VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

SYSTÉM PRO INICIACI PLATEB A ČTENÍ INFORMACÍ Z BANKOVNÍHO ÚČTU V RÁMCI PSD2

SYSTEM FOR PAYMENT INITIATION AND READING INFORMATION FROM BANK ACCOUNT WITH PSD2

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **PETR JŮDA**
AUTHOR

VEDOUCÍ PRÁCE **Ing. VLADIMÍR BARTÍK, Ph.D.**
SUPERVISOR

BRNO 2018

Zadání bakalářské práce/20518/2017/xjudap00

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav informačních systémů Akademický rok 2017/2018

Zadání bakalářské práce

Řešitel: **Jůda Petr**
Obor: Informační technologie
Téma: **Systém pro iniciaci plateb a čtení informací z bankovního účtu v rámci PSD2**
System for Payment Initiation and Reading Information from Bank Account with PSD2
Kategorie: Informační systémy

Pokyny:

1. Seznamte se se stávajícím informačním systémem firmy Investiční aukce a programovacím jazykem Python.
2. Prostudujte API zvolené banky pro práci s bankovními účty (na základě direktivy EU PSD2).
3. Navrhněte rozšíření stávajícího IS o možnost čtení informací z bankovního účtu a iniciace plateb prostřednictvím zvoleného API, které bude obsahovat obecnou mezivrstvu umožňující napojení na API více různých bank.
4. Navržené rozšíření implementujte a ověřte jeho funkčnost.
5. Zhodnoťte dosažené výsledky a navrhněte další možné pokračování tohoto projektu.

Literatura:

- Pilgrim, M.: Ponořme se do Python(u) 3. CZ.NIC, 2010. ISBN: 978-80-904248-2-1.
- European Comission: Payment services (PSD 2) - Directive (EU) 2015/2366. Dostupné na: https://ec.europa.eu/info/law/payment-services-psd-2-directive-eu-2015-2366_en
- Fio Banka: API Bankovnictví (dokumentace). Dostupné na: https://www.fio.cz/bankovni-sluzby/api-bankovnictvi

Pro udělení zápočtu za první semestr je požadováno:

- Body 1-2, částečně bod 3.

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Bartík Vladimír, Ing., Ph.D.**, UIFS FIT VUT
Datum zadání: 1. listopadu 2017
Datum odevzdání: 16. května 2018

L.S.

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav informačních systémů
612 66 Brno, Božetěchova 2

doc. Dr. Ing. Dušan Kolář
vedoucí ústavu

Abstrakt

Cílem této práce je vytvořit systém, který firmě Platební instituce Roger a.s. umožňuje iniciovat platbu nebo stáhnout platební historii z bankovních účtů vedených u Fio banky. Komunikace s bankou je realizována přímo pomocí bankovního REST API, které je zpřístupněno na základě rozsáhlých legislativních změn v oblasti bankovnictví. Tyto změny přinesla nově aktualizovaná směrnice Evropského parlamentu a Rady EU o platebních službách na vnitřním trhu (PSD2), která bankovním institucím ukládá povinnost zpřístupnit veřejné rozhraní třetím stranám. Při tvorbě aplikace byl prozkoumán a zaznamenán současný stav zavádění PSD2 služeb na českém bankovním trhu. Program byl navržen tak, aby mohl být do budoucna dále rozšiřován.

Abstract

The purpose of this thesis is to create a payment system for the company Platební instituce Roger a.s. The application allows performing a payment initiation or downloading transaction history from Fio bank accounts. Communication is established directly through bank's REST API that was published as a result of extensive legal changes in the banking sector. These changes come from the new Payment System Directive 2 submitted by the European Commission. Due to this new law banks have to release a public interface to third-party companies. During the development of the application the current state of the implementation of PSD2 services in the Czech Republic has been explored and recorded. This application was designed with regard to extensibility. So that it could be continued to develop in the future.

Klíčová slova

PSD2, bankovní účet, Fio banka, iniciace platby, stažení transakční historie, REST API, bankovní rozhraní, Python, rámec web2py

Keywords

PSD2, bank account, Fio bank, initiation of payment, download transaction history, REST API, banking interface, Python, framework web2py

Citace

JŮDA, Petr. *Systém pro iniciaci plateb a čtení informací z bankovního účtu v rámci PSD2.* Brno, 2018. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Vladimír Bartík, Ph.D.

Systém pro iniciaci plateb a čtení informací z bankovního účtu v rámci PSD2

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Vladimíra Bartíka, Ph.D. Další informace mi poskytl Ing. Tomáš Slobodník. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Petr Jůda
28. dubna 2018

Poděkování

Rád bych poděkoval panu Ing. Vladimíru Bartíkovi, Ph.D. za odborné vedení mé práce, panu Ing. Tomášovi Slobodníkovi z firmy Platební instituce Roger a.s. za cenné rady a čas, který mi při tvorbě práce poskytnul. Chtěl bych také poděkovat své rodině a přátelům za poskytnutí podpory během studia.

Obsah

1 Úvod **3**

2 Směrnice PSD2 **5**
2.1 Schvalovací proces 5
2.2 Nově uzákoněné změny 6
2.3 Technické normy a bezpečnostní rizika 7
2.3.1 Silné ověření klienta 8
2.3.2 Bezpečnostní hrozba screen scraping 8

3 Architektura REST API **10**
3.1 Metody přístupu ke zdrojům 10
3.2 Ověření uživatele 11
3.2.1 Autentizace pomocí HTTP-basic 11
3.2.2 Autentizace pomocí HTTP-digest 12
3.2.3 Autentizace pomocí přístupového tokenu 12
3.3 Formáty pro reprezentaci zdrojů 15
3.3.1 JSON 15
3.3.2 XML 16
3.4 Příklad komunikace s REST API 18

4 Specifikace požadavků na systém **19**
4.1 Obecné požadavky 19
4.1.1 Architektura systému 19
4.1.2 Požadované operace 19
4.1.3 Aplikační rozhraní 20
4.2 Technologické požadavky 20
4.2.1 Programovací jazyk a rámec 20
4.2.2 Databázový systém 20

5 Přehled dostupných bankovních API **21**
5.1 Fio banka 21
5.1.1 Dostupné služby 21
5.1.2 Autentizace 22
5.1.3 Způsob iniciace platby 23
5.1.4 Čtení bankovní historie 25
5.2 Česká spořitelna 25
5.2.1 Dostupné služby 26
5.2.2 Autentizace 26

5.2.3 Způsob iniciace platby . . . 27
5.2.4 Čtení bankovní historie . . . 27
5.3 Banka Creditas . . . 27
5.3.1 Dostupné služby . . . 28
5.3.2 Autentizace . . . 28
5.3.3 Způsob iniciace platby . . . 28
5.3.4 Čtení bankovní historie . . . 29
5.4 Zhodnocení stavu . . . 29

6 Návrh řešení systému **30**
6.1 Aplikační rámec web2py . . . 30
6.1.1 Architektura . . . 31
6.1.2 Abstraktní databázová vrstva . . . 32
6.1.3 Logování . . . 33
6.1.4 Plánovač událostí . . . 33
6.2 Databázový model . . . 34
6.3 Dekompozice programu . . . 35
6.3.1 Mezivrstva . . . 37
6.3.2 Napojení na bankovní API . . . 38
6.4 Návrh testování aplikace . . . 38

7 Implementace **39**
7.1 Popis důležitých tříd . . . 39
7.1.1 Třída FioBank . . . 39
7.1.2 Třída FioValidator . . . 40
7.1.3 Třída FioApi . . . 42
7.2 Proces vytvoření platby . . . 43
7.3 Odeslání připravených plateb . . . 43
7.4 Stahování a zpřístupnění transakční historie . . . 45

8 Testování **46**
8.1 Automatické testování . . . 46
8.2 Manuální testování . . . 47

9 Závěr **48**

Literatura **49**

Přílohy **52**

A Diagram tříd **53**

B Obsah přiloženého paměťového média **55**

C Manuál **56**
C.1 Spuštění aplikace . . . 56
C.2 Návod k použití . . . 57

Kapitola 1

Úvod

Historie vzájemného obchodování a potřeba výměny zboží jsou úzce spjaty se samotným vývojem lidstva. Podstatným milníkem byl vynález peněz, který nahradil prostý směnný obchod. Následující rozvoj oběhu peněz s sebou přinesl potřebu vzniku bankovních institucí. Zavedením počítačů a jejich následným připojením do internetové sítě se od základu změnilo fungování bankovních systémů. Tato změna přispěla k masivnímu přílivu klientů a položila základy moderního bankovnictví.

Dnes je podstatná část transakcí tvořena pomocí elektronických peněz. Bez bankovního účtu bychom nebyli schopni tyto peníze přijímat ani odesílat. Lze tedy říci, že se bez něj již téměř neobejdeme a tvoří podstatnou součást našeho života. Většina bank nabízí možnost si vlastní bankovní účet založit pomocí osobního počítače či chytrého telefonu s internetovým připojením odkudkoliv na světě. Za účelem správy konta, získávání informací o příchozích transakcích a zadávání odchozích plateb je možné využít webové rozhraní internetové bankovnictví nebo mobilní aplikaci. Autory těchto prostředků jsou jednotlivé banky, které tyto možnosti nabízí svým vlastním zákazníkům. Pro výběr hotovosti, bezhotovostní platby v kamenných obchodech nebo převod prostředků pomocí online platební brány nám slouží platební karty.

Propojení jednotlivých bankovních systémů tvoří složitou uzavřenou finanční infrastrukturu s vysokým důrazem na bezpečnost. Všechny subjekty proto musí splňovat rozsáhlou státní a evropskou legislativu. Dohled a kontrolu nad dodržováním těchto pravidel má v České republice na starost Česká národní banka. Na základě velkého technického rozvoje, probíhajícího zejména v posledních letech, bylo nutné provést odpovídající legislativní úpravy a změny.

Tato práce byla zadána firmou Platební instituce Roger a.s. (dříve Investiční aukce s.r.o) a zaměřuje se na prozkoumání aktuálně nových možností zadávání a čtení plateb, které přináší aktualizovaná směrnice Evropského parlamentu a Rady EU o platebních službách na vnitřním trhu, která nedávno vstoupila v platnost. Tento zákon je též označován pod zkratkou PSD2. Jeho schválení přináší zásadní průlom vzhledem k zpřístupnění speciálních bankovních rozhraní i pro nebankovní instituce a veřejnost.

Cílem práce je navrhnout, implementovat a otestovat systém, který prostřednictvím těchto nových technologií umožní zadávat platební příkazy a získávat informace o příchozích platbách. Aplikace bude sloužit jako rozšíření aktuálního firemního systému a umožní automatizovat operace, které se doposud musely zadávat ručně. Na základě těchto inovací dojde k ušetření lidských zdrojů, eliminaci možných chyb vzniklých při zadávání a celkovému zrychlení událostí, které jsou ve společnosti navázány na bankovní převody. Aplikace bude založena na rozhraní, kterým nyní disponuje Fio Banka, a.s., u které má společnost

většinu svých bankovních účtů. Systémový návrh však musí vycházet i z ostatních doposud dostupných řešení tak, aby jej bylo možné nadále snadno rozšiřovat o podporu dalších bank.

Práce je rozdělena do 9 kapitol. Kapitola 2 shrnuje nejzásadnější informace ohledně nových legislativních úprav a jejich dopad na bankovní sektor. Kapitola 3 se věnuje popisu architektury REST API. Následující kapitola 4 specifikuje firemní požadavky na systém. Kapitola 5 zkoumá veřejně dostupná API řešení českých bank. Kapitola 6 obsahuje návrh rozdělení funkcionality a průběhu testování aplikace. Implementace aplikace je popsána v kapitole 7. V posledních dvou kapitolách 8 a 9 je obsažen popis testování aplikace a celkové shrnutí spolu s plánem dalšího rozvoje.

Kapitola 2

Směrnice PSD2

V této kapitole jsou uvedeny souhrnné informace, které se týkají směrnice Evropského parlamentu a Rady evropské unie č. 2015/2366 ze dne 25.listopadu 2015, označované jako PSD2 (*Payment Services Directive verze 2*). Předchozí platná směrnice PSD (*Payment Services Directive*) č. 2007/64/ES byla přijata v prosinci roku 2007 na základě návrhu Evropské komise vytvořeného již v roce 2005. Od té doby došlo na trhu pro malé platby k významným technickým pokrokům, rychlému nárůstu počtu elektronických a mobilních plateb a vzniku nových druhů platebních služeb, jež pro dosavadní legislativu představovaly velké výzvy. Na řadu inovativních platebních produktů a služeb se působnost směrnice PSD zcela nebo z velké části nevztahovala. Ukázalo se, že zejména vynětí některých prvků z působnosti zákona, jako například určité činnosti související s platbami, byly vzhledem k vývoji trhu specifikovány příliš obecně, nejednoznačně nebo byly prostě zastaralé. To v určitých oblastech vedlo k právní nejistotě a k potenciálním bezpečnostním rizikům v platebním řetězci a nadále nedostatečné ochraně spotřebitele. V posledních letech vzrostla bezpečnostní rizika související s elektronickými platbami. To je způsobeno tím, že elektronické platby jsou technicky složitější a jejich objem celosvětově nepřetržitě roste. Zajištění dalšího rozvoje bezpečných elektronických plateb a ochrany spotřebitelů má zásadní význam pro podporu růstu ekonomiky EU. Tyto a další důvody vedly k rozhodnutí, že je potřeba přijmout nová pravidla, která zaplní mezery v původních předpisech a zároveň zaručí větší srozumitelnost a jednotné uplatňování tohoto právního rámce v celé Evropské unii. [10]

2.1 Schvalovací proces

První návrh nové direktivy byl do Evropského parlamentu předložen v červnu 2013. Finální text byl schválen a publikován v prosinci roku 2015. Od 13. ledna 2016 tak nabyl právní účinnost. Tímto okamžikem započalo dvouleté období, ve kterém měly jednotlivé členské státy Evropské unie provést nutnou úpravu své národní legislativy tak, aby odpovídala předepsaným nařízením. V České republice byly změny zapracovány schválením zákona č. 370/2017 Sb. o platebním styku. [30] Potřebné právní úpravy jsou v ČR platné od 13. ledna 2018. Vzhledem k zdlouhavým legislativním procesům se však nepodařilo potřebné zákony schválit všem členským státům EU. Aktuální stav zavádění je možné sledovat na webových stránkách Evropské komise. [8]

2.2 Nově uzákoněné změny

Aktualizovaný zákon přináší široké spektrum změn ve finančním sektoru. Dále se zde budeme zabývat pouze těmi, které se týkají tématu této práce. PSD2 výrazným způsobem otevírá trh tzv. třetím stranám. TPP (Third Party Payment Service Providers) jsou poskytovatelé platebních služeb. Tyto nebankovní platební instituce budou nyní moci na základě souhlasu klienta získat přístup k jeho bankovnímu účtu.

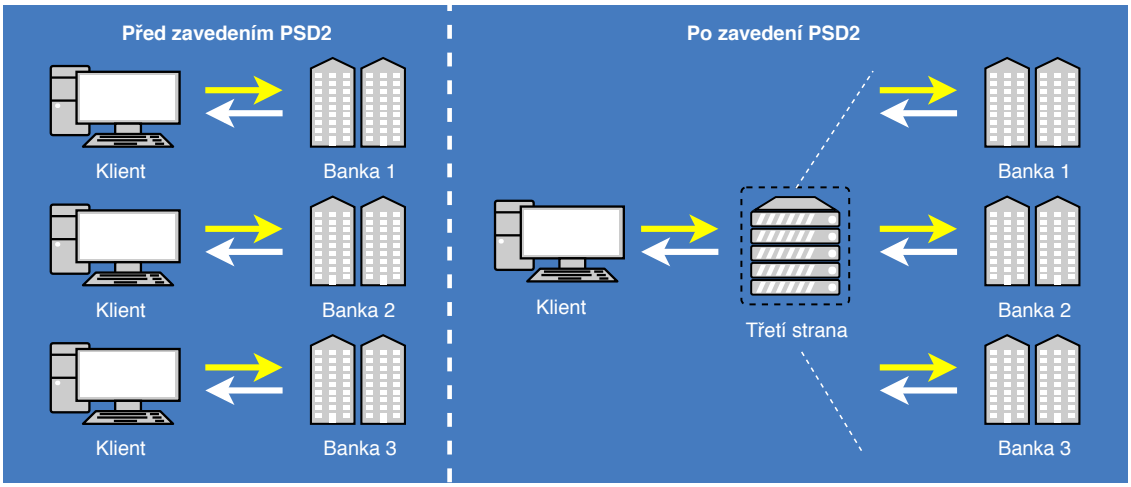
Nově zaváděné služby lze rozdělit do dvou kategorií:

- **Payment Initiation Service** (PIS) - Služba platební iniciace umožní třetím stranám iniciovat platbu z účtu klienta (plátce), aniž by k tomu bylo nutné použít jeho platební kartu, jak tomu bylo doposud. K iniciování těchto plateb dochází prostřednictvím softwarového mostu mezi systémem třetí strany a platformou pro internetové bankovnictví poskytovatele platebních služeb, který vede účet plátce (banka). [14]
- **Account Information Service** (AIS) - Služba informování o platebním účtu umožní třetím stranám získávat data klienta. Společnost tak bude mít přístup k jeho transakční historii a peněžnímu zůstatku.

Tyto nově zavedené služby přináší kromě velkých příležitostí i další potenciální bezpečnostní hrozby. Aby bylo možné nakládání s klientskými daty kontrolovat a regulovat, musí se platební instituce třetích stran registrovat u lokálně příslušných regulátorů. Subjekty založené v České republice mohou o potřebnou licenci zažádat u České národní banky. [29]

PSD2 ukládá povinnost jednotlivým bankám zpřístupnit tyto služby autorizovaným institucím prostřednictvím nového API (*Application Programming Interface*). Směrnice však nestanovuje jednotný standard tohoto rozhraní. Každá banka může využít rozdílné technologie, což značně komplikuje a rozšiřuje nároky na technická řešení TPP.

Na základě těchto technických prostředků mohou třetí strany poskytovat svým klientům zcela nové služby. Například přehled o událostech na platebních účtech klienta vedených u různých bank z jediné webové stránky nebo mobilní aplikace. Tato situace je zachycena na obrázku 2.1. Další výraznou kategorií uplatnění je oblast automatizace.



Obrázek 2.1: Jednotné rozhraní pro správu účtů vedených u různých bank. (Autor: vlastní)

2.3 Technické normy a bezpečnostní rizika

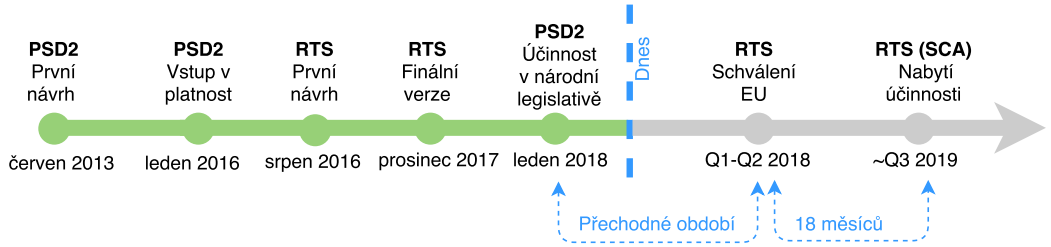
Zároveň s projednáváním změn, které jsou popsány v sekci 2.2, bylo nutné začít se věnovat i bezpečnostním otázkám a technickým standardům těchto služeb. Bylo zjištěno, že je potřeba uzákonit další nařízení, která by poskytovateli platebních služeb (bankám), přesně definovala, jakým způsobem bude prováděno ověření uživatelů a další technické parametry. Tyto subjekty jsou v direktivě PSD2 označovány jako ASPSP (*Account Servicing Payment Service Provide*). [10]

Vypracování těchto dokumentů má na starosti Evropský orgán pro bankovnictví (EBA). Prvotní jednání započala již v roce 2015. V polovině roku 2016 byly předloženy první návrhy regulujících technických standardů (RTS) a implementačních technických standardů (ITS). Tímto okamžikem započalo jejich projednávání. Vzhledem k rozsáhlému připomínkovému řízení a množství pozměňovacích návrhů se tyto dokumenty do této doby nepodařilo schválit. Jelikož datum účinnosti PSD2 nebylo nijak svázáno s ustanovením RTS a ITS, nastalo dnem 13. ledna 2018 tzv. přechodné období mezi uzákoněním RTS a vstupem PSD2 v platnost. [27] PSD2 neukládá bankám postihy za nezveřejnění API v tomto časovém úseku. Mohou tak čekat na schválení potřebných standardů. Některé banky vsadily na nejistotu a na svých API řešeních pracovaly i bez záruky splnění všech norem s rizikem, že svá řešení budou muset v budoucnu znovu upravovat. Analýze aktuálně dostupných řešení se dále věnuje kapitola č. 5.

