Mendelova univerzita v Brně
Provozně ekonomická fakulta

Informační systém pro reporting výkonu a úspěšnosti reklamních kampaní

Bakalářská práce

Vedoucí práce:

Ing. Oldřich Faldík

Lukáš Vožda

Brno 2016

Děkuji vedoucímu bakalářské práce Ing. Oldřichu Faldíkovi za vedení mé práce, za cenné rady, připomínky, čas a tipy na zajímavé technologie, které byly v této práci využity. Zároveň děkuji obchodnímu řediteli agentury PROFICIO, s.r.o. Ing. Petru Halíkovi za poskytnutí informací a konzultací nezbytných ke správnému navržení systému.

Čestné prohlášení

Prohlašuji, že jsem tuto práci: **Informační systém pro reporting výkonu a úspěšnosti reklamních kampaní**
vypracoval/a samostatně a veškeré použité prameny a informace jsou uvedeny v seznamu použité literatury. Souhlasím, aby moje práce byla zveřejněna v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách ve znění pozdějších předpisů, a v souladu s platnou *Směrnicí o zveřejňování vysokoškolských závěrečných prací.*

Jsem si vědom/a, že se na moji práci vztahuje zákon č. 121/2000 Sb., autorský zákon, a že Mendelova univerzita v Brně má právo na uzavření licenční smlouvy a užití této práce jako školního díla podle § 60 odst. 1 Autorského zákona.

Dále se zavazuji, že před sepsáním licenční smlouvy o využití díla jinou osobou (subjektem) si vyžádám písemné stanovisko univerzity o tom, že předmětná licenční smlouva není v rozporu s oprávněnými zájmy univerzity, a zavazuji se uhradit případný příspěvek na úhradu nákladů spojených se vznikem díla, a to až do jejich skutečné výše.

V Brně dne 23. května 2016 _____

Abstract

Vožda, L. Information system for reporting success and performance of marketing campaigns. Brno: Mendel University, 2016.
The bachelor thesis deals with automatization of reporting system for online-marketing agency PROFICO, s.r.o. At this agency, reports are every month repeating routine which comes with unnecessary costs and alternative costs in the form of time that employees spend on reports. Goal of this thesis is to design a solution that allows automatically and easily create reports through clear and simple visualization of data. System will be implemented with the web framework Django, written in the programming language Python and deployed on a cloud IBM Bluemix.

Keywords

Information system, Python, Django, MySQL, Javascript, IBM Bluemix

Abstrakt

Vožda, L. Informační systém pro reporting výkonu a úspěšnosti reklamních kampaní. Brno: Mendelova Univerzita v Brně, 2016.
Tato bakalářská práce se zabývá řešením automatizace reportingového systému pro online-marketingovou agenturu PROFICIO, s.r.o. Reporty představují pro agenturu každý měsíc se opakující rutinní činnost, která přináší nadbytečné finanční náklady a oportunitní náklady v podobě času, který nad reporty tráví zaměstnanci společnosti. Cílem práce je navrhnout takové řešení, které umožní automatizovaně a jednoduše tvořit reporty prostřednictvím přehledné a jednoduché vizualizace dat. Systém bude implementován ve webovém frameworku Django napsaném v programovacím jazyce Python, nasazen bude cloudu IBM Bluemix.

Klíčová slova

Informační systém, Python, Django, MySQL, Javascript, IBM Bluemix

Obsah

1 Úvod a cíl práce

1.1 Úvod

Automatizace rutinních činností je v posledních letech čím dál více probíraným tématem všech společností. Snižování nákladů je základ pro úspěch v konkurenčním boji, nejde ale však jen o snižování nákladů. Roli hraje také čas – rychlejší vyhrává, dalším faktorem je nedostatek profesionálů v oboru, proto jim chceme uvolnit ruce a dát prostor věnovat se pokročilejším problémům a činnostem.

Online-marketingové prostředí je známé svými možnostmi automatizace, která je na určitých místech výslovně nezbytná, protože v reálném čase dochází ke zpracovávání velkého množství dat, na které časem člověk přestává stačit. Dobře fungující a prosperující agentuře v tomto odvětví v čase roste počet klientů a objem prací prudce roste. V usnadnění těchto prací tkví hlavní myšlenka této práce.

1.2 Cíl práce

Cílem práce je navrhnout takové řešení, které přinese společnosti časovou úsporu každý měsíc, při tvorbě reportů. Zároveň je žádoucí dodržet jednoduchost a udržovatelnost aplikace, aby do budoucna bylo co nejméně zapotřebí technických zásahů či asistence kvalifikované osoby.

Reportingový systém bude fungovat jako jednoduchý informační systém, který umožní klientům instantní přístup k datům, která jsou pro ně zásadní, naopak bude filtrovat data, která klienty nezajímají. Řešení je nutné připravit tak, aby bylo přístupné odkudkoliv, z různých druhů zařízení, aby bylo uživatelsky přívětivé i pro uživatele, kteří nejsou zdatní v používání informačních technologií. Data musí být vizualizována přehledně a smysluplně.

Do budoucna se dá počítat s rozšiřováním systému o další systémy, s kterými bude komunikovat. Proto je třeba dbát při návrhu systému na budoucí rozšiřitelnost a univerzálnost.

Návrh reportingového systému bude probíhat na základě požadavků zadavatele, které budou průběžně konzultovány. To umožní průběžně ověřovat, zda se vývoj přibližuje cílovému řešení a případně zapracovat změny, pokud se v průběhu změní představa zadavatele o funkcionalitě.

2 Současný stav

V současnosti tvorba reportů spočívá v třech krocích:

- přihlásit se do systémů (převážně Google AdWords a Seznam Sklik), najít si klienta, zobrazit si požadovaná data a exportovat si je do csv souboru
- odstranit ze souboru nepotřebná data, která systémy vždy automaticky vygenerují, zpracovat data a převést je do graficky přívětivější podoby
- data slovně popsat a vyhodnotit výsledky, které se následně e-mailem zasílají klientovi

Tento systém tvorby a zasílání reportů panuje ve většině digitálních agentur v ČR. Tento osobní a individuální přístup má své výhody. Po výslechu zaměstnanců, kteří reporty provádějí však vyšlo najevo, že je pro nich tato rutinní činnost začátkem každého měsíce, únavná a svým způsobem otravná. Marketingoví specialisté jsou kreativní lidé, kteří se často nechtějí těmito věcmi příliš zaobírat.

Tvorba jednoho takového reportu zabere cca 40-90 minut práce (záleží na rozsahu kampaní a velikosti klienta). Bráno v úvahu, že jeden zaměstnanec agentury má na starosti v průměru 20 klientů, pro které zpracovává reporty, stráví přibližně 20 hodin každý měsíc jen tvorbou reportů. Tento čas by však mohli využít jinak například optimalizací kampaně, testování, kreativní tvorbou pro nové klienty nebo rozšiřováním si znalostí.

Agentura PROFICIO Marketing s.r.o., pro kterou bude tento reportingový systém stavěn, má pod sebou v současnosti 4 specialisty věnujícím se PPC reklamám. Tvorba reportů pro agenturu představuje také značné finanční náklady. Lze to snadno vypočítat: máme-li 4 specialisty, každý stráví v průměru 20 hodin měsíčně nad reporty, dohromady to dělá 80 hodiny měsíčně. Každý z těchto specialistů je odměňován jinak (na základě zkušeností, druh pracovního poměru atd.) a agentura PROFICIO si nepřeje tyto detaily zveřejňovat. Přibližně však lze odhadnout tyto náklady na 20 000 Kč za měsíc. To jsou náklady ve výši, kdy má rozhodně smysl začít uvažovat nad automatizovaném řešením.

Je jasné, že informačním systémem se nepovede automatizovat vše a nedojde ke snížení nákladů o 100 %. Například slovní popis dat a vyhodnocení výsledků se bude stále muset řešit individuálně s osobním přístupem. V prvních dvou bodech, které jsou uvedeny v prvním odstavci této kapitoly, by však mělo dojít ke značné úspoře času.

2.1 Funkční a nefunkční požadavky na reportingový systém

V této kapitole budou popsány funkční a nefunkční požadavky na reportingový systém, které jsou zásadní k tomu, aby výsledkem práce bylo co nejpřesnější řešení problému.

2.1.1 Funkční požadavky reportingového systému

Na základě vyslechnutí zaměstnanců agentury, od reportingového systému se očekávají tyto funkční požadavky:

- přihlásit se do systému jako administrátor (zaměstnanec agentury)
- přidat spravované účty ze systémů Sklik nebo AdWords
- vybrat, které klienty chce zaměstnanec v rámci systému zpracovat
- u jednotlivých klientů vytvořit report, vybrat, které metriky chceme zobrazit a za jaké období
- možnost upravit report zpětně
- vygenerovat unikátní URL adresu pro každého klienta, kterou mu pak budou moci zaslat

2.1.2 Nefunkční požadavky reportingového systému

Co se týká pasivních vlastností, od systému jsou očekávány následující nefunkční požadavky:

- bezpečné uložení autentizačních údajů do systémů
- bezpečný přenos dat
- uživatelská přívětivost
- data uložena ve vlastní databázi
- nepřípustnost dat klientů pro veřejnost

2.2 Srovnání existujících systémů

V této podkapitole bude proveden průzkum mezi stávajícími možnostmi reportingových systému, které budou navzájem srovnány. Na základě toho vyjde najevo, proč vlastně má nebo nemá smysl vyvíjet pro agenturu nový nástroj, jestli by například nebylo lepší využít některou z dostupných služeb.

2.2.1 Kritéria hodnocení

Po prodiskutování se zástupci agentury Proficio byly stanoveny tyto kritéria, která jsou pro agenturu zajímavá jako požadavky na systém:

1. **Neomezený počet reportů** – je žádoucí, aby nebyl omezený počet reportů nebo uživatelů, kteří budou systém využívat. Jde o prosperující odvětví, agentura za poslední rok zdvojnásobila počet zaměstnanců, proto je dobré myslet dopředu.
2. **Rozšiřitelnost o další platformy** – reportovat výsledky ze Seznamu a Googlu je pro agenturu základ. Do budoucna ale přemýšlejí o rozšíření o další platformy, jako může být například Facebook, Twitter atd. Proto systém musí být připraven na toto rozšíření.
3. **Vizuální modifikace** – do této kategorie spadá například nastavení firemních barev, import loga, vlastní doména atd. Agentura si zakládá na své firemní identitě a chce podle toho před svými klienty vypadat.
4. **Volba vlastních sloupců** – systémy defaultně mnohdy nezobrazují právě ta data, která jsou důležitá a má je smysl ukazovat zákazníkům. Proto je důležité, aby systém umožnil navolit si metriky, které chceme v tabulce či grafu mít.
5. **Roztřídění kampaní podle typu** – systémy mají také tendence shlukovat všechny kampaně do jedné skupiny, kterou pak reportují jako celek. Takto to však není vhodné řešit, kampaně musí systém třídit podle typu, o roztřídění bude uvedeno více v implementační části textu.

