Jihočeská univerzita v Českých Budějovicích
Přírodovědecká fakulta

Bakalářská práce

2019 **Pavel Máca**

Jihočeská univerzita v Českých Budějovicích
Přírodovědecká fakulta

SDK v PHP pro API v otevřeném bankovnictví

Bakalářská práce

Pavel Máca

Vedoucí práce: PhDr. Milan Novák, Ph.D.

České Budějovice 2019

Bibliografické údaje

Máca, P., 2019: SDK v PHP pro API v otevřeném bankovnictví. [SDK in PHP for the Open banking API. Bc. Thesis, in Czech.] – 53 p., Faculty of Science, University of South Bohemia, České Budějovice, Czech Republic.

Anotace

Tato práce se zabývá tvorbou PHP knihovny pro API dle českého standardu pro otevřené bankovnictví. Popisuje standard, vydaný Českou národní bankou pro otevřené bankovnictví, reagující na směrnici evropské unie o platebním styku (PSD2). Zkoumá požadavky na knihovnu pracující s REST API a v praktické části obsahuje návrh a popis implementace vytvořené knihovny. Tato knihovna je určena k použití v aplikacích, které chtějí přistupovat k finančním datům svých uživatelů.

Abstract

This bachelor‘s thesis focuses on the creation of a library in PHP language for the API specified by czech standard for the Open banking. First part is targeted to analyse the czech standard in conjunction with the european Payment Service Directive, known as the PSD2. Next part describe requirements of the libraly for consuming REST API. Practical part contains creation of the library and it’s usage. This library is designed for use in a financial application and managing user financial informataion and payments.

Prohlášení

Prohlašuji, že svoji bakalářskou práci jsem vypracoval samostatně pouze s použitím pramenů a literatury uvedených v seznamu citované literatury.

Prohlašuji, že v souladu s § 47b zákona č. 111/1998 Sb. v platném znění souhlasím se zveřejněním své bakalářské práce, a to v nezkrácené podobě elektronickou cestou ve veřejně přístupné části databáze STAG provozované Jihočeskou univerzitou v Českých Budějovicích na jejích internetových stránkách, a to se zachováním mého autorského práva k odevzdanému textu této kvalifikační práce. Souhlasím dále s tím, aby toutéž elektronickou cestou byly v souladu s uvedeným ustanovením zákona č. 111/1998 Sb. zveřejněny posudky školitele a oponentů práce i záznam o průběhu a výsledku obhajoby kvalifikační práce. Rovněž souhlasím s porovnáním textu mé kvalifikační práce s databází kvalifikačních prací Theses.cz provozovanou Národním registrem vysokoškolských kvalifikačních prací a systémem na odhalování plagiátů.

V Českých Budějovicích dne 9. 12. 2019 Pavel Máca

Poděkování

Za podporu děkuji svým přátelům a rodině.

Obsah

1 Úvod

Automatizace a data jsou dnes velmi důležitou součástí moderních technologií. Větší dostupnost a elektronizace se nyní dostává i do světa finančních služeb. Transakce mezi finančními institucemi, které trvají minuty, namísto dnů, centralizovaná správa financí, analytika finančních transakcí, nástroje umožňující hlubší integraci mezi různými systémy.

Tato práce je určena pro vývoj aplikací, které chtějí mít možnost přistupovat k finančním datům svých uživatelů. Číst jejich transakční historii a zprostředkovaně vytvářet nové transakce.

Může se jednat o aplikaci platební brány, analýzu osobních financí. Případně i univerzální internetové bankovnictví, kde pod jedním bezpečným přihlášením budou k dispozici všechny finanční prostředky a přehledy uživatele. V neposlední řadě může jít o chytré účetnictví s automatickým importem transakcí, možností úhrad přijatých dokladů a kontrolu úhrad vystavených dokladů.

1.1 Cíle Práce

Cílem je vytvořit a popsat tvorbu objektové knihovny, které umožní přístup k bankovním účtům českých bank. Knihovna bude podporovat snadné rozšíření o další bankovní API. Budou dodržovány poslední trendy ve vývoji webových aplikací.

Pro naplnění hlavního cíle práce je nezbytné splnit několik dílčích úkolů:

- Analyzování pojem otevřené bankovnictví
- Analyzování podobných řešení a knihoven
- Specifikace funkční a nefunkční požadavků
- Vytvoření scénářů použití
- Návrh architektury knihovny
- Implementace knihovnu
- Vytvoření příkladů použití
- Testování knihovny
- Publikování knihovny

1.2 Metody práce

Práce je zpracována formou souhrnné analýzy aktuální situace v oblasti otevřeného bankovnictví. Budou shrnuty poznatky, termíny z této oblasti a související technologie.

Bude provedena analýza standardů, které udávají strukturu služeb. Jejich požadavky na technologie a funkční řešení. Budou porovnány současné implementace standardu a již hotových řešení. Na základě toho bude vytvořen souhrn softwarových a uživatelských požadavků.

V praktické části bude, dle analýzy a stanovených požadavků vytvořen návrh knihovny. Návrh bude rozdělen na technologickou část a strukturální. Technologická část zajistí kompatibilitu s ostatními aplikacemi, strukturální rozšiřitelnost a použitelnost.

Součástí budou popsané procesy, které vyplynou z funkčních požadavků a analýzy standardu.

Podle vytvořeného návrhu dojde k implementaci knihovny, která bude splňovat stanovené požadavky. Součástí bude také popis funkcí a její použití.

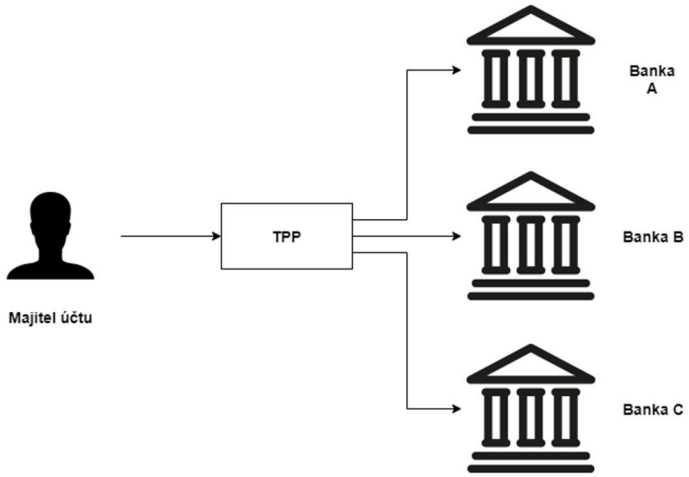
Na závěr dojde k otestování knihovny a vytvoří se příklady použití.

2 Analýza

2.1 Otevřené bankovnictví

Dne 16. 11. 2015 Evropská unie přijala směrnici o platebních službách č. 2015/2366 [1] pod anglický názvem Payment Service Directive, který je často označována pod zkratkou PSD2. Směrnice upravuje platební služby na vnitřním trhu EU. Vznikla díky potřebě přizpůsobit se novým rozvíjejícím informačním technologiím a službám. Do české legislativy byla směrnice implementována zákonem č. 370/2017 Sb., o platebním styku [2] (ZoPS).

Směrnice otevírá některé, dosud jen uzavřené bankovní služby třetím stranám. Třetí stranou je zde myšlen subjekt, jiný než bankovní instituce a příjemce bankovních služeb. Mezi tyto služby patří poskytování informací o bankovních účtech, historie transakcí a inicializace plateb pomocí prostředníka. Tyto služby mají být nově dostupné elektronicky přes aplikační rozhraní API (Obrázek 1).



Obrázek 1 - Komunikace prostřednictvím TPP

Služby nejsou dostupné přímo koncovým uživatelům a klientům bankovních služeb, nýbrž zákon definuje tzv. Poskytovatele služeb třetích stran, anglicky „Third party provider“ (TPP). Subjekt se stává TPP na základě splnění kritérií stanových v zákoně a udělení licence v jednom ze členských států EU. V rámci České republiky uděluje tuto licenci Česká národní banka.

TPP může přistupovat k datům bankovních služeb pouze po udělení oprávnění od uživatele bankovních služeb. Služba pro inicializace plateb vyžaduje autorizaci uživatele při každém zadání příkazu k úhradě.

Bankovní instituce se stávají TPP automaticky. Pro ně to znamená menší formální zátěž, jelikož samy o sobě musejí splňovat přísná kritéria pro získání licence k poskytování finančních služeb. Díky tomu mohou dříve pracovat s těmito novými elektronickými zdroji data a poskytovat svým uživatelům nové typy služeb.

2.1.1 Český standard pro Open banking

V zákoně není specifikován způsob komunikace a formát poskytovaných dat mezi bankovními subjekty a TPP. Z toho důvodu vznikl Český standard pro Open banking [3] v rámci spolupráce České bankovní asociace. Cílem standardu je stanovit pravidla této komunikace.

Standard popisuje komunikace pomocí protokolu HTTP ve formátu REST služeb a metody ověření TPP. Služby se dělí do třech hlavních částí.

- Poskytnutí informací o účtu, anglicky „Account Information Provider“ (AISP), viz. ZoPS § 41 a § 140
- Nepřímé podání platebního příkazu, anglicky „Payment Initiation Service Provider“ (PISP), viz. ZoPS § 161, § 162 a § 140
- Potvrzení o zůstatku peněžních prostředků, anglicky „Card Issuing Service Provider“ (CISP), viz. ZoPS § 178

Standard zachovává vysoký stupeň univerzálnosti, tím že specifikuje volitelná pole. Ty mohou využít jednotlivé bankovní subjekty podle potřeby.

Nejnovější verze standardu je 3.0, který byl vydán k datu 1. 4. 2019 a jeho účinnost platí od 1. 1. 2020. V roce 2019 je platný standard ve verzi 2.0, která byl vydán 21. 12. 2018 a je brán jako podklad pro tuto práci.

Technický popis

Standard vyžaduje komunikace pomocí protokolu HTTP 1.1 [4] nebo HTTP 2.0 [5]. Architektura rozhraní je založena na návrhu REST API. Data jsou během přenosu ve formátu JSON. Prázdné, nebo nevyplněné hodnoty jsou přenášeny pomocí hodnoty `null`. Prázdné kolekce jsou přenášeny jako prázdná pole.

Každý požadavek musí obsahovat hlavičku `Content-Type: application/json`. Výchozí kódování pro přenos je UTF-8 a lze jej změnit pomocí hlavičky Content-Type v HTTP požadavku.

Pro práci se ZD jsou v rámci HTTP protokolu používány metody GET, POST, PUT a DELETE. Tyto metody umožnují získávat informace, upravovat a případně je i mazat.

Změny ve standardu jsou řešeny pomocí číselného verzování, případně s použitím prefixu, např. „v1“. Verze je také zahrnuta v URI, která identifikuje jednotlivé ZD. Napříč jednotlivými ZD lze používat různé verze standardu.

Filtrování, řazení a stránkování dat

Pro vyhledávání požadovaných dat v ZD se používá velmi často filtrování, řazení a stránkování dat.

