testy byl použitý nástroj SonarQube[48], který je podporován platformou CodeNOW. Pokud byla nalezena nějaká potenciální chyba nástrojem SonarQube, tak byla opravena.

8.2 Unit testy

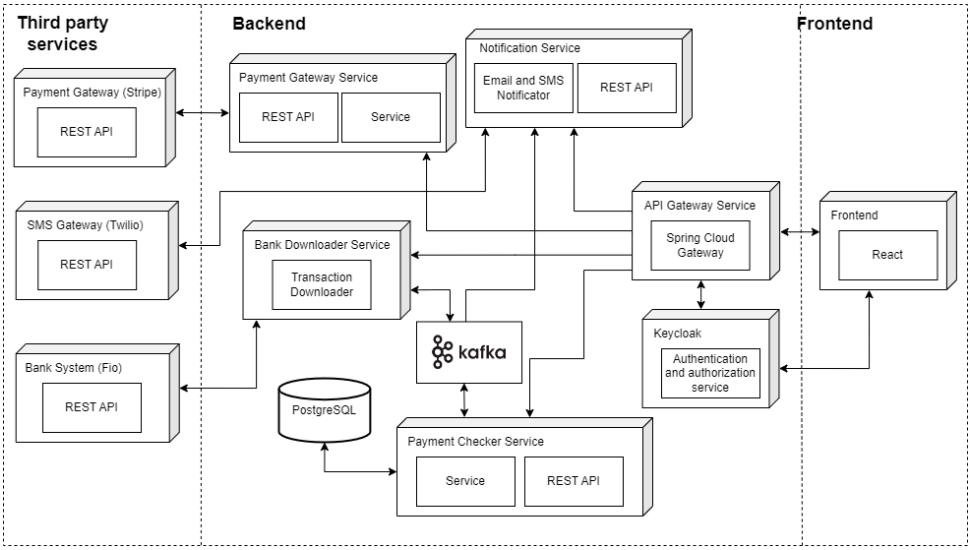
Unit testy slouží k ověření správnosti jednotlivých částí systému. Tento typ testování je plně automatizovaný a pro jeho provodení byl využit populární framework JUnit. [58] Jeho použití umožňuje snadné definování a spouštění testovacích scénářů.

Testovány byly nově přidané funkcionality. Jednou z těchto funkcionalit bylo správné vytvoření QR kódu při odesílání emailu a také odesílání SMS v mikroslužbě notificator. Další důležitou částí testování bylo vytvoření a aktualizace plateb v mikroslužbě platební brány (payment-gateway). Tímto způsobem byla ověřena správná manipulace s platbami, včetně jejich vytvoření a aktualizace.

8.3 Integrační testy

Jelikož se systém rozšířil o platební bránu Stripe, API Gateway a SMS bránu Twilio, tak bylo potřeba otestovat zda-li fungují jednotlivé části systému dohromady. Tyto rozšíření lze vidět na obrázku 8.1. Bylo ověřeno správné

sestavení požadavků a testování endpointů pro kontrolery. Pro testování API byl používán nástroj Postman [40]. V něm jsem vytvořil kolekci dotazů s různými daty. Po dotazu jsem kontroloval správnost chování aplikace a případně opravoval chyby.



Obrázek 8.1: Diagram architektury systému po implementaci všech změn a úprav.

8.4 Smoke testy

Smoke testy se používají používají k ověření hlavních funkcí systému, který nebývají příliš často upravovány. Tyto testy byly prováděny po nasazení nové verze systému manuálně mnou pod kontrolou mého školitele, aby se ověřilo, že jsou všechny hlavní funkce systému stále funkční a zda je aplikace připravena pro další fázi vývoje.

Kapitola 9
Závěr

Cílem této bakalářské práce bylo rozšířit zadaný systém Kontrola plateb o nové funkce a implementaci několika vzorů typických pro architekturu mikroslužeb.

Byla přidaná nová mikroslužba API Gateway (viz kapitola 3). Její hlavní odpovědností je směřování požadavků do mikroslužeb. Je to tedy jediný vstupní bod do systému. Dále zabezpečuje mikroslužby s JWT (JSON Web Token) získaný ze služby Keycloak. Pokud by došlo k dalším rozšíření systému, tak by API Gateway mohla být využita k agregaci požadavků. To znamená, že pokud by bylo potřeba odeslat několik klientských požadavků, které cílí na více interních mikroslužeb, tak je sdružit do jednoho požadavku klienta.

Další nově přidanou mikroslužbou je platební brána (viz kapitola 4). Tato mikroslužba je integrována s platební bránou Stripe. Její prozatimní funkce je k možnosti podpoření (donate) vyvíjeného systému.