Po provedeném průzkumu na internetu bylo vybráno 5 již existujících systémů, jejichž služby se nejvíce blíží stanoveným požadavkům. V následující tabulce je znázorněno nakolik systémy splňují kritéria.

Tab. 1 Srovnání existujících systémů

Systém	Neomezený počet reportů	Rozšiřitelnost o další platformy	Vizuální modifikace	Volba vlastních sloupců	Roztřídění kampaní podle typu	Cena za měsíc
Measureful	✘	✘	✘	✓	✘	2 400 Kč
Ninjacat	✘	✓	✓	✓	✓	20 000 Kč
Reportgarden	✘	✓	✘	✓	✓	3 200 Kč
Zoho	✓	✓	✓	✓	✘	3 400 Kč
Raventools	✓	✘	✓	✓	✓	6 000 Kč

2.2.2 Vyhodnocení

V tabulce č. 1 vidíme výsledky jednotlivých systémů po provedeném průzkumu či krátkém otestování. Mimo zvolená kritéria je zde také sloupec s cenami, který sice nebyl definován jako základní kritérium, přirozeně však tuto hodnotu musíme brát v úvahu, protože cena je dnes pro každého důležitým ukazatelem. Systémy využívají odlišných modelů cen, většinou jde o balíky služeb nebo škálování ceny s počtem reportů či uživatelů. V těchto případech byla zvolena ta cena, která odpovídá aktuálním kapacitním požadavkům. Toto řešení však není zcela ideální, protože, jak už bylo zmíněno na začátku této kapitoly, počítá se s rychlým nárůstem, co se týká počtu zaměstnanců a klientů.

Systém Measureful je ze zkoumaného vzorku nejlevnější. Zvoleným kritériím většině případů bohužel nevyhovuje. Nabízí sice další doplňkové služby jako automatické doručování e-mailů uživatelům, ale to v této fázi není pro naši agenturu tolik klíčové (Measureful, 2015).

Ninjacat, další ze vzorku, je platforma vynikající hlavně počtem platforem, s kterými lze jejich systém propojit. V tomto je při srovnání s ostatními systémy bezkonkurenčně nejlepší. Propojit jej lze s různými analytickými nástroji, přes sociální sítě až po vyhledávače. Zřejmě proto je také cena mnohonásobně vyšší oproti ostatním systémům, které byly vybrány. I v ostatních kritériích si Ninjacat stál dobře, až na omezený počet reportů. Takto vysoká cena je však pro agenturu nepřijatelná (Ninjacat, 2016).

Reportgarden již přichází s přijatelnější cenou, avšak je opět omezen co do počtu reportů a nemá moc možností vlastních vizuálních úprav reportů. Zajímavé na tomto systému je, že umožňuje spravovat projekty od počáteční fáze až po kompletaci (jednoduchá forma projektového řízení), což může být přínosem, avšak není to hlavní požadavek na systém(Reportgarden, 2016).

Asi nejvhodnějším kandidátem je systém Zoho, který nabízí přívětivý poměr cena/výkon. Systém nenabízí pouze dělení kampaní podle druhu, to je ale dáno tím, že se specializuje na více platforem, u kterých nemusí být dělení tak specifické jako u našeho Seznamu a Googlu. Systém je jinak velice bohatý na možnosti, nabízí například přidávání reportů na web (embed), což umožňuje volnost ve sdílení mezi klientem a agenturou. Systém sice nabízí neomezený počet uživatelů a reportů, je zde však omezení na počet záznamů(řádků) v databázi (Zoho, 2016).

Raventools přináší také dobré možnosti s mírným nárůstem ceny. Jediným nedostatkem je jeho užší specializace na konkrétní systémy a nemožnost připojení dalších reklamních platforem. Systém je zajímavý tím, že má vlastní roboty, které prohlížejí a auditují stránky klientů a poskytují další informace formou reportů (Raventools, 2016).

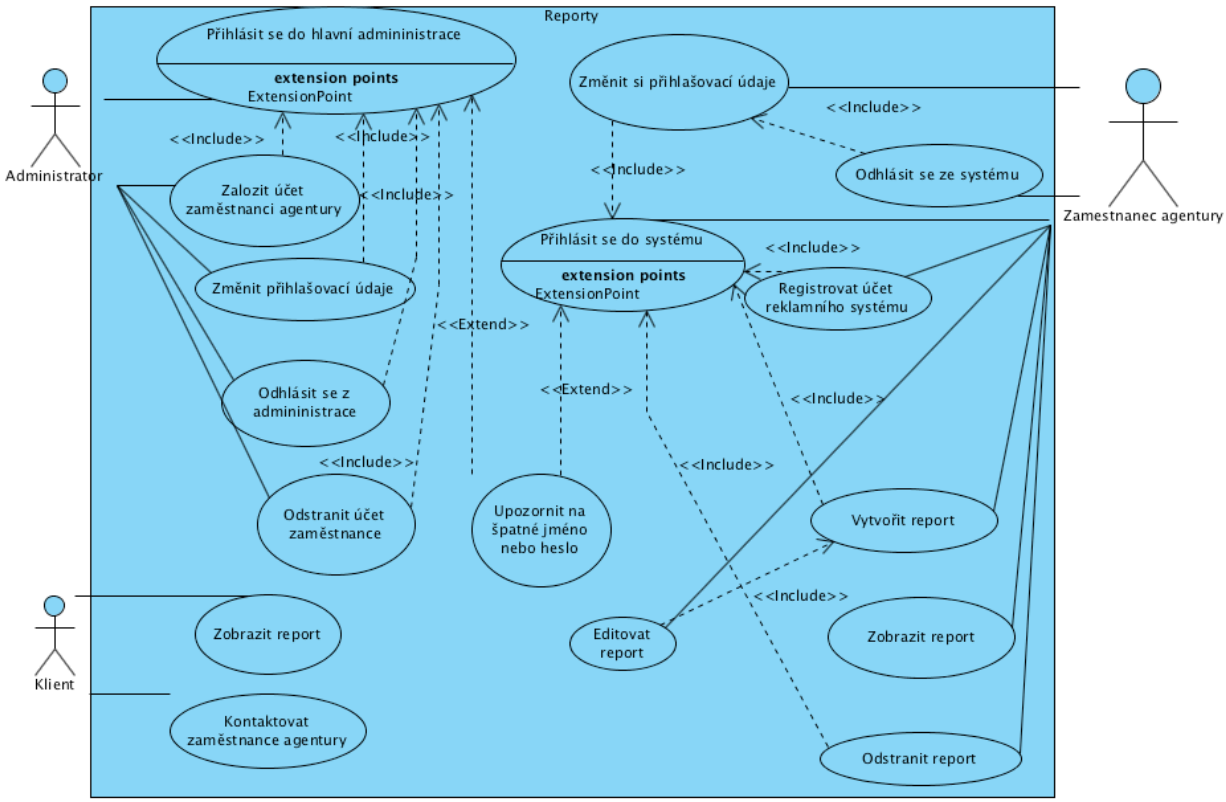
Na závěr této kapitoly shrneme výsledky. Z nasbíraných informací o systémech se jako nejlepší kandidát jeví systém Zoho. Disponuje skoro všemi požadovanými vlastnostmi až na třídění kampaní. Problémem však zůstává cena. V tomto případě je cena 3 400 Kč za měsíc při vytížení v aktuálním čase. Systém by pro firmu představoval roční náklady 40 800 Kč, což jsou peníze, které může využít jinak, případně je investovat do vlastního řešení, které bude naprosto na míru potřebám. Také je potřeba pokládat si otázku, co když za rok bude potřeba mnohem větší kapacita (systém je omezen na počet řádků v databázi)? Další balík služeb v platebním modelu systému Zoho s názvem ENTERPRISE stojí už 12 000 Kč za měsíc. To jsou pro firmu Proficio nepředstavitelné roční náklady ve výši 144 000 Kč ročně.

3 Vlastní řešení

V minulé kapitole byly identifikovány požadavky na reportingový systém a byl proveden průzkum mezi existujícími systémy. Požadavky agentury byly natolik specifické, že žádný z porovnávaných systémů jim v plné míře nevyhověl. Navíc sebou nesly značné finanční náklady formou měsíčního tarifu. Proto bude nejlepším postupem vytvořit nové řešení, které bude agentuře Proficio stavěné na míru, bude do budoucna rozšiřitelné a nepřinese takovou finanční zátěž. V této kapitole bude představen návrh vlastního řešení.

3.1 Use case

Předtím, než se pustíme do návrhu uživatelského rozhraní a implementace v kódu, je potřeba si ujasnit, kdo bude systém pro tvorbu reportů používat a jakým způsobem. V této subkapitole bude uveden use case diagram, ve kterém budou tyto věci znázorněny. Use case diagram si lze prohlédnout na následujícím obrázku:



Obr. 1 Use case diagram systému pro tvorbu reportů

V diagramu je vidět, že interagovat se systémem budou 3 druhy uživatelů:

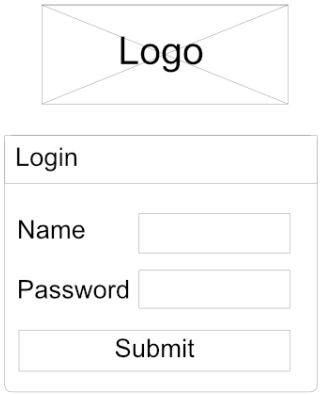
- **administrátor** – bude mít přístup do hlavní administrace v Djangu a tím pádem bude moci řídit celou aplikaci, jeho úkolem bude primárně registrovat účty zaměstnanců a dohlížet nad chodem aplikace
- **zaměstnanec agentury** – bude moci se přihlásit do aplikace (ne však do hlavní administrace Djanga), zde bude jeho úkolem registrovat si reklamní účty klientů a poté vytvářet reporty
- **klient** – dostane unikátní URL adresu ke svému reportu, který si bude moci prohlédnout a uvidí zde i kontaktní údaje zaměstnance, který report vytvořil

V podstatě veškeré činnosti, které se budou dát uskutečnit v systému, budou vyžadovat přihlášení. V aplikaci se bude pracovat s citlivými daty klientů a je nežádoucí, aby se k nim dostal někdo bez přihlášení. Jediné zobrazení reportu bude zatím bez přihlášení, jednotlivé reporty nebudou indexované vyhledávačích a URL bude mít každý report unikátní. Pokud by toto řešení časem nevyhovovalo, nebude implementačně složité vyžadovat přihlášení i u klientů.

