Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačního inženýrství

Bakalářská práce

Integrace internetového bankovnictví do CRM systému Salesforce

Jakub Mužík

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Jakub Mužík

Systémové inženýrství a informatika
Systémové inženýrství

Název práce

Integrace internetového bankovnictví do CRM systému Salesforce

Název anglicky

Integration of an internet banking into the Salesforce CRM system

---

Cíle práce

Bakalářská práce je zaměřena na problematiku integrace bankovních služeb do existujícího CRM systému s využitím REST API. Hlavním cílem bude zprovoznění komunikace a ovládání internetového bankovnictví ze CRM platformy – Salesforce. Dílčím cílem bude popsat využité technologie a postupy.

Metodika

Práce sestává ze dvou částí, teoretické a praktické. Metodika zpracování teoretické části práce vychází ze studia odborných informačních zdrojů. Na základě syntézy zjištěných poznatků budou popsána teoretická východiska pro zpracování praktické části práce.

Praktická část práce spočívá v integraci internetového bankovnictví do existujícího CRM systému na platformě Salesforce. Bude provedena analýza a návrh rozšíření CRM systému, které bude následně implementováno a otestováno. Serverová část implementace bude provedena v jazyce Apex, klientská část aplikace pak bude implementována s využitím standardních webových technologií (HTML, JavaScript atd.).

Výsledná nová funkcionalita CRM systému bude otestována v praxi. Závěrem budou shrnuty poznatky získané při implementaci a testování a budou popsány možnosti případného dalšího rozvoje a úprav vytvořeného rozšíření.

Doporučený rozsah práce

35-40 stran

Klíčová slova

Salesforce,CRM,Apex,REST API,HTML,JavaScript

Doporučené zdroje informací

Fio banka.FIO API BANKOVNICTVÍ [online]. 2019 [cit. 2019-05-6]. Dostupné z WWW: <https://www.fio.cz/docs/cz/API_Bankovnictvi.pdf>.

Salesforce.com. Apex Developer Guide [online]. 2019 [cit. 2019-05-6]. Dostupné z WWW: <https://developer.salesforce.com/docs/atlas.en-us.apexcode.meta/apexcode/apex_dev_guide.htm>.

Salesforce.com. Lightning Aura Components Developer Guide [online]. 2019 [cit. 2019-05-6]. Dostupné z WWW: <https://developer.salesforce.com/docs/atlas.en-us.lightning.meta/lightning/intro_components.htm>.

Salesforce.com. REST API Developer Guide [online]. 2019 [cit. 2019-05-6]. Dostupné z WWW: <https://developer.salesforce.com/docs/atlas.en-us.api_rest.meta/api_rest/intro_what_is_rest_api.htm>.

Předběžný termín obhajoby

2019/20 LS – PEF

Vedoucí práce

Ing. Jiří Brožek, Ph.D.

Garantující pracoviště

Katedra informačního inženýrství

Elektronicky schváleno dne 19. 2. 2020

Ing. Martin Pelikán, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 19. 2. 2020

Ing. Martin Pelikán, Ph.D.

Děkan

V Praze dne 16. 03. 2020

Čestné prohlášení

Prohlašuji, že svou bakalářskou práci "Integrace internetového bankovnictví do CRM systému Salesforce" jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu použitých zdrojů na konci práce. Jako autor uvedené bakalářské práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne 15.3.2020 _____

Poděkování

Rád (a) bych touto cestou poděkoval panu Ing. Jiřímu Brožkovi, Ph.D. za odbornou pomoc ve všech aspektech ohledně tvorby mé bakalářské práce a za ochotu a rychlou domluvu při plánování osobních konzultací.

Integrace internetového bankovnictví do CRM systému Salesforce

Abstrakt

Teoretická část této bakalářské práce obsahuje popis nejdůležitějších pojmů pro tvorbu webových aplikací a také technologií používaných pro rozšíření jejich funkcionalit pomocí komunikace s externími systémy. Dále je zde proveden stručný popis technologie CRM a následně platformy Salesforce, která je na této technologii postavená.

V praktické části této práce popisuji celý proces tvorby aplikace využívající data z internetového bankovnictví od analýzy požadavků, základní konfigurace na platformě Salesforce až po samotnou integraci internetového bankovnictví pomocí REST API.

Klíčová slova: Salesforce, CRM, Apex, REST API, HTML, JavaScript

Integration of an internet banking into the Salesforce CRM system

Abstract

The theoretical part of this thesis contains a description of the most important concepts for creating web applications and technologies used to extend their functionalities through communication with external systems. Then there is a brief description of CRM technology and Salesforce platform itself, which is based on this technology.

In the practical part of this thesis I describe the whole process of creating an application using data from internet banking from the analysis of requirements, basic configuration on the Salesforce platform to the actual integration of internet banking using the REST API.

Keywords: Salesforce, CRM, Apex, REST API, HTML, JavaScript

Obsah

Seznam obrázků

Seznam tabulek

1 Úvod

Téměř každá firma v dnešní době používá nějaký informační systém, který jim může sloužit pro nespočet účelů. Může se jednat o procesy a správu komunikace s jejich zákazníky, vedení záznamů o zaměstnancích, řízení a definici vnitropodnikových procesů či o vedení účetnictví. Ve své práci se zabývám dnes nejrozšířenějším a nejvíce používaným Customer Relationship Management systémem pro správu a komunikaci se zákazníky.

Téma této bakalářské práce bylo autorem vybráno z důvodu jeho zaměstnání na pozici Salesforce Developera a s tím spojený zájem o programování nových funkcionalit na CRM platformě Salesforce a také nepochybně o platformu jako takovou. Výběrem tohoto tématu tak mohl autor efektivně a současně pracovat na posouvání se k úspěšnému zakončení jeho bakalářského studia a zároveň se vzdělávat v oboru jeho zaměstnání, ve kterém zkušenosti získané tvorbou této práce určitě zužitkuje.

2 Cíl práce a metodika

2.1 Cíl práce

Cílem této bakalářská práce je seznámit čtenáře s obecným pojmem CRM, jeho konkrétním řešením v podobě Salesforce a také s integrací bankovního systému do existujícího Salesforce prostředí s využitím REST API za cílem rozšířit jeho funkcionality ve prospěch účetních procesů společnosti.

Hlavním cílem práce bude zprovoznění komunikace s internetovým bankovnictvím na platformě Salesforce, ukládání takto získaných dat a jejich následné využití pro různé provozní činnosti firmy. Dílčím cílem pak bude popsat využité technologie a s tím i spojené postupy.

2.2 Metodika

Práce sestává ze dvou částí, teoretické a praktické. Metodika zpracování teoretické části práce vychází ze studia odborných informačních zdrojů. Na základě syntézy zjištěných poznatků budou popsána teoretická východiska pro zpracování praktické části práce a to konkrétně pojmy, technologie a programovací jazyky používané v CRM systému Salesforce.

Praktická část práce spočívá v integraci internetového bankovnictví do existujícího CRM systému na platformě Salesforce. Bude provedena analýza a návrh rozšíření CRM systému, který bude v Salesforce následně implementován pomocí základní konfigurace, vytvořením uživatelského prostředí a naprogramování jeho potřebných funkcionalit. Serverová část implementace bude provedena v jazyce Apex, klientská část pak bude implementována použitím technologií Salesforce Lightning Component Frameworku a využitím standartním webových technologií jako jsou HTML, JavaScript a CSS.

Výsledná nová funkcionalita CRM systému bude otestována. Závěrem budou shrnuty poznatky získané při implementaci a testování a budou popsány možnosti dalšího rozvoje a úprav vytvořeného rozšíření.

3 Teoretická východiska

3.1 Vymezení pojmů

3.1.1 HTML

HTML, v celém anglickém znění HyperText Markup Language, představuje značkovací jazyk, pomocí kterého můžeme tvořit webové stránky. Dovoluje nám vytvářet komplexní a strukturované stránky složené například pomocí jednotlivých sekcí, paragrafů či nadpisů. Nepleťme si však HTML s programovacím jazykem. Pomocí HTML jsme schopni pouze uspořádat a formátovat dokumenty, nikoliv tvořit dynamické stránky. Struktura HTML jazyka je tvořena pomocí jednotlivých tagů a atributů. Každý tag má specifické vlastnosti, které dotváříme a modifikujeme pomocí jejich atributů. HTML jazyk byl vytvořen ve Švýcarsku ve výzkumném ústavu CERN Angličanem Timem Berners-Leem. Jeho první verzi, která obsahovala pouze 18 tagů, vydal v roce 1991. Díky veliké popularitě jazyka, je dnes považován jako oficiální webový standart. (1)

3.1.2 CSS

CSS – Cascading Style Sheet language, známý také pod českým názvem Kaskádové Styly, je jazyk, pomocí kterého stylujeme stránky napsané například zmíněným jazykem HTML. Jazyky HTML a CSS jsou vzájemně velmi silně provázané a při tvorbě webových stránek jdou ruku v ruce. Nemůžeme ale říci, že CSS jazyk je technicky nezbytný. Byl vytvořen v roce 1996 institucí W3C (World Wide Web Consortium). Jazyk se zaměřuje pouze na vzhled stránky a odděluje tak obsah stránky od její vizuální prezentace. Dovoluje nám tak například vytvořit dynamicky se měnící vzhled stránky podle toho, na jakém zařízení si stránku prohlížíme. (2)

CSS stylování můžeme aplikovat pomocí tří metod. Interní, Externí a Inlinové metody. Pokud použijeme Interní metodu stylování, CSS styly jsou načítány při každém načtení dané stránky a nejsme schopni pře použít definované CSS styly pro vícero stránek. Tuto vlastnost má také Inlinová metoda, ve které se na rozdíl od Interní metody styly aplikují na jednom místě v hlavičce stránky, jednotlivé styly aplikujeme přímo do jednotlivých HTML tagů pomocí style atributu. Poslední metoda, která je zároveň nejvíce používaná, je metoda Externí, kde se veškeré styly vkládají do externího souboru

s příponou .css. Pomocí tohoto přístupu jsme schopni definované styly ve zmíněném souboru aplikovat pro vícero stránek najednou a redukovat tak množství napsaného kódu. (2)

3.1.3 JavaScript

JavaScript, jako poslední z trojice hlavních jazyků pro tvorbu webových aplikací, představuje jeden ze světově nejznámějších interpretovaných programovacích jazyků. Zmíněnou dvojici jazyků pro tvorbu a stylování obsahu webové stránky, tak dokonale doplňuje o její dynamickou povahu. JavaScript byl vytvořen v roce 1995 Brandanem Eichem. Jméno jazyka se postupně s jeho vývojem několikrát změnilo, počínaje názvem Mocha, přes jména Mona a LiveScript, až k jeho dnes známému názvu JavaScript. Konkrétní kód se do webové stránky přidává buď napřímo pomocí script tagu, nebo pomocí odděleného souboru s příponou .js. JavaScript je jazyk běžící ve webovém prohlížeči u uživatele, respektive na klientské straně aplikace. Díky tomu je uživatel schopný pomocí nástrojů webových prohlížečů kód vypnout. Jazyk nám dále umožňuje definovat chování stránky pro události, jako jsou například stisknutí tlačítka či najetí kurzoru na určitý element stránky. Výhoda JavaScriptu je bezpochyby jeho přenositelnost mezi různými webovými prohlížeči. Jako nevýhodu lze brát jeho velkou popularitu, která přivolává velké množství hackerů. (3)

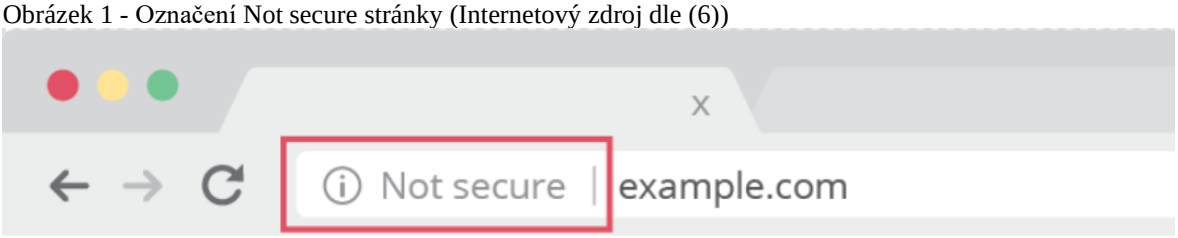
3.1.4 HTTP

Hypertext Transfer Procotocol představuje komunikační protokol a také soubor pravidel pro přenos souborů, jako jsou například textové záznamy, obrázky, zvukové stopy či videa skrze internet. Komunikace probíhá mezi klientem a serverem. Proces komunikace zpravidla začíná na straně klienta, tedy webovým prohlížečem, který vytvoří request na server – cílovou webovou stránku. HTTP poté tento request přebírá a zprostředkovává spojení mezi daným klientem a serverem pomocí TCP protokolu. Poté je daný request pomocí HTTP na server odeslán a jeho odpověď následně vrácena zpět klientovi. Když server odesílá odpověď zpět klientovi, první věc, kterou odešle, je stav vyřízení žádosti. Stav vyřízení je prezentován číselným kódem, kde první číslo vyjadřuje typ stavu a následná čísla specifickou událost pro daný typ. Stav odpovědi, který zpravidla vyžadujeme, je úspěch, tedy úspěšné vyřízení požadovaného requestu. Tento stav se označuje číslem 200. (5)