Tento systém nově dovoluje možnost notifikovat plátce i přes SMS. Pro tuto funkcionalitu byla zvolena SMS brána Twilio (viz 5.2.6). V případě, že je číslo neplatné a nebo se nepovede zprávu doručit, tak je uživatel na chybu upozorněn.

V kapitole 6 jsou rozebrány vzory CQRS a Event sourcing. CQRS je architektonický vzor, který odděluje operace zápisu a čtení do samostatných modelů. Tím je možné mít model zápisu a k němu několik modelů na čtení, které jsou optimalizované pro dotazování. Event sourcing uchovává stav aplikace jako uspořádanou posloupnost událostí, čímž nabízí možnost obnovit stav aplikace v libovolném okamžiku. Po zvážení výhod a nevýhod, které by tyto vzory přinesly do tohoto systému bylo po prodiskutování s mým školitelem rozhodnuto, že se nebudou implementovat.

Systém byl průběžně testován (viz kapitola 8), dokumentován a nasazován na platformě CodeNOW(viz 2.3.8). Také byla pro jednotlivé části udělána rešerše pro výběr těch nejvhodnějších technologií.

Všechny body ze zadání bakalářské práce byly splněny s výjimkou vzorů CQRS a Event sourcing, které se po zvážení výhod a nevýhod neimplementovaly. Oproti původnímu zadání byl systém rozšířen o odesílání QR kódu v e-mailu (viz 7.1), čímž může dojít ke zjednodušení platby pro plátce a eliminuje chybu napsání některého z údajů špatně.

9.1 Budoucnost systému

V rámci této bakalářské práce byly identifikovány další možné vylepšení, které by se mohli v budoucnu přidat. Mezi nové případy užití patří, aby uživatel mohl vytvořit hromadně platební předpisy z CSV.

Dále by bylo dobré zvážit možnost, aby plátce mohl nastavit trvalý příkaz pokud se jedná o měsíční příspěvěk. To by znamenalo pro uživatele, že by mohl vytvořit platební balíček na základě již existujícího platebního balíčku.

Poté by byla dobrá funkce, že v případě pokud by tento systém používala například základní škola, tak pokud by vytvořila novou platbu pro rodiče svých studentů, tak v tuto chvíli je možné odeslat e-mail pouze jednomu plátci (tedy jednomu rodiči) o platbě. Kdyby rodiče byli rozvedení, tak odpovědnost spadá pouze na jednoho z nich. Z toho důvodu by bylo vhodné umožnit plátcům mít více e-mailových adres, na které by bylo odesláno upozornění o platbě, a měli by tak možnost platbu zaplatit na půl.

Mezi další případná rozšíření by mohlo být upravení práv pro odesílání SMS závisle na dostupném kreditu uživatele. V současné době je uživatel schopen odesílat SMS zasílat neomezeně, avšak platí se za každou odeslanou SMS (viz 5.2.6). Tyto SMS jsou účtovány prostřednictvím propojeného Twilio účtu. Jednou z možností by mohlo být u každého uživatele uchovávat informaci o jeho "kreditu". Tento kredit by mohl být doplňován pomocí platební brány a při odesílání SMS by se kredit uživatele automaticky snižoval o odpovídající částku.

Literatura

[1] 13 contributors. What is Azure Application Gateway?, 20 October 2022. Dostupné z: https://learn.microsoft.com/en-us/azure/application-gateway/overview.

[2] Amazon Pay. AMAZON PAY FOR BUSINESS. Dostupné z: https://pay.amazon.com/business.

[3] Amazon Web Services. Amazon API Gateway. Dostupné z: https://aws.amazon.com/api-gateway/.

[4] Amazon Web Services. Amazon SNS. Dostupné z: https://aws.amazon.com/sns/.

[5] Amazon Web Services. Amazon SNS pricing. Dostupné z: https://aws.amazon.com/sns/pricing/.

[6] Apache Kafka. INTRODUCTION. Dostupné z: https://kafka.apache.org/intro.

[7] APEXX. Payment Gateway Security, 27 Jun, 2022. Dostupné z: https://apexx.global/blog/payment-gateway-security.

[8] Atatus. A Guide for Choosing the Best API Gateway, 19 September 2022. Dostupné z: https://www.atatus.com/blog/a-guide-for-choosing-the-best-api-gateway/.

[9] Auth0. Introduction to JSON Web Tokens. Dostupné z: https://jwt.io/introduction.