3.2 Návrh rozhraní administrace systému

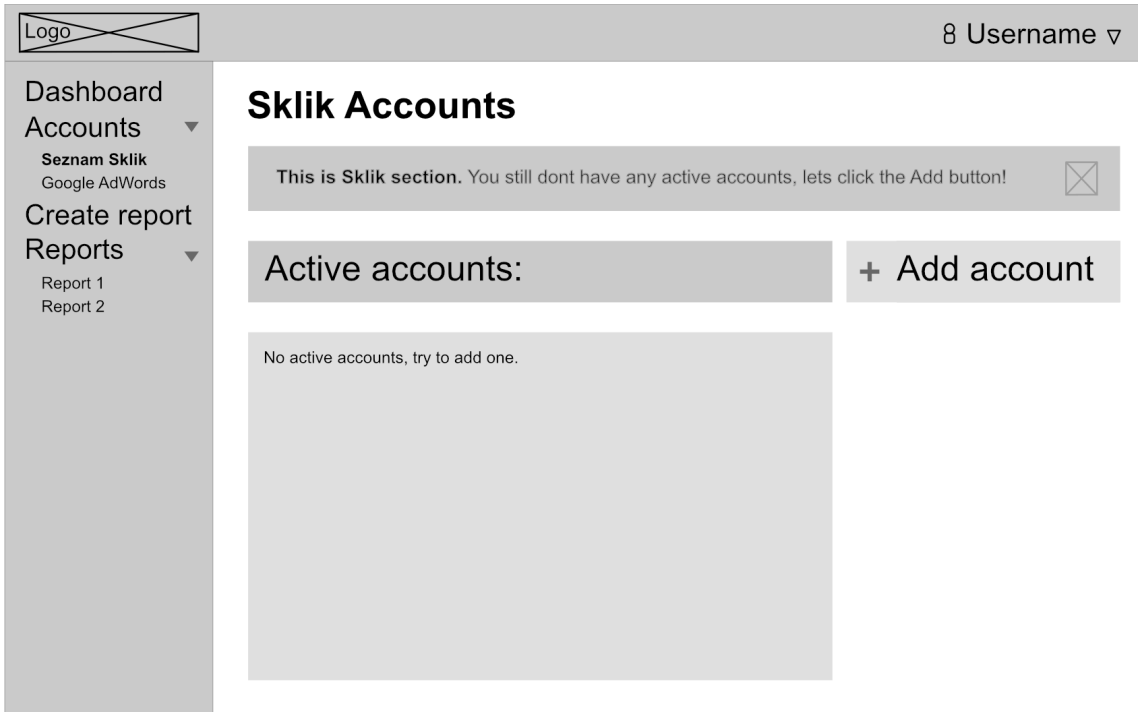
Systém budou používat lidé bez hlubšího technického vzdělání. Proto je žádoucí navrhnout takové rozhraní, které bude jednoduché a přehledné na první pohled. Proto byl vytvořen wireframe aplikace, který bude před samotnou implementací zkonzultován s firemním UX designérem.

Podstatnou částí systému je přihlašování. Aplikace bude uchovávat citlivá data klientů a nakládat s přístupovými údaji do jednotlivých systémů. O samotném řešení autentizace si více řekneme v implementační části. Nyní si ukážeme drátový návrh (wireframe) přihlašovací stránky:



Obr. 2 Wireframe přihlašovací stránky

Na přihlašovací stránce bude pouze jednoduchý formulář pro přihlášení. Registrace nebude volně umožněna, bude ji provádět vždy administrátor systému.



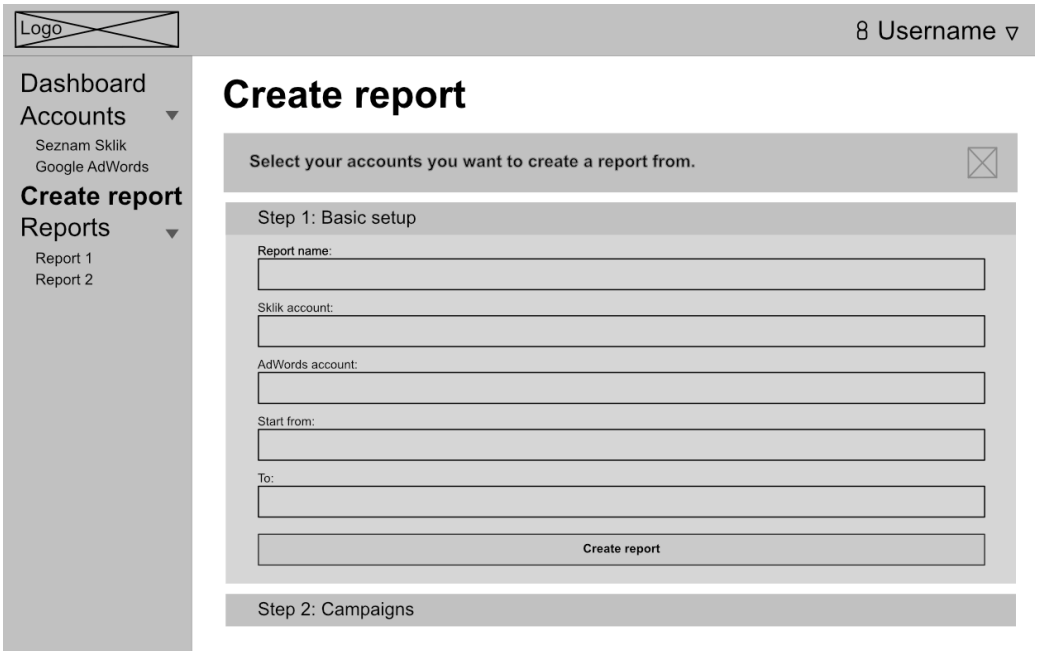
Obr. 3 Wireframe stránky s přihlášenými účty reklamních systémů

Na obrázku č. 3 je již vidět administrační část aplikace, konkrétně stránka s přehledem přihlášených účtů inzertního systému Sklik. V levé části pod logem je blok s navigací, položky menu jsou smysluplně seskládány do hierarchie tak, aby uživatele navedli co nejplynuleji aplikací od úvodní stránky po přihlášení (Dashboard) až po jednotlivé vytvořené reporty.

Pod rozevírací položkou Accounts se nachází podúčty z jednotlivých systémů, zde se dají prohlížet a registrovat marketingové účty Skliku a AdWords. Při kliknutí na tlačítko „Add account“ se zobrazí registrační formulář, který vyzve uživatele k zadání přihlašovacích údajů, následně je přesměrován zpět do seznamu účtů, kde již uvidí své registrované účty.

Na následujícím obrázku č. 4 je návrh stránky s formulářem pro vytvoření reportu. Skládá se z 5 vstupních polí a potvrzujícího tlačítka. Pole pro název reportu je čistě textové, pole pro Sklik nebo AdWords účet budou výběrová, po kliknutí vyjede box se všemi registrovanými účty daného uživatele.

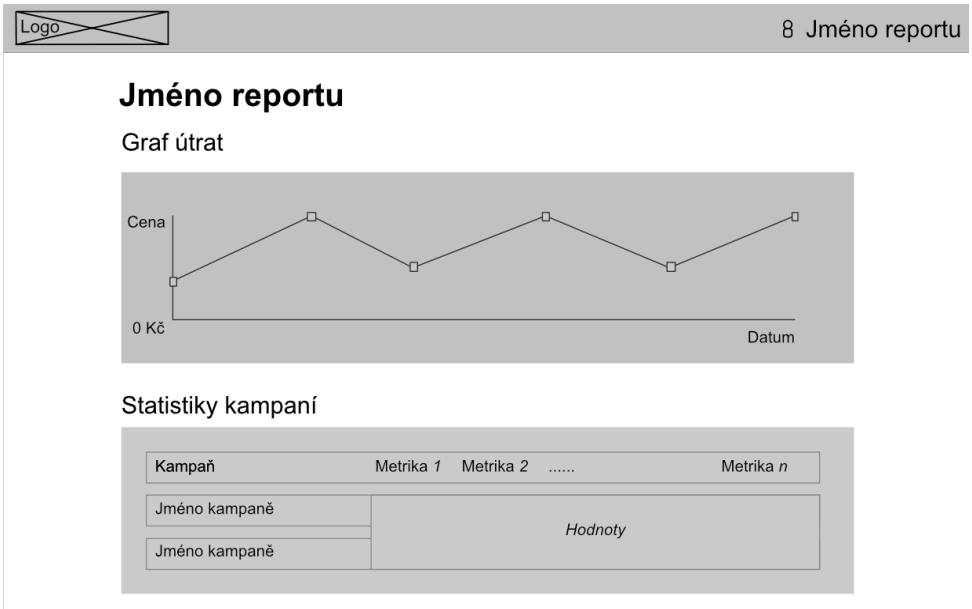
Jak bylo vidět na obrázcích, administrativní část je v angličtině. Je tomu tak proto, že uživatelé jsou na angličtinu zvyklí z reklamních systémů a je pro nich tento jazyk přirozený.



Obr. 4 Wireframe stránky pro přidání nového reportu

3.3 Návrh rozhraní vygenerovaného reportu

Rozhraní s vygenerovaným reportem bude mít poněkud jiný charakter než administrativní rozhraní, je totiž konstruováno pro klienty agentury. Zobrazení tedy musí být ještě jednodušší a přímočařejší než u administrace, navíc bude kompletně v češtině.



Obr. 5 Wireframe stránky s vyobrazeným reportem

Na obrázku č. 5 vidíme wireframe stránky s vygenerovaným reportem. Na první pohled je vidět absence levého panelu s navigací. Klient totiž ve svém reportu neuvidí žádné menu, zobrazí se mu pouze stránka s reportem a nebude moci se proklikat hlouběji do administrativní části aplikace.

Stránka obsahuje dvě hlavní komponenty:

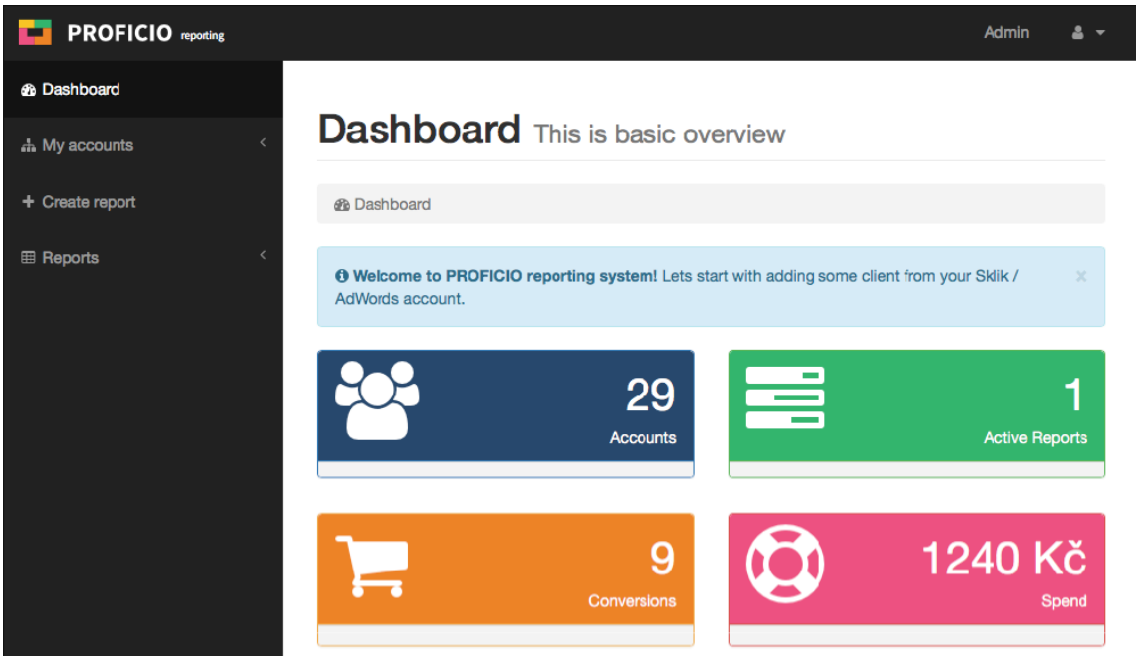
- graf útrat
- statistiky kampaní

V grafu útrat bude zobrazen vývoj útrat ve zvoleném období. Statistiky kampaní budou zobrazeny formou tabulky, kdy ke každé kampani budou vypsány hodnoty spadající pod příslušené metriky.