3.1.4.1 HTTPS

Funkčnost a specifikace protokolu HTTPS je stejná jako pro http, až na zřejmé rozšíření o písmeno "S", které ve slovní definici znamená Secure – zabezpečený. Jedná se tedy o zabezpečenou verzi protokolu HTTP. Pokud si skrze internet vyměňujeme informace mající citlivý charakter, jako například platební či přihlašovací údaje, určitě budeme chtít používat komunikační protokol, který obsahuje prvky pro ochranu úniku těchto dat před možnými hackery. Těmito prvky se rozumí vzájemná autorizace klienta a serveru a následné šifrování přenášených dat, o které se starají SSL/TLS protokoly. Jakákoliv webová stránka, která vyžaduje přihlašovací údaje, by proto měla využívat protokolu HTTPS. V moderních webových prohlížečích, jako je například Chrome, jsou stránky, které nepoužívají zabezpečený protokol HTTPS speciálně označeny příznakem Not Secure nalevo od pole, kam se zadává URL adresa. (6)

Obrázek 1 - Označení Not secure stránky (Internetový zdroj dle (6))



3.1.5 API

API, v celém anglickém znění – Application Programming Interface, je rozhraní umožňující dvěma odděleným systémům vzájemně komunikovat a vyměňovat si potřebná data. Pomocí API dokážeme ze serveru vytáhnout námi požadovanou informaci a následně ji zobrazit v čitelném formátu. Pokaždé když si na telefonu otevřeme nějakou internetovou stránku, nebo například jen zkontrolujeme počasí, využíváme API. (4)

3.1.6 REST

REST – Representational State Transfer, je architektura rozhraní, která poskytuje standardy pro jednoduchou vzájemnou komunikaci mezi systémy. Tyto systémy využívající REST principy, často také nazývané jako RESTful systémy, jsou charakteristické především svým oddělením klientské a serverové části. To znamená, že

kód na klientské části může být kdykoliv změněn bez následného dopadu na část serverovou a umožnit tak jejich oddělený a samostatný vývoj. Nutno podotknout, že serverová část nikdy nedrží a neukládá informace o klientovi a naopak. Kvůli této nevědomosti o předchozí komunikaci, či stavu, tak server, či klient, vždy rozumí přijaté zprávě, protože obsahuje informace důležité pro její kompletní popis. Informace se vždy týkají určitého zdroje (resource), který je definován svým vlastním identifikátorem – URI. (7)

Požadavek, který odesílá klient na server s cílem získat, či modifikovat, určitý zdroj dat, se skládá z následujících částí: Metoda přístupu k datům, Hlavička, Cesta ke zdroji, Tělo. (7)

REST Metody pro přístup ke zdrojům

Klient odesílá požadavky buď pro získání, či editace, určitého zdroje pomocí 4 hlavních REST metod pro přístup ke zdrojům: (7)

- GET – pro získání konkrétní informace
- POST – pro vytváření nových dat
- PUT – pro editaci stávající informace
- DELETE - pro mazání dat

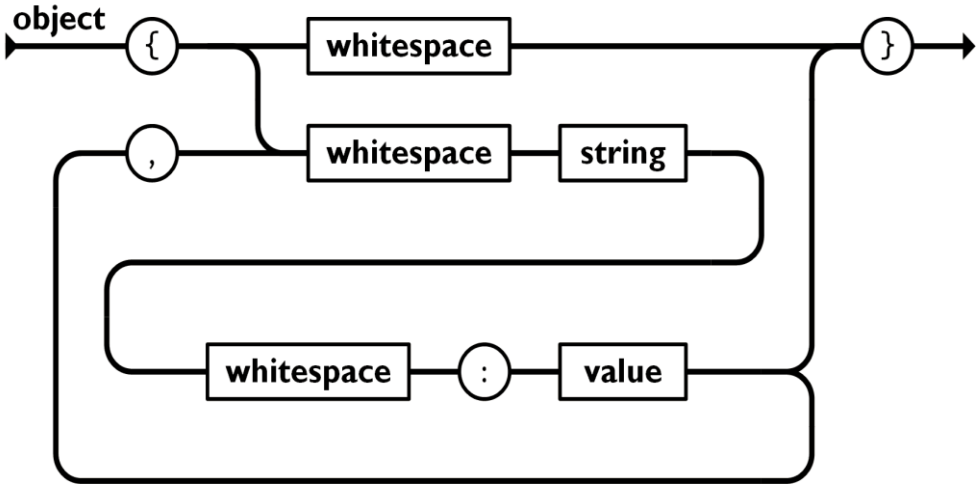
3.1.7 JSON

Jedním z datových formátů, který metody REST podporuje, a který jsem zároveň použil v mé práci, je JSON (JavaScript Object Notation). JSON umožňuje uložit informaci v organizovaném, jednoduše přístupném a člověku čitelném formátu. JSON je sestavený ze dvou základních struktur: Objekt a Pole. Obě tyto struktury se mohou vzájemně prolínat a tvořit tak dohromady složitou a obsáhlou datovou strukturu. (8)

Objekt

Objekt představuje neuspořádanou kolekci dvojic, složených ze jména a hodnoty. Každý objekt začíná znakem “{“ a končí znakem “}“. Za každým jménem následuje znak “:“ a jednotlivé dvojice se oddělují čárkou. (8)

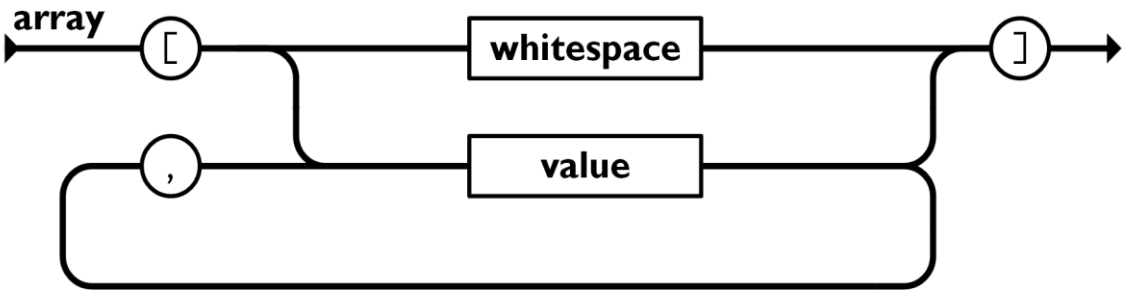
Obrázek 2 - Struktura JSON Objektu (Internetový zdroj dle (8))



Pole

Pole je uspořádaná kolekce hodnot, která je vymezená znaky "[" a "]" a jednotlivé hodnoty se oddělují čárkou. (8)

Obrázek 3 - Struktura JSON Pole (Internetový zdroj dle (8))



3.1.8 MVC

MVC – Model-View-Controller je programovací model, který rozděluje aplikaci do 3 základních komponent. Každá z těchto komponent pak plní svou specifickou roli v rámci vývojářských aspektů aplikace. MVC je jeden z nejpoužívanějších modelů při vývoji webových aplikací. (9)

Model

Model obsahuje veškerou datovou logiku, se kterou uživatel pracuje. Může se jednat o data, která jsou předávaná mezi komponentou View a komponentou Controller či

jakoukoliv businessovou logiku. Například proces, kdy získáme data o uživateli z databáze, následně data upravíme a nakonec je vrátíme zpět do databáze, či je použijeme pro jejich zobrazení. (9)

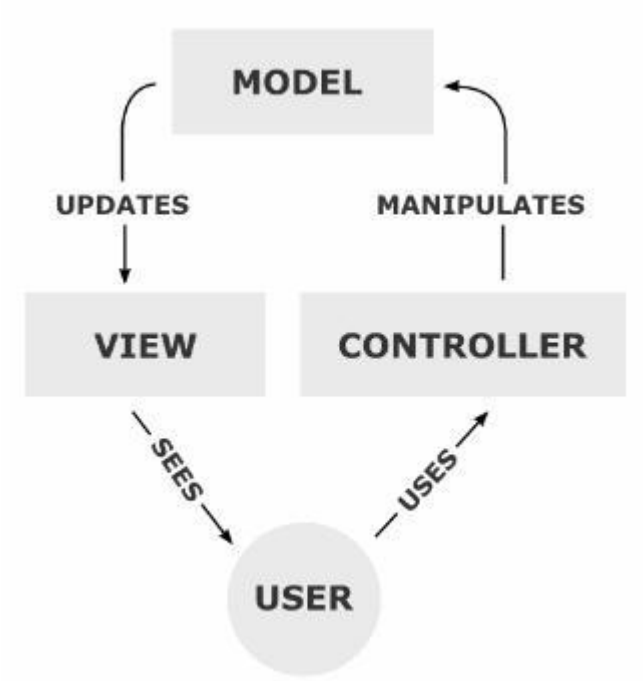
View

View obecně představuje vše spojené s uživatelským rozhraním aplikace, tedy reprezentace dat. (9)

Controller

Controller představuje rozhraní mezi komponentami Model a View, které zprostředkovává veškerou businessovou logiku, manipuluje s daty za použití Modelu a poskytuje finální zprocesovaná data zobrazovaná ve View. (9)

Obrázek 4 - MVC architektura (Internetový zdroj dle (9))



3.2 CRM

CRM, v jeho celém aglickém znění Customer Relationship Management, lze chápat jako strategii firem používanou pro interakci s jejich potencionálními i stávajícími zákazníky, či pro řízení nebo také automatizaci obchodních, marketingových nebo servisních procesů. Používaná strategie má za cíl například zvyšovat výnosy a loajalitu zákazníků anebo také snižovat náklady firmy. (10)

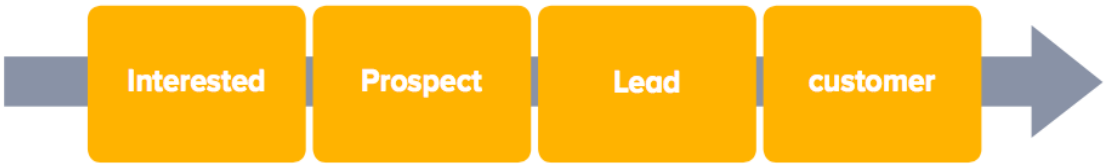
Ať už se jedná o základní informace o zákazníkovi, jeho historické interakce s firmou, či potencionální a stálé běžící příležitosti k obchodu, CRM systém sdružuje všechny tyto informace skrze mnohá oddělení firmy přehledně na jednom místě a pomáhá tak poskytovat lepší služby jejich zákazníkům a z toho vyplývající zvýšené prodeje a výnosy. Zaměstnanci firmy z různých oddělení, například obchodníci, marketingový specialisté či zákaznická podpora, tak mají rychlý přístup k informacím o zákazníkovi ze všech business směrů a nemusejí složitě dohledávat informace z různých zdrojů. (10)

Prodej

Jak jsem již zmínil, v CRM se vyskytují 3 hlavní oblasti - prodej, marketing a servis. Celý CRM proces většinou začíná vytvořením Leadu – záznamu o potencionálním zákazníkovi. Vytvoření takového záznamu probíhá většinou vyplněním nějakého formuláře na internetu, ve kterém o sobě zákazník poskytne informace. Následně je tento Lead zpracováván prodejními procesy definovanými firmou, jako jsou například: (10)

- Kvalifikování a převod Leadu na kontakt
- Správa procesu kvalifikace a jeho sledování
- Sledování a ukládání proběhlé komunikace se zákazníkem
- Ukládání dokumentů a smluv

Obrázek 5 - Fáze vývoje zákazníka v CRM (Internetový zdroj dle (10))



Marketing

Jedná se o strategii firmy, která má za cíl vygenerovat co největší množství Leadů. V CRM se tak může jednat například o: (10)

- Automatizace rozesílání různých marketingových nabídek
- Vytváření a řízení kampaní
- Zobrazování výsledků uspěšnosti marketingových kampaní

Servis

Poslední, a neméně důležitá oblast CRM systému, je zákaznický servis, který obsahuje následující aktivity: (10)

- Komunikace a poskytování servisu zákazníkům skrze mnoho rozličných kanálů: sms, telefonní hovory, e-maily…
- Ukládání historie výsledků komunikace a její následné akce či postupy
- Napovídání zaměstancům ze zákaznické podpory pro případné řešení na základě předchozích podobných případů

3.3 Salesforce

Salesforce je cloudové řešení, které se momentálně bere jako světový leader v oblasti CRM a cloudových systémů. Nutno podotknout, že před nástupem Salesfoce, byla veškerá data CRM systémů ukládána na vlastních serverech firem, což není úplně levná a výhodná záležitost. Toto byla hlavní myšlenka a také hlavní důvod, proč je tato platforma tak úspěšná – poskytnout CRM řešení jako Software-as-a-service známé pod zkratkou SaaS.