Standard stanovuje filtrování, řazení a stránkování dat, pomocí parametrů v URL dotazu.

Příkladem je ZD v AISP, kde lze přefiltrovat seznam transakcí na bankovním účtu pomocí parametrů v GET požadavku `fromDate` a `toDate`. Tím se získají pouze transakce v požadovaném období.

Data mohou být seřazena pomocí GET parametrů `sort` a `order`. Parametry obsahují čárkou oddělené názvy datových polí a směr řazení. Seznam podporovaných polí je závislý na konkrétním ZD.

Stránkování dat je řešeno pomocí GET parametrů `page` pro číslo stránky a `size`, pro požadovaný počet záznamů na stránce. Pokud není velikost stránky určena, vrátí se všechny záznamy.

Stránkovaná data obsahují tyto parametry:

Parametr	Povinný	Funkce
pageNumber	Ano	Číslo aktuální stránky
pageCount	Ano	Celkový počet stránek
nextPage	Ne	Číslo následující stránky. Pokud je aktuální stránka poslední, hodnota je `null`, nebo není parametr uveden.
pageSize	Ano	Velikost stránky
page	Ano	Obsahuje kolekci záznamů

Chybové stavy

Standard specifikuje dva typy chybových stavů. Jedná se o chyby způsobené neplatným požadavkem, nebo chyby způsobené na straně serveru.

O chybě je klient informován pomocí HTTP hlavičky se stavovým kódem. Stavové kódy odpovídají svým významem specifikaci HTTP a ve standardu je uveden jejich hlubší význam, viz. následující tabulka.

Kód	**Význam**	**Účel**
200	OK	Správná odpověď s obsahem
201	OK	Správná odpověď = nový záznam vytvořen
204	OK	Správná odpověď = záznam byl smazán
304	NOT MODIFIED	Zdroj beze změny = je možné použít mezipaměť
400	BAD REQUEST	Jedná se o neplatný dotaz, na který není možné odpovědět. Např. v případě, že JSON obsah není pro tento zdroj validní.
401	UNAUTHORIZED	Pro provedení dotazu je potřeba autentizace uživatele
403	FORBIDDEN	Přístup k požadovanému zdroji není udělen, resp. pro daného uživatele není možný.
404	NOT FOUND	Požadovaný objekt/stránka neexistuje, resp. nebyl/a nalezen/a
415	UNSUPPORTED MEDIA TYPE	Požadavek na nepodporovaný typ přenosu (např. ve vztahu k hlavičkám Accept a Accept-Language)
422	UNPROCESSABLE ENTITY	Tato chyba může být použita v případě, že není možné zpracovat požadovaný objekt, nebo pokud chybí povinný parametr dotazu.
500	INTERNAL SERVER ERROR	Chyba serveru, která může být vyvolána technickými problémy, anebo v případě neošetřeného chybového stavu.
501	NOT IMPLEMENTED	Může být použito v případě, že server nepodporuje požadovanou operaci.

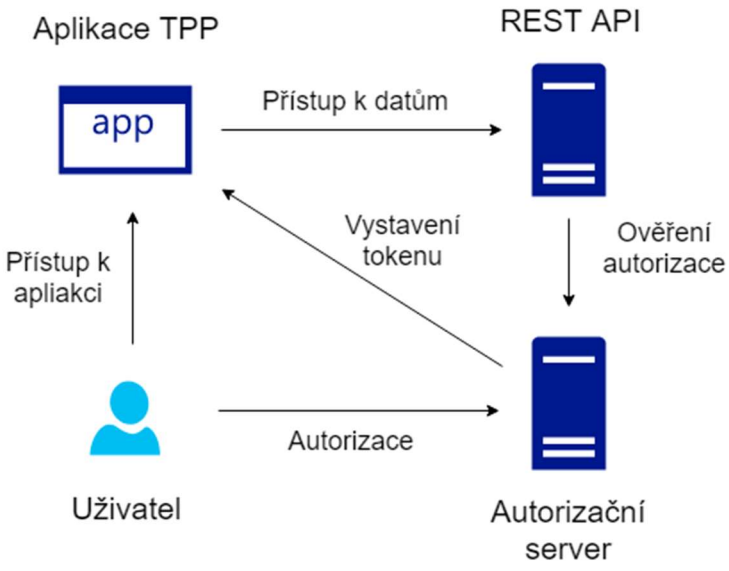
Všechny ošetřené chybové stavy na straně serveru vrací data ve formátu JSON s informacemi o chybě. V datech je obsažen například seznam chybně uvedený parametrů požadavku, nebo informace o nenalezení požadovaného záznamu. Data spojené s chybovým stavem obsahují také validační chyby u požadavků jako je inicializace platby.

Autentizace a autorizace požadavků

Autentizaci a autorizaci probíhá pomocí standardu OAuth2 [6], který je založený na systému tokenů (Obrázek 2). Pro přístup k API se aplikace registruje u autentizačního serveru. Vygeneruje se privátní a veřejný klíč, kterým si identifikuje při požadavku na autorizaci a nastaví se zpětná URL do aplikace, na kterou je uživatel přesměrován po dokončení autorizace.

Postup autorizace zahrnuje vygenerování odkazu na autorizační a identifikační systém banky s požadavkem na autorizaci konkrétních služeb. Lze například žádat pouze o přístup k informacím o bankovním účtu. Uživatel je přesměrován na vygenerovanou URL adresu autorizačního serveru. Po přihlášení a udělení přístupu TPP je uživatel přesměrován zpět na návratovou URL aplikace třetí strany. Ta na základě kódu obsaženého v parametru návratové URL získá obnovovací token nazvaný „refresh_token“. Tento token má obvykle omezenou platnost. Po skončení platnosti ho buďto lze obnovit, nebo zažádat o nový opakovaným procesem autorizace.

Pomocí obnovovacího tokenu se vygeneruje dočasný token určený pro autentizaci a autorizaci jednotlivých HTTP požadavků.



Obrázek 2 - Autorizace TPP

Služba informace o účtu

Definována směrnicí Evropské unie jako „Account Information Provider“ (AISP). Poskytuje seznam účtů klienta s detaily jako je číslo účtu v národním [7] i mezinárodním formátu IBAN [8]. Aktuální zůstatek na účtu, včetně povoleného debetu. Jednoznačný identifikátor účtu v rámci banky pro další požadavky v rámci standardu a transakční historii.

Seznam účtů je stránkovaný a umožňuje třídění.

Služba inicializace plateb

Definována směrnicí jako „Payment Initiation Service Provider“ (PISP). Umožňuje vytváření nových plateb a poskytuje rozhraní pro autorizaci. Autorizace je prováděna na straně konkrétního bankovního subjektu a umožňuje zachování stávající metody autentizace přes běžné internetové bankovnictví jako je ověření SMS kódem, certifikátem na čipové kartě či heslem. Platby mohou být tuzemské, SEPA, mezinárodní v rámci EHP i mezinárodní mimo EHP.

Ve verzi standardu 3.0 přibývá vytváření trvalých příkazů k úhradě a hromadných plateb, tedy možnost inicializovat a autorizovat více plateb současně. Jejich autorizace probíhá stejně jako u běžných plateb.

Autorizace plateb probíhá v několika krocích a je závislá na konkrétní implementaci. Prvním krokem je vygenerování autorizačního identifikátoru. Ten je vygenerován vytvořením požadavku, který obsahuje informaci o požadované platbě. Součástí požadavku je také volba způsobu autorizace platby, např. pomocí SMS, přihlášením do internetového bankovnictví, push-up notifikací.

Druhým volitelným krokem je získání stavu konkrétního autorizačního scénáře.

Třetím a hlavním krokem je zahájení samotné autorizace. Autorizace se zahájí požadavkem na zdroj označený jako inicializace platby. V odpovědi je poskytnuta URL adresa s parametry a detaily, na kterou je následně uživatel přesměrován. Uživatel provede autorizace dle zvoleného scénáře a následně je přesměrován zpět do aplikace TPP.

Posledním krokem je ověření, zda autorizace proběhla úspěšně ze strany TPP.

Služba ověření dostatku prostředků

Definována směrnicí jako „Card Issuing Service Provider“ (CISP). Umožňuje ověření zůstatku na bankovním účtu, nebo platební kartě. Odpověď obsahuje pouze informaci, zda je možné transakci provést, bez jiných detailů o bankovním účtu, nebo platební kartě.

V standardu je popsána jako součást služby PISP.

2.1.2 Realizace českých bank

Český standard pro Open banking je pouze doporučenou příručkou pro implementaci požadovaných služeb dle směrnice PSD2. Jeho implementace je v českém prostředí pojata volně.

Česká spořitelna

Nejpřesnější implementaci najdeme u České spořitelny, která je součástí skupiny Erste Group Bank AG a aktivně se podílí na vytváření českého standardu. Aktuálně ve verzi v1 plně implementuje AISP, CISP a částečně PISP [9]. U služby PISP plně podporuje zahraniční platby a jejich autentizaci. Nepodporuje získání stavu autorizace u inicializované platby. Podporovaná verze standardu je 2.0. Poskytuje testovací prostředí, obsahující identifikační sever a také možnost testování autorizace pro aplikaci třetí strany. Testovací prostředí však neobsahuje validaci požadavků a vrací statické odpovědi na každý HTTP požadavek.

Komerční banka

Komerční banka ve velké míře kopíruje standard [10]. Podporuje verzi standardu 2.0. u služby PISP nicméně podporuje pouze tuzemské platby. Vývojářům poskytuje testovací prostředí bez identifikačního serveru. Testovací prostředí vyžaduje použití certifikátu, který banka poskytuje po registraci na testovací portál a odeslání požadavku emailem pouze právnickým osobám.

Token pro komunikaci je nutné vygenerovat ručně v nastavení aplikace. Není možné otestovat celý autorizační proces uživatele s přesměrováním na identifikační portál a zpět do aplikace.

Equa Bank

Equa Bank podporuje všechny tři služby, AISP, CISP i PISP. Každou službu však poskytuje na mírně odlišné adrese se samostatným verzováním. Je možné používat starší verzi služeb,

pokud jsou stále dostupné, nebo verzi z adresy zdroje vynechat a používat aktuální produkční verzi.

Podporována je verze standardu 2.0 s vlastní iterací, která je současně na verzi 2.0.5 případně 2.0.6 u některých služeb. U služby PISP jsou podporovány pouze tuzemské platby, stav transakce a její autorizace.

Není poskytováno identifikační prostředí pro testování uživatelské autorizace. Je možné použít produkční prostředí s existujícím účtem a autorizovat i testovací aplikaci. Taková aplikace je stále na testovacím prostředí a nelze získat autentizační token z produkce. Testovací prostředí vrací statická data bez validace vstupních požadavků. Pro účely testování je k dispozici testovací autentizační server. Certifikát pro testovací prostředí není potřeba.