3.4 Grafický návrh

Grafický návrh sice není podstatou této práce. Zmíníme ale ve stručnosti pár základních prvků, které byly při grafickém návrhu využity.

Na obrázku č. 6 je k vidění grafická podoba aplikace. V levém horním rohu je logo agentury PROFICIO doplněno o slovo reporting.



Obr. 6 Grafický návrh reportingového systému

Barvy, které byly pro návrh využity byly čerpány z agenturního vizuálu. Jedná se především o barvy v následující tabulce.

Tab. 2 Barvy použité pro grafický návrh systému

Barva	Hex kód barvy
San Juan	#2F4C71
Medium Sea Green	#43B970
Cadmium Orange	#EA812F
Dark Pink	#E94B82

Mimo tyto barvy byla využita bílá barva na pozadí stránky a černá na pozadí menu a horní lišty.

Jako písmo bylo zvoleno Helvetica Neue, kterého se využívá ve všech textech i nadpisech systému. Grafické doplňky jako jsou ikony a podobně byly čerpány z Bootstrap šablony a jsou volně ke stažení spolu s celým Bootstrap setem (Bootstrap, 2015).

Využití Bootstrapu a pár dalších nástrojů, které se podílely na vzhledu aplikace z uživatelské strany, je ještě stručně popsáno v kapitole 4.11 Frontend v kontextu implementace.

3.5 Předpokládané finanční dopady vlastního návrhu

V této subkapitole bude odhadnut finanční přínos navrženého řešení. Jak bylo odhaleno při analýze současného stavu, tvorba reportů sebou nese každý měsíc opakující se náklady. Tyto náklady jsou reprezentovány časem, který zaměstnanci agentury stráví tvorbou reportů. Jednak jde o finanční náklady, jejichž výši zjistíme vynásobením počtu hodin a hodinovou sazbou zaměstnance, ale také o náklady obětované příležitosti, tedy ušlé příležitosti, kdy by mohli zaměstnanci čas využít lépe.

Při současném stavu, kdy agentura zaměstnává 4 zaměstnance věnující se PPC reklamám a každý z nich spravuje v průměru 20 účtů klientů, sebou tvorba reportů nese náklady zhruba 20 000 Kč za měsíc (agentura si nepřeje zveřejňovat hodinové sazby jednotlivých zaměstnanců, proto zde není odhalen celý výpočet). Navržený systém pro tvorbu reportů osvobozuje zaměstnance od ručního stahování dat v tabulkách, očištění od nepodstatných dat a převod do agenturního vizuálu. Tato část tvoří cca 90 % časové náročnosti celého procesu. Zbylých 10 % tvoří už jenom slovní ohodnocení výsledků, u kterého je osobní dohled nezbytný. **Úspory na mzdových nákladech tedy činní 18 000 Kč za měsíc**.

Vlastní reportingový systém přirozeně přinese náklady na údržbu, aktualizace, opravy chyb a drobné úpravy. Zároveň aplikace poběží na cloudu, kde jsou služby také zpoplatněny. Samotná aplikace na cloudu s vyhrazenými 256 MB operační paměti, počtem http requestů menším než 300 za sekundu a počtem uživatelů menším než 100 je zdarma. Jelikož aplikaci bude využívat agentura pouze interně, s tímto plánem by si měli na období dvou let snadno vystačit. MySQL databáze je však ve verzi zdarma velmi omezená z hlediska kapacity (5 MB), počtu připojení (4) a rychlosti. Tento plán je dobrý spíše na vývoj a testování než na ostrý provoz. Pro ostrý provoz vyhovuje první placený tarif s názvem Boost DB za cenu 250 Kč/měsíc, který nabízí větší rychlost, kapacitu a počet připojení (IBM Bluemix, 2016).

V následující tabulce je uveden očekávaný přehled měsíčních nákladů na aplikaci.

Tab. 3 Předpokládané měsíční náklady na provoz systému

Náklad	**Cena za měsíc**
IBM Bluemix	0 Kč
Databáze (plán Boost DB)	250 Kč
Správa systému (3h za měsíc)	900 Kč
Celkem	**1 150 Kč**

Částka za správu systému není striktně vymezena. Nemusí jí být využito každý měsíc. Jde pouze o odhad na případné opravy chyb, vizuálních nedostatků, na které se přijde v průběhu užívání. Zároveň může být částka využita na postupný rozvoj systému.

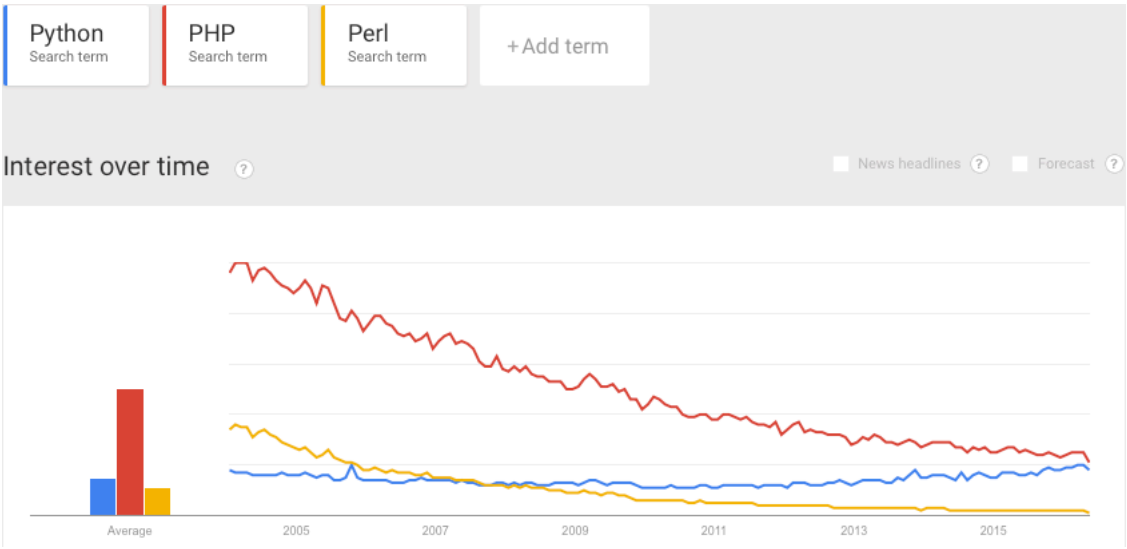
Celkem tedy systém přináší měsíční úsporu 18 000 Kč na mzdových nákladech a náklady na jeho provoz 1 150 Kč. Po odečtení nákladů na provoz od úspor na mzdových nákladech se dostáváme na částku 16 850 Kč, což představuje finální očekávané měsíční úspory na nákladech po nasazení systému do ostrého provozu. S náklady, které sebou nese systém by měly zůstat fixní minimálně na období 2 let, poté je těžké odhadnout, jak na tom bude agentura s kapacitou, zda nabyde nějaké nové požadavky na systém, které budou vyžadovat komplexnější zásah do systému atd. Potenciální měsíční úspora na mzdových nákladech se však zvýší s každým novým zaměstnancem.

4 Implementace

V této kapitole budou popsány technologie, nástroje a principy, kterých bude využito k realizaci reportingového systému včetně ukázek praktické implementace (výtažky z kódu apod.).

4.1 Programovací jazyk Python

Python je programovací jazyk, který je známý svými specifickými syntaktickými pravidly pro psaní kódu. Poslední dobou se více a více dostává do popředí díky projektům fungujícím ve světovém měřítku například Instagram nebo český Skypicker. Tyto projekty využívají denně desetitisíce lidí a vsadili právě na tento programovací jazyk.



Obr. 7 Statistika hledanosti vybraných programovacích jazyků od roku 2014 do 2016 (Google Trends, 2016).

Na obrázku č. 7 je znázorněno srovnání hledanosti vybraných programovacích jazyků, které se obvykle využívají pro tvorbu webových aplikací, za období od roku 2004 do roku 2016. Samozřejmě, zdroj Google Trends nepodává pohled, kolik reálně programátorů aktivně využívá právě tento programovací jazyk. Trend má však určitou vypovídací hodnotu o popularitě programovacího jazyka a z grafu lze jednoznačně vyčíst klesající trend u PHP i Perlu, kdežto Python v trendu roste (Google Trends, 2016).

Programovací jazyk samozřejmě nebyl vybrán pouze na základě trendů. Filozofie Pythonu je postavena na základních myšlenkách:

- pěkné je lepší než škaredé
- explicitní je lepší než implicitní
- jednoduché je lepší než komplexní
- komplexní je lepší než komplikované
- čitelnost hraje roli

Pro tyto zásady Python podporuje jednoduchou a čitelnou syntaxi (na rozdíl například od Perlu). Odmítá komplikované a komplexní zabudování veškeré funkcionality do jádra, spíše podporuje rozšiřitelnost (Peters, 2004).

Python je zdarma dostupný a nabízí verze pro nejpoužívanější operační systémy jako je Linux, OS X nebo Windows. Python se dá použít jako **skriptovací jazyk pro webové aplikace** přes modul mod_wsgi a Apache web server. Díky tomu může být využit pro účely této práce (Sweigart, 2014, strana 20).

4.2 Webový Framework Django

Django je skvělý open source framework pro tvorbu webových aplikací založený na Pythonu. Zaměřuje se na zjednodušení procesu vytváření databázově řízených webových aplikací (George, 2015, strana 15).

Potenciál Djanga spočívá v tom, že se v něm dají velmi rychle budovat dynamické stránky za relativně krátkou dobu. Automatizuje spoustu rutinních procesů, které se skrývají za tvorbou webových stránek. Jeho vývoj začal v roce 2003, vydaný jako open source je od roku 2005 pod BSD licencí (Lennon, 2009).

Autoři Djanga prezentují nejsilnější stránky frameworku prostřednictvím následujících vlastností:

- neobvykle rychlý
- se vším všudy
- spolehlivě bezpečný
- mimořádně škálovatelný
- univerzální

Stejně tak jako programovací jazyk Python, ve kterém je framework napsaný, i samotné Django našlo oblibu u světových projektů jako je např. Disqus, Pinterest, Instagram nebo Mozilla (Djangoproject, 2016).