V únoru 2017 dokončila EBA finální verze návrhů dokumentů, které doporučila Evropské komisi a parlamentu ke schválení. [9] Tato pravidla by se již neměla výrazně měnit a jejich přijetí je odhadováno na první polovinu roku 2018.

RTS pro oblast silné autentizace a zabezpečené komunikace

Nejzásadnější dokument představuje regulace vázající se k silné autentizaci uživatelů (SCA). Regulace má své vlastní přechodné období trvající 18 měsíců od vstupu v platnost. Tuto dobu budou banky moci využít k prostudování potřebných specifik a představení svých odpovídajících řešení. Vypršením uvedené lhůty by měl celý nový legislativní systém již naplno fungovat. Kompletní časový průběh znázorňuje osa na obrázku č. 2.2.



Obrázek 2.2: Časová osa závádění legislativních změn. (Autor: vlastní)

Nejdiskutovanější úpravy a připomínky se v rámci procesu tvorby RTS týkaly oblastí *silného ověření klienta* a problému *screen scraping*. (Do češtiny lze přeložit jako škrábání obrazovky. Dále se však budeme držet anglického názvu, protože český výraz se v dané problematice nepoužívá.)

2.3.1 Silné ověření klienta

Silné ověření klienta (SCA) definuje pravidla, podle kterých musí být provedeno ověření uživatele, než dojde k udělení přístupu k jeho informacím. Dodržování těchto zásad je nezbytné pro zajištění bezpečné ochrany klientů. Silná autentizace vyžaduje využití alespoň dvou ověřovacích prvků z různých kategorií.

Bezpečnostní kategorie jsou rozděleny na tři skupiny: [11]

- **znalost** - prvek, který zná pouze klient (např. heslo, PIN kód)
- **držení** - prvek, který má k dispozici pouze klient (např. bezpečnostní token, mobilní telefon, platební karta)
- **inherence** - biometrické ověření klienta (např. digitální otisk prstu, sken oční duhovky)

Silné ověření klienta na základě těchto prvků vyústí k vygenerování ověřovacího kódu. Ověřovací kód musí být nenapodobitelný. Znalost předchozího ověřovacího kódu nesmí umožnit vygenerování nového. Z ověřovacího kódu nesmí být zjistitelný žádný ověřovací prvek. K vytvoření ověřovacího kódu musí být zajištěna nezávislost ověřovacích prvků. Pokud dojde k prolomení jednoho z těchto prvků, nesmí dojít k ohrožení integrity dalších. Všechny osobní bezpečnostní údaje klientů musí být v průběhu celého ověřovacího procesu (tj. zobrazení, přenos a ukládání) skryty a nemohou být společně s kryptografickými materiály ukládány ve formě nešifrovaných textových souborů.

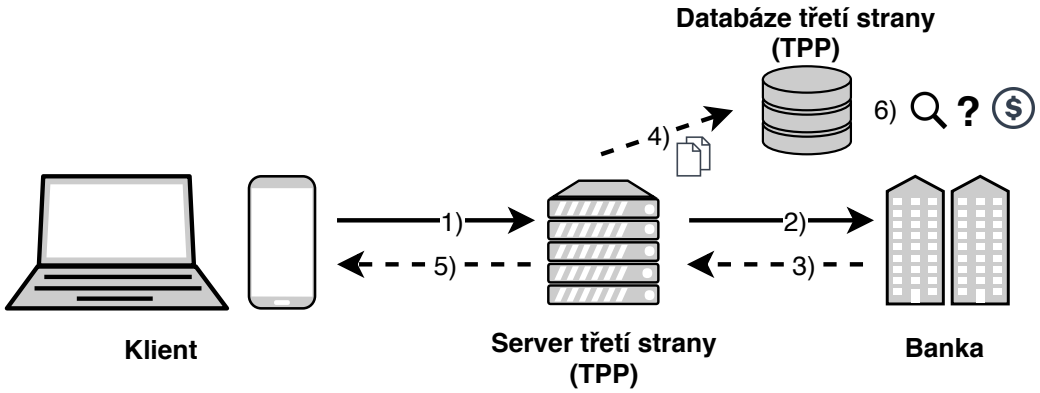
V pevně stanovených případech může poskytovatel platebních služeb využít výjimky z povinnosti provádět silné ověření klienta. Při schvalování takových transakcí musí zohlednit i další rizikové faktory, jakou jsou platební zvyklosti plátce, abnormální chování klienta vzhledem k jeho platební historii nebo geografická poloha plátce a příjemce prováděných finančních přesunů. Způsob zajištění bezpečnostních opatření musí být pečlivě dokumentován, periodicky testován a auditován. Auditní zpráva musí být na základě žádosti poskytnuta dozorčím orgánům. [11]

2.3.2 Bezpečnostní hrozba screen scraping

Screen scraping je jednou z technik sběru a analýzy klientských dat. Osobní data a z nich vytvářené profily zákazníků jsou v posledních letech stále cennější. Rozsáhlému sběru těchto informací se věnují i velké nadnárodní společnosti, jako jsou například Google nebo sociální síť Facebook. Prostřednictvím vyhodnocování těchto znalostí mohou automatizované systémy nabízet zákazníkům další podpůrné služby nebo na ně lépe cílit reklamu.

Zavedení nové služby *informování o platebním účtu* popsané v sekci 2.2 umožňuje třetí straně získat přístup k celé bankovní historii klienta. Platební instituce tak mohou během jediného okamžiku zjistit, jaké zboží a služby zákazník nakupuje nebo jak vysoký je jeho měsíční plat a zůstatek. Tato extrémně citlivá data by po provedení analýzy mohla být třetí stranou zneužita k vlastnímu užitku.

Chování tohoto typu jde hrubě proti základnímu smyslu direktivy PSD2, kterým má být zvýšení ochrany spotřebitele. Schvalování takového jednání by vedlo k možnému porušování bezpečnostních předpisů i ochraně osobních údajů. [7] Screen scraping je v této souvislosti graficky znázorněn diagramem na obrázku č. 2.3.



1) Klient provede prostřednictvým TPP autorizaci.
2) TPP se před bankou vydává za klienta a zažádá o klientská data.
3) Banka poskytne TPP požadované informace.
4) TPP si uloží kopii získaných informací.
5) TPP předloží data klientovi.
6) TPP nakládá s citlivými údaji.

Obrázek 2.3: Zakázané nakládání s citlivými údaji v podobě screen scraping. (Zdroj: vlastní)

Po upozornění Evropské bankovní federace na tento problém byla ve finální úpravě RTS možnost takto nakládat s citlivými údaji zcela zakázána. Je důležité mít na paměti, že dokud nevstoupí RTS v platnost, není zatím screen scraping nikterak právně omezen. Musíme si dávat zvýšený pozor na to, komu svěřujeme přístup k naším osobním údajům.

Kapitola 3

Architektura REST API

Architektura REST (*Representational State Transfer*) API (*Application Programming Interface*) je založena na klasickém modelu klient-server, kde klient zasílá požadavky na server (angl. *requests*) a ten na ně odpovídá (angl. *response*). Jako prostředek komunikace se zpravidla používá protokol HTTP (*Hypertext Transfer Protocol*). Pomocí něj lze získat přístup k datům umístěným na straně serveru. V dnešní době se jedná o masivně využívanou technologii, která se markantně prosazuje především u nově budovaných systémů. Na rozdíl od starších technologií jako jsou SOAP (*Simple Object Access Protocol*) či XML-RPC (*eXtensible Markup Language - Remote Procedure Call*), které byly orientovány procedurálně je REST zaměřen na zdroje (angl. *resources*). Zdrojem chápeme data a stavy aplikace, které jsme prostřednictvím dat schopni vyjádřit. Každý přístupný zdroj je rozlišen pomocí vlastního URI identifikátoru. [15]

3.1 Metody přístupu ke zdrojům

Pro přístup ke zdrojům definuje REST čtyři základní operace:

- **POST** - Požadavek slouží k *vytvoření zdroje*. Obsahuje data v předem specifikovaném tvaru, která jsou odeslána na domluvený společný identifikátor (angl. *endpoint*). Ten slouží pro vytváření stejné skupiny zdrojů.
- **GET** - Metoda pro *získání zdroje*. Jde o základní požadavek odeslaný pomocí HTTP funkce GET na URI adresu s cílem získat konkrétní data.
- **PUT** - Operace s cílem *změnit data zdroje*. Podobá se operaci POST s tím rozdílem, že požadavek je odeslán na konkrétní URI adresu zdroje, která je od jeho vytvoření známá.
- **DELETE** - Pro příkaz ke *smazání zdroje* lze využít volání HTTP metody DELETE na URI adresu zdroje, která je velmi podobná jako při použití GET.

Tyto operace jsou vykonávány prostřednictvím stejnojmenných metod implementovaných v protokolu HTTP. V REST je označujeme pod zkratkou CRUD. (*Create, Read, Update a Delete*) [15]

Odpověď na zaslaný požadavek obsahuje stavový kód, případně i data v některém serializačním formátu. Nejpoužívanější přenosové formáty jsou popsány v sekci č. 3.3.

Stavový kód reprezentuje výsledek operace. Interpretováním tohoto kódu lze zjistit, zda operace uspěla, případně jaký typ chyby vznikl při zpracování. Kódy jsou rozděleny do pěti

tříd. Skupiny určují o jakou kategorii výsledku se jedná a lze je rozlišit podle počáteční číslice. (tj. 1xx - informace pro klienta, 2xx - úspěšná operace, 3xx - přesměrování požadavku, 4xx - chyba na straně klienta, 5xx - chyba na straně serveru)

V REST API se můžeme nejčastěji setkat s následujícími kódy: [22] [21]

- **200 OK** - Server požadavek úspěšně zpracoval.
- **201 Created** - Vytvořen nový obsah na serveru. (Při použití POST.)
- **204 No Content** - Server požadavek zpracoval, ale nevrátil korektní odpověď.
- **304 Not Modified** - Klient požádal o data, která se od posledního požadavku nezměnila. (Při použití GET.)
- **400 Bad Request** - Špatně zadaný požadavek. (Např. syntaktická chyba.)
- **401 Unauthorized** - Klient není autentizován.
- **403 Forbidden** - Klient nemá oprávnění pro přístup k žádanému zdroji.
- **404 Not Found** - Požadovaný zdroj nebyl nalezen.
- **409 Conflict** - Požadavek nemohl být zpracován, kvůli konfliktu aktuálního stavu zdroje.
- **500 Internal Server Error** - Neočekávaná chyba na straně serveru.

Příklad s vysvětlením jednotlivých částí odpovědi na odeslaný požadavek je zachycen v sekci č. 3.4.

3.2 Ověření uživatele

Důležitou vlastností REST je bezstavovost komunikace. Pro ověřování uživatele nelze použít relace (angl. *sessions*) nebo *cookies*. Každý požadavek vyžadující autentizaci musí obsahovat informace, které jsou pro provedení ověření nezbytné. Zásluhou toho lze ušetřit značné výpočetní zdroje na straně serveru. Dále jsou rozepsány metody využívané k autentizaci uživatelů.

3.2.1 Autentizace pomocí HTTP-basic

Nejjednodušším způsobem ověření uživatele je prostá autentizace pomocí HTTP. Tato metoda je označována jako HTTP-basic. Do hlavičky požadavku jsou přidány přihlašovací údaje klienta. Přihlašovací jméno a heslo jsou společně zašifrovány pouze pomocí kódování Base64. Pokud se těchto údajů někdo zmocní, je bez problému schopný je dekódovat zpět do původní podoby. Z toho vyplývá, že údaje nejsou dostatečně zabezpečeny. Bez využití šifrování TLS (*Transport Layer Security*) na nižší vrstvě, je možné údaje zneužít pomocí obyčejného odposlechu sítě. Jde o nejprimitivnější a v praxi ve své základní podstatně nevyužívanou metodu. [19]

3.2.2 Autentizace pomocí HTTP-digest

Nevýhody prosté autentizace se snaží odstranit metoda HTTP-digest. Princip tohoto přístupu je založen na tom, že po síti již nejsou odesílány původní přihlašovací údaje, ale pouze jejich vypočtený otisk. Tento otisk slouží k prokázání totožnosti uživatele. K výpočtu se využívá hashovací funkce MD5. Na základě kombinace přihlašovacího jména, hesla, serverem vygenerované hodnoty a požadované URI adresy je vytvořen otisk, který je odeslán v hlavičce HTTP. Zvýšení bezpečnosti je možné přidáním dalších volitelných parametrů specifikovaných v QOP (*Quality Of Protection*). [20]

3.2.3 Autentizace pomocí přístupového tokenu

Dalším způsoben autentizace uživatele je využití softwarového přístupového API tokenu. Pod pojmem API token si můžeme představit textový řetězec o určité šířce, který na první pohled obsahuje náhodně vygenerované znaky. Na základě provedení ověření klienta (např. pomocí formuláře) je serverem vygenerovaný API token přidělen a zpřístupněn klientovi. Tento API token je následně v průběhu komunikace přenášen mezi klientem a serverem v hlavičce požadavku nebo tvoří přímo součást URI adresy. Softwarové tokeny můžeme rozdělit do kategorií, kterým lze následně přidělit skupiny oprávnění pro provádění určitých operací. Každý přístupový token by měl mít nastavenou svoji dobu platnosti. Po vypršení expirační doby je token zneplatněn a není možné jej nadále využívat. S tímto přístupem ověřování klientů se při používání moderních REST API setkáváme nejčastěji.

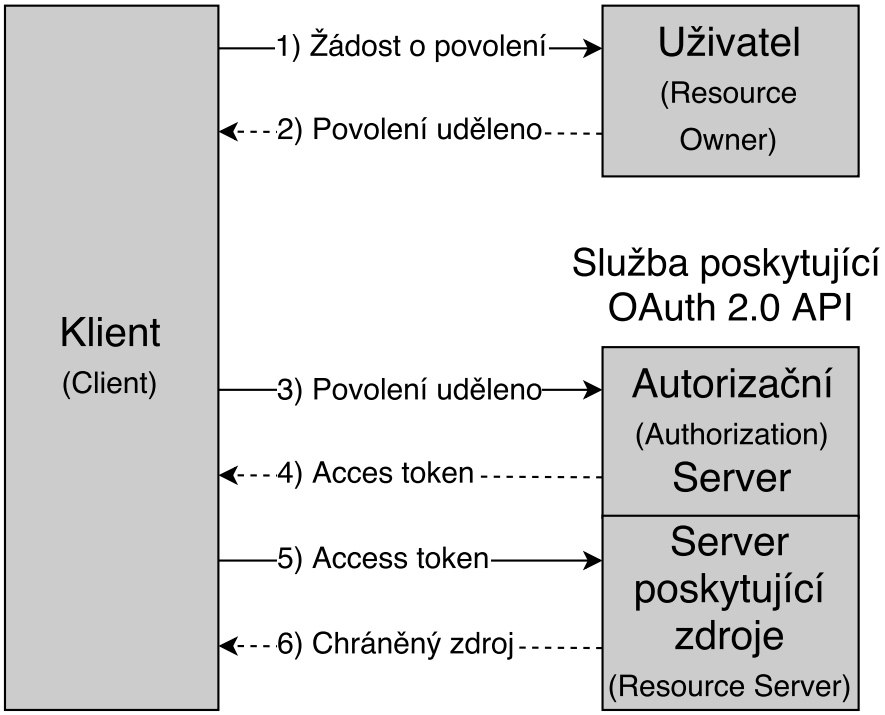
OAuth 2.0

OAuth 2.0 je sofistikovaný autentizační rámec postavený nad metodou autentizace pomocí přístupových tokenů. Jeho využití představuje vysokou úroveň zabezpečení. Jedná se o vylepšení původní verze, která byla standardizována v roce 2006. Rámec umožňuje, aby aplikace třetí strany provedla autentizaci vůči API poskytovatele OAuth služeb a získala tak omezený přístup k autorizovaným zdrojům uživatele. Po udělení souhlasu může žadatel přistupovat na pozadí k definovaným zdrojům. Cílem je poskytnout bezpečný způsob sdílení uživatelských dat navzájem mezi různými aplikacemi bez nutnosti toho, aby klient musel poskytovat své heslo všem. OAuth nabízí autorizaci pro přístup přímo z webu, desktopové aplikace nebo dalších mobilních zařízení. [13]

OAuth definuje čtyři role: [13]

- **resource owner** - Entita nebo osoba odpovědná za udělení rozsahu přístupu k chráněnému zdroji.
- **resource server** - Server poskytující chráněné zdroje, zpřístupněné na základě platného access tokenu.
- **authorization server** - Server zodpovědný za ověření uživatele. Na základě úspěšné autentizace zpřístupňuje access token pro autorizované zdroje.
- **client** - Klient (aplikace nebo zařízení) žádající o přístup k chráněnému zdroji. Před požadavkem musí získat povolení od *resouce owner* a provést autentizaci vůči *authorization server*.

Vzájemná komunikace mezi jednotlivými stranami nutná pro získání přístupu (access token) je nastíněna na obrázku č. 3.1.



Obrázek 3.1: Interakce mezi rolemi nutná pro získání access tokenu. (Zdroj: [13])

Před tím, než klient (aplikace) může využívat služby OAuth, musí se zaregistrovat u poskytovatele OAuth služeb. Registrace může proběhnout prostřednictvím API nebo webového portálu, pokud je vývojářům zpřístupněn. Při registraci je nutné zadat: [13]

- *Jméno aplikace*
- *Webovou stránku aplikace*
- *Redirect URI/Callback URI* - Předem domluvená adresa, na kterou OAuth API přesměruje klienta po provedení povolení. Tato část aplikace zpracovává autorizační kódy nebo access tokeny získané od autorizačního serveru.

Po úspěšné registraci získá klient dvojici identifikačních údajů: [13]

- **Client ID** - Veřejně známý unikátní identifikátor klienta. Slouží k rozlišení klienta na straně OAuth API.
- **Client secret** - Privátní klíč pro ověření klienta. Používá se mezi OAuth API a klientem v momentě, kdy klient zažádá o přístup k uživatelskému účtu.

Pro získání ověření k přístupu od uživatele definuje OAuth čtyři typy žádostí o povolení. Každá z možností se využívá pro jiné druhy případů užití: [13]

- **Authorization Code** - Klient požadující ověření přesměruje uživatele na adresu autorizačního serveru. Ten je zodpovědný za provedení ověření. Přihlašovací údaje jsou

předloženy pouze na straně autorizačního serveru a jsou tak od klienta zcela odstíněny. Po úspěšném provedení autentizace je odeslán autorizační kód zpět na adresu *Callback URI*, která byla definována při registraci klienta. S tímto kódem kontaktuje klient autorizační server za účelem získání Access/Refresh tokenu. Při tomto procesu jsou prováděny i další kontroly, např. zda adresa, ze které přišel požadavek, odpovídá adrese uvedené při registraci.

- **Implicit** - Implicitní metoda představuje zjednodušení přístupu *Authorization Code*. Je optimalizovaná pro webové klienty používající skriptovací jazyky, jako např. Javascript. Počáteční posloupnost operací je shodná s původní metodou. Rozdíl nastává v akci následující po ověření klienta. Na rozdíl od odeslání autorizačního kódu je klientovi přímo zaslán access token. Tento způsob umožňuje získat pro určité typy klientů vyšší efektivitu. Zvyšuje však i bezpečnostní rizika na straně klienta.

- **Client Credentials** - Tento typ přístupu je používán v případě, že klient je současně majitelem zdrojů. Není zde vyžadována spolupráce koncového uživatele. Klient získá od autorizačního serveru access token na základě svých identifikačních údajů *Client ID* a *Client Secret*.

- **Resource Owner Password Credentials** - Poslední řešení je možné využít pouze pokud existuje vzájemná důvěra mezi klientem a uživatelem. Přihlašovací jméno a heslo je v tomto případě svěřeno přímo do rukou klienta. Ten tyto údaje použije při prvotním kontaktu autentizačního serveru. Na základě ověření je klientovi vystaven access token, případně i dlouhodobě platný refresh token, které si klient může uložit.