Ostatní banky

Mezi další bankovní subjekty, které podporují otevřené bankovnictví a mají dostupnou API jsou Československá obchodní banka, Creditas a Raiffeisenbank.

ING Bank má vlastní strukturu API podobnou českému standardu. Je výrazně zjednodušena a neobsahuje všechny datová pole, jak udává český standard.

Moneta Money Bank poskytuje informace o API pouze subjektům s licencí PSD2.

Fio banka již několik let poskytuje vlastní API pro historii transakcí a inicializaci plateb. Struktura však neodpovídá PSD2 a umožňuje přístup aplikacím třetích stran bez licence ČNB. K přístupu stačí pouze uživatelsky vygenerovaný token v internetovém bankovnictví.

2.2 REST API

REST (Representational State Transfer), je architektura pro komunikaci v distribuovatelných systémech orientovaná na zdroje. Prvně jí popsal Roy Fielding ve své disertační práci „Architectural Styles and the Design of Network-based Software Architectures“ [11]. Dnes ji můžeme vidět zejména u webových služeb, kde slouží k přenosu dat.

Služba využívající REST architekturu se někdy nazývá také „RESTful service“. Architektura umožňuje komunikovat mezi dvěma systémy pomocí předem definovaných bezstavových operací. Proto, abychom mohli rozhraní systému nazývat RESTfull, musí splňovat několik základních kritérií.

2.2.1 Principy REST architektury

Komunikace

Komunikace funguje na principu klient – server (Obrázek 3). Klient odešle požadavek na server, ten přijme požadavek, zpracuje ho a vrátí odpověď. Jde o bezstavovou komunikaci, kde není potřeba udržovat relaci mezi koncovými body. To snižuje paměťovou a výpočetní náročnost a umožňuje lepší škálování.

Obrázek 3 - REST komunikace

Bezstavovost

Každý požadavek musí být úplný, aby byl srozumitelný a mohl být serverem zpracován. Není možné se spoléhat na server, který by mohl v kontextu předchozích požadavků vykonat požadavek jiným způsobem. Mezi klientem a serverem neexistuje žádná relace. Všechny informace jsou předány v jednom požadavku.

Mezipaměť

Mezipaměť je u klienta paměť, která slouží k ukládání předchozích odpovědí.

V odpovědi na požadavek musí být označení, pokud ji má klient uložit do mezipaměti spolu s délkou platnosti dat v odpovědi. Klient tak nemusí znovu odesílat shodný požadavek na server. Tím se snižuje zatížení komunikačního rozhraní, výpočetní výkon a další zdroje serveru. Klient však musí počítat s rizikem používání již neaktuálních dat.

Jednotné rozhraní

Použitím principů softwarového inženýrství k vývoji rozhraní jednotlivých komponent aplikace dosáhneme zjednodušení a předvídatelnosti interakcí. K tomu je potřeba splnit několik podmínek pro vytvoření takových komponent. Jedná se o identifikaci komponent, manipulaci se zdroji pomocí jejich vlastní reprezentace, samo vysvětlující požadavky a odpovědi, obsah je reprezentací stavu.

Vícevrstvý systém

Vícevrstvý systém umožňuje oddělit jednotlivé komponenty. Každá komponenta vidí jen ty, se kterými provádí interakce.

2.2.2 Zdroj

Hlavní bodem REST architektury jsou zdroje [12]. Veškeré informace, které mohou být pojmenovány, mohou být zdrojem. Například dokument, obrázek, stav měření, karta zákazníka, seznam hodnot.

Pro jejich identifikaci se používá URI. Identifikace by měla být intuitivní a jasně popisující daný zdroj.

Aktuální stav zdroje v daném čase je reprezentován daty, metadaty a odkazy Hypermedii [13], které umožní přechody do dalšího stavu.

Adresování

Klient odesílá požadavek na tzv. zdroj, který je identifikován pomocí URI. Nejčastěji se volí taková hierarchie zdrojů a URI, která je pro uživatele intuitivní a čitelná.

Formát dat

Zdroj může být reprezentován více formáty jako je JSON, XML, ale také se můžeme setkat s PDF či obrázky. Každý typ formátu by měl představovat stejná data.

Následují příklady odpovědí ve formátu JSON a XML se stejnými daty.

```
{"employees":[
  { "firstName":"John", "lastName":"Doe" },
  { "firstName":"Anna", "lastName":"Smith" },
  { "firstName":"Peter", "lastName":"Jones" }
]}
```

Kód 1 - Odpověď ve formátu JSON

```
<employees>
  <employee>
    <firstName>John</firstName> <lastName>Doe</lastName>
  </employee>
  <employee>
    <firstName>Anna</firstName> <lastName>Smith</lastName>
  </employee>
  <employee>
    <firstName>Peter</firstName> <lastName>Jones</lastName>
  </employee>
</employees>
```

Kód 2 - Odpověď ve formátu XML

CRUD

Jde o zkratku založenou na čtyřech anglických slovech (create, read, update, delete). Popisuje základní operace REST architektury nad zdroji. Tyto operace jsou implementovány pomocí metod HTTP protokolu. [14]

Operace	**Význam**	**Metoda HTTP**
CREATE	Vytvoření nového záznamu.	POST
READ	Čtení záznamu, nebo seznamu záznamů.	GET
UPDATE	Aktualizace celého záznamu, nebo jeho části.	PUT, PATCH
DELETE	Smazání záznamu.	DELETE

2.2.3 Podobné architektury

SOAP je zkratka „Simple object access protocol“. Protokol umožňuje komunikaci mezi distribuovanými systémy. Na rozdíl od REST architektury je založena na odesílání zpráv ve formě XML. Obsahem zprávy je prováděná akce, ať jde o vytváření dat nebo čtení. Pro komunikaci používá také HTTP protokol, ale lze použít i SMTP. Datový formát určuje specifikace WSDL. Oproti REST architektuře vyžaduje větší výpočetní výkon, jelikož data jsou pevně validována pomocí WSDL. Obecně požadavky i odpovědi mají větší bitovou velikost.

GraphQL [15] je další způsob komunikace v rámci API. Bylo vytvořeno firmou Facebook, Inc. Jedná se o dotazovací jazyk, podobný SQL. Data jsou definována objektově. Používá HTTP protokol a data jsou ve formátu JSON.

2.2.4 Komunikační protokol HTTP

Pro komunikaci REST rozhraní je nejrozšířenější HTTP protokol. REST API architektura je sice nezávislá na komunikačním protokolu, ale vzhledem k jejímu využití v oblasti webových technologií se dnes v drtivé většině spojuje právě s HTTP protokolem.

Definice HTTP

Jedná se protokol v aplikační vrstvě sítového přenosu. Je specifikován mezinárodním standardem RFC 7230 [16]. Původně vznikl v evropské organizaci pro nukleární výzkum CERN, v týmu vedeném profesorem Tim Berners-Lee. Současně s ním vznik termín Word Wide Web známý pod zkratkou „WWW“. Původně obsahoval pouze metodu GET, která sloužila jako požadavek na server a odpověď byla vždy ve formátu HTML stránky. Současně existuje několik verzí protokolu. První verze HTTP 0.9 byla vydána v roce 1991. Současně nejrozšířenější je HTTP/1.1 používající TCP spojení a HTTP/2.0, které se zaměřuje mimo jiné na zabezpečení spojení. Připravuje se také verze HTTP/3.0, která používá UDP protokol pro komunikaci.

Zpráva s požadavkem

Prvním krokem HTTP komunikace je vytvoření požadavku „request“, který odesílá klient na server. Požadavek obsahuje URL adresu, verzi HTTP protokolu, hlavičky a obsah.

URL adresa je povinná, označuje adresu serveru a cílový zdroj včetně parametrů. Dle použité metody může obsahovat tělo, které je nositelem datových informací.

```
GET /sandbox/aisp/accounts/my/accounts HTTP/1.1
Host: api.equa.cz
Accept: application/json
Cache-Control: no-cache
Host: api.equa.cz
```

Kód 3 - HTTP Požadavek

Zpráva s odpovědí

Po zpracování požadavku na straně serveru, ale i v případě chybového stavu, server vrací odpověď. U ní je uveden stavový kód, který označuje, zda byl požadavek správně zpracován.

Dále v závislosti na stavovém kódu a požadavku, obsahuje odpověď tělo, které nese požadovaná data.

```
HTTP/1.1 200 OK
Date: Thu, 21 Nov 2019 14:26:13 GMT
Server: Apache
Last-Modified: Thu, 21 Nov 2019 14:26:13 GMT
Content-Length: 44
Connection: close
Cache-Control: no-cache
Content-Type: application/json; charset=UTF-8
```

Kód 4 - HTTP Odpověď

Stavové kódy odpovědi jsou označeny trojmístným číslem. Dělí se do následujících skupin:

- 1xx – informační
- 2xx – úspěšné zpracování požadavku
- 3xx – přesměrování, vyžaduje interakci klienta
- 4xx – chyba na straně klienta
- 5xx – chyba na straně serveru

Metody HTTP pro REST API

Metody HTTP určují akci, která se má nad daným zdroj provést.

- GET – žádost o zaslání dat
- HEAD – žádost o zaslání dat, ale odpověď obsahuje pouze hlavičky
- POST – odeslání dat
- DELETE – smazání dat
- CONNECT – vytvoření spojení, používané spolu s proxy
- OPTIONS – ověření dostupných metod daného zdroje
- TRACE – ověření, jak je požadavek zpracován na cestě k cílovému serveru
- PATCH – částečná modifikace dat

2.2.5 Autentizace

Použitím protokolu HTTP je možné dosáhnout ověření klienta několika způsoby.

HTTP Basic Authentication

Jedná se o základní autentizaci popsanou standardem HTTP. V hlavičce požadavku se nachází zakódované uživatelské jméno a heslo. Tento způsob je doporučený pouze

v případě, kdy je spolu s HTTP použito některé šifrování jako je TLS nebo SSL. Heslo není šifrované.

Bearer

Autentizace probíhá pomocí tokenů. Token lze získat přihlášením, nebo jeho vygenerování. Platnost tokenu může být časově omezena. V každém požadavku na REST API je token předávám v hlavičce s názvem „Authorization“. Oproti předchozí autorizaci jsou chráněné přihlašovací údaje, ale samotný token lze stále zneužít při zachycení požadavku. Proto by měla být komunikace také šifrovaná.

OAuth 2.0

Autentizace je postavená na předchozí metodě Bearer, ale specifikuje podrobněji získávání tokenu a řeší i autorizaci.

Nabízí řadu scénářů pro autentizaci, od webových aplikací, přes desktopové, mobilní telefony až po malé zařízení v domácnosti. [6]

Pro webové aplikace se nejčastěji používá scénář, kdy se na identifikačním serveru aplikace zaregistruje. K tomu je potřeba URL aplikace pro návrat po dokončení autentizace a vygenerování soukromého a veřejného klíč.

Identifikační server je zdroj dat, který slouží primárně k identifikaci uživatele a autorizaci. Může být implementován samostatně mimo datový server, který poskytuje REST API.