Cloudové řešení vám poskytuje okamžitý přístup ke všem vašim datům odkudkoli z internetu, kdy data jsou uložena na serveru u poskytovale. Díky tomu odpadá spousta problémů, jako například zabezpečení dat, kontrola kvality, ztráta dat anebo také údržba serverů. (11)

Obrázek 6 - 4 základní řešení Salesforce CRM (Internetový zdroj dle (10))

Sales Cloud

Sales Cloud reflektuje prodejní oblast CRM. Je navržen pro správu obchodních procesů od začátku do konce. Dále se tu spravují informace o zákaznicích a celkově se pomáhá zvyšovat produktivita obchodních teamů cestou sdílených informací. (12)

Patří sem: (12)

- Vytváření a konvertování Leadů skrze externí webové stránky
- Jednoduchý přístup k datům zákazníků
- Automatické odpovědi zákazníkům
- Chatter funkcionalita, skrze kterou si zaměstnaci sdílí informace, soubory nebo zadávají úkoly či připomínky

Service Cloud

Hlavní úkol Service Cloudu je řešení požadavků, či reklamací, od již stávajících zákazníků. Zaměstnancům je po specifické konfiguraci umožněno komunikovat se zákazníky přímo z platformy Salesforce skrze Live chatu, Call centra, pomocí sms nebo e-mailů. Všechny tyto proběhlé komunikace se ukládají a je možno do nich rychlým a přehledným způsobem zpětně nahlédnout. (13)

Dále sem patří tyto funkcionality: (13)

- Automatické přiřazování požadavků zákazníka zaměstnancům ze servisního oddělení
- Znalostní báze, která může například zobrazovat již vyřešené podobné případy a s jistou intuicí odkazovat na možné historické řešení
- Odesílání dotazníků zákazníkům

Marketing Cloud

Marketing Cloud je prostředí, které nabízí nástroje pro monitorování aktivit zákazníků, například na sociálních sítích. Pro každého zákazníka je možné nakonfigurovat marketingový proces, který obsahuje například doručování zpráv specifickým kanálem na základě zvolených pravidel. (14)

Community Cloud

Community Cloud je určen pro propojení a pro vytvoření komunikace mezi společností, jejími zaměstnanci, zákazníky a partnery. Community cloud umožňuje vytváření externích webových stránek, které mají přístup k veškerým datům Salesforce

instance. Nemusíme tedy vytvářet žádnou duplicitní integraci dat pro jejich zobrazení na webových stránkách, což značně usnadňuje vývoj. (15)

Tyto vytvořené webové stránky mohou být myšleny pro mnoho případů – řešení problémů spojených s produkty či službami firmy, nábor nových zaměstnanců, marketingové úmysly, fórum a podobně. (15)

3.3.1 Zabezpečení dat

Salesforce má definovaný komplexní a zároveň také flexibilní model zabezpečení dat. Flexibility lze dosáhnout pomocí nabízených nástrojů, které umožňují sdílení dat mezi různými subjekty a vrstvami navržené struktury Salesforce prostředí dle obchodních požadavků. (16)

3.3.1.1 Zabezpečení na úrovni Objektů

Dříve než uživateli bude povoleno zobrazit záznam, Salesforce první zkontroluje přístup uživatele pro objekt, ve kterém se daný záznam nachází. Tento typ přístupu lze definovat pomocí 2 typů konfigurace: Profily a Permission sety. V obou případech lze vybrat z následujících typů přístupu: (16)

- Čtení
- Editace
- Vytváření
- Mazání

Na první pohled by se mohlo zdát, že mezi profilem a Permission Setem není rozdíl, ale není tomu tak. Představme si, že ve firmě máme dva oddělené týmy, jeden pro marketing a druhý3 pro prodej. Pro každý z těchto týmů tedy vytvoříme jeden profil, který následně přiřadíme daným uživatelům a poskytneme jim přístup k objektům, se kterými se v dané oblasti pracuje. Teď si ale představme, že bychom chtěli, aby jeden vybraný uživatel, například z prodejního týmu, měl přístup i do určitých objektů z týmu marketingového. Profil použít nemůžeme, protože pokud bychom nastavili profilu pro prodejní tým přístup do objektů z týmu marketingového, dostali by tak tento přístup všichni uživatelé a to nechceme. Na řadu tedy přichází Permission Set. Pod Permission setem si lze představit dodatečnou sadu přístupů, jež se přidávají jednotlivým uživatelům,

kteří už mají přiřazený profil. Je důležité zmínit, že Permission Set je schopen přístup pouze přidávat, nikoli odebírat. (16)

3.3.1.2 Zabezpečení na úrovni Polí

Pokud máme přístup do jednotlivých objektů, přichází na řadu druhá vrstva zabezpečení a to na úrovni jednotlivých polí, které daný objekt obsahuje. (16)

Lze vybrat z následujících typů přístupu: (16)

- Čtení
- Editace

3.3.1.3 Zabezpečení na úrovni Záznamů

Pokud máme přístup k objektům a jeho jednotlivým polím, přichází na řadu poslední vrstva zabezpečení - na úrovni záznamů. (16)

Ve výchozím stavu máme přístup pouze k záznamům, které sami vytvoříme. Nyní se dostanu k objasnění flexibility modelu zabezpečení dat, o kterém jsem psal výše. Salesforce poskytuje 5 nástrojů, pomocích kterých lze sdílet záznamy skrze jednotlivé uživatele, či jejich skupiny: (16)

Organization-wide sharing defaults

OWD umožňuje pro jednotlivé objekty nastavit výchozí přístup pro ostatní uživatele. Můžeme tedy například nastavit, že pokud uživatel vytvoří záznam v určitém objektu, tak k němu všichni uživatelé budou mít přístup. (16)

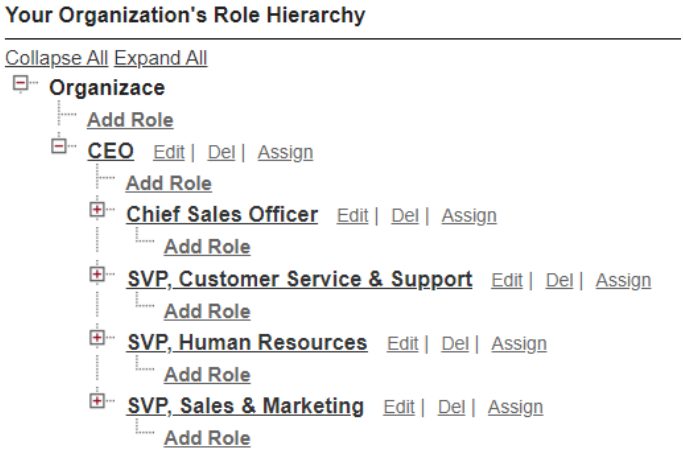
Lze nastavit následující oprávnění: (16)

- Privátní
- Veřejné pro čtení
- Veřejné pro čtení a editaci
- Veřejné pro čtení, editaci a převod majitele

Role hierarchies

V Salesforce je možné si definovat skupiny uživatelů – Role a určitou hierarchii mezi nimi. Hierarchie se vytváří pomocí stromového diagramu. Nastavuje se tedy struktura sdílení dat ve vertikálním směru. V reálném světě tato konfigurace představuje strukturu firmy. Od majitele k jednotlivým manažerům, až po jednotlivé zaměstnance. Manažer tak získá přístup ke všem záznamům, které patří podřízeným zaměstnancům. (16)

Obrázek 7 - Nastavení Role Hierarchy (Vlastní zpracování)



Sharing rules

Sharing rules slouží ke sdílení dat v horizontálním směru, tj. například sdílení mezi různými odděleními na stejné úrovni hierarchie podniku. Můžeme nastavit vstupní pravidla pro záznamy a po splnění takto definovaných pravidel, jejich následné sdílení se skupinou uživatelů. (16)

Lze nastavit tyto 2 skupiny pravidel: (16)

- Sdílení dle majitele záznamu
- Sdílení dle specifických hodnot v polích

Manual sharing

Manual sharing poskytuje majiteli, či uživateli s dostatečným oprávněním k záznamu, sdílet daný záznam s jiným konkrétním uživatelem. Tento způsob sdílení je používán pouze v případě, kdy kvůli struktuře podniku a jejímu nastavenému modelu zabezpečení dat, nelze využít některý z předchozích uvedených nástrojů. (16)

Apex Managed Sharing

Apex Managed Sharing umožňuje sdílení dat na programovací bázi pomocí jazyka Apex. (16)

3.3.2 Vývoj

3.3.2.1 Developer Console

Developer Console je Salesforce nativní vývojové prostředí, které vývojáři používají pro vytváření, debugování a testování aplikací. Níže popíši její nejčastěji používané nástroje a funkcionality. (17)

Vývoj a editace kódu (17)

- Otevírání, editace a vytváření Lightning Komponent, Lightning Aplikací, Apexových tříd a Triggerů.
- Automatické doplňování metod, či příkazů Apexového kódu.
- Možnost automatického formátování kódu.

Debugování (17)

- Zobrazení detailních informací ohledně exekuce jednotlivých metod v transakci, jako jsou například databázové operace, validace či exekuce triggerů.
- Možnost nastavení Checkpointů v apexové třídě. Po skončení transakce si tak můžeme v určité části kódu zobrazit například hodnoty uložené v proměnných.

Manipulace s databází (17)

- Výpis záznamů z databáze pomocí SOQL příkazů.
- Pro vypsané záznamy možnost jejich editace, či mazání.

Spouštění Apexového kódu (17)

- Pomocí Execute Anonymous Apex nástroje možnost spouštět Apexové metody, SOQL dotazy či DML operace.

3.3.2.2 Apex

Apex je silně typový a objektově orientovaný programovací jazyk vytvořený Salesforce společností, pomocí kterého můžeme spouštět příkazy pro řízení transakcí, či volání API na platformovém serveru Force.com. Pomocí syntaxe, která vychází z jazyku Java, dovoluje Apexovým vývojářům přidat businessovou logiku pro většinu systémových událostí, jako je například stisknutí tlačítka, či aktualizace záznamů v databázi. Apexový kód můžeme vytvářet a ukládat ve dvou formách: (18)

- **Apex Třída** - představuje šablonu složenou z jednotlivých metod, proměnných či vnořených tříd, ze které jsou buď následně vytvářeny instance či pouze volány metody
- **Apex Trigger** - kód, který se spustí před nebo po určité datové manipulaci na serveru

Apexový kód ve výchozím stavu běží v systémovém kontextu. To znamená, že nehledí na přístupové oprávnění k objektům, fieldům a záznamům aktuálně přihlášeného

uživatele, který daný kód spustil. Apex jako programovací jazyk mimo jiné také nabízí integrované prostředky pro manipulaci s daty: DML a SOQL. (18)

DML

Data Manipulation Language slouží pro přímou manipulaci s daty na serveru. K dispozici jsou následující Apexové DML příkazy: (18)

- Insert – slouží pro ukládání nových záznamů
- Update – pro aktualizaci stávajících záznamů
- Upsert – aktualizuje či vytváří nový záznam podle toho, jestli v databázi najde shodu se zadanou hodnotou pro daný sloupec
- Delete – pro mazání záznamů
- Undelete – slouží pro obnovení záznamů, které jsou uchovávány v koši po dobu 15 dnů ode dne jejich smazání
- Merge – sloučí až tři záznamy do jednoho