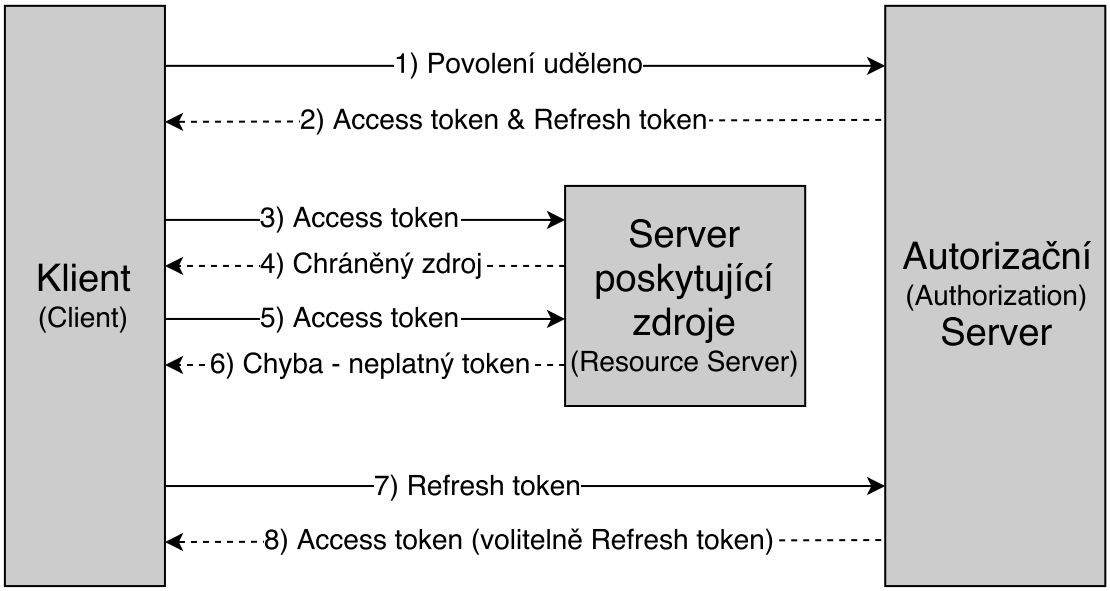
Podrobné informace a příklady pro jednotlivé metody je možné nalézt v RFC 6749. [13]

Splnění všech potřebných prerekvizit umožní klientovi získat již zmiňované přístupové tokeny. Rozlišujeme dva typy těchto klíčů:

- **Access token** - Tento token se používá při přístupu k chráněnému zdroji. Server poskytující zdroje na jeho základě provede autorizaci a zpřístupnění požadovaných informací. Přístupový řetězec reprezentuje abstraktní vrstvu zapouzdřující ověřovací prvky (např. jméno a heslo). Access token je generován pomocí kryptografických metod a jeho rozsah platnosti určuje vydávající autorizační server. Token se v průběhu komunikace předává pomocí části *Authorization: Bearer* v HTTP hlavičce nebo tvoří přímou součást URI adresy. [17] Životnost tokenu bývá omezena na kratší časový úsek, po jehož vypršení je nutné vygenerovat nový platný kód. [13]

- **Refresh token** - Refresh token je z hlediska časového omezení oproti access tokenu dlouhodobý. Jeho účelem je získání nového platného access tokenu. Na rozdíl od access tokenu může refresh token komunikovat pouze s autorizačním serverem. Na výzvu klienta provede autorizační server ověření platnosti refresh tokenu. Dále zneplatní původní access token a vygeneruje nový. Volitelně může být zneplatněn a nově vygenerován i refresh token. Zamezení možnosti získávat nové access tokeny pomocí refresh tokenu je možné odebráním autorizačního povolení klientovi. [13]

Vzájemné použití tokenů je znázorněno obrázkem č. 3.2.



Obrázek 3.2: Životní cyklus OAuth tokenů. (Zdroj: [13])

3.3 Formáty pro reprezentaci zdrojů

Při komunikaci s REST API nezachází klient přímo se zdrojem, ale pracuje pouze s jeho reprezentací. Pro reprezentaci zdrojů slouží různé datové formáty, které umožňují jejich přenos. Server vlastnící zdroje většinou umožňuje získat jeden zdroj v několika různých formátech reprezentace. Klient si může z nabízeného seznamu vybrat ten, který je schopen zpracovat. Setkat se můžeme s různými typy formátů. Omezení na jejich použití není nijak přesně specifikováno.

Pro přenos dat se dnes nejčastěji používají serializační formáty JSON (*JavaScript Object Notation*) a XML (*eXtensible Markup Language*).

3.3.1 JSON

JavaScript Oject Notation je datový formát vycházející z jazyku JavaScript. Pro svoji jednoduchost a efektivitu se z něj stal multiplatformní standard, který je možné použít pro zápis a přenos dat napříč různými programovacími jazyky.

JSON byl navržen tak, aby byl dobře čitelný a přehledný pro člověka a zároveň snadno zpracovatelný strojově. Pro vytváření JSON schémat lze využít dva základní stavební prvky:

- *kolekce* - Kolekci tvoří posloupnost předem neomezeného počtu hodnot stejného datového typu.
- *struktura* - Struktura je uspořádaná n-tice s pevným počtem prvků typu jméno/hodnota obecně různých datových typů.

Vzájemným zanořováním lze vytvářet propracované složitější struktury, jako jsou objekty obsažené v jiných objektech, asociativní pole, aj. Takto vzniklá schémata mohou být složena z libovolných podporovaných datových typů. Nelze však vytvořit cyklickou závislost mezi objekty. JSON také není určen pro přenos dat v prosté binární podobě.

JSON podporuje následující datové typy: [5]

- *objekt* - Neuspořádaná množina párů název/hodnota.
- *pole* - Seřazená posloupnost hodnot.
- *číslo* - Celá a desetinná čísla. Desetinná čísla lze vyjádřit pomocí exponentu. Není však umožněn zápis nekonečna.
- *textový řetězec* - Nula nebo více znaků s podporou kódování Unicode.
- *logická hodnota* - Vyjádření logické hodnoty true/false.
- *prázdný typ* - Speciální datový typ null.

Příklad zápisu jednotlivých datových typů v notaci JSON je reprezentován výpisem č. 3.1.

```
{
  "objekt": {
    "atribut_a": "Normální text.",
    "atribut_b": {
      "vnorený_text_a": "Text s Unicode znakem: \u002f.",
      "vnorený_text_b": "Test s escape sekvencí: \n"
    },
    "prazdný_typ": null,
    "pole": [-1.1,2.2, 3e-3],
    "index": 10245
  },
  "log_hodnoty": [{
      "hodnota_c": true,
      "hodnota_d": false
    },
    {
      "hodnota_c": false,
      "hodnota_d": true
    }
  ]
}
```

Výpis 3.1: Struktura formátu JSON.

Formát JSON se v posledních letech těší velké oblibě při používání REST API. Syntaxe jazyka je podstatně jednodušší a efektivnější než formát XML. JSON na rozdíl od XML nemá standardem definovanou možnost pro vytváření validačních schémat, proto vzniklo neoficiální rozšíření JSON Schema, které tuto možnost nabízí. [26]

3.3.2 XML

Extensible markup language je otevřený značkovací jazyk vyvinutý a standardizovaný konsorciem W3C. XML je podobně jako JSON navrženo tak, aby jeho struktura byla čitelná i pro člověka. XML se více zaměřuje na obecně širší možnosti použití a poskytuje rozsáhlou podporu pro definování validačních schémat. Pomocí těchto definicí lze vytvářet další

standardizované formáty. Jazyk XML je vysoce rozšířený a jeho zpracování je podobně jako JSON možně ve většině programovacích jazyků. Oba formáty jsou mezi sebou navzájem převoditelné.

XML nebylo prvotně navrženo pouze za účelem serializovaného odeslání dat. [25] Jazyk je tak mnohem komplexnější. Struktura XML je tvořena pomocí značek (angl. *tag*), jejich hodnot a atributů. Vzájemným zanořováním značek lze vytvářet libovolné datové struktury. Nesmí však docházet k vzájemnému překryvu. Posloupnost elementů lze definovat pomocí *Document Type Definition* (DTD). Samotné značky nemají přímo specifikovaný datový typ, ten je možné definovat pomocí rozšíření *XML Schema definition* (XSD).

Efektivita XML je oproti JSON závislá na složitosti struktury a velikosti obsahu souboru. Výhodu při zpracování představuje možnost adresovat a odkazovat jednotlivé značky. Pro rychlé prohledání dokumentů vznikl jazyk XPath (*XML Path Language*).

Struktura XML se skládá z následujících částí: [24]

- *deklarace* - Deklarace dokumentu specifikuje použitou verzi XML a kódování dokumentu.
- *komentář* - XML umožňuje vkládat do své struktury komentáře. Poznámky se uzavírají mezi párové značky <!– –>.
- *element* - Element se skládá z dvojice otevírající *<element>* a ukončující *</element>* značky. Výjimku tvoří pouze prázdný element *<element />*. Název značky (elementu) je citlivý na velikost písmen. Mezi párem značek se nachází obsah elementu. Tato hodnota může obsahovat i další zanořené podelementy.
- *atribut* - Atributy slouží jako volitelné doplňující hodnoty, které lze ve formátu *atribut="hodnota"* připsat k otevírající značce elementu.

Ukázka základní struktury XML dokumentu je formulována ve výpisu č. 3.2.

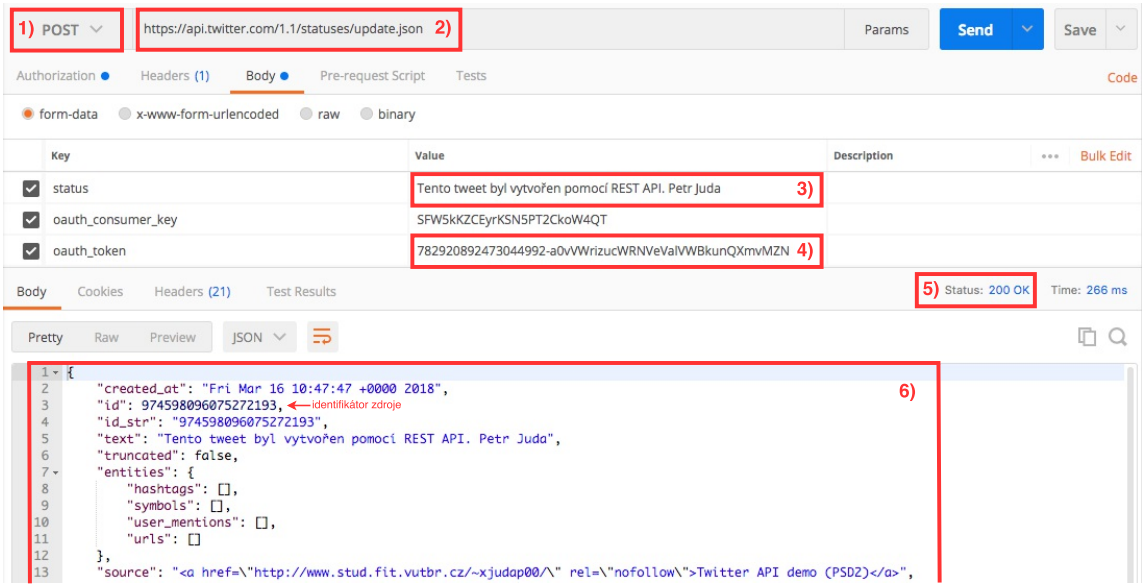
```
<?xml version="1.0" encoding="UTF-8"?>
<kořen>
  <jméno atr="atribut">Petr Juda</jméno>
  <!-- Adresa -->
  <adresa>
    <ulice>Božetěchova 1/2</ulice>
    <město>Brno</město>
    <PSČ>612 00</PSČ>
    <země>Česká republika</země>
  </adresa>
  <telefon>
    <číslo>123456789</číslo>
  </telefon>
  <telefon>
    <číslo>987654321</číslo>
  </telefon>
  <prázdný_element atr1="1" atr2="dva"/>
</kořen>
```

Výpis 3.2: Struktura formátu XML.

3.4 Příklad komunikace s REST API

Pro testování komunikace s REST API lze použít např. aplikaci Postman. Obrázek č. 3.3 pochází z tohoto programu. Snímek zobrazuje propojení jednotlivých částí, které byly popsány v předchozích sekcích této kapitoly. Za účelem demonstrace bylo využito veřejné REST API rozhraní sociální sítě Twitter.

Obrázek č. 3.3 zachycuje složení požadavku a odpovědi na vytvoření nového příspěvku (cílový zdroj) na sociální síti. Jako způsob ověření byla použita autentizace OAuth.



Obrázek 3.3: Vytvoření příspěvku (zdroje) pomocí REST API. (Zdroj: vlastní)

Nejdůležitější části požadavku jsou na obrázku zvýrazněny červenou barvou:

1) Výběr HTTP metody POST pro vytvoření zdroje.
2) Předem domluvená URI adresa (*endpoint*) pro nahrávání příspěvků. [23]
3) Text nahrávané zprávy.
4) OAuth access token.
5) Stavový kód odpovědi.
6) Obsah odpovědi ve formátu JSON.

Kapitola 4

Specifikace požadavků na systém

Aplikace je tvořena pro společnost Platební instituce Roger a.s. Sloužit má jako interní firemní nástroj, který bude operovat pouze s účty vlastněnými společností Roger a.s. Systém prakticky ověří využití výhod plynoucí z nových služeb popsaných v sekci 2.2. Platební instituce je pod dozorem České národní banky a bude žádat o licenci na využívání PSD2 služeb. Firemní portfólio se specializuje na malé a střední podniky, kterým nabízí možnost financování pohledávek prostřednictvím investičních aukcí. Tyto aukce se snaží nalézt vhodného investora nabízejícího nejlepší podmínky. Z konzultací s panem Ing. Tomášem Slobodníkem (CTO - *Chief executive officer*) společnosti Roger a.s. vznikly následující systémové požadavky.

4.1 Obecné požadavky

Tato část popisuje požadavky vztahující se k obecné funkcionalitě programu. Dále je zde popsáno, jakým způsobem bude se systémem zacházeno.

4.1.1 Architektura systému

Systém bude plně samostatně funkční a bude možné jej spustit i bez závislosti na ostatním programovém vybavení firmy. Musí však být zajištěna i možnost propojení nového modulu se stávajícími programy skrz používaný firemní systém. Bude umožněno, aby aplikace spolupracovala s více různými firemními moduly. Zároveň bude možné odlišit, kdo a kdy zadal příslušný příkaz k platbě. Tato historie bude po určitou dobu ukládána v lokální databázi.

4.1.2 Požadované operace

Systém prostřednictvím svého rozhraní umožní vytvořit tři typy plateb:

- *domácí platba* - (prostředí České republiky)
- *europlatba* - (oblast eurozóny)
- *zahraniční platba* - (v rámci celého světa)

Proces vytvoření platby bude schopný ověřit, zda jsou zadány všechny potřebné údaje. Dále ověří, že všechny vložené hodnoty odpovídají předepsanému tvaru, který je definován bankou. Operace zadávání plateb bude neblokující. Vytvořené platby budou automatizovaně rozdělovány do dávek a nahrány do banky.

Aplikace dále z předem specifikovaných bankovních účtů zpřístupní transakční historii. Účetní historie bude periodicky stahována a doplňována do lokální databáze. Zde bude po stanovenou dobu uchovávána. Informace o jednotlivých transakcích budou ze systému předávány v unifikovaném tvaru.

4.1.3 Aplikační rozhraní

Interakce se systémem bude probíhat pomocí rozhraní hlavní programové třídy. To umožní provádět úkony definované v předchozí části 4.1.2. Prostřednictvím těchto operací bude v ostatních modulech možné vytvářet nové případy užití, které povedou k rozsáhlé automatizaci.

Rozhraní bude napojeno na bankovní systém Fio banky. Návrh aplikace bude založen na studii více současných existujících bankovních řešení. (Dále popsáno v kapitole č. 5.)

Program musí být dobře škálovatelný, aby v budoucnu nedocházelo k nutnosti zásadně měnit stanovené rozhraní při rozšiřování systému o podporu dalších bankovních API. Vstupní a výstupní data budou předávána v unifikovaném tvaru. Zpracování výstupů bude probíhat mimo vyvíjený systém.

4.2 Technologické požadavky

Níže jsou specifikovány požadavky na technologii tak, aby bylo zaručeno bezproblémové začlenění nového nástroje do stávajícího firemního systému.

4.2.1 Programovací jazyk a rámec

Dosavadní firemní systém je vytvořen v programovacím jazyce **Python** ve verzi 2.7 s využitím aplikačního webového rámec *web2py*. Architektura rámce je popsána v sekci č. 6.1. Ačkoliv nový bankovní modul nebude disponovat vlastním webovým uživatelským rozhraním, je z důvodu zajištění bezproblémové integrace použití *web2py* nezbytné.

Celý výsledný systém bude provozován na linuxové platformě CentOS.

4.2.2 Databázový systém

Společnost využívá objektově-relační databázový systém **PostgresSQL** ve verzi 9, často zkráceně jako *Postgres*. S tímto systémem bude aplikace testována. Při vývoji systému bude použita abstraktní databázová vrstva, kterou rámec *web2py* disponuje. Tato vrstva je blíže popsána v sekci č. 6.1.2. Využití tohoto nástroje umožní systém v případě potřeby připojit i k ostatním podporovaným databázovým systémům.

Kapitola 5

Přehled dostupných bankovních API

Tato kapitola se zaměřuje na rozbor a porovnání doposud zveřejněných bankovních rozhraní, která umožňují provádět nově definované operace. Tyto operace jsou popsány dříve v sekci č. 2.2. Průzkum se věnuje aktivně působícím bankám na území České republiky. V aktuální době jsou veřejně dostupná řešení od Fio banky, České spořitelny a Banky Creditas. Vzhledem k tomu, že API Fio banky bude použito při implementaci programu, je zde zkoumáno podrobněji než ostatní. Lze očekávat, že se v blízké době objeví řešení dalších bankovních institucí. Jednotlivá níže popisovaná API jsou postavena na architektuře REST, které se věnuje kapitola č. 3.

5.1 Fio banka

Fio banka, a.s. nabízí své rozhraní na trhu nejdéle. Služba se jmenuje Fio API bankovnictví a v provozu je již od roku 2013. Dostupná je zdarma, jak pro firemní zákazníky, tak i jako doplněk k osobním účtům. Prostřednictvím AP,I v kombinaci s internetovým bankovnictvím, je možné provádět hromadné importování různých typů plateb nebo export historie bankovních transakcí či výpisů z účtu.

V následujících podsekcích jsou detailněji popsány principy fungování systému. Dále také podporované požadavky a formáty. Informace v této sekci pocházejí převážně z programové dokumentace rozhraní. [12]

5.1.1 Dostupné služby

Fio banka umožňuje pomocí svého API provádět následující kategorie operací.
Získávání dat z účtu:

- Stažení pohybů z účtu za dané období.
- Stažení pohybů na účtu od posledního stažení.
- Stažení oficiálního výpisu z účtu.
- Stažení karetních transakcí na platebních terminálech/platební bráně za dané období.

Nastavení zarážky, která slouží jako výchozí bod pro metodu *stažení pohybu od posledního stažení*. Pro nastavení zarážky je možné použít dvě hodnoty:

- Datum posledního neúspěšného stažení.
- ID posledního staženého pokynu.

Podávání platebních příkazů:

- Platba a inkaso v rámci ČR
- Europlatba
- Zahraniční platba
- SEPA platba a SEPA inkaso

Pro získání informací z účtu a nastavení zarážky se používá metoda GET. Nahrávání dat je realizováno pomocí metody POST. Každá operace má přesně definované složení URI adresy a formáty, ve kterých je zdroje možné odesílat či přijímat. V podsekcích č. 5.1.3 a 5.1.4 se více zaměříme na podstatné operace, které vyplývají z předchozí definice požadavků na systém (podsekce č. 4.1.2). Podrobný popis všech metod je k dispozici v programové dokumentaci.

5.1.2 Autentizace

Pro potřeby ověření uživatele se využívá autentizace pomocí přístupového tokenu. Metoda byla vysvětlena v podsekci č. 3.2.3. Fio banka rozděluje tokeny do dvou kategorií oprávnění:

- **Sledování účtů** - Typ tokenu umožňující pouze exportování dat z banky.
- **Sledování účtů a zadávání platebních a inkasních příkazů** - Token pro získávání informací z účtu, zároveň umožňující i import platebních příkazů do banky.

Každý token je vázán pouze k jednomu konkrétnímu účtu. Pro jeden účet lze vygenerovat více různých klíčů. Příslušný přístupový token je možné získat po přihlášení do internetového bankovnictví a splnění následujících kroků:

1) Přihlášení oprávněné osoby do internetového bankovnictví.
2) Vytvoření žádosti o zřízení požadovaného typu tokenu. Žádost musí být standardně autorizována. Pokud má příslušný účet nastavenou autorizaci více osobami, musí požadavek podepsat všechny odpovědné autority.
3) Po pěti minutách od úspěšné autorizace lze token začít využívat. Časová platnost tokenu je nastavena při podání žádosti.