V prvním kroku projde uživatel autentizací. Uživatel je přesměrován na URL adresu vygenerovanou pomocí veřejného klíče aplikace na stranu identifikačního serveru, například přihlášení do internetového bankovnictví. Dále uživatel udělí přístup aplikaci a rozsah oprávnění. Po udělení autorizace se uživatel přesměruje zpět do aplikace na URL, která byla uvedena při její registraci pro přesměrování. Aplikace, získá klíč s omezenou životností, na jehož základě může zažádat o token z identifikačního serveru. Získaný token lze následně použít pro Bearer autentizaci.

Certifikát

Slouží jako dodatečný prostředek pro ověření identity. Vyžaduje použití zabezpečeného protokolu HTTP prostřednictvím TLS/SSL. Namísto ověřování identity serveru pomocí certifikátu, je při tomto typu ověření vyžadováno i ověření klienta certifikátem. Ověření certifikátem se používá při každém požadavku na ZD, který je takto chráněný.

2.3 Podobné knihovny a aplikace

V současné době nejsou volně dostupné žádné knihovny podporující otevřené bankovnictví, nebo český standard pro Open banking.

Některé dokumentace českých bank umožňují vygenerovat kód pro připojení do REST API, ale tento kód je specifický pro danou implementaci.

Existuje několik aplikací, které využívají otevřené bankovnictví.

saltedge.com

Jedná se o agregátor API pro široké spektrum bankovních institucí. Agregátor je vyvíjen anglickou společností Salt Edge Limited. Primárně podporuje banky z území EU a tedy PSD2. Pro přístup k datům je potřeba vlastnit certifikát a být TPP.

Výhodou tohoto řešení je jednotné rozhraní pro mnoho bankovních institucí s minimální údržbou. Nevýhodou je naopak zpoplatnění služby a závislost na prostředníkovi při získávání dat.

Trustly

Trustly nabízí možnost vytváření bankovních plateb v jednoduchém a bezpečném řešení. Jde o platební bránu využívající rozhraní PSD2 pro inicializace plateb. Určená je zejména pro komerční účely.

Spendee

Spendee je mobilní aplikace na správu osobních a rodinných financí. Aplikace využívá služby AISP pro stahování transakční historie na bankovních účtech.

Ověření identity fyzických osob

Některé společnosti využívají získanou licenci TPP a přístup k bankovním informacím jako prostředek ověření identity uživatelů. Jde především o loterijní společnosti. Mezi tyto společnosti patří níže uvedené.

- SYNOT TIP, a.s.
- Tipsport.net a.s.
- CHANCE a.s.
- SAZKA a.s.

2.4 Logický rámec

Metoda logického rámce je postup, který umožňuje navrhnout a uspořádat základní charakteristiky projektu ve vzájemných souvislostech. Uplatnění této metodiky je důležité nejen ve fázi přípravy projektu či programu, ale je i klíčovým nástrojem pro jeho implementaci a hodnocení. Je to postup, s jehož pomocí je možné stručně, přehledně a srozumitelně popsat projekt. [17]

Přínosy	**Objektivně měřitelné ukazatele**	**Zdroje a prostředky k ověření**	**Vnější předpoklady**
Hlavní cíl Možnost tvorby nových aplikací s přístupem k bankovním datům a platbám. Rozvoj stávajících finančních aplikací.	Rozšíření aplikací pro práci s více bankovními účty. Hlubší integrace finančních aplikací s bankovními službami.	Statistiky používání u nově vzniklých aplikací.	
Účel projektu Vytvoření knihovny pro otevřené bankovnictví.	Použití knihovny alespoň v pěti aplikacích. Rozvoj knihovny díky otevřenosti kódu.	Počet stažení knihovny pomocí statistik na github.com a packagist.org	Konkurenční implementace otevřeného bankovnictví. Zvolení nepopulární platformy pro aplikace zaměření na finanční služby. Špatná dohledatelnost knihovny.

Výstupy projektu Analýza otevřeného bankovnictví SDK knihovna Příklady použití Automatické testy	Knihovna umožní přístup díky jednotné implementaci do různých bankovních API. Snadné zprovoznění knihovny a příklady použití.	Připojení alespoň do dvou bankovních API. Zprovoznění demo aplikace pomocí příkladů použití.	Různá implementace bankami znemožní jednotné rozhraní SDK. Špatná čitelnost příkladů k použití, nečitelná implementace SDK.
Aktivity projektu 1. Analýza požadavků 2. Návrh řešení 3. Implementace obecné definice 4. Implementace konkrétních řešení 5. Vytvoření příkladů použití 6. Vytvoření testů	Publikovaná dokumentace otevřeného bankovnictví Implementace otevřeného bankovnictví českými bankami Dokumentace a přístup k testovacím prostředím konkrétních implementací	1. 01–02/2018 2. 02–04/2018 3. 09–12/2018 4. 01–02/2019 5. 10–11/2019 6. 11/2019	Dostatek časových prostředků.

2.5 Specifikace softwarových požadavků

Analýza softwarových požadavků napomáhá při rozhodování o potřebách, které se musí naplnit pro splnění stanoveného cíle projektu. Specifikuje základní funkce výstupu a prostředky pro jejich dosažení.

Prvním bodem je sběr požadavků, které se provádí pomocí komunikace se zákazníkem a uživateli. Následně se požadavky analyzují, vyloučí se protichůdné názory a jejich proveditelnost. Požadavky se dokumentují a slouží jako podklad pro daný projekt.

Tento projekt je vytvářen jako knihovna, která má za úkol umožnit ostatním aplikacím snadnou implementaci otevřeného bankovnictví. Požadavky na funkčnost vycházejí ze specifikace otevřeného bankovnictví a z požadavků na snadnou integraci knihovny v různých aplikacích.

2.5.1 Funkční požadavky

Na základě specifikace otevřeného bankovnictví a z něj vyplývajících uživatelských scénářů byly vymezeny funkční požadavky na knihovnu SDK.

Scénáře se zaměřují na integraci SDK knihovny do projektu, autentizaci do bankovní API, získání seznamu bankovních účtů, seznamu transakcí, vytvoření platby a napojení na více API současně.

- Knihovna umožní autentizaci
- Knihovna získá seznam bankovních účtů uživatele
- Knihovna získá seznam transakcí na konkrétním bankovním účtu
- Knihovna umožní vytvoření nové transakce
- Získaná data budou reprezentována objektově

2.5.2 Nefunkční požadavky

- Knihovna bude dostupná jako open-source
- Knihovna bude dostupná jako balíček programu Composer
- Vytvoření příkladů použití
- Vytvoření automatických testů
- Použití jazyka PHP
- Knihovna bude rozšiřitelná a přizpůsobitelná dle konkrétní implementace jednotlivých bank
- Zdrojový kód knihovny bude čitelný a okomentovaný

2.6 Uživatelské scénáře

Použití knihovny se odvíjí od základních uživatelských scénářů. Jedná se o scénáře, které popisují implementaci kódu ve finální aplikaci, či ukázce. V každé tabulce se nachází analýza scénáře, popis, funkční požadavky a předpoklady.

Název: Autentizace	**Priorita:** Nezbytná	**Pořadí implementace:** 1
Popis: Aplikace chce přistoupit k REST API otevřeného bankovnictví. V prvním kroku je nutný provést proces autentizace. Proběhne inicializace autentizačního objektu, která může zahrnovat specifické nastavení pro konkrétní autentizační server. Aplikace dále získá vlastními prostředky OAuth 2.0 token pro přístup k REST API. Získaný token je nastaven do instance autentizačního objektu. Volitelně lze nastavení certifikát pro přístup k REST API na produkční prostředí. Autentizační objekt je dále používán při tvorbě konektoru.		
Funkční požadavky: 1. Rozhraní obecného autentizačního objektu 2. Implementace obecného rozhraní dle českého standardu 3. Metoda pro nastavení OAuth 2.0 tokenu 4. Metoda pro nastavení certifikátu		
Předpoklady: • Aplikace obsahuje kód pro získání OAuth 2.0 tokenu		

Název: Standardní konektor	**Priorita:** Nezbytná	**Pořadí implementace:** 2

Popis:

Vytvoření objektu, reprezentujícího připojení ke konkrétní bankovní REST API. Konektor vyžaduje autentizační objekt. Poskytuje nastavení parametrů konkrétní REST API. Konektor může podporovat pouze některé služby z dostupných AISP, CISP a PSIP.

Sestaví HTTP požadavek, zachytí chybové stavy HTTP požadavků a vrací odpověď na požadavek ve formátu datového pole.

Funkční požadavky:

1. Obecné rozhraní základního konektoru
2. Obecné rozhraní pro konektory AISP, CISP, PISP
3. Implementace konektoru dle českého standardu
4. Předání autentizačního objektu jako parametr konstruktoru
5. Sestavení a zpracování HTTP požadavků
6. Zachycení chybových stavů
7. Návrat dat z HTTP odpovědi ve formátu pole

Předpoklady:

- Knihovna pro práci s HTTP požadavky „guzzlehttp/guzzle“

Název: Seznam bankovních účtů	**Priorita:** Nezbytná	**Pořadí implementace:** 3
Popis: Inicializace objektu pro práci se službou AISP. Předání konektoru v konstruktoru. Po zavolání metody `getAccountList` je vrácen objekt typu `AccountList`, který představuje stránkovaný seznam bankovních účtů. Seznam účtů obsahuje metody pro čtení velikosti a čísla aktuální stránky. Zavoláním metody `getAccountList` s parametry pro stránkování lze načíst další stránky seznamu.		
Funkční požadavky: 1. Rozhraní pro práci s AISP 2. Implementace rozhraní AISP dle českého standardu 3. Příjem konektoru podporující AISP rozhraní při inicializaci v konstruktoru 4. Metody pro získání seznamu bankovních účtů 5. Metody pro stránkování 6. Hydratace dat		
Předpoklady: • Připojení k internetu • Přístup k REST API • Knihovna pro práci s HTTP požadavky „guzzlehttp/guzzle“ • Knihovna „doctrine/annotations“ pro hydrataci dat		

2.7 Zásady vývoje

- Zdrojový kód bude formátování dle standardu PSR-2
- Konstrukce v rámci zdrojového kódu budou pojmenovány srozumitelně a budou respektovat specifikaci otevřeného bankovnictví
- Složité konstrukce budou obsahovat komentář
- Změny v knihovně, příkladech a testy budou evidovány verzovacím systému Git
- Dodržování principů DRY, KISS, SOLID