Dobré zázemí také přináší dokumentace frameworku, která je přehledná, nabízí ukázky kódu na reálných příkladech a dá se přecházet napřič jednotlivými verzi Djanga. Framework je neustále vyvíjen a přicházejí nové a nové verze, proto je dobré mít přehled o změnách napříč jeho distribucemi a mít tu možnost nahlížet do dokumentace staršího vydání, než je to nejaktuálnější (Djangoproject, 2016).

4.3 Cloudový server IBM Bluemix

Bluemix je nejnovější cloudové řešení od společnosti IBM. Umožňuje organizacím a vývojářům rychle a jednoduše vytvořit, zavést a řídit aplikace na cloudu. Architektura Bluemix je postavena na tzv. Cloud Foundry, což je open source platforma poskytována jako služba (PaaS). Díky tomu můžou společnosti integrovat jejich aplikace s cloudem aniž by věděli, jak je nainstalovat nebo nakonfigurovat (Tomala-Reyes, 2015).

Cloud Foundry umožňuje řídit aplikace na cloudu pomocí příkazové řádky naprosto jednoduše se sadou pár příkazů. Na straně cloudu figuruje tzv router, který směřuje požadavky na příslušnou komponentu, což je obvykle tzv. cloud controller nebo běžící aplikace na daném uzlu. K autentizaci se využívá OAuth2 server, který spolupracuje s login serverem a provádí tak správu identit. Cloud controller je zodpovědný za životní cyklus aplikace. Dále zde běží tzv Droplet Execution Agent (DEA), který řídí instance jednotlivých aplikací a vysílá informace o jejich aktuálním stavu. Když například nahrajeme na server aplikaci, je to právě DEA, který nás uvědomí o tom, zda se aplikace podařila nahodit nebo došlo k určitému problému. Message Bus potom slouží k distribuci zpráv mezi jednotlivými komponenty. Všechno je ukládáno do logů a statistik pomocí metrics collectoru, který sbírá data z jednotlivých komponent a poskytuje operátoru informace k monitorování instancí na Cloud Foundry (CloudFoundry, 2015).

Bluemix pro účely této práce přináší nejen potenciál v jeho vlastnostech jako je rychlost a relativní jednoduchost. Veliký přínos je v jeho univerzálnosti, tedy možnost spustit si jakoukoliv službu z rozmanitého výběru na daném programovacím jazyku. Najít kvalitní hostovací server pro python aplikaci s omezeným rozpočtem pro účely této práce není zcela jednoduché. Hodně takových serverů nabízí podporu např. pouze starších verzí Pythonu nebo Djanga. Tento problém řeší Bluemix, na kterém se po určitém vynaloženém úsilí podaří zprovoznit nejnovější verzi Djanga (vyžaduje to konfiguraci, o které bude v textu více později).

4.4 Spuštění Django na lokálním serveru

Pro spuštění aplikace napsané v Djangu na lokálním serveru je potřeba nejprve nainstalovat nástroj pip. Jde o manažera modulů napsaných v Pythonu. Pomocí pipu nainstalujeme Django a veškeré další moduly, které budou pro tvorbu reportingového systému využity (Pip, 2014).

Modely využité v této práci jsou vypsány v souboru requirements.txt nacházejícím se v hlavním adresáři projektu. Django nainstalujeme v terminálu příkazem *pip install django.* Stejným způsobem nainstalujeme i další moduly. Pokud nemáme vytvořen Django projekt, založíme nový projekt v terminálu příkazem

```
django-admin startproject reportingSystem
```

Tímto příkazem Django vytvoří projektový adresář reportingSystem. Pojem projektový adresář nebo hlavní adresář projektu se v textu dále vyskytuje, je tím myšlen právě tento adresář, který vytvořilo Django. Django navíc rozlišuje pojem projekt a aplikace. Doposud byl založen pouze projekt. Každý projekt v sobě může obsahovat několik aplikací. Tyto aplikace může vývojář sám napsat anebo použít aplikace od někoho jiného. Pro založení aplikace uvnitř projektu použijeme příkaz

```
./manage.py startapp reportsApp
```

Po tomto příkazu se nám v projektovém adresáři objeví nový adresář s názvem reportsApp. Zde se nacházejí všechny soubory dané aplikace. Nyní máme založený čistý projekt a aplikaci v Djangu, můžeme zavolat v terminálu příkaz

```
./manage.py runserver
```

Pokud proběhlo vše v pořádku, na standardní výstup se vypíše:

```
Performing system checks...

System check identified no issues (0 silenced).
May 20, 2016 - 21:05:54
Django version 1.9.5, using settings 'reportingSystem.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

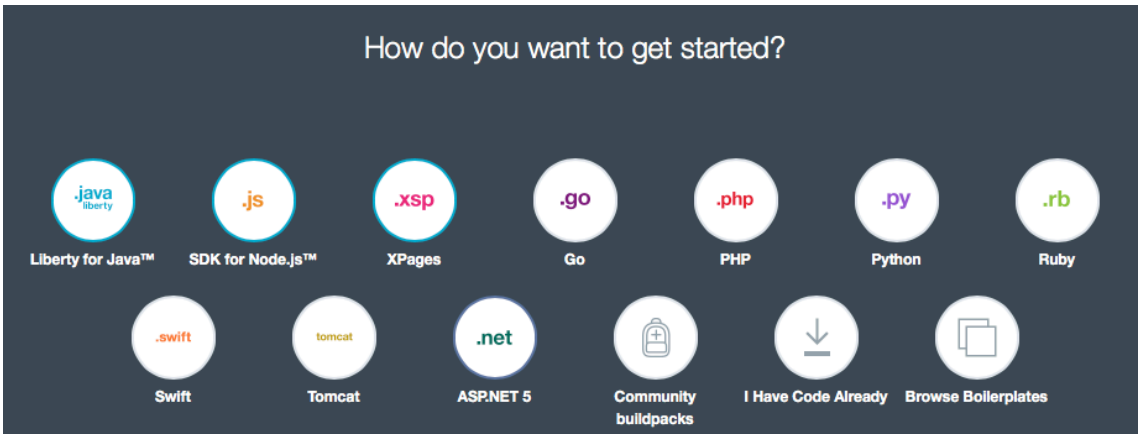
Nyní můžeme zkusit do prohlížeče zadat adresu http://127.0.0.1:8000/, měla by naběhnout uvítací stránka Django a může se začít s vývojem. Samozřejmě je ještě třeba nastavit databázi v souboru settings.py umístěném v adresáři projektu.

4.5 Implementace Djanga na Bluemix

Podstatnou částí práce bylo nastudování principů, jak funguje cloudový server Bluemix a jakým způsobem zde spustit vlastní aplikaci. Jak je zřejmé z předchozí kapitoly cloudový server má jisté odlišnosti od klasického hostingového serveru. Bluemix nabízí ve svém administrativním rozhraní velké množství služeb, které můžeme na pár kliknutí zprovoznit a začít programovat. Tyto služby se dělí do 5 základních kategorií:

- Cloud Foundry aplikace
- kontejnery
- virtuální servery
- data a analytika
- služby a API

V každé z těchto kategorií se pak nachází ty hotové balíčky, které se dají na pár kliků spustit. Můžeme si tak vybrat mezi databázovými systémy jako je PostgreSQL a další nebo spustit mobilní či webovou aplikaci na vybraném frameworku.



Obr. 8 Volba programovacích jazyků pro webovou aplikaci v Bluemixu

Na obrázku č. 8 je vidět seznam programovacích jazyků, z kterých si můžeme vybrat při vytváření webové aplikace, což je prvním krokem v tomto procesu. Pod volbou „Browse Boilerplates“ se pak nacházejí, některé webové frameworky. Bohužel se mezi nimi nenachází framework Django, na kterém poběží aplikace v této práci. Proto bylo potřeba zvolit si jeden z tzv. komunitních buildpacků. Buildpack je balíček, který zajišťuje podporu běhu aplikace na daném programovacím jazyku. Jsou v něm definovány závislosti (dependencies), které je třeba nainstalovat pro správnou komunikaci s připojenými službami (Cloud Foundry Docs, 2016).

Abychom mohli nahrát na Bluemix projekt s aplikací v Djangu, musíme přidat pár konfiguračních souborů, upravit konfigurační soubory a nainstalovat moduly, které umožní běh aplikace na cloudu. Nezbytné moduly jsou následující:

- dj-database-url
- gunicorn
- mysqlclient
- dj-static

Názvy těchto modulů musejí být napsány v souboru requirements.txt, který je umístěn v hlavním adresáři projektu (Ferreira, 2014).

Modul dj-database-url umožní načíst environmentální proměnnou ze systému. Umožní nám to mít vlastní databázi na lokálním serveru a cloudu a zároveň nebudeme muset při každém updatu měnit konfigurační soubory, protože Django si získá URL databáze automaticky z environmentální proměnné (Python, 2015). V konfiguračním souboru settings.py, kde se kromě veškerého nastavení aplikace nastavuje také databáze to pak vypadá následovně:

```
DATABASES['default'] = dj_database_url.config()
```

Nenastavujeme tedy klasicky hostname, uživatele, heslo a port, ale odkážeme Django na environmentální proměnnou, kterou sami definujeme. V operačním systému MAC OS X se například definuje environmentální proměnná ve skrytém souboru .bash_profile nacházejícím se v domovském adresáři uživatele.

Modul gunicorn je Python WSGI server pro UNIX. Díky tomuto modulu může server vystát HTTP požadavkům a odesílat na stranu klienta obsah naší aplikace. Modul je kompatibilní hned s několika webovými frameworky, je jednoduše implementovatelný a rychlý (Gunicorn, 2015). V souboru run.sh, který se spouští po uploadu aplikace na cloud zavoláme příkaz pro spuštění serveru, který se odkazuje na .wsgi konfigurační soubor v adresáři aplikace:

```
gunicorn reportingSystem.wsgi --workers=3
```

Modul mysqlclient slouží ke komunikaci Djanga s databázovým systémem MySQL, pokud bychom používali jiný databázový systém, museli bychom zvolit příslušný modul. V konfiguračním souboru settings.py je pak podle toho definován databázový systém následovně:

```
'ENGINE': 'django.db.backends.mysql'
```

Na základě tohoto řádku kódu Django ví, s jakým databázovým systémem aplikace komunikuje a zařídí se podle toho při migracích databáze, vytváření tabulek podle objektů a další (Digital Ocean, 2015).

Modul dj-static Djangu slouží k poskytování statických souborů. Django samo o sobě k tomu není stavěné a původně statické soubory byly uloženy někde na serveru nebo v CDN. Pomocí tohoto modulu však můžeme poskytovat statické soubory přímo v aplikaci. Statickými soubory jsou myšleny například soubory se styly, skripty nebo obrázky, což se nám všechno bude v rámci projektu hodit (Ultimate Django, 2015).

Abychom mohli servírovat statický obsah, musíme uvést cestu k adresáři s tímto obsahem v konfiguračním souboru settings.py:

```
STATIC_ROOT = 'staticfiles'
STATIC_URL = '/static/'