Jeden token je možné použít pouze jedenkrát během 30 sekund. Na využití konkrétního tokenu před uplynutím definovaného časového intervalu reaguje systém chybou *409 Conflict.*

Veškerá komunikace mezi bankou a cílovou stanicí probíhá pomocí protokolu SSL s minimálně 128bitovým šifrováním.

5.1.3 Způsob iniciace platby

Platební příkazy se do banky odesílají skrze jednotnou ustanovenou URL adresu:

`https://www.fio.cz/ib_api/rest/import/`

Po úspěšném nahrání dat do banky se příkazy sdruží do dávky, která musí být dodatečně autorizována oprávněnou osobou v internetovém bankovnictví. Bez dodatečného ověření (SMS kód, Fio podpis) nebudou příkazy finálně zpracovány.

Pro úspěšné odeslání musí požadavek obsahovat platný token, volbu formátu odesílaného souboru, soubor s příkazy a volitelně požadovaný jazyk odpovědi. Platební příkazy jsou odesílány v souborech. Tyto soubory mají pevně definovanou strukturu a formát. Pro nahrání pouze domácích plateb v české měně je k dispozici formát ABO. ABO je datový formát používaný v České republice a na Slovensku pro výměnu finančních zpráv. Pokročilejší možnosti nahrávání plateb umožňuje formát Fio XML. Struktura Fio XML je přesně definována pomocí XSD schématu a připouští v jednom souboru nahrát více typů platebních pokynů. Tento formát plně splňuje požadavky, které jsou kladeny na vyvíjený systém, proto je zde rozebrán podrobněji.

Tabulka č. 5.1 zachycuje parametry Fio XML, které lze nastavit u domácí platby. Tabulky č. 5.2 a 5.3 porovnávají rozdíly mezi platbou v eurozóně a mezinárodní transakcí. Z vyobrazených tabulek vyplývá, že každý typ platby má ve Fio XML definovanou vlastní strukturu a vyžaduje rozdílnou specifikaci parametrů.

Element	**Povinný**	**Popis**
accountFrom	ano	číslo účtu odesílatele
currency	ano	měna
amount	ano	částka příkazu
accountTo	ano	číslo účtu příjemce
bankCode	ano	banka příjemce
ks	ne	konstantní symbol
vs	ne	variabilní symbol
ss	ne	specifický symbol
date	ano	datum splatnosti
messageForRecipient	ne	zpráva pro příjemce
comment	ne	označení pro odesílatele
paymentReason	ne	platební titul
paymentType	ne	Typ platby (inkaso, standardní/zrychlená platba)

Tabulka 5.1: Elementy domácí platby u Fio banky. (Zdroj: [12])

Zásadní podmínku tvoří seřazení jednotlivých pokynů v souboru, které musí být v tomto pořadí: **tuzemské platby, europlatby, zahraniční platby**. Kompletní validační XSD schéma, včetně specifikace datových typů, je dostupné v programové dokumentaci. [12]

Element	Povinný	Popis
accountFrom	ano	číslo účtu odesílatele
currency	ano	měna
amount	ano	částka příkazu
accountTo	ano	mezinárodní číslo bankovního účtu (IBAN)
ks	ne	konstantní symbol
vs	ne	variabilní symbol
ss	ne	specifický symbol
bic	ne	mezinárodní bankovní identifikace (SWIFT kód)
date	ano	datum splatnosti
comment	ne	označení pro odesílatele
benefName	ne	majitel účtu příjemce
benefStreet	ne	ulice bydliště majitele účtu příjemce
benefCity	ne	město bydliště majitele účtu příjemce
benefCountry	ne	země bydliště majitele účtu příjemce
remittanceInfo1	ne	informace pro příjemce 1
remittanceInfo2	ne	informace pro příjemce 2
remittanceInfo3	ne	informace pro příjemce 3
paymentReason	někdy	platební titul
paymentType	ne	Typ platby (inkaso, standardní/zrychlená platba)

Tabulka 5.2: Elementy euro platby u Fio banky (Zdroj: [12])

Element	Povinný	Popis
accountFrom	ano	číslo účtu odesílatele
currency	ano	měna
amount	ano	částka příkazu
accountTo	ano	mezinárodní číslo bankovního účtu (IBAN)
bic	ano	mezinárodní bankovní identifikace (SWIFT kód)
date	ano	datum splatnosti
comment	ne	označení pro odesílatele
benefName	ano	majitel účtu příjemce
benefStreet	ano	ulice bydliště majitele účtu příjemce
benefCity	ano	město bydliště majitele účtu příjemce
benefCountry	ano	země bydliště majitele účtu příjemce
remittanceInfo1	ano	informace pro příjemce 1
remittanceInfo2	ne	informace pro příjemce 2
remittanceInfo3	ne	informace pro příjemce 3
remittanceInfo4	ne	informace pro příjemce 4
detailsOfCharges	ano	nastavení poplatků za platbu (vše příjemce/odesílající, každý své)
paymentReason	ano	platební titul

Tabulka 5.3: Elementy zahraniční platby u Fio banky (Zdroj: [12])

5.1.4 Čtení bankovní historie

Pohyby na účtu lze získat v několika různých formátech. Dostupné jsou Fio XML, JSON, GPC, OFX, CSV a HTML. Obsah přístupných informací je však velmi podobný. Schéma je rozděleno do dvou částí. První část obsahuje informace o účtu, jako jsou bankovní identifikace, počáteční a koncový finanční zůstatek, atd. Druhá část pak obsahuje informace o jednotlivých provedených transakcích. Čtení plateb má na rozdíl od zadávání pevně definovanou strukturu získatelných položek. Struktura dostupných atributů je vložena do tabulky č. 5.4. Pokud transakce některé atributy neobsahuje, jsou ponechány prázdné.

Atribut	**Popis**	**Atribut**	**Popis**
ID pohybu	Jedinečný identifikátor pohybu.	SS	Specifický symbol.
Datum	Datum zpracování pohybu.	Uživatelská identifikace	Označení platby.
Objem	Částka příchozí/odeslané platby.	Zpráva pro příjemce	Zadaná zpráva pro příjemce.
Měna	Měna příchozí/odeslané platby.	Typ	Typ provedené operace.
Protiúčet	Číslo protiúčtu.	Provedl	Odpovědná osoba, která zadala příkaz.
Název protiúčtu	Název protiúčtu.	Upřesnění	Specifikace transakce.
Kód banky	Číslo banky protiúčtu.	Komentář	Upřesňující informace.
Název banky	Název banky protiúčtu.	BIC	Mezinárodní bankovní identifikace. (SWIFT kód)
KS	Konstantní symbol.	ID pokynu	Číslo bankovního příkazu.
VS	Variabilní symbol.		

Tabulka 5.4: Atributy platebních pokynů u Fio banky (Zdroj: [12])

Pro potřeby projektu jsou podstatné následující URL adresy. Získání pohybů na účtě za určené období:

`{base_url} + periods/{token}/{datum_od}/{datum_do}/transactions.{format}`

Získání pohybů na účtě od posledního stažení:

`{base_url} + last/{token}/transactions.{format}`

Nastavení zarážky na poslední úspěšné stažené ID pohybu: (Tato možnost se využívá v případě, že dojde k chybě zpracování dat na straně klienta.)

`{base_url} + set-last-id/{token}/{id}/`

Kde `{base_url}` je `https://www.fio.cz/ib_api/rest/`, `{token}` je vygenerovaný přístupový klíč, `{datum_od}` - počáteční datum požadované transakční historie, `{datum_do}` - koncové datum požadované historie, `{format}` - formát požadovaných dat a `{id}` je ID pohybu.

5.2 Česká spořitelna

Česká spořitelna, a.s. je největší bankou působící na českém trhu. Banka je součástí skupiny Erste Group, která operuje ve střední a východní Evropě. Česká spořitelna nabízí doposud nejpropracovanější možnosti práce s bankovním API. Řešení obsahuje několik nezávislých REST API, která zpřístupňují mnoho služeb vysoko nad rámec direktivy PSD2.

Třetím stranám a veřejnosti je k dispozici rozsáhlý vývojářský portál Erste API Hub. [6] Ten obsahuje kompletní dokumentaci všech přístupných služeb České spořitelny. Později se zde budou objevovat i API dalších bank skupiny Erste. Stránky obsahují také testovací prostředí (*sandbox*), kde je možné si jednotlivé operace bezpečně vyzkoušet a testovat. Řešení České spořitelny je stále ve stavu aktivního vývoje, proto nyní nelze garantovat finální podobu řešení. Zdrojem informací pro tuto sekci je vývojářský portál Erste hub. [6]

5.2.1 Dostupné služby

Banka nabízí široké spektrum operací a požadavků. Jednotlivá API odpovídají portfóliu služeb, které instituce poskytuje svým klientům. Rozdělena jsou do následujících skupin:

- *AISP API* - Služba informování o platebním účtu. (Definováno PSD2.)
- *PISP API* - Služba platební iniciace. (Definováno PSD2.)
- *Corporate API* - Detailní informace o účtech korporátních a firemních klientů.
- *Personal Accounts API* - Detailní informace o osobních účtech klientů.
- *Transparent Accounts* - Informace o transparentních účtech.
- *Exchange Rates* - Získání aktuální i historické hodnoty kurzovních lístků.
- *Mortgage calculator API* - Kalkulace splátek hypotéky dle zadaných parametrů.
- *Know-your-customer API* - Ověření osobních údajů na základě souhlasu klienta.
- *Places* - Veškeré informace o pobočkách a bankomatech České spořitelny.

Zpřístupnění všech zdrojů je k dispozici pouze ve formátu JSON.

5.2.2 Autentizace

Způsob provedení autentizace a autorizace je u České spořitelny značně složitější než v případě Fio banky. Třetí strana musí svou aplikaci zaregistrovat na vývojářském portálu Erste Hub. Následně aplikaci propojit s bankou České spořitelny a požadovanými API. Aplikace od banky získá unikátní **API token**, který musí být vložen do hlavičky všech požadavků. API token neslouží jako zabezpečovací prvek, ale pouze jako identifikace aplikace na straně bankovního serveru. Po splnění předchozích kroků je aplikaci udělen přístup do testovacího prostředí *sandbox*. Pokud chce třetí strana přistupovat k požadavkům, které vyžadují autorizaci od bankovního klienta, je nutné získat jeho souhlas prostřednictvím protokolu OAuth2. Přístup lze získat na základě žádosti typu `Implicit` nebo `Authorization Code`, který je zvolen při registraci aplikace. Popis fungování rámce OAuth2 byl vysvětlen v podsekci č. 3.2.3.

Po dokončení vývoje a testování aplikace následuje předání do schvalovacího procesu. Zde zaměstnanci banky podrobí aplikaci bezpečnostní prohlídce. Pokud jsou podepsány všechny potřebné právní dokumenty a aplikace je bankou schválena, získá třetí strana přístupové údaje do produkčního prostředí.

5.2.3 Způsob iniciace platby

Iniciace plateb probíhá pomocí *PISP API*. Implementace je postavena na Českém standardu pro Open Banking (COBS). Tento dokument byl vydán Českou bankovní asociací a představuje rozsáhlá praktická doporučení pro implementaci řešení jednotlivých bank. Použití standardu je dobrovolné a jednotlivé implementace se od něj mohou lehce odchylovat. Cílem je stanovit jednotnou strukturu API různých bank, která by umožnila jednodušší integraci a snížení nároků na programové vybavení třetích stran. [28]

Vytvoření platby má na starosti požadavek:

```
POST {adresa_prostředí} + webapi/api/v1/payment-initiation/my/payments
```

Tento prostředek slouží k vytvoření různých druhů plateb. Přípustné jsou domácí platba s minimálním obsahem parametrů, plná domácí platba, SEPA platba, plná SEPA platba, platba uvnitř Evropské unie a platba mimo EU. Prostřednictvím API lze získávat informace o stavu vytvořených transakcí, případně mazat neautorizované transakce.

Atributy jednotlivých plateb jsou členěny detailněji než v případě Fio banky. Nejpodstatnější zadávané informace jsou však téměř stejné. Přesné schéma požadavku, včetně potřebných hlaviček a parametrů, je možné nalézt a vyzkoušet v online dokumentaci Erste Hub. [6]

5.2.4 Čtení bankovní historie

Základní informace o účtu, jako jsou bankovní zůstatek nebo historie bankovních transakcí, lze získat přes *AISP API*. Toto API vychází rovněž z Českého standardu pro Open Banking.

Podrobnější informace, jako detaily o pojištění a spoření nebo manipulaci s platebními kartami, jsou dostupné prostřednictvím Personal Accounts API a Corporate API.

Transakční historii lze získat pomocí požadavku:

```
GET {adresa_prostředí} +
          webapi/api/v1/account-information/my/accounts/{id}/transactions
```

Kde `id` je unikátní proměnný identifikátor účtu, získaný z odpovědi na dotaz o detailech účtu. Transakce lze pomocí nastavení rozšiřujících parametrů dotazu třídit a řadit. Schéma, obsahující popis dostupných atributů pro jednotlivé typy plateb, je dostupné v příslušné dokumentaci. [6]

5.3 Banka Creditas

Banka Creditas, a.s. vstoupila na český bankovní trh v roce 2017. Své potenciální klienty láká především na zajímavé možnosti úročení vkladů a nyní i na plně funkční Creditas API. Creditas API je rozděleno do dvou skupin. Obecná kategorie obsahuje služby, zpřístupněné na základě nařízení direktivy PSD2. Kategorie premium pak obsahuje požadavky vytvořené nad rámec uložených povinností bance. Obě tyto kategorie jsou zdarma dostupné všem klientům banky a třetím stranám, které s bankou spolupracují na vývoji nových služeb. [1]

5.3.1 Dostupné služby

Prostřednictví API lze provádět následující operace: [3]

- Získávání informací o účtu. (běžný účet, spořící účet, termínované vklady)
- Export výpisů z účtu i po jednotlivých transakcích.
- Zadání samostatné platby. (domácí platba, mezinárodní platba, SEPA platba)
- Hromadný import plateb pomocí souboru.
- Vyvolání procesu autorizace pomocí SMS PIN.
- Registrace klientské aplikace pro OAuth.

Všechny požadavky jsou dostupné pomocí metody POST a přenášeny ve formátu JSON.

5.3.2 Autentizace

Ověření uživatele Banky Creditas kombinuje doposud představená řešení Fio banky a České spořitelny. Přístup je koordinován pomocí OAuth access tokenu. Ten je možné získat pomocí dvou způsobů. První možností je vygenerování přístupového klíče pomocí grafického rozhraní internetového bankovnictví. Druhá možnost je vytvořit bezpečnostní klíč skrz aplikaci třetí strany, která je přímo připojena k OAuth autentizačnímu serveru banky. Obě metody vyústí k vygenerování **access tokenu** a **accountId** (jednoznačný identifikátor účtu). Tyto informace jsou nutnou součástí každého odeslaného požadavku. Metody generování přístupových tokenů však nejsou ekvivalentní. [1] [2]

Přístupové tokeny jsou rozděleny do tří skupin dle oprávnění: [1]

- Oprávnění ke čtení. (Přístup k informacím o účtu, zůstatku a transakcím.)
- Oprávnění k zadávání plateb. (Neumožňuje autorizovat platbu.)
- Oprávnění k realizaci platby.

Přístupovým tokenům, které jsou vytvořeny pomocí internetového bankovnictví, nelze přiřadit oprávnění pro realizaci plateb. Další podstatný rozdíl představuje rozsah použitelnosti tokenu. Token vytvořený v internetovém bankovnictví je napevno svázán s jedním účtem a stanoveným oprávněním, přičemž token vygenerovaný přímo pomocí protokolu OAuth, může poskytovat přístup k více účtům majitele a zastávat i více typů oprávnění. [1] [2]

Pro zajištění bezpečnosti proti zneužití přístupu třetí stranou, je v určitých případech realizace platby vyvolán proces autorizace uživatele. Tento proces je realizován prostředky API a vyžaduje získání autorizačního kódu od klienta banky pomocí bezpečnostního prvku, který zákazník využívá v internetovém bankovnictví. (např. SMS kód) [1]

5.3.3 Způsob iniciace platby

Do bankovního systému je možné nahrát samostatnou platbu prostřednictvím odpovídajícího typu požadavku nebo celý soubor s připravenou transakční dávkou. Podle typu zvoleného tokenu lze platbu následně autorizovat v internetovém bankovnictví nebo pomocí OAuth.

Při zpracování transakční dávky jsou podobně jako u Fio banky k dispozici dva druhy souborů. Formát ABO nebo CSV slouží pro definici pouze domácích plateb. K vytvoření mezinárodních transakcí je nutné použít speciální bankovní standard XML. Tento formát vychází z normy ISO 20022. Všechny nachystané soubory jsou API předávány v kódování Base64. Další podrobné informace, včetně specifikace formátů, jsou dostupné v technické dokumentaci Creditas OpenAPI. [3]

5.3.4 Čtení bankovní historie

Princip čtení informací z účtu je velmi podobný, jako v případě Fio banky nebo České spořitelny. Rozdíly představují pouze použité formáty a jejich struktura. Transakční historii je možné získat ve formátu CSV nebo XML. Struktura XML téměř odpovídá formátu, který se u Creditas používá pro nahrávání mezinárodních plateb. Schéma je opět dostupné v technické dokumentaci Creaditas OpenAPI. [3]

5.4 Zhodnocení stavu

Z porovnání představených řešení je patrné, že jednotlivá bankovní API se od sebe v mnoha ohledech zásadně odlišují. Rozdíly panují v nabízených službách i technických detailech řešení.

Nejrozsáhlejší okruh služeb nabídne Česká spořitelna, která třetím stranám umožní přístup téměř ke všem svým produktům. Oproti tomu řešení Fio banky podporuje pouze služby definované v PSD2. Se zajímavou myšlenkou přišla Banka Creditas. Ta nejen, že zveřejnila své rozhraní, ale současně se stala i třetí stranou a svým klientům zprostředkovává možnost ovládat účty vedené u Fio banky prostřednictvím internetového bankovnictví Creditas IB. [4] Bude zajímavé sledovat, jak na tuto situaci zareagují ostatní bankovní subjekty v ČR.

Zásadní technické rozdíly mezi bankami jsou ve způsobu provedení autentizace uživatele a možnostech, jaká oprávnění lze třetím stranám přidělit. Situaci dále komplikuje používání různých formátů pro reprezentaci bankovních transakcí u jednotlivých bank, přestože dané bankovní operace mají standardem definovaná schémata, včetně specifikace povinných a nepovinných atributů. Odlišně postavená hierarchie požadavků značně rozšiřuje složitost návrhu řešení aplikace třetí strany. Náznak zlepšení tohoto problému lze pozorovat u České spořitelny, jejíž řešení vychází z Českého standardu pro Open Banking. Tento jednotný standard by mohl být využit i u dalších bank, které svá řešení teprve připravují.

Kapitola 6

Návrh řešení systému

Návrh řešení systému vychází z kapitoly č. 5 o dostupných bankovních API. Dále respektuje požadavky na systém, které jsou definovány v kapitole č. 4.

Sekce č. 6.1 vysvětluje složení a podstatné funkce aplikačního rámce web2py. S ohledem na architekturu rámce jsou navrženy další části programu, které je nutné implementovat. V sekci č. 6.2 je popsán návrh relačního modelu databáze. Sekce č. 6.3 se věnuje dekompozici programu na skupiny podproblémů a jejich následné rozdělení do třídní hierarchie. Poslední sekce kapitoly 6.4 popisuje, jakým způsobem bude aplikace testována.

6.1 Aplikační rámec web2py

Web2py je aplikační rámec vytvořený v jazyce Python 2.7, který umožňuje jednoduše vytvářet webové aplikace. Projekt původně vznikl na akademické půdě pro potřeby výuky. Zásluhou otevřené licence *(LGPL3)* a pokročilých funkcí se z něj stal oblíbený prostředek využívaný i v korporátním prostředí. Zakládajícím autorem byl Massimo Di Pierro. Z jeho knihy *web2py Complete Reference Manual* [16] pochází převážná část informací obsažených v této sekci.

Inspirací pro vytvoření web2py byla snaha zkombinovat nejlepší vlastnosti Python rámce *Django* a rámce *Ruby on Rails*, používaného v programovacím jazyce Ruby. Web2py soustředí svoji pozornost na tři hlavní pilíře:

- **Jednoduchost použití** - Cílem je snížit nároky na vývoj i nasazení projektu, proto je web2py rámec koncipován jako *full-stack*. To znamená, že jeho spuštění nevyžaduje žádnou instalaci závislostí nebo složitou konfiguraci. Rámec si sám umí poradit s nastavením většiny webových serverů, databázových serverů a Python knihoven. Web2py je multiplatformní řešení spustitelné na operačních systémech Windows, Mac OS X, iOS a Unix/Linux.