3 Design

3.1 Architektura knihovny

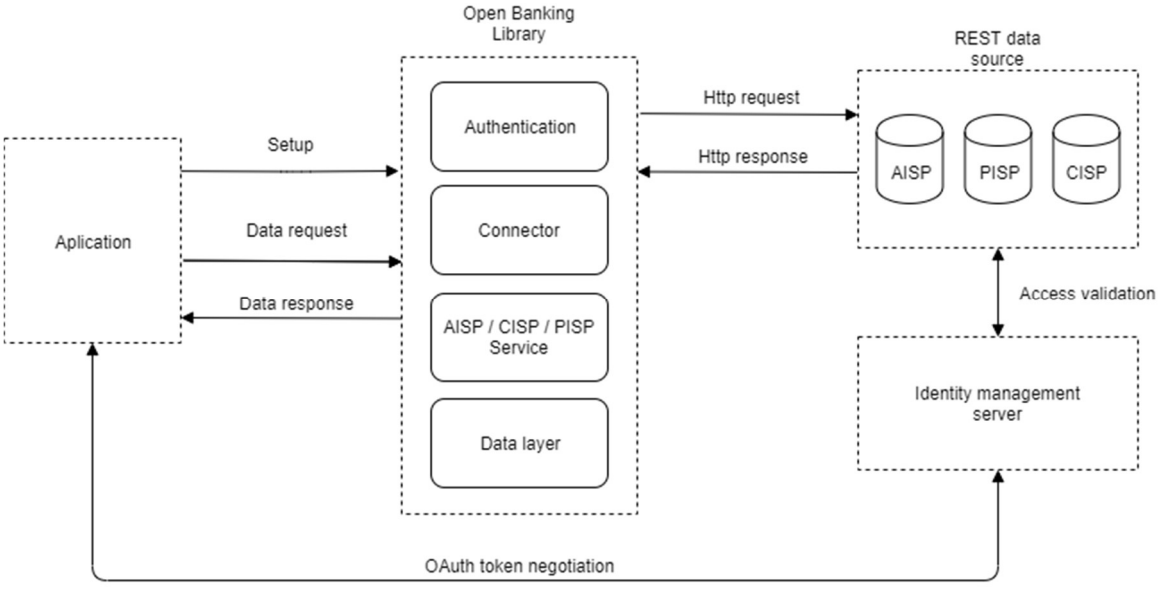
Návrh architektury je klíčový pro implementaci knihovny. Správně navržená architektura knihovny zajistí logické uspořádání objektových vrstev a rozšiřitelnost.

Během návrhu byl kladen důraz na tyto požadavky:

- Jednoduchá použitelnost
- Rozšiřitelnost a modularita
- Minimální závislost na dalších knihovnách
- Objektový přístup

Jednoduchost použití knihovny lze dosáhnout několika způsoby. Obsáhlými uživatelskými scénáři. Pojmenováním objektů, metod a jejich parametrů tak, aby odpovídaly svému účelu a použití. Výsledná implementace za použití kódu knihovny by měla být samo vysvětlující.

Rozšiřitelnosti a modularity bude dosaženo použitím SOLID principů. Mezi ně patří princip jedné odpovědnosti, princip otevřenosti a uzavřenosti, Liskovové princip zaměnitelnosti, princip oddělení rozhraní a princip obrácení závislostí. Většina z těchto principů byla při návrhu aplikována.



Obrázek 4 - Architektura knihovny

Na obrázku č. 4 je vyobrazena architektura knihovny a její funkce při práci s otevřeným bankovnictvím. Knihovna je rozdělena do čtyřech vrstev. Každá vrstva má svou zodpovědnost a je závislá na ostatních pouze pomocí základního rozhraní.

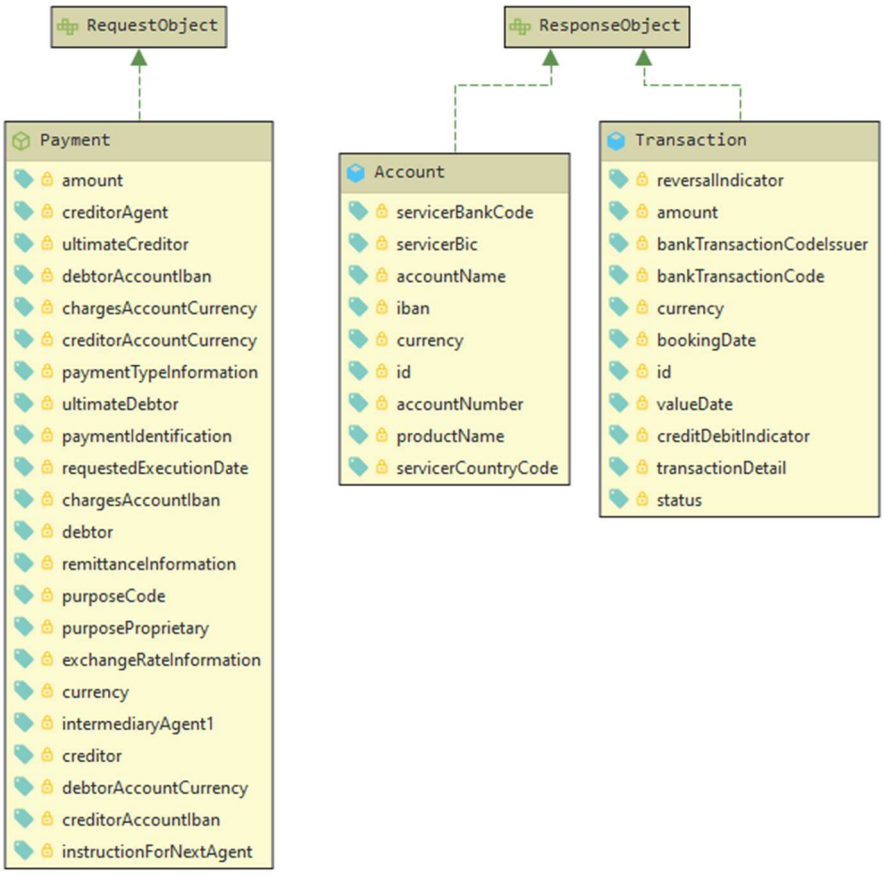
Knihovna komunikuje s REST API jednotlivých bank. O získání autentizačního tokenu se stará finální aplikace.

3.2 Datový model

Datový model kopíruje specifikaci českého standardu pro otevřené bankovnictví. Obsahuje základní objekty `Account`, `Payment` a `Transaction`. Ty se skládají z dalších objektů jako je `TransactionParties` představující příjemce nebo odesílatele transakce, `PostalAddress` představující adresu a mnoho dalších dílčích objektů. Dílčí objekty vždy představují určitý datový typ dle českého standardu a jsou použity jako property jiných datových objektů.

Rozhraní `PagingInterface` představuje stránkování v seznamech a je implementován v objektech `TransactionList` a `AccountList`. Ty dále poskytují pole základních objektů typu `Transaction` a `Account`.

Následuje ukázka vlastností základních datových objektů.



Obrázek 5 - UML základních datových objektů

3.3 Použité technologie

V této kapitole jsou popsány technologie a nástroje použité pro vývoj knihovny.

3.3.1 PHP

PHP je skriptovací jazyk používaný pro vývoj webových aplikací. Jeho tvůrcem je Rasmus Lerdorf, který vytvořil první sadu skriptů v roce 1994. Původní význam zkratky je „Personal Home Page“.

Jedná se o interpretovaný jazyk. Jeho jádro je napsáno v jazyce C a C++. Script je prováděn na straně serveru. Výsledný výstup je odeslán po dokončení skriptu. PHP lze použít i v příkazové řádce pro automatizaci různých úkolů. Soubory skriptu mají koncovku „.php“.

Jazyk se velice rozšířil díky jeho jednoduchosti a rozmachu webových technologií. Proměnná jazyka PHP není striktně typová, může obsahovat jakýkoliv datový typ a ten se může měnit i za běhu scriptu.

Typovost se v jazyce začíná objevovat s verzí 7.0 a v dalších verzích se postupně rozšiřuje. Ve verzi 7.2 je již možné typově omezit parametry metod a návratové typy.

Výhody použití PHP:

- Jednoduché pro začátečníky
- Multiplatformní
- Open-source
- Velké množství vývojářů
- Velké množství frameworků a knihoven

Nevýhody:

- Nižší rychlost díky nutnosti interpretace
- Slabší návrh jazyka oproti vyspělejším jazykům jako Java (v posledních verzí již dohání vyvinutější jazyky)
- Složitější údržba starších aplikací

3.3.2 Git

Git je distribuovaný systém správy verzí vytvořený Linusem Torvaldsem. Slouží ke sledování změn v textových souborech a jejich distribuci. Původně byl vyvinutý pro správu a vývoj Linuxového jádra Kernel. Nejčastěji se používá v týmovém prostředí při vývoji aplikací a při publikování zdrojového kódu jako open-source.

Je dostupný na všech hlavních platformách jako je Windows, Linux a MacOS.

3.3.3 Composer

Composer je program napsaný v jazyce PHP a slouží pro správu závislostí v PHP projektech. Každý projekt používající Composer se stává balíčkem, na který lze vytvořit odkaz v dalším projektu. Program importuje všechny definované závislosti a nainstaluje je do složky `vendor`. V tomto adresáři je vygenerován soubor „autoload.php“, který zajistí dostupnost všech závislých skriptů v projektu.

Nástroj podporuje všechny operační systémy od Linuxu, přes Widnows až po MacOS.

Inicializace projektu probíhá příkazem v terminálu `composer init`. Spustí se průvodce, pomocí něhož se vygeneruje soubor s konfigurací „composer.json“. Konfigurace je ve formátu JSON. Obsahuje název balíčku, popis, licenci a seznam závislosti. Specifikovat lze

také závislosti na verzi jazyka PHP a jednotlivá rozšíření PHP, jako je například (extjson, ext-pdo).

Závislosti se přidávají pomocí příkazu `composer require monolog/monolog`, kde poslední část obsahuje název závislosti v podobě balíčku.

Po instalaci závislostí je vytvořen v projektu soubor „composer.lock“, který obsahuje seznam konkrétních verzí nainstalovaných závislostí.

Veškeré závislosti jsou instalovány ve složce v kořenovém adresáři projektu s názvem `vendor`. Ta obsahuje vygenerovaný soubor `autoload.php`, kterým se načítají závislosti do aktuální aplikace.

Pro open-source závislosti lze použít veřejný registr balíčků Packagist [18]. Zde najdeme veškeré moderní PHP frameworky, jejich rozšíření, knihovny pracující s databází, texty, soubory, různými protokoly a další.

