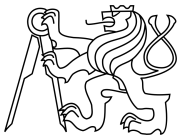
ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ



ZADÁNÍ DIPLOMOVÉ PRÁCE

Název:	Mobilní aplikace pro provádění Peer-to-Peer plateb
Student:	Bc. David Kukačka
Vedoucí:	Ing. Pavel Krejčí
Studijní program:	Informatika
Studijní obor:	Webové a softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce letního semestru 2017/18

Pokyny pro vypracování

Definujte pojem peer-to-peer (P2P) platba, popište existující řešení pro provádění P2P plateb. Představte koncept Open Banking a jeho možné využití. Popište obecné architektury využívané pro mobilní aplikace. Navrhněte koncept a architekturu mobilní aplikace pro P2P platby. Analyzujte a navrhněte způsob komunikace mezi aplikací a Open Banking API. Porovnejte využití protokolů SOAP a REST pro komunikaci v rámci aplikace. Ve vhodném frameworku implementujte prototyp serverové části aplikace a její komunikaci s klientskou částí aplikace. Určete náklady a harmonogram vývoje projektu. Stanovte metriky pro měření úspěšnosti projektu.

Seznam odborné literatury

Dodá vedoucí práce.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

prof. Ing. Pavel Tvrdík, CSc.
děkan

V Praze dne 3. ledna 2017

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra softwarového inženýrství

Diplomová práce

Mobilní aplikace pro provádění Peer-to-Peer plateb

Bc. David Kukačka

Vedoucí práce: Ing. Pavel Krejčí

3. května 2017

Poděkování

Na úvod bych rád poděkoval svým rodičům za podporu, jak materiální, tak psychickou, kterou mi v rámci mého dosavadního studia poskytovali. Dále bych jim rád poděkoval za asistenci při tvorbě mé závěrečné práce, především za způsob, jakým se zhostili rolí testera a korektora. Rád bych také poděkoval Janu Fluksovi za pomoc při tvorbě diplomové práce, konkrétně za doporučení zajímavých nástrojů pro implementaci a dobré rady do začátku. Dále bych rád poděkoval Ivanu Mouchovi, který ochotně odpovídal na mé technicky směrované dotazy stran bankovních systémů a připravované legislativní úpravy. V poslední řadě bych rád poděkoval vedoucímu této práce, Pavlu Krejčímu, který mi umožnil zpracovat toto zajímavé téma a na kterého jsem se mohl vždy obrátit, ztratil-li jsem správný směr. Bez pomoci vás všech bych jen stěží dokázal tuto práci dotáhnout až ke zdárnému konci.

Prohlášení

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

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 3. května 2017 .....

Odkaz na tuto práci

Kukačka, David. *0.0.0.* Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2017.

Abstrakt

Tato práce se zabývá možností využití evropské směrnice o platebních službách PSD2 pro implementaci mobilní aplikace, sloužící k provádění plateb na kontakt. Práce představuje návrh aplikace, která umožní abstrahovat uživatele od čísel bankovních účtů a vytvářet nové transakce na základě telefonních čísel, uložených v kontaktním seznamu zařízení. V rámci práce jsme provedli systémovou specifikaci a následně vytvořili funkční prototyp aplikace, který demonstruje definovanou funkcionalitu. Na základě vytvořeného návrhu a prototypu jsme následně stanovili harmonogram realizace produkční verze aplikace společně s odhadovanými náklady. Práce se též krátce věnuje marketingové strategii uvedení produktu na trh a metrikami pro měření úspěšnosti projektu. Dohromady tak práce poskytuje podklady pro rozhodování o realizaci projektu implementace aplikace pro provádění Peer-to-Peer plateb. Hlavním přínosem práce je systémová specifikace, popisující návrh systému a prototyp, na kterém je možné požadovanou funkcionalitu demonstrovat.

Klíčová slova

PSD2, P2P placení, SOAP, REST, Návrh mobilní aplikace, hybridní mobilní aplikace, Apache Cordova

Abstract

This diploma thesis introduces second Payment Services Directive (PSD2), designed by European Union, as a possible mean for implementing mobile application for Peer-to-Peer payments. Aforesaid application abstracts users from bank account numbers and enables them to submit bank transactions based on phone number stored in contact list of mobile device. Within diploma thesis we formulated system specification of the application and implemented functional prototype, in order to demonstrate applications behavior. Furthermore, this thesis examines costs of implementing aforesaid application and suggests schedule of given tasks. Thesis also introduces general marketing strategy for product releas as well as defines success metrics. The main benefit of this thesis lies in created system specification for Peer-to-Peer application and prototype, which demonstrates required functionality.

Keywords

PSD2, P2P payments, SOAP, REST, Mobile application design, hybrid mobile application, Apache Cordova

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Dne 13. ledna 2016 vstoupila v platnost nová evropská směrnice o platebním trhu PSD2. Cílem této legislativy je umožnění vývoje jednotného integrovaného trhu pomocí standardizace pravidel, posílení bezpečnosti systému a zajištění vysoké úrovně hospodářské soutěže. S posledním zmíněným bodem je úzce spjata povinnost poskytovatelů bankovních služeb zpřístupnit svá systémová rozhraní třetím stranám. Přes toto veřejně vystavené rozhraní budou moci certifikované třetí strany zadávat platební příkazy a provádět další operace, týkající se bankovních účtů[5]. Přístup k těmto rozhraním bude podmíněn bezpečnostní prověrkou pro získání příslušné certifikace a manipulace s bankovními účty bude vyžadovat souhlas majitele.

Tato právní úprava na trhu platebního kontaktu vytváří zcela nový prostor pro poskytování služeb spojených s bankovními účty klientů bankovních institucí. Dominantní postavení bank v poskytování těchto služeb tak bude značně otřeseno a lze očekávat příchod nových subjektů, které na nové legislativě vytvoří své podnikání[5].

Ve své diplomové práci se zabývám možností vytvoření mobilní aplikace pro provádění Peer-to-Peer plateb, tedy odeslání peněz jiné osobě pouze na základě znalosti kontaktu této osoby. Myšlenka této funkcionality se opírá o skutečnost, že odeslání finanční částky jiné osobě je v současné chvíli relativně složitý proces, ve kterém jsou obě strany identifikovány pomocí čísel bankovních účtů. Zpřístupněním bankovních rozhraní třetím stranám je možné vybudovat aplikaci, která bude provádět překlad telefonního čísla na číslo bankovního účtu. Uživatel je tak zcela odstíněn od složitých čísel bankovních účtů a k zadávání transakcí stačí telefonní číslo uložené v kontaktním seznamu zařízení.

Diplomová práce je zpracovaná na základě skutečného projektu, realizovaného pro nejmenovanou banku. Práce se zabývá analýzou uživatelských požadavků a definuje návrh aplikace pro provádění Peer-to-Peer plateb. Podle tohoto návrhu je v práci následně implementován prototyp aplikace, který umožňuje demonstraci definovaných funkcionalit na cílovém zařízení. V závěru

se práce zaměřuje na stanovení nákladů a harmonogramu projektu implementace produkční verze aplikace a definuje metriky pro měření jeho úspěšnosti.

Kapitola 1

Kontext a dopady směrnice PSD2

Úvod teoretické části této diplomové práce představuje evropskou směrnici PSD2 v rozsahu nutném pro později navrhovanou aplikaci. S touto směrnicí jsou velmi úzce spojeny pojmy Open API a Open Banking. Tato kapitola si klade za cíl představit tyto termíny a zhodnotit jejich možný vliv na bankovní sektor.

1.1 Kontext evropské směrnice PSD2

Od 13. ledna 2018 musí členské státy evropské unie implementovat do své národní legislativy evropskou směrnici 2015/2366/EU, známou také jako PSD2. Směrnice vychází ze svého předchůdce, evropské direktivy PSD z roku 2010, která již není dostatečně aktuální pro progresivně se transformující platební trh. Podíváme-li se na celosvětový počet bezhotovostních převodů, jen v roce 2014 došlo oproti roku 2013 k nárůstu o 9%. Celkový počet bezhotovostních transakcí tak byl 338,8 miliard v hodnotě 883,4 miliard USD[6].

Tento trend digitalizace navíc zatím nejeví žádné známky zpomalení a stávající odhady předpokládají do roku 2018 nárůst o dalších 19% ve srovnání s rokem 2014. Tato rostoucí míra digitalizace je dána především následujícími faktory [6]:

1. Šířením moderních technologických zařízení, mezi které patří především smartphony a tablety.

2. Změnou návyků spotřebitelů, kteří rádi používají efektivnější platební metody.

3. Marketingovými strategiemi, zaměřenými na vyšší využití elektronických zařízení ke shromáždování informací o chování zákazníků.

Mezi hlavní přínosy této evropské směrnice patří [6]:

- posílení ochrany spotřebitele,
- umožnění vývoje nových platebních řešení,
- regulace nových účastníků platebního trhu,
- jednotné poplatky za platby kartou v souladu s regulací mezibankovních poplatků,
- zvýšení úrovně hospodářské soutěže,
- obecné zvýšení efektivity prostřednictvím standardizace infrastruktury.

Jedním z klíčových bodů této direktivy je povinnost bank zpřístupnit registrovaným třetím stranám API[1], které bude umožňovat přístup k bankovním účtům klientů a platebním systémům bank. Zpřístupnění takového rozhraní otevře na trhu zcela nové možnosti poskytování finančních služeb a bude tak mít výrazný dopad na stávající podobu bankovního sektoru.

1.2 Koncept API

Termín API je možné definovat jako softwarové rozhraní, které umožňuje jedné aplikaci využívat funkcionality jiné aplikace. Nejedná se však o pouhé synonymum slova rozhraní. API lze chápat spíše jako architektonický přístup, který se zaměřuje na škálovatelnost, znuvupoužitelnost a bezpečnost rozhraní s cílem dosažení efektivní komunikace mezi systémy[7]. Z technického pohledu musí každé API definovat:

1. Datový přenos – v jakém formátu jsou data bezpečně přenášena mezi systémy. Většina API jako transportní vrstvu využívá HTTP/HTTPS protokol.
2. Formát předávaných dat – v jakém formátu jsou posílána data mezi systémy. Většina API pracuje se dvěma standardními formáty: XML, JSON. Formát XML poskytuje více možností nežli častěji používaný JSON, který je ceněný převážně díky rychlejší výměně a jednoduší strojové čitelnosti.
3. Přístup k datům – definuje, kdo má k jakým datům přístup. Mezi nejčastěji používané standardy autentizace patří SAML a OAuth 2.0.
4. Návrh API – jakým způsobem je API navrženo. Nejčastější principy návrhu API jsou REST a SOAP. Porovnání obou přístupů k návrhu API je provedeno v sekci 3.3.

[1] Application Programming Interface je systémové rozhraní, které poskytuje programátorovi přístup k funkcionalitě daného systému.

V bankovním sektoru však definice technického rozměru API není dostačující pro vystavení bezpečného veřejného rozhraní pro komunikaci se třetími stranami. Pomocí veřejného bankovního API bude docházet k zadávání finančních transakcí a k přenosu citlivých osobních údajů. Z tohoto důvodu je nezbytné pracovat s další dimenzí API, která bude určovat, jak může být s daty nakládáno v závislosti, zda se například jedná o zapisovací, nebo čtecí operaci, zda se jedná o osobní údaje klienta, nebo zda se jedná o agregované údaje klientů. Pro takovéto API je proto nutné implementovat pokročilejší mechanismy kontroly. Z tohoto pohledu lze API pro bankovní sektor rozdělit na čtyři úrovně standardizace[7]:

1. Právní – tato úroveň definuje práva a povinnosti, které se vztahují na všechny zúčastněné strany.

2. Operační – na této úrovni je definovaná operační otázka API, pod kterou mimo jiné spadá výkonnost, dostupnost, nebo úroveň podpory.

3. Funkční – tato úroveň definuje funkcionalitu, kterou bude API uživatelům poskytovat.

4. Technická – technická úroveň API, která je podrobně rozepsaná v úvodu této sekce.

1.3 Open API

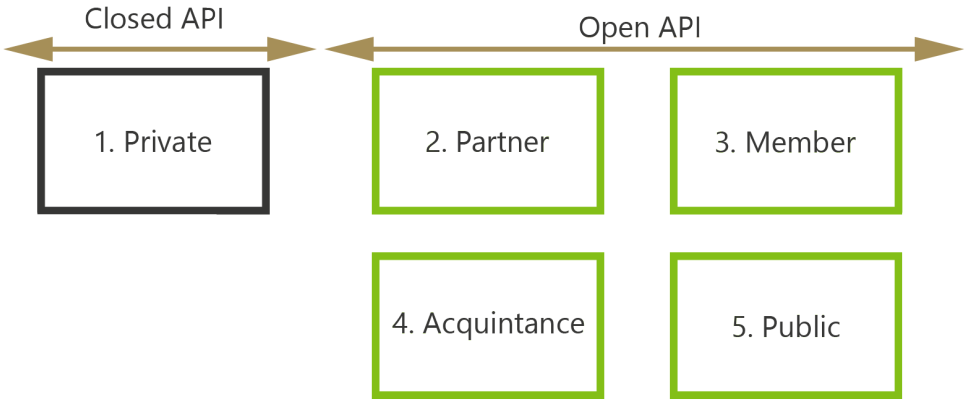
API umožňují bezpečný, kontrolovaný a efektivní přístup ke zdrojům nebo funkcionalitě systému. Pokud je k API možné přistupovat pouze zevnitř organizace, pak se jedná o takzvané Closed API (Uzavřené API). Naopak pokud k API mohou přistupovat také třetí strany, pak se jedná o Open API (Otevřené API). Převážná většina API se pak bude nacházet mezi těmito krajními body a jsme tak schopni definovat úroveň otevřenosti API[8]:

Private API — Jedná se o uzavřené API, které je dostupné pouze zevnitř vlastní organizace.

Partner API — API, které je zpřístupněné konkrétním partnerům na základě bilaterální dohody. Příkladem interakce může být výměna specifických dat mezi bankovním systémem a ERP systémem konkrétní firmy.

Member API — Typ API, které je zpřístupněno každému formálnímu členu určité komunity s přesně definovanými členskými pravidly.

Acquintance API — Tento typ je přístupný každému, kdo souhlasí s předem definovanými pravidly, nebo obecnými pravidly použití.



Obrázek 1.1: Rozdělení úrovní otevřenosti API.

Public API — Nejvíce otevřený typ, který je přístupný každému. Pro přístup je typicky nutná registrace, která slouží pro autentizaci konzumenta.

Rozdělení podle úrovně otevřenosti API je zachyceno na obrázku 1.1. Pod pojmem Open API tedy chápeme API, které je určitým způsobem otevřené třetím stranám.

V soukromém sektoru je na myšlence OPEN API postavena řada úspěšných modelů. Společnost Twitter například umožňuje vývojářům do vyvíjených aplikací použít část své hotové funkcionality, která získává data přes veřejné API Twitteru. Vývojář tak má zjednodušenou práci, Twitter naopak profituje rozšířením svého systému do nové aplikace a zvýšením provozu[8].

1.4 Open Banking

Open Banking je termín, který se neustále vyvíjí a pro který existuje více definic. Jedná se o způsob standardizace způsobu, jakým banky sdílejí svá data mezi sebou a jakým způsobem poskytují data třetím stranám jednotným a bezpečným způsobem. Open Banking lze také popsat jako evoluci bankovního sektoru způsobenou vývojem technologií [7]. V principu Open Banking cílí na umožnění klientům používat svá bankovní data v rámci aplikací, nabízených třetími stranami.

1.5 Dopady Open Bankingu

Zavedení Open API a s tím spojený rozvoj odvětví směrem k Open Bankingu s sebou přináší pro banky řadu rizik[9]:

1. Úspěšné zavedení Open Bankingu přináší riziko ztráty dominantní pozice poskytovatele finančních služeb. Lze předpokládat rostoucí trend přechodu klientů ke službám a aplikacím nabízeným FinTech společnostmi[2]. Bude tak docházet ke ztrátě kontaktu mezi klienty a bankami, což může vést ke snížení využívání produktů a služeb bank a v důsledku až ke ztrátě dominantní pozice jako poskytovatele finančních služeb.

2. Zavedení Open API je v bankovním sektoru relativně revoluční koncept, který s sebou však přináší také zásadní bezpečnostní hrozbu. Rizikem pro banku mohou být podvodné třetí strany, digitální útoky, nedovolené použití poskytovaných dat, nebo zneužití osobních údajů. Jakýkoliv bezpečnostní incident se negativně zobrazí do vnímání banky ze strany klientů.

3. Další výzvou pro banky je samotná technická implementace Open API a jeho začlenění do stávající infrastruktury. Bankovní systémy budou muset být schopny zpracovat zátěž plynoucí z vystaveného Open API a zároveň udržet svoji stávající výkonnost a dostupnost.

Podobně jako v soukromém sektoru, i v bankovním prostředí přináší koncept Open Bankingu řadu příležitostí pro samotné banky[9]:

1. Otevřením svého systému se bankám nabízí možnost zkvalitnění a rozšíření poskytovaných služeb. Banky tak například mohou využít data o klientech z ostatních bank a zpracovávat přesnější analýzy, lépe hodnotit klienty a následně jim nabízet své produkty.

2. Z důvodu uzavřenosti systémů nemohou v současné chvíli banky nabízet své produkty přes digitální platformy jiných bank nebo třetích stran. Společně s Open Banking však dojde k vytvoření prostředí s modelem standardizovaných sdílených služeb. Takto je možné vytvořit model, ve kterém si klient může navolit produkty z balíčků nabízených služeb, často od různých poskytovatelů – zkvalitnění poskytovaných služeb klientům.

[2]Financial Technology společnosti, které využívají nové technologie a inovace směrem k poskytování finančních služeb.

Kapitola 2

Koncept služby Peer-to-Peer platba

V přechozí kapitole je představena evropská direktiva PSD2 a její dopady na Open Banking a bankovní sektor. Tato kapitola navazuje na přechozí představení a blíže se věnuje možnosti využití zpřístupněných bankovních API pro poskytování služby Peer-to-Peer platba. V rámci kapitoly jsou představeny koncepty stávajících řešení této služby, včetně jejich nedostatků. V závěru kapitoly je představen koncept služby s využitím Open API, který cílí na zjednodušení a odstranění výše zmíněných nedostatků.

2.1 Peer-to-Peer služba

Termín Peer-to-Peer služby nemá přesnou literární definici, ze které by bylo možné v této práci vycházet. Samotný pojem Peer-to-Peer vychází z informačních technologií, přesněji z decentralizovaného síťového modelu. V takovémto síťovém modelu vystupuje každá stanice ve stejném postavení, na rozdíl od modelu klient–server. Termín Peer-to-Peer bychom tak mohli přeložit jako „rovný s rovným“[10]. Tento pojem byl následně přenesen do dalších odvětví, mimo jiné do ekonomického prostředí, kde značí interakci, ve které dvě strany interagují přímo bez jakéhokoliv prostředníka.

Spojení Peer-to-Peer platba pak chápeme jako službu, která umožňuje jednomu uživateli převést finance ze svého bankovního účtu na bankovní účet druhé osoby pomocí webové nebo telefonní aplikace[11]. V rámci této práce navíc termín Peer-to-Peer platby reprezentuje proces provedení bankovní transakce, který nevyžaduje po uživateli znalost čísla bankovního účtu protistrany.

Obrázek 2.1: Model P2P platby poskytovaný třetí stranou.

2.2 Stávající řešení Peer-to-Peer plateb

Před návrhem nové mobilní aplikace pro provádění Peer-to-Peer plateb je nutné se podívat na stávající řešení, která jsou na trhu k dispozici. Z hlediska celkového konceptu lze rozlišit dva základní přístupy[12]:

1. Aplikace třetí strany pro Peer-to-Peer platby.
2. Bankovní aplikace pro Peer-to-Peer platby.

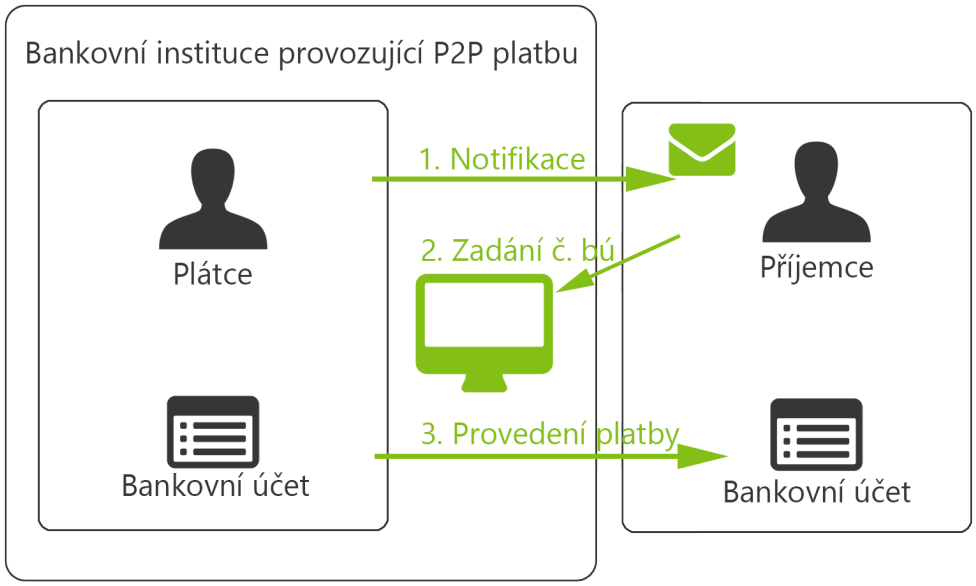
2.2.1 Aplikace třetí strany

Jedná se o model, ve kterém poskytovatelem služby není bankovní instituce. Uživatel si ve službě založí uživatelský účet, který následně zajistí bankovním účtem, nebo platební kartou. Jakmile je účet zajištěn, uživatel může pomocí této služby odesílat a přijímat platby od jakéhokoliv jiného uživatele přihlášeného k této službě.

Jednotliví uživatelé jsou v rámci této Peer-to-Peer služby identifikováni pomocí jednoznačného osobního údaje, kterým je zpravidla emailový kontakt. Pro provedení platby stačí znát emailový kontakt druhé strany a aplikace sama strhne částku z uživatelského účtu plátce a převede ji na uživatelský účet příjemce. Ten si může částku ze svého uživatelského účtu převést na svůj bankovní účet, což bývá zpravidla procentuálně zpoplatněno[12]. Výše popsaný model je zachycen na obrázku 2.1.

Typickým úspěšným provozovatelem tohoto modelu Peer-to-Peer platby je americká společnost PayPal. Mezi hlavní nevýhody tohoto modelu patří:

- Poplatky spojené s využíváním služby.
- Možnost zadávat platby pouze ostatním uživatelům služby.



Obrázek 2.2: Model P2P platby poskytovaný bankovní institucí.

2.2.2 Bankovní aplikace

Ve druhém modelu vystupuje jako poskytovatel služby bankovní instituce. Tato banka svým klientům nabízí aplikaci, která je spárovaná s klientovým účtem a která umožňuje uživatelům provádění Peer-to-Peer plateb.

Pro provedení platby uživatel vyplní kontaktní údaje druhé strany a příjemci je zaslána notifikace ohledně prováděné platby. V rámci notifikace je příjemce vyzván k otevření webového formuláře, do kterého vyplní číslo bankovního účtu, na který si přeje částku převést. Pro lepší zabezpečení je tento krok často doplněn autentizací příjemce. Jakmile jsou údaje vyplněny, bankovní aplikace převede částku z bankovního účtu plátce na bankovní účet vyplněný příjemcem. Tento model je zachycen na obrázku 2.2.

Výhodou tohoto přístupu oproti předchozímu modelu je fakt, že oba účastnící Peer-to-Peer platby mohou využívat služby jiného poskytovatele bankovních služeb. Oproti předchozímu modelu také příjemce nemusí být uživatelem aplikace. Nevýhodou tohoto modelu je naopak celková komplikovanost procesu, která brání jednoduchému provádění Peer-to-Peer plateb. Na českém trhu mezi zástupce tohoto řešení patří například aplikace Friends24 České spořitelny[13].

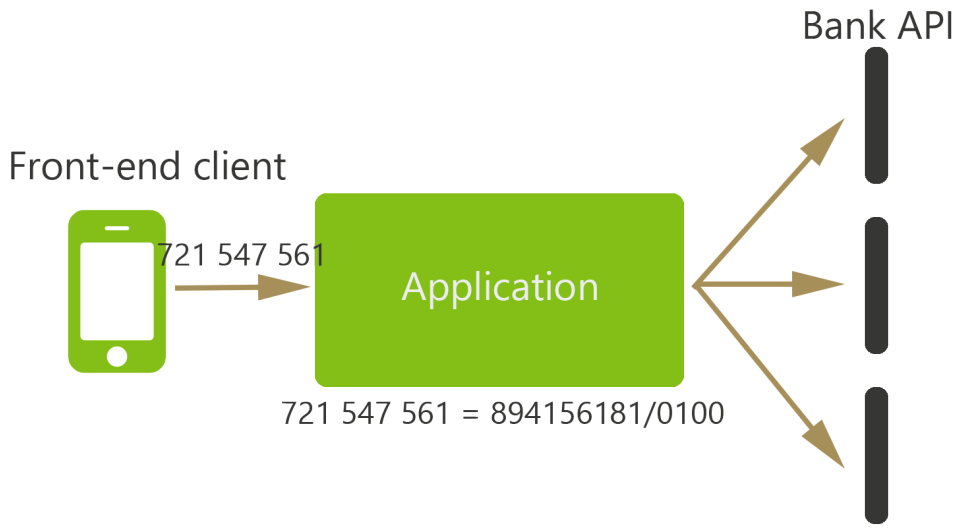
2.3 Koncept aplikace s využitím legislativy PSD2

V přechozí sekci jsou představeny dva nejčastější modely realizace Peer-to-Peer plateb, z nichž každý přináší své výhody a omezení. Jedním z důležitých bodů směrnice PSD2 je zajištění vysoké úrovně hospodářské soutěže, s čímž je také spojená povinnost bank zpřístupnit třetím stranám svá systémová rozhraní neboli API[7]. V této sekci je obecně představen koncept aplikace pro provádění Peer-to-Peer plateb, který je postaven na využití tohoto veřejně dostupného bankovního rozhraní.

Cílem konceptu aplikace je využít zpřístupněných bankovních rozhraní takovým způsobem, aby byl proces vytváření Peer-to-Peer plateb co nejjednodušší a odstraňoval nevýhody dříve uvedených modelů. Pro zjednodušení platebního procesu bude aplikace abstrahovat uživatele od čísla bankovního účtu, jednotliví uživatelé budou identifikováni jen za pomoci svého telefonního čísla.

Uživatelé si stáhnou aplikaci a v rámci registrace uvedou své kontaktní údaje a souhlas s jejich využíváním pro potřeby zadávání plateb. V aplikaci následně budou moci zadat telefonní číslo a detail platby, kterou si přejí zahájit. Tyto informace jsou následně přeneseny na aplikační server, kde dochází k překladu telefonního čísla na číslo bankovního účtu. Podle banky iniciátora platby je na vystavené rozhraní správné banky odeslán požadavek na provedení platby.

Vystavené bankovní API předá požadavek na vytvoření nové platby příslušným platebním systémům, které mají na starosti samotné provedení finanční transakce. Oba účastníci transakce jsou následně pomocí aplikace informováni o úspěšně provedené platbě. Tento proces je zachycen na obrázku 2.3.



Obrázek 2.3: Model P2P platby s využitím zpřístupněných bankovních API.

Kapitola 3

Vývoj mobilní aplikace

V závěru předchozí kapitoly je zevrubně představen koncept mobilní aplikace pro provádění Peer-to-Peer plateb s využitím bankovního Open API. Tato kapitola se zabývá popisem základních přístupů pro tvorbu mobilních aplikací. Druhá část představuje dva základní přístupy architektury komunikace v rámci a mezi aplikacemi – SOAP a REST.