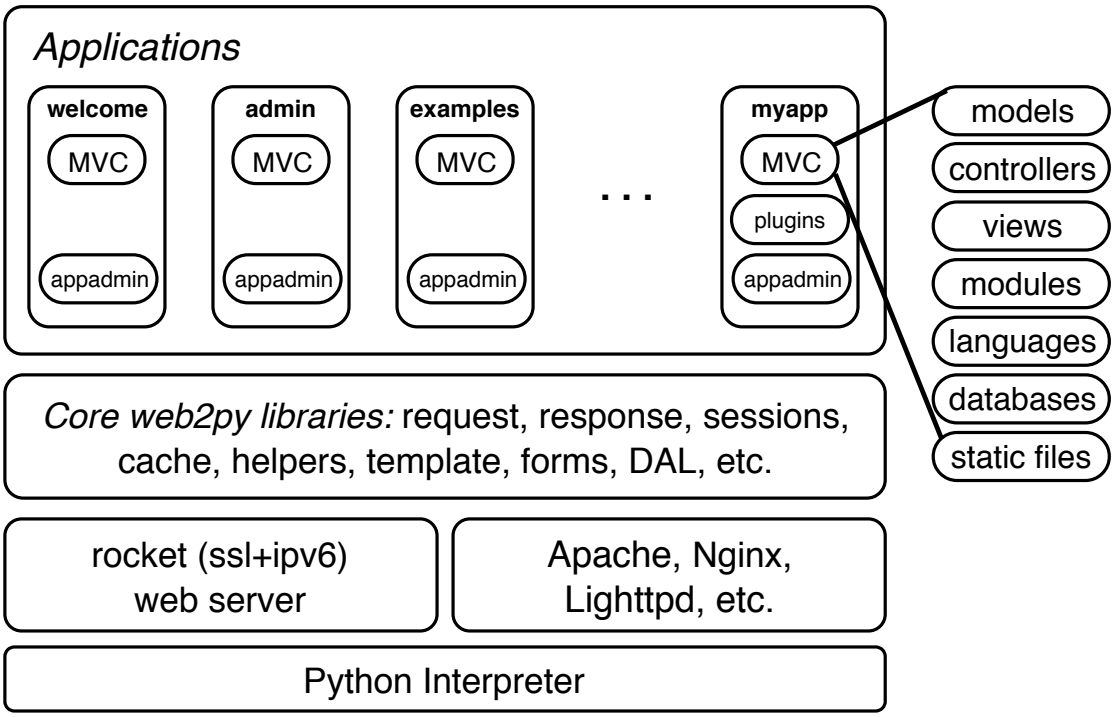
- **Rychlost vývoje** - Vývoj nové aplikace začíná vygenerováním a následnou úpravou základní šablony. Ta obsahuje již zcela přichystaný systém na zpracování požadavků. Každá aplikace má k dispozici webový administrační panel, kterým lze provádět úpravy kódu za běhu. Předem připravené funkce mají jasně stanovené chování a snaží se maximálně zvýšit intuitivnost a čitelnost kódu. Tyto funkce umí automaticky generovat formuláře nebo tabulky z předložených dat. Pro potřeby ladění programu obsahuje web2py tiketovací systém pro záznam chyb. Tento systém administrátorovi důkladně zaznamená události, které vedly k chybě.

- **Bezpečnost** - Návrh rámce byl od počátku zaměřen na bezpečnost. Web2py obsahuje několik pokročilých opatření proti bezpečnostním hrozbám. Pro práci s databází se používá abstraktní databázová vrstva (DAL), která eliminuje útoky typu *SQL Injections*. Hesla jsou ukládána ve formě otisku (hashe). Šablonovací jazyk brání proti zranitelnosti *Cross-Site Scripting*. Generátor formulářů kontroluje validitu zadaných hodnot a blokuje *Cross-Site Request Forgeries*. Sezení jsou ukládána na straně serveru tak, aby se předešlo hrozbě *Cookie Tampering* a *Session Hijacking*.

6.1.1 Architektura

Jak již bylo zmíněno web2py je *full-stack* rámec. Jeho použití nevyžaduje složitou přípravu vývojového prostředí. Struktura rámce je rozdělena do několika úrovní. Členění hierarchie (*stack*) je znázorněno obrázkem č. 6.1. Na nejnižším stupni je postaven interpret jazyka Python. Nad ním je umístěn webový server. O úroveň výše se nachází knihovny rámce. Pro vývojáře je podstatná především poslední skupina, kterou představují samotné aplikace.

Web2py umožňuje paralelní běh několika různých aplikací, které mezi sebou mohou komunikovat. Každá aplikace přijímá své požadavky samostatně. K jejich zpracování však může využít dostupné moduly jiných aplikací.



Obrázek 6.1: Struktura rámce web2py. (Zdroj: [16])

Aplikace jsou ve web2py tvořeny pomocí návrhového vzoru *Model-View-Controller* (MVC). Ten vývojářům předepisuje rozdělit práci s daty do tří sekcí. První je **reprezentace** dat *(model)*, druhá je jejich **prezentace** *(view)* a třetí je **vnitřní logika operací** *(controller)*. V pravé části obrázku č. 6.1 je zachyceno vnitřní rozdělení aplikace, která se skládá z následujících složek:

- **models** - Soubory popisující datovou reprezentaci.

- **controllers** - Soubory popisující vnitřní logiku aplikace.
- **views** - Soubory popisující prezentaci dat.
- **languages** - Soubory obsahující jazykové mutace.
- **modules** - Python moduly, které přiléhají k aplikaci.
- **databases** - SQL logy.
- **static files** - Statické soubory, jako jsou CSS, JavaScript, atd.

Pokud aplikace nedisponuje vlastním webovým rozhraním, nemusí obsahovat všechny vyjmenované části. Zpracování klasického HTTP požadavku probíhá následujícím způsobem:

1) Požadavek dorazil na připojený webový server. (Server zpracovává každý požadavek paralelně ve vlastním vlákně.)
2) Na základě analýzy hlavičky požadavku je dotaz předán správné web2py aplikaci.
3) Pokud požadavek cílí na data umístěná ve složce *static files*, je automaticky zpracován a data jsou odeslána klientovi. Ostatní požadavky jsou namapovány na akce *(controller)* příslušné aplikace.
4) Před vyvoláním akce se odehraje kontrola sezení a postupné spuštění skriptů, které jsou umístěny ve složce model.
5) Po zpracování akce je výsledek prezentován klientovi. Pokud není specifikováno jinak, představuje celá operace jednu databázovou transakci.
6) Pokud stavový kód signalizuje úspěch, je výsledek automaticky potvrzen pomocí databázové operace `commit`.
7) Pokud operace selhala, je vytvořen záznam do tiketovacího systému a provedené změny v databázi jsou zrušeny příkazem `rollback`.

Protože bankovní funkce vyvíjeného programu mají být používány ostatními aplikacemi, je nutné zdrojové kódy umístit do složky `modules`. Jelikož aplikace nebude zpracovávat vlastní HTTP požadavky, musí návrh počítat i s odlišnostmi vůči klasickému postupu zpracování. Především je potřeba věnovat pozornost tomu, že skripty ve složce `models` nebudou spuštěny. Složka `models` v běžných situacích obsahuje konstrukci potřebných objektů z knihovny web2py. Jako jsou např. definice objektu abstraktní databázové vrstvy nebo objekt pro plánování události. Umístění těchto prostředků do modelů zajišťuje jejich zpřístupnění i v dalších částech aplikace. *(controller* a *view)*

Aby bylo možné spouštět potřebné funkce, jak z ostatních aplikací, tak při samostatném běhu aplikace, je potřeba přesunout deklaraci web2py objektů z modelů do modulů a obalit je novou třídou. Díky tomu lze následně vytvářet instance těchto obalených tříd v modelech i modulech. Potom je požadovaná funkcionalita přístupná napříč celým programem.

6.1.2 Abstraktní databázová vrstva

Abstraktní databázová vrstva (DAL) zapouzdřuje komunikaci s databázovým serverem. DAL obsahuje vlastní syntaxi pro deklaraci databázových tabulek, provádění dotazů nebo práci se záznamy. Jazyk je navržen tak, aby práce s ním byla stručná a intuitivní. Jeho oblíbenost postupně vzrostla natolik, že byl od vývoje web2py oddělen. Dnes je tedy možné používat DAL i samostatně bez web2py i v ostatních Python projektech.

Vrstva podporuje široké spektrum databázových systémů jako jsou: *SQLite*, *MySQL*, *PostgreSQL*, *MSSQL*, *FireBird*, *Oracle*, *IBM DB2*, *MongoDB*, *SAPDB*, *Informix* a *Ingres*.

DAL připojený databázový systém automaticky rozpozná a v reálném čase pro něj dynamicky generuje příslušné SQL. Programátor se tak může vyhnout práci s čistým SQL, jehož syntaxe se u každého systému může lehce odlišovat. Ve výchozím nastavení přidává DAL do vytvořených tabulek číselnou položku `id`, která je označena jako primární klíč.

Abstraktní databázová vrstva bude v projektu použita jako prostředek pro práci s databází.

6.1.3 Logování

Logování slouží jako nástroj pro zaznamenání událostí, které se v systému odehrály. Nasbírané informace se shromažďují v textových souborech (logy). Jejich přítomnost je potřebná zejména při analýze chyb. Web2py k logování využívá standardní knihovnu jazyka Python. Ta umožňuje zapisované zprávy rozdělit do pěti kategorií podle důležitosti: *debug*, *info*, *warning*, *error* a *critical*.

Nastavení cesty k logovacímu souboru a zpřístupnění logování pro aplikaci se provádí úpravou konfiguračního souboru `logging.conf`, který se nachází v kořenovém adresáři web2py. Pro zpřístupnění logování stačí přidat jméno aplikace (*eps*) do seznamu uvedeného ve výpise č. 6.1.

```
1 [loggers]
2 keys=root,rocket,markdown,web2py,rewrite,app,welcome,eps
```

Výpis 6.1: Zpřístupnění logování pro aplikaci *eps*.

Výpis č. 6.2 zobrazuje nastavení detailů logování. Nastavit lze úroveň zaznamenávaných zpráv, logování do souboru či konzole a plně kvalifikované jméno, které slouží pro rozpoznání souboru z kódů aplikace.

```
1 [logger_eps]
2 level=DEBUG
3 handlers=consoleHandler,rotatingFileHandler
4 qualname=web2py.app.eps
5 propagate=0
```

Výpis 6.2: Nastavení detailů logování pro aplikaci *eps*.

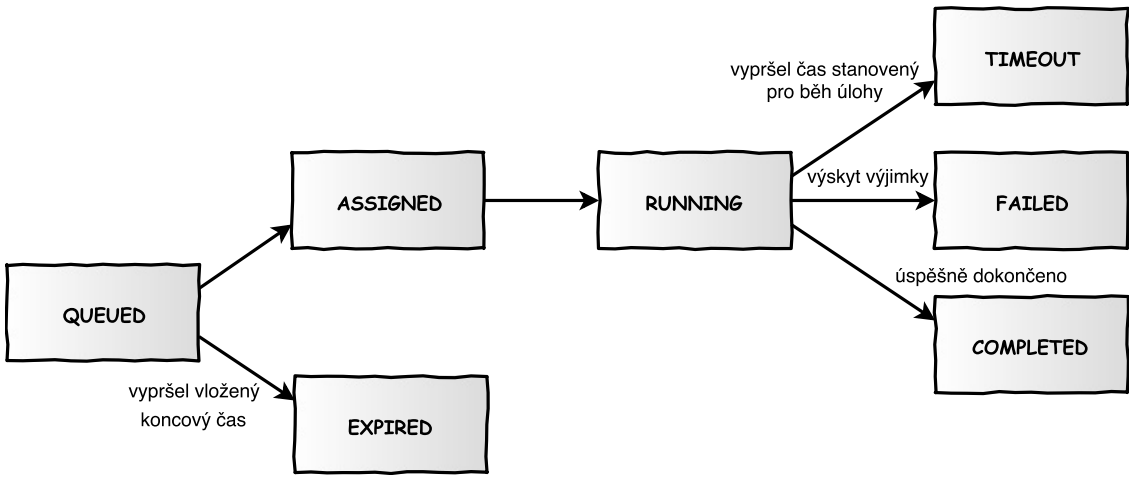
Navrhovaná aplikace bude své zásadní prováděné operace logovat do souboru. Informace budou ukládány v takovém tvaru, aby zpětnou analýzou výpisů bylo možné jednoznačně určit, kdy a s jakým výsledkem provedené operace skončily.

6.1.4 Plánovač událostí

Pro potřeby automatizovaného paralelního vykonávání zadaných úkolů na pozadí disponuje web2py vlastním plánovačem událostí (*Scheduler*). Plánovač nabízí sofistikované metody pro vytvoření, organizování a monitorování úkolů. O vykonání úlohy se starají speciální *pracovní* procesy (*workers*), které si jednotlivé úkoly vyzvedávají z databáze. Ve speciálních tabulkách databáze jsou ukládány všechny potřebné informace, jako jsou: *zadané parametry úlohy, stav úlohy, expirační doba, počet opakování, atd.* V databázi je udržován podrobný přehled o tom, které úkony byly provedeny a s jakým výsledkem skončily.

Životní cyklus úkolu je graficky znázorněn na obrázku č. 6.2. Počáteční událostí je vytvoření a naplánování úkolu pomocí příkazu `queu_task`. Parametry příkazu nastavují jeho vlastnosti. Specifikovat lze: jméno a parametry volané funkce, počáteční čas zahájení úkonu,

koncový čas *(Po jeho uplynutí nesmí být příkaz vykonán.)*, *timeout* operace, periodu s počtem opakování a maximální počet výjimek, který se během provádění úkonu může objevit. Po vytvoření se příkaz nachází ve stavu **QUEUED** a čeká, než bude přiřazen některému z volných zpracovatelských procesů. Jestliže byl zadán koncový čas události, který již uplynul, je úkon převeden do koncového stavu **EXPIRED** bez spuštění. Jinak je spuštěn a jeho běh může skončit třemi výsledky. **COMPLETED** znamená, že úloha byla úspěšně dokončena. Stav **FAILED** naznačuje, že během zpracování došlo k výskytu neošetřených výjimek a úloha proběhla neúspěšně. Poslední označení **TIMEOUT** signalizuje, že běh úkolu přesáhl nastavený časový limit a nebyl dokončen.



Obrázek 6.2: Životní cyklus plánované úlohy. (Zdroj: [16])

Plánovač událostí bude v projektu použit pro automatické stahování platební historie z účtu, která bude postupně přidávána do lokální databáze. Dále pak také pro nahrávání připravených plateb z lokální databáze do banky.

6.2 Databázový model

Pro potřeby uchování přístupových informací k účtu, historii zadaných transakcí a pohybů na účtě, je nutné navrhnout relační schéma databáze.

Jelikož rozbor jednotlivých API řešení v kapitole č. 5 ukázal, že každá banka používá jiný způsob ověření uživatele a rozdílné atributy formátů pro získání nebo odeslání platby. Rozhodl jsem se schéma rozdělit tak, aby každá banku měla k dispozici své vlastní tabulky pro uchování potřebných dat. Tím, že banka bude pracovat pouze se svými tabulkami, bude zajištěno, že další rozšiřování systému neovlivní strukturu již vytvořených schémat. Dále bude také zajištěna vyšší přehlednost a logické rozdělení dat, než by tomu bylo v případě, kdyby banky sdílely tabulky společně.

Jediným místem, kde budou uchovány informace o účtech vedených v různých bankách, je tabulka **account**. Ta bude obsahovat základní informace o účtu, jako jsou číslo účtu a kód banky nebo mezinárodní označení účtu IBAN (*International Bank Account Number*). Přístup k tabulkám jednotlivých bank bude rozlišen na základě jedinečného číselného kódu banky nebo mezinárodního bankovního identifikátoru BIC (*Bank Identifier Code*), který

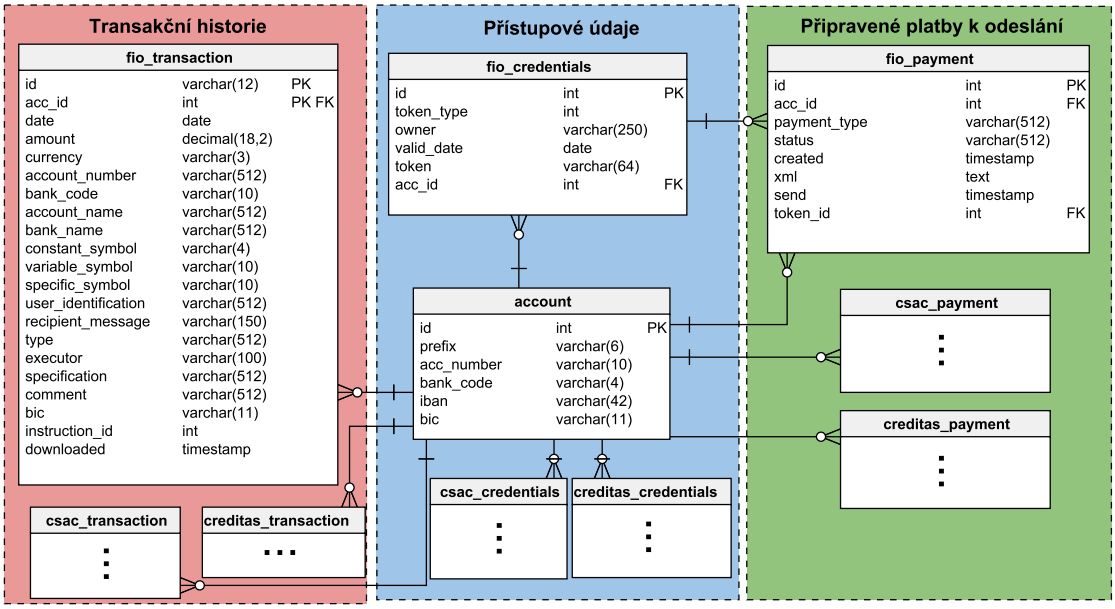
je bankám přidělován od společenství SWIFT (*Society for Worldwide Interbank Financial Telecommunication*).

Každý účet vedený ve Fio bance může mít více přístupových tokenů. Tyto údaje budou ukládány do tabulky **fio__credentials**. Ta kromě přístupového tokenu bude obsahovat i doplňující informace o rozsahu platnosti, expirační době a majiteli tokenu.

Pro ukládání plateb připravených k odeslání je navržena tabulka **fio__payment**. Záznam v tabulce je propojen s konkrétním `id` účtu a identifikátorem přístupového tokenu. To zaručuje snadné rozpoznání majitele, který transakci zadal. Struktura tabulky obsahuje připravenou transakci (část formátu Fio XML) i informace o tom, kdy byla transakce vytvořena a odeslána. Výsledek provedené operace půjde zjistit z hodnoty v sloupci *status*.

Poslední navrženou tabulkou je **fio__transactions**. Ta bude sloužit jako lokální mezipaměť (*cache*) transakční historie konkrétních účtů. Její složení kopíruje Fio XML schéma pro získávání informací z účtu Fio banky. Toto schéma bylo představeno v tabulce č. 5.4.

Vzájemné propojení jednotlivých tabulek s výhledem na budoucí rozšiřování systému je znázorněno ER diagramem na obrázku č. 6.3.



Obrázek 6.3: ER diagram navržené databáze. (Zdroj: vlastní)

6.3 Dekompozice programu

Tato podkapitola se věnuje návrhu rozdělení programu. Popsány zde budou jednotlivé části a jejich účel. Nastíněno bude i jejich zamýšlené propojení. Části obsahují i popis, který se vztahuje k dalšímu případnému rozvoji aplikace.