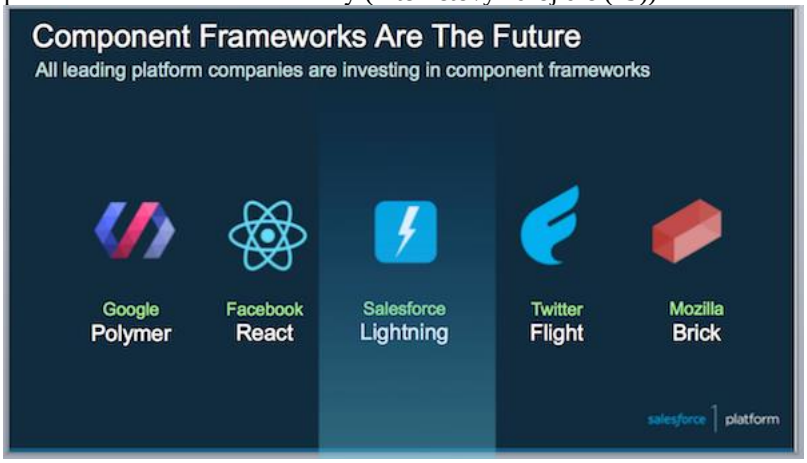
SOQL

Salesforce Object Query Language slouží pro získávání záznamů z databáze ve formě kolekce, samostatného záznamu, či čísla, představující počet záznamů. (18)

3.3.2.3 Lightning Component Framework

Lightning Component Framework je framework, který slouží jako podpora při vývoji dynamických aplikací pro mobilní a desktopové zařízení na platformě Salesforce. Jak už název napovídá, jedná se o komponentově založený vývoj. Pomocí předpřipravených či naprogramovaných Lightning Komponent tak můžeme rychle a efektivně sestavit konzistentní uživatelské rozhraní. (19)

Obrázek 8 - Komponentově založené Frameworky (Internetový zdroj dle (19))



3.3.2.3.1 Lightning Komponenty

Lightning Komponenty jsou samostatné jednotky, které představují jednotlivé sekce uživatelského rozhraní. Lightning Komponenta může přestavovat pouhý řádek textu či složitou a komplexní aplikaci. Komponenta je tvořena pomocí jazyků HTML, CSS a Javascriptu, či dalších vnořených Komponent. (19)

Salesforce nabízí širokou škálu předpřipravených základních Komponent, pod názvem Lightning Base Components, dostupné ve jmenném prostoru lightning, které můžeme libovolně používat například pro zobrazení vstupního pole, tlačítka či dynamické tabulky. (19)

Abychom mohli jednotlivé komponenty začít používat, musíme je nejprve pomocí Salesforce nástrojů vložit do uživatelského rozhraní. (19)

V Developer Consoli pro vytvoření Lightning Komponenty klikneme na **File** -> **New** -> **Lightning Component**. (19)

3.3.2.3.2 Lightning Aplikace

Framework také nabízí vytváření takzvaných Lightning Aplikací, které slouží jako samostatně fungující jednotky, dostupné pod unikátní URL adresou definovanou jménem Aplikace, což je hlavní rozdíl od výše zmíněných Komponent. Pokud chceme například otestovat nově vytvořenou Komponentu bez nutnosti přidávat jí do uživatelského rozhraní Salesforce, vložíme ji do Lightning Aplikace, kterou si následně jednoduše otevřeme a umožníme tak otestování Komponenty. (19)

V Developer Consoli pro vytvoření Lightning Aplikace klikneme na **File** -> **New** -> **Lightning Application**. (19)

3.3.2.3.3 Struktura

Každá Komponenta a Aplikace je rozdělena do následujících souborů: (19)

Component nebo Application

Obsahuje hlavní zdrojový kód popisující jak se má komponenta zobrazit či referenci dalších vnořených Komponent.

Controller

Jedná se o Controller na straně klienta. Obsahuje JavaScript metody pro řízení generovaných událostí, jako je například stisknutí tlačítka, nebo změna hodnoty ve vstupním poli. Každá metoda obsahuje tři parametry: Component, Event a Helper.

Helper

Představuje druhou a pomocnou vrstvu klientského Controlleru, kde jednotlivé metody mohou být volány libovolně mezi sebou. Z hlediska správné struktury kódu by tento soubor měl obsahovat pouze metody, které obsahují složitější a komplexnější logiku.

Style

Obsahuje CSS styly.

Documentation

Obsahuje textovou dokumentaci. Přidáním dokumentace pomůžeme ostatním uživatelům jednoduše a rychle zjistit funkčnost a použití dané Komponenty či Aplikace.

Renderer

Jakmile je vygenerovaná událost dokončena, framework automaticky zavolá rerender metodu, jež aktualizuje danou komponentu kvůli možným datovým změnám. V souboru Rerender tuto metodu můžeme přepsat a můžeme tak definovat vlastní logiku pro aktualizaci Komponenty.

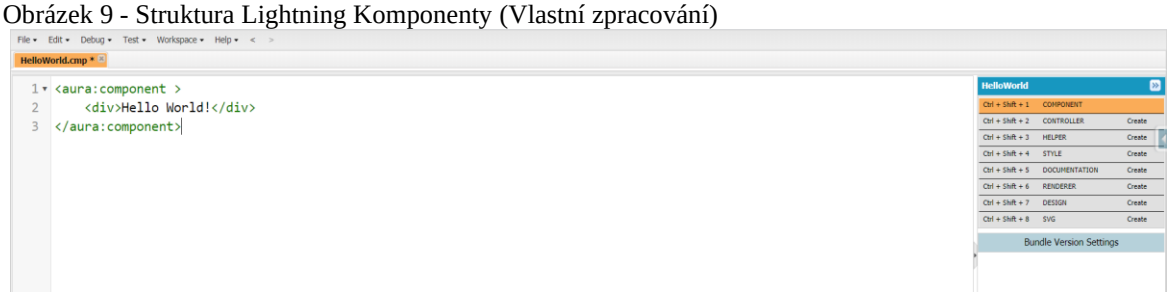
Design

Jakmile programátor dokončí vývoj Komponenty, přichází na řadu práce administrátora, který danou Komponentu skrze Salesforce nástroje vloží do uživatelského rozhraní a uvede ji tak do provozu. V Design souboru můžeme definovat atributy, které administrátor následně může vyplnit a může tím tak ovlivnit vlastnosti a chování Komponenty. Tento atribut je dostupný pouze u Lightning Komponent.

SVG

Zde definujeme zdroj pro ikonku Komponenty či Aplikace.

Obrázek 9 - Struktura Lightning Komponenty (Vlastní zpracování)



3.3.2.3.4 Atributy

Atributy představují proměnné, do kterých můžeme ukládat data. Přidávají se pomocí `<aura:attribute>` tagu. Každý atribut musí mít definované jméno a datový typ. Pomocí jména atributu můžeme následně v hlavním zdrojovém souboru či Controlleru Komponenty s jeho daty manipulovat či je zobrazovat. (19)

4 Vlastní práce

4.1 Analýza požadavků

Před tím, než můžeme vůbec začít s implementací jako takovou, je potřeba si určit a definovat požadavky, které od aplikace požadujeme. V této části práce popíši základní využití dané implementace a definuji její základní funkce. V další části práce popíši jednotlivé funkce detailněji včetně postupu jejich implementace a zprovoznění.

Aplikace bude sloužit všem společnostem, které vlastní a spravují jeden nebo větší množství bankovních účtů s internetovým bankovnictvím u Fio banky a chtějí o nich získat jednoduchý a srozumitelný přehled přímo v prostředí Salesforce, ve kterém pracují. Dále bude sloužit jako doplněk a pomocník v účetních procesech společnosti. Může se jednat například o neziskovou organizaci, která vlastní a spravuje velké množství fondů nebo například jen o firmu, která Salesforce používá pro správu svých proběhlých obchodů a chtějí pro takto využívané procesy zprovoznit určitou automatizaci účetnictví.

Po analýze požadavků požaduji následující funkcionality:

- Ukládání a mazání bankovních účtů a jejich přehled na jedné stránce
- Zobrazení detailu bankovního účtu a seznamu jeho synchronizovaných transakcí
- Ukládání vystavených faktur pro jednotlivé účty
- Automatické stahování a aktualizace transakcí
- Spárování stažených transakcí s vystavenými fakturami pomocí variabilního symbolu a cílového účtu
- Po splacení celé dlužné částky odeslání děkovného emailu

Z hlediska uživatelského rozhraní a klientské části implementace budeme vytvářet stránku v Salesforce, která bude obsahovat dvě záložky.

První záložka bude sloužit pro zobrazení, ukládání a mazání faktur a také pro zobrazení spárovaných přijatých plateb k jednotlivým fakturám. Pro tento požadavek využijeme standartního zobrazení objektů, které Salesforce poskytuje.

Druhá záložka bude sloužit pro interakci s bankovními účty. Tedy pro přidání nových, editaci či zobrazení stávajících účtů a také pro otevření nové stránky s detailem bankovního účtu. V Salesforce nenalezneme žádnou ze standartních funkcionalit, které bychom pro tento požadavek mohli využít. Budeme muset tedy obsah této záložky celý kustomizovat a naprogramovat.

Dále budeme muset vytvořit samostatnou stránku, která bude zobrazovat detail bankovního účtu a jeho stažených transakcí.

Z hlediska serverové části implementace budeme muset naprogramovat třídy - Controllery, které budou komunikovat s naprogramovanými komponentami, respektive klientskou částí aplikace, v rámci interakce uživatele s obsahem stránky a budou mu poskytovat relevantní data a informace z databáze či plnit funkce jako ukládání nebo mazání účtů. Dále potřebujeme naprogramovat sadu tříd, které budou sloužit pro automatické stahování transakcí k daným bankovním účtům na hodinové bázi, párování transakcí skrze variabilní symboly či pro komunikaci s veřejnými API, které poskytují informace o bankovních účtech. Všechny tyto třídy a jejich metody budou navrženy a naprogramovány tak, aby od sebe byly logicky odděleny dle jejich využití a my tak mohli využívat možnosti je znovu použít.

Třídy budeme programovat a oddělovat dle těchto skupin použití:

- Zobrazení a ovládání obsahu stránky – Controller třídy
- Získávání dat z databáze – Selector třídy
- Businessová logika
- Logika definující akce po změně v databázi - Trigger Handler třídy

Vývoj klientské i serverové části bude probíhat paralelně.

4.2 Datový model

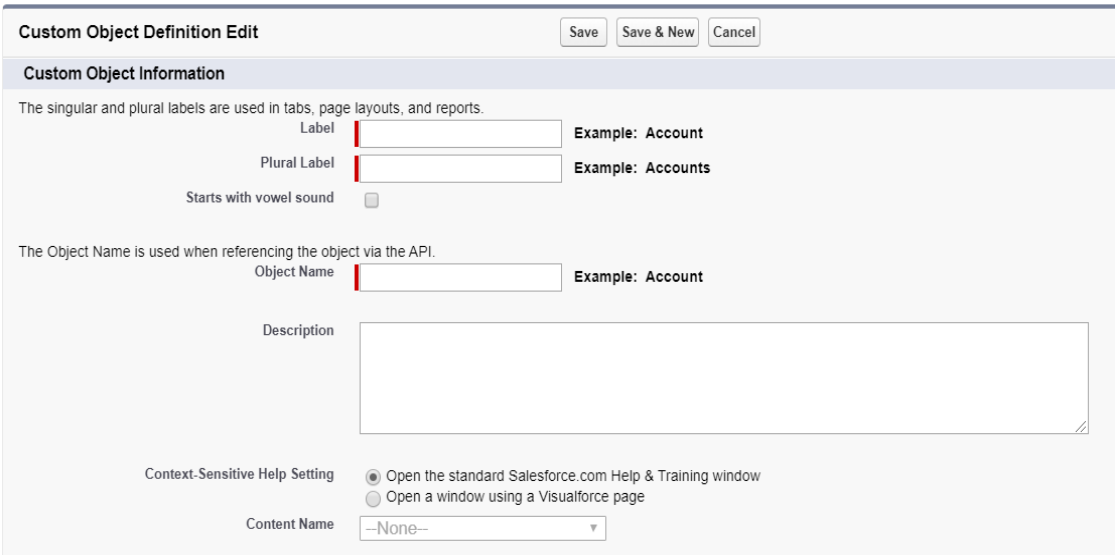
Po analýze požadavků jsem se dobral k tomu, že pro jejich splnění budeme potřebovat vytvořit v databázi tyto tři objekty:

- Bank Account
- Transaction
- Invoice

V Salesforce se objekty tvoří pomoci intuitivního uživatelského rozhraní a každému kdo má alespoň nějakou představu o tom, jak se tvoří tabulky v databázi a jak celkově fungují, nebude tento proces dělat problém. Ještě před tím, něž nějaký objekt vytvoříme, je třeba si definovat všechna pole a jejich datové typy, které bude objekt obsahovat.

K tvorbě objektů se v Salesforce dostaneme kliknutím na **Settings** -> **Object Manager** -> **Create Custom Object**. Následně vyplníme informace o objektu. Z těch nejdůležitějších informací vyplňujeme: Název objektu, API Name objektu, název nově vytvořeného záznamu či formát generování automatických názvů. Pokud nějaké nastavení zapomeneme definovat, či ho chceme zpětně změnit, je to opět možné skrze jednoduché rozhraní pro editaci objektu.