3.1 Mobilní aplikace

Pod pojmem mobilní aplikace zde označujeme software navržený takovým způsobem, aby fungoval na mobilních zařízeních, kterými jsou mobilní telefony a tablety[14]. Z pohledu vývoje mobilní aplikace jsme schopni rozlišit tři základní možnosti pro vývoj mobilních aplikací[15]:

- nativní aplikace,
- HTML5 aplikace,
- hybridní aplikace.

3.1.1 Nativní mobilní aplikace

Nativní aplikace jsou vyvíjeny specificky pro zvolenou mobilní platformu, jakými jsou například iOS, nebo Android. Každá platforma má také vlastní programovací jazyk, který daná platforma podporuje: XCode a Objective-C v případě iOSu, Java v případě Androidu. Obecně se tyto aplikace chovají a vypadají na cílovém zařízení nejlépe. Zároveň je však nutné aplikaci implementovat pro každou platformu zvlášť a implementace vyžaduje vysokou úroveň znalostí vývojáře, tedy je nákladná[16].

3.1.2 HTML5 aplikace

HTML5 aplikace je ve své podstatě skupina webových stránek, které jsou uzpůsobeny pro spuštění na malých obrazovkách. Takováto aplikace tedy je jednoduše dostupná pomocí jakéhokoliv mobilního prohlížeče. K implementaci jsou využity standardní webové technologie, kterými jsou HTML, Javascript a CSS. Využití těchto technologií umožňuje vytvoření multiplatformní aplikace, která bude fungovat napříč jednotlivými platformami.

Použité technologie však přináší také svá zásadní omezení. Na rozdíl od nativních aplikací nemá HTML5 aplikace přístup k operačnímu systému zařízení a aplikace nikdy nedosáhne očekávaného chování, jakým je například ovládání gesty. Další klíčové nedostatky jsou spojené s ukládáním dat a zabezpečením, které jsou pro velkou skupinu aplikací nedostatečné[16].

3.1.3 Hybridní aplikace

Hybridní mobilní aplikace kombinují výhody nativního a HTML5 přístupu. Hybridní aplikace jsou implementovány pomocí standardních webových technologií a pomocí nativního kontejneru, jakým je například PhoneGap, přistupují k funkcionalitám daného zařízení.

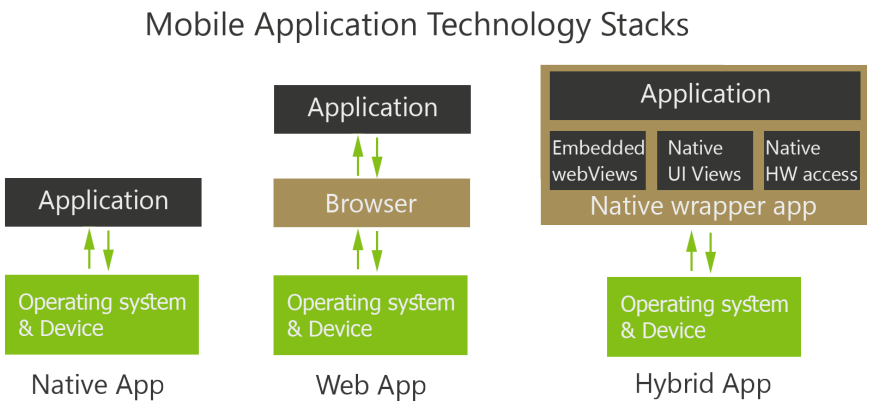
Hlavní výhodou oproti nativnímu přístupu je možnost implementace platformně nezávislé mobilní aplikace pomocí standardních webových technologií. Vývoj takové aplikace je tedy výrazně méně finančně nákladný než v případě nativní aplikace. Nevýhodou naopak zůstává fakt, že poskytnuté API nezpřístupňuje veškerou funkcionalitu zařízení. Hybridní aplikace také nedosahuje stejného výkonu jako nativní aplikace a lze v některých případech tedy očekávat pomalejší reakce. Tato nevýhoda je však rychle překonávána díky rostoucímu výkonu hardwaru mobilních zařízení, u většiny aplikací tak uživatel nepozná rozdíl[16].

Rozdíl v konceptu jednotlivých přístupů k vývoji mobilních aplikací je zachycen na diagramu 3.1.

3.2 Apache Cordova

Součástí praktické části této diplomové práce je také implementace prototypu mobilní aplikace. Jelikož tento prototyp bude vyvíjen jako hybridní mobilní aplikace, je vhodné v této sekci blíže představit koncept nástroje Apache Cordova, pomocí kterého bude prototyp implementován.

Apache Cordova je open-source framework pro vývoj hybridních mobilních aplikací. Lze si tento nástroj představit jako předpřipravenou nativní aplikaci pro jednotlivé platformy, která má přes celou obrazovku WebView komponentu, do které se načte obsah definovaný příslušným HTML, JS a CSS kódem[17]. Tato předpřipravená nativní aplikace přistupuje k API operačního systému a umožňuje tak aplikaci využívat funkcionalitu zařízení.



Obrázek 3.1: Koncept vývoje mobilních aplikací.

Při pohledu na architekturu mobilní aplikace pomocí Apache Cordova jsme schopni rozlišit následující klíčové části[1]:

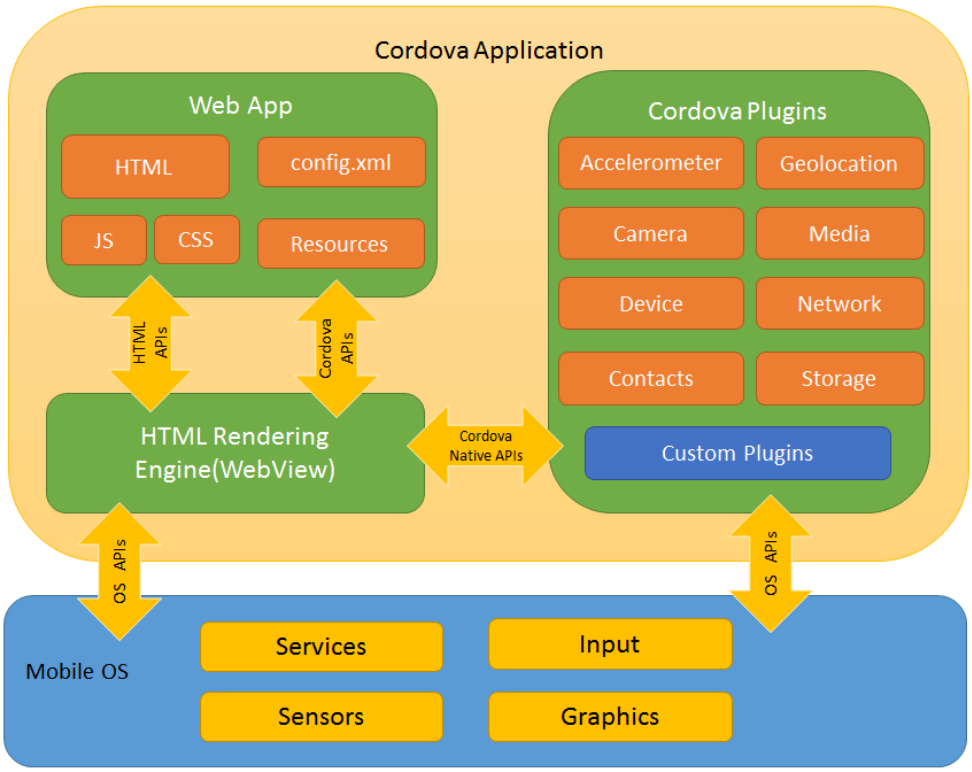
WebApp V této části jsou obsaženy veškeré HTML, JS a CSS kódy, které definují strukturu, vzhled a chování aplikace. Tato část také obsahuje obrázky a ostatní grafické prvky, které bude mobilní aplikace zobrazovat. V poslední řadě je zde uložena konfigurace Apache Cordova.

WebView V části WebView je aplikace spuštěna, dochází zde tedy k zobrazení a vykonání kódu definovaného v části WebApp.

Plugins Tato část poskytuje rozhraní pro komunikaci s API zařízení. Díky této části je možné spustit nativní funkce zařízení přímo v JS kódu aplikace.

3.3 Komunikace v rámci aplikace

Na začátek tvorby architektury mobilní aplikace je nutné určit, zda se bude jednat o aplikaci typu tenký klient, nebo tlustý klient. V prvním případě bude na mobilním zařízení implementována pouze prezentační vrstva, zatímco aplikační a datové vrstvy budou umístěné na serverové části[14]. Podíváme-li se na návrh aplikace pro provádění Peer-to-Peer plateb, jedná se o aplikaci typu tenký klient a bude proto nutné se věnovat také komunikaci v rámci aplikace.



Obrázek 3.2: Pohled na architekturu Apache Cordova mobilní aplikace[1].

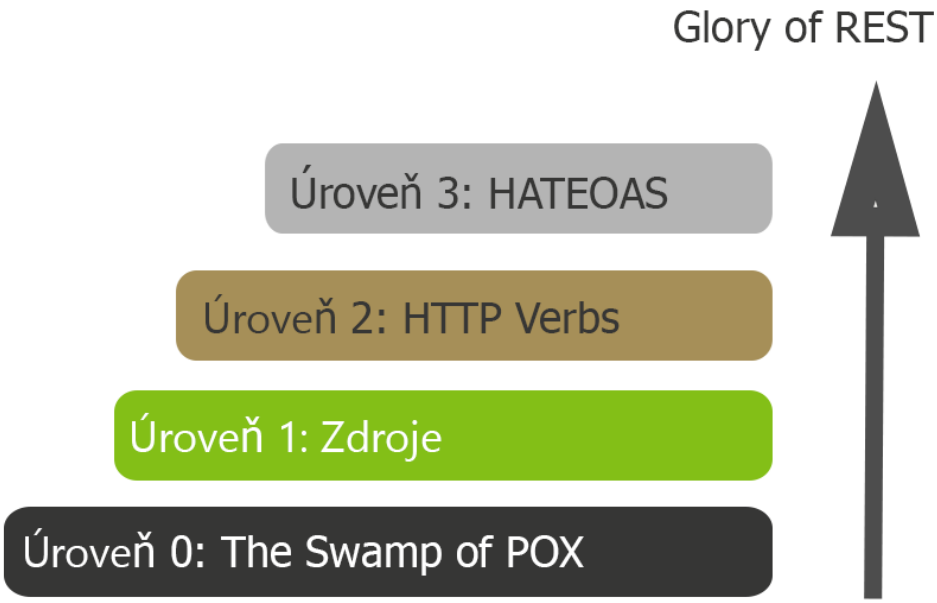
V současné době rozlišujeme dva základní druhy služeb: SOAP služby a RESTful služby. Zatímco SOAP služby odpovídají principu RPC[3], RESTful služby se orientují spíše na zdroje[18]. Volba vhodného přístupu se odvíjí od konkrétního případu použití. V této sekci představíme a porovnáme oba přístupy. Jako příklady jednotlivých operací jsou použity konkrétní požadavky z prototypu implementovaného v rámci praktické části práce.

3.3.1 RESTful služby

Representational State Transfer (REST) je architektura rozhraní navržená pro distribuované prostředí. Tuto architekturu představil v roce 2000 v disertační práci Roye Fielding, spoluautor protokolu HTTP. I z tohoto důvodu toho má REST s protokolem HTTP mnoho společného.

REST je na rozdíl od služeb SOAP, orientován datově, nikoliv proceduálně. REST tak slouží jako jednotný a jednoduchý přístup ke zdrojům –

[3]Remote procedure call je termín, který popisuje v informatice situaci, kdy jedno zařízení volá proceduru, která je vykonána na jiném zařízení v rámci sítě.



Obrázek 3.3: Leonard Richardson - REST maturity model[2].

datům. Veškerá data jsou identifikována vlastním jednotným identifikátorem, zvaným URI[4][19].

Abychom lépe pochopili, jak REST funguje, představíme si čtyř-úrovňový model zralosti definovaný Leonardem Richardsonem. Tento model, zobrazený na obrázku 3.3, popisuje čtyři úrovně RESTu od The Swamp of POX až po takzvané Glory of REST.

3.3.1.1 Nultá úroveň – The Swamp of POX

Nultá úroveň modelu definuje využití HTTP jako protokolu pro vzdálenou komunikaci, bez využití jiných mechanismů webu[2]. Vezmeme-li v potaz rozšíření protokolu HTTP, je pro nás téměř nepředstavitelné vystavět REST architekturu na jiném protokolu. Splnění nulté úrovně modelu zralosti je tedy pro navrhovanou aplikace spíše samozřejmostí.

[4]Uniform Resource Identifier, textový řetězec sloužící k přesné specifikaci zdroje informací.

3.3.1.2 První úroveň - Zdroje

První úroveň vyžaduje zavedení zdrojů takovým způsobem, aby veškeré požadavky nebyly směrovány na jeden koncový bod, nýbrž aby každý zdroj měl svůj vlastní koncový bod, se kterým je možné komunikovat[2].

Jako konkrétní příklad si lze představit určitou platbu, která byla aplikací provedena a která musí být pro aplikaci dostupná. Koncový bod pro směrování požadavku na tuto platbu by mohl vypadat následovně:

```
/payment/78949851658
```

Požadavek tedy bude směrován na kolekci všech evidovaných plateb, ze které je vybrána platba s daným identifikátorem.

3.3.1.3 Druhá úroveň – HTTP Verbs

V HTTP protokolu jsou definovány čtyři základní metody, které umožňují CRUD operace – POST, GET, PUT, DELETE. Druhá úroveň vyspělosti REST architektury klade důraz na co nejpřesnější využití těchto metod dle HTTP standardu[2].

U těchto operací lze sledovat dvě vlastnosti: bezpečnost a idempotenci. Operace je bezpečná, pokud nijak nemění hodnotu dotazovaného zdroje a nevytváří žádný nový zdroj – slouží pouze pro čtení. Idempotentní operace je pak taková, která opakovaným voláním vrací vždy stejný výsledek[20].

GET (Retrieve/Read)

Základní metodou je operace GET, která slouží pro získání zdroje. Podle HTTP specifikace je operace GET určená pouze pro čtení dat, a nikoliv jejich editaci. Jedná se tedy o bezpečnou operaci, která nikterak cílová data nezmění. Opakované volání stejná operace vždy vrací stejné výsledky, jedná se tedy o idempotentní operaci. Jako příklad uvedeme volání na rozhraní našeho prototypu pro získání kolekce plateb (payment). V případě úspěšného volání je vrácena odpověď ve formátu JSON, XML nebo HTTP[20]. Konkrétní požadavek GET na platbu vypadá následovně:

```
GET /payment HTTP/1.1
Host: kbbillme.herokuapp.com
```

POST (Create)

Operace POST slouží k vytváření nových zdrojů podřízených jiným zdrojům. Novému zdroji je při vytváření přiřazen unikátní identifikátor – URI. V případě úspěšného vytvoření zdroje je vrácena odpověď 201 (Created) společně s odkazem na nově vytvořený zdroj. Součástí těla POST požadavku jsou informace, které mají být novému zdroji vyplněny. Z principu je jasné, že se nejedná o bezpečnou metodu, jelikož slouží pro vytváření nových zdrojů. Její

opakované volání navíc vyústí ve vytvoření několika zdrojů se stejnou informací, nejedná se tedy o idempotentní operaci. Jako příklad ukážeme požadavek na vytvoření nové platby, jejíž obsah je ve formátu JSON[20].

```
POST /payment HTTP/1.1
Host: kbbillme.herokuapp.com
Content-Type: application/json
{
        "initiator":"bb4f7c4f42d21a9b",
        "amount":21,
        "reciever":"800 154 156",
        "message": "Na chleb"
}
```

PUT (Update)

PUT je operace velmi podobná operaci POST a slouží k úpravě existujícího zdroje. Na rozdíl od operace POST však PUT obsahuje URI konkrétního zdroje a ve svém těle nese informace, které mají být danému zdroji nastaveny. V případě, kdy PUT nese ve svém těle informaci o svém identifikátoru, který dosud neexistuje, může být použita také pro vytvoření nového zdroje. Tento případ je však poměrně matoucí a je proto lepší se mu vyhnout. Z podstaty operace je jasné, že se nejedná o bezpečnou operaci, avšak jedná se o idempotentní operaci[20].

Příklad PUT si opět předvedeme na naší aplikaci na konkrétním příkladu. Cílem volání je změnit stav žádosti o platbu (request) se známým identifikátorem na dokončeno (completed) a zároveň k ní přidat odkaz na provedenou platbu (payment).

```
PUT /request?id=75fa755e42fdf812 HTTP/1.1
Host: kbbillme.herokuapp.com:2403
Content-Type: application/json
{
        "state": "completed",
        "payment": "46156158"
}
```

DELETE (delete)

Poslední základní operace DELETE slouží, jak název napovídá, ke smazání zdroje definovaného jednoznačným identifikátorem. V případě úspěšného smazání vrací operace kód 200 (OK), pokud zdroj není nalezen, pak kód 404 (NOT FOUND). Z principu se nejedná o bezpečnou operaci, avšak zanedbáme-li návratový kód, můžeme operaci nazvat idempotentní[20].

Jako příklad si ukážeme volání operace DELETE na kolekci plateb, kde si přejeme smazat platbu se známým identifikátorem.

```
DELETE /payment?id=e50a229fac4af812 HTTP/1.1
Host: kbbillme.herokuapp.com:2403
Content-Type: multipart/form-data;
```

3.3.1.4 Třetí úroveň – Hypermedia Controls

Tento princip se označuje také pomocí akronymu HATEOAS (Hypertext as the engine of application state) a adresuje problém závislosti znalosti koncových bodů jednotlivých zdrojů. V praxi to znamená, že můžeme znát pouze jeden koncový bod, který nám společně s daty vrací také odkazy na následující zdroje. Správná implementace tohoto principu přináší zřejmou výhodu v tom, že klient není závislý na koncových bodech jednotlivých zdrojů a ty se mohou dynamicky měnit. V praxi se však jedná o dosud nepříliš často používaný princip[2].

3.3.1.5 Shrnutí REST architektury

REST je architektura, která je orientovaná na zdroje a k jejich získání využívá standardní HTTP metody. Mezi hlavní přednosti REST API tak patří jednoduchost a nenáročnost. Díky své bezstavosti REST vychází vstříc moderním metodám vývoje webových aplikací založených na distribuovaném obsahu. Díky možnosti definovat přenášený obsah pomocí notace JSON se jedná o častou volbu architektury webových aplikací, především z důvodu jednoduché implementace.

3.3.2 SOAP

SOAP (Simple Object Access protocol) je komunikační protokol založený na standardu XML a v současnosti se jedná o základ webových služeb. Podobně jako REST, také protokol SOAP umožňuje komunikaci a přenos dat mezi aplikacemi v heterogenním prostředí. SOAP slouží pro volání vzdálených procedur (princip RPC) – jedna aplikace v XML zprávě předá druhé aplikaci požadavek, ta požadavek obslouží a pošle odpověď původnímu iniciátorovi komunikace. Svou orientací na volání vzdálených procedur je protokol SOAP neodmyslitelnou součástí SOA architektur[21].

3.3.2.1 Struktura zprávy dle SOAP

Zpráva posílaná pomocí SOAP protokolu je jednoduchý XML dokument s kořenovým elementem Envelope (obálka). V této obálce jsou uzavřeny dva bloky: Header (hlavička) a Body (tělo). Hlavička SOAP obálky je nepovinná a obsahuje pomocné informace pro zpracování zprávy – identifikaci uživatele, autentizaci odesílatele a podobné informace.

V těle zprávy se pak nacházejí veškeré přenášené informace, které identifikují volanou službu a předávané parametry, jejichž struktura musí splňovat XSD strukturu definovanou volanou službou. Pro definování jednotlivých částí XML zprávy používá SOAP příslušné jmenné prostory[21]. SOAP zpráva pro zavolání vzdálené funkce CreateNewPayment, služby paymentService, pro založení nové platby by mohla vypadat následujícím způsobem. Pro jednoduchost zpráva neobsahuje hlavičku.