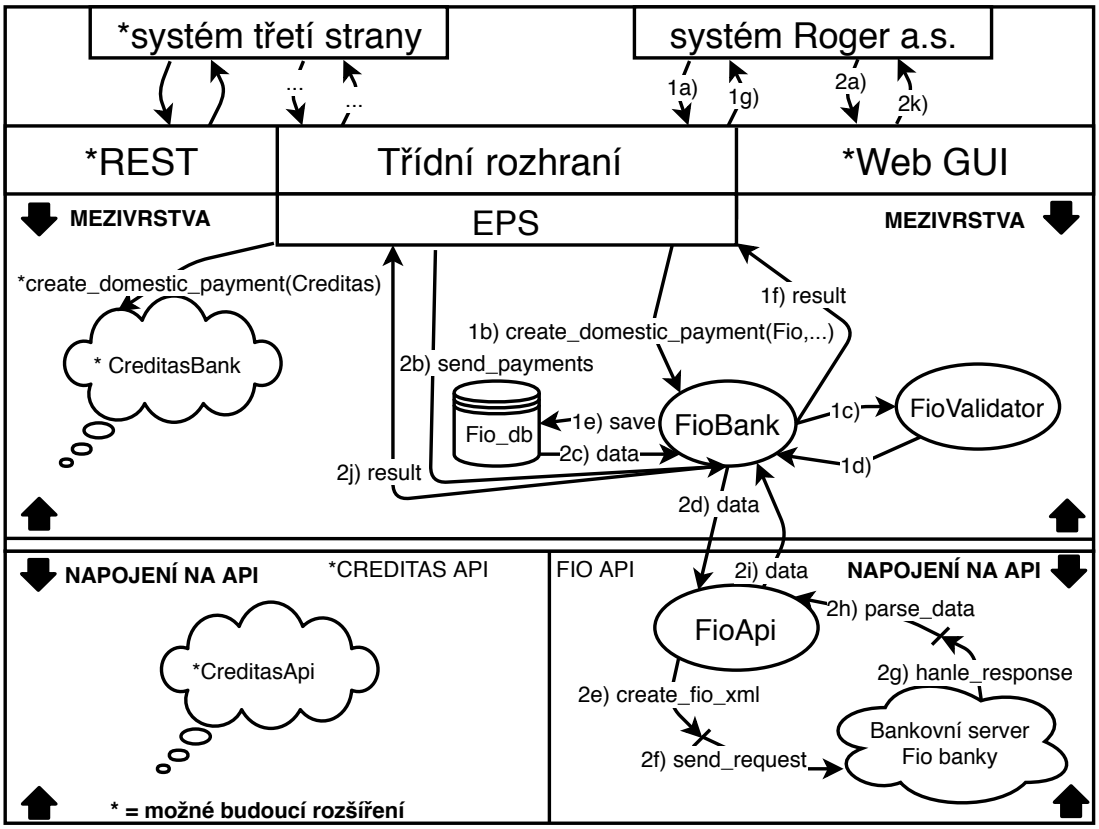
Návrh programu jsem rozdělil do dvou hlavních částí:

1) Obecná **mezivrstva** bude obsahovat jednotné rozhraní, které zpřístupní nové funkce a ovládání programu ostatním aplikacím pomocí veřejně dostupné třídy. Tato třída

dostala název podle zkratky jména vyvíjeného programového modulu - **EPS** *(Extraordinary Payment System)*. Mezivrstva bude mít dále za úkol rozpoznat banku, na kterou je požadovaná operace směřována. Pokud je banka systémem podporována, provede mezivrstva validaci vložených vstupních dat. Podle typu operace, buď uloží předem připravená data do databáze nebo z ní vyčte požadované informace.

2) **Napojení na bankovní API** bude kompletně odstíněno od práce s databází. Úlohou této části bude zprostředkovat komunikaci s bankovními API. Odpovědná třída převezme připravená data od *mezivrstvy*. Následně provede jejich převod do formátu, který vyžaduje banka. Podle specifikace banky přidá i další potřebné parametry a vytvořený požadavek odešle na bankovní server. Obsahuje-li příchozí odpověď data, bude provedena jejich separace. Informace o stavu výsledku a případná získaná data budou předány zpět do *mezivrstvy*.

Rozdělení do těchto částí je takto navrženo, protože dostupná bankovní API mohou svou podobu v blízké době ještě měnit. Pokud dojde ke změně struktury bankovních požadavků (URL adresy, formáty, atd.), bude stačit provést úpravy pouze v části *napojení na bankovní API*, bez nutnosti provádět změny v logice *mezivrstvy*. Základní rozdělení zpracování požadavku je pro lepší představu ilustrováno obrázkem č. 6.4. Obrázek do návrhu zahrnuje i další případné způsoby interakce s EPS, které by nad systémy šly v budoucnu vybudovat. Toto rozšíření by pak mohlo zpřístupnit vytvořené služby veřejným zákazníkům.



Obrázek 6.4: Rozdělení hlavních částí programu EPS. (Zdroj: vlastní)

6.3.1 Mezivrstva

V této sekci se blíže podíváme na návrh a funkce *mezivrstvy*. Vstupním bodem aplikace je rozhraní třídy **Eps**. Tato třída musí nabízet metody, které umožní vkládat a mazat přístupové údaje k účtům, zadávat platební transakce a zpřístupňovat platební historii. Toto rozhraní by se po integraci do firemního systému nemělo z dlouhodobého hlediska výrazně měnit tak, aby při rozšiřování funkčnosti nebylo nutné upravovat již funkční kód.

Návrh třídy EPS obsahuje tyto veřejné metody:

- `insert_or_update_credentials()` - Vkládání a upravování informací o účtu včetně přístupových tokenů.
- `remove_credentials()` - Smazání všech nebo části vložených informací.
- `create_domestic_payment()` - Vytvoření domácí platby.
- `create_euro_payment()` - Vytvoření platby v eurozóně.
- `create_foreign_payment()` - Vytvoření mezinárodní platby.
- `get_transactions()` - Získání pohybů na účtě.

Vzhledem k tomu, že každá banka vyžaduje rozdílné atributy ve svých požadavcích, budou parametry do většiny těchto metod předávány v asociativních polích (klíč-hodnota). Tyto pole se v Pythonu nazývají slovníky. Slovníky jsou z pohledu navrhovaného řešení vhodné, protože je lze bezproblémově rozšiřovat o další klíče. Podrobný popis struktury jednotlivých parametrů metod bude představen v sekci č. 7.1 kapitoly 7, která se věnuje implementaci. Hlavní úlohou těchto metod bude ověřit validitu základních údajů, jako je např. číslo zdrojového účtu. Z něj metoda rozpozná, u které banky je účet veden a požadavek předá dál ke zpracování objektu příslušné *bankovní třídy*. Případné chyby budou propagovány pomocí mechanismu výjimek.

O logiku, která je spojena s kontrolou vložených dat a práci s databází, se budou starat **bankovní třídy**. Návrh těchto tříd vychází z abstraktní třídy `Bank`. Ta bude obsahovat jednak abstraktní metody, které odpovídají metodám definovaným ve třídě *Eps*. Dále bude obsahovat i metody implementované, což jazyk Python umožňuje. Tím bude zajištěna možnost sdílet mezi bankami univerzální metody, které jsou nezávislé na konkrétní bance. Každá banka tedy bude mít svou vlastní třídu, která bude dědit vlastnosti od třídy `Bank`. Dědičnost zajistí, že objekt této třídy bude muset implementovat požadované chování abstraktních metod tak, aby mohl převzít zpracování požadavků od *Eps*. Toto rozdělení zároveň zajistí, že metody každé bankovní třídy se mohou libovolně odlišovat a přizpůsobovat potřebám pro práci s konkrétní bankou. Jak již bylo nastíněno v sekci č. 6.2, každá třída bude pracovat pouze se svojí částí databáze. Návrh třídy `FioBank`, která vychází z tohoto konceptu, je zachycen na obrázku A.1, který je součástí přílohy A.

Pro kontrolu, zda zadaná data odpovídají správnému formátu, budou bankovní třídy využít **validátory**. Struktura třídní hierarchie validátorů je podobná jako v případě bankovních tříd. Základní třídu tvoří `Validator` s tím rozdílem, že se nejedná o abstraktní třídu. Tato třída bude nabízet metody, které zkontrolují formát položek se stejnou specifikací. (Většinou definované bankovními standardy.) Od této třídy budou dědit validátory pro jednotlivé banky. Ty budou kontrolovat položky, jejichž specifikace je u každé banky rozdílná.

V případě třídy `FioValidator` se kontrolní pravidla budou řídit specifikací, která je uvedená v programové dokumentaci Fio API bankovnictví. Schéma navrhované struktury je rovněž zachyceno na prvním obrázku přílohy A. Pokud kontrola nebude odpovídat předepsanému stylu, vyvolá validátor odpovídající výjimku. Výjimka v sobě budou obsahovat relevantní textovou informaci, která umožní identifikovat chybně zadanou položku.

6.3.2 Napojení na bankovní API

Vzhledem k tomu, že každá banka disponuje úplně rozdílným složením požadavků, jak z pohledu datových formátů, tak URL adres nebo způsobu ověření klienta, nenalezl jsem žádné důvody, které by vedly k potřebě sdílet mezi bankami některé metody. Proto se o napojení na bankovní API budou starat samostatně oddělené třídy.

Hlavní úlohou těchto tříd bude převzít předem připravená data od *mezivrstvy* a následně je zabalit do příslušného formátu. V případě třídy `FioApi` bude pro odesílání plateb použit formát Fio XML. Odpovědností této třídy bude také na pokyn od mezivrstvy realizovat komunikaci s bankovním serverem. Výsledný stav operace bude předán zpět do mezivrstvy.

Pro čtení platební historie bude třída `FioApi` využívat formát JSON. Pomocí metod této třídy budou stažená data separována do jednotného tvaru, který očekává mezivrstva. Zároveň se získanou transakční historií bude do mezivrstvy předán i identifikátor posledního staženého ID pohybu. Tento údaj lze získat z hlavičky odpovědi. Identifikátor pohybu slouží jako zarážka. Po každém úspěšném stažení platební historie je na straně bankovního serveru tato zarážka automaticky nastavena na ID posledního odeslaného pohybu. Pro stažení dalších nových transakcí pak následně bude stačit zavolat pouze požadavek *získání pohybů na účtě od posledního stažení*. (Bylo představeno v sekci 5.1.4.) Vnitřní složení této třídy je graficky znázorněno obrázkem A.2 v příloze A.

6.4 Návrh testování aplikace

Pro potřeby testování aplikace je zapotřebí získat přístup k bankovnímu účtu u Fio banky. U zmíněné banky si tedy hodlám otevřít svůj osobní bankovní účet, protože ladění programu na firemních účtech není z bezpečnostního hlediska možné.

Pro automatické testování již v průběhu vývoje bude použit přístup *unit testing*. Takto napsané testy se budou zaměřovat na základní metody, které ověřují platnost dat, separují data, připravují data do formátu Fio XML a pracují s databází. Součástí těchto testů bude i negativní testování, které ověří zda metody správně reagují na nevhodně zadaná data.

Další částí testovacího procesu bude manuální testování na lokálním prostředí, ve kterém je projekt vyvíjen. (*Mac OS X, PostgreSQL*) Zde bude kladen důraz na otestování programu jako celku. Testovány budou zejména automatizované úlohy, které mají za cíl nahrát/získávat informace přímo z banky. Při tomto testování je nutný lidský zásah. Člověk totiž musí ručně zkontrolovat, zda nahraná/získaná data odpovídají těm údajům, které se nachází v internetovém bankovnictví.

Poslední fází testování bude zprovoznění aplikace ve firemním testovacím prostředí. Nasazení programu bude probíhat ve spolupráci s vývojovým týmem firmy. Následně bude programu přidělen přístup k firemním bankovním účtům. Ve spolupráci s finančním oddělením společnosti Roger a.s. bude ověřeno, zda program pracuje správně. Po schválení bude program nasazen do produkčního prostředí.

Kapitola 7

Implementace

V této kapitole bakalářské práce se blíže podíváme na implementaci aplikace, tzn. na popis fungování nejdůležitějších částí a metod programu. Podkapitola č. 7.1 věnuje svoji pozornost podstatným třídám a jejich obsahu. Z tohoto textu následně čerpá podkapitola č. 7.2, kde je vysvětleno, jak funguje proces vytvoření platby. Popisu, jak pracuje automatizované odesílání předem připravených plateb, se věnuje podkapitola č. 7.3. Poslední podkapitola č. 7.4 ujasňuje detaily ohledně zpřístupnění a stahování transakční historie.

7.1 Popis důležitých tříd

V jednotlivých částech této podkapitoly si postupně představíme klíčové parametry a metody tří tříd. Konkrétně se jedná o třídy `FioBank`, `FioValidator` a `FioApi`, jejichž úkoly byly nastíněny v návrhu.

7.1.1 Třída FioBank

Fio banka je zatím jedinou systémem podporovanou bankou. Proto parametry veřejných metod třídy *Eps* téměř zcela odpovídají těm, které si představíme v této sekci. *Eps* v současné době provádí připojení k databázi. Dále ověření, že při volání některé z dostupných metod bylo zadáno platné číslo zdrojového účtu vedeného ve Fio bance. Na požadavky směřované ostatním bankám reaguje *Eps* vyvoláním výjimky `UnsupportedBankError`. Následně *Eps* vytvoří objekt třídy *FioBank* a zavolá odpovídající metodu, které předá potřebná data.

Třída *FioBank* obsahuje jediný konstruktor, který přebírá připravené argumenty od *Eps*:

def ___init___(self, db, prefix=None, acc_num=None, bank_code=None)

Parametr `db` je *DAL* objekt připojený k databázi. Další parametry `prefix`, `acc_num`, `bank_code` jsou rozdělené části čísla zdrojového bankovního účtu. Během konstrukce je vytvořen také objekt třídy *FioValidator*, který je uložen jako privátní atribut. Tento objekt následně používají další metody při kontrole zadaných argumentů.

První akce, kterou je při práci s programem potřeba provést, je nahrání přístupových údajů k účtu. Bez toho, aniž by byl tento proces dokončen, není možné zadávat platby nebo číst transakční historii.

Pro tyto účely slouží metoda:

def insert_or_update_credentials(self, data)

Kde `data` je slovník, který musí obsahovat následující klíče:

- `account_number` nebo `iban` - Číslo bankovního účtu nebo mezinárodní identifikaci účtu IBAN. (povinný)
- `permission` - Typ oprávnění přístupového tokenu. Povolené jsou hodnoty **PAY** nebo **READ**. (povinný)
- `valid_until` - Datum expirace platnosti tokenu. (povinný)
- `owner` - Jméno majitele přístupových práv. (povinný)
- `token` - Vygenerovaný token k účtu. (povinný)
- `bic` - Mezinárodní označení banky - SWIFT kód. (volitelný)

Nejprve je ověřeno, zda slovník obsahuje všechny povinné položky, potom je postupně zkontrolován i jejich obsah. Pokud vše vyhovuje, jsou přístupové údaje uloženy do databáze. V případě, že stejný typ tokenu a jméno majitele přístupových práv bylo již dříve do systému vloženo, jsou informace aktualizovány. Na vložení přístupových práv jsou navázány i akce, které se týkají plánování automatizovaných operací. Tento postup bude vysvětlen v textu č. 7.3 a 7.4. Pro jeden účet se očekává, že dojde k vložení více přístupových tokenů s oprávněním **PAY**. K jejich rozlišení slouží parametr `owner`, který označuje jméno majitele přístupových práv, který bude zadávat platby. U tokenu typu **READ** stačí, aby jej zadal pouze jeden majitel.

Vložené přístupové údaje je možné kdykoliv odstranit pomocí metody:

def remove_credentials(self, account_or_iban, token_type='ALL', owner=None)

Metoda potřebuje znát pouze číslo účtu nebo IBAN (`account_or_iban`), jehož přístupové údaje mají být smazány. Ve výchozím nastavení jsou smazány všechny přístupové údaje. Volitelně je možné blíže specifikovat rozsah prováděné operace. Parametr `token_type` omezí aplikování změn na konkrétní typy tokenů. Podporovány jsou možnosti: **READ** - pouze tokeny oprávněné ke čtení, **PAY** - pouze tokeny umožňující zadání platby, **ALL** - všechny přístupové tokeny. Aby se smazání údajů dotklo pouze konkrétního majitele přístupových práv, je zapotřebí zadat jeho jméno do parametru `owner`. Při mazání údajů je nutné mít na paměti, že pokud v systému nezůstane žádný přístupový token pro konkrétní účet, jsou kvůli nastavení propojení cizích klíčů v databázi na *ON DELETE CASCADE* smazány i všechny ostatní informace (připravené/odeslané platby, stažená bankovní historie).

Třída *FioBank* dále nabízí i metody pro zadávání plateb a čtení historie. Jejich popisu se věnují samostatné podkapitoly č. 7.2 a 7.4.

7.1.2 Třída FioValidator

Třída *FioValidator* poskytuje metody pro kontrolu vstupních dat. Některé z těchto metod jsou označeny jako statické. Kvůli udržení logické souvislosti celku jsou přesto zařazeny do této třídy. Třída obsahuje pouze výchozí konstruktor, který nevyžaduje žádné argumenty.

Metody lze rozdělit podle toho, ke kterým operacím se vztahují. U většiny z nich je zkontrolováno, zda vložená proměnná odpovídá některému z očekávaných datových typů. Textové proměnné jsou kontrolovány na maximální délku a přítomnost pouze povolených znaků. K této kontrole jsou použity regulární výrazy.

Některá pravidla jsou nastavena tak, aby vyhovovala předepsanému XSD schématu.[1] Specifikace všech ostatních položek je rovněž dostupná v programové dokumentaci. [12]

Pro kontrolu přístupových údajů jsou připraveny následující metody:

- *def validate_iban(self, iban)* - Kontrola správného formátu mezinárodního označení účtu IBAN.
- *def validate_bank_account(self, account, use_modulo=False)* - Kontrola správného formátu čísla bankovního účtu. Číslo je tvořeno z volitelné části předčíslí (až 6 číslic) odděleného od samotného čísla účtu (až 10 číslic) pomlčkou. Poslední čtyřmístnou číselnou část tvoří kód banky, který je odlišen lomítkem. Parametr `use_modulo` určuje, zda má být formát kontrolován podle českých bankovních předpisů.
- *def validate_permission(permission)* - Kontrola, zda `permission` obsahuje hodnotu *READ* nebo *PAY*.
- *def validate_owner(self, owner)* - Kontrola, zda `owner` obsahuje textovou hodnotu.
- *def validate_token_insert(token)* - Kontrola, zda `token` má požadovanou délku 64 znaků.
- *def validate_date(date_val, test_future=True)* - Kontrola, zda `date_val` je objekt typu *datetime* nebo správně formátovaný textový řetězec. Parametr `test_future` nastavuje, zda může být zadáno datum z minulosti. Všechny časové údaje s kterými program pracuje, jsou v časové zóně UTC.

Některé z vyjmenovaných metod se používají i při kontrole platebních příkazů. Třída `FioValidator` obsahuje i další metody, které jsou schopny zkontrolovat všechny zadané hodnoty platebních příkazů. Nejdůležitější z nich jsou:

- *def validate_currency_and_amount(self, currency, amount)* - Tato metoda kontroluje, zda je zadaná měna (`currency`) správně reprezentována třímístnou textovou zkratkou nebo číselným kódovým označením. Zároveň je zkontrolována i částka odesílané transakce. Při vkládání hodnoty platby očekává program objekt typu *decimal*, celé číslo nebo textovou reprezentaci částky oddělenou tečkou. *Decimal* je použit pro práci s desetinnými čísly kvůli své přesnosti a pokročilým možnostem zaokrouhlování.
- *def validate_pay_numbers(self, data)* - Metoda kontroluje správný tvar konstantního symbolu (až 4 číslice) a variabilního nebo specifického symbolu (až 10 číslic).
- *def validate_pay_address(self, data)* - Tato metoda zapouzdřuje více odlišných metod, které kompletně zkontrolují všechny části zadané adresy příjemce.
- *def validate_pay_messages(self, data)* - Metoda dohlíží na korektní obsah dodatečně volitelných textových informací, které se v platbě mohou vyskytnout. Jako jsou např. zpráva pro příjemce, upřesnění platby, atd.

Špatně zadaný formát některého z argumentů způsobí vyvolání výjimky *ValueError*. Na neočekávaný datový typ proměnné reagují metody chybou *TypeError*. V ojedinělých případech, kdy vyprší expirační platnost některého z tokenů, může výsledek operace skončit výjimkou *TokenError*.

[1] https://www.fio.cz/schema/fio_xml_type.xsd

7.1.3 Třída FioApi

Třída *FioApi* zprostředkovává operace, které vyžadují komunikaci s bankovním serverem. Dokáže zabalit potřebná data do formátu FioXML nebo separovat odpověď podle transakčního JSON schéma. *FioApi* obsahuje pouze výchozí konstruktor, který nastavuje hodnoty následujících konstant:

- `FIO_XML_HEADER` - Hlavička FioXML souboru.
- `FIO_XML_FOOTER` - Ukončení FioXML souboru.
- `MAX_XML_FILE_SIZE` - Maximální přípustná velikost nahrávaného souboru. (2 MB)
- `BASE_URL` - Základní URL adresa bankovního REST API serveru.
- `ACTION` - Slovník obsahující URI adresy jednotlivých požadavků.
- `TRANSACTION_SCHEMA` - Transakční schéma dat zasílaných odpovědí.

Pro tvorbu jednotlivých plateb ve formátu FioXML slouží metoda:

def create_xml(self, pay_type, data)

Metoda vrací odpovídající vytvořenou část souboru FioXML, která neobsahuje hlavičku ani zápatí. Parametr `pay_type` určuje, jaký typ platby se má vygenerovat. Podporované hodnoty jsou **domestic** (domácí platba), **euro** (europlatba) nebo **foreign** (mezinárodní platba). Slovník `data` obsahuje zkontrolované povinné, případně i volitelné parametry transakce, které jsou převzaty od třídy *FioBank*.