[10] Barbara Cook. Braintree vs PayPal: How Does Braintree Stack Up for Payment Processing? Dostupné z: https://tipalti.com/en-eu/braintree-vs-paypal/.

[11] Braintree. Boost Revenue with a Global Payments Partner. Dostupné z: https://www.braintreepayments.com/cz.

[12] CodeNOW. CodeNOW Documentation, 2022. Dostupné z: https://docs.codenow.com/.

[13] CodeNOW. Coding Examples, 2022. Dostupné z: https://docs.codenow.com/docs/coding-examples#list-of-tutorials.

[14] CodeNOW. Ready-made technological services, 2022. Dostupné z: https://docs.codenow.com/docs/ready-made-services.

[15] CodeNOW. Using CQRS - Command and Query Responsibility Segregation Pattern, Copyright © 2022. Dostupné z: https://docs-codenow-dev.stxcn.codenow.com/articles/cqrs.

[16] Dhara Tuvar. List of the Best SMS Gateways to Use in 2023: Detailed Comparison of Top SMS APIs. Dostupné z: https://meetanshi.com/blog/best-sms-gateway-providers/.

[17] emerchantpay. Payment gateway – what is it and how does it work?, 7 December 2022. Dostupné z: https://www.emerchantpay.com/insights/what-is-a-payment-gateway-and-how-does-it-work.

[18] Express Gateway. Introduction. Dostupné z: https://www.express-gateway.io/about/.

[19] From Wikipedia, the free encyclopedia. Apache ZooKeeper, 23.12. 2022. Dostupné z: https://en.wikipedia.org/wiki/Apache_ZooKeeper.

[20] GatewayAPI. Ultimate Guide to Choosing an SMS Gateway, 7 March 2022. Dostupné z: https://gatewayapi.com/blog/ultimate-guide-to-choosing-an-sms-gateway/.

[21] Grafana. Dashboard anything. Observe everything. Dostupné z: https://grafana.com/grafana/?plcmt=footer.

[22] Graham Charlton. Pros and cons of 11 payment gateways for web, mobile & apps. Dostupné z: https://dotknowledge.uk/articles/view-article/pros-and-cons-of-11-payment-gateways-for-web-mobile-apps.

[23] IBM Cloud Education. What Are API Gateways?, 31 March 2022. Dostupné z: https://www.ibm.com/cloud/blog/api-gateway.

[24] Pazdera Jiří. Cloud native aplikace pro kontrolu plateb. B.S. thesis, České vysoké učení technické v Praze. Vypočetní a informační centrum., 2021. Dostupné z: https://dspace.cvut.cz/handle/10467/94732.

[25] Kanika Sharma. SMS Gateway: How it Works & How to Get Started?, 28 March 2023. Dostupné z: https://www.revesoft.com/blog/sms-platform/sms-gateway/.

[26] Karl Kalvik. What Is An SMS Gateway: The Core Facts You Need To Know. Dostupné z: https://messente.com/blog/what-is-sms-gateway.

[27] Kathleen Casey. Explore 6 popular API gateway tools and how to choose one, 8 February 2021. Dostupné z: https://www.techtarget.com/searchapparchitecture/feature/A-feature-rundown-of-6-popular-API-gateway-tools.

[28] Keycloak. Open Source Identity and Access Management. Dostupné z: https://www.keycloak.org.

[29] Kong. Kong Gateway. Dostupné z: https://docs.konghq.com/gateway/latest/.

[30] KrakenD. Introduction to KrakenD Community Edition. 26 November 2018. Dostupné z: https://www.krakend.io/docs/overview/.

[31] MessageBird. Simple, flexible pricing. Dostupné z: https://messagebird.com/pricing.

[32] Microsoft. CQRS pattern. Dostupné z: https://learn.microsoft.com/en-us/azure/architecture/patterns/cqrs.

[33] Microsoft. Event Sourcing pattern. Dostupné z: https://learn.microsoft.com/en-us/azure/architecture/patterns/event-sourcing.

[34] npm. About semantic versioning | npm Docs. Dostupné z: https://docs.npmjs.com/about-semantic-versioning.

[35] npm. npm, 2023. Dostupné z: https://www.npmjs.com.

[36] Pankaj. Java QR Code Generator - zxing example, August 3 2022. Dostupné z: https://www.digitalocean.com/community/tutorials/java-qr-code-generator-zxing-example.

[37] PayPal. Payflow Payment Gateway. Dostupné z: https://www.paypal.com/us/webapps/mpp/payflow-payment-gateway.

[38] PostgreSQL. About. Dostupné z: https://www.postgresql.org/about/.