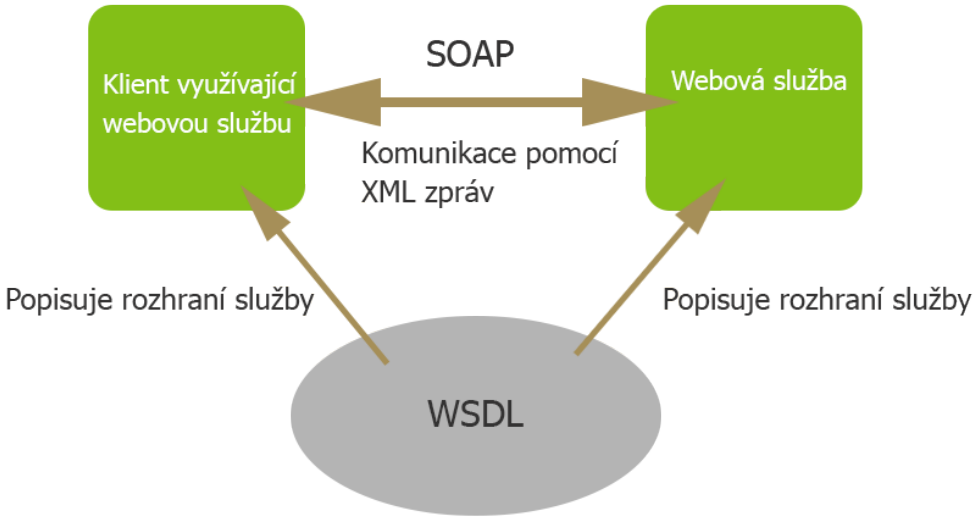
```
<SOAP-ENV:Envelope
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
 SOAP-ENV:encodingStyle=
 "http://schemas.xmlsoap.org/soap/encoding/">
   <SOAP-ENV:Body>
    <m:CreateNewPayment
         xmlns:m="urn:x-example:services:PaymentService">
     <amount>100</amount>
     <initiator>bb4f7c4f42d21a9b</ initiator >
     <reciever>a840c0e72570689c</ reciever >
  <message>Za pomoc pri rekonstrukci vikyre</message>
  </m: CreateNewPayment >
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Podobně jako architektura REST, i SOAP v drtivé většině případů využívá jako transportní mechanismus protokol HTTP. Důvodem je především široká podpora protokolu HTTP napříč aplikacemi[21].

3.3.2.2 Web Service Description Language

S pojmem SOAP se velmi úzce pojí termín WSDL. Jedná se o jazyk založený na XML formátu, sloužící k popisu síťových služeb jako množiny koncových bodů [12]. Operace a zprávy jsou popisovány na abstraktní úrovni a teprve následně jsou svázány s konkrétním datovým formátem a síťovým protokolem. Díky tomuto je možné vytvořit popis rozhraní, které nabízí jednu službu několika způsoby [13]. Základní části každého WSDL popisu jsou následující[3]:

Types	Definice datových struktur využitých ve zprávách.
Message	Formát předávaných zpráv pomocí dříve definovaných datových typů.
Operation	Abstraktní definice operací, které jsou službou podporovány, definuje vstupy a výstupy.
PortType	Sdružuje dohromady několik operací.
Binding	Slouží pro navázání určitého portu na protokol a formát přenosu zpráv.



Obrázek 3.4: Vztah mezi SOAP a WSDL[3].

Port — Jeden koncový bod služby definovaný jako kombinace síťové adresy a vazby binding.

Service — Sdružuje několik koncových bodů služby.

Vztah mezi WSDL a SOAP a jeho místo v rámci komunikace je zobrazen na obrázku číslo 3.4.

3.3.3 Porovnání SOAP a REST

V předchozích sekcích práce je představen SOAP i REST jako možný přístup pro komunikaci v rámci aplikace. Oba přístupy přináší své výhody a omezení, které je nutné ve fázi návrhu vzít v potaz. Tyto vlastnosti jsou zachyceny v tabulce 3.1.

SOAP	REST
Zavedenější technologie, podpora ostatních standardů (WSDL, WS-*).	Ve srovnání se SOAP se jedná o novější technologii, postrádá standardizaci.
Stále atraktivní technologie pro větší systémy, které se nachází například v bankách.	RESTful implementace služeb do tohoto odvětví zatím teprve proniká, přesto i na této úrovni existují úspěšné implementace.
Interakce mezi klientem a serverem je úzce provázána.	Interakce mezi klientem a serverem je volně provázaná.

SOAP	REST
Úpravy služeb často zahrnují složité úpravy kódu na straně klienta.	Úprava služeb nemá žádný dopad do klientské části kódu.
Zasílané zprávy mají nadbytečnou velikost.	Zprávy obsahují minimální overhead.
Vyžaduje binární parsování příloh.	Přímo podporuje veškeré datové typy.
Není přívětivý pro využití při bezdrátovém připojení.	Je přívětivý také pro využití při bezdrátovém připojení.
SOAP webové služby vždy vracejí data ve formátu XML.	REST webové služby umožňují definovat formát, ve kterém budou data navrácena.
Nezávislý na jazyce, platformě a transportním protokolu.	Nezávislý na jazyce a platformě.
Navržený pro použití v distribuovaném výpočetním prostředí.	Navržený pro point-to-point komunikační model, nikoliv pro distribuované prostředí, kde zpráva prochází přes více prostředníků.
Složitější implementace, závislost na vhodných nástrojích.	Jednodušší na implementaci, nevyžaduje specializované nástroje.

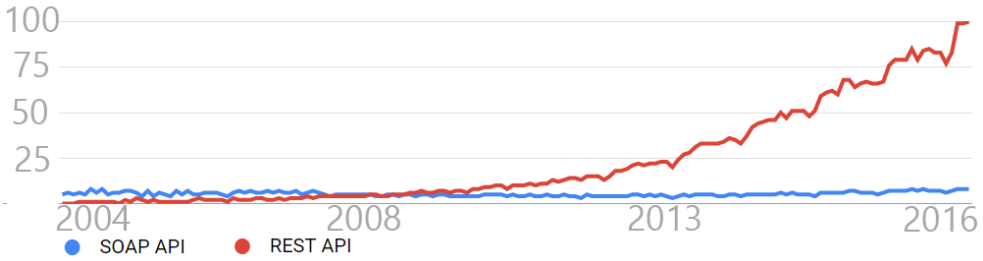
Tabulka 3.1: Porovnání SOAP a REST architektury služeb[22].

Z výše uvedeného popisu je evidentní, že využití konkrétní technologie se odvíjí od konkrétního případu použití[23]. V současném době však výrazným způsobem převládá REST API, především z důvodu jednoduchosti použití a menší velikosti předávaných zpráv. Tento trend je zachycen v grafu 3.6, který zobrazuje trend zájmu o SOAP a REST API od roku 2004 po současnost.

Vytvářet SOAP API tak v dnešní době může mít význam v prostředí bankovních systémů, nebo informačních systémů velkých společností. V drtivé většině případů je však flexibilnější a jednodušší volbou implementace RESTful architektury.

3.4 Architektura mikroslužeb

Architekturou založenou na mikroslužbách rozumíme takový přístup ke tvorbě softwaru, ve kterém je aplikace složena z mnoha malých služeb. Každá takováto mikroslužba má na starost jeden konkrétní úkol. Jednotlivé mikroslužby jsou navzájem nezávislé a mají mezi sebou pouze volné vazby. Komunikace mezi nimi probíhá pomocí API, často typu REST z důvodu jednoduchosti a rychlosti[24].



Obrázek 3.5: Procentuální vývoj trendu zájmu o SOAP a REST API od roku 2004 po rok 2016. Převzato ze služby Google Trends.

Jedná se tak o reakci na monolitickou architekturu, využívanou na serverové části, která definuje celou aplikaci jako jednu funkční jednotku. Monolitická architektura odpovídá přirozenému způsobu implementace systému – veškerá logika tvoří jeden proces, systém lze vhodně rozdělit do patřičných tříd a jmenných prostorů podle funkcionality. Tento přístup však přináší zásadní nedostatky, kterými jsou[24]:

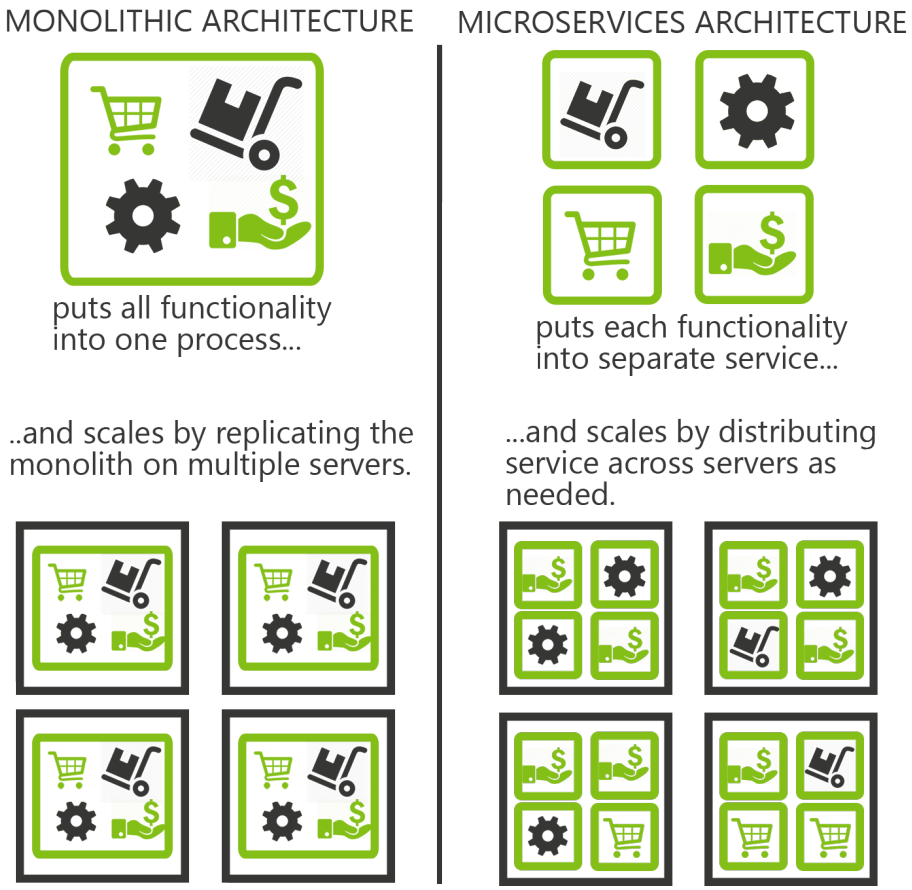
1. Úprava malé části aplikace vyžaduje sestavení a nasazení celého systému.
2. V případě nutnosti škálování lze škálovat pouze celý systém, tedy vystavit novou instanci systému za příslušný load balancer. Škálování celého systému je nákladné řešení.

Jako reakce na tyto nedostatky monolitické architektury aplikace vznikla architektura mikroslužeb, která tyto problémy adresuje. Výhody tohoto přístupu jsou[24]:

1. Jednotlivé mikroslužby je možné samostatně nasadit – při změně tak není nutné sestavovat a nasazovat celý systém, ale pouze tu část, ve které došlo ke změně.
2. Každá mikroslužba je samostatně škálovatelná. Není nutné škálovat celý systém podle nutnosti nejvytíženějšího místa, ale každá část systému může být škálovatelná dle své potřeby.
3. Mikroslužby jsou na sobě nezávislé, mohou tak být napsány v různých programovacích jazycích a spravovány různými odděleními.

3.4.1 Charakteristika mikroslužeb

Tato sekce se zabývá charakteristikou mikroslužeb a jejich vlastnostmi, se kterými je nutné v rámci návrhu počítat.



Obrázek 3.6: Porovnání monolitické architektury a architektury mikroslužeb[4].

3.4.1.1 Rozsah a zaměření mikroslužby

Přesná definice rozsahu mikroslužby přirozeně neexistuje. Každá mikroslužba musí mít na starost určitou jednotku práce a její rozsah musí být takový, aby byla jednoduše udržovatelná. K mikroslužbě je nutné přistupovat stejným způsobem jako k aplikaci – zdrojové kódy musí být správně verzovány a musí být definovaný proces sestavení a nasazení služby. Díky malému rozsahu dochází také ke zvýšení znovu použitelnosti mikroslužby a schopnosti rychle reagovat na potřeby trhu[24].

3.4.1.2 Volné provázání

Nezbytnou charakteristikou mikroslužeb je jejich vzájemné volné provázání, tedy minimální vzájemná závislost. Každou mikroslužbu musí být možné nasadit samostatně bez nutnosti koordinace s nasazením ostatních mikroslužeb. Díky této vlastnosti je možné provádět rychlá nasazení, čímž dochází ke zkrácení času potřebného k opravě nebo vytvoření nové funkcionality[24].

3.4.1.3 Jazyková nezávislost

Při implementaci funkcionality je nutné používat nejvhodnější prostředky a technologie, které daný problém řeší. Vhodný programovací jazyk pro řešení jednoho problému nemusí být nutně vhodnou volbou pro jinou část systému. Mikroslužby dohromady vytváří systém, avšak každá z nich může být implementována pomocí vlastního programovacího jazyka, který se pro danou funkcionalitu nejlépe hodí[24].

Komunikace mezi jednotlivými službami následně probíhá běžně pomocí HTTP protokolu a REST API. Na této úrovni by také měla být provedena veškerá standardizace a vzájemná integrace, samotné mikroslužby by však měly zůstat jazykově nezávislé.

3.4.1.4 Nezávislé na kontextu

Implementace mikroslužby musí být nezávislá na kontextu ostatních mikroslužeb – tedy vybraná mikroslužba nesmí nic předpokládat o způsobu fungování svého okolí. Správně by tedy mikroslužba měla využívat pouze rozhraní okolních služeb bez znalosti postupu řešení nebo konkrétní implementace nabízené funkcionality[24].

3.4.2 Shrnutí architektury mikroslužeb

Rozdělení serverové části aplikace pomocí mikroslužeb je vhodné především v případě podnikových systémů, pro které není monolitická architektura dostatečně flexibilní. Díky rozdělení aplikace do mikroslužeb tak lze dosáhnout větší míry flexibility při nasazování oprav případě nových funkcionalit do produkčního prostředí.

Kapitola 4

Vymezení praktické části diplomové práce

Praktická část této diplomové práce se zabývá návrhem mobilní aplikace pro provádění Peer-to-Peer plateb a následnou implementací prototypu této aplikace. Při návrhu aplikace práce vychází ze své teoretické části, ve které jsou představeny použité koncepty. Návrh vzniká na základě reálného projektu nejmenované banky, ve spolupráci s Michalem Kovářem. Tento projekt je rozdělen do tří částí:

1. Business analýza a tvorba UX návrhu – v rámci této části projektu je vytvořen business model aplikace, proveden kvantitativní průzkum s cílem určit klíčové funkce a vytvořen UX návrh aplikace, který je podroben kvantitativnímu testování. Tuto fázi projektu pokrývá Michal Kovář ve své diplomové práci[25].

2. Návrh aplikace a implementace funkčního prototypu – do druhé fáze projektu spadá technická analýza a návrh architektury aplikace pro provádění Peer-to-Peer plateb. Dalším cílem této fáze je vytvoření funkčního prototypu na základě UX návrhu, na kterém bude možné provést testování aplikace. Tato fáze projektu je zároveň obsahem praktické části této diplomové práce.

3. Produkční implementace – cílem poslední fáze projektu je transformace funkčního prototypu na produkční verzi mobilní aplikace pro provádění Peer-to-Peer plateb podle vytvořené analýzy. Tato fáze navazuje na dvě předchozí a její realizace závisí na rozhodnutí nejmenované banky.

4.1 Postup práce

V první fázi se práce věnuje části Návrh, ve které je stručně představen business účel mobilní aplikace pro provádění Peer-to-Peer plateb. Následně jsou

definovány požadavky kladené na výslednou aplikaci. Z business analýzy vychází technická analýza, ve které je představen datový model a celková architektura včetně aplikačních procesů. Vypracovaný návrh představuje konečnou podobu a architekturu aplikace a liší se tak od implementovaného prototypu. V rámci návrhu je kladen důraz na definování, které funkcionality nebudou v rámci prototypu implementovány.

Část realizace se věnuje implementaci prototypu mobilní aplikace, který vychází z fáze návrhu. V této části jsou také zdůrazněny odchylky od produkční verze aplikace, která je podrobněji analyzována v návrhu. Cílem prototypu je vytvoření mobilní aplikace, která bude fungovat na cílovém zařízení a bude demonstrovat funkcionalitu výsledného produktu. Cílem naopak není implementace produkční verze aplikace, proto v prototypu není kladen důraz na zabezpečení.

Třetí kapitolou praktické části je ekonomicko-manažerské zhodnocení projektu, ve kterém se práce zabývá harmonogramem realizace, odhadem nákladů a stanovením metrik pro měření úspěšnosti.

Kapitola 5

Návrh mobilní aplikace

Na počátku návrhu je nutné stručně představit výsledný produkt a jeho business model. Na základě business modelu jsme schopni definovat funkční a nefunkční požadavky, které budou na aplikaci kladeny. Následně podle těchto požadavků definujeme případy použití a také aplikační procesy. Návrh se zabývá také technickým pohledem na architekturu aplikace, včetně stavových diagramů entit a způsobu ukládání dat.

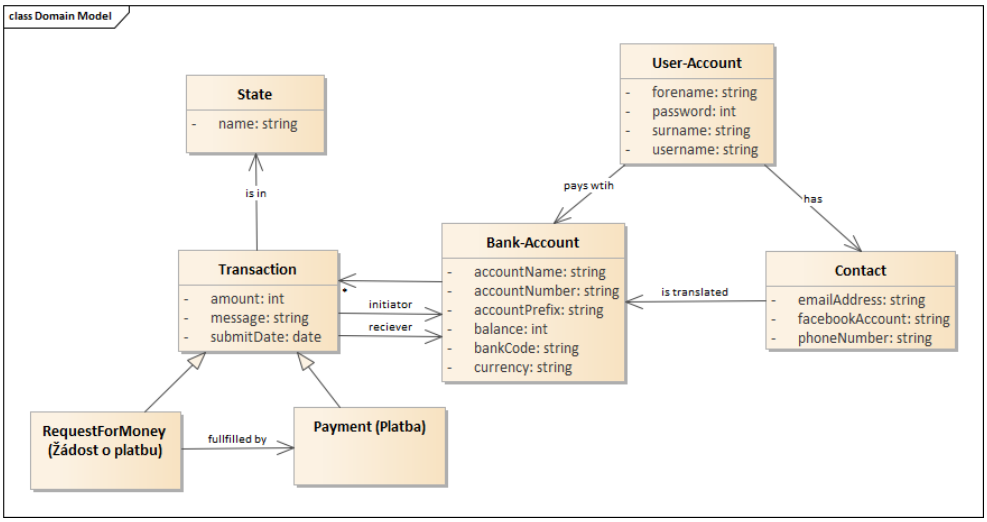
5.1 Business potřeby

Před zahájením celé analýzy nového produktu je potřeba zodpovědět klíčový dotaz: Je o produkt na trhu skutečně zájem? Před zodpovězením této otázky je nutné alespoň stručně definovat, co vlastně nový produkt je a na koho cílí.

V našem případě můžeme business model produktu definovat asi takto: „Jedná se o mobilní aplikaci, která bude uživatelům umožňovat platby pomocí kontaktu, například telefonního čísla. Cílovou skupinou této aplikace jsou především mladí lidé a lidé, kteří mají kladný přístup k nový technologiím“.

Nyní již víme, co náš produkt je, nijak jsme však neodpověděli na otázku, zda o něj bude na trhu zájem. Podle některých průzkumů téměř polovina startupových projektů skončí z důvodu, že po implementaci a vydání produktu na trh není o výsledný produkt zájem. Abychom s mobilní aplikací předešli stejnému osudu, rozhodli jsme se nejprve celý koncept Peer-to-Peer plateb ověřit pomocí kvantitativního průzkumu.

Kvantitativní průzkum jsme zaměřili na naši cílovou skupinu a při zadávání jsme definovali potřeby jejích zástupců, které by naše aplikace mohla pokrývat. Samotným zpracováním kvantitativního průzkumu byla pověřena nejmenovaná agentura. Na základě výstupů tohoto průzkumu jsme ověřili, že o tento produkt je na trhu zájem a zároveň jsme byli schopni určit, jaké funkce by tato aplikace měla uživatelům nabízet. Než však představím funkční a ne-



Obrázek 5.1: Navržený doménový model aplikace.

funkční požadavky plynoucí z kvantitativního výzkumu, dovolím si představit doménový model, abychom si sjednotili používanou terminologii.

5.2 Doménový model

Doménový model slouží k popisu a zachycení domény, se kterou bude vytvářený systém pracovat. Jedním z hlavních přínosů doménového modelu je sjednocení pohledu a terminologie modelovaného světa[26]. Doménový model se skládá z klíčových entit a vztahů mezi nimi. U jednotlivých entit jsou navíc popsány klíčové atributy a vlastnosti. Vytvářený model by naopak neměl obsahovat informace důležité pro pozdější implementaci a měl by být platformně nezávislý. Výsledný doménový model je zachycen na obrázku 5.1.

5.2.1 Popis doménového modelu

Uživatelé se budou do aplikace registrovat, na základě čehož jim bude vytvářen uživatelský účet. Uživatelský účet bude jednoznačně identifikovatelný pomocí uživatelského jména a pro přihlášení bude vyžadováno heslo – tato skutečnost je v doménovém diagramu reprezentována pomocí entity User-Account (Uživatelský účet).

Aplikace bude umožňovat zasílat jiným uživatelům peníze na základě znalosti jejich kontaktního údaje. Tím může být například telefonní číslo, email, nebo facebookový účet. V rámci implementace prototypu si pro jednoduchost vystačíme pouze s telefonním číslem, logika komunikace pomocí ostatních kon-

taktních údajů však bude stejná. Entita Contact (Kontakt) v doménovém modelu reprezentuje kontakt, na který bude možné uživateli posílat peníze.

Aby bylo možné finanční prostředky skutečně jinému uživateli zaslat, každý uživatel musí mít evidovaný také bankovní účet – entita Bank-Account (Bankovní účet). Tento bankovní účet musí mít evidované mimo jiné číslo účtu a kód banky, aby aplikace věděla, přes jaké rozhraní má s bankou komunikovat. Jakmile uživatel zadá platbu na určité telefonní číslo, aplikace ve své databázi toto číslo vyhledá a přeloží jej na číslo účtu, na které bude finanční částka zaslána.

Klíčovou funkcionalitou celé aplikace je posílání Plateb a Žádostí o platby jiným uživatelům. Podíváme-li se na obě entity, zjistíme, že mají řadu společných faktorů. Vždy se bude jednat o transakci, která probíhá mezi dvěma uživateli a která má řadu atributů společných. Z toho důvodu můžeme z obou souhrnně vytvořit entitu Transaction (Transakce), která bude obsahovat společné rysy obou entit.

První entitou, která bude z třídy Transakce dědit je třída Payment (Platba). Platba reprezentuje jednu konkrétní platbu jednoho uživatele jinému. Při platbě tedy dojde ke skutečnému převedení finanční částky mezi dvěma bankovními účty.

Druhou entitou je třída Request For Money (Žádost o platbu), která reprezentuje připomenutí platby jednoho uživatele druhému. U Žádosti o platbu tedy nedochází ihned k žádnému převodu peněz. Tato entita slouží pouze k připomenutí se dlužníkovi o určitou částku. Dlužník má následně možnost obdrženou Žádost o platbu akceptovat. V tomto případě se vytvoří nová Platba z účtu dlužníka na účet věřitele.

Poslední důležitou entitou, která se v naší doméně nachází a která je velmi úzce spojená s transakcí je třída State (Stav). Tato třída reprezentuje stav, ve kterém se transakce zrovna nachází. Jednotlivé instance třídy Transakce se během svého životního cyklu mohou vyskytnout v různých stavech: vytvořeno, zamítnuto, přijato apod. Tyto stavy je potřeba mít bezpečně podchycené, aby nedocházelo k chybným zpracováním transakcí.

5.3 Funkční a nefunkční požadavky

V předchozí sekci jsme si blíže představili doménu, kterou bude výsledný systém pokrývat, a sjednotili používanou terminologii. Nyní se můžeme podívat na seznam funkčních a nefunkčních požadavků, které jsou na základě kvantitativního průzkumu na aplikaci pro provádění Peer-to-Peer plateb kladeny.

5.3.1 Funkční požadavky

Funkční požadavky definují sadu funkcionalit, které bude systém svým uživatelům poskytovat[27]. Funkční požadavky aplikace pro provádění Peer-to-Peer

ID	Doména	Název požadavku	Implementováno v prototypu
FRQ1.01	Bezpečnost	Přihlášení uživatele do aplikace	Ano
FRQ1.02	Bezpečnost	Autorizace plateb	Ano
FRQ1.03	Bezpečnost	Autorizace žádostí	Ano
FRQ2.01	Zpracování transakcí	Zadání platby	Ano
FRQ2.02	Zpracování transakcí	Zadání žádosti o platbu	Ano
FRQ2.03	Zpracování transakcí	Odsouhlasení obdržené žádosti o platbu	Ano
FRQ2.04	Zpracování transakcí	Odmítnutí obdržené žádosti o platbu	Ano
FRQ2.05	Zpracování transakcí	Zobrazení přehledu transakcí	Ano
FRQ3.01	Ostatní	Úprava osobních údajů	Ano
FRQ3.02	Ostatní	Založení uživatelského účtu	Ano
FRQ3.03	Ostatní	Zobrazení telefonních kontaktů zařízení	Ano
FRQ3.04	Ostatní	Vytvoření nového kontaktu	Ano
FRQ3.05	Ostatní	Pozvání kontaktu k užívání aplikace	Ne

Tabulka 5.1: Přehled funkčních požadavků aplikace.

plateb jsou uvedeny v tabulce 5.1. Návaznost jednotlivých požadavků a jejich zařazení do domén je zobrazeno na diagramu 5.2.

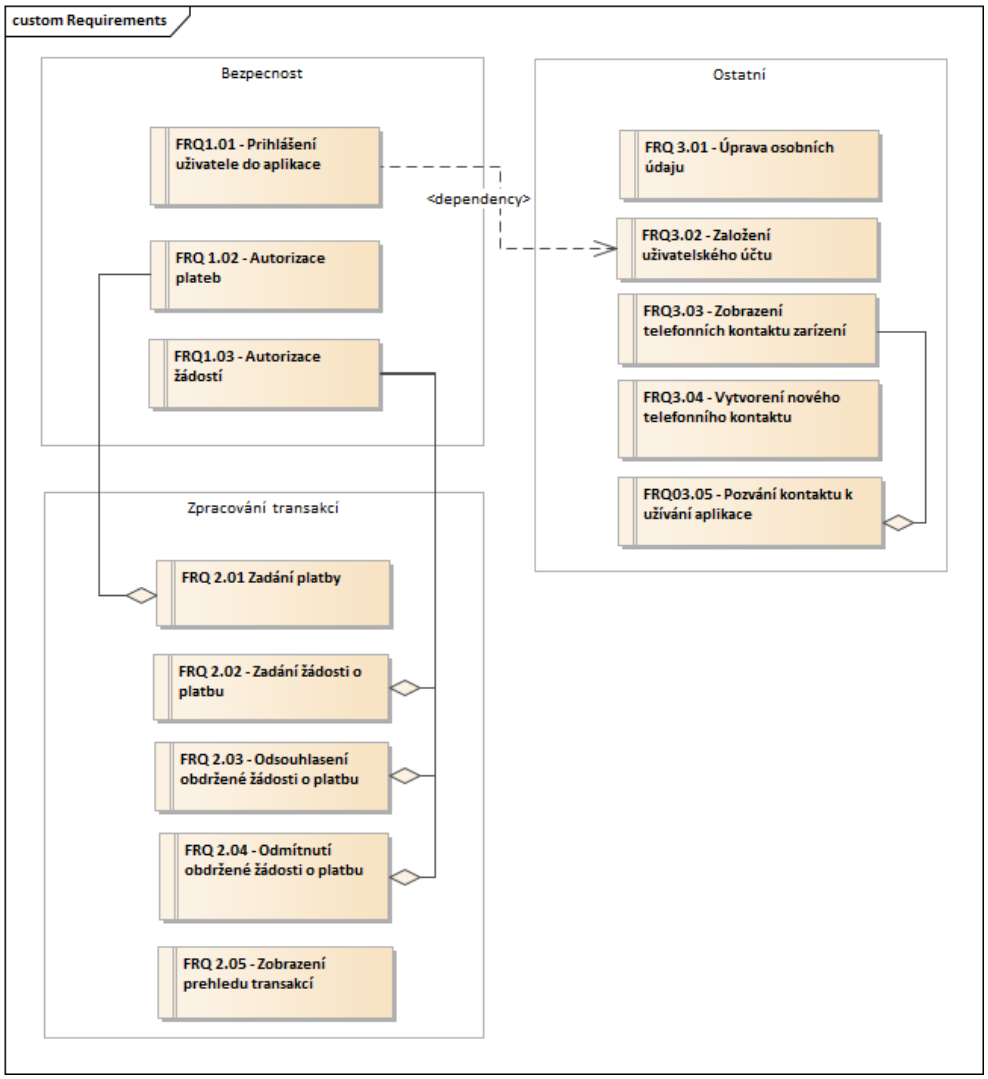
5.3.2 Nefunkční požadavky

Nefunkční požadavky z pohledu softwarového vývoje zachycují omezení, která jsou na systém kladená[27]. Seznam nefunkčních požadavků kladených na aplikaci pro provádění Peer-to-Peer plateb je uveden v tabulce 5.2.

Z uvedených funkčních a nefunkčních požadavků je evidentní, že implementovaný prototyp klade důraz především na splnění funkčních požadavků, které jsou klíčové pro možnost otestování aplikace na cílovém zařízení. Splnění nefunkčních požadavků je pak z větší části přesunuto až do projektové fáze produkčního vývoje.

5.4 Use case diagram

Use case diagram slouží k zachycení funkcionality, kterou systém nabízí uživatelům. Pomocí Use case diagramu jsme schopni popsat, jaké funkce bude



Obrázek 5.2: Diagram funkčních požadavků kladených na aplikaci.

ID	Název požadavku	Implementováno v prototypu
NRQ01	Aplikace musí fungovat na operačních systémech: Android, iOS	Ne
NRQ02	Zobrazování transakcí v reálném čase	Ano
NRQ03	Zabezpečení komunikace mezi zařízením a serverovou částí	Ne
NRQ04	Zabezpečení uložení dat v zařízení	Ne
NRQ05	Schopnost správně reagovat na změny v kontaktním seznamu zařízení	Ano

Tabulka 5.2: Přehled nefunkčních požadavků aplikace.

ID	Název Use Case	Pokrytý FRQ	Implementováno v prototypu
UC01	Vytvoření uživatelského účtu	FRQ3.02	Ano
UC02	Zadání platby	FRQ2.01, FRQ1.02	Ano
UC03	Odeslání žádosti o platbu	FRQ2.02, FRQ1.03	Ano
UC04	Zobrazení seznamu transakcí	FRQ2.05	Ano
UC05	Odpověď na Žádost o platbu	FRQ2.03, FRQ2.04, FRQ1.03	Ano
UC06	Vytvoření nového kontaktu	FRQ3.04	Ano
UC07	Pozvání kontaktu k používání aplikace	FRQ3.05	Ne
UC08	Změna osobních údajů	FRQ3.01	Ano

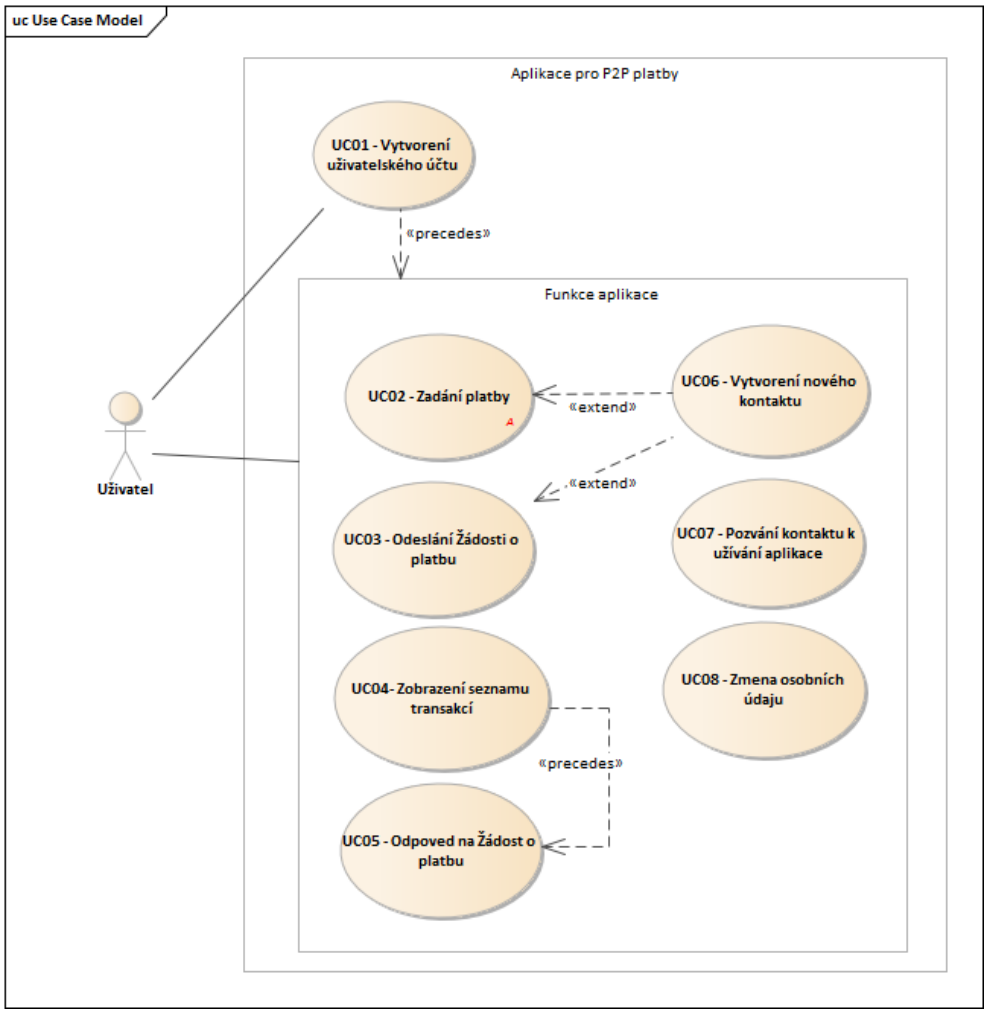
Tabulka 5.3: Přehled Use case případů aplikace.

systém uživateli poskytovat, neříká nám však nic o tom, jak bude systém tyto funkce provádět[27].

Use case model tak musí pokrývat veškeré funkční požadavky, které jsme v minulé sekci na aplikaci definovali. Seznam případů použití aplikace je zachycen v tabulce 5.3. Na obrázku 5.3 je zachycen Use case model včetně vzájemných vazeb.

5.4.1 UC01 Vytvoření uživatelského účtu

Aplikace musí umožňovat uživatelům vytváření uživatelských účtů, kterými se budou do aplikace přihlašovat. Případ použití je blíže popsán v tabulce 5.4.



Obrázek 5.3: Use case model funkcionality aplikace.

Název Use Case	UC01 - Vytvoření uživatelského účtu.
Aktéři	Uživatel Systém
Vstupní podmínky:	Uživatel má staženou a spuštěnou aplikaci.
Hlavní scénář	1. Uživatel vybere možnost Registrovat se. 2. Systém nabídne formulář pro vyplnění detailu účtu. 3. Uživatel vyplní požadované informace. 4. Systém provede kontrolu unikátnosti emailového kontaktu, telefonního kontaktu a čísla bankovního účtu. 5. Uživatel je informován o úspěšné registraci.
Výstupní podmínky	V systému je vytvořen nový uživatelský účet.

Tabulka 5.4: Popis UC01 - Vytvoření uživatelského účtu.

5.4.2 UC02 - Zadání platby

Zadání platby je naprosto základní případ použití aplikace, který umožňuje uživateli vytvořit platbu na kontakt uložený v mobilním zařízení. Po dokončení platby jsou převedeny finance z účtu iniciátora platby na účet příjemce. Případ použití je blíže popsán v tabulce 5.5.

Název Use Case	UC02 - Zadání platby.
Aktéři	Uživatel Systém
Vstupní podmínky:	Uživatel je přihlášen do aplikace. Uživatel se nachází na hlavní stránce aplikace.

Hlavní scénář	1. Uživatel vybere možnost zaplatit. 2. Systém zobrazí stránku se seznamem kontaktů v zařízení. 3. Uživatel vybere kontaktní osobu. 4. Systém zobrazí formulář pro vyplnění částky. 5. Uživatel vyplní částku Platby. 6. Systém zobrazí stránku se shrnutím Platby a s možností autorizace. 7. Uživatel autorizuje platbu pomocí svého PINu. 8. Systém zobrazí uživateli stránku s potvrzením úspěšně zadané Platby.
Výstupní podmínky	V systému je vytvořena nová Platba.

Tabulka 5.5: Popis UC02 - Zadání platby.

5.4.3 UC03 - Odeslání Žádosti o platbu

Odeslání Žádosti o platbu je druhou z klíčových funkcionalit, které aplikace pro provádění Peer-to-Peer plateb bude poskytovat. Jedná se o možnost připomenout se jinému uživateli o dlužnou částku. Případ použití je podrobně popsán v tabulce 5.6.

Název Use Case	UC03 - Odeslání Žádosti o platbu.
Aktéři	Uživatel Systém
Vstupní podmínky:	Uživatel je přihlášen do aplikace. Uživatel se nachází na hlavní stránce aplikace.

Hlavní scénář	1. Uživatel vybere možnost Připomenout se. 2. Systém zobrazí stránku se seznamem kontaktů v zařízení. 3. Uživatel vybere kontaktní osobu. 4. Systém zobrazí formulář pro vyplnění částky. 5. Uživatel vyplní částku Žádosti o platbu. 6. Systém zobrazí stránku se shrnutím Žádosti o platbu s možností autorizace. 7. Uživatel autorizuje platbu pomocí svého PINu. 8. Systém zobrazí uživateli stránku s potvrzením úspěšně odeslané Žádosti o platbu.
Výstupní podmínky	V systému je vytvořena nová Žádost o platbu.

Tabulka 5.6: Popis UC03 - Odeslání Žádosti o platbu.

5.4.4 UC04 - Zobrazení seznamu transakcí

Jedná se o velice jednoduchý případ použití, který umožňuje uživateli zobrazit a filtrovat příchozí a odchozí transakce, které proběhly v rámci dané aplikace. V seznamu se tedy nebudou vyskytovat transakce, které se týkají daného účtu, avšak neproběhly pomocí této aplikace. Případ použití je podrobněji rozepsán v tabulce 5.7.

Název Use Case	UC04 - Odeslání Žádosti o platbu.
Aktéři	Uživatel Systém
Vstupní podmínky:	Uživatel je přihlášen do aplikace Uživatel se nachází na hlavní stránce aplikace
Hlavní scénář	1. Uživatel filtruje transakce dle kategorie – odchozí platby, příchozí platby, nevyřízené transakce. 2. Systém zobrazuje žádané položky.

Výstupní podmínky	Systém zobrazuje seznam transakcí podle filtračního kritéria.

Tabulka 5.7: Popis UC04 - Zobrazení seznamu transakcí.

5.4.5 UC05 - Odpověď na Žádost o platbu

Jedná se o případ, který je velice úzce provázán s UC03 – Odeslání Žádosti o platbu. Jakmile uživatel obdrží od druhého uživatele Žádost o platbu, záleží na něm, zda si přeje Žádost potvrdit nebo naopak odmítnout. Tyto scénáře jsou blíže popsány v tabulce 5.8, respektive 5.9.

Název Use Case	UC05 - Odpověď na Žádost o platbu.
Aktéři	Uživatel Systém
Vstupní podmínky:	Uživatel je přihlášen do aplikace. Uživatel se nachází na hlavní stránce aplikace.
Hlavní scénář	1. Uživatel vyfiltruje nevyřízené platby. 2. Systém zobrazí seznam nevyřízených plateb. 3. Uživatel vybere nevyřízenou Žádost o platbu. 4. Systém zobrazí detail obdržené transakce. 5. Uživatel potvrdí a pomocí PINu autorizuje Žádost o platbu. 6. Systém vytvoří platbu a přesměruje uživatele na stránku s potvrzením platby.
Výstupní podmínky	Žádost o platbu byla potvrzena. Byla vytvořena nová platba.

Tabulka 5.8: UC05 - Odpověď na Žádost o platbu.

5.4.5.1 Alternativní scénář: Odmítnutí Žádosti o platbu

Vstupní podmínky:	Dle hlavního scénáře
Alternativní scénář	1-4. Dle hlavního scénáře 5. Uživatel odmítne žádost o platbu. 6. Systém přesměruje uživatele na stránku s potvrzením odmítnutní platby.
Výstupní podmínky	Žádost o platbu byla odmítnuta.

Tabulka 5.9: UC05 - Alternativní scénář - Odmítnutní Žádosti o platbu.

5.4.6 UC06 -Vytvoření nového telefonního kontaktu

Aplikace bude umožňovat uživateli procházet dostupné kontakty a v případě potřeby také vytvořit nový kontakt. Popis vytvoření nového kontaktu je blíže popsán v tabulce 5.10.

Název Use Case	UC06 - Vytvoření nového telefonního kontaktu
Aktéři	Uživatel Systém
Vstupní podmínky:	Uživatel je přihlášen do aplikace. Uživatel se nachází na hlavní stránce aplikace.
Hlavní scénář	1. Uživatel přejde na stránku s kontaktním listem. 2. Systém zobrazí kontaktní list zařízení. 3. Uživatel vybere možnost vytvořit nový kontakt. 4. Systém zobrazí formulář pro zadání nového kontaktu. 5. Uživatel vyplní zobrazený formulář. 6. Systém uloží do kontaktního seznamu zařízení nový kontakt.
Výstupní podmínky	Nový kontakt byl uložen do zařízení.

Tabulka 5.10: UC06 - Vytvoření nového telefonního kontaktu.

5.4.7 UC07 - Pozvání kontaktu k používání aplikace

Aplikace bude umožňovat posílat pozvánku k užívání aplikace kontaktům pomocí SMS zprávy, později též pomocí napojení na sociální sítě. Možnost sdílení aplikace pomocí SMS zprávy je podrobněji popsána v tabulce 5.11.

Název Use Case	UC07 - Pozvání kontaktu k používání aplikace.
Aktéři	Uživatel Systém
Vstupní podmínky:	Uživatel je přihlášen do aplikace. Uživatel se nachází na hlavní stránce aplikace.
Hlavní scénář	1. Uživatel přejde na stránku s kontaktním listem. 2. Systém zobrazí kontaktní list zařízení. 3. Uživatel zvolí kontakt, který není vedený v aplikaci. 4. Systém zobrazí detail daného kontaktu. 5. Uživatel klikne na tlačítko sdílet. 6. Systém zobrazí šablonu SMS zprávy k odeslání vybranému kontaktu. 7. Uživatel SMS zprávu odešle.
Výstupní podmínky	SMS zpráva byla odeslána vybranému kontaktu.

Tabulka 5.11: UC07 - Pozvání kontaktu k používání aplikace.

5.4.8 UC08 - Změna osobních údajů

Aplikace bude umožňovat uživateli měnit osobní údaje, které při registraci do aplikace zadal. Popis daného případu užití je uveden v tabulce 5.12.

Název Use Case	UC08 - Změna osobních údajů.
Aktéři	Uživatel Systém
Vstupní podmínky:	Uživatel je přihlášen do aplikace. Uživatel se nachází na hlavní stránce aplikace.

Hlavní scénář	1. Uživatel přejde na stránku s osobním nastavením. 2. Systém zobrazí aktuální informace uživatele. 3. Uživatel upraví údaje a potvrdí uložení. 4. Systém uloží změněné údaje a potvrdí uživateli jejich aktualizaci.
Výstupní podmínky	V Systému jsou uživatelské údaje aktualizovány.

Tabulka 5.12: UC08 - Změna osobních údajů.

5.5 Stavový diagram

Stavový diagram slouží k zachycení jednotlivých stavů, kterými entita prochází v průběhu svého životního cyklu. Přechody mezi jednotlivými stavy jsou způsobeny vnějšími nebo vnitřními událostmi dané entity[28].

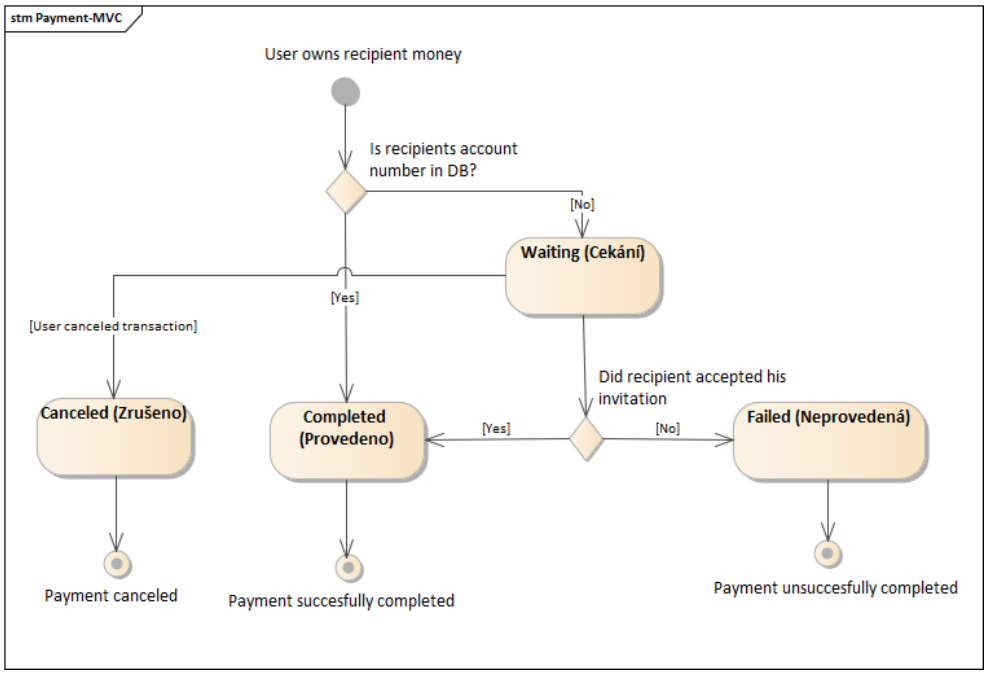
Z pohledu domény aplikace pro provádění Peer-to-Peer plateb je důležité sledovat stav entit Platba a Žádost o platbu. Nutnost pro sledování těchto stavů jsme zdůraznili také v doménovém modelu, kde jsme ke každé transakci navázali třídu Stav.

5.5.1 Stavový diagram entity Platba

Hlavní problém, který při zadávání nové platby může nastat, je v případě, kdy kontakt, na který se uživatel snaží platbu zadat, není uživatelem aplikace. V tomto případě není možné provést překlad kontaktu na číslo bankovního účtu a nejsme schopni transakci provést. V prototypu není povoleno zadávat platby na kontakty, které nejsou v aplikaci uvedeny, v produkční aplikaci toto však musí být vyřešeno.

V takovémto případě se tedy vytvoří nová platba, u které je nastaven stav Čekání a na kontaktní číslo je odeslána zpráva s odkazem na webový formulář. Do tohoto formuláře následně příjemce vyplní své číslo účtu a platbu potvrdí. V tuto chvíli již aplikace zná číslo účtu příjemce a je možné platbu dokončit. Entita Platba se tak přesune ze stavu Čekání do stavu Provedeno. Musíme však také uvažovat, že příjemce webového formuláře nemusí chtít finanční prostředky touto cestou zaslat. Místo zadání svého čísla účtu tak má možnost platbu odmítnout. Platba se tak ze stavu Čekání přesune do stavu Neprovedeno.

Stejně jako příjemce webového formuláře, i plátce si může svou čekající platbu rozmyslet. Z tohoto důvodu je třeba plátci umožnit zrušení platby



Obrázek 5.4: Stavový diagram entity Platba.

ve stavu Čekání do stavu Zrušeno. Zrušení čekající platby v tomto případě bude probíhat skrze aplikaci a je nutné webový formuláři příjemce jasně označit jako zrušený. Pro tento scénář není použit stav Neprovedeno, aby bylo jasně odlišitelné, která strana inicializovala zrušení platby.

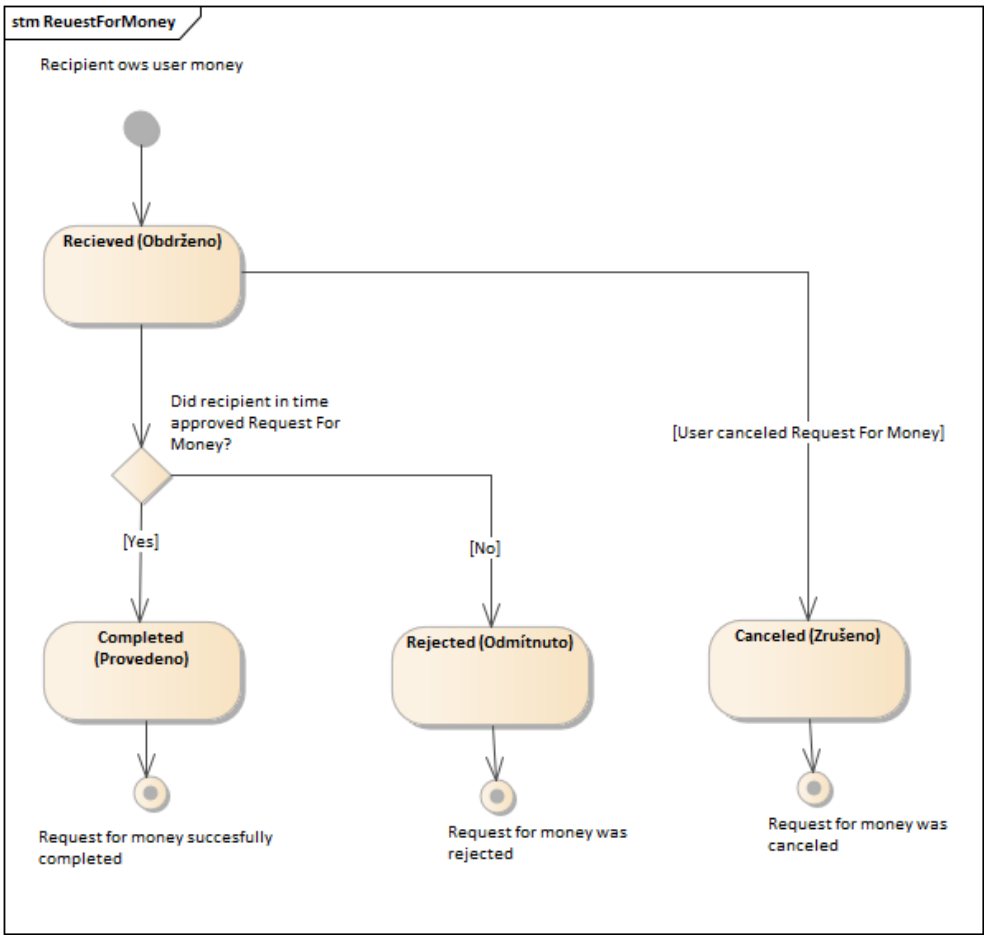
Poslední a nejčastější scénář je ten, kdy je kontakt příjemce uložen v databázi aplikace. V tomto případě se platba úspěšně provede a je dokončená ve stavu Provedeno. Přechody mezi jednotlivými stavy jsou zachyceny na diagramu 5.4.

5.5.2 Stavový diagram entity Žádost o platbu

V rámci prototypu aplikace budeme povolovat zaslání Žádosti o platbu pouze uživatelům aplikace. Po vytvoření a autorizaci Žádosti o platbu se entita vždy přesune do stavu Obdrženo – druhá strana obdržela žádost, avšak ještě ji nijak nezpracovala.

Odesílatel žádosti musí mít možnost odeslanou Žádost zrušit. V takovémto případě bude žádost přesunuta do stavu Zrušeno a příjemci se již nadále nebude zobrazovat.

Příjemce Žádosti o platbu bude mít dvě možnosti, jak příchozí žádosti zpracovat. První možností je přijetí Žádosti a autorizace platby. V tomto případě se k dané žádosti vytvoří nová platba a Žádost se přesune do stavu



Obrázek 5.5: Stavový diagram entity Žádost o platbu.

Provedeno. Druhou možností je odmítnutí Žádosti o platbu. V tomto případě se žádná platba neuskuteční a Žádost o platbu se přesune do stavu Odmítnuto.

Podobně jako v případě entity Platba, stavy Zrušeno a Odmítnuto jsou zde odlišeny z toho důvodu, aby bylo jednoznačně identifikovatelné, která strana inicializovala odmítnutí Žádosti o platbu. Stavový diagram entity Žádost o platbu je zachycen na diagramu 5.5.

5.6 Architektura aplikace

Dosud jsme se v rámci našeho návrhu soustředili především na definování business stránky aplikace – tedy komu a k čemu má sloužit. V této části návrhu se naopak blíže podíváme na technický návrh aplikace, tedy jak bude

požadované funkce vykonávat. V této části bude důraz kladen na architekturu samotné aplikace, která bude přes vlastní rozhraní s API bank komunikovat.

5.6.1 Rozdělení aplikace

Mobilní aplikace pro provádění Peer-to-peer plateb bude mít tři důležité části. První částí bude klientská část aplikace, kterou si uživatel bude moci stáhnout do svého mobilního telefonu pomocí mobilního obchodu (Google Play / App Store). Tato část mobilní aplikace bude spouštěna přímo na mobilním zařízení uživatele, kam také bude ukládat svá data. Z tohoto důvodu je nutné omezit velikost této části aplikace, abychom zbytečně neplýtvali dostupnou pamětí cílového zařízení.

Druhou důležitou aplikační vrstvou je serverová část, která bude spuštěna na externím serveru. Tato část aplikace bude přijímat HTTP požadavky z klientské části aplikace, vykonávat aplikační logiku, získávat a upravovat data v databázi, zasílat požadavky na bankovní API a posílat upozornění na klientskou část aplikace. Poslední, třetí vrstvou aplikace, je databáze. Databáze se stará o ukládání, úpravy a zpracování dat zaslaných ze serverové části aplikace.

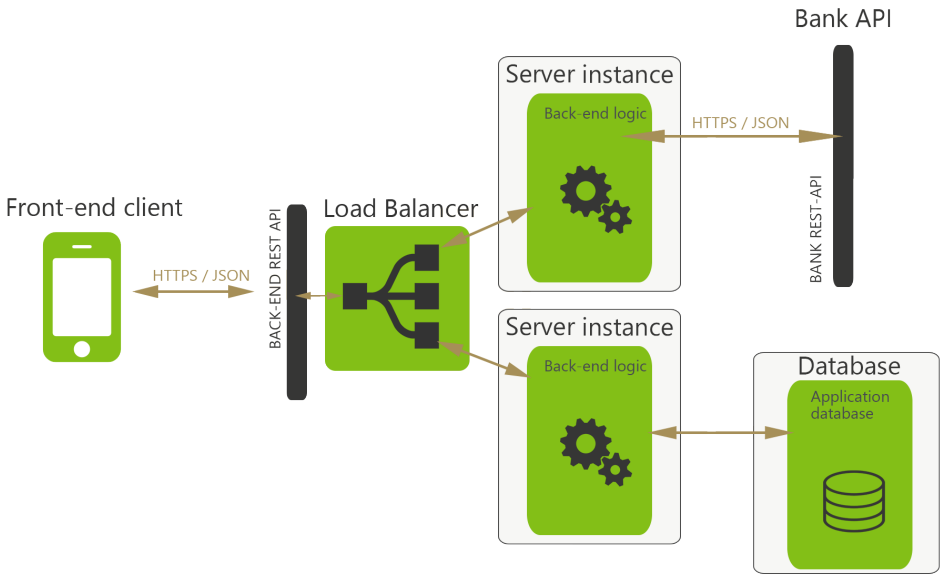
5.6.2 Load balancing (Vyvažování zátěže)

Load balancing je technika pro efektivní distribuování zátěže mezi skupinu aplikačních serverů k tomu určených [9]. Tato technika se u aplikací se serverovou částí zavádí ze dvou hlavních důvodů:

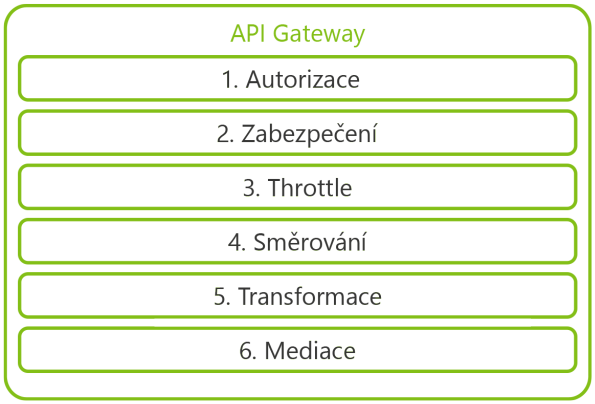
1. Distribuce požadavků z důvodu vytížení serveru. Servery s vysokým provozem musí být schopny obsloužit statisíce až miliony požadavků v krátkém čase. Z tohoto důvodu je nutné rozdělit zátěž mezi více serverových instancí, které jsou společně schopny tento provoz obsloužit.

2. Zajištění dostupnosti služby. Pro mnoho služeb je nezbytná jejich dostupnost a nemohou si dovolit ani krátké výpadky. To však s jedním serverem není možné. Pokud tedy chceme zaručit určitou dostupnost služby, je nutné se uchýlit k horizontálnímu škálování serverových instancí a k load balancingu.

Samotný load balancer je tedy síťový prvek, který se stará o distribuci příchozích požadavků mezi dostupné servery, které jsou schopné tyto požadavky obsloužit. Z pohledu této aplikace je load balancing klíčový především z pohledu dostupnosti služby. Jelikož se jedná produkt zajišťující provádění bankovních převodů, je nutné veškeré výpadky služby minimalizovat pomocí horizontálního škálování a load balancingu. Návrh architektury aplikace je zachycen na diagramu 5.6.



Obrázek 5.6: Pohled na architekturu aplikace.



Obrázek 5.7: Pohled na jednotlivé funkce rozhraní.

5.6.3 Komunikace v rámci aplikace

V teoretické části, v kapitole 3.3, jsme představili a porovnali dvě základní architektury webových služeb: REST a SOAP. U této aplikace můžeme rozlišit dva hlavní kanály, v rámci kterých bude probíhat komunikace jednotlivých částí aplikace:

- Komunikace klientské části aplikace se serverovou částí aplikace.
- Komunikace serverové části aplikace s bankovními rozhraními.

Navržení druhé zmíněné komunikace nezáleží na nás, neboť naše aplikace bude pouze konzumovat rozhraní vystavené jednotlivými bankami. Toto rozhraní dosud není veřejné, avšak bude se jednat o architekturu typu REST. Abychom omezili nutnost mediace a transformace vyžadované na serverové straně, s myšlenkou architektury REST navrhneme také rozhraní serverové části aplikace. Komunikace mezi klientskou a serverovou částí tak bude probíhat pomocí HTTPS a data budou přenášená ve formátu JSON. To nám zároveň značně zjednoduší vytváření požadavků na straně mobilního zařízení.

5.6.4 API rozhraní aplikace

Požadavky z klientské části aplikace jsou směrovány na API serverové části aplikace, přes které probíhá vzájemná komunikace. Toto rozhraní, zachycené na diagramu 5.7, má v rámci zajišťování komunikace následující funkce:

1. Autorizace: rozhraní provádí ke každému obdrženému požadavku nebo množině požadavků autorizaci, zda má žadatel oprávnění o danou službu žádat. V případě nedostatečných oprávnění vrací rozhraní chybový kód a požadavek není směrován dále. V případě úspěšné autorizace jsou informace spojené s autorizací předány dále. Díky tomuto je možné na cílové službě, nebo zdroji, implementovat vlastní logiku pro práci s uživateli.

2. Zabezpečení: rozhraní se stará o zabezpečení dat, která jsou posílána mezi klientskou částí aplikace a rozhraním. Tato komunikace musí být zabezpečená pomocí SSL. Rozhraní se také stará o odstranění bezpečnostních prvků, které pro volání interních služeb nejsou nutné.

3. Throttle: rozhraní si musí být vědomé omezení jednotlivých služeb, na které požadavky směruje, a podle aktuálního vytížení požadavky správně distribuovat. V našem případě se o toto směrování namísto rozhraní stará load balancer.

4. Směrování: rozhraní spolupracuje s registry služeb, případně speciálními procesy pro objevování služeb, s cílem získat aktuální endpoint požadované služby. Důvodem pro toto je flexibilita rozhraní v dynamickém prostředí, kde služby zanikají a vznikají.

5. Transformace a mediace: u mikroslužeb je běžné, že data, která přijímají nebo vracejí, se nemusí vždy shodovat s formátem dat, který vyžaduje klientská část aplikace. V takovém případě je možné na rozhraní provádět transformaci dat, aby odpovídala požadovanému formátu a syntaxi. V našem případě budeme předávat data ve formátu JSON a na straně klienta i mikroslužeb budeme očekávat stejnou strukturu. Obejdeme se tak bez transformací.

5.6.5 Rozdělení serverové logiky pomocí mikroslužeb

V rámci teoretické části jsme si představili architekturu mikroslužeb, na základě které postavíme logiku serverové části aplikace. Abychom byli schopni provést tuto dekompozici je nutné nejprve definovat, které funkcionality musíme mikroslužbami pokrýt:

- správa uživatelských účtů,
- kontaktní seznam, překlad telefonního čísla na bankovní účet,
- zadávání Plateb,
- zadávání Žádosti o platbu.

Podle takto rozdělených domén jsme schopni rozdělit serverovou logiku aplikace do mikroslužeb uvedených v tabulce 5.13. Na obrázku 5.8 je zachyceno rozdělení serverové části aplikace pomocí mikroslužeb. V následujících sekcích jsou blíže popsány jednotlivé mikroslužby včetně logiky, kterou mají na starost.

ID	Název služby
MS01	userMicroservice
MS02	contactListMicroservice
MS03	paymentMicroservice
MS04	requestMicroservice

Tabulka 5.13: Seznam mikroslužeb, které budou pokrývat logiku serverové části aplikace.

5.6.5.1 MS01 - userMicroservice

Mikroslužba userMicroservice řídí práci s uživatelskými účty aplikace, na starost bude mít následující akce:

1. Přihlašování uživatele - uživatelé aplikace budou autentizováni za pomocí uživatelského jména a hesla. Zaslané údaje budou porovnány se záznamy uloženými v databázi a v případě shody bude uživateli navrácen token, který bude sloužit pro autorizaci následujících požadavků.

2. Odhlašování uživatele - uživatelé budou mít možnost se z aplikace odhlásit na základě jednoduchého REST požadavku. Při úspěšném odhlášení dojde ke zrušení tokenu, kterým se uživatel dříve autorizoval.

3. Načtení uživatelských dat - akce bude sloužit pro získání uživatelských údajů, typicky bude prováděna během inicializace aplikace. Součástí načtených dat budou také seznamy příchozích a odchozích Plateb, respektive Žádostí o platbu. Tyto údaje budou získány z mikroslužeb paymentMicroservice a requestMicroservice.

4. Registrování uživatele - tato akce bude mít na starost logiku spojenou s registrováním nových uživatelů do aplikace. Před vytvořením nového uživatelského účtu je nutné provést kontrolu, zda zaslané údaje - email, telefon, číslo bankovního účtu, jsou v aplikaci unikátní. Validaci unikátnosti bude mít na starost mikroslužba contactListMicroservice. Pokud údaje jsou unikátní, proběhne ještě kontrola, zda je registrovaná osoba skutečně majitelem uváděných kontaktních údajů a následně dojde k vytvoření nového uživatelského účtu.

5. Změna uživatelských informací - uživatelé si mohou pomocí aplikace měnit své osobní údaje, které v průběhu registrace do aplikace uvedli. V případě žádosti o změnu unikátních údajů je nutné provést stejné mechanismy kontroly, jaké jsou uvedeny u akce Registrování uživatele.

5.6.5.2 MS02 - contactListMicroservice

Mikroslužba MS02 – contactListMicroservice spravuje kompletní seznam všech kontaktů, které jsou v aplikaci uložené a má na starosti s tím spojené akce:

1. Vytvoření nového kontaktu - akce, která má na starosti zakládání nových kontaktů v aplikaci. Tato operace je volána z mikroslužby userMicroservice v rámci registrování nového uživatele do aplikace.

2. Získání detailu kontaktu dle ID - operace sloužící pro získání detailu kontaktu na základě zaslaného ID.

3. Získání detailu kontaktu dle telefonního čísla - pro volání akce je nutné do těla požadavku vložit telefonní číslo, ke kterému mikroslužba vrátí detail kontaktu.

4. Překlad telefonního čísla na číslo bankovního účtu - akce bude volána z mikroslužeb paymentMicroservice, respektive requestMicroservice, při zakládání nových transakcí.

5. Kontrola unikátnosti údajů - akce sloužící pro ověření, zda jsou zaslané údaje - emailový kontakt, telefonní kontakt a číslo bankovního účtu - unikátní.

6. Změna kontaktu - operace sloužící pro změnu detailu kontaktu. Tato operace bude volána z mikroslužby userMicroservice při změně uživatelských údajů.

5.6.5.3 MS03 - paymentMicroservice

Mikroslužba MS03 – paymentMicroservice slouží ke zpracování zadaných Plateb a s nimi spojené logiky.

1. Vytvoření nové Platby - operace sloužící pro zadání nové Platby. V rámci operace dochází ke kontrole, zda má odesílatel na účtu dostatečný zůstatek. Následně dojde k překledu telefonního čísla příjemce na číslo bankovního účtu pomocí mikroslužby contactListMicroservice. Poté operace poskládá správný požadavek, který je odesílán na bankovní rozhraní uživatelovy banky. Na závěr jsou oba účastníci transakce informování o nové Platbě.

2. Získání odesílaných Plateb - akce slouží pro získání veškerých Plateb, ve kterých uživatel se zadaným ID vystupuje jako odesílatel.

3. Získání příchozích Plateb - operace slouží pro získání veškerých Plateb, ve kterých uživatel se zadaným ID vystupuje jako příjemce.

5.6.5.4 MS04 - requestMicroservice

Mikroslužba MS04 – requestMicroservice slouží ke zpracování zadaných Žádostí o platbu. Mikroslužba obsahuje následující akce:

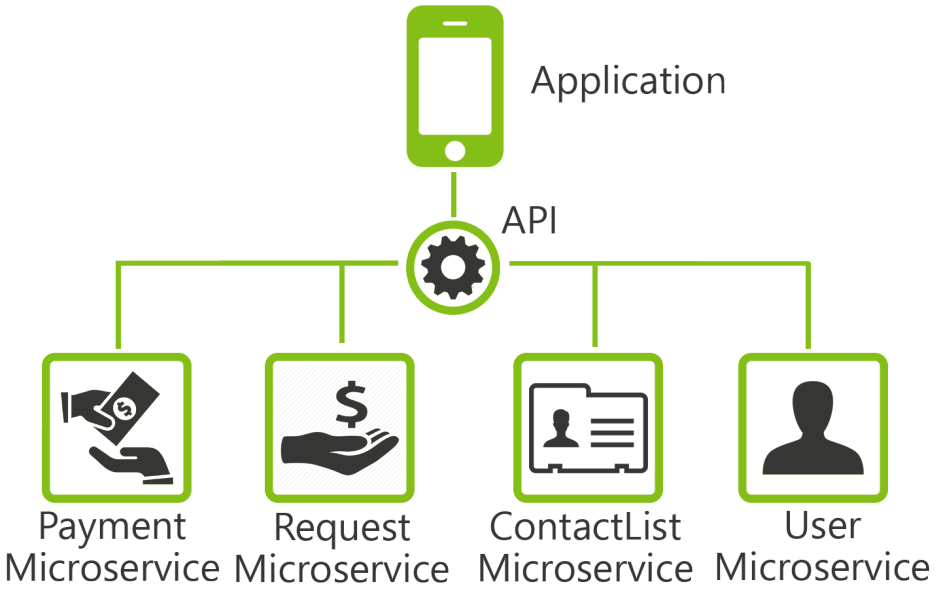
1. Vytvoření Žádosti o platbu - operace slouží pro zadání Žádosti o platbu. Operace vyhledá příjemce pomocí operace mikroslužby contactListMicroservice a příjemci odešle informaci o nové Žádosti o platbu.

2. Schválení Žádosti o platbu - akce slouží pro dokončení Žádosti o platbu v případě, kdy ji uživatel potvrdí. Následně je volána mikroslužba paymentMicroservice a oba účastníci jsou informováni o provedené Platbě.

3. Zrušení Žádosti o platbu - operace dokončí Žádost o platbu v případě, kdy odesílatel svoji Žádost zrušil. Žádost je přesunuta do stavu Zrušeno a oba účastníci transakce jsou o změně stavu informováni.

4. Odmítnutí Žádosti o platbu - akce slouží k dokončení Žádosti o platbu v případě, kdy příjemce obdrženou Žádost odmítnul. Žádost je přesunuta do stavu Odmítnuto a oba účastníci transakce jsou o změně stavu informováni.

5. Získání odesílaných Žádostí o platbu - akce vrací veškeré Žádosti o platbu, ve kterých uživatel se zadaným ID vystupuje jako odesílatel.



Obrázek 5.8: Architektura mikroslužeb navržená pro mobilní aplikaci.

6. Získání přijatých Žádostí o platbu - operace získá veškeré Žádosti o platbu, ve kterých uživatel se zadaným ID vystupuje jako příjemce.

5.7 Datový model

Další téma, kterým je nutné se v rámci analýzy zabývat, je datový model aplikace, tedy jakým způsobem a v jakém formátu budeme data ukládat. Nemusíme se však zabývat konkrétním schématem databáze, neboť pro naši aplikaci nepoužijeme běžnou relační databázi, nýbrž databázi dokumentovou. K volbě dokumentové databáze se přikloníme především z následujících důvodů:

1. Možnost horizontálního škálování – dokumentové databáze obsahují úmyslné redundance dat, které zvyšují odolnost systému proti výpadkům a umožňují operace s velkými objemy dat v krátkém čase. Dokumentové databáze jsou často distribuované v cloudu. Relační databáze jsou naopak normalizované, tedy neobsahují redundance, a jsou škálovatelné pouze vertikálně.

2. Problematické datové typy – dokumentové databáze umožňují ukládat data ve formátu klíč-hodnota. Jsme tedy schopni ukládat celé objekty

a kolekce objektů. Naopak v relačních databázích je možné ukládat data pouze ve formátu základních datových typů.

3. Iterativní vývoj – použití relační databáze vyžaduje na začátku přesně definovat jasné databázové schéma. Jakékoliv následné zásahy do databázového schématu jsou s ohledem na uložená data velice problematické. Dokumentové databáze naopak nemají žádné schéma, a proto jsou mnohem lépe schopny reagovat na měnící se požadavky aplikace.

5.7.1 Návrh dokumentového modelu

Namísto tabulek se v dokumentové databázi vše ukládá do kolekcí, jednotlivé řádky tabulky jsou pak reprezentovány dokumenty. Není tedy nutné modelovat složité relační modely, avšak je potřeba rozhodnout, jaké kolekce budeme v aplikaci ukládat a jaká data ukládané dokumenty ponesou. Na rozdíl od relačních databází by pak každý dokument měl být, co do uložených dat, samostatný. V tomto přístupu je návrh dokumentové databáze značně odlišný od návrhu relační databáze. Při definici potřebných kolekcí dokumentů vyjdeme z vypracovaného UseCase modelu a identifikujeme klíčové celky:

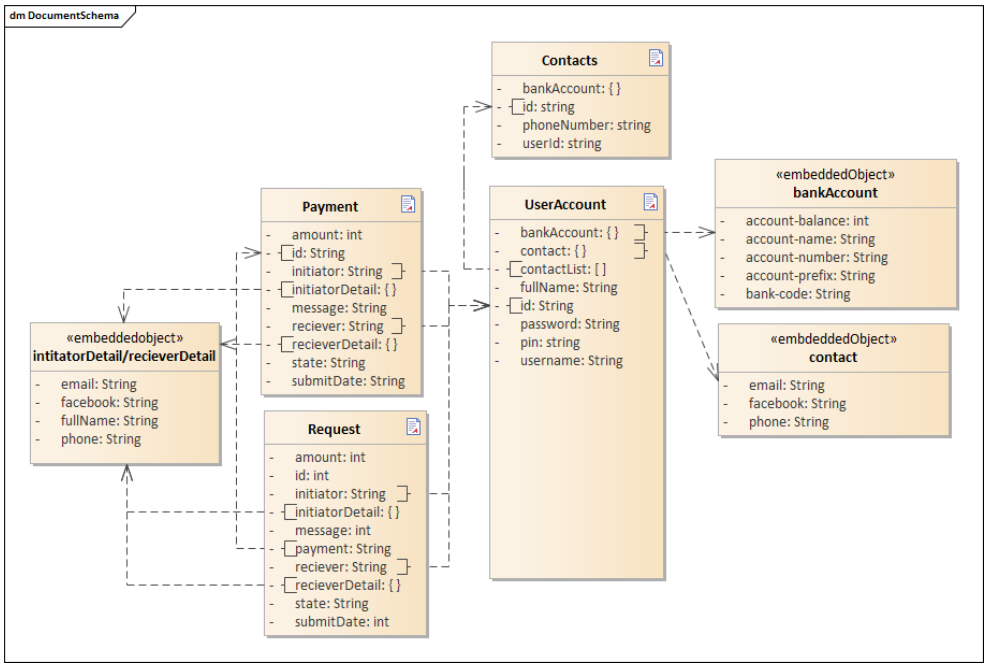
UserAccount — První kolekcí je kolekce UserAccount, která bude mít na starosti uživatelské účty. Dokumenty v této kolekci budou reprezentovat účty jednotlivých uživatelů.

Payment — Kolekce Payment bude obsahovat všechny platby, které byly pomocí aplikace uskutečněny. Aby bylo možné platbu spojit s účty mezi kterými proběhla, jsou zde klíče initiatorDetail a requestDetail, které v sobě obsahují referenci na dokument UserAccount, ke kterému se vztahují.

Request — Kolekce Request bude obsahovat veškeré Žádosti o platbu, které pomocí aplikace byly uskutečněny. Aby bylo možné Žádost o platbu spojit s konkrétními účty, každý dokument obsahuje referenci na UserAccount, ke kterému se vztahují. Navíc je zde vedena také reference na Platbu, pokud byla Žádost příjemcem akceptována a zaplacena.

Contacts — Kolekce Contacts obsahuje kompletní kontaktní seznam všech uživatelů aplikace. Pomocí referencí na dokumenty v této kolekci je tak možné pro každého uživatele udržovat v aplikaci uložený kompletní telefonní seznam a reagovat na případné změny.

Kompletní model navržených kolekcí včetně ukládaných dat je zobrazen na diagramu 5.9.



Obrázek 5.9: Návrh dokumentového modelu aplikace.

5.8 Aplikační procesy

Na úvod analýzy Peer-to-peer aplikace jsme definovali funkční požadavky, které bude aplikace splňovat. Následně jsme tyto funkční požadavky pokryli pomocí jednotlivých user-casů a navrhli architekturu aplikace, která bude danou funkcionalitu umožňovat. Na závěr návrhu je nutné se podívat na aplikaci z procesního pohledu a definovat, jakým způsobem bude požadované funkcionality dosaženo. V této kapitole se proto detailněji podíváme, jakým způsobem budou probíhat následující procesy, které je možné v aplikaci identifikovat:

ID	Název procesu	Pokrytý UC
P01	Vytvoření uživatelského účtu	UC01
P02	Přihlášení uživatele do aplikace	-
P03	Zadání Platby	UC02
P04	Zadání Žádosti o platbu	UC03
P05	Vyřešení Žádosti o platbu	UC05

Tabulka 5.14: Seznam procesů aplikace pro provádění Peer-to-Peer plateb.

5.8.1 P01 - Vytvoření uživatelského účtu

Vytvoření uživatelského účtu je první proces, se kterým se uživatel po stažení aplikace setká a zároveň se jedná o nejsložitější proces celé aplikace. Hlavním důvodem pro tuto komplexitu je nutnost ověření, že žadatel je skutečně majitelem zadaných údajů a následné vytvoření příslušných dokumentů v databázi.

Při tvorbě nového uživatelského účtu je nutné od uživatele získat následující osobní údaje, společně s jeho souhlasem ke zpracování daných údajů pro účely aplikace:

- Email – bude sloužit jako uživatelské jméno, pod kterým se bude hlásit do aplikace. Email bude zároveň sloužit jako kanál komunikace mezi poskytovatelem služby a uživatelem.
- Telefonní číslo – bude sloužit k překladu na číslo účtu.
- Jméno, příjmení a rodné číslo – bude sloužit pro jednoznačnou identifikaci uživatele.
- Číslo bankovního účtu – číslo účtu, na které bude telefonní číslo překládáno.
- Heslo pro vstup do aplikace – heslo, kterým se bude uživatel autentizovat při vstupu do aplikace.
- PIN – kód, kterým bude uživatel autorizovat finanční transakce.

Tyto informace budou uloženy společně s uživatelským účtem a budou během chodu aplikace aktivně využívány. Před jejich uložením a vytvořením uživatelského účtu je však nutné provést dodatečné kontroly, zda jsou uvedené údaje skutečně pravdivé.

Z tohoto důvodu je nutné pomocí HTTPS požadavku poslat informace na aplikační server, kde jsou implementovany bezpečnostní mechanismy. V první řadě je třeba provést kontrolu unikátnosti přihlašovacích údajů. Z tohoto pohledu musí být unikátní:

- telefonní číslo,
- číslo bankovního účtu,
- emailový účet.

Pokud je splněna podmínka na unikátnost jednotlivých údajů, je nutné provést ověření, zda vyplněné údaje skutečně patří danému žadateli o účet. Toto ověření probíhá ve dvou krocích.

5.8.1.1 Ověření uživatelských údajů

V rámci ověřování uživatelských údajů bude aplikace kontrolovat, zda zadaný emailový a telefonní kontakt skutečně patří dané osobě. Kontrolu, zda bankovní účet patří dané osobě není nutné, neboť se jedná o složitou a v rámci aplikace nezneužitelnou záležitost.

Pro ověření emailového kontaktu vygeneruje aplikační server zdroj, který bude vystaven pod URL zakončenou dvaceti náhodnými znaky. Na zadaný emailový kontakt je následně odeslán email s informačním sdělením a odkazem na vystavený zdroj. V případě, kdy uživatel klikne na URL vystaveného zdroje, aplikační server obdrží GET požadavek na daný zdroj a emailový kontakt je tak považován za ověřený.

Druhým krokem v ověřování je kontrola, zda telefonní číslo skutečně patří žadateli o účet. Pro autentizaci aplikační server pomocí SMS brány odešle SMS zprávu, která je zaslána na zadané telefonní číslo. Součástí SMS je náhodně vygenerovaný kód, který je nutné zadat do mobilní aplikace a odeslat na aplikační server, kde dochází k ověření správnosti.

5.8.1.2 Vytvoření uživatelského účtu

Pokud proběhlo ověření zadaných kontaktů úspěšně, je vytvořen nový dokument v kolekci UserAccount, který obsahuje veškeré zadané údaje. Následně je nutné nový záznam uložit také do kolekce Contacts, aby ostatní uživatelé mohli k danému kontaktu vytvářet transakce. Tímto je uživatelský účet nového uživatele úspěšně dokončen. Proces je schematicky zachycen na diagramu 5.10.

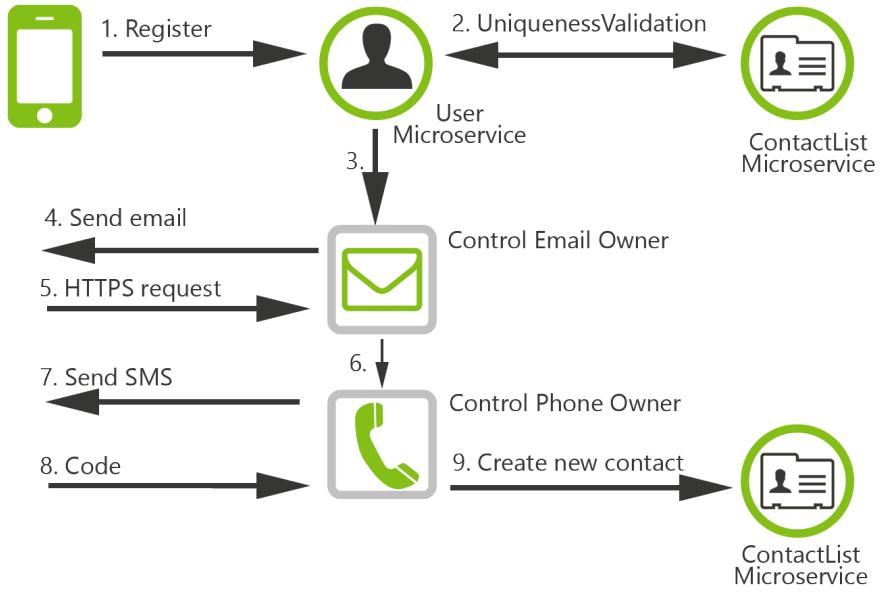
5.8.2 P02 - Přihlášení uživatele do aplikace

Jakmile má uživatel vytvořený uživatelský účet, může se do aplikace přihlásit pomocí svého emailového účtu, který slouží zároveň jako uživatelské jméno, a uživatelského hesla. Odeslaný požadavek je směrován na mikroslužbu userMicroservice, která mezi uživatelskými účty vyhledá ten, kterému odpovídá zadané uživatelské jméno a následně ověří, zda se shodují uživatelská hesla. Pokud kterékoliv z těchto operací skončí neúspěšně, pak je aplikaci vrácena chyba 401 – Chybné přihlašovací údaje. Přihlášení do aplikace je zachyceno na diagramu 5.11.

V případě úspěšného přihlášení musí aplikace provést dva klíčové úkony:

1. Získat aktualizované informace o účtu.

2. Získat aktualizovaný kontaktní list zařízení.



Obrázek 5.10: Proces registrace uživatele do aplikace.

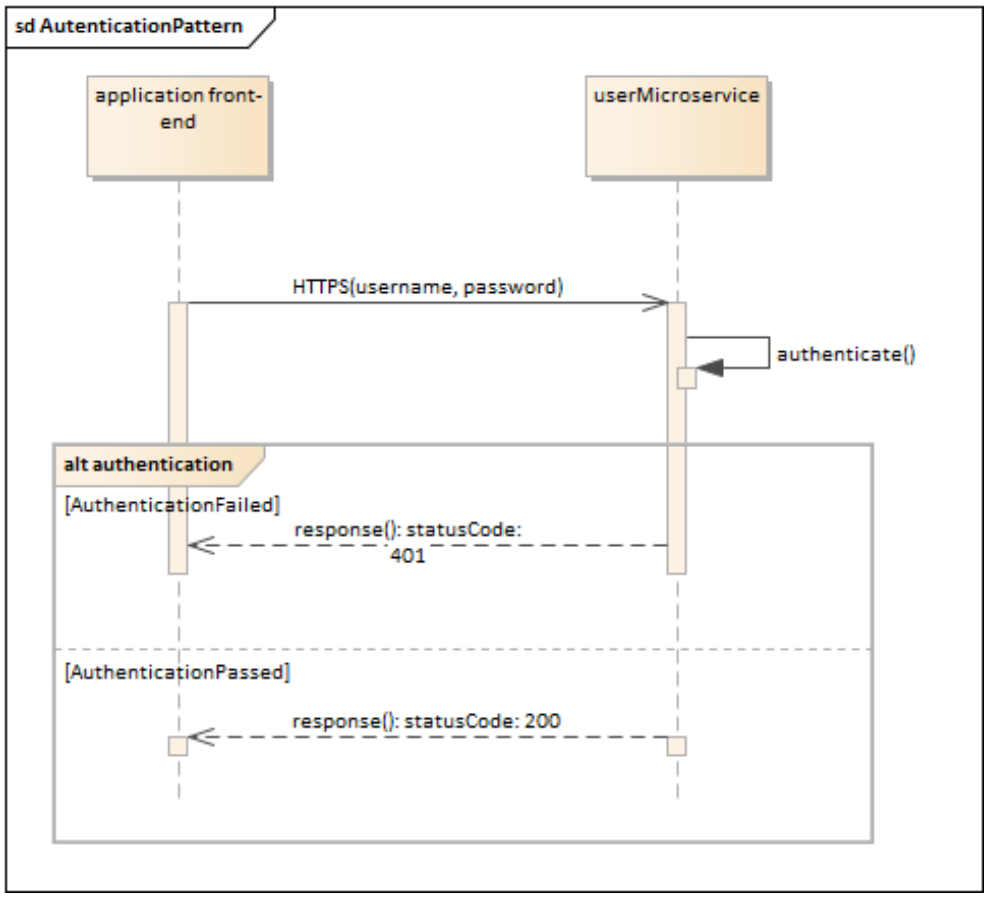
5.8.2.1 Aktualizace informací o účtu

Aby došlo k aktualizaci informací o účtu, zavolá aplikace mikroslužbu userMicroservice, která zajišťuje navrácení aktuálních informací o účtu. V první řadě musí služba zjistit zůstatek na účtu, za tímto účelem je vytvořen požadavek GetAccount na API příslušné banky. Dále je nutné získat kolekce Plateb a Žádostí o platby, které se týkají daného účtu. Pro tyto účely nám slouží reference na ID účtu u každého dokumentu v kolekcích Platba a Žádost o platbu. O získání těchto údajů se postarají mikroslužby paymentMicroservice a requestMicroservice. Jakmile jsou všechny akce dokončené, je aplikaci navrácena odpověď, se strukturou uvedenou v příloze B, v sekci B.1.

Získaná data jsou v aplikaci uložena do paměti zařízení pro následné využití. Podproces získání detailu účtu je zachycen na diagramu 5.12.

5.8.2.2 Aktualizování kontaktního seznamu

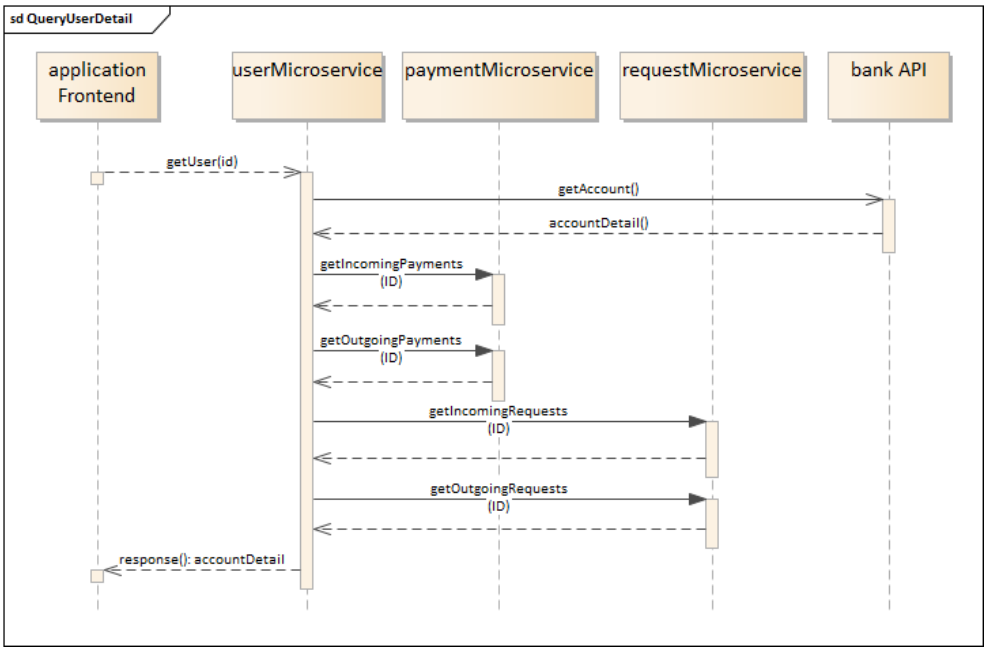
Aby uživatelé mohli odesílat peníze svým kontaktům, je nutné pracovat s kontaktním seznamem zařízení. Po úspěšném přihlášení do aplikace je tak přes API operačního systému zařízení vyžádán kontaktní list. Kontaktní seznam je vrácen v JSON syntaxi. Formát získaného kontaktního seznamu je uveden v příloze B, v sekci B.2.



Obrázek 5.11: Sekvenční diagram procesu přihlášení uživatele do aplikace.

Aplikace pro své účely zjednoduší kontaktní seznam na informace, které skutečně potřebuje a doplní o indikátor validnosti. Výsledná kolekce, se kterou bude aplikace pracovat, bude mít tedy následující strukturu:

```
[{   "id": "123",
     "displayName": "",
     "phoneNumbers": [
       {
         "id": "1711",
        "value": "972+54+7777777",
       }
     ],
     "photos": null,
      "valid": false
  }]
```



Obrázek 5.12: Sekvenční diagram procesu získání detailu uživatelského účtu.

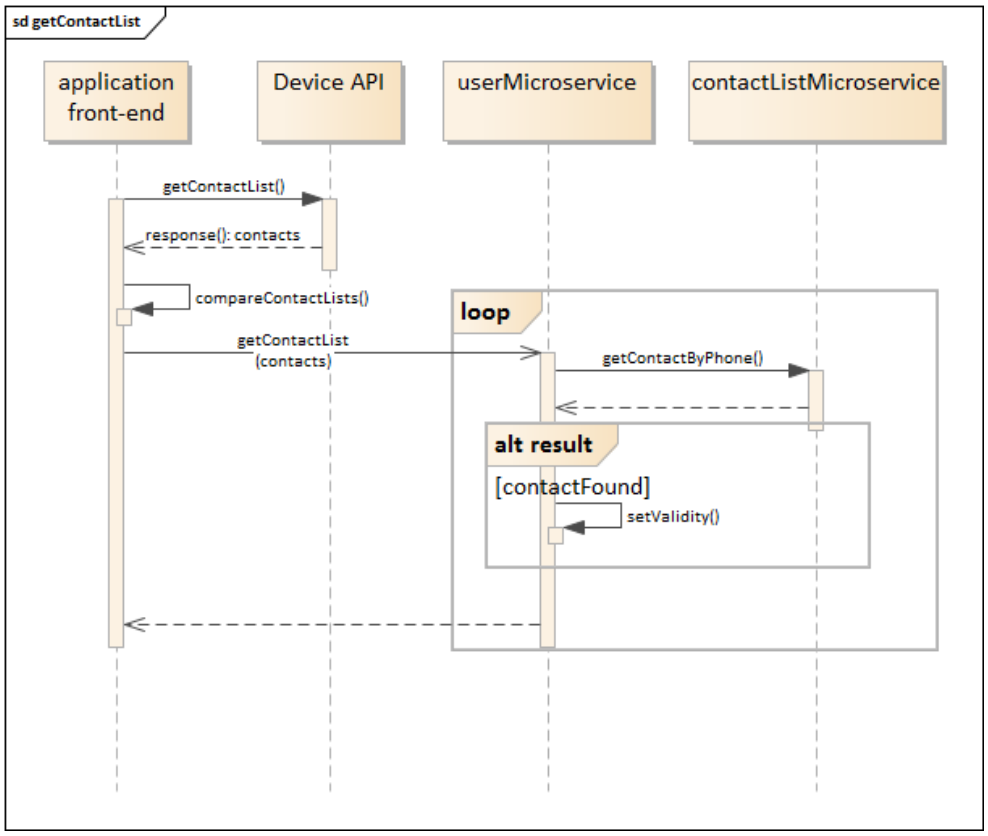
Takovýto kontaktní list aplikace porovná proti kontaktnímu listu, který je již uložen v paměti zařízení od předchozího běhu aplikace. Výsledkem porovnání je kolekce objektů, které byly v kontaktním seznamu zařízení od posledního běhu aplikace změněny. Pokud se jedná o první přihlášení do aplikace, pak výsledkem porovnání bude celý kontaktní seznam uživatele.

Následně je zavolána mikroslužba userMicroservice, které je předána kolekce upravených kontaktů. Pro každý z obdržených kontaktů je následně volána mikroslužba contactListMicroservice, která zjišťuje, zda je daný telefonní kontakt uložen v aplikaci. Úspěšně nalezeným kontaktům je nastaven indikátor valid na hodnotu true, aby aplikace poznala, ke kterým uživatelům lze vytvářet transakce. Reference na každý úspěšně nalezený kontakt je také ukládána do kolekce contactList dokumentu userAccount. Díky tomu je aplikace schopná udržovat kompletní telefonní seznam každého uživatele a reagovat tak na případné změny.

Kontaktní seznam, doplněný o nastavený identifikátor valid, je navrácen aplikaci, kde je uložen do paměti zařízení.

5.8.2.3 Inicializace aplikace

Jakmile jsou obě výše popsané operace dokončeny, uživateli je zobrazen dashboard aplikace, na kterém jsou zobrazené informace o bankovním účtu a probíhající transakce. V tuto chvíli je tak možné využívat veškerou funkčnost apli-



Obrázek 5.13: Sekvenční diagram procesu získání kontaktního seznamu zařízení.

kace, která je popsána v následujících procesech. Proces získání kontaktního seznamu je zachycen na diagramu 5.13.

5.8.3 P03 - Zadání platby

Proces zadání platby je pro aplikaci pro provádění Peer-to-peer plateb klíčový. Aplikace musí umožnit uživateli v rámci zadávání platby zvolit:

- Kontakt, kterému si uživatel přeje zaslat peníze.
- Částku, kterou si uživatel přeje zaslat.
- Zprávu, kterou si uživatel přeje připojit k Platbě.

Při zadání částky probíhá kontrola, zda zadaná částka nepřesahuje stav účtu. Před dokončením platby musí uživatel platbu autorizovat pomocí svého PINu. Pokud je PIN správně zadaný, je zavolána mikroslužba paymentMicroservice

s požadavkem na vytvoření nové Platby. Tělo volání operace má následující obsah:

```
{
        "amount":21,
        "recieverDetail": {
                "phone": "800 154 156",
        },
        "message": "Na chleb"
}
```

Operace získá pomocí mikroslužby contactListMicroservice detail příjemce, včetně čísla bankovního účtu. Na základě těchto dat operace sestaví požadavek CreateTransaction, který je odeslán na API rozhraní daného bankovního poskytovatele. Poté je vytvořen nový dokument do kolekce Payment, který reprezentuje danou platbu.

Následně je nutné notifikovat oba účastníky transakce o nově vytvořené platbě. Toto bude zajištěno pomocí socket.io. V rámci notifikace je oběma účastníkům zaslán detail transakce, který aplikace uloží do příslušné kolekce plateb, aktualizují se zobrazená data a stav účtu. Proces zadání platby je zachycen na diagramu 5.14.

5.8.4 P04 - Zadání Žádosti o platbu

Proces zadání Žádosti o platbu je v mnohém velice podobný procesu zadání Platby. V rámci zadávání aplikace umožní uživateli vyplnit:

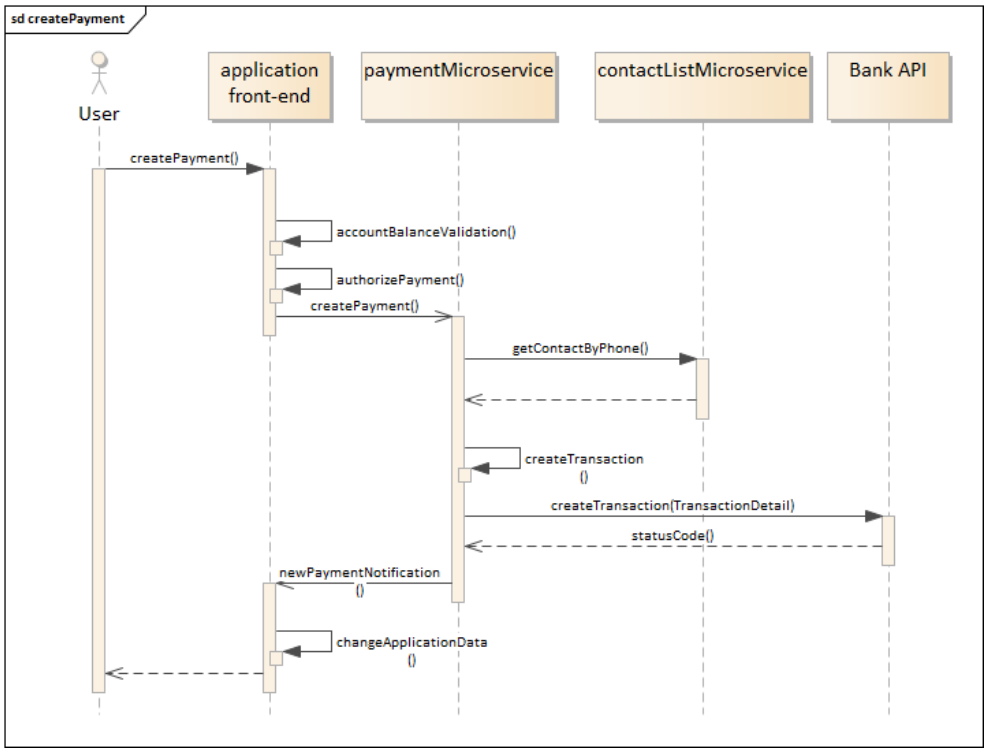
1. Kontakt, na který si přeje zaslat Žádost o platbu.

2. Částku, o kterou si uživatel přeje zažádat.

3. Zprávu, kterou si uživatel k přeje Žádosti o platbu připojit.

Jelikož odeslání Žádosti o platbu nevede přímo k vytvoření nové bankovní transakce, není po uživateli vyžadováno vyplnění PINu. Požadavek na vytvoření nové Žádosti je směrován na mikroslužbu requestMicroservice. Ta pomocí obdrženého telefonního čísla a mikroslužby contactListMicroservice získá detail příjemce a vytvoří nový dokument do kolekce Request, který reprezentuje zadanou Žádost o platbu.

Následně je nutné informovat oba účastníky transakce o nové odchozí, respektive příchozí Žádosti o platbu. Tato notifikace je zajištěna pomocí socket.io. Oba účastníci tak obdrží zprávu o nové Žádosti včetně jejího detailu. Aplikace upraví příslušné kolekce Žádostí a aktualizuje zobrazená data. Diagram procesu zadání připomínky je zachycen na diagramu 5.15.



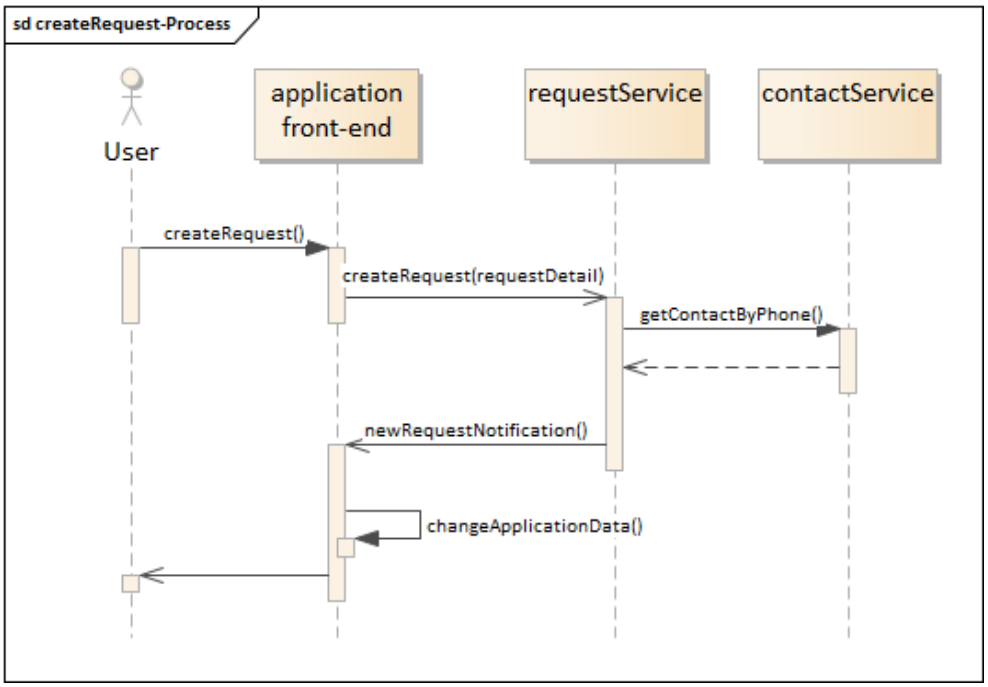
Obrázek 5.14: Sekvenční diagram procesu zadání nové Platby.

5.8.5 P05 - Dokončení Žádosti o platbu

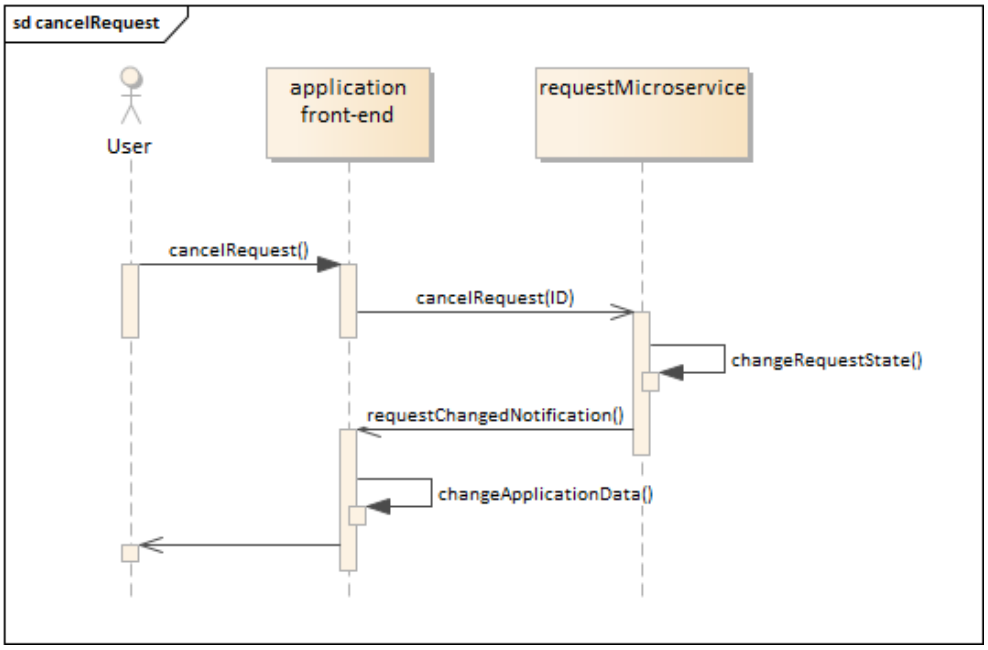
Proces dokončení Žádosti o platbu navazuje na proces Zadání žádosti o platbu. Žádost o platbu může být vyřešena několika způsoby:

- Zrušení ze strany odesílatele.
- Odmítnutí ze strany příjemce.
- Přijetí ze strany příjemce.

Zrušit Žádost o platbu lze pomocí zobrazení detailu transakce a následně zvolit její zrušení. Tímto je odeslán požadavek na mikroslužbu requestMicroservice s ID Žádosti, která má být zrušena. Mikroslužba nalezne vybranou transakci a změní její stav na Zrušeno. Následně je nutné zpravit oba účastníky transakce o jejím zrušení. Tato notifikace společně s detailem zrušené Žádosti o platbu bude odeslaná pomocí socket.io. Aplikace následně na základě obdržených dat upraví kolekci Žádostí a aktualizuje zobrazená data. Tento scénář je zachycen na diagramu 5.16.



Obrázek 5.15: Sekvenční diagram procesu zadání nové Žádosti o platbu.



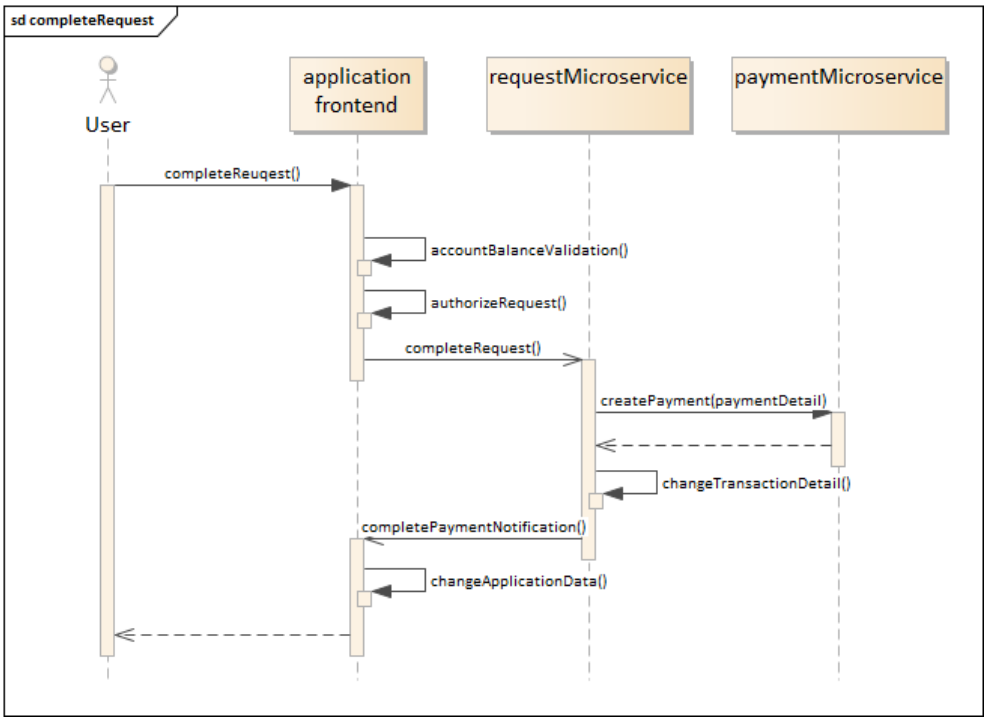
Obrázek 5.16: Sekvenční diagram procesu zrušení odeslané Žádosti o platbu.

Druhým scénářem je odmítnutí Žádosti o platbu ze strany příjemce. Tento scénář je velmi obdobný předchozímu scénáři zrušení Žádosti – pouze je inicializován ze strany příjemce. Aplikace provede volání mikroslužby requestMicroservice, která změní stav dané Žádosti na Odmítnuto. Následně jsou oba účastníci transakce o provedené změně notifikováni prostřednictvím socket.io.

Posledním scénářem je přijetí Žádosti o platbu příjemcem. Před akceptací provádí aplikace kontrolu, zda je na účtu uživatele dostatečný zůstatek. Jelikož na základě přijetí dojde k vytvoření nového bankovního převodu, aplikace musí vyžadovat uživatelovu autorizaci pomocí PINu. V případě úspěšně zadaného PINu je odeslán požadavek na dokončení transakce mikroslužbě requestMicroservice.

Tato mikroslužba vytvoří požadavek na založení nové Platby, který je směrován na mikroslužbu paymentMicroservice, která zadá na bankovní rozhraní dané banky novou platbu. Reference na tuto platbu je následně uložena do dokumentu původní Žádosti, aby ji bylo možné dohledat. Operace dále změní stav Žádosti o platbu na stav Provedeno.

Jakmile je Žádost o platbu dokončena, oba účastníci transakce jsou notifikováni o dokončené transakci pomocí socket.io. Aplikace upraví příslušné kolekce Žádostí a aktualizuje zobrazená data. Diagram procesu zadání Žádosti o platbu je zachycen na diagramu 5.17.



Obrázek 5.17: Sekvenční diagram procesu dokončení Žádosti o platbu.

Kapitola 6

Implementace

V části Návrh jsme provedli podrobnou analýzu aplikace pro provádění Peer-to-Peer plateb. Součástí praktické části diplomové práce je kromě vytvoření tohoto návrhu také implementace prototypu mobilní aplikace, která bude požadovanou funkcionalitu demonstrovat. Cílem implementace tedy není vytvoření aplikace, kterou by bylo možné nasadit do produkčního prostředí, ale vytvoření takové aplikace, kterou bude možné spustit na cílovém zařízení a která bude podporovat definovanou funkcionalitu.

V rámci implementace práce vychází z provedeného návrhu, který je v určitých bodech pro účely prototypu zjednodušen. Druhým zdrojem, ze kterého implementace prototypu částečně vychází, je UX návrh zpracovaný Michalem Kovářem [12].

V úvodu kapitoly jsou popsány nástroje, které byly v rámci implementace prototypu využity. Následně je v kapitole uvedena upravená architektura využitá pro potřebu prototypu aplikace a použitý návrhový vzor. Kapitola obsahuje také popis implementace vybraných procesů – není však cílem opakovat obsah kapitoly Návrh. Při popisu implementace procesů je proto kladen důraz pouze na odchylky a problematická místa, na která jsme rámci implementace narazili.

6.1 Projektové řízení implementace prototypu

Projekt, v rámci kterého tato diplomová práce vzniká, má agilní charakter. Nejedná se tedy o implementaci pomocí vodopádové metodiky, ve které by před zahájením byl vytvořený kompletní návrh, který je nutné realizovat. Formování obsahu a rozsahu projektu se naopak neustále v průběhu jeho realizace mění.

S postupem projektu tak přibývají požadavky, které musí prototyp aplikace splňovat, aby na něm bylo možné demonstrovat funkcionalitu výsledné aplikace. Tyto požadavky dohromady tvoří takzvaný Product backlog, neboli

seznam požadavků, které je nutné do prototypu zapracovat. Ke každému produktovému požadavku je navíc odhadována pracnost implementace.

Samotná implementační část projektu je rozdělena do krátkých ucelených etap, nazývaných sprinty. Na úvod každého sprintu jsou ze seznamu Product backlog vybrány nové požadavky, které budou v rámci sprintu implementovány. Požadavky jsou vybírány na základě jejich přínosu a náročnosti takovým způsobem, aby náročnost každého sprintu byla pokaždé stejná, a aby přínos sprintu byl pro projekt co nejvyšší.

Pro podporu tohoto agilního způsobu implementace byl v rámci projektu využit nástroj Visual Studio Team Services, díky kterému je možné jednoduše sledovat postup projektu, přiřazovat jednolivé úkoly a efektivně spravovat Product backlog.

6.2 Použité nástroje

V této sekci práce jsou stručně představeny nástroje, které byly pro implementaci prototypu využity:

6.2.1 Verzování projektu – Git

Pro verzování projektu byl využit open source nástroj Git. Verzovací systém slouží pro uchovávání veškerých změn provedených ve zdrojových kódech implementované aplikace. Verzovací systém tak umožňuje jednoduchou spolupráci více uživatelů a zároveň provádění správy verzí vyráběného systému.

Ačkoliv na implementaci prototypu pracuji sám a možnost spolupráce tak v rámci implementace nevyužiji, správa verzí je na projektu takovéhoto typu velice důležitá. V rámci projektu bylo nezbytné prototyp pravidelně představovat, avšak zároveň prezentace aktuální verze produktu nesměla bránit implementaci nových vlastností. Z tohoto důvodu pro každý sprint byla vytvořena samostatná větev (branch), která nesla jméno sprintu a do které přibývala nová funkčnost. V hlavní větvi (master) naopak zůstává poslední funkční verze, kterou je možné kdykoliv prezentovat. Na konci sprintu, pokud vše bylo implementováno dle plánu, dochází ke spojení (merge) nové větve s hlavní větví.

Konkrétně pro verzování projektu byl využit GitHub, webová služba podporující vývoj softwaru za pomoci verzovacího nástroje Git.

6.2.2 Deployd

Deployd je open source nástroj, který slouží pro tvorbu webových API rozhraní. Tento nástroj v sobě má již implementované mechanismy pro práci s databází, jakými jsou směrování a přístup k databázi. Při implementaci pak stačí pouze vytvořit kolekce dokumentů, které mají být ukládány, a definovat jejich aplikačně závislou logiku.

Deployd používá k ukládání dat dokumentovou databázi MongoDB. Nástroj umožňuje definovat kolekce dokumentů, nad kterými bude probíhat logika aplikace. Nad definovanými dokumenty je následně vystaveno REST API, které umožňuje s kolekcemi provádět CRUD operace. Ke každé operaci lze také definovat vlastní logiku pomocí Javascriptu. Deployd navíc nabízí vlastní funkce, které je možné při definování aplikační logiky použít.

6.2.3 MongoDB

MongoDB je open source dokumentová databáze a zároveň jedna z nejčastěji používaných NoSQL databází. MongoDB funguje na konceptu kolekcí a dokumentů. Databáze je tak fyzické úložiště, ve kterém jsou uloženy jednotlivé kolekce. Každá kolekce je pak množinou dokumentů, které nemají přesně definované schéma. Dokumenty v rámci jedné kolekce tak mohou mít různou strukturu. Dokument je množina dvojic klíč: hodnota, ve které jsou uloženy konkrétní záznamy. Databáze prototypu má stejnou strukturu, jako je definováno v Návrhu v sekci 5.7.1.

6.2.4 Monaca

Monaca je sada nástrojů, které umožňují implementaci mobilních aplikací pomocí Apache Cordova. Z této sady nástrojů jsem pro implementaci využil:

- Monaca Cloud IDE – vývojové prostředí v cloudu, které podporuje vývoj hybridních mobilních aplikací.
- Monaca Debugger – mobilní aplikace, která umožňuje propojení s projektem vyvíjeným pomocí Monaca IDE a jeho následné spuštění na cílovém zařízení.

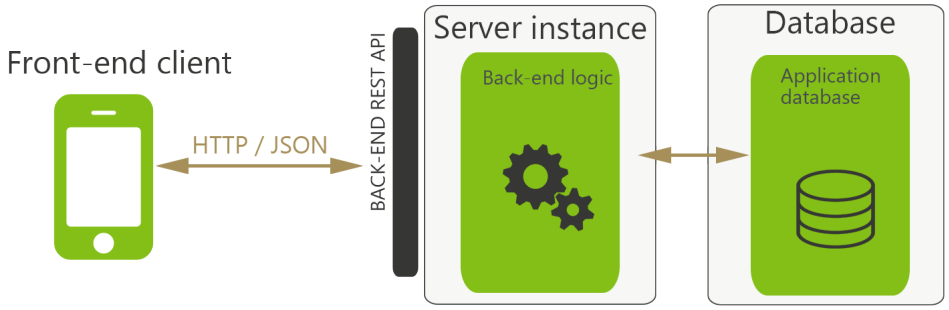
6.2.5 Onsen UI

Onsen UI je framework, který poskytuje frontendové komponenty, které je možné využít pro tvorbu hybridních mobilních aplikací. Díky tomuto frameworku je tedy možné vytvářet hybridní mobilní aplikace, které svým vzhledem není možné odlišit od nativních aplikací. Tyto frontendové komponenty jsou vystaveny na technologiích Angular 1, Angular 2 nebo React.

6.2.6 Heroku

Heroku je cloudová služba typu PaaS[5], která poskytuje možnost jednoduše publikovat aplikaci. Aplikaci je možné na Heroku nasadit za pomoci Gitu. Heroku podporuje Node.js, který je nutný pro fungování nástroje Deployd. Na tomto hostingu tedy bude nasazena serverová část aplikace.

[5]Platform as a service je typ cloudové služby, která poskytuje uživatelům platformu pro vývoj aplikací.



Obrázek 6.1: Zjednodušená architektura prototypu aplikace pro zadávání Peer-to-Peer plateb.

6.2.7 mLab

Posledním použitým nástrojem je mLab. Jedná se o cloudovou platformu typu Database-as-a-Service, která poskytuje hosting pro MongoDB databázi. Na tomto hostingu tak bude nasazena aplikační databáze.

6.3 Architektura prototypu

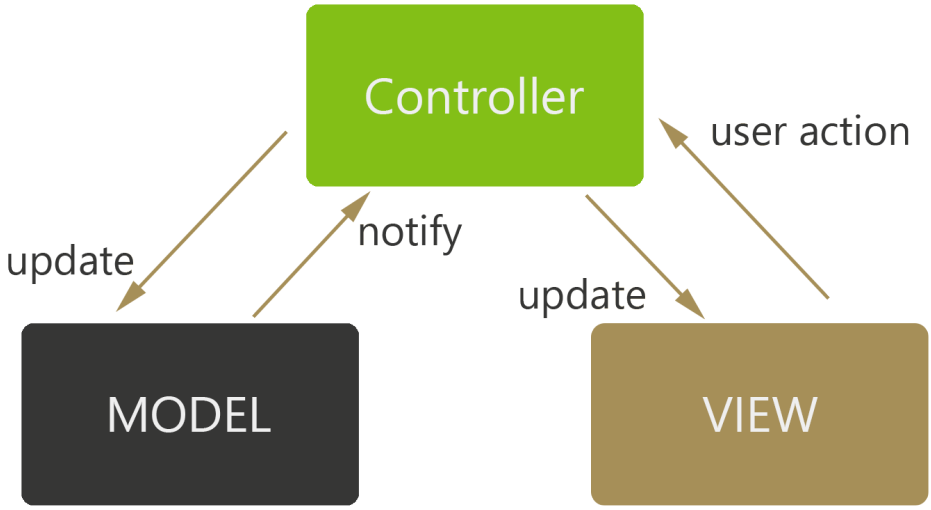
Architektura prototypu se bude lišit od architektury uvedené v části Návrh. Pro účely prototypu bude postačující třívrstvá architektura složená ze:

- Klientská část aplikace, implementovaná pomocí Apache Cordova.
- Serverová část aplikace, implementovaná pomocí nástroje Deployd.
- Dokumentová databáze typu MongoDB.

Na serverové straně bude vystaveno REST API, přes které bude klientská část aplikace komunikovat. Bankovní API bude pro potřeby prototypu zcela vynecháno, neboť v současné chvíli není testovací API k dispozici. Pro demonstraci prototypu však tato vrstva není nutná, neboť pro testování postačí data uložená v databázi. Dále v prototypu nebudeme řešit Load balancing, neboť budeme mít pouze jednu instanci aplikačního serveru. Zjednodušená architektura pro potřeby prototypu je zobrazena na obrázku 6.1.

6.4 Návrhový vzor MVC

Front-end aplikace využívá návrhový vzor Model-View-Controller. Tento návrhový vzor rozděluje aplikační logiku do tří oddělených částí[29]:



Obrázek 6.2: Vztah mezi jednotlivými částmi MVC návrhového vzoru.

Model — Jedná se o část aplikace, kde jsou uloženy veškeré objekty v aplikaci. Model nemá přehled o částech View ani Controller. Když se model změní, obvykle o tom notifikuje své observery.

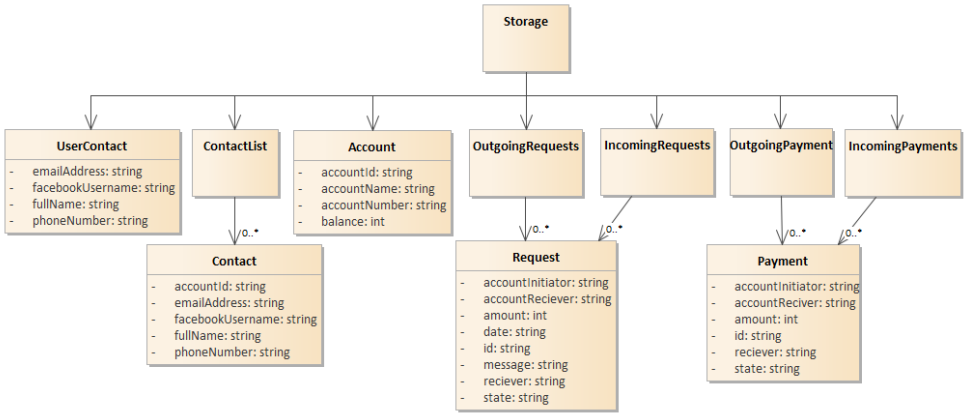
View — Ve view je uložena část aplikace, která se uživateli zobrazuje a se kterou uživatel interaguje. Jedná se o část aplikace, která má přístup k DOMu.

Controller — Poslední částí aplikace je controller, který implementuje rozhodovací logiku a slouží jako prostředník mezi částmi View a Model.

Využití tohoto návrhového vzoru přináší mimo jiné následující výhody:

1. znovupoužitelnost kódu,
2. oddělení logiky aplikace od prezentovaného vzhledu,
3. jednodušší udržitelnost.

Volba tohoto návrhového vzoru je důležitá především z pohledu agilního pojetí implementace. Jelikož na začátku projektu není jasné celé zadání a rozsah aplikace, oddělením logiky od prezentační vrstvy jsme schopni implementovat nové vlastnosti s menším dopadem do zbytku aplikace. Vztah mezi jednotlivými částmi je zachycen na diagramu 6.2.



Obrázek 6.3: Zobrazení třídy storage, sloužící pro uložení dat v aplikaci.

6.4.1 Model

Model je v aplikaci reprezentován pomocí třídy Storage. Třída Storage se stará o ukládání dat získaných z aplikačního serveru. Při inicializaci aplikace dojde k získání detailu uživatelského účtu včetně příchozích a odchozích transakcí a kontaktního listu zařízení. Struktura dat uchovaných v zařízení je zachycena na diagramu 6.3.

userContact	Objekt uchovává kontaktní detail uživatelského účtu.
contactList	Kolekce obsahující seznam kontaktů v daném zařízení.
account	Objekt obsahující detail bankovního účtu uživatele.
outgoingRequests	Kolekce obsahující seznam odchozích Žádostí o platbu.
incomingRequests	Kolekce obsahující seznam příchozích Žádostí o platbu.
outgoingPayments	Kolekce obsahující seznam odchozích Plateb.
incomingPayments	Kolekce obsahující seznam příchozích Plateb.

6.4.2 Controller

Controller má v rámci aplikace na starosti logiku. Z důvodu lepší udržitelnosti kódu je controller rozdělen na několik částí podle funkcionality, kterou má na starosti:

loginController	Má na starosti logiku spojenou s přihlašováním a registrováním do aplikace.

navigationController	Má na starosti navigaci mezi jednotlivými stránkami aplikace.
pageController	Řídí plnění obsahu jednotlivých stránek.
communicationController	Pomocí tohoto controlleru probíhá veškerá komunikace se serverovou částí aplikace.
userController	Tento controller má na starost úpravy uživatelských údajů a rekace na jejich změny.
transactionController	Řídí veškerou logiku spojenou s zadáváním nových transakcí.

6.4.3 View

View aplikace definuje sémantickou strukturu jednotlivých stránek aplikace a vzhled jednotlivých prvků. Tuto vrstvu lze rozdělit na tři základní části:

HTML Soubory, které definují sémantickou strukturu jednotlivých stránek aplikace.

CSS Kaskádní styly, které definují vzhled aplikace.

JS Javascriptový kód, který definuje chování vlastních a obsah front-end komponent, které jsou v aplikaci využity.

6.5 Navigace v rámci aplikace

Mobilní aplikace se skládá z více stránek, přechody mezi nimi jsou definované v UX návrhu vytvořeném Michalem Kovářem[25]. Pro umožnění přechodu mezi jednotlivými stránkami nabízí Onsen UI několik navigačních prvků:

ons-tabbar	Navigace na spodu stránky.
ons-sliding-menu	Boční dynamické menu.
ons-navigator	Navigační prvek, který umožňuje skládat stránky do formy zásobníku. Tento prvek následně poskytuje funkce pro manipulaci s obsahem uloženým v daném zásobníku.

Podle UX návrhu aplikace je evidentní, že aplikace si pro navigaci nevystačí pouze s jedním navigačním prvkem, ale bude nutné zvolit vhodnou kombinaci těchto prvků.

První úroveň navigace řeší logiku aplikace před samotným přihlášením do aplikace – tedy přihlašování a registraci do aplikace. Pro tuto navigaci využijeme navigační prvek ons-navigator a řazení prvků do zásobníku.

Druhou úroveň tvoří spodní menu aplikace, které umožňuje uživateli rychlé přechody mezi pěti hlavními stránkami. Tyto stránky jsou:

- hlavní stránka aplikace,
- uživatelský detail,
- kontaktní list zařízení,
- sdílení aplikace,
- více možností.

Přechod mezi těmito stránkami je realizován pomocí ons-tabbar elementu.

Třetí úroveň navigace slouží pro pohyb mezi stránkami, které podporují jednotlivé aplikační procesy. Každý z aplikačních procesů je realizován více stránkami aplikace. Stránka, ze které je proces zahájen, má pro tyto účely vlastní prvek ons-navigator, takže každá stránka má vytvořený vlastní zásobník aktivních stránek, což umožňuje vhodnou navigaci napříč aplikací. Pro potřeby prototypu aplikace obsahuje následující navigační prvky třetí úrovně:

- pageNavigator – navigační prvek pro přechod mezi stránkami, které slouží k vytváření transakcí a zobrazování detailu transakcí.
- contactNavigator – navigační prvek, který slouží pro přechody mezi stránkami při vytváření nových kontaktů.
- userNavigator – navigační prvek používaný pro přechody mezi stránkami při změně uživatelských údajů.
- optionsNavigator – navigační prvek, který slouží pro zobrazování stránek definovaných na stránce Více možností.

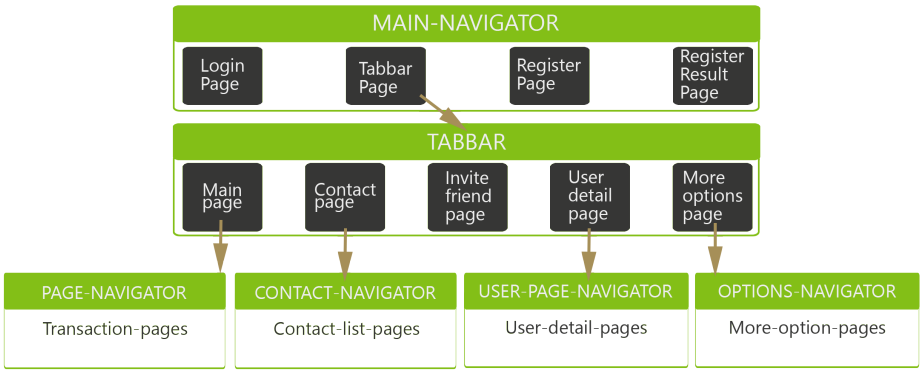
Schéma použitých navigačních prvků je zachycen na diagramu 6.4.

6.6 Realizace jednotlivých procesů

V této sekci je popsána realizace jednotlivých procesů tak, jak jsou implementovány v prototypu. Realizace jednotlivých procesů se v rámci prototypu v některých částech liší od Návrhu, popsaného v minulé kapitole.

6.6.1 Implementace procesu Registrace uživatele

Proces registrace v prototypu je zahájen kliknutím na tlačítko „Registrovat se“ na úvodní stránce aplikace. V tuto chvíli je uživateli nabídnuta nová stránka s formulářem pro vyplnění osobních údajů, kterou není možné dokončit bez vyplnění všech povinných polí. Pro pole telefonní číslo, bankovní účet a PIN je



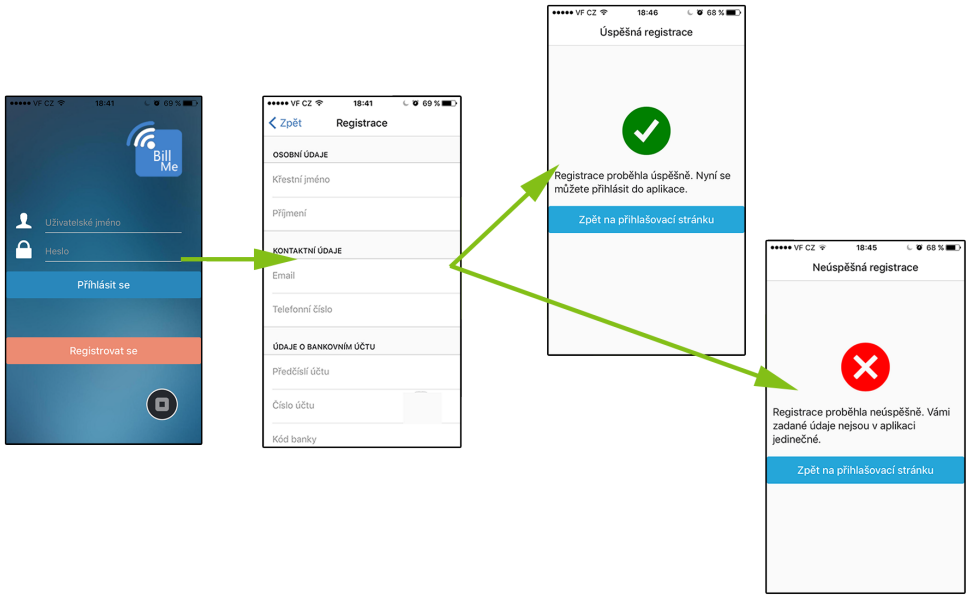
Obrázek 6.4: Navigační vzor použitý pro aplikaci.

nabídnuta pouze numerická klávesnice. Po vyplnění údajů a zvolení možnosti Dokončit registraci dojde ke kontrole, zda jsou všechny údaje vyplněny.

Pokud je vše správně vyplněno, pak dojde ke konverzi telefonního čísla na standardní formát. Tato konverze je nutná, jelikož různé operační systémy ukládají telefonní čísla v různých formátech. Dále je potřeba vzít v potaz oddělení jednotlivých trojčíslí a přítomnost předvolby. Proto jsou všechna čísla konvertována na formát řetězce složeného z devíti číslic. Po dokončení konverze jsou zadané údaje odeslané ve formátu JSON pomocí požadavku POST na REST API serverové části aplikace, na zdroj /user. Tělo HTTP požadavku má následující strukturu:

```
{
        "fullName": "Jan Stransky",
        "bankAccount": {
                "accountNumber": "8489515115",
                "bankCode": "2020",
                "accountName": "Muj uctik"
        },
        "username": "487774",
        "password": 123,
        "phoneNumber": 789456123,
        "pin": 123
}
```

Při provolání POST požadavku na kolekci User je na serverové části implementována logika, která se postará o založení nového uživatelského účtu. Pro uživatelskou kolekci Deployd nabízí speciální typ kolekce, který se stará o šifrování hesel a automatickou kontrolu, zda uživatelské jméno je unikátní. Zbývá tedy pouze kontrola unikátnosti telefonního čísla a bankovního účtu,



Obrázek 6.5: Proces registrace uživatele v prototypu aplikace.

která je delegována na kolekci Contacts, která se stará o celkový kontaktní list zařízení.

Pokud jsou veškeré záznamy unikátní, pak jsou vytvořeny dokumenty v kolekcích User a Contacts. Následně je klientské části aplikace vrácena HTTP odpověď se stavovým kódem 200 a aplikace zobrazí stránku s úspěšnou registrací. Pokud kterýkoliv z údajů není unikátní, žádný nový dokument není vytvořen a klientská část aplikace obdrží HTTP odpověď se stavovým kódem 400. V tomto případě je uživatel přesměrován na stránku s neúspěšnou registrací. Tento proces z pohledu aplikace je zachycen na diagramu 6.5.

6.6.2 Implementace procesu Přihlášení do aplikace

Po registraci prototyp aplikace umožňuje přihlášení pomocí zadaného uživatelského jména a hesla. Pro přihlášení uživatel musí vyplnit jméno a heslo, pokud některý z těchto údajů není vyplněn, aplikace zobrazí varovnou hlášku. V případě, kdy jsou veškeré údaje vyplněny, aplikace sestaví HTTP POST požadavek na kolekci /user/login:

```
{
        "username": "kukacdav@gmail.com",
        "password": "tajneHeslo"
}
```

Tím, že se jedná o speciální typ uživatelské kolekce, Deployd má v sobě implementovanou funkcionalitu pro ověření přihlašovacích údajů. V případě nalezení uživatelského účtu vrací HTTP požadavek stavový kód 200 a uživatel je přesměrován na hlavní stránku aplikace a je zahájena inicializace aplikace. Tuto fázi, jak je popsáno v kapitole Návrh, lze rozdělit do dvou částí:

- získání detailu uživatelského účtu,
- zpracování kontaktního listu zařízení.

Pro získání detailu uživatelského účtu je volán HTTP požadavek GET na kolekci User s vyplněným ID daného uživatelského účtu. Na této kolekci na operaci GET je vystavena logika, která postupně provolá kolekce Payment a Request s účelem získat příchozí a odchozí Platby a Žádosti o platby. Obsah kolekcí je setříděn podle data vytvoření a velikost každé z kolekcí je omezena na 15 záznamů.

Pro zpracování kontaktního listu je nutné nejprve získat kontaktní list zařízení. Toto je dosaženo pomocí cordova-plugin-contact, který umožňuje pracovat s kontaktním listem zařízení. V této fázi je nutné vzít v potaz odchylky jednotlivých operačních systémů. Hlavní problém, se kterým jsem se v rámci implementace prototypu potýkal, je uložení jména kontaktu v zařízení, který se liší mezi operačním systémem Android a iOS. Výsledná funkce pro získání jména telefonního kontaktu je uvedena v příloze B, v sekci B.3.

Takto sestavený kontaktní list je následně odeslán na serverovou část na kolekci User v těle HTTP požadavku PUT. Následně probíhá kontrola, které kontakty jsou v databázi aplikace uloženy, jak je popsáno v kapitole Návrh.

Celý tento proces je z pohledu diagramu obrazovek aplikace zachycen na diagramu 6.6.

6.6.3 Implementace procesu Zadávání transakcí

Při zadávání transakcí není obecně rozlišováno, zda se jedná o Platbu či Žádost o platbu – z pohledu aplikace proces prochází stejnými stránkami. Jejich obsah je plněn na základě hodnoty atributu transactionType, který nese informaci, o jaký typ transakce se jedná.

Proces zadávání transakce prochází postupně přes jednotlivé stránky, na kterých je skládán objekt, reprezentující transakci. Proces nejprve přechází na stránku contact-list-page, na které je uživateli nabídnut seznam kontaktů, které mají v aplikaci uživatelský účet. Následně je uživatel přesměrován na stránku set-amount-page, na které je možné definovat hodnotu transakce. Zároveň je zde implementována logika, která kontroluje, že zadaná částka je nenulová, nezáporná a nižší než zůstatek na bankovním účtu.

Následující stránka zobrazuje zadané údaje a umožňuje k transakci připojit zprávu. Na závěr je nutné vyplnit PIN pro autorizaci transakce. Pokud je

Obrázek 6.6: Proces přihlášení uživatele v prototypu aplikace.

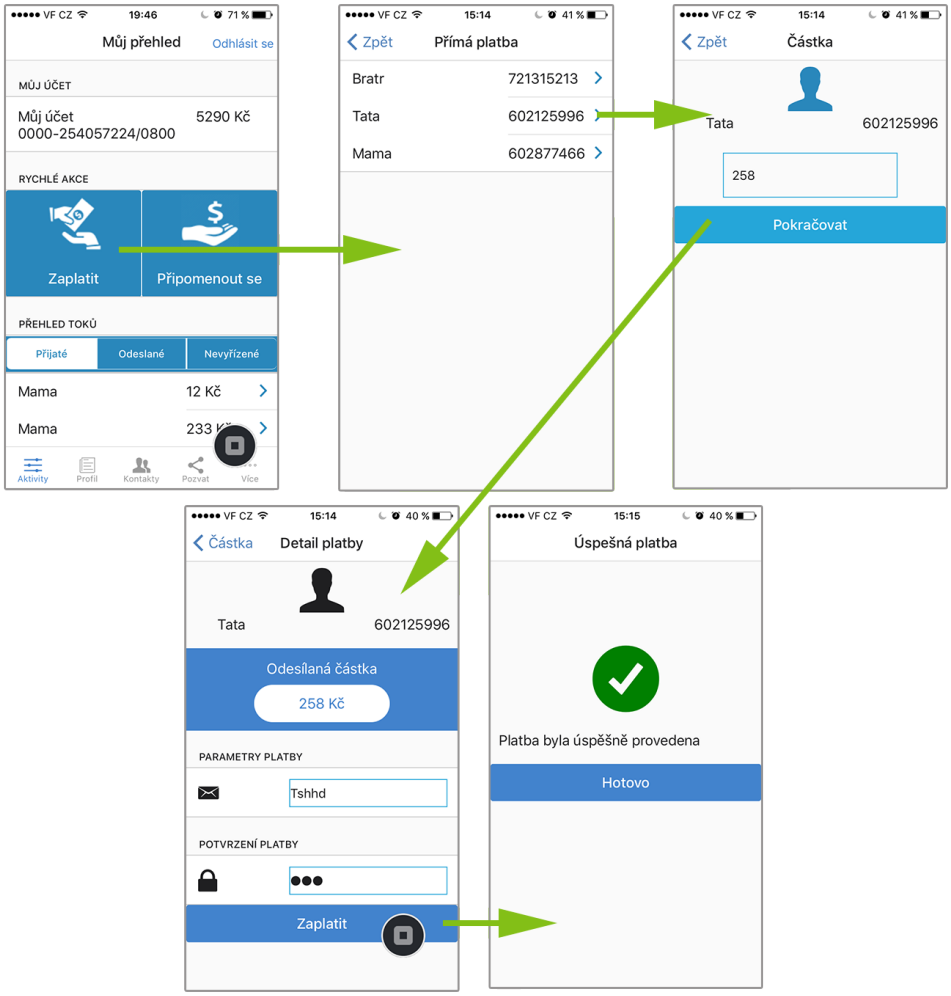
PIN špatně vyplněn, uživatel je na tuto skutečnost upozorněn textovým popisem. V případě správného vyplnění všech údajů je na serverovou část aplikace odeslán HTTP POST požadavek na kolekci Payment, respektive Request.

Na serverové části je vytvořen nový dokument do příslušné kolekce a následně jsou pomocí socket.io o tomto faktu informováni příjemce i odesílatel, kterým se zobrazí nová transakce na hlavní stránce aplikace. Na front-endu aplikace je mezitím uživatel přesměrován na stránku success-submit-page, kde je upozorněn na úspěšně zadanou novou transakci. Celý proces vytvoření nové transakce je z pohledu aplikace zachycen na diagramu 6.7.

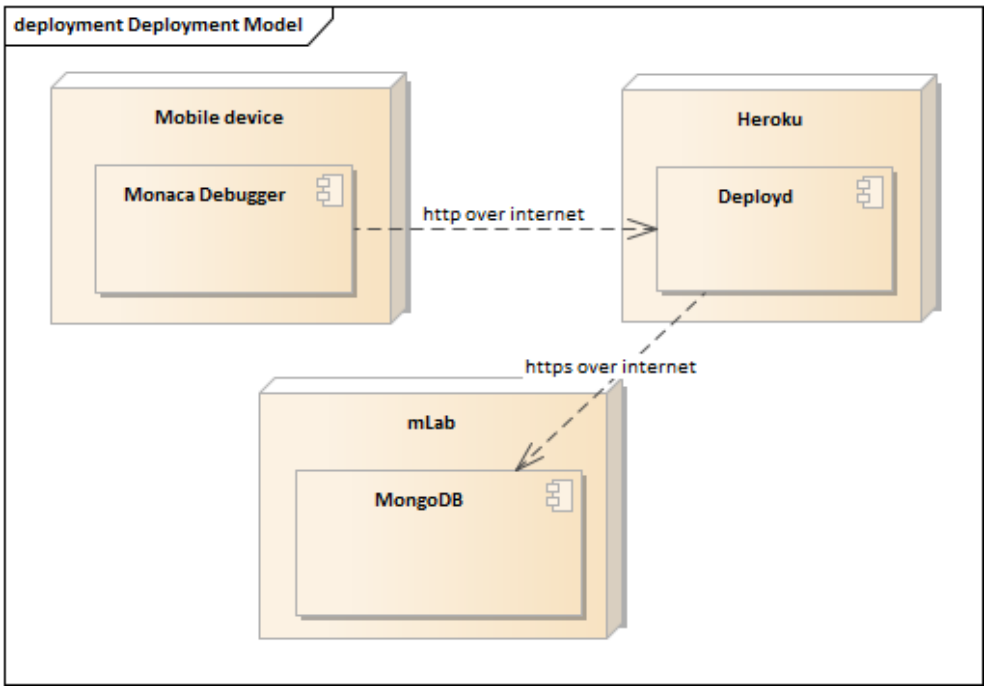
6.7 Nasazení prototypu

Pro možnost testování prototypu na cílovém zařízení je nutné zdrojové kódy klientské a serverové části nasadit na veřejně dostupné prostředí. Pro testování klientské části aplikace je v rámci této práce využita aplikace Monaca Debugger. Jedná se o aplikaci, která umožňuje synchronizaci zdrojových kódů se vzdáleným úložištěm a následně jsou zdrojové kódy aplikace spuštěny ve WebView této aplikace. Tento nástroj tak umožňuje jednoduché spuštění hybridní aplikace bez nutnosti vystavování aplikace v příslušném obchodě.

Druhá část aplikace, kterou je nutné nasadit na veřejně dostupné prostředí,



Obrázek 6.7: Proces vytvoření nové transakce v prototypu aplikace.



Obrázek 6.8: Diagram nasazení funkčního prototypu.

je serverová část. Zde je využit hosting Heroku, který umožňuje nahrání a build Deployd aplikace. Tento hosting však bohužel nepodporuje potřebnou MongoDB. Pro nasazení databáze je proto využit hosting mLab, který umožňuje nasazení MongoDB a na který serverová část aplikace směřuje své požadavky. Nasazení aplikace je zachyceno na diagramu 6.8.

6.8 Shrnutí prototypu

Prototyp aplikace, vyvinutý v rámci této diplomové práce, si klade za cíl demonstraci zamýšlené funkcionality aplikace pro provádění Peer-to-Peer plateb. Aby bylo tohoto cíle dosaženo, podporuje výsledná implementace následující akce:

1. Registrace do aplikace.

2. Přihlašování / odhlašování uživatele.

3. Vytváření nových Plateb - vytvoření nové Platby je možné pouze k jinému uživateli aplikace, jehož telefonní číslo je uloženo v paměti zařízení. Příjemce je o nové platbě ihned informován pomocí socket.io.

4. Vytváření nových Žádostí o platbu - vytvoření nové Žádosti o platbu je možné pouze k jinému uživateli aplikace, jehož telefonní číslo je uloženo v paměti zařízení. Příjemce je o nové Žádosti o platbu informován pomocí socket.io.

5. Zobrazení transakcí - transakce jsou filtrovány a zobrazovány pod jménem kontaktu v kontaktním listu zařízení.

6. Vyřízení obdržených Žádostí o platbu - jejich přijetí a následné zaplacení nebo odmítnutí. O změně stavu Žádosti o platbu je druhá strana ihned notifikována pomocí socket.io.

7. Zrušení odeslaných Žádostí o platbu - o jejich zrušení je ihned notifikována druhá strana pomocí socket.io.

8. Změna základních osobních údajů - v rámci prototypu pouze neunikátních osobních údajů, které osobu na datové úrovni neidentifikují.

9. Změna uživatelského hesla.

10. Změna PINu.

11. Vytvoření nového kontaktu - nový kontakt je vytvořen pomocí cordova-contacts-plugin přímo do paměti mobilního zařízení.

12. Zobrazení návodů pro práci s aplikací

13. Zobrazení právního rámce aplikace - jedná se pouze o demonstrativní, nikoliv kompletní, podobu právních ustanovení.

14. Zobrazení finančního přehledu - jednoduchý finanční přehled příjmů a výdajů za dané období.

Výsledný prototyp však také obsahuje několik nedostatků, které z určitých důvodů nebylo možné zcela odstranit:

1. Optimalizace pouze pro operační systém iOS - ačkoliv se jedná o hybridní aplikaci, která by měla fungovat napříč jednotlivými operačními systémy, každý operační systém má své odchylky. Z tohoto důvodu není možné očekávat, že výsledná aplikace bude na všech platformách vypadat a chovat se stejně, bez důsledného odladění. Prototyp aplikace byl testován na operačních systémech iOS a Android, avšak z důvodu náročnosti proběhla optimalizace vizuální stránky pouze pro platformu iOS.

2. Orientace obrazovky - Apache Cordova nabízí možnost ve své konfiguraci nastavit, zda má aplikace umožňovat orientování obrazovky v závislosti na poloze zařízení. Tato vlastnost však nefunguje u Apache Cordova

zcela správně a i přes striktní zablokování otáčení obrazovky k němu stále dochází. Tento problém je řešitelný pomocí pluginu cordova-plugin-screen-orientation. Použité IDE však import tohoto pluginu v neplacené verzi neumožňoval, proto ve výsledném prototypu tato chyba přetrvává.

3. Bezpečnost aplikace - v rámci implementace prototypu nebyl kladen důraz na zabezpečení aplikace. Stav zabezpečení aplikace je proto nedostatečný, cílem prototypu však nebylo věnovat se této doméně.

Kapitola 7

Harmonogram a projektová část implementace

V předchozích kapitolách jsme postupně představili návrh a koncept mobilní aplikace pro provádění Peer-to-Peer plateb, následně jsme na implementovaném prototypu demonstrovali funkcionalitu aplikace. Tato kapitola rozebírá problematiku implementace aplikace z projektového pohledu. Za tímto účelem je zde rozpracován postupný harmonogram implementace včetně nákladů na implementaci a uvedení na trh. Dále jsou zde představeny metriky pro měření úspěšnosti projektu pro možnost jeho zhodnocení.

7.1 Harmonogram vývoje

Při stanovení harmonogramu vývoje aplikace je nutné pracovat s termínem 13. 01. 2018, do kterého musí členské státy EU implementovat směrnici PSD2 do své národní legislativy. Lze tedy očekávat, že od této doby budou zpřístupněny systémové API jednotlivých bank a bude možné aplikaci využívat. Úspěch aplikace závisí na včasném spuštění v produkčním prostředí, proto se jedná o termín, ke kterému aplikace musí být otestovaná a bezpečně spuštěna.

Aktivity, které budou nutné v rámci vývoje vykonat, lze rozdělit do následujících domén:

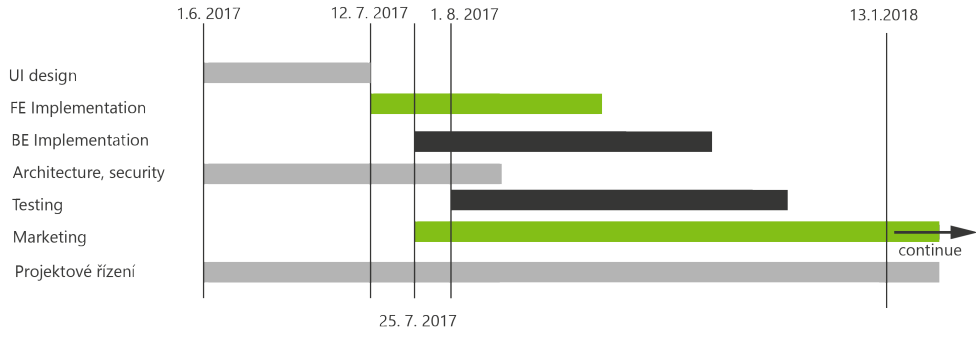
Název domény	Aktivity
Návrh UI	Lze vyjít ze zde implementovaného prototypu a návrhu Michala Kováře[25]. Doplnění grafiky. Opakované kvalitativní testování akutalizovaného prototypu.
Návrh architektury	Lze vyjít z návrhu v této práci. Úprava návrhu na základě vydaného technického standardu API.

Zabezpečení aplikace	Definování bezpečnostních mechanizmů aplikace. Splnění bezpečnostních norem definovaných standardem pro získání certifikace.
Implementace serverové části	Vystavění REST API. Nasazení dokumentové databáze. Implementace mikroslužeb. Dokumentace serverové části aplikace.
Implementace klientské části	Vývoj hybridní aplikace dle testovaného prototypu. Optimalizace aplikace pro jednotlivé platformy.
Testování aplikace	Penetrační testy. Regresní testy. Uživatelské testování.
Marketing	Kreativa. Reklamní kampaň zaměřená na cílovou skupinu. Analýza, optimalizace kampaně.

Tabulka 7.1: Rozdělení nutných aktivit pro vývoj produktu.

Pro vytvoření harmonogramu projektu bude nutné nejprve ohodnotit pracnost jednotlivých aktivit. Toto ohodnocení pracnosti je v rámci práce provedeno kombinací náročnosti implementace prototypu a empirickou zkušeností softwarového architekta s podobnými projekty. Kompletní harmonogram a náklady spojené s implementací projektu jsou podrobněji popsány v příloze C.

Na základě výsledků odhadů pracnosti a konzultace časové náročnosti a návaznosti jednotilvých aktivit jsme sestavili Ganttův diagram, který představuje předběžný harmonogram prací na projektu implementace produkční verze aplikace. Výsledný diagram je znázorněn na obrázku 7.1.



Obrázek 7.1: Harmonogram projektu formou Ganttova diagramu.

7.2 Uvedení produktu na trh

Pro úspěšné uvedení produktu na trh je klíčový propracovaný marketing. S nástupem Open Bankingu lze očekávat nárůst počtu Fintechů, které budou svou činnost zaměřovat na podobnou doménu jako navrhovaná aplikace. Z tohoto důvodu je klíčové na trh proniknout co nejdříve a zaujmout dominantní postavení coby poskytovatel služby Peer-to-Peer platba. Aby toto bylo alespoň trochu možné, je nutné vypracovat kvalitní marketingovou kampaň zaměřenou na cílovou skupinu. Tato kampaň musí být zahájena v dostatečném předstihu před spuštěním aplikace a postupně by měla gradovat a pokračovat i po spuštění aplikace. S ohledem na harmonogram vývoje by kampaň měla být spuštěna nejpozději koncem července 2017.

Pro oslovení cílové skupiny, mladí lidé a lidé se zájmem o nové technologie, bude hlavním kanálem reklama na sociálních sítích a webová prezentace, demonstrující využití produktu. Důraz by zde měl být kladen na zachycení jednoduchosti placení pomocí této aplikace, který může být postaven do kontrastu nesmyslné složitosti stejného procesu bez využití aplikace. Reklama na sociálních sítích i webová prezentace musí být navržena přitažlivou formou pro cílovou skupinu,

Hlavním cílem marketingové kampaně před vydáním produktu je navázat kontakt s potenciálními klienty. Tento kontakt může být vytvořen pomocí tzv. oblíbení stránky na sociální síti nebo přihlášením svého emailového účtu k odběru novinek. Cílem je dostat produkt do povědomí a zároveň vytvořit klientskou základnu potenciálních uživatelů. V této fázi je klíčový obsah a způsob komunikace s možnými zákazníky, aby nedošlo k jejich odrazení častým zasíláním obsahu a zároveň zůstal vztah dostatečně živý. Tohoto musí být dosaženo vhodně zvolenou formou a obsahem marketingové kampaně.

Druhá fáze marketingu nastává společně s nasazením aplikace do produkčního prostředí. Cílem této fáze by měla být konverze dříve vytvořené klientské základny na skutečné zákazníky, kteří aplikaci běžně využívají pro placení. Současně by se marketing měl soustředit na získání nových uživatelů aplikace. Marketingová kampaň by měla být pravidelně analyzována, do jaké míry plní kladená očekávání, a příslušně modifikována. Odhad nákladů na uvedení aplikace do provozu ze strany marketingu jsou uvedeny v příloze C.

7.3 Metriky měření úspěšnosti projektu

Metriky měření úspěšnosti projektu nám určují, kdy označíme projekt jako úspěšný. Tyto podmínky je nutné určit před samotnou implementací projektu, abychom následně mohli porovnávat aktuální stav s původně stanovenými očekáváními. Při stanovování metrik úspěšnosti není nutné pokrýt každý aspekt uvedení a provozu aplikace, stačí se soustředit pouze na vlastnosti, které jsou pro aplikaci klíčové. Zvolené metriky úspěšnosti jsou uvedené v tabulce 7.2.

Název metriky	Slovní popis
Počet registrovaných uživatelů	Pro použitý koncept je esenciální počet registrovaných uživatelů, neboť jednoduchost aplikace se odvíjí od schopnosti překladu telefonního čísla na číslo bankovního účtu. Čím více uživatelů je tedy registrovaných do aplikace, tím více transakcí může probíhat jednoduchou formou.
Počet pravidelně platících uživatelů	Uživatelé, kteří pravidelně provádí pomocí aplikace transakce, jsou klíčoví pro úspěch produktu. Kromě samotného využívání služby tito uživatelé svou činností rozšiřují aplikaci mezi nové potenciální klienty. Za pravidelně platící uživatele budeme považovat uživatele, kteří aplikací využijí alespoň třikrát měsíčně.
Počet závažných incidentů	Aplikace cílí na doménu spojenou s bankovnictvím a financemi lidí. Jedná se tak o velice citlivou doménu a jakékoliv incidenty mohou mít výrazné dopady na důvěru a následné využívání služby klienty. Počet veškerých incidentů je proto nutné minimalizovat. Jako závažný označíme takový incident, který ovlivňuje korektní provedení finančí transakce.
Dostupnost služby v %	Hlavním mottem aplikace je co nejjednodušší způsob placení. Pokud klient nebude schopen tuto službu využít z důvodu jejího výpadku, v budoucnu se na ni nebude příliš spoléhat.

Tabulka 7.2: Metriky úspěšnosti projektu.

Následně k jednotlivým metrikám stanovíme odhad, který bude značit hranici, od které budeme dosažené hodnoty považovat za úspěch. Metriky byly stanoveny na základě empirického odhadu a diskuze, výsledné hodnoty jsou uvedeny v tabulce 7.3.

Název metriky	Hranice úspěšnosti
Počet registrovaných uživatelů	200 000 po prvním roce.
Počet pravidelně platících uživatelů	10% z registrovaných uživatelů po prvním roce.
Počet závažných incidentů	méně než 3 měsíčně.
Počet registrovaných uživatelů	v průměru prvního roku vyšší než 99%.

Tabulka 7.3: Metriky úspěšnosti projektu.

Závěr

V rámci této diplomové práce jsme představili přínos evropské směrnice PSD2 směrem k rozvoji Open Bankingu, zejména díky zpřístupnění bankovních API třetím stranám. Na základě této myšlenky jsme navrhli koncept aplikace pro provádění Peer-to-Peer plateb, který umožňuje zasílání peněz pouze na základě znalosti telefonního kontaktu protistrany. Pro úspěšnou realizaci tohoto konceptu je nezbytné vytvoření široké databáze uživatelů, která bude umožňovat překlad telefonního čísla na číslo bankovního účtu. Pro toto využití práce navrhuje využití konceptu dokumentové databáze, kterou lze díky cloudovému nasazení efektivně horizontálně škálovat.

Na úvod systémové specifikace jsme navrhli doménový model, který podrobně popisuje doménu navrhovaného systému a sjednocuje používanou terminologii. Dále práce definuje business model této aplikace, který byl podroben kvantitativnímu průzkumu se zástupci cílové skupiny - mladí lidé a lidé s kladným přístupem k technologiím. Na základě výstupů kvantitativního průzkumu jsme definovali funkční a nefunkční požadavky, které jsou na aplikaci kladené. Tyto požadavky jsme rozpracovali do podoby konkrétních případů použití, které musí výsledná aplikace pokrývat.

Následně jsme se v rámci systémové specifikace zaměřili na technickou stránku návrhu, ve které jsme definovali jednotlivé části architektury systému. Na základě porovnání přístupů SOAP a REST jsme pro komunikaci mezi klientskou a serverovou částí aplikace zvolili architekturu typu REST. Hlavním důvodem byla menší komplexnost implementace, menší velikost předávaných zpráv a odstranění nutnosti jejich transformace, neboť chystaná bankovní API budou také vyvinutá podle principu REST. Pro návrh serverové části jsme nejprve představili architekturu mikroservis jako flexibilnější řešení než které představuje monolitická architektura. Podle tohoto představení jsme dekomponovali logiku serverové části aplikace a na základě této dekompozice jsme logiku rozdělili mezi jednotlivé mikroservisy.

Systémová specifikace se dále věnuje návrhu dokumentové databáze, tedy v jakých kolekcích dokumentů budou aplikační data ukládána, aby práce s nimi

byla efektivní.

Během návrhu jsme také zmapovali jednotlivé stavy, kterými prochází klíčové entity aplikace: Platba a Žádost o platbu. Obě entity se během svého životního cyklu mohou postupně nacházet v několika různých stavech, mezi kterými je nutné mít správně definované možné přechody.

Na závěr systémové specifikace jsme definovali aplikační procesy, které budou podporovat případy použití. Podrobně jsme zde tak rozpracovali procesy registrace, přihlašování, zadávání nových transakcí a dokončení Žádosti o platbu.

Na základě vypracovaného návrhu jsme následně implementovali prototyp mobilní aplikace pomocí frameworku Apache Cordova. Navrženou dokumentovou databázi jsme realizovali pomocí MongoDB. Pro implementaci serverové části jsme využili nástroj Deployd, který umožňuje vytvořit požadované kolekce a následně k nim definovat logiku REST API. Výsledný prototyp splňuje funkční požadavky a podporuje aplikační procesy definované v systémové specifikaci. Výsledný prototyp tak slouží k demostraci navrhované funkcionality.

V závěru práce jsme stanovili odhady pracnosti implementace produkční verze aplikace na základě empirických odhadů a zvolených heuristik. Z tohoto odhadu vychází také stanovení očekávaných nákladů na implementaci projektu a rámcový harmonogram prací. V této části práce zdůrazňuje také potřebu vhodné marketingové kampaně jako nástroje pro úspěšné uvedení produktu na trh. Na závěr práce jsme představili metriky měření úspěšnosti, které slouží jako podklad pro budoucí vyhodnocování úspěšnosti projektu.

Výsledkem práce je tak systémová specifikace navržené mobilní aplikace, ze které je možné vycházet v rámci implementace produkční verze aplikace. Funkční prototyp, vytvořený na základě upraveného návrhu, demonstruje výslednou funkcionalitu a lze jej využít v rámci uživatelského testování aplikace, případně z něj vycházet v rámci implementace produkční verze aplikace. Společně se stanoveným odhadem nákladů a rámcovým harmonogramem práce vytváří podklad pro rozhodování o implementaci produkční verze aplikace. Na základě systémové specifikace a funkčního prototypu je možné zhodnotit přínosy aplikace v kontrastu se stanovenými náklady a pracností.

Literatura

[1] The Apache Software Foundation: Cordova Architecture. 2015. Dostupné z: https://cordova.apache.org/docs/en/latest/guide/overview/index.html

[2] Fowler, M.: Richardson Maturity Model. 05 2010. Dostupné z: https://martinfowler.com/articles/richardsonMaturityModel.html

[3] Kosek, J.: Využití webových služeb a protokolu SOAP při komunikaci. 2005. Dostupné z: http://www.kosek.cz/diplomka/html/websluzby.html

[4] Fowler, M.: Microservices. 2014. Dostupné z: https://www.martinfowler.com/articles/microservices.html

[5] Temenos Group AG: Payment Services Directive 2, 2016, paper describing context and impact of european legislative known as PSD2. Dostupné z: https://www.temenos.com/globalassets/mi/wp/16/temenos_psd2_whitepaper_v2.pdf

[6] PWC: PSD2 v kostce, 2016. Dostupné z: https://www.pwc.com/cz/cs/bankovnictvi/assets/psd2-v-kostce-n01-cz.pdf

[7] EBA Working Group: Understanding the business relevance of Open APIs and Open Banking for banks, 03 2016, information paper. Dostupné z: https://www.abe-eba.eu/downloads/knowledge-and-research/EBA_May2016_eAPWG_Understanding_the_business_relevance_of_Open_APIs_and_Open_Banking_for_banks.pdf

[8] Bolotin, L.: The Open Banking Standard, 2016, unlocking the potential of open banking. Dostupné z: https://www.paymentsforum.uk/sites/default/files/documents/Background%20Document%20No.%202%20-%20The%20Open%20Banking%20Standard%20-%20Full%20Report.pdf

[9] The Open Banking Working Group: Open Banking Standard. 2016, helping customers, banks and regulators take banking into digital economy. Dostupné z: https://hollandfintech.com/wp-content/uploads/2016/02/298568600-Introducing-the-Open-Banking-Standard.pdf

[10] Hong Kong Administrative: Peer-to-Peer network. 2008. Dostupné z: https://www.infosec.gov.hk/english/technical/files/peer.pdf

[11] Investopedia: Peer-to-Peer Service. 2016. Dostupné z: http://www.investopedia.com/terms/p/peertopeer-p2p-service.asp

[12] InvestingAnswers: Person-to-Person Payments. 2016. Dostupné z: http://www.investinganswers.com/financial-dictionary/personal-finance/person-person-payments-p2p-2584

[13] Česká spořitelna: Uživatelská příručka služeb, 2017. Dostupné z: http://www.csas.cz/static_internet/cs/Obchodni_informace-Produkty/Prime_bankovnictvi/Soukroma_klientela/Prilohy/S24_prirucka.pdf

[14] Meier, J.; Homer, A.; Hill, D.: Mobile Application Architecture Guide, 2017. Dostupné z: http://www.csas.cz/static_internet/cs/Obchodni_informace-Produkty/Prime_bankovnictvi/Soukroma_klientela/Prilohy/S24_prirucka.pdf

[15] Korf, M.; Oksman, E.: Understanding Mobile Application Development Options. 06 2016. Dostupné z: https://developer.salesforce.com/page/Native,_HTML5,_or_Hybrid:_Understanding_Your_Mobile_Application_Development_Options

[16] IBM: Native, web or hybrid mobile-app development, 04 2012. Dostupné z: ftp://public.dhe.ibm.com/software/pdf/mobile-enterprise/WSW14182USEN.pdf

[17] Hujer, M.: Mobilní aplikace pomocí webových technologií. 01 2014. Dostupné z: https://www.zdrojak.cz/clanky/cordova-sencha-touch-mobilni-aplikace/

[18] Aminer, S.: Web Services: SOAP vs. REST. 11 2006. Dostupné z: https://static.aminer.org/pdf/PDF/000/369/383/the_quest_for_the_web_services_stack_a_fast_trip.pdf

[19] Malý, M.: REST: architektura pro webové API. 08 2009. Dostupné z: https://www.zdrojak.cz/clanky/rest-architektura-pro-webove-api/

[20] Fielding, R.; Irvine, U.; Gettys, J.: Hypertext Transfer Protocol – HTTP/1.1. 06 1999. Dostupné z: https://www.w3.org/Protocols/HTTP/1.1/rfc2616.pdf

[21] Mitra, N.; Lafon, Y.: SOAP version 1.2. 06 1999. Dostupné z: https://www.w3.org/TR/2007/REC-soap12-part0-20070427/

[22] A Comparative study of SOAP Vs REST Web Services Provisioning Techniques for Mobile Host. *Journal of Information Engineering and Application*, ročník 2, č. 5, 2012.

[23] Kumari, V.: Web Services Protocol: SOAP vs REST. 03 2015. Dostupné z: http://ijarcet.org/wp-content/uploads/IJARCET-VOL-4-ISSUE-5-2467-2469.pdf

[24] Daya, S.; Duy, N. V.; Eati, K.: Microservice from Theory to Practice, 09 2015. Dostupné z: https://www.redbooks.ibm.com/redbooks/pdfs/sg248275.pdf

[25] Kovář, M.: *User Experience v mobilní aplikaci pro provádění Peer-to-Peer plateb*. Diplomová práce, České vysoké učení technické v Praze, Fakulta informačních technologií, Praha, 2017.

[26] Sparx Systems: Domain model. 2013. Dostupné z: http://www.sparxsystems.com/enterprise_architect_user_guide/10/modeling_basics/domain_model_pattern.html

[27] Malan, R.; Bredemeyer, D.: Functional Requirements and Use Cases. 2001. Dostupné z: http://www.bredemeyer.com/pdf_files/functreq.pdf

[28] Sparx Systems: State machine diagram. 2013. Dostupné z: http://www.sparxsystems.com.au/resources/uml2_tutorial/uml2_statediagram.html

[29] Gulzar, N.: MVC architecture. 2002. Dostupné z: http://media.techtarget.com/tss/static/articles/content/StrutsFastTrack/StrutsFastTrack.pdf

[30] Mintz, L.: How to Determine the Perfect Marketing Budget for Your Company. 2015. Dostupné z: https://www.entrepreneur.com/article/243790

Příloha **A**

Seznam použitých zkratek

API Application Programming Interface

CRUD Create, Read, Update, Delete

CSS Cascade Style Sheet

DOM Document Object Model

ERP Enterprise Resource Planning

HATEOAS Hypertext As The Engine Of Application State

HTML HyperText Markup Language

HTTP Hypertext Transfer Protocol

HTTPS Hypertext Transfer Protocol Secure

IDE Integrated Development Environment

JS Javascript

JSON Javascript Object Notation

MD Man-day

MVC Model, View, Controller

PaaS Platform-as-a-Service

POX Plain Old XML

PSD2 Payment Services Directive 2

P2P Peer-to-Peer

REST Representational State Transfer

RPC Remote Procedure Call

SAML Security Assertion Markup Language

SOAP Simple Object Access Protocol

SSL Secure Sockets Layer

UC Use Case

UI User Interface

URI Uniform Resource Identifier

URL Uniform Resource Locator

UX User Experience

WSDL Web Service Description Language

XML eXtensible Markup Language

XSD XML schema definition

Příloha B

Příklady volání

B.1 Aktualizace informací o účtu

Při získání aktuálních informací o uživatelském účtu je aplikaci ze serverové strany vrácena následující odpověď:


```
{
  "fullName": "David Kukacka",
  "bankAccount": {
    "accountPrefix": "0000",
    "accountNumber": "25487661",
    "bankCode": "0800",
    "accountName": "My account"
  },
"phoneNumber": "744513156",
  "username": "David",
  "accountBalance": 4409,
  "id": "695059bb64f48a28",
  "incomingPayments": [],
  "outgoingRequests": [
    {
      "initiator": "695059bb64f48a28",
      "initiatorDetail": {
        "fullName": "David Kukacka",
        "phone": "744513156"
      },
      "reciever": "7bf3bf8ce95c19fd",
      "recieverDetail": {
        "fullName": "Jan Stransky",
        "phone": "777888999"
      },
```

```
      "amount": 52,
      "message": "Vypalne ",
      "submitDate": "1489946867988",
      "state": "recieved",
      "id": "fc5fb69ba456c927"
    }
  ],
  "incomingRequests": [],
  "outgoingPayments": []
}
```

B.2 Formát kontaktního seznamu

Aplikace pomocí Apache Cordova cordova-contact-plugin načte z mobilního zařízení kontaktní list zařízení. Tento seznam je získán v JSON syntaxi následujícího formátu:

```
[{
    "id": "123",
    "rawId": "123",
    "displayName": "",
    "name": {
      "givenName": "",
      "formatted": ""
    },
    "nickname": null,
    "phoneNumbers": [
      {
        "id": "1711",
        "pref": false,
        "value": "972+54+7777777",
        "type": "mobile"
      }
    ],
    "emails": [],
    "addresses": [],
    "ims": [],
    "organizations": null,
    "birthday": null,
    "note": "",
    "photos": null,
    "categories": null,
    "urls": null
  }]
```

B.3 Funkce pro získání názvu kontaktu

Způsob uložení názvu telefonního kontaktu se liší mezi operačními systémy Android a iOS. Následující funkce se postará o vrácení správného názvu kontaktu nezávisle na operačním systému.

```
function getName(c) {
    var name = c.displayName;
    if(!name || name === "") {
        if(c.name.formatted) return c.name.formatted;
        if(c.name.givenName && c.name.familyName)
        return c.name.givenName +" "+c.name.familyName;
        if(c.name.givenName) return c.name.givenName;
        if(c.name.familyName) return c.name.familyName;
        return "Nameless";
    }
    return name;
}
```

Příloha C

Stanovení nákladů a harmonogramu vývoje

Tato příloha se zabývá postupem pro stanovení nákladů a harmonogramu projektu implementace aplikace pro provádění Peer-to-Peer plateb.

C.1 Stanovení nákladů

Pro stanovení nákladů projektu využijeme empirické odhady a vybrané heuristiky, které byly výsledkem konzultace daného tématu se softwarovým architektem banky. Náklady na projekt jsme schopni rozdělit do tří dílčích částí:

1. Náklady na vytvoření UI návrhu.
2. Projektové náklady - architektura, implementace, testování, řízení projektu.
3. Marketing a propagace produktu

C.1.1 Náklady na UI návrh

Do nákladů za vývoj UI budeme započítávat následující aktivitu:

1. Vytvoření UX návrhu aplikace formou prototypu včetně grafiky.
2. Kvalitativní testování prototypu s cílem identifikovat slabá místa.

Tyto aktivity by měly být opakovány alespoň třikrát, aby byl výsledný návrh dostatečně kvalitní. Náklady na vytvoření UI návrhu stanovíme podle heuristiky uvedené v tabulce C.1.

Typ stránky	Náročnost (hodin)	Počet v prototypu
Jednoduchá stránka	2	13
Středně náročná stránka	4	6
Obtížná stránka	8	1
Celková náročnost		56 hodin

Tabulka C.1: Heuristika pro určení nákladů spojených s návrhem UI.

Do celkové náročnosti vytvoření UI návrhu je následně nutné započítat výše uvedené tři iterace, kterými bude návrh procházet. Pracnost UI návrhu po provedených iteracích spočítáme podle následujícího vzorce:

$$designTime = n * timeOfIteration = 3 * 56 = 168 \tag{C.1}$$

, kde n značí počet iterací, $timeOfIteration$ náročnost jedné iterace a výsledek vzorce je v hodinách. Výsledná hodnota značí náročnost vypracování kvalitního návrhu v čistém čase.

Abychom zjistili skutečnou časovou náročnost zpracování návrhu, musíme do výpočtu zapracovat ještě efektivitu lidských zdrojů, která je heuristicky stanovena na 70%. Skutečný čas, potřebný k vytvoření návrhu tak vypočteme pomocí vzorce

$$UI_{time} = designTime * \frac{10}{7} = 240 \tag{C.2}$$

, kde $designTime$ značí čistý čas návrhu, výsledek je opět uveden v hodinách. Vydělíme-li toto číslo osmi, dostaneme dobu tvorby UI návrhu v jednotce MD[6]. Celková doba potřebná na vytvoření a otestování kvalitního UI návrhu aplikace je tedy 30 pracovních dní.

C.1.2 Projektové náklady

Pro empirický odhad pracnosti implementace aplikace vyjdeme ze zkušenosti s implementací prototypu a ze zkušenosti systémové architekta s podobnými aplikacemi. Pracnost systémové specifikace a následné implementace prototypu byla stanovena na 80 MD. Náročnost implementace prototypu a produkční verze aplikace se však bude v mnohém lišit. Pro určení náročnosti implementace produkční verze aplikace tak využijeme další heuristiku, popsanou vzorcem:

$$totalTime = prototypeImplementationTime * 4 \tag{C.3}$$

Tento vzorec tedy definuje, že pracnost vytvoření a nasazení produkční verze aplikace se rovná čtyřnásobku doby implementace. V tomto čase je započítána analýza, architektura a návrh, zabezpečení, nasazení na produkční prostředí a projektové řízení.

[6]Man-day, jednotka práce, která reprezentuje jeden pracovní den člověka.

Opět se však jedná o čistý čas, skutečný čas budeme muset upravit podle výše uvedeného vzroce. Celkový čas vytvoření produkční verze aplikace v MD dle definovaného UI návrhu je dán vzorcem:

$$totalTime = 80 * 4 * \frac{10}{7} = 457 \tag{C.4}$$

Výsledný čas nám udává pracnost projektu implementace mobilní aplikace pro provádění Peer-to-Peer plateb.

C.1.3 Náklady na marketing

Odhadovat pracnost marketingu není účelné, neboť narozdíl od předchozích aktivit se jedná o do určité míry nekončící činnost. Náklady za marketing tedy budeme odhadovat za dobu jednoho roku. Při samotném odhadu budeme vycházet opět z kvalitativního odhadu za jednotlivé klíčové aktivity, které jsou součástí většiny marketingových kampaní[30]. Tento empirický odhad je uveden v tabulce C.2.

Aktivita	Odhad nákladů v Kč
Kreativa	500 000
Webová prezentace	450 000
Sociální média	300 000
Placená reklama	200 000
Tvroba obsahu	600 000
Pořádání událostí	300 000
Analýza dopadu kampaně	12* 50 000
Celkové náklady za 1 rok	2 950 000 Kč

Tabulka C.2: Empirické odhady nákladů za první rok marketingu.

C.1.4 Celkové náklady spojené s projektem

Abychom na základě pracnosti byli schopni určit náklady na projekt, je nutné definovat cenu za jednotku MD. Hodnota jednoho MD se obecně u podobných typů určuje jako 10 000 Kč s DPH / 1 MD. Cena je spočítána jako průměr nákladů za externí a interní pracovníky. V tabulce C.3 jsou zachyceny náklady na jednotlivé části projektu. Do výpočtu nejsou započítány náklady na HW.

Část projektu	Náročnost (MD)	Náklady (Kč)
Návrh a design	30	30 000
Implementace formou projektu	457	4 570 000
Marketing	-	2 950 000
Celková náklady		7 550 000 Kč

Tabulka C.3: Odhad nákladů na jednotlivé části projektu.

Celkové náklady na implementaci projektu a kvalitní marketingovou kampaň po dobu prvního roku tak byly odhadnuty na 7 550 000 Kč.

C.2 Harmonogram projektu

Na základě výše uvedených údajů jsme schopni vytvořit hrubý harmonogram prací na projektu. Jelikož však vycházíme pouze z odhadů pracnosti, nelze považovat harmonogram za přesný. Jedná se pouze o hrubé plánování náročnosti a závislosti aktivit. Pracnost projektu, kterou jsme stanovili za pomocí heuristiky na základě náročnosti analýzy a implementace prototypu, budeme nyní muset rozdělit mezi jednotlivé klíčové aktivity. Toto rozdělení je uděláno na základě emprického odhadu, výsledek je zachycen v tabulce C.4.

Část projektu	Náročnost (%)	Poznámka
Návrh UI	samostatný výpočet	Řešeno jedním člověkem na plný úvazek. Stanovená náročnost je 30 MD.
Architektura a zabezpečení	25%	Jeden systémový architekt, jeden bezpečnostní analytik. Odhadovaná vytíženost na poloviční pracovní úvazek.
Implementace FE	20%	Závislé na dokončení fázi Návrh UI. Počítáno se dvěma developery na plný pracovní úvazek.
Implementace BE, nasazení aplikace	20%	Závislé na prvních výstupech návrhu a architektury. Počítáno se dvěma developery na plný pracovní úvazek.
Testování aplikace	25%	Závislé na výstupech architektury, implementace FE, implementace BE. Počítáno se dvěma testery na plný pracovní úvazek.
Projektové řízení a ostatní aktivity	10%	Bude probíhat po dobu celého projektu.

Tabulka C.4: Rozdělení projektu do aktivit pro vytvoření hrubého harmonogramu.

Ganttův diagram, do kterého je promítnuta závislost zahájení vybrané aktivity na výstupech předchozích aktivit a alokace více zdrojů, je zobrazen na obrázku 7.1.

Příloha D

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `exe` ..... adresář se spustitelnou formou implementace
- `src`
  - `impl` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF
  - `thesis.ps` ..... text práce ve formátu PS