Obrázek 10 - Definice nového Objektu (Vlastní zpracování)



4.2.1 Objekt Bank Account

Jako první vytvoříme objekt Bank Account, který bude představovat jednotlivé bakovní účty. Objekt bude obsahovat pole definující základní informace o bankovním účtu.

Před tím, než začneme skrze uživatelské rozhraní vytvářet jednotlivá pole, musíme se podívat do dokumentace Fio Api bankovnictví, které atributy vrací HTTP dotaz pro získání informací o bankovním účtu. Tyto atributy poté budeme ukládat do námi vytvořených polí v Salesforce. (20)

Pro vytvoření polí si otevřeme v již zmíněném Object Manageru detail vytvořeného objektu a pod sekcí **Fields & Relationships** přes tlačítko **New** vytvoříme jednotlivá pole.

Obrázek 11 - Definice polí v Objektu (Vlastní zpracování)

Dle analýzy poskytovaných atributů jsem v objektu Bank Account vytvořil následující pole a jejich datové typy:

Tabulka 1 - Pole v Objektu Bank Account (Vlastní zpracování)

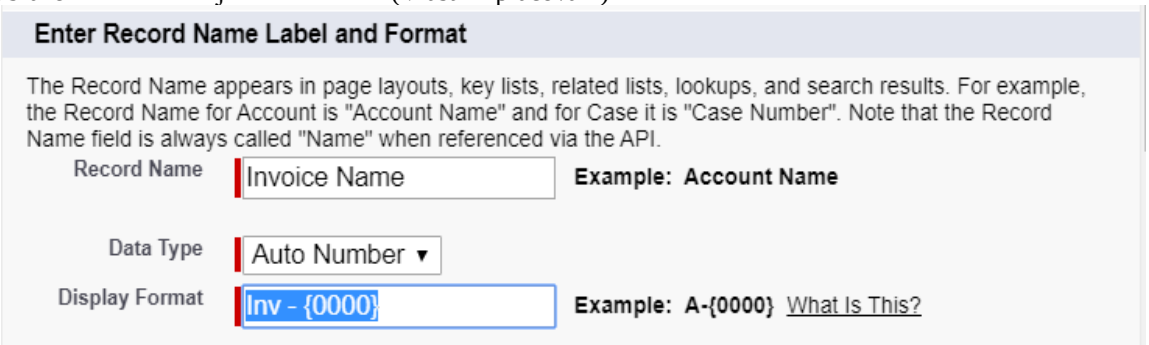
Název pole	**Datový typ**
Account Number	Text
Balance	Number
Bank Code	Text
BIC	Text
Currency	Text
IBAN	Text
Last Sync	Date/Time
Name	Text
Sync From	Date

4.2.2 Objekt Invoice

Objekt Invoice bude sloužit pro ukládání vystavených faktur. Bude obsahovat informace důležité pro automatizaci účetních procesů, jako například variabilní symbol pro párování s přijatými platbami, číslo bankovního účtu, kam se mají platby posílat, či E-mailovou adresu zákazníka kam budeme odesílat děkovný e-mail po splacení celé dlužné částky.

Při vytváření objektu nastavíme, aby se jméno faktury generovalo automaticky ve formátu Inv - {0000}. Jméno takto bude zároveň sloužit jako jedinečný identifikátor faktury.

Obrázek 12 - Formát jména záznamů (Vlastní zpracování)



Dále budeme muset tento objekt propojit vazbou 1:N s objektem Bank Account, pro definici bankovního účtu, na který se mají platby posílat. V Salesforce, před provázáním objektu s jiným objektem vždy nastává otázka, jaký typ provázání použijeme. Pokud nastane situace, kdy smažeme záznam bankovního účtu, tak podřazené záznamy faktur budeme chtít zachovat. Použijeme tedy nepovinnou Lookup vazbu z objektu Invoice na objekt Bank Account.

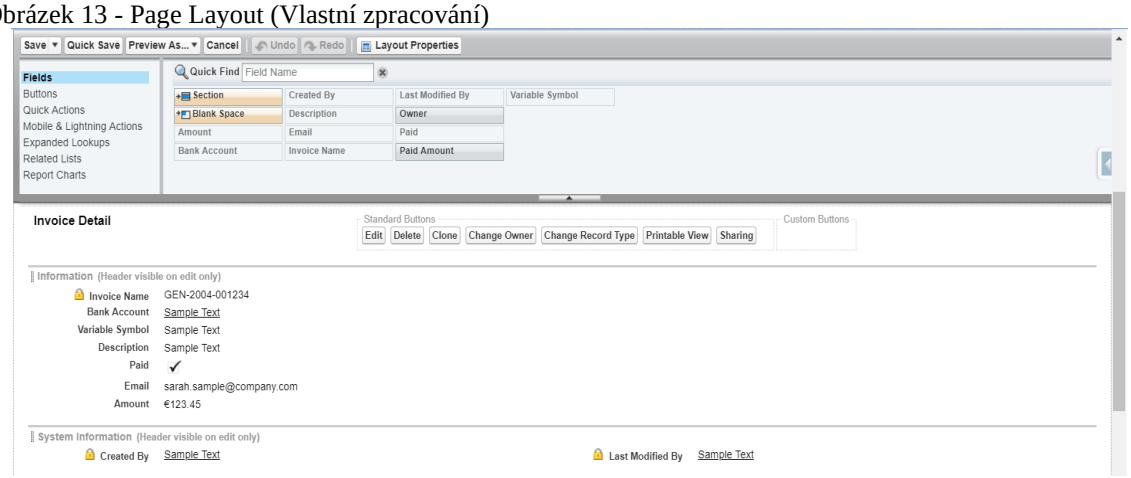
V objektu vytvoříme následující pole:

Tabulka 2 - Pole v Objektu Invoice (Vlastní zpracování)

Název pole	**Datový typ**
Amount	Number
Bank Account	Lookup(Bank Account)
Description	Text
Email	Email
Paid	Checkbox
Paid Amount	Number
Vs	Text

Jelikož pro zobrazování a vytváření záznamů v objektu Invoice budeme využívat standartní uživatelské rozhraní Salesforce, musíme ještě nakonfigurovat jeho Page Layout, kde určíme v jakém pořadí a na jakém místě se jednotlivá pole budou zobrazovat.

Obrázek 13 - Page Layout (Vlastní zpracování)



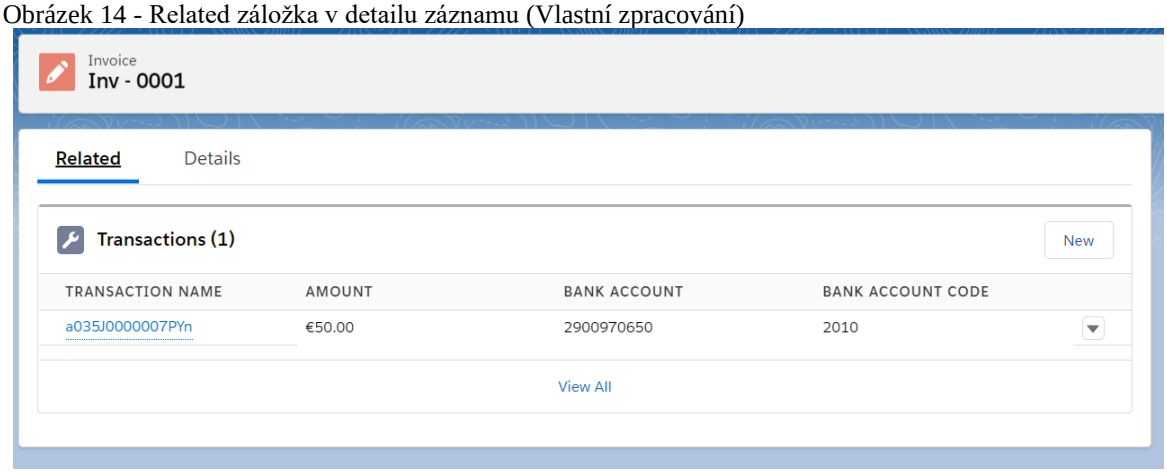
4.2.3 Objekt Transaction

Třetí vytvořený objekt s názvem Transaction představuje jednotlivé stáhnuté transakce k jednotlivým účtům.

Jak již ze samotné definice vyplývá, potřebujeme tento objekt provázat s objektem Bank Account. V tomto případě, pokud smažeme záznam Bankovního účtu, již nedává smysl v Salesforce databázi držet záznamy o jeho stažených transakcích. Pro tento případ se nám tedy hodí použít vazbu 1:N s názvem Master-Detail relationship. Tato vazba automaticky smaže všechny podřazené záznamy po smazání záznamu nadřazeného. Zároveň se stane toto pole, respektive cizí klíč, povinné a není možné uložit záznam transakce bez definování nadřazeného bankovního účtu.

Dále je potřeba vytvořit ještě jednu vazbu, a to na objekt Invoice. Pomocí tohoto provázání poté budeme v Salesforce schopni zobrazit všechny jeho přijaté platby skrze detail faktury a její záložku Related.

Obrázek 14 - Related záložka v detailu záznamu (Vlastní zpracování)



Opět tedy vystává otázka jaký typ provázání zvolit. Musíme si uvědomit, že může nastat případ, kdy stáhneme transakci z bankovního účtu, která nebude obsahovat Variabilní Symbol, a tudíž ji nebudeme schopni provázat se záznamem faktury. Použijeme tedy vazbu 1:N typu Lookup.

V objektu Transaction jsem opět dle analýzy poskytovaných atributů z Fio Api dokumentace vytvořil následující pole a jejich datové typy: (20)

Tabulka 3 - Pole v Objektu Transaction (Vlastní zpracování)

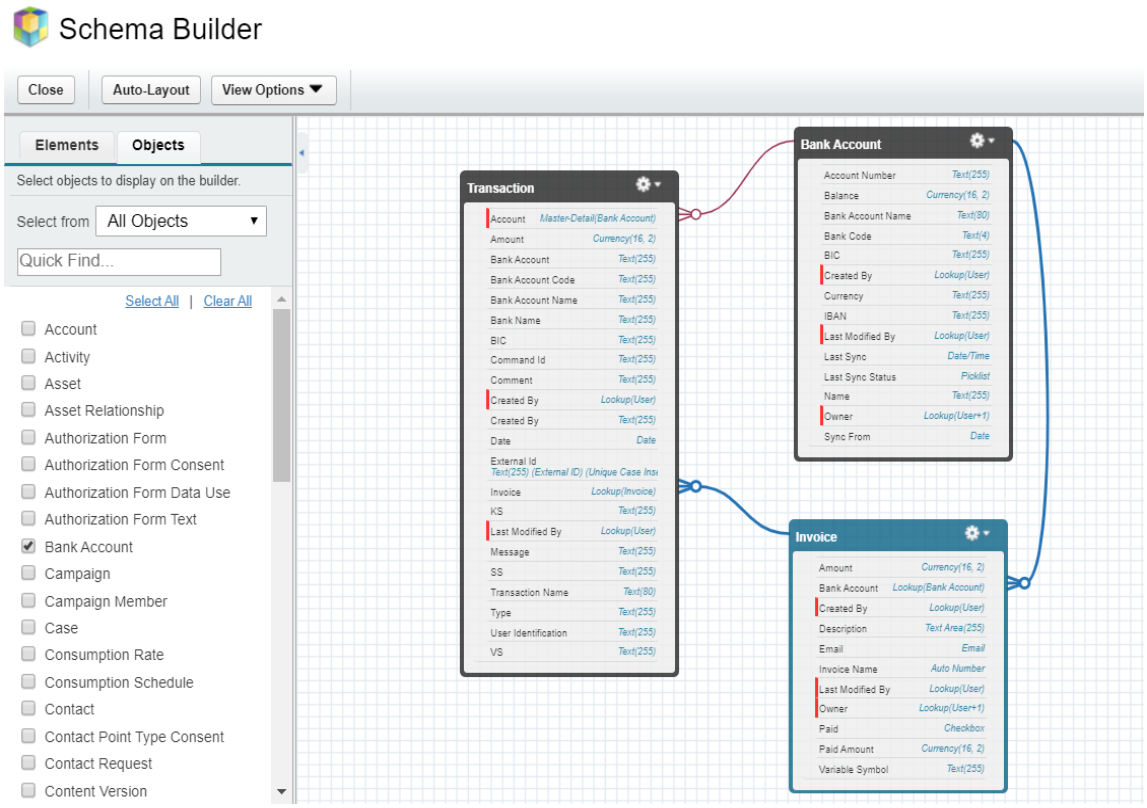
Název pole	**Datový typ**
Account	Master-Detail(Bank Account)
Amount	Number
Bank Account	Text
Bank Account Number	Text
Bank Account Code	Text
Bank Name	Text
BIC	Text
Command Id	Text
Comment	Text
Created By	Text
Date	Date
External Id	Text, External Id, Unique Case Sensitive
Invoice	Lookup(Invoice)
KS	Text
Message	Text
SS	Text
Type	Text
User Identification	Text
VS	Text

Zde je důležité zmínit význam pole External Id použité jako External Id a zároveň Unique Case Sensitive. Při každé synchronizaci transakcí k jednotlivým účtům může nastat případ, kdy nově stažená transakce už je v naší Salesforce databázi uložená. Pro tento

případ potřebujeme nějaký unikátní identifikátor, pomocí kterého dokážeme tuto situaci rozeznat a daný záznam transakce buď vytvořit jako nový, nebo ho pouze aktualizovat. Použití External Id pole je pro tento požadavek ideální volba.