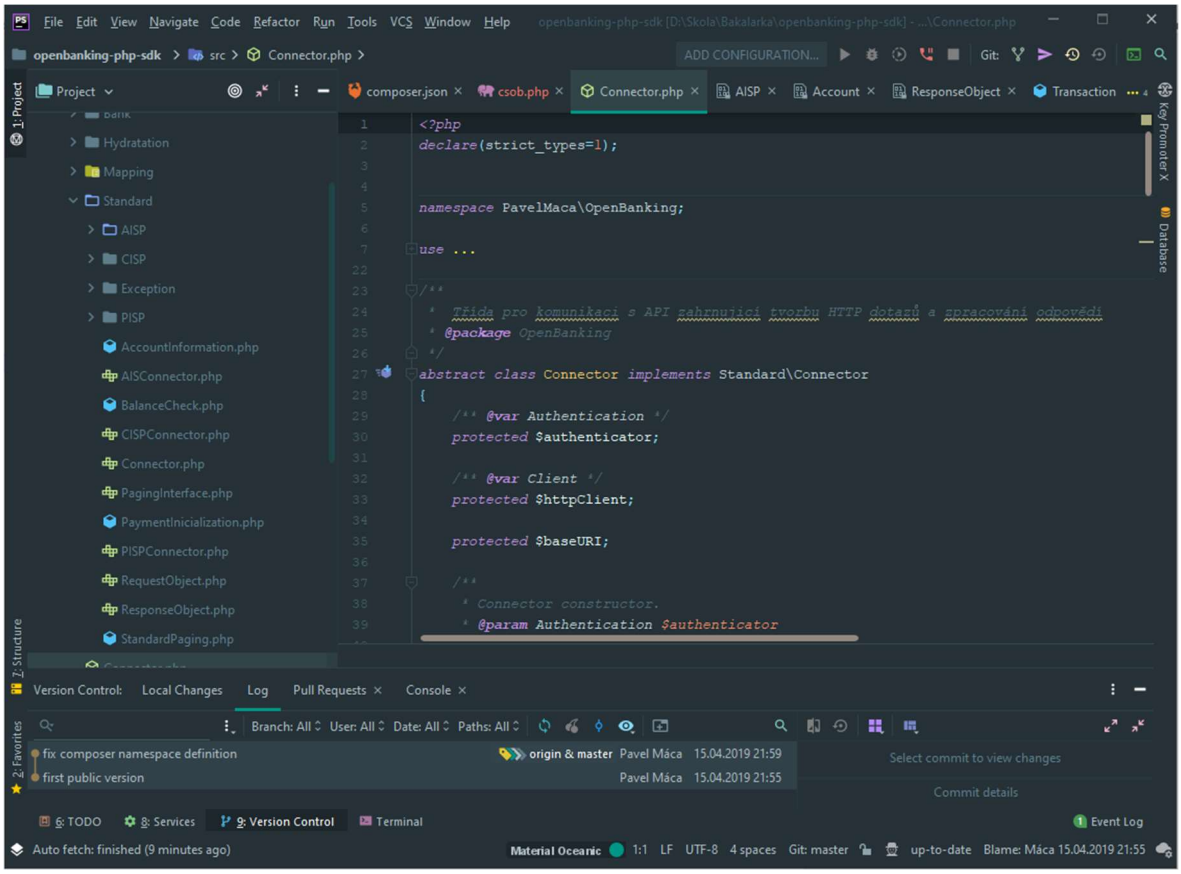
3.3.4 Vývojové prostředí PHPStorm

PHPStorm je jeden z produktů firmy JetBrains. Jedná se o vývojové prostředí určené pro PHP projekty a webové aplikace.

Obsahuje podporu verzování pomocí nástroje Git, integraci s nástrojem Composer. Prostředí umožňuje rozšiřitelnost pomocí pluginů. Je postavené na jednotné platformě jako ostatní produkty firmy JetBrains, a proto je seznam pluginů velice rozsáhlý.

Mezi základní vlastnosti patří:

- Indexování projektu
- Automatické dokončování a napovídání názvů tříd, metod, proměnných
- Nástroje na statickou analýzu kódu
- Nástroje pro verzování projektu
- Chytrá navigace v kódu
- Rychlý a bezpečný refactoring
- Jednoduché ladění



Obrázek 6 - Vývojové prostředí PHPStorm

3.3.5 Postman

Postman je počítačová aplikace pro práci s REST API službami. Umožňuje snadné vytváření požadavků a zobrazování odpovědí. Vytváření kolekcí požadavků, sdílení v týmech, parametrizaci a vytváření automatických testů.

4 Implementace

V této kapitole je popsána implementace navržené architektury pro knihovnu.

Knihovna je složena ze tří částí. Rozhraní pro komunikaci s REST API, vrstvu mapující data na objektový model a samotný objektový model. Objektový model reprezentuje služby specifikované českým standardem pro otevřené bankovnictví a jejich data.

Název knihovny a hlavní jmenný prostor je `\PavelMaca\OpenBanking`. Níže uvedené názvy tříd budou dále bez tohoto prefixu.

4.1 Autentizace

Proces autentizace není v knihovně obsažen. Český standard určuje autentizaci pomocí OAuth 2.0. Získávání a uchovávání tokenů pro přístup není v rozsahu knihovny. Z pohledu bezpečnosti je lepší nechat konkrétní aplikaci použít bezpečné uložiště a ověřenou knihovnu pro tento proces. Vhodná knihovna pro OAuth 2.0 je například „league/oauth2-client“.

Knihovna musí přidávat autentizační údaje k jednotlivým HTTP požadavkům. K tomu účelu je vytvořeno rozhraní `Auth\Authentication`. Rozhraní obsahuje metody pro získání autentizačních hlaviček HTTP požadavku a cestu k souboru s certifikátem.

```
interface Authentication
{
    public function getAuthHeaders(): array;

    /** @return null|string|array */
    public function getCertificate();
}
```

Kód 5 - Rozhraní autentikátoru

Přístupové údaje je mnohdy nutné ukládat do mezipaměti, proto je ideální implementovat toto rozhraní ve vrstvě aplikace, která má na starosti OAuth 2.0 tokeny a jejich správu.

Výsledná implementace se následně použije jako argument konektoru, který použije veřejné metody tohoto rozhraní pro sestavování HTTP požadavků.

4.2 Konektor

Úkolem konektoru, je zprostředkování komunikaci knihovny s cílovým REST API.

Základem je prázdné rozhraní `Standard\Connector`, které představuje libovolnou implementaci. Vlastní implementací tohoto rozhraní lze nahradit výchozí, která je obsažena

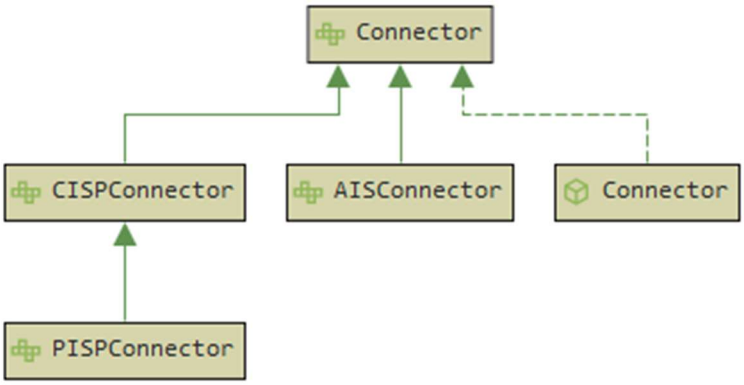
v knihovně jako abstraktní třída `Connector`. Výchozí konektor používá knihovnu „guzzlehttp/guzzle“ [19] pro zprostředkování HTTP komunikace.

Během inicializace je vyžadován objekt typu `Auth\Authentication` pro autentizaci jednotlivých HTTP požadavků. Konektor implementuje HTTP metody GET a POST, které jsou potřebné pro účely knihovny.

Součástí konektoru je odstínění chybových stavů vzniklých při komunikaci prostřednictvím HTTP protokolu a použité knihovny. Mezi tyto chyby patří výjimky knihovny Guzzle, které jsou zachyceny a obaleny vlastní typem výjimky se společným předkem `OpenBankingException`, dle typu chyby.

Odpovědi REST API, obsahující chybový stavový kód, jsou zpracovány a detaily obsažené v odpovědi jsou dále mapovány na objekt typu `Standard\Exception\ResponseError`. Tyto detaily jsou dostupné prostřednictvím výjimek odvozených ze třídy `Standard\Exception\StandardException`, metodou `getResponseErrors`.

Pro práci s jednotlivými službami českého standardu musí finální konektor implementovat některé z rozhraní `AISPConnector`, `CISPConnector` nebo `PISPConnector`. Tyto rozhraní definují metody pro práci s jednotlivými ZD REST API a jejich parametry. Třída `Bank\StandardConnector` obsahuje implementaci dle českého standardu pro všechny tato rozhraní. (Obrázek 7)



Obrázek 7 - Rozhraní konektorů

Pokud některá banka implementuje odlišnou strukturu ZD, lze vytvořit vlastní konektor.

Použití samostatného konektoru a dat v podobě pole je také jedna z možností, která představuje rychlý přístup k datům, ale bez objektové struktury dat.

```
/**
 * @param Payment $paymentInicialization
 * @return array
 * @throws StandardException
 */
public function createPayment(Payment $paymentInicialization);
```

Kód 6 - Metoda createPayment ze třídy PISPConnector

Součástí konektoru `Bank\StandardConnector` je rozhraní pro zpracování získaného vícerozměrného pole do objektu. Tato část je nazvaná hydratací dat a je popsána v další části práce.

4.3 Rozhraní služeb

Jednotlivé služby poskytované v REST API rozhraní jako jsou AISP, CISP a PISP mají každá své objektové rozhraní. Implementace se nachází ve třídách `AccountInformation`, `BalanceCheck` a `PaymentInicialization`. Každá z těchto tříd vyžaduje jako parametr instanci konektoru, stejnojmenné služby.

Každá služby obsahuje metody s parametry dle možností interakce s konkrétním ZD v REST API. Návratové hodnoty jsou objekty datové vrstvy, včetně stránkování, pokud je k dispozici.

```
/**
 * @param BalanceCheckRequest $balanceChackRequest
 * @return CISP\BalanceCheckStatus
 * @throws Exception\StandardException
 */
public function getBalanceCheck(
      BalanceCheckRequest $balanceChackRequest
 ) : BalanceCheckStatus
{
    // 3.2.3 Dotaz na dostatek prostředků
    // (POST /my/payments/balanceCheck)
    $data = $this->bankApi->getBalanceCheck($balanceChackRequest);
    return $this->hydratator->hydrateBalanceCheck($data);
}
```

Kód 7 - Metoda getBalanceCheck ve třídě Standard\BalanceCheck

Tato vrstva používá obecné rozhraní konektorů pro získávání dat v podobě vícerozměrných polí. Zároveň získává z konektoru rozhraní pro hydrataci dat a vrací hydratované datové objekty představující odpovědi REST API.

4.4 Hydratace dat

Pojem hydratace dat znamená naplnění již inicializovaného objektu daty. Na rozdíl od serializace a deserializace, které převádějí objekty přímo na proud bytů a jejich součástí je i inicializace objektu. Hydratace může být narozdíl od deserializace neúplná [20].

Pojem je často používán v oblasti ORM databázových vrstev, kde jsou data získaná z databáze převedena do objektových typů [21].

Hydratace je použita pro převod získaných dat z konektoru, v podobě pole, do objektové datové vrstvy.

Proces využívá anotace z datové vrstvy k určení, kde v poli navráceném v HTTP odpovědi se nacházejí konkrétní hodnoty pro daný atribut.