Další důležitá metoda, která předem připravené části skládá do kompletních FioXML souborů, je:

def create_payment_files(self, parts, domestic_count, euro_count, foreign_count)

Tato metoda převezme jednotlivé transakce v seznamu `parts`. Následně je rozdělí do samostatných souborů tak, aby jejich velikost vyhovovala konstantě `MAX_XML_FILE_SIZE`. Do souboru jsou přidány hlavičky a zápatí. Společně s platbami jsou metodě předány i počty jednotlivých typů transakcí, aby bylo možné soubory seřadit v předepsaném pořadí: 1) **domácí platba**, 2) **europlatba**, 3) **mezinárodní platba**.

Aby takto předem připravené soubory byly kompletní, stačí již přidat pouze přístupový token k účtu. Import plateb do banky zajistí metoda:

def send_payment(self, token, upload_file)

Kde `token` je přístupový *PAY* token a `upload_file` je soubor typu FioXML.

Získání platební historie je z pohledu prováděných operací podstatně jednodušší. Její stažení zprostředkovává metoda:

def download_transactions(self, token, action='last')

Metoda očekává přístupový *READ* nebo *PAY* token. Parametr `action` není nutné zadávat. V případě úspěchu vrátí operace seznam slovníků. Každý slovník představuje jeden pohyb na účtě a obsahuje oddělené informace o provedené transakci. Klíče tohoto slovníku kopírují jména sloupců v databázové tabulce *fio_transaction*. Tyto názvy je možné dohledat v obrázku č. 6.3.

7.2 Proces vytvoření platby

Po představení hlubšího kontextu o jednotlivých třídách a metodách se nyní můžeme podívat na jejich vzájemné propojení. Pro vytvoření platby nabízí třída *Eps* tři metody:

- *def create_domestic_payment(self, account_from, token_owner, currency, amount, account_to, others=None)*
- *def create_euro_payment(self, account_from, token_owner, currency, amount, account_to, others)*
- *def create_foreign_payment(self, account_from, token_owner, currency, amount, account_to, others)*

Parametry metod jsou na první pohled stejné. Formáty některých položek jsou však rozdílné. Parametr `account_from` představuje číslo účtu, ze kterého má být platba odeslána. Označení `token_owner` se váže ke jménu majitele přístupových údajů. Toto jméno je specifikováno při vložení přístupového tokenu do systému. Parametry `currency` a `amount` označují měnu a celkovou částku prováděné transakce. U domácí platby označuje `account_to` číslo bankovního účtu příjemce. Europlatba a mezinárodní transakce zde vyžaduje označení IBAN. Do parametru `others` se vkládá slovník, který obsahuje další volitelné (u některých plateb povinné) atributy. Odlišnosti těchto atributů u různých typů plateb jsou k porovnání v tabulkách, které obsahuje předchozí sekce č. 5.1.3. Všechny podporované klíče tohoto slovníku jsou u jednotlivých metod podrobně popsány v programové dokumentaci.

Vnitřní logika těchto metod je v zásadě velmi podobná. Po vyvolání akce je nejprve zkontrolováno, že databáze přístupových údajů obsahuje odpovídající *PAY* token s jménem majitele zdrojového účtu. V případě, že tomu tak není, reaguje systém vyvoláním výjimky typu `CredentialsError`. Pokud je vše v pořádku, jsou pomocí objektu třídy *FioValidator* zkontrolována všechna vložená data, která jsou potom převedena do unifikované podoby. Jestliže nebylo zadáno datum splatnosti nebo priorita platby, systém automaticky nastaví standardní dobu zpracování s dnešním datem účinnosti. Následně je zkonstruován objekt typu *FioApi*, který připravenou transakci převede do formátu *FioXML*. Takto nachystaná platba je společně s časovým razítkem uložena do databáze jako záznam v tabulce *fio_payment*. Zde čeká ve stavu **PENDING** na odeslání.

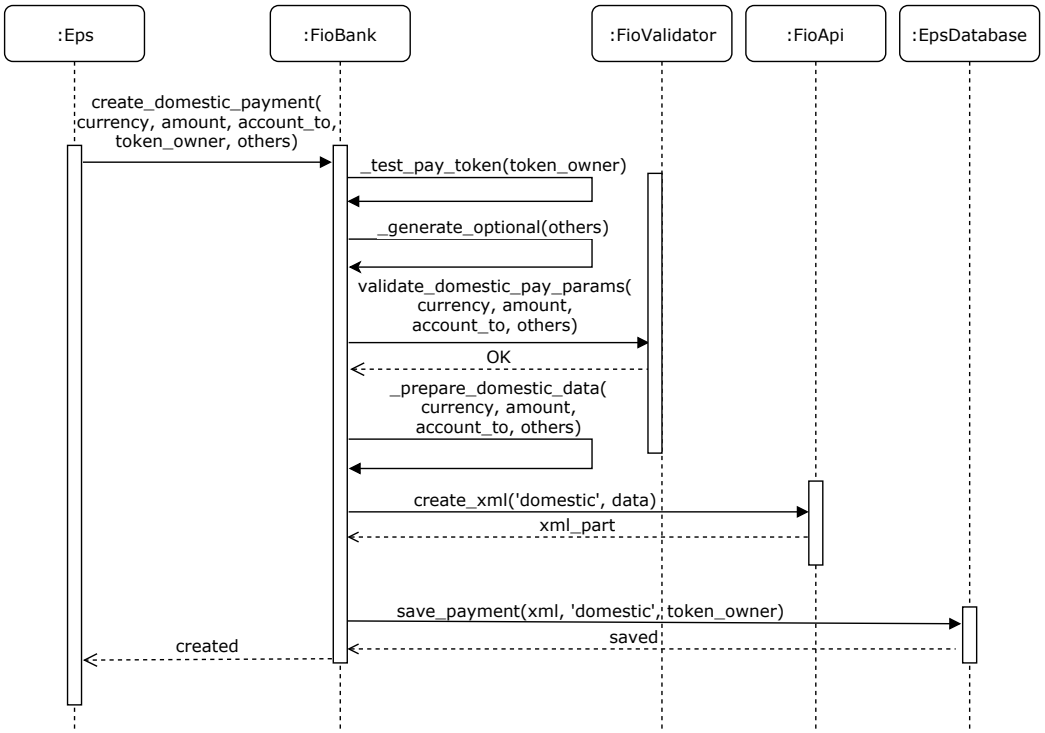
Průběh tohoto procesu je zachycen sekvenčním diagramem na obrázku č. 7.1.

7.3 Odeslání připravených plateb

Pro odesílaní nachystaných plateb používá systém automatizované úlohy v plánovači událostí, jehož princip byl popsán v sekci č. 6.1.4.

Vložení přístupového tokenu typu *PAY* je zároveň počáteční okamžik pro plánování automatického odesílání. Jako poslední část této operace je do plánovače přidána událost `fio_plan_send`. Tato událost potřebuje ke svému chodu dva parametry. První z nich je identifikátor čísla účtu, přidělený po přidání do databáze. Druhý parametr je jméno majitele přístupového tokenu. Úloha je spouštěna s periodou 1 hodina a nekonečným počtem opakování. Pokud jsou z databáze odstraněny příslušné přístupové údaje, skončí úloha ve stavu **FAILED** a její chod je přerušen.

V každém běhu události `fio_plan_send` se na pozadí odehrávají následující operace. Nejprve je zkontrolováno, zda v databázi nejsou uloženy nějaké platby ve stavu **PENDING**, které byly zadány jménem majitele přístupových údajům. Pokud ano, jsou tyto



Obrázek 7.1: Sekvenční diagram použitých operací při tvorbě domácí platby. (Zdroj: vlastní)

transakce načteny z databáze a jejich stav změněn na **PROCESSING**. Získané platby jsou pomocí metody `create_payment_files` objektu třídy *FioApi* rozděleny do potřebných souborů. Pro každý FioXML soubor je následně naplánována úloha `fio_send_payment`. Tím, že plánování probíhá pro každý *PAY* token odděleně, je zajištěna eliminace škod v případě neočekávaného selhání některého z těchto plánovacích procesů.

Úloha `fio_send_payment` získá na svém vstupu FioXML soubor a přístupový token. Čas na vykonání (*timeout*) této úlohy je nastaven na 120 sekund. Vstupní údaje jsou předány metodě `send_payment`, kterou poskytuje objekt třídy *FioApi*. Ta provede odeslání požadavku na bankovní server, který může skončit jedním z následujících scénářů. Pokud byly všechny platby úspěšně nahrány do internetového bankovnictví, je jejich stav v databázi změněn na **SENT**. Pokud požadavek neuspěl z důvodu, že stejný přístupový token byl použit méně než před 30 sekundami, je proces odesílání na půl minuty přerušen a poté se opakuje znovu. Jestliže operace neuspěla kvůli tomu, že soubor FioXML obsahoval nesprávně zadanou platbu, kterou proces validace nemohl odhalit (např. volba mezinárodní transakce, místo europlatby), je chybná transakce v souboru identifikována, situace zaznamenána do příslušných logů a transakce je převedena do stavu **ERROR**. Ostatní platby, které se v nahrávaném souboru nacházely, jsou znovu převedeny do stavu **PENDING**. Jejich odeslání bude znovu naplánováno při dalším spuštění plánovacího cyklu. Poslední případ představuje situaci, kdy selže nahrání všech plateb v souboru. Tato komplikace může nastat pouze v omezených případech, které systém nemůže ovlivnit. Např. výpadek na straně bankovního serveru nebo případ, kdy majitel přístupových údajů do systému omylem vloží neplatný token. V těchto extrémních situacích skončí všechny připravené

platby ve stavu **FAILED**. Potřebné informace k analýze problému jsou opět zalogovány. Pro následné oživení takto ukončených plateb je nutný zásah administrátora systému.

Historie úspěšně odeslaných plateb je v databázi uchovávána 3 dny. Pak jsou záznamy automatizovaně smazány úlohou `fio_clean_db`, která se spouští jednou denně. Na blížící se dobu expirace přístupového tokenu je uživatel upozorněn 14 dní před vypršením prostřednictvím varovných logů.

7.4 Stahování a zpřístupnění transakční historie

Počátečním bodem pro zahájení automatického stahování bankovní historie je vložení přístupového tokenu typu *READ*. Následně je naplánována úloha `fio_download_history`. Nastavení parametrů běhu této úlohy je stejné jako v případě plánování odesílaných plateb. Úkol očekává jediný vstupní parametr, kterým je unikátní identifikátor čísla účtu. Každou hodinu je pro konkrétní účet spuštěna metoda `download_transactions`, kterou disponuje třída *FioApi*. Získaná data jsou společně s přijatým identifikátorem posledního úspěšně staženého pohybu předány do mezivrstvy.

Mezivrstva následně zkontroluje, zda toto `ID` odpovídá identifikátoru posledního pohybu, který je uložen v tabulce `fio_transaction`. Pokud tomu tak není, znamená to, že v databázi nejsou korektně uloženy všechny transakce. Následně tedy mezivrstva vyvolá resetování zarážky pomocí metody `set_last_id` objektu třídy *FioApi*. Tato metoda nastaví zarážku na straně bankovního serveru na ID posledního pohybu, které je uloženo v databázi. Tím je zajištěno, že v dalším cyklu stahování bude zahrnuta i historie, která již stejným tokenem stažena byla, ale není uložena v databázi. K této situaci může dojít v případě, že přístupový token byl použit i mimo systém Eps.

Transakční historie je pro potřeby zpracování přístupná po dobu 3 dní. Poté je opět smazána úlohou `fio_clean_db`. Případné prodloužení této doby je možné nastavit upravením vstupního parametru této úlohy.

Pro získání takto uložené historie z databáze nabízí třída *Eps* metodu:

def get_transactions(self, account_or_iban, date_from=None, date_to=None, only_incoming=False, only_outgoing=False)

Parametr `account_or_iban` specifikuje účet, jehož historie je požadována. Ostatní parametry slouží pro možnost filtrování. Nastavením parametru `date_from` a `date_to` je možné omezit časový interval uskutečněných pohybů, `date_to` je ve výchozím nastavení dnešní datum. Specifikováním parametru `only_incoming` nebo `only_outgoing` je možné nastavit, zda mají být do výběru zahrnuty pouze příchozí nebo odchozí platby.

Kapitola 8

Testování

Tento text shrnuje průběh testování programu. Aplikace byla otestována, jak při samostatném běhu, tak i jako modul importovaný do ostatních *web2py* aplikací. V podkapitole č. 8.1 je vysvětlen průběh automatického testování a jeho rozsah. Následující podkapitola č. 8.2 představuje princip manuálního testování na lokálním a firemním prostředí.

8.1 Automatické testování

Pro účely automatického testování aplikace byla použita standardní knihovna jazyka Python `unittest`. Tato knihovna nabízí prostředky pro oddělené testování různých částí kódu. Cílem tohoto pocesu je napsat vzájemně nezávislé testy, které hlídají správnou funkci zkoušených operací. Po úpravě kódu lze díky spuštění těchto testů zjistit, zda provedená změna negativně neovlivnila některou z dalších částí programu. Při tvorbě těchto testů jsem čerpal ze své předchozí zkušenosti s testováním softwaru a také z programové dokumentace knihovny. [18]

Knihovna nabízí široké spektrum možností, jak napsat potřebné testy. Důležitou částí je třída `unittest.TestCase`, od které dědí jednotlivé skupiny testů. Třída obsahuje dvě důležité metody. Metoda `setUp` se spouští před každým samostatným testem. Tuto metodu je tedy možné použít pro přípravu potřebných objektů a dat před provedením testu. Druhá metoda `tearDown` se stará o provední úkonů po skončení testu. Lze ji využít například pro odstranění vytvořených dat z databáze, atd.

Jednotlivé testy pak tvoří metody, jejichž název začáná slovem `test_`. Samostatný test se zaměřuje na provádění různých operací s daty a jejich následné porovnání s očekávanou hodnotou. Pro tyto účely obsahuje třída `TestCase` skupinu metod s počátečním označením *assert*. Metody slouží například pro kontrolu, zda se získaná hodnota shoduje s očekávanou nebo zda byl vyvolán očekávaný typ výjimky.

Automatické testování bylo v projektu nejprve cíleno na malé samostatné metody, které mají za úkol prověřit validitu dat. Následně byly testy rozšířeny i na složitější operace pracující s databází. Výsledné pokrytí testů obsahuje všechny procesy týkající se ověření, uložení a mazání přístupových údajů. Dále také vytvoření všech druhů plateb. Zde je otestována kontrola dat transakce, její správná transformace do formátu FioXML a následné uložení do databáze. Testovány jsou i metody, které zpřístupňují staženou transakční historii z databáze. Tyto testy pracují s fiktivními bankovními daty. Automatické testy nepokrývají komunikaci s bankovním serverem, protože ten vyžaduje reálný přístup ke konkrétnímu bankovnímu účtu. Komunikace by šla nasimulovat promocí falešně vygenerovaných odpo-

vědí. Vzhledem k tomu, že systém byl manuálně otestován na reálném bankovním účtu, nepovažuji tuto simulaci za přínosnou.

Vytvořené testy jsou přiloženy společně se zdrojovými kódy na paměťovém médiu. Jejich umístění je možné nalézt v příloze B. Návod, jak lze testy spustit, je dostupný v příloze C.

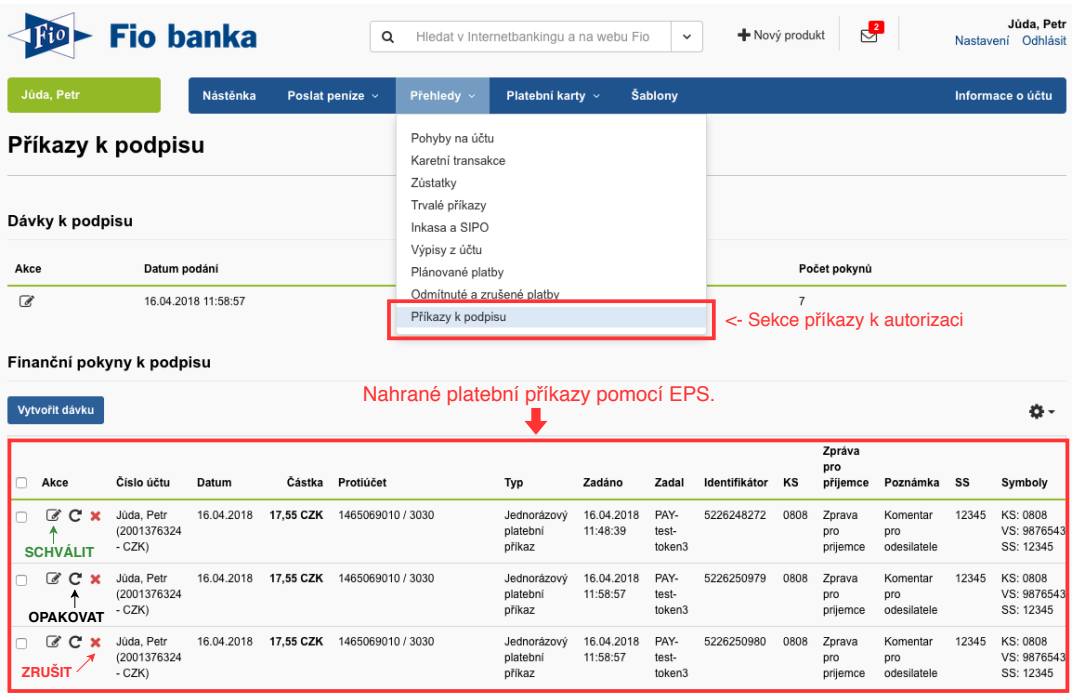
8.2 Manuální testování

K manuálnímu testování jsem využil svůj osobní bankovní účet vedený u Fio banky. Testování probíhalo nejprve na lokálním prostředí. Za tímto účelem byl vytvořen Python skript a další *web2py* aplikace `import_tester`. Tato aplikace neopsahuje žádné vlastní operace. Pouze importuje modul *Eps* a demonstruje jeho použití v ostatních aplikacích. Python skript byl různě upravován a spouštěn v *shell* prostředí **Eps** a **import_tester**. Postupně tak byly otestovány všechny dostupné operace s připojením na reálný bankovní server Fio banky.

Pro kontrolu, zda se hodnoty uložené v databázi shodují s těmi, které jsou v internetovém bankovnictví, byl použit program PgAdmin. Ten umožňuje přehledně procházet obsah jednotlivých tabulek a záznamů v databázovém systému PostgreSQL. Jak vypadají úspěšně zadané platby pomocí Eps zachycuje snímek internetového bankovnictví na obrázku č. 8.1.

K manuálnímu testování je možné upravit připravený skript `examples.py`, který obsahuje příklady použití systému. Tento soubor je přiložen na paměťovém médiu. Jeho umístění a návod ke spuštění je taktéž vysvětlen v přílohách B a C.

Po úspěšném odladění na lokálním prostředí byl program nasazen na testovací prostředí firmy. Zde byla znovu ověřena jeho funkčnost. Účetní oddělení firmy před autorizací jednotlivých transakcí kontrolovalo, zda jejich obsah odpovídá správnému tvaru. Program při testování uspěl a nyní čeká na nasazení do plné produkce.



Obrázek 8.1: Nahrané platby do internetového bankovnictví. (Zdroj: vlastní)

Kapitola 9

Závěr

Cílem této práce bylo seznámit se s novou bankovní legislativou, navrhnout a implementovat systém, který umožní zadávat příkazy k platbě a stahovat transakční historii z bankovních účtů vedených ve Fio bance. Tento systém bude sloužit jako rozšíření dosavadního firemního softwaru společnosti Platební instituce Roger a.s.

Legislativní úpravy vycházejí ze směrnice Evropského parlamentu a Rady EU o platebních službách na vnitřním trhu, která nabyla právní účinnost v lednu 2018. Tato direktiva je známá pod zkratkou PSD2 a nařizuje bankovním institucím zveřejnit svá API rozhraní třetím stranám. Pomocí bankovních API je nově umožněno komunikovat přímo s bankovním serverem. Třetí strana může se souhlasem klienta iniciovat platbu nebo stahovat informace o pohybech na klientském účtu.

Návrh aplikace vychází z rozboru dostupných API řešení několika českých bank. Analýze byla podrobena rozhraní Fio banky, České spořitelny a Banky Creditas. Tyto prostředky jsou vytvořeny pomocí technologie REST API. Aplikace je připojena k rozhraní Fio banky. Architektura však počítá s dalším rozšiřováním funkcionality o podporu dalších bank.