Po vytvoření těchto tří objektů si nyní můžeme námi nově vytvořený datový model zobrazit ve Schema Builderu, pro kontrolu vazeb mezi objekty a celkově pro lepší vizualizaci toho, co jsme právě vytvořili.

Obrázek 15 - Schema Builder (Vlastní zpracování)



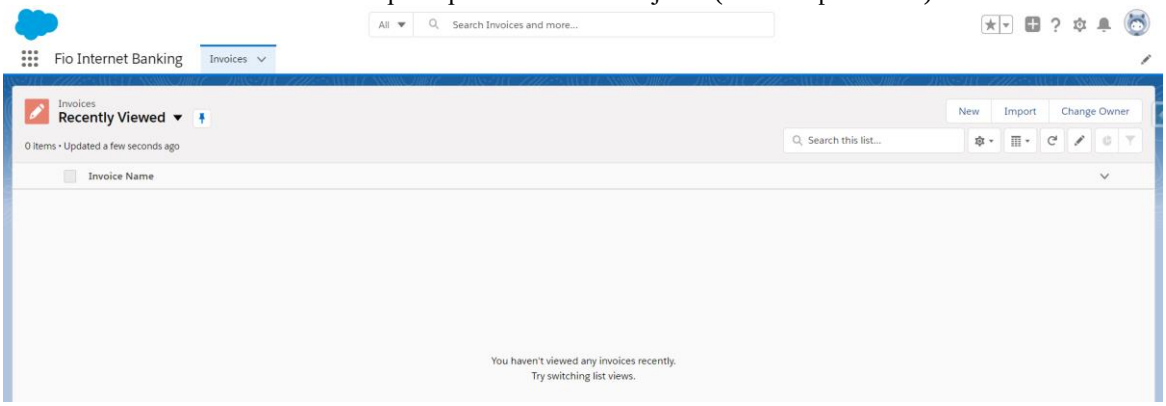
4.3 Implementace

4.3.1 Tvorba uživatelského rozhraní

V tento moment máme veškerou nutnou konfiguraci pro ukládání dat do Salesforce hotovou. Nyní potřebujeme ještě vytvořit místo v uživatelském rozhraní Salesforce, kam budeme vkládat všechny naše naprogramované komponenty a objekty, se kterými budou uživatelé pracovat. Toho docílíme vytvořením nové Lightning Appky, v Salesforce tedy cestou **Settings** -> **App Manager** -> **New Lightning App**, kde vybereme její jméno, popis a další doplňující informace a v posledním kroku zvolíme taby, které bude tato nově

vytvořená Lightning Aplikace obsahovat. Z nabízených tabů vybereme námi už vytvořený objekt Invoice a klikneme na Uložit.

Obrázek 16 - Uživatelské rozhraní s přístupem do Invoice Objektu (Vlastní zpracování)



V tuto chvíli mají uživatelé v rámci námi vytvořené Fio Internet Banking Aplikace přístup k objektu Invoice skrze přidaný tab a můžou si tak vytvářet, editovat či mazat faktury. Nyní nám zbývá do naší Aplikace vytvořit a přidat ještě jeden další tab, který bude manipulovat se dvěmi zbylými objekty - Bank Account a Transaction. Obsah tohoto tabu s názvem Bank Accounts bude plně kustomizovaný a naprogramovaný pomocí Lightning Component a dle analýzy požadavků bude plnit následující funkcionality:

- Přidávání, mazání či editace bankovních účtů
- Výpis všech přidaných bankovních účtů a jejich základních atributů
- Možnost otevřít si detail bankovního účtu v novém okně a zobrazení jeho detailu společně se staženými transakcemi

Obsah tabu Bank Accounts bude vytvořen pomocí tří komponent:

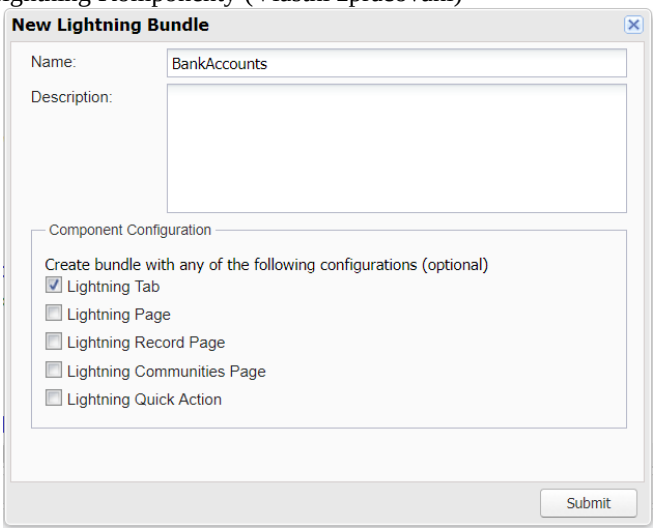
- Bank Accounts
- Bank Account
- Modal

4.3.1.1 Bank Accounts Komponenta

První a zároveň hlavní komponenta, se jménem Bank Accounts, bude obsahovat hlavičku stránky s lištou se základním přehledem o přidaných bankovních účtech a s tlačítkem pro přidání nového účtu. Zároveň bude sloužit jako vrchní modul, který budeme schopni přidat do Tabu, který poté vložíme do naší Fio Internet Banking Aplikace.

Otevřeme tedy Developer Consoli, a vytvoříme novou Lightning Componentu přes **File** -> **New** -> **Lightning Component** a v sekci Component Configuration zaškrtneme políčko Lightning Tab.

Obrázek 17 - Definice Lightning Komponenty (Vlastní zpracování)



Tím nám framework automaticky pro nově vytvořenou komponentu implementuje rozhraní force:appHostable, které nám zajistí požadovanou zmiňovanou funkcionalitu. Pokud bychom políčko zapomněli zaškrtnout, můžeme zpětně pomocí implements atributu, v hlavním tagu komponenty, implementovat námi požadované rozhraní.

```
<aura:component implements="force:appHostable">
```

Nyní je v komponentě ještě potřeba definovat Controller třídu, ze které budeme volat jednotlivé metody pro komunikaci se serverem. Toho docílíme definováním controller atributu v hlavním tagu komponenty, do kterého napíšeme název námi požadované třídy.

V Developer Consoli se jednotlivé třídy vytvářejí skrze cestu **File** -> **New** -> **Apex Class**. Vytvoříme tedy novou třídu s názvem BankAccountController, kterou následně definujeme ve zmíněném controller atributu v Bank Accounts komponentě.

Komponenta bude obsahovat dva Atributy. První se jménem - data a typem proměnné – List a druhý se jménem totalAmount a typem proměnné Decimal. Dále je zde definován Handler typu - Init. Po načtení stránky s otevřenou Bank Accounts komponentou, framework jako první spustí JavaScript metodu definovanou v Init Handleru, která následně zavolá metodu getBankAccounts ze Server-side Controlleru. Metoda z databáze získá všechny bankovní účty, uloží je do vhodného formátu a vrátí zpět

do Client-Side controlleru, který čeká na odpověď ze severu a který data následně uloží do atributu komponenty – Data.

Komponenta poté tento atribut, tedy list bankovních účtů iteruje a pro každý z těchto účtů vloží do DOMu novou podřazenou komponentu Bank Account, které zároveň předá objekt s daty o bankovním účtu a referenci metody onInit. Referenci předáváme proto, abychom mohli po jakékoliv změně zavolat z podřazené komponenty metodu, která znovu načte data a aktualizuje obsah stránky.

```
<aura:iteration items="{!v.data}" var="i">
    <div class="slds-m-bottom_x-small">
        <c:BankAccount accountData="{!i}" refreshPage="{!c.onInit}" />
    </div>
</aura:iteration>
```

4.3.1.2 Bank Account Komponenta

Komponentu Bank Account si lze představit jako detail jednoho řádku v seznamu všech bankovních účtů vygenerovaného rodičovskou komponentou Bank Accounts. Bude fungovat na bázi přepoužitelné šablony. To znamená, že pokud z databáze získáme pět bankovních účtů, budeme mít pět Bank Account komponent, kde každá z nich obsahuje a pracuje s daty pouze jednoho bankovního účtu.

Budeme zde zobrazovat pouze základní informace, které uživateli poskytnou základní přehled o účtu. Konkrétně tedy číslo a jméno účtu, zůstatek, datum poslední synchronizace a počáteční synchronizace. Číslo účtu bude sloužit jako odkaz pro přesměrování na detail bankovního účtu.

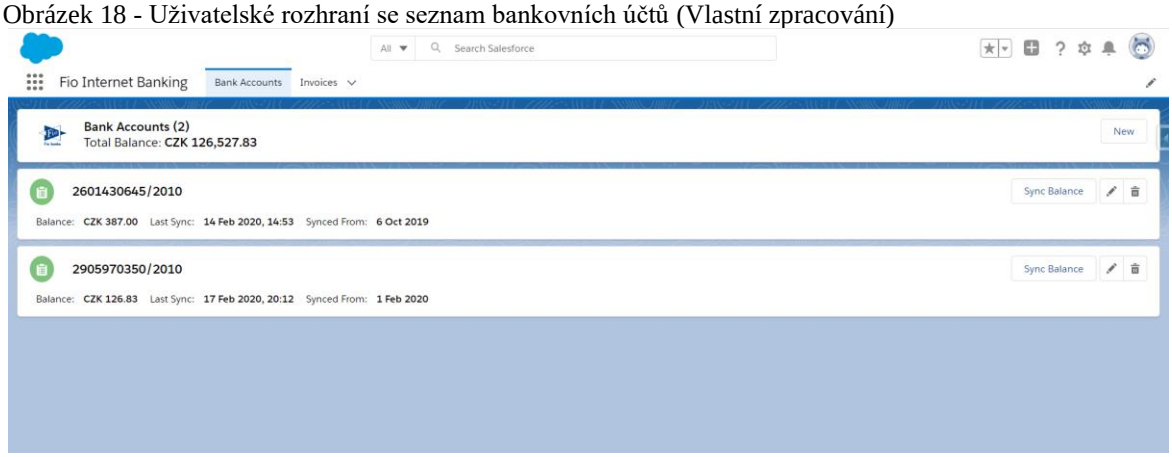
Dále zde budou tlačítka pro editaci a mazání účtu, či rychlou synchronizaci jeho zůstatku.

4.3.1.3 Modal Komponenta

Komponenta Modal přestavuje modální okno, které si bude uživatel dle libosti zobrazovat a schovávat. Představuje opět určitou šablonu, ve které definujeme její základní vzhled, jako jsou okraje, tlačítka či šířku zobrazení, avšak její obsah bude dynamický a pro každou definici této komponenty jiný. Toho docílíme přidáním {!v.body} atributu v těle komponenty. Díky tomuto provedení můžeme v jakékoliv komponentě definovat libovolné množství modálních oken, kde každé z nich bude obsahovat něco jiného.

Například v komponentě Bank Account používáme modální okna dvě. Jedno okno pro editaci účtu, které obsahuje vstupní pole, jako je jméno účtu a datum, od kdy se má účet synchronizovat a druhé okno pro smazání účtu, které obsahuje pouze potvrzovací hlášku pro smazání.

Obrázek 18 - Uživatelské rozhraní se seznam bankovních účtů (Vlastní zpracování)



Nyní máme definované a vytvořené komponenty pro zobrazení seznamu bankovních účtů z hlediska jejich struktury, obsahu a využití. V následující části mé práce popíši jejich dílčí funkcionality.