Anotace jsou v PHP textové komentáře. Vyskytují se u jednotlivých struktur PHP jazyka jako je třídy, vlastnost a metoda. Jazyk PHP s těmito anotacemi přímo nepracuje. Pro jejich zpracování se používá knihovna „doctrine/annotations“ [22]

```
abstract class Payment implements RequestObject
{
    use DomesticPaymentHelper;

    /**
     * @PavelMaca\OpenBanking\Mapping\Property(
     *   path="paymentIdentification",
     *
type="\PavelMaca\OpenBanking\Standard\PISP\Parts\PaymentIdentification")
     * @var PaymentIdentification
     */
    protected $paymentIdentification;

    /**
     * @ PavelMaca\OpenBanking\Mapping\Property(
     *   path="paymentTypeInformation",
     *
type="\PavelMaca\OpenBanking\Standard\PISP\Parts\PaymentTypeInformation")
     * @var PaymentTypeInformation|null
     */
    protected $paymentTypeInformation = null;
```

Kód 8 - Ukázka anotací

Během hydratace se nejprve načtena reflexe třídy pomocí `\ReflectionClass`, která se má hydratovat. Reflexe udává objektový popis třídy, její vlastností a metody za běhu scriptu. Díky tomu lze procházet jednotlivé vlastnosti třídy a dále s nimi pracovat.

Knihovna „doctrine/annotations“ načte anotace, případně je uloží do mezipaměti. Dále vytvoří z anotace objekt, který je instancí třídy dle názvu anotace.

Knihovna používá vlastní anotaci s názvem `Property`.

```
/**
 * @package PavelMaca\OpenBanking
 * @Annotation
 * @Target({"PROPERTY"})
 * @Attributes({
 *   @Attribute("path", type = "string", required = true),
 *   @Attribute("type", type = "string", required = false),
 * })
 */
class Property
```

Kód 9 - Anotace třídy Property

Třída reprezentující vlastní anotaci musí obsahovat anotace definované knihovnou. Díky nim lze definovat vlastnosti anotačního objektu a validační pravidla pro načítání anotace v datové vrstvě.

Anotace `@Annotation` zde označuje třídu `Property` jako vlastní typ anotace. Anotace `@Target({"PROPERTY"})` označuje místo použití vlastní anotace. Zde se jedná o použití u vlastností jiných tříd. Pole `@Attributes` dále specifikuje jednotlivé atributy anotace, jejich typ a příznak, zda jsou povinné.

Námi specifikovaná anotace má dva atributy. Prvním je `path`, který je povinný a označuje cestu ve formě řetězce. Druhým je datový typ `type`, který je nepovinný a výchozí hodnotou bude řetězec.

Parametr `path` specifikuje cestu, pod kterou je nalezena hodnota, která má být uložena ve vlastnosti s touto anotací. Cesta je tvořena řetězci indexů oddělenými tečkami.

V příkladu níže je JSON řetězec, který je díky vlastní anotaci a cestě uvedené v hodnotě `path` převeden do vlastnosti `iban`.

```
/**
 * @PavelMaca\OpenBanking\Mapping\Property(path="identification.iban")
 * @var string
 */
protected $iban;
```

Kód 10 - Vlastní anotace a definice cesty

```
"identification": {
    "iban": "CZ0708000000001019382023"
}
```

Kód 11 - Příklad struktury dat pro hydrataci

Pro čtení anotací je potřeba vytvořit instanci `AnnotationReader`. Ta metodou `getPropertyAnnotation` spolu s parametrem obsahující reflexi konkrétní vlastnosti

a název námi vytvořené anotační třídy. Po přečtení se vrátí instance naší anotační třídy `Property`, které se do konstruktoru předají vlastnosti anotace, parametry `path` a `type`.

Parametr anotace `type` specifikuje, který datový typ se má vytvořit. Pokud datový typ implementuje `ResponseObject` provede se hydratace i tohoto objektu. Tím je zajištěna rekurze pro hydrataci složitějších struktur.

Pomocí reflexe v jazyce PHP lze vytvořit instanci třídy bez volání konstruktoru. To umožňuje vytvoření datového objektu přímo z odpovědi zdroje bez vazby na povinné parametry konstruktoru. Jelikož jde o „immutable“ objekty a jejich metody jsou pouze pro čtení stavu, nepředstavuje toto velký problém.

Pomocí metody `setAccessible(true)` reflexe je vlastnost objektu nastavena jako přístupná i když je v kódu označena jako `private` nebo `protected` a lze ji nastavit hodnotu (viz. Kód 12). Vlastnost samotná zůstává `protected`, případně `private`, pro ostatní prováděný kód. Pouze reflexe umožňuje porušení této ochrany a přístup k hodnotám zvenčí.

```
/**
 * @param string $dataObjectClass
 * @param array $data
 * @return object
 */
protected function hydrate(string $dataObjectClass, array $data)
{
    $reflectionClass = new ReflectionClass($dataObjectClass);
    $properties = $reflectionClass->getProperties();

    $classInstance = $reflectionClass->newInstanceWithoutConstructor();

    foreach ($properties as $property) {
        $value = $this->getPropertyValue($property, $data);
        if ($property->isProtected()) {
            $property->setAccessible(true);
        }
        $property->setValue($classInstance, $value);
    }

    return $classInstance;
}
```

Kód 12 - Hydratace datových objektů

Při vytváření požadavku typu POST jsou parametry předávány také pomocí datového objektu. Postup vytváření pole dat pro JSON je přesně opačný proti hydrataci. Hodnota z vlastnosti datového objektu je uložena podle definované cesty do více rozměrného pole. Postupným procházení všech vlastností je složené celé datové pole požadavku, a to je vráceno pro odeslání HTTP požadavkem. Podporována je opět rekurze u reflexe, pokud typ vlastní anotace implementuje rozhraní `RequestObject`.

4.5 Instalace knihovny

Pro instalaci knihovny je zapotřebí program Composer.

Příkazem `composer require pavelmaca/open-banking` nastavíme závislost vyvíjené aplikace na knihovně. Následuje ukázka použití knihovny a výpisu bankovních účtů z fiktivní banky.

```php
require_once __DIR__ .'/../vendor/autoload.php';

$auth = new \PavelMaca\OpenBanking\Auth\StandardAuthentication();
$auth->setCertificate(__DIR__. '/../data/cert.crt', 'secretPassword');
$auth->setAccessToken('timeLimitedToken');

$client = new \OpenBanking\Bank\StandardConnector($auth,
'http://banka.cz/', 'v1');
$aisp = new \PavelMaca\OpenBanking\Standard\AccountInformation();
$accountList = $aisp->getAccountList();

var_dump($accountList);
```

Kód 13 - Příklad použití knihovny v kódu aplikace

První řádek v ukázce načte všechny závislosti z programu Composer do běžícího skriptu.

Následuje inicializace autentizačního objektu a nastavení OAuth tokenu. Pro připojení do banky je použit standardní konektor s parametry autentizačního objektu, základní URL REST API a její verze.

Ukázka představuje načtení seznamu bankovních účtů. Proto vytvoříme instanci služby pro AISP, `AccountInformation` a zavoláme metodu `getAccountList`, která vrátí seznam bankovních účtu v objektu `AccountList`.

4.6 Testování

Pro testování a vývoj byly také použity vývojářské rozhraní Equa Bank a České spořitelny. V nich byly provedeny testy autentizace a autorizace. Na základě těchto testů byly vytvořeny specifické třídy konektorů a autentizace, které obsahují základní data potřebné pro snadné připojení k těmto REST API rozhraním. Samotný proces autentizace není součástí knihovny.

Součástí knihovny jsou automatické testy, vytvořené pomocí frameworku PHPUnit.

Pro testování byly použity příklady požadavků a odpovědí z definice českého standardu. Příklady jsou uloženy ve složce `/tests/data/` s příponou „json“.

Testuje se základní inicializace konektoru za použití mock rozhraní z knihovny guzzle.

Proces mockování je nahrazení reálného objektu za testovací imitaci.

Díky použití mock rozhraní, je možné před použitím knihovny nastavit přímo v kódu testu odpověď na požadavek, který knihovna vytvoří. Testuje se zpracování odpovědi knihovnou, stránkování a zpracování chybových stavů.

Následuje příklad odpovědi ve formátu JSON a kódu, který testuje knihovnu, zda jsou data správně zpracována.


```
{
  "pageNumber": 0,
  "pageCount": 2,
  "pageSize": 100,
  "nextPage": 1,
  "accounts": [
    {
      "id": "D2C8C1DCC51A3738538A40A4863CA288E0225E52",
      "identification": {
        "iban": "CZ0708000000001019382023",
        "other": "1019382023"
      },
      "currency": "CZK",
      "servicer": {
        "bankCode": "0800",
        "countryCode": "CZ",
        "bic": "GIBACZPX"
      },
      "nameI18N": "Muj hlavni osobni ucet",
      "productI18N": "Osobní účet ČS"
    }
  ]
}
```


Kód 14 - Příklad JSON odpovědi pro dotaz na seznam bankovních účtů

```
public function testGetAccountList()
{
    $aisp = new AccountInformation($this->connector);

$this->connector->getMockHandler()->append(new Response(200, [],
file_get_contents(__DIR__ . '/../data/aisp/accounts.json')));
    $accountList = $aisp->getAccountList();
    $this->assertInstanceOf(AccountList::class, $accountList);

    // Test paging
    $this->assertTrue($accountList->hasPaging());
    $this->assertSame(0, $accountList->getPageNumber());
    $this->assertSame(2, $accountList->getPageCount());
    $this->assertSame(100, $accountList->getPageSize());
    $this->assertSame(null, $accountList->getTotalCount());
    $this->assertSame(1, $accountList->getNextPage());

    $accounts = $accountList->getAccounts();
    $this->assertCount(2, $accounts);
    $this->containsOnlyInstancesOf(Account::class)->evaluate($accounts);

    $firstAccount = reset($accounts);
    $this->assertSame('Muj hlavni osobni ucet',
$firstAccount->getAccountName());
    $this->assertSame('1019382023', $firstAccount->getAccountNumber());
    $this->assertSame('CZK', $firstAccount->getCurrency());
    $this->assertSame('Osobní účet ČS', $firstAccount->getProductName());
    $this->assertSame('0800', $firstAccount->getServicerBankCode());
    $this->assertSame('GIBACZPX', $firstAccount->getServicerBic());
    $this->assertSame('CZ', $firstAccount->getServicerCountryCode());
}
```

Kód 15 - Testování metody getAccountList, pro získání seznamu bankovních účtů

U služeb CISP a PISP je testováno sestavení požadavku. Datový objekt, představující požadavek, je převeden pomocí hydratace do pole, a to je porovnáváno s očekávaným polem.

```
public function testGetBalanceCheckRequest()
{
    $transactionDetail = new TransactionDetail('CZK', 10050.15);
    $balanceCheckRequest = new BalanceCheckRequest('123456',
'CZ0708000000001019382023', $transactionDetail);
    $balanceCheckRequest->setDebtorAccountCurrency('CZK');

    $card = new Card('1234**********6789');
    $card->setCardholderName('Jan Novák');
    $balanceCheckRequest->setCard($card);

    $merchant = new Merchant("471 16 129", "NEOLUXOR", "Neoluxor s.r.o.",
"5192");
    $merchant->setAddress("Hlavní 5, Praha 1");
    $merchant->setCountryCode('CZ');
    $balanceCheckRequest->setMerchant($merchant);

    $balanceCheckRequest->setAuthenticationMethod('NPIN');

    $data = $this->connector->getCISPHydratator()
->serializeBalanceCheckRequest($balanceCheckRequest);

    $exampleData = Json::decode(file_get_contents(__DIR__ .
'/../data/cisp/balanceCheck_request.json'), Json::FORCE_ARRAY);
    $this->assertEquals($exampleData, $data);
}
```

Kód 16 - Test vytvoření požadavku

Během vytváření testů bylo potřeba upravit inicializace standardního konektoru, aby bylo možné použít mockovací rozhraní knihovny guzzle.