[39] PostgreSQLTutorial. Install PostgreSQL on Windows, 2022. Dostupné z: https://www.postgresqltutorial.com/postgresql-getting-started/install-postgresql/.

[40] Postman. What is Postman? Dostupné z: https://www.postman.com/product/what-is-postman/.

[41] React. React - A Javascript library for building user interfaces, 2023. Dostupné z: https://reactjs.org/.

[42] Red Hat. What is Grafana??, 13.5. 2022. Dostupné z: https://www.redhat.com/en/topics/data-services/what-is-grafana.

[43] Red Hat. What is Kubernetes?, 27.3. 2020. Dostupné z: https://www.redhat.com/en/topics/containers/what-is-kubernetes.

[44] Redis. Introduction to Redis. Dostupné z: https://redis.io/docs/about/.

[45] Chris Richardson. *Microservices patterns: with examples in Java*. Manning Publications, 2018. Dostupné z: https://learning.oreilly.com/library/view/microservices-patterns/9781617294549/.

[46] SEVITECH CZ. Řešíte otázku Cloud vs On-premise řešení?, 24. March 2020. Dostupné z: https://www.sevitech.cz/resite-otazku-cloud-vs-on-premise-reseni/.

[47] Skrill. Secure online payments for your business. Dostupné z: https://www.skrill.com/en/business/.

[48] SONAR. ABOUT US. Dostupné z: https://www.sonarsource.com/company/about/.

[49] Spring. Spring Boot. Dostupné z: https://spring.io/projects/spring-boot.

[50] Spring Framework. Spring Cloud Gateway. Dostupné z: https://docs.spring.io/spring-cloud-gateway/docs/current/reference/html/#gateway-starter.

[51] Stripe. PaymentIntents. Dostupné z: https://stripe.com/docs/payments/payment-intents.

[52] Stripe. Payments infrastructure for the internet. Dostupné z: https://stripe.com/en-cz/global.

[53] Synology. Jak se používá server SMTP služby Gmail k odesílání e-mailů pro systém SRM?, 21.9. 2022. Dostupné z: https://kb.synology.com/cs-cz/SRM/tutorial/How_to_use_Gmail_SMTP_server_to_send_emails_for_SRM.

[54] Telesign. Engage your customers globally with SMS. Dostupné z: https://www.telesign.com/products/sms-api.

[55] Telesign. Simple, flexible pricing. Dostupné z: https://www.telesign.com/pricing/sms-and-voice.

[56] Telnyx. Messaging API pricing. Dostupné z: https://telnyx.com/pricing/messaging.

[57] Telnyx. SMS API. Dostupné z: https://telnyx.com/products/sms-api.

[58] The JUnit Team. About, Copyright © 2023. Dostupné z: https://junit.org/junit5/.

[59] Twilio. REST API: Auth Token. Dostupné z: https://www.twilio.com/docs/iam/api/authtoken.

[60] Twilio. SMS Gateway. Dostupné z: https://www.twilio.com/docs/glossary/what-is-a-sms-gateway.

[61] Twilio. SMS Pricing. Dostupné z: https://www.twilio.com/sms/pricing/cz.

[62] Twilio. String Identifier (SID). Dostupné z: https://www.twilio.com/docs/glossary/what-is-a-sid/.

[63] Twilio. Twilio Customer Engagement Platform. Dostupné z: https://www.twilio.com/en-us/.

[64] TypeScript. TypeScript: JavaScript With Syntax For Types, 2023. Dostupné z: https://www.typescriptlang.org/.

[65] Vonage. SMS API pricing. Dostupné z: https://www.vonage.co.uk/communications-apis/sms/pricing/.

[66] Xperience. Cloud Vs On Premise Software: Which Is Best For Your Business?, 24 February 2017. Dostupné z: https://www.xperience-group.com/news-item/cloud-vs-on-premise-software/.

[67] Zoho Books Team. How to Choose the Best Payment Gateway for Your Business. Dostupné z: https://www.zoho.com/books/articles/how-to-choose-the-right-payment-gateway.html.

Příloha A

Obsah elektronické přílohy

K této bakalářské práci přikládám následující soubory:

- api-gateway-service.zip - zdrojové kódy mikroslužby API Gateway
- bank-downloader-service.zip - zdrojové kódy mikroslužby stahující transakce z banky
- frontend.zip - zdrojové kódy frontendu
- notificator-service.zip - zdrojové kódy notifikační mikroslužby
- payment-checker-service.zip - zdrojové kódy mikroslužby na párování plateb
- payment-gateway-service.zip - zdrojové kódy mikroslužby platební brána
- postmanCollection.json - kolekce testů do nástroje Postman