4.3.2 Implementace dílčích funkcionalit

4.3.2.1 Přidání nového účtu

Před tím, než budeme schopni přidávat nové bankovní účty do Salesforce, musíme nejprve zprovoznit volání veřejných REST API poskytovaných Fio Bankou, pro získávání a následné ukládání požadovaných dat o bankovních účtech. (20)

Api bankovnictví Fio banky poskytuje šest možností získávání dat o bankovním účtu: (20)

- Pohyby na účtu za určité období
- Oficiální výpisy pohybů z účtu
- Pohyby na účtu od posledního stažení
- Nastavení zarážky
- Karetní transakce obchodníka za určené období
- Číslo posledního vytvořeného oficiálního výpisu

Pro potřeby naší aplikace a její celkové implementace, nám vystačí možnost získat pohyby na účtu za určité období.

Obrázek 19 - Struktura dotazu (Internetový zdroj dle (20))

Struktura:	https://www.fio.cz/ib_api/rest/periods/{token}/{datum od}/{datum do}/transactions.{format}	
	Token	unikátní vygenerovaný token
	Datum od	datum - začátek stahovaných příkazů ve formátu rok-měsíc-den (rrrr-mm-dd)
	Datum do	datum - konec stahovaných příkazů ve formátu rok-měsíc-den (rrrr-mm-dd)
	Formát	formát pohybů
Příklad:	Získání pohybů v období od 25.8.2012 do 31.8.2012 v xml	
https://www.fio.cz/ib_api/rest/periods/aGEMQB9Idh35fh1g51h3ekkQwyGIQ/2012-08-25/2012-08-31/transactions.xml		

Ze sktruktury dotazu a jeho atributů vidíme, že pro tento dotaz jsou potřeba čtyři atributy. Pro vytvoření nového účtu budeme od uživatele potřebovat jen dva a to - vygenerovaný API token z jeho Fio internetového bankovnictví pro účet, který chce přidat a datum, od kdy se mají ukládat a synchronizovat jeho transakce. Zbylé dva atributy doplníme automaticky. Datum do bude pro tento případ vždy dnešní datum a jako Formát budeme vždy používat JSON. Atribut formát nám určuje, v jakém formátu chceme dostávat odpovědi.

JSON odpověď je v následujícím formátu, kde jsou pro reprezentativní případ zobrazeny pouze její hlavní atributy:

```
{"accountStatement":
    {
        "info":{....},
        "transactionList":{"transaction":[...., ...., ....]}
    }
}
```

Odpověď je celá obalena do atributu accountStatement, který následně obsahuje dva, pro nás hlavní, atributy: info a transactionList. V atributu info nalezeneme informace o účtu jako takovém a v atributu transactionList poté seznam jeho transakcí a jejich náležité informace.

Pro každý ze zmíněných atributů v JSON odpovědi musíme vytvořit další oddělenou vnitřní třídu, do které následně budeme daný atribut i s jeho proměnnými ukládat.

```
public class Statement {
    public Info accountStatement;
}
public class Info {
    public BankAccount info;
    public TransactionList transactionList;
}
public class TransactionList {
    public List<TransactionX> transaction_x;
}
```

Pro volání dotazů a vracení jejich odpovědí nám bude sloužit metoda readData ve třídě FioConnector.

```
public static Statement readData(String token, Date syncFrom, Date syncTo) {
    HttpRequest req = new HttpRequest();
    String endpoint = 'https://www.fio.cz/ib_api/rest/periods/' + token;
    endpoint += '/' + formatDate(syncFrom == null ? System.today() : syncFro
m);
    endpoint += '/' + formatDate(syncTo == null ? System.today() : syncTo) +
 '/transactions.json';
    req.setEndpoint(endpoint);
    req.setMethod('GET');
    Http h = new Http();
    HttpResponse res = h.send(req);
    String s = res.getBody().replace('"transaction":', '"transaction_x":');
    Statement st = (Statement) JSON.deserialize(s, Statement.class);
    return st;
}
```

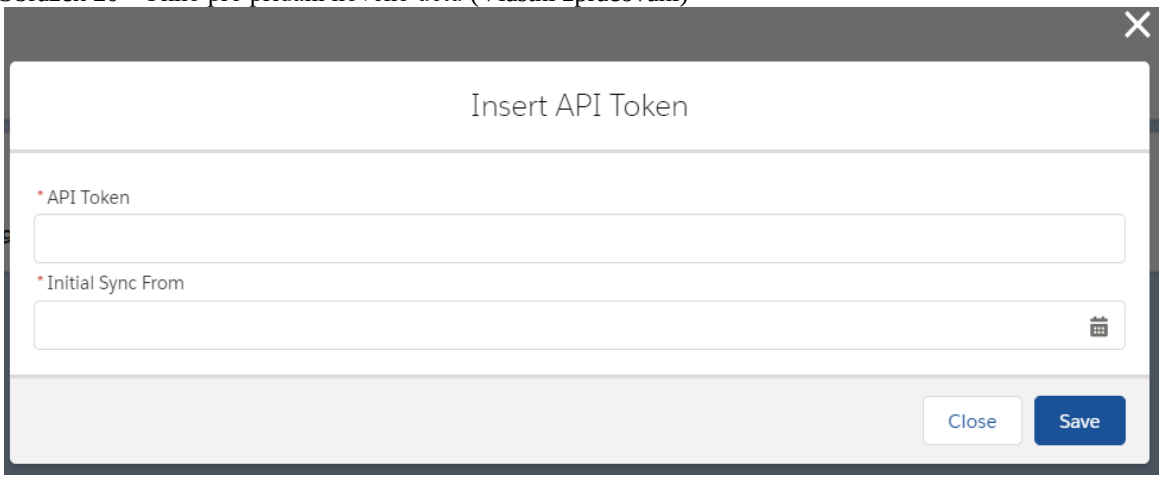
Metoda obsahuje tři atributy: token, synchronizace od a synchronizace do, pomocí kterých se dynamicky poskládá URL dotaz, a poté se pomocí metody GET a HTTP protokolu požadovaný dotaz zavolá.

Po úspěšném zavolání dotazu odpověď nejdříve uložíme do proměnné typu String a následně data rozparsujeme do výše zmíněných tříd pomocí systémové metody deserialize volané ze třídy JSON. V tuto chvíli máme data uložena v námi definovaném formátu, přístup k nim je umožněn jednoduše přes tečkovou notaci a můžeme tak odpověď z metody vrátit tomu, kdo ji zavolal.

Serverová část je nyní připravena, a je na čase umožnit uživateli tuto funkncionalitu využívat z klientské části aplikace. Konkrétně tedy pro přidávání nových bankovních účtů. K tomu slouží tlačítko New v horní pravé části obrazovky. Po stisknutí tlačítka se uživateli

zobrazí modální okno, kam zadá API token a datum od kdy se mají transakce synchronizovat a stiskne tlačítko Save.

Obrázek 20 - Okno pro přidání nového účtu (Vlastní zpracování)



Po stisknutí tlačítka Save se zavolá metoda saveAccount definovaná v Sever-Side Controlleru. Tato metoda zavolá REST dotaz pomocí metody popsané výše a její odpověď uloží do námi definovaných objektů v databázi. Data o bankovním účtu do objektu Bank Account a jednotlivé transakce do objektu Transactions.

Nyní je potřeba ještě zkontrolovat, zda některé z ukládaných transakcí nepředstavují platby pro námi vystavené faktury. Pokud ano, je potřeba pro tyto transakce ještě před uložením vyplnit jejich cizí klíče představující vazbu na objekt Invoice pro spárování transakce s vystavenou fakturou.

Pro tento případ využijeme Before Triggeru definovaného na objektu Transactions, který se spustí ještě před uložením záznamů do databáze a umožní nám tak záznamy ještě dodatečně upravit. V triggeru pomocí proměnné Trigger.isBefore zjistíme, zda-li jsme v kontextu Before Triggeru, tedy ve fázi procesu ukládání záznamů, ještě před jejich uložením do databáze a pokud ano, zavoláme metodu checkRelatedInvoices, definovanou ve třídě TransactionTriggerHandler, které pomocí proměnné Trigger.new předáme List obsahující instance nově vytvářených záznamů pro Transaction objekt.

```
Trigger TransactionTrigger on Transaction__c
(after insert, after update, before insert, before update) {
    if (Trigger.isBefore) {
        TransactionTriggerHandler.checkRelatedInvoices(Trigger.new);
```

```
    }

    if (Trigger.isAfter) {
        TransactionTriggerHandler.processIncomeTransactions(Trigger.new);
    }
}
```

Metoda následně pro každou transakci typu příjem zkontroluje, jestli v databázi máme uloženou fakturu, která obsahuje shodné číslo bankovního účtu a variabilního symbolu, definované v kontrolované transakci. Pokud ano, vytáhneme si z databáze skrze tyto hodnoty Id záznamu dané faktury a vyplníme ho v této transakci. Po provedení těchto dodatečných úprav pokračuje proces ukládání záznamů, a jelikož v části Before Triggeru nemáme definované žádné další metody, záznamy se uloží do databáze.

Jakmile na straně serveru a jeho příslušné databáze proběhne v rámci ukládání transakcí vše bez problémů, tak ještě před samotným Commitem, tedy finálním potvrzením transakce, přichází na řadu exekuce After Triggeru. V After Triggeru jsou záznamy transakcí přístupné pouze pro čtení, proto se také tato část procesu ukládání dat využívá především pro aktualizaci souvisejících dat z jiných objektů. Toho využijeme a můžeme tak editovat splacenou částku definovanou v souvisejících záznamech faktur, pro které jsme právě přijali platby. V triggeru tedy zavoláme metodu processIncomeTransactions opět z třídy TransactionTriggerHandler, které opět předáme List instancí záznamů Transaction Objektu. Metoda následně pomocí částky definované v záznamu transakce aktualizuje výši splacené částky v dané faktuře a pokud už je splátka celá splacená, odešle děkovný e-mail na e-mailovou adresu uloženou v záznamu faktury.

V poslení fázi transakce ukládání nového bankovního účtu je odpověď vrácena do Client-Side Controlleru naší Bank Accounts komponenty, která aktualizuje stránku a zobrazí nově přidaný bankovní účet.

4.3.2.2 Editace přidaných účtů

Poté co uživatel úspěšně přidá bankovní účet, bude mu umožněno tento účet editovat. Uživatel si bude schopen svůj účet dle libosti pojmenovat, pro lepší identifikaci o jaký účet se jedná. Dále bude schopen zpětně upravit datum, od kdy se mají synchronizovat jeho transakce.

Uživatel jednoduše klikne na tlačítko editace u účtu, který chce editovat, změní hodnoty a klikne na tlačítko Save. Pokud je nové datum pro synchronizaci transakcí zadané

uživatelem více aktuální, je potřeba smazat všechny transakce, které jsou starší než datum od kdy se má účet synchronizovat. Pro tento případ opět využijeme Trigger definovaný na objektu Bank Account.

V Triggeru pomocí proměnné Trigger.isUpdate zjistíme, že jsme v kontexu aktualizace záznamů, nikoliv jejich ukládání a poté zavoláme metodu handleUpdatedAccounts ze třídy BankAccountTriggerHandler a předáme jí mapy záznamů bankovních účtů obsahující hodnoty před a po aktualizací.

```
Trigger BankAccountTrigger on Account__c (before update) {
    if (Trigger.isUpdate && Trigger.isBefore) {
        BankAccountTriggerHandler.handleUpdatedAccounts(Trigger.newMap, Trig
ger.oldMap);
    }
}
```

Metoda pomocí předaných map následně zkontroluje, zda je nově zadané datum pro synchronizaci starší či novější, než datum definované před aktualizací. Pokud je novější, zavoláme metodu deleteOlderTransactions, která smaže všechny neaktuální transakce k danému účtu. Pokud je starší, musíme provést dodatečnou synchronizaci transakcí. Synchronizaci provedeme pomocí, už námi použité metody, readData ve třídě FioConnector .

4.3.2.3 Přesměrování na detail bankovního účtu

Nyní potřebujeme v Salesforce vytvořit stránku, která bude dosažitelná pomocí definované unikátní URL adresy a na kterou se bude uživatel moci přesměrovat přímo ze seznamu zobrazovaných bankovních účtů a zobrazit si tak jejich detail.

Toho docílíme vytvořením Lightning Aplikace. V Developer Consoli tedy klikneme na File -> New -> Lightning Application a jméno aplikace definujeme jako InternetBankingApp. Toto jméno bude zároveň sloužit jako definice cesty v URL do této námi vytvářené stránky. Konkrétně tedy přidáním /c/InternetBankingApp.app za základní URL adresu naší Salesforce organizace.