5 Závěr

Cílem práce bylo vytvořit knihovnu pro práci s REST API rozhraním pro otevřené bankovnictví v programovacím jazyce PHP. Po analýze pojmu otevřeného bankovnictví a souvisejících technologických standardů jsem definoval softwarové požadavky. Vytvořil jsem logický rámec, který znázorňuje nejen hlavní cíle projektu, ale také myšlenku, která vedla k jeho vytvoření. Uživatelské scénáře mi umožnili blíže určit použití knihovny ve finálních aplikacích, pro které je určena. Oddělil jsem autentizační proces a ponechal jeho řešení na výsledné aplikaci.

Během procesu návrhy architektury došlo k rozdělení knihovny do několika částí. Primární se stala sada rozhraní, umožňující modularitu mezi jednotlivými vrstvami. Datový model byl vytvořen podle PDF dokumentace českého standardu pro Open-banking. Struktura dat ve standardu je popsána formou tabulek, které se rozléhají až na několik stránek. Celý proces byl tedy velice složitý a náchylný na chybovost.

Během vývoje byla knihovna testována na vývojových prostředích banky Equa Bank a České spořitelny. Ty bohužel poskytují pouze statická data, a tak šlo spíše o test autentizace, mimo knihovnu. Hlubší testování probíhalo na zkušebních datech pomocí PHPUnit testů.

Díky různým procesům při autorizace plateb, byla tato část z knihovny vypuštěna. Knihovna obsahuje metody pro inicializaci platba, ale již neřeší proces autorizace. Ten je možné dokončit například v internetovém bankovnictví. V plánu je tento proces implementovat, jakmile vejde v platnost verze standardu 3.0, který obsahuje hromadné platba.

Knihovna najde pravděpodobně malé využití, dokud nezíská více firem licenci pro TPP. Proces získávání licence není současně dobře popsaný formou postupů. Požadavky jsou specifikovány pouze v zákoně o platebním styku a není dostupný jednoznačný výklad. Nicméně tato situace se postupně mění. Na podzim roku 2019 již licenci získalo několik subjektů.

Zdrojový kód knihovny je publikována na serveru Github [23] a v balíčkovacím systému Packagist [18] jako pavelmaca\open-banking [24] pod licencí MIT.

6 Odkazy a literatura

[1] *SMĚRNICE EVROPSKÉHO PARLAMENTU A RADY (EU) 2015/2366*. In: . Štrasburk: Úředním věstníku Evropské unie, 2015, ročník 2108, číslo 2366. Dostupné také z:

https://eur-lex.europa.eu/legal-content/CS/TXT/HTML/?uri=CELEX:32015L2366

[2] *Zákon o platebním styku*. In: . Sbírka zákonů: ISSN 1211-1244, 2017, ročník 2017, číslo 370. Dostupné také z: https://www.mfcr.cz/cs/legislativa/legislativni-dokumenty/2018/zakon-c-370-2017-sb-30664

[3] Český standard pro Open banking. *Česká bankovní asociace* [online]. 2019 [cit. 2019-04-14]. Dostupné z: https://www.czech-ba.cz/cs/cesky-standard-pro-open-banking-1

[4] RFC 2616. *Hypertext Transfer Protocol -- HTTP/1.1*. THE INTERNET SOCIETY, 1999.

[5] RFC7540. *Hypertext Transfer Protocol Version 2 (HTTP/2)*. ISSN: 2070-1721. THE INTERNET SOCIETY, 2015.

[6] RFC 6749. *The OAuth 2.0 Authorization Framework*. 2012. THE INTERNET SOCIETY, 2012.

[7] *Vyhláška o stanovení pravidel tvorby čísla účtu v platebním styku*. In: . Sbírka zákonů, 2011, ročník 2011, částka 169. Dostupné také z: http://www.sagit.cz/info/sb11169

[8] ISO 13616-1. *Financial services - International bank account number (IBAN)*. 1. International Organization for Standardization, 2007.

[9] *Česká spořitelna - OpenBanking dokumentace* [online]. 2019 [cit. 2019-04-14]. Dostupné z: https://developers.erstegroup.com/docs/guides/csas-tutorials

[10] *Komerční banka - API dokumentace* [online]. Praha: Komerční banka, a. s., b.r. [cit. 2019-04-14]. Dostupné z: https://api.kb.cz/portal

[11] FIELDING, Roy. Architectural Styles and the Design of. *PhD Dissertation University of California*. Irvine: University of California, 2000.

[12] *What is REST*. b.r. Dostupné také z: https://restfulapi.net/

[13] Hypermedia. *Https://en.wikipedia.org/wiki/Hypermedia*. b.r.

[14] MALÝ, Martin. REST: architektura pro webové API. In: *Zdrojak.cz* [online]. zdrojak.cz: zdrojak.cz, 2009 [cit. 2019-03-26]. Dostupné z: https://www.zdrojak.cz/clanky/rest-architektura-pro-webove-api/

[15] FOUNDATION, The. *Introduction to GraphQL*. b.r. Dostupné také z: https://graphql.org/learn/

[16] RFC 7230. *Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing*. ISSN: 2070-1721. THE INTERNET SOCIETY, 2014.

[17] EVROPSKÝ SOCIÁLNÍ FOND ČR. Metodika logického rámce. *Https://www.esfcr.cz/documents/21802/782328/02_Metodika_logickeho_ramce.pdf/b840b4ad-5d37-44c4-ade4-70f663f8047f*. 2019.

[18] *Packagist* [online]. b.r. [cit. 2019-04-15]. Dostupné z: https://packagist.org/

[19] *Guzzle, PHP HTTP client* [online]. Michael Dowling, b.r. [cit. 2019-04-15]. Dostupné z: https://guzzlephp.org

[20] ROBERTSON, Erick. What does it mean to hydrate an object?. *Https://stackoverflow.com/a/20787106/3944698*. 2013.

[21] PIVETTA, Marco. Doctrine ORM Hydration Performance Optimization. *Https://ocramius.github.io/blog/doctrine-orm-optimization-hydration*. 2015.

[22] *PHP Doctrine DocBlock Annotations Parser library* [online]. Doctrine group, b.r. [cit. 2019-04-15]. Dostupné z:

https://www.doctrine-project.org/projects/annotations.html

[23] *GitLab Inc.* [online]. San Francisco, Kalifornie, USA, b.r. [cit. 2019-04-15]. Dostupné z: https://gitlab.com

[24] *PHP knihovna pavelmaca/OpenBanking* [online]. GitHub, Inc., b.r. [cit. 2019-04-15]. Dostupné z: https://github.com/pavelmaca/OpenBanking

7 Seznam použitých symbolů a zkratek

AISP	Account Information Provider
API	Aplication programming interface
CISP	Card Issuing Service
HTTP	Hypertxt Transfer Protocol
JSON	JavaScript Object Notation
PISP	Payment Initiation Service
PSD2	Payment Service Directive
REST	Representational State Transfer
TTP	Third party provider
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
SQL	Structured Query Language
WSDL	Web Service Definition Language
ZoPS	Zákon o platebním styku
ZD	Zdroj dat

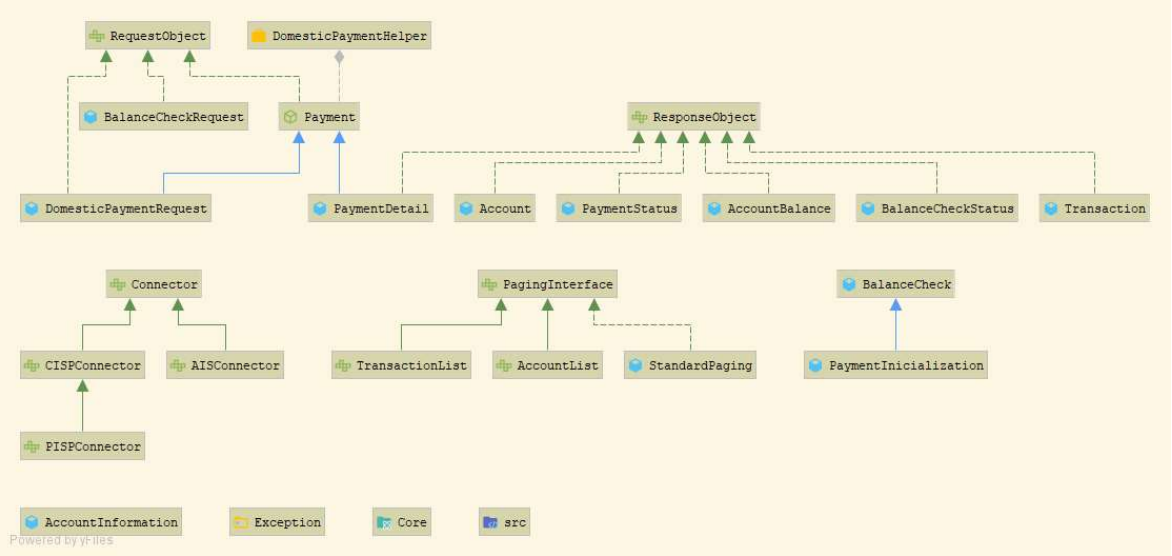
9 Obrázky

10 Seznam ukázek kódu

11 Přílohy

- Příloha A – Diagram jmenného prostoru Standard v knihovně
- Příloha B – Obsah přiloženého CD

Příloha A – Diagram jmenného prostoru Standard v knihovně



Digram zobrazující struktur jmenného prostoru `Standard` a vztahy mezi objekty. Například použití konektoru `CISPConector` jako předka pro `PISPConector`.

Příloha B – Obsah přiloženého CD

PavelMaca_standard_diagram.png – Diagram jmenného prostoru “Standard“

PavelMaca_open-banking-master.zip – Kód knihovny, tak jak je zveřejněn na webu https://github.com/pavelmaca/open-banking/ ve verzi 1.0

PavelMaca_bakPrace.pdf – Bakalářská práce ve formátu pdf.