Program je vytvořen v programovacím jazyce Python s využitím aplikačního rámce *web2py*. Jeho funkčnost byla ověřena na reálných bankovních účtech. Nově vytvořené funkce umožnily zautomatizovat operace, které jsou ve firmě spojené se zadáváním platebních příkazů.

V rámci této bakalářské práce mi bylo umožněno podílet se na vývoji reálného firemního projektu. Konzultace s lidmi z vývojového týmu společnosti mi pomohly zdokonalit se v návrhu a tvorbě *backend* softwaru. Zajímavou zkušeností bylo pracovat s moderními technologiemi, které byly při tvorbě použity.

Se společností hodlám nadále spolupracovat na rozvoji vytvořeného systému. Podpora zpracování bankovních operací bude obohacena o napojení na další bankovní API. V blízké době bude zprovozněna komunikace s Českou spořitelnou a Bankou Creditas. Výhledově pak i s Air bank a dalšími českými bankami. Jádro práce poslouží jako základní stavební kámen pro další směry rozvoje projektu. Jedním z nich je vytvoření nového portfólia služeb, které by dostupné bankovní operace využívalo při interakci s firemními zákazníky. Další zajímavou myšlenkou je zpřístupnění veřejného uživatelského rozhraní, které by mohlo mít podobu vlastního REST API nebo GUI. Tyto záměry budou případně realizovány až po udělení PSD2 licence od České národní banky a legislativním schválení potřebných technologických a bezpečnostních standardů.

Literatura

[1] Banka Creditas, a.s.: *Dokumentace: Creditas OpenAPI.* [Online; navštíveno 30.03.2018].
URL https://www.creditas.cz/documents/20705/59823/Creditas+API+-+dokumentace.pdf/a62ca458-f9b3-49f2-9551-bf85cf8e3b78

[2] Banka Creditas, a.s.: *Manuál: Creditas OpenAPI.* [Online; navštíveno 30.03.2018].
URL https://www.creditas.cz/documents/20705/59823/Creditas+API+-+manu%C3%A1l.pdf/ac791179-6251-4542-823f-2d582f2fe938

[3] Banka Creditas, a.s.: *Technická dokumentace: Creditas OpenAPI.* [Online; navštíveno 30.03.2018].
URL https://app.swaggerhub.com/apis/Creditas/OpenAPI/1.0.0#/

[4] Banka Creditas, a.s.: *Tisková zpráva: Banka Creditas integruje ve svém internetovém bankovnictví účet jiné banky.* [Online; navštíveno 31.03.2018].
URL https://www.creditas.cz/documents/20705/46404/Tiskov%C3%A1+zpr%C3%A1va_Banka+CREDITAS+jako+prvn%C3%AD+integruje+do+IB+jinou+banku/f71c0f00-3cfb-43b9-b87c-3e90e3489fce

[5] Ecma International: *The JSON Data Interchange Syntax.* 2017, [Online; navštíveno 15.03.2018].
URL http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf

[6] Erste Group Bank AG: *Dokumentace: Erste API Hub.* [Online; navštíveno 27.03.2018].
URL https://www.ersteapihub.com

[7] Evropská bankovní federace: *EBF video: What is screen-scraping?* [Online; navštíveno 22.01.2018].
URL https://www.ebf.eu/what-is-screen-scraping/

[8] Evropská komise: *Platební služby (PSD2) - status zavádění.* [Online; navštíveno 20.01.2018].
URL https://ec.europa.eu/info/publications/payment-services-directive-transposition-status_en

[9] Evropský orgán pro bankovnictví: *Technical Standards on the EBA Register under PSD2.* [Online; navštíveno 21.01.2018].
URL https://www.eba.europa.eu/regulation-and-policy/payment-services-and-electronic-money/technical-standards-on-the-eba-register-under-psd2

[10] Evropský parlament, Rada Evropské unie: *Směrnice Evropského parlamentu a Rady (EU) 2015/2366 ze dne 25. listopadu 2015 o platebních službách na vnitřním trhu, kterou se mění směrnice 2002/65/ES, 2009/110/ES a 2013/36/EU a nařízení (EU) č. 1093/2010 a zrušuje směrnice 2007/64/ES*. Úřední věstník, L 337, Prosinec 2015, 35–127 s., ISSN: 1977-0626.
URL http://eur-lex.europa.eu/legal-content/CS/ALL/?uri=uriserv:OJ.L_.2015.337.01.0035.01.CES

[11] Filip Michalec: *Silné ověření klienta podle RTS ke směrnici PSD2*. [Online; navštíveno 22.01.2018].
URL https://www.epravo.cz/top/clanky/silne-overeni-klienta-podle-rts-ke-smernici-psd2-105724.html?mail

[12] Fio banka, a.s.: *Dokumentace: Fio API bankovnictví*. [Online; navštíveno 26.03.2018].
URL https://www.fio.cz/docs/cz/API_Bankovnictvi.pdf

[13] Hardt, D.: *RFC 6749: The OAuth 2.0 Authorization Framework*. 2012.
URL https://tools.ietf.org/html/rfc6749

[14] Jarmila Tornová: *PSD 2 – regulace platebních služeb v novém kabátě*. [Online; navštíveno 21.01.2018].
URL https://www.epravo.cz/top/clanky/psd-2-regulace-platebnich-sluzeb-v-novem-kabate-100956.html

[15] Leonard Richardson, Sam Ruby: *RESTful web services*. Sebastopol : O'Reilly, 2007, ISBN 978-0-596-52926-0.

[16] Massimo Di Pierro: *web2py Complete Reference Manual*. Experts4Solutions, 2013, ISBN 978-0-578-12021-8.

[17] Michael B. Jones, D. H.: *RFC 6750: The OAuth 2.0 Authorization Framework: Bearer Token Usage*. 2012.
URL https://tools.ietf.org/html/rfc6750

[18] Python Software Foundation: *Unit testing framework*. [Online; navštíveno 17.04.2018].
URL https://docs.python.org/2/library/unittest.html

[19] Reschke, J. F.: *RFC 7617: The 'Basic' HTTP Authentication Scheme*. 2015.
URL https://tools.ietf.org/html/rfc7617

[20] Rifaat Shekh-Yusef, S. B., David Ahrens: *RFC 7616: HTTP Digest Access Authentication*. 2015.
URL https://tools.ietf.org/html/rfc7616

[21] Roy T. Fielding, J. F. R.: *RFC 7231: Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content*. 2014.
URL https://tools.ietf.org/html/rfc7231

[22] Roy T. Fielding, J. F. R.: *RFC 7372: Hypertext Transfer Protocol (HTTP/1.1): Conditional Requests*. 2014.
URL https://tools.ietf.org/html/rfc7232

[23] Twitter Inc.: *Twitter API dokumentace*. [Online; navštíveno 15.03.2018].
URL https://developer.twitter.com/en/docs/tweets/post-and-engage/api-reference/post-statuses-update

[24] World Wide Web Consortium: *Extensible Markup Language (XML) 1.1*. [Online; navštíveno 15.03.2018].
URL https://www.w3.org/TR/2006/REC-xml11-20060816/

[25] WWW stránky: *Extensible Markup Language*. [Online; navštíveno 15.03.2018].
URL https://cs.wikipedia.org/wiki/Extensible_Markup_Language/

[26] WWW stránky: *JSON Schema*. [Online; navštíveno 15.03.2018].
URL http://json-schema.org/

[27] Česká bankovní asociace: *Sdělení České bankovní asociace k přechodnému období PSD2 a nového zákona o platebním styku*. [Online; navštíveno 22.01.2018].
URL https://www.czech-ba.cz/sites/default/files/tz_psd2_a_novy_zakon_o_platebnim_styku.pdf

[28] Česká bankovní asociace: *Český standard pro Open Banking*. [Online; navštíveno 28.03.2018].
URL https://www.czech-ba.cz/sites/default/files/cesky-standard-pro-open-banking/ceskystandardproopenbankingv021.pdf

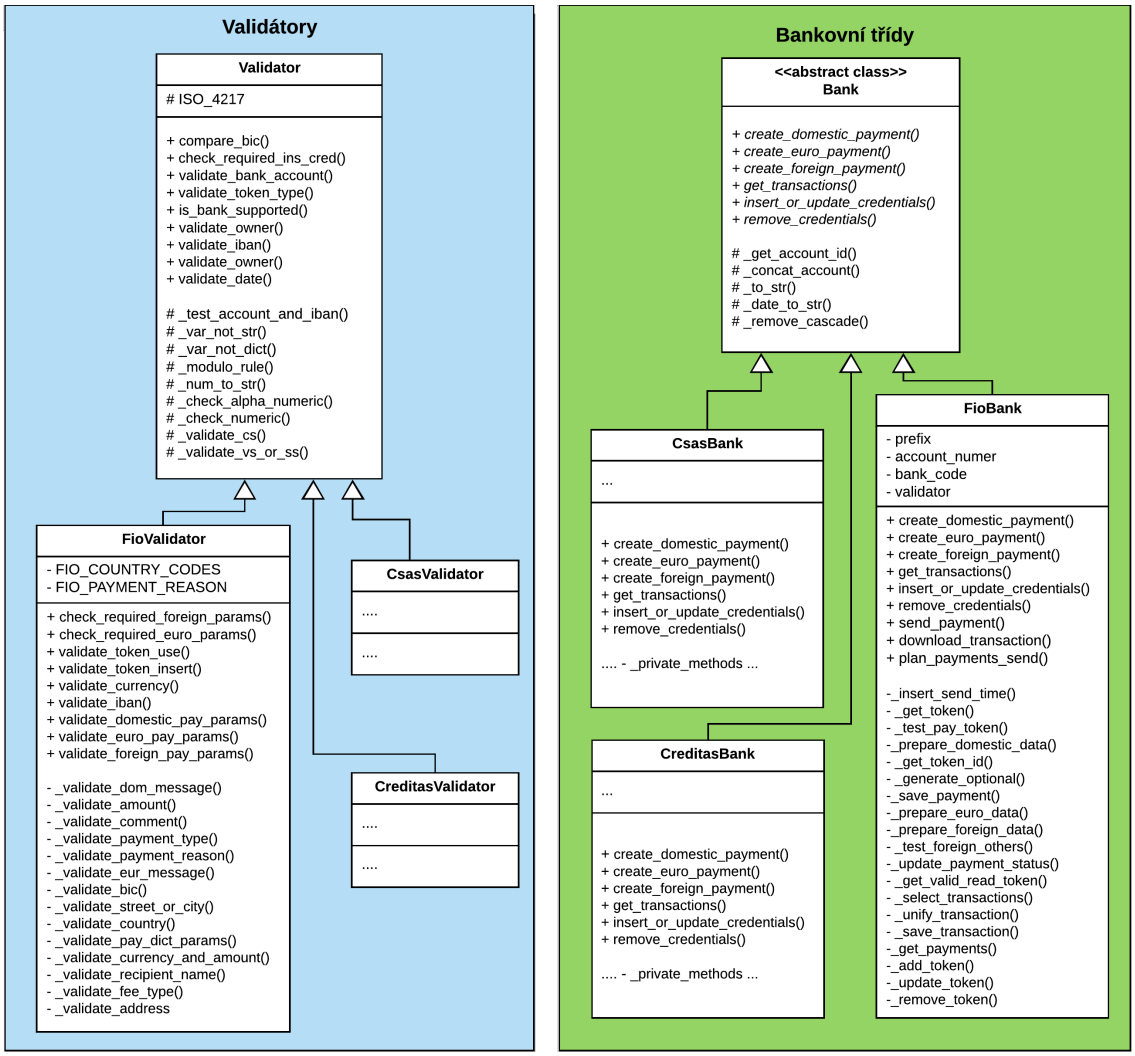
[29] Česká národní banka: *Upozornění pro veřejnost k nové právní úpravě v oblasti poskytování platebních služeb*. [Online; navštíveno 21.01.2018].
URL https://www.cnb.cz/cs/dohled_financni_trh/vykon_dohledu/upozorneni_pro_verejnost/20180104_upozorneni_k_nove_pravni_uprave_v_oblasti_poskytovani_platebnich_sluzeb.html

[30] Česká republika: *Zákon č. 370/2017 o platebním styku*. Sbírka zákonů České republiky, 2017, ISSN: 1211-1244.
URL http://aplikace.mvcr.cz/sbirka-zakonu/ViewFile.aspx?type=c&id=38322

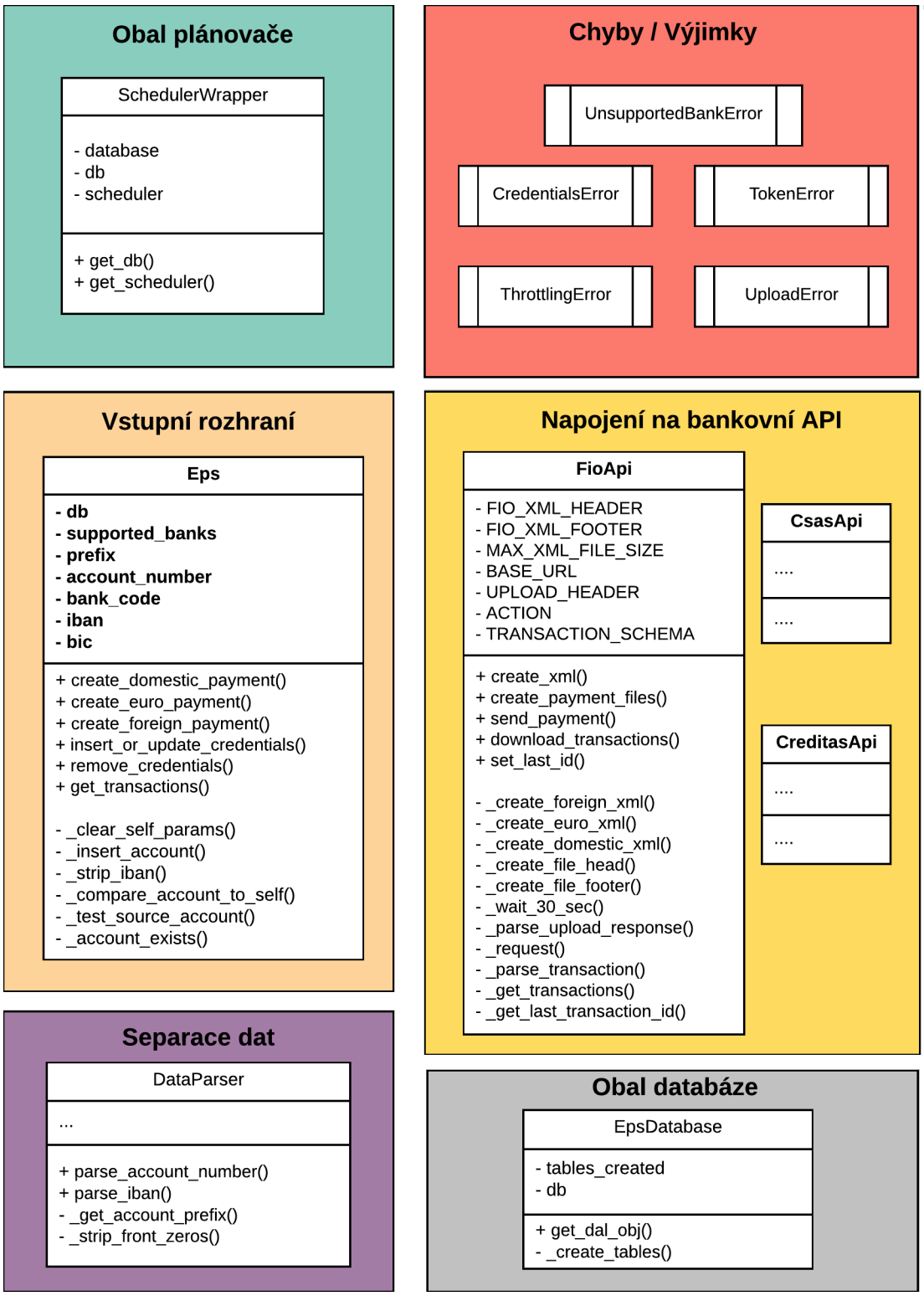
Přílohy

Příloha A

Diagram tříd



Obrázek A.1: První část diagramu tříd programu EPS. (Zdroj: vlastní)



Obrázek A.2: Druhá část diagramu tříd programu EPS. (Zdroj: vlastní)

Příloha B

Obsah přiloženého paměťového média

```
/
├── web2py/ ..... Kompletní zdrojové soubory rámce web2py.
│   ├── applications/
│   │   ├── eps/ ..... Zdrojové soubory bankovního modulu EPS.
│   │   │   ├── modules/
│   │   │   │   ├── eps.py ..... Veřejné rozhraní programu.
│   │   │   │   └── ...
│   │   │   ├── private/
│   │   │   │   ├── db_credentials.txt ..... Konfigurace přístupových údajů k databázi.
│   │   │   │   ├── examples.py ..... Příklady použití EPS při samostatném běhu.
│   │   │   │   └── tests/
│   │   │   │       └── run_tests.py ..... Spuštění automatických testů.
│   │   │   └── ...
│   │   └── import_tester/
│   │       ├── private/
│   │       │   └── examples.py ..... Příklady použití modulu EPS z jiné aplikace.
│   │       └── ...
│   ├── logs/
│   │   └── eps.log ..... Soubor obsahující logy.
│   ├── web2py.py ..... Hlavní spouštěcí skript web2py rámce.
│   └── ...
├── latex/ ..... Zdrojové soubory technické zprávy psané v LaTeX.
└── xjudap00-psd2.pdf ..... Elektronická verze této práce.
```

Příloha C

Manuál

V první řadě je nutné zajistit všechny technologické prerekvizity:

- Interpret jazyka Python ve verzi 2.7.x.
- Knihovna `requests` pro Python verzi 2.7.[1]
- Webový server podporovaný rámcem *web2py*. Např. Apache ve verzi 2.4.28.
- Databázový systém PostgreSQL minimálně ve verzi 9.

Nastavení rámce *web2py* nevyžaduje žádnou dodatečnou konfiguraci. Systém sám rozpozná, zda je nainstalován potřebný software, který případně i nakonfiguruje.

C.1 Spuštění aplikace

K úspěšnému spuštění programu je nutné splnit následující kroky:

1) Vložení přístupových údajů k vytvořené databázi. (soubor `db_credentials.txt`)
2) Prvotní spuštění EPS, které otestuje připojení k databázi a vytvoří potřebné tabulky a jejich vazby. Všechny skripty se zapínají z kořenového adresáře `/web2py`. Pro tvorbu databáze slouží příkaz:
   ```
   $ python ./web2py.py --nogui --password=X -M --shell=eps
   --run=./applications/eps/private/create_db.py
   ```
3) Po úspěšném vytvoření databázového schématu je nutné upravit konstruktor třídy `EpsDatabase`, který se nachází v souboru ./applications/eps/modules/db.py. Změna se týká jednoho řádku kódu a je komentována v tomto souboru.
4) Následně se doporučuje spustit automatizované testy, které ověří správnou konfiguraci programového rámce. Automatizované testy po svém dokončení čistí kompletní obsah databáze. Proto je nutné je spustit hned na počátku. Později je lze spustit pouze nad testovací databází. Nikdy je však nezapínejte přímo nad produkčními daty. Testy se spouští příkazem:
   ```
   $ python ./web2py.py --nogui --password=X -M --shell=eps
   --run=./applications/eps/private/tests/run_tests.py
   ```
5) Pro automatizované úlohy je nutné na pozadí spustit proces pro plánování událostí:
   ```
   $ python ./web2py.py --nogui --password=X --scheduler=eps -X &
   ```

[1] http://docs.python-requests.org/en/master/

C.2 Návod k použití

Pro předvedení funkčnosti jsou na paměťovém médiu připraveny dva skripty. Cesta k umístění těchto souborů je:

- /web2py/applications/eps/private/examples.py
- /web2py/applications/import_tester/private/examples.py

Soubory obsahují předem připravené bloky kódu, které stačí odkomentovat a upravit. Důležitým krokem je nahrazení vstupních parametrů pravými přístupovými údaji k bakovnímu účtu. Následně lze dle vzoru upravovat parametry nahrávaných plateb nebo vypisovat obsah získané transakční historie.

První skript se spouští v samostatně běžícím prostředí aplikace `EPS`. Ke spuštění zadaných úkolů slouží příkaz:

```
$ python ./web2py.py --nogui --password=X --shell=eps
--run=./applications/eps/private/examples.py
```

Druhý soubor je spuštěn v prostředí aplikace `import_tester`. Demonstruje tak provoz importovaného modulu z jiné *web2py* aplikace. Skript se zapíná příkazem:

```
$ python ./web2py.py --nogui --password=X --shell=import_tester
--run=./applications/import_tester/private/examples.py
```