Zároveň ale potřebujeme této stránce nějakým způsobem sdělit, jaký bankovní účet otevíráme. Salesforce poskytuje force:hasRecordId rozhraní, které Lightning Komponentám a Aplikacím po jeho implementaci umožňuje z URL adresy extrahovat recordId atribut. Při procesu přesměrovávání na tuto stránku musíme tedy zajistit, aby URL adresa obsahovala ještě Id záznamu bankovního účtu, který otevíráme. Toto bude

definováno v Bank Account komponentě a dosaženo pomocí kombinace href atributu, a tagu a jejího accountData atributu, který obsahuje Id záznamu.

```
<a href="{!'/c/InternetBankingApp.app?recordId=' + v.accountData.recordId}">
```

Po načtení InternetBankinkApp stránky si opět pomocí definovaného handleru typu init a pomocí recordId hodnoty z databáze vytáhneme všechny informace k danému účtu a zobrazíme uživateli na frontendu. Konkrétně tedy v hlavičce stránky informace o otevřeném bankovním účtu a v těle stránky rolovatelnou tabulku obsahující jednotlivé stažené transakce a jejich informace a tlačítko pro ruční synchronizaci nejnovějších transakcí.

4.3.2.4 Automatizace synchronizace transakcí

Nyní chceme pro bankovní účty, které jsme přidali do Salesforce, automaticky stahovat a aktualizovat jejich transakce na hodinové bázi.

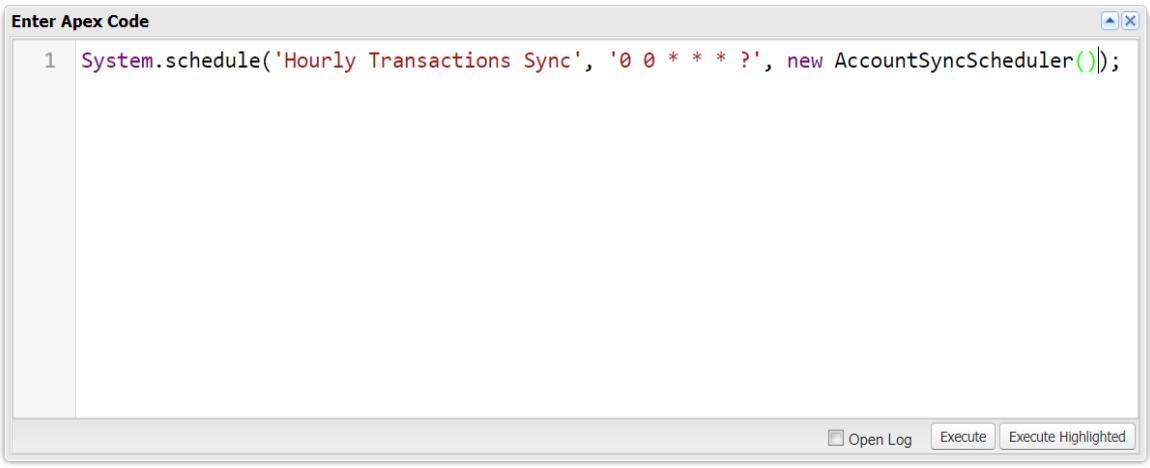
Salesforce poskytuje možnost nastavit automatické exekuce třídy, respektive její execute metody, v konkrétní čas na pravidelné bázi. Abychom toho byli schopni, musíme primárně třídě implementovat Schedulable rozhraní a poté definovat execute metodu, ve které bude daná vykonávací logika. Jakmile toto uděláme, můžeme pro tuto třídu naplánovat automatickou exekuci dvěma způsoby. Spuštěním kódu pomocí Execute Anonymous Apex nástroje v Developer Consoli nebo skrze deklarativní nastavení v Salesforce. Deklarativní nastavení nám ale umožňuje třídu spouštět pouze na denní či měsíční bázi, tudíž využijeme způsobu druhého. A to konkrétně zavoláním systémové metody Schedule ze třídy System. Tato metoda obsahuje tři argumenty: jméno pro takto vytvořené naplánování, CRON výraz pro definici času a data exekuce a jméno spouštěné třídy.

Nyní potřebujeme sestavit CRON výraz, který bude obsahovat definici spouštění na hodinové bázi, každý den v měsíci. Sestavuje se pomocí sekvence pěti po sobě jdoucích znaků dle jejich významu z hlediska času exekuce v následujícím pořadí: Sekunda, Minuta, Hodina, Den v měsíci, Měsíc, Den v týdnu. My chceme spouštět automatickou synchronizaci transakcí každý den a každou hodinu. Použijeme tedy následující CRON výraz: **0 0 * * * ?**. Hvězdička znamená použití všech hodnot a otazník vyjadřuje nespecifikaci hodnoty. Náš sestavený výraz tedy z hlediska pořadí znaků definuje

následující čas exekuce: v 0 sekund, v 0 minut, každou hodinu, každý den v měsíci a každý měsíc nehledě na tom, jaký den v týdnu je.

Nyní nám už zbývá jen zavolat zmiňovanou Schedule metodu v anonymním okně v Developer Consoli. Editor pro spouštění anonymního kódu otevřeme cestou **Debug** -> **Open Execute Anonymous Window**.

Obrázek 21 - Execute Anonymous okno (Vlastní zpracování)



5 Výsledky a diskuse

Při různých diskuzích o mé bakalářské práci a jejího přínosu pro firmy z hlediska usnadnění jejich účetních procesů, jsem se často setkával s reakcemi typu: „Toto by se určitě hodilo i u nás ve firmě, je možné tam tvoji aplikaci začít používat?“. Z této reakce jsem měl vždy radost, nicméně je nutno podotknout, že tato konkrétní integrace a její funkcionality jsou ušité přímo na míru a fungují pouze na platformě Salesforce. Nicméně základní myšlenku, rozšířit Salesforce o funkcionality pomáhající účetním procesům firmy za použití veřejně dostupných dat z internetového bankovnictví, lze po nové a specifické integraci určitě použít na jakémkoliv jiném CRM systému.

Takto získaná data o pohybech na bankovních účtech firmy lze v CRM systému použít na nespočetně mnoho dalších věcí. Kromě těch, které jsou zprovozněny v mé práci, lze uvažovat například o reportingu nad bankovními účty, posílání plateb přímo z platformy Salesforce či generování faktur a jejich zasílání společně s e-maily. Z toho také vyplývá vize autora ohledně této integrace – zprovoznit komunikaci s internetovým bankovnictvím a vytvořit tak základní funkcionality pro stahování a ukládání těchto dat. Firmy tak následně tuto integraci mohou využít jako základ pro jejich komplexní a specifické účetní aplikace běžící na platformě Salesforce.

Tato práce se však zabývá integrací internetového bankovnictví pouze Fio banky. Jako možné rozšíření této práce tak lze určitě uvažovat o přidání možnosti stahovat transakce i u jiných bankovních společností, než je Fio banka.

6 Závěr

Cílem této práce bylo seznámení s pojmem CRM systém a jeho řešením v podobě platformy Salesforce a také se souvisejícími technologiemi používanými pro tvorbu aplikací na této platformě či pro rozšiřování jejích funkcionalit. Toho bylo úspěšně dosaženo v teoretické části této práce, která vycházela především z informací získaných studiem odborné literatury dostupné online. Téměř veškeré informace o platformě Salesforce byly čerpány z oficiální dokumentace vydávané touto firmou, u které se můžeme spolehnout, že je aktuální.

Dalším a zároveň hlavním cílem práce bylo vytvoření integrace na platformě Salesforce, která demonstruje možné využití takto získaných a ukládaných dat z internetového bankovnictví Fio banky. Tento cíl se mi podařilo splnit v praktické části mé práce, kde popisuji veškerý postup a náležitosti potřebné pro zprovoznění této integrace. Pro veškerý vývoj a konfiguraci spojenou s platformou Salesforce jsem využíval mých praktických znalostí získaných z mého současného zaměstnání na pozici Salesforce Developera spolu s teoretickými znalostmi z odborné literatury. Vlastní práce začíná přípravou prostředí Salesforce v podobě vytvoření prostoru, kde se bude nacházet námi tvořené uživatelské rozhraní a následné vytvoření datového modelu. Po dokončení přípravy prostředí a její základní konfigurace následuje samotný vývoj. I přes to, že vývoj jednotlivých komponent, tedy jejich klientských a serverových částí probíhá paralelně a tudíž je komplikované tento postup vývoje nějak logicky a uceleně popsat, jsem se snažil být co nejsrozumitelnější a co nejvíce ho oddělovat z hlediska logických celků. Popis vývoje tedy začíná vytvářením klientských částí komponent, kde popisuji jejich vzhled, strukturu a rozvržení jejich obsahu, například v podobě umístění jednotlivých tlačítek. V další části popisu vývoje se již věnuji implementaci a zprovoznění jejich jednotlivých funkcionalit a veškeré logiky programované na serverové části implementace.

7 Seznam použitých zdrojů

1. Hostinger. What is HTML? The Basics of Hypertext Markup Language Explained [online]. 2019-11-25 [cit. 2020-03-3]. Dostupné z WWW: https://www.hostinger.com/tutorials/what-is-html
2. Hostinger. What is CSS [online]. 2019-01-18 [cit. 2020-03-3]. Dostupné z WWW: https://www.hostinger.com/tutorials/what-is-css
3. Hostinger. What is JavaScript? A Basic Introduction to JS for Beginners [online]. 2019-11-25 [cit. 2020-03-3]. Dostupné z WWW: https://www.hostinger.com/tutorials/what-is-javascript
4. MuleSoft. What is an API? (Application Programming Interface) [online]. [cit. 2020-03-3]. Dostupné z WWW: https://www.mulesoft.com/resources/api/what-is-an-api
5. SmartyStreets. What is HTTP? [online]. [cit. 2020-03-3]. Dostupné z WWW: https://smartystreets.com/articles/what-is-http
6. CloudFlare. What is HTTPS? [online]. [cit. 2020-03-3]. Dostupné z WWW: https://www.cloudflare.com/learning/ssl/what-is-https/
7. Codecademy. What is REST? [online]. [cit. 2020-03-3]. Dostupné z WWW: https://www.codecademy.com/articles/what-is-rest
8. JSON. Introducing JSON [online]. [cit. 2020-03-3]. Dostupné z WWW: https://www.json.org/json-en.html
9. Tutorialspoint. MVC Framework - Introduction [online]. 2019-11-25 [cit. 2020-03-3]. Dostupné z WWW: https://www.tutorialspoint.com/mvc_framework/mvc_framework_introduction.htm
10. SuperOffice. What is CRM? The Definitive Guide to CRM Success (Strategy, Process, Trends) [online]. 2020-02-13 [cit. 2020-02-10]. Dostupné z WWW: https://www.superoffice.com/blog/what-is-crm/
11. TheStreet. What is Salesforce and What Does It Do in 2020? [online]. 2020-01-9 [cit. 2020-02-10]. Dostupné z WWW: https://www.thestreet.com/technology/what-is-salesforce-14796378
12. Trailblazer. Sales Cloud Basics [online]. [cit. 2020-02-10]. Dostupné z WWW: https://help.salesforce.com/articleView?id=sales_core.htm&type=5

13. Trailblazer. Service Cloud [online]. [cit. 2020-02-10]. Dostupné z WWW: https://help.salesforce.com/articleView?id=service_cloud.htm&type=5
14. Trailblazer. Salesforce Communities Overview [online]. [cit. 2020-02-10]. Dostupné z WWW: https://help.salesforce.com/articleView?id=networks_resources.htm&type=5
15. Trailhead. Marketing Cloud Basics [online]. [cit. 2020-02-10]. Dostupné z WWW: https://trailhead.salesforce.com/en/content/learn/modules/mrkt_cloud_basics
16. Developers. Salesforce Data Security Model — Explained Visually [online]. 2017-04-10 [cit. 2020-02-10]. Dostupné z WWW: https://developer.salesforce.com/blogs/developer-relations/2017/04/salesforce-data-security-model-explained-visually.html
17. Trailblazer. Developer Console [online]. [cit. 2020-02-10]. Dostupné z WWW: https://help.salesforce.com/articleView?id=code_dev_console.htm&type=5
18. Developers. Getting Started with Apex [online]. [cit. 2020-02-10]. Dostupné z WWW: https://developer.salesforce.com/docs/atlas.en-us.apexcode.meta/apexcode/apex_intro_get_started.htm
19. MST Solutions. Lightning Component Framework [online]. [cit. 2020-02-10]. Dostupné z WWW: https://www.mstsolutions.com/technical/lightning-component-framework/
20. Fio banka. FIO API BANKOVNICTVÍ [online]. 2019 [cit. 2020-02-10]. Dostupné z WWW: https://www.fio.cz/docs/cz/API_Bankovnictvi.pdf

8 Přílohy