STATICFILES_DIRS = (
   os.path.join(BASE_DIR, 'static_in_pro'),
)
```

Tímto byly Djangu poskytnuty informace o tom, že statické soubory projektu se nachází v adresáři *static_in_pro* s tím, že chceme, aby Django po příkazu *collectstatic* seskupil statické soubory v adresáři s názvem *staticfiles.* Cesta v URL adrese všech statických souborů pak bude začínat řetězcem */static/* například: http://example.com/static/logo.png.

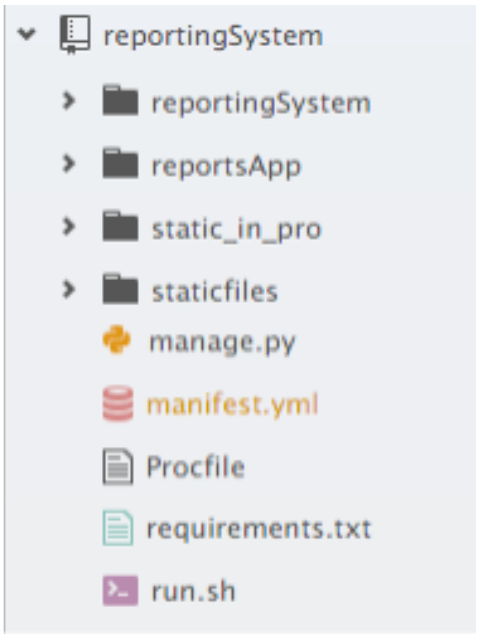
Mimo tyto moduly samozřejmě musíme v souboru requirements.txt uvést také samotné Django, které není implicitně na cloudu nainstalováno. Zde je třeba dát si pozor na verzi Djanga, s kterou chceme pracovat. Ve starších verzích Djanga neexistují migrace databází a jsou zde další mírné odlišnosti, které mohou spuštění aplikace na cloudu s uvedenou konfigurací zkomplikovat. Pro náš reportingový systém bude využita nejaktuálnější verze Djanga 1.9.4. Budeme tak moci využívat nejnovějších funkcionalit a čerpat z nejaktuálnější verze dokumentace.

Protože na cloud nenahráváme jeden z přednastavených frameworků, ale framework vlastní, kromě konfiguračního souboru Djanga musíme uvést v projektovém adresáři také soubor manifest.yml. Tento soubor je klíčový pro Bluemix, identifikuje v něm aplikaci a služby, které jsou k ní připojeny. V našem případě půjde o aplikaci s komunitním python buildpackem a s databází jako připojenou službou. Mimo to v tomto souboru také můžeme uvést příkaz, který se vykoná po uploadu aplikace na cloud v našem případě je to příkaz spouštěcí soubor s shell skriptem run.sh. V poslední řadě definujeme, kolik chceme pro naši aplikaci vyhra-

dit prostoru v operační paměti, pro naši aplikaci bude stačit 256 MB. Soubor manifest.yaml pro naši aplikaci bude vypadat následovně:

```
applications:
- name: reportingSystem
  memory: 256M
  command: bash ./run.sh
  buildpack: python_buildpack
  path: .
declared-services:
  mypostgres:
    label: ClearDB MySQL Database
    plan: 100
services:
  - reportingSystemDB
```

Nyní jsme podnikli veškeré kroky v projektové složce naší aplikace, která je nyní připravena k uploadu na cloudové úložiště (repository). Projektová složka se všemi výše uvedenými konfiguračními soubory vypadá následovně:



Obr. 9 Struktura souboru v projektovém adresáři aplikace

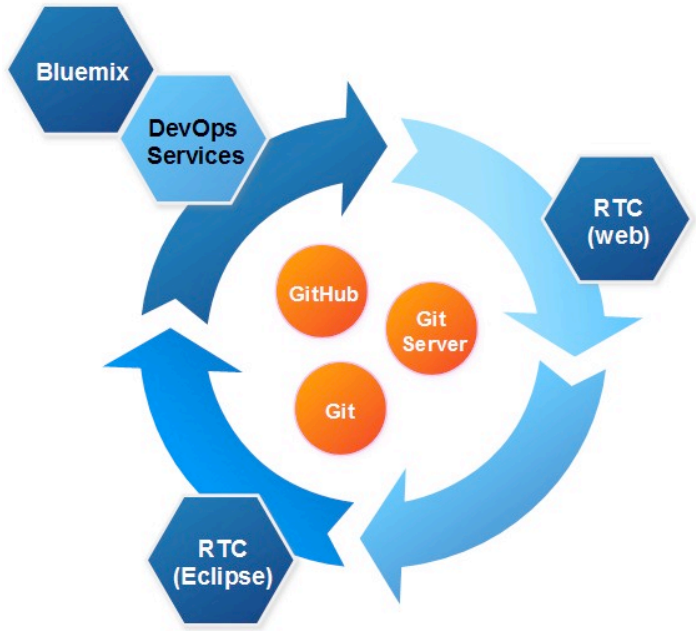
Možnosti, jakým způsobem můžeme nahrát projekt na cloud budou popsány v následující kapitole.

4.6 Nasazení aplikace na Bluemix

Existují tři možnosti, jak nasadit aplikaci na Bluemix:

- Cloud Foundry
- plugin do Eclipse
- služba IBM DEVOPS

Možnosti jsou dobře graficky zpracovány i na následujícím diagramu:



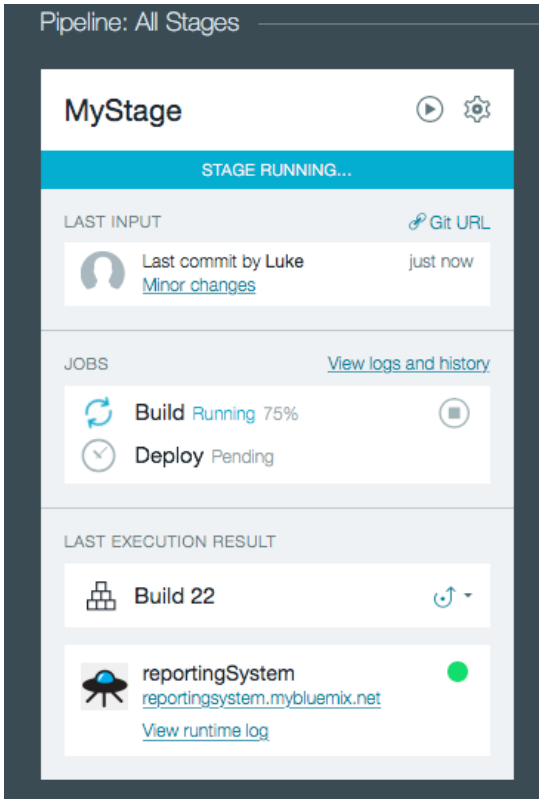
Obr. 10 Způsoby nasazení aplikace na Bluemix (Katzen, 2015)

První možnost je asi nejjednodušší, je potřeba si nainstalovat do příkazové řádky Cloud Foundry interface. Potom už jednoduše pomocí příkazu na přihlášení zvolíme workspace a pomocí dalšího příkazu *cf push <app_name>* zahájíme nahrávání aplikace na cloud. Po nahrátí aplikace se automaticky spustí její instance, pokud vše proběhlo bez chyb. Sledovat průběh nahrávání můžeme v terminálu, pokud však nastane problém a chceme ho hlouběji analyzovat, musíme si prohlédnout logy, ke kterým se dostaneme příkazem *cf logs <app_name> -recent.* Pomocí parametru recent říkáme, že chceme jen logy z nedávného pokusu o spuštění instance (Baxter, 2014).

Další možnost je využít pluginu do známého integrovaného vývojového prostředí Eclipse, který je volně ke stažení. Tato možnost je nejvíce užitečná pro java vývojáře, pro které je velmi pohodlné odesílat aplikace na cloud, aniž by opustili své oblíbené vývojové prostředí (Katzen, 2015).

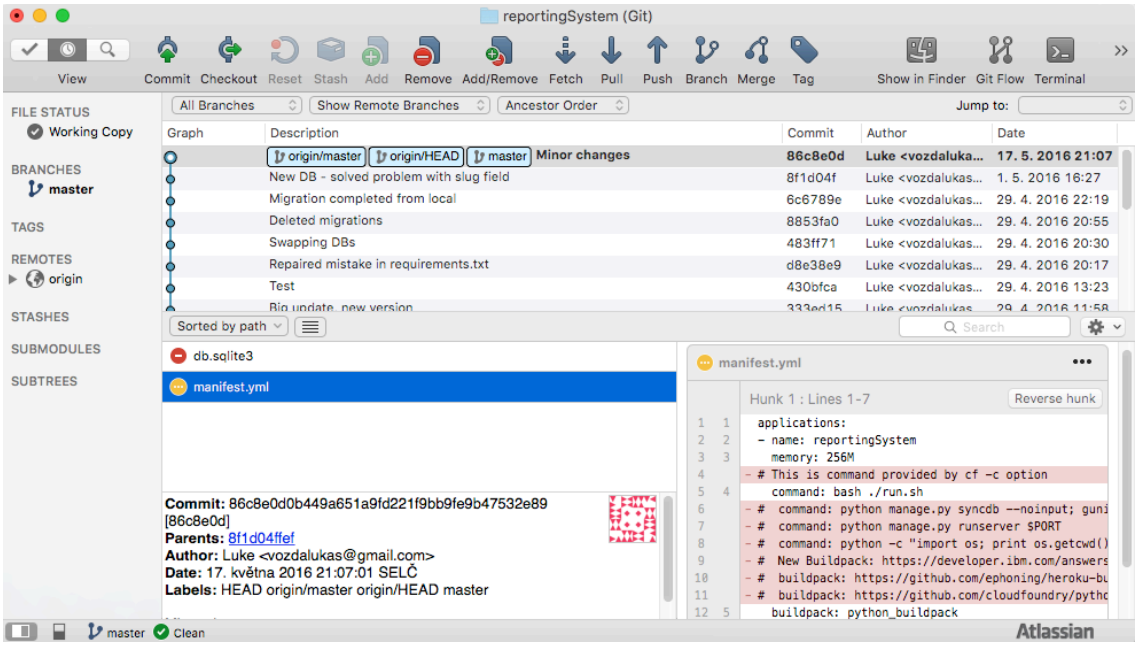
Třetí možnost je nejvíce komplikovaná na nastavení, přináší však spoustu benefitů a proto jí bude využito pro tvorbu reportingového systému v této práci. Pomocí služby DevOps od IBM je totiž možné řídit vše přes git. Pomocí klienta na počítači jako je například aplikace SourceTree pak jednoduše pomocí funkcionality commit a push nahrajeme aktuální verzi na cloud, můžeme projekt nasdílet dalším osobám a pracovat v týmu, vytvářet nové větve a mít tak transparentní pohled na průběh vývoje. Toto jsou praktiky, kterých využívají moderní společnosti a vývojářské týmy (IBM Bluemix DevOps Services, 2016).

Aby bylo možné využít tyto praktiky je třeba na Bluemixu vytvořit tzv. git repository pro konkrétní aplikaci. V této fázi získáme adresu na git k naší aplikaci a můžeme si ji přidat například do již zmíněného SourceTree. Spuštění instance zde však neprobíhá automaticky jako u příkazové řádky. Musí být v administraci Bluemixu nastaven úkol (job), který po nahrátí aplikace do gitu zařídí to, že se aplikace zavede na cloud a spustí (build and deploy). Na následujícím obrázku je vidět náhled nastaveného build and deploy úkolu z administrace Bluemixu:



Obr. 11 Ukázka build and deploy procesu při zavádění nové verze aplikace z gitu

Pro operaci build je nutné zvolit typ stavitele, v případu naší aplikace stačí typ simple. Pro deploy operaci nastavujeme workspace a cílovou aplikaci na Bluemixu.



Obr. 12 Repositář reportingového systému v softwaru SourceTree

Na obrázku č. 12 vidíme náhled do již zaběhlého repositáře v programu SourceTree. Můžeme zde spatřit jednotlivé updaty provedené historicky v čase a v pravém dolním rohu je okno s kódem jednotlivých souborů, kde můžeme sledovat změny na úrovni řádků kódu.

4.7 Django komponenty

V této podkapitole budou popsány jednotlivé komponenty Djanga, které z něj dělají tak užitečný nástroj a jejichž benefitů bylo využito při implementaci informačního systému v této práci.

4.7.1 Django Admin

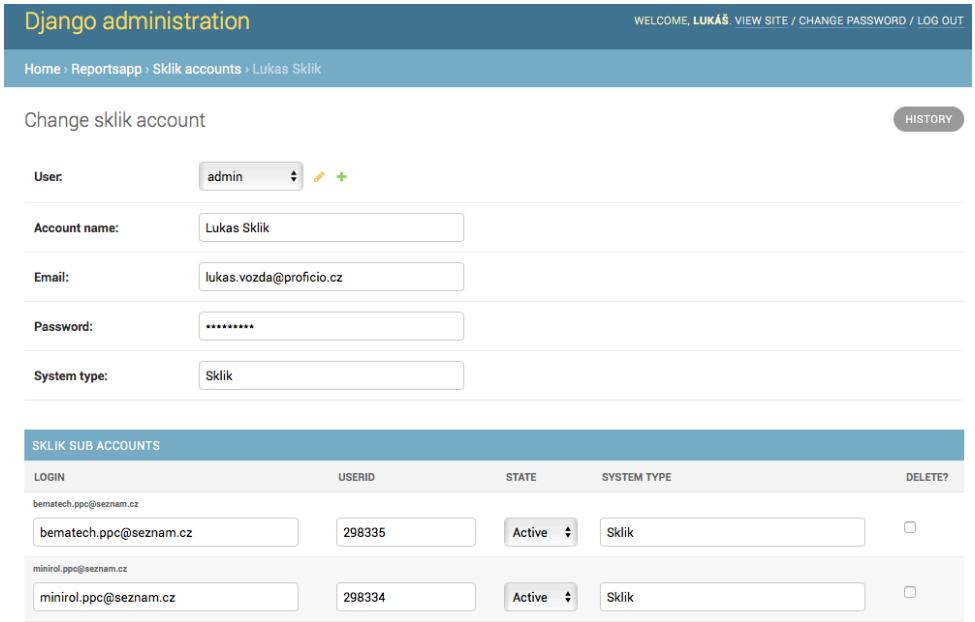
Django přichází s administrativním rozhraním, které je ihned po instalaci nového projektu připraveno k použití. Django Admin načítá metadata z modelů, které jsme vytvořili v rámci naší aplikace a poskytuje rychlé a přehledné rozhraní, ve kterém mohou uživatelé s dostatečnými právy řídit a spravovat databázi. Tato administrace však primárně slouží pro interní využití. Není navržena k tomu, abychom na ni stavěli celou aplikaci z uživatelské strany (Jaiswal, 2015, str. 6).

Administrace v Djangu má ještě jednu zajímavou vlastnost – je snadno a dobře upravitelná. V souboru admin.py, který leží v adresáři aplikace, můžeme přeregistrovat modely, které chceme v administraci zobrazit jinak. Můžeme například ovlivnit zobrazovaná pole, nastavit filtry a sloupce, podle kterých chceme vy-

hledávat. Zajímavým přídavkem jsou tzv. inline třídy, kterých můžeme využít k tomu, abychom v zobrazení připojili k sobě hodnoty z různých tabulek propojené cizími klíči. V případě reportingového systému takto můžeme na jedné stránce v administraci vidět například jednotlivé administrátorské účty Skliku (třída SklikAccount) a k nim přiřazené podúčty (SklikSubAccount). Na následující ukázce kódu je demonstrován zmíněný případ z naší aplikace:

```
class SubAccountInline(admin.TabularInline):
    model = SklikSubAccount
class SklikAccountAdmin(admin.ModelAdmin):
    list_display = ["__unicode__", "user", "system_type", "date_added"]
    inlines = [
        SubAccountInline,
    ]
    class Meta:
        model = SklikAccount
admin.site.register(SklikAccount, SklikAccountAdmin)
```

Nejprve definujeme inline třídu SubAccountInline, přiřadíme ji k modelu SklikSubAccount a poté deinfujeme třídu SklikAccountAdmin, v níž nastavíme sloupce, které chceme v administraci zobrazit (list_display) a přiřadíme inline třídu.



Obr. 13 Django administrace – ukázka propojených inline tříd

Na obrázku č. 13 si lze prohlédnout jak vypadá výsledek. Vidíme zde formulář s Sklik účtem a pod ním jsou vypsány veškeré podúčty. Můžeme je prohlížet, edito-

vat i mazat. Je tak pěkně vidět, co k čemu patří a jde to jednoduše spravovat. Pomáhá to hlavně při vývoji aplikace, samotná databáze není tolik přehledná. Vývojář má přehled nad modely a vidí, jestli se mu načítají správně data do databáze.

4.7.2 Autorizační systém

Django sebou přináší zabudovaný autorizační systém uživatelů. Tohoto systému je dobré využít, chceme-li v aplikaci využívat klasického přihlašování a zároveň se nechceme trápit s vlastním zabezpečením a vyvíjet znovu to, co již někdo vyvinul (Django, 2016).

Autorizační systém Djanga je dobře zabezpečený a zvládá všechny základní věci, které bychom od takového systému očekávali:

- uživatelské účty
- skupiny
- práva
- password hasing system
- formuláře a nástroje pro přihlašování
- sessions založené na cookies

Uživatelské účty jsou implementovány objektem *user*, který je jádrem autentizačního systému. Reprezentuje lidi, kteří přišli na web aplikace a mají možnost registrace, změny v profilu, asociace obsahu s uživatelem apod. Základními atributy jsou *username, password, email, fiirst_name, last_name* (George, 2015).

Reportingový systém, který je navrhován v této práci, bude využíván pouze interně cca pěti zaměstnanci. Alespoň je tak předpokládáno pro horizont jednoho roku. Proto není potřeba dbát důraz na skupinovou logiku uživatelů. V našem systému budou figurovat pouze administrátoři, kteří uvidí do administrace aplikace a jeden hlavní admin účet, který uvidí do administrace celého Djanga, bude moci vytvářet účty ostatním administrátorům a udělit jim tak přístup do systému (samotná registrace z venku nebude možná, protože jde o čistě interní projekt).

V šabloně *login.html* je připraven přihlašovací formulář, na který je automaticky uživatel přesměrován ze všech administrativních stránek systému, pokud do něj není přihlášen. Formulář se skládá jen z dvou vstupních textových polí pro uživatelské jméno a heslo a potvrzovacího tlačítka. Zajímavější je pohled do pozadí, tedy jak se zpracovávají informace poslané formulářem na straně serveru a jak probíhá autentizace uživatele v kódu.

```
def login(request):
    username = request.POST['username']
    password = request.POST['password']
    user = authenticate(username=username, password=password)
    if user is not None:
        if user.is_active:
            login(request, user)
            return redirect('/')
        else:
            # vrácení zprávy o neaktivním účtu
else:
        # vrácení zprávy o špatném jménu nebo heslu
```

Tento způsob je demonstrován v samotné dokumentaci Djanga, kdy formulář odesílá na server POST request s informacemi zadanými uživatelem, request je zpracován v Djangu a předán podle URL adres (o kterých bude více v příslušné kapitole) do patřičného pohledu, v našem případě je to pohled *login.* Provede se autentizace uživatele a pokud se zadaří a uživatel je aktivní, dojde k přesměrování na hlavní stránku administrace. Pokud se nepodaří přihlásit, vrátí se uživateli informace, kde se stala chyba.

U dalších pohledů bude nutné ověřovat, zda je uživatel přihlášen do systému či ne. Docílíme toho pomocí umístění dekorátoru před každou deklaraci pohledu:

```
@login_required(login_url='login')
def home(request):
    # definice pohledu těla pohledu home
```

Pomocí dekorátoru *@login_required()* zajistíme, aby došlo k ověření toho, zda je uživatel přihlášený ještě předtím, než spadne na danou stránku. V parametru dekorátoru předáváme *login_url,* tedy adresu, na kterou má být uživatel přesměrován v případě, že není přihlášen, aby tak mohl učinit. Django navíc za URL stránky s loginem při přesměrování připojí parametr *next* a k němu přiřadí cestu na to místo v aplikaci, kde se původně žádal dostat. To je dobré k tomu, aby po úspěšném přihlášení mohl pokračovat v práci tam, kde přestal.

Pro odhlášení uživatele v reportingovém systému je využito funkce logout, která je volána v následujícím pohledu:

```
def log_out(request):
    logout(request)
    return redirect('/login')
```

4.7.3 Modely a Object-Relational Mapper

Modely jsou ve své podstatě standardní třídy v Pythonu s přidanými vlastnostmi. Djangu jsou všechny modely podtřídou třídy *django.db.models.Model.* Na pozadí figuruje tzv. **Object-Relational Mapper** (dále jen ORM), který zajišťuje komunikaci Djanga s databází a jeho úkolem je převést modely do databázového jazyka. Díky tomuto nástroji při návrhu aplikace vývojář nemusí vůbec zasahovat do SQL kódu, stará se pouze o třídy a o vše ostatní se postará ORM. ORM se automaticky přizpůsobí databázovému systému, který je připojen k naší aplikaci, vývojář tedy vůbec nemusí řešit specifika a syntaxi daného systému, prostě se stará o své třídy napsané v Pythonu (Jaiswal, 2015, str. 71).

Ukázka, jak se definují modely v Djangu, přichází v následujícím kusu kódu na konkrétním případu modelu účtu Skliku:

```
class SklikAccount(models.Model):
    user = models.ForeignKey(settings.AUTH_USER_MODEL, default=1)
     account_name = models.CharField(max_length=50, unique=True,
null=False)
    email = models.EmailField(unique=True, blank=False, null=False)
    password = models.CharField(max_length=50)
    system_type = models.CharField(max_length=20, default="Sklik")
    date_added = models.DateTimeField(auto_now=False, auto_now_add=True)

def __unicode__(self):
     return unicode(self.account_name)
```

Po definici jména modelu přicházejí deklarace jednotlivých atributů. Povšimnout si lze hned v prvním atributu s názvem *user*, jak se zde řeší cizí klíče. Jelikož nezasahujeme do SQL, musíme propojení tabulek nadefinovat přímo v modelu. Docílíme toho tak, že do atributu přiřadíme typ pole models.*ForeignKey(),* kde v prvním parametru se odkážeme na model, s kterým chceme tabulku integrovat. V našem případě je to user model, zakomponovaný v Djangu, ale mohlo by to být jméno jakéhokoliv modelu, který jsme vytvořili. Na konci je uveden podprogram *__unicode__(),* který vrací Djangu pro uživatele čitelnou formu názvu modelu. V tomto případě je to atribut se jménem účtu, nadefinovat si však můžeme v podstatě cokoliv. Tento podprogram není povinný, pokud ho nedefinujeme Django bude implicitně zobrazovat generovaný řetězec.

Modely jsou znázorněny na obrázku č. 18 s diagramem tříd v přílohách dokumentu.

4.7.4 Migrace

Databázové migrace přicházejí do Djanga s verzí 1.7. Jde o poměrně zlomovou funkcionalitu, která ve starších verzích nebyla bez externích modulů realizovatelná. Migrace pomáhají s:

- zrychlují proces změny schématu databáze
- zjednodušují vysledování změn ve schématu databáze
- udržují synchronizované schéma databáze s kódem

Předtím, než přišly migrace, bylo velkým problém provádět změnu v již naplněné databázi, aniž by došlo ke ztrátě nebo poškození dat. Tak jako nám git pomáhá s přehledem nad verzemi kódu, migrace nám pomáhají udržet si přehled nad změnami v databázi (Real Python, 2014).

Migrace v Djangu řídíme pomocí příkazové řádky, kdy v projektovém adresáři naší aplikace zavoláme příkaz *./manage.py makemigrations*. Po jeho spuštění Django zkontroluje změny provedené v modelech a vytvoří nové migrace. Informaci o výsledku nám vypíše na standardní výstup:

```
Migrations for 'ReportsApp‘':
  0001_initial.py:
    - Create model SklikAccount
    …
```

Po tomto kroku se nám v adresáři *reportsApp/migrations* vytvoří soubor s názvem *0001_initial.py,* ve kterém jsou vygenerované provedené migrace. Nyní je třeba zavolat příkaz *./manage.py migrate*, který je zodpovědný za aplikování migrací do schémata databáze. Tento příkaz se zároveň postará o odstranění těch tabulek, jejichž příslušné modely byly z kódu odstraněny (Django, 2016).

4.7.5 Formuláře

Django disponuje vlastním formulářovým systémem. Ten se stará o všechno od ověřování uživatelských vstupů po převedení vstupů na objekty. Zároveň mají renderovací metody, které vygenerují formuláře přesně na míru modelům a samozřejmě jsou i upravitelné (Effective Django, 2013).

Formuláře vytváříme prostřednictvím třídy Form v souboru forms.py nacházejícím se v adresáři aplikace. V tomto souboru definujeme všechny formuláře. Jak vypadá implementace na konkrétním případu formuláře pro zaregistrování účtu ze systému Sklik do reportingového systému si můžeme prohlédnout v následujícím kódu.

```
class SklikAccountForm(forms.ModelForm):
   password = forms.CharField(widget=forms.PasswordInput())
   account_name = forms.CharField(max_length=20)
   email = forms.EmailField(max_length=200)
   error_css_class = "error-message"
class Meta:
      model = SklikAccount
      fields = [
            "user",
            "account_name",
            "email",
            "password",
      ]
```

Takto si definujeme formulář příslušící k modelu SklikAccount. V kódu vidíme jednotlivá pole, například pole na heslo s widgetem *PasswordInput()* se stará o zabezpečení hesla, *EmailField()* se zase postará o to, aby uživatel skutečně zadal validní e-mail a pokud tak neučiní, Django mu automaticky vrátí chybovou hlášku identifikující problém. V seznamu *fields* si nadefinujeme, která pole chceme ve formuláři vyplnit (často nechceme vyplňovat všechny atributy modelu, některé se generují automaticky apod.). Možností úprav a nastavení je celá řada, všechny jsou podrobně popsány v dokumentaci Django.

Užitečnou vlastností formulářů Django je také jednoduchost propojení obsahu formulář se samotným modelem. V pohledu můžeme zavolat instanci objektu z databáze a automaticky vykreslit v šabloně formulář s vyplněnými hodnotami. Nebo opačnou cestou, když uživatel vyplní formulář, Django ví, ke kterému modelu formulář patří a jak zpracovat vstupní hodnoty. Demonstrovat to můžeme na ukázce kódu z pohledu *create*, kde se vytváří report:

```
def create(request):
   if request.POST:
         form = ReportForm(request.POST)
         if form.is_valid():
           instance = form.save(commit=False)
           instance.save()
          ...
```

Tento výtažek z kódu zařídí to, že po přijetí POST požadavku na server Django přiřadí přijaté hodnoty z požadavku k objektu formuláře. Poté proběhne validace formuláře, pokud validace proběhne úspěšně, instance se uloží do databáze. V opačném případě Django vrátí formulář zpátky uživateli s vypsanými chybovými

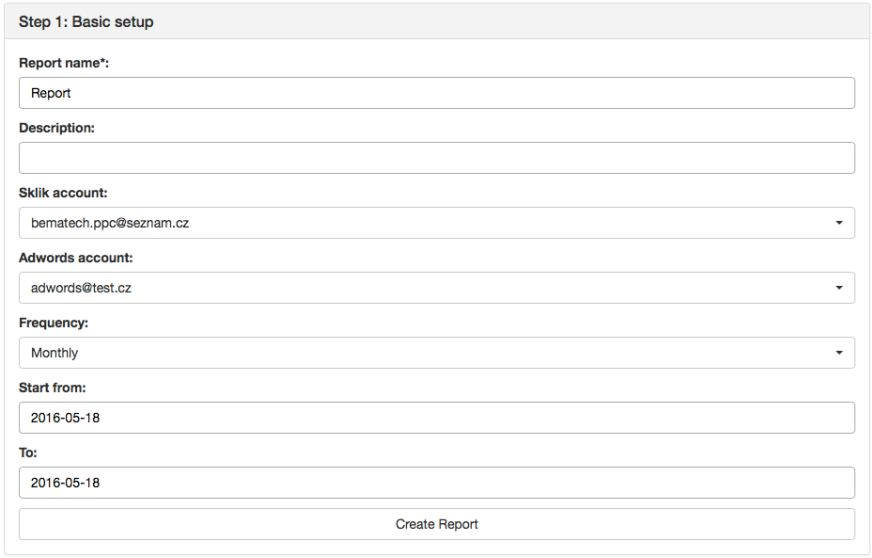
hláškami, které ho navedou ke správnému vyplnění. Tento způsob zpracovávání a tvorby formulářů usnadňuje vývojářům práci a šetří hodně času. Obzvlášť pro začátečníky je dobré, že se můžou opřít o zkušenosti vývojářského týmu frameworku a dospějí tak k rychlejšímu, efektivnějšímu a bezpečnějšímu řešení, než kdyby se snažili danou funkcionalitu navrhnout sami.

Poslední věc, kterou je třeba pro vykreslení formuláře do šablony zajistit, je aplikace tzv. *cross site request forgery protection* (dále jen CSRF). Stručně řečeno jde o ochranu proti útokům, kdy by se mohl někdo pokusit z jiného webu přes formulář, odkaz nebo skript vykonat nějakou akci na našem webu (Django, 2016).

Ochranu implementujeme do šablony pomocí tzv. CSRF tokenu, který umístíme do formuláře, vygeneruje neviditelné pole, které se odesílá spolu s formulářem a Django si na pozadí provádí ověřování, o nic se starat nemusíme:

```
<form action="" class="create-report-form" method="POST">
        {{ form.as_p }}{% csrf_token %}
        <button class="create btn"  type="submit">
          Create report
        </button>
</form>
```

Ukázaný kód vygeneruje do šablony formulář, který byl předán v rámci kontextu z pohledu do šablony k vyrenderování, v tomto případě jde o formulář příslušící třídě *Report*. Máme několik možností jak vykreslit formuláře pomocí voleb vykreslení, v tomto případě byla zvolena volba as_p (as paragraph), což znamená, že formulář bude obalen v <p> tagu. Další z možností vykreslení jsou například jako tabulka nebo jako nečíslovaný výčet.



Obr. 14 Ukázka vykresleného formuláře pro tvorbu reportu

Na obrázku č. 14 je zobrazen vygenerovaný formulář, u kterého jsme demonstrovali implementaci v kódu. Na tomto případu je pěkně vidět, jakou práci odvádí formuláře v Djangu, které automaticky vygenerovalo pole podle příslušných atributů třídy. Například u pole označeného Sklik account bylo automaticky vygenerováno výběrové okno, které po rozkliknutí nabízí seznam všech registrovaných účtů Skliku, které se nabízí pro tvorbu reportu. Formuláře v Djangu tedy skutečně šetří čas a síly vývojářů.

4.7.6 URL adresy

Každá stránka webové aplikace potřebuje svoji vlastní URL adresu, aby aplikace věděla jaký obsah uživateli zobrazit. Django využívá pro správu URL adres tzv. URLconf,. Je to v podstatě množina vzorců (masek), které se Django snaží spojit (najít shodu) s URL adresou požadavku, aby našlo ten správný pohled (Django Girls, 2015).

Zmíněné vzorce či masky jsou reprezentovány v Djangu regulárními výrazy v souboru url.py ležícím v projektovém adresáři aplikace. Regulární výrazy sebou nesou široké možnosti, jak definovat URL adresy. Některé příklady budou představeny v následujících ukázkách kódu.

```
url(r'^accounts/sklik/create', reportsApp.views.sklik_accounts),
url(r'^accounts/sklik', reportsApp.views.sklik_accounts_create),
```

Nejprve přichází regulární výraz a druhým parametrem je pohled, který má Django při shodě regulárního výrazu a URL adresy zpracovat. Detailně vysvětlovat regu-

lární výrazy je mimo téma tohoto textu, v uvedené ukázce si jde však všimnout například znaku ^, který představuje začátek řádku. Můžeme takto využít všech možností, které regulární výrazy přinášejí. Důležité je pořadí definovaných URL adres, jakmile totiž dojde ke shodě, Django přestane dál pokračovat v hledání a automaticky přejde na daný pohled. Musíme tedy zachovat hierarchii URL adres, tak jako je uvedené na ukázce kódu výše, nejprve musíme uvádět ty specifické regulární výrazy a poté postupovat k těm obecnějším. Proto je nejprve regulární výraz s maskou *accounts/sklik/create* a až za ním výraz *accounts/sklik.* V opačném případě by se stalo to, že na stránku s účty Skliku by nebylo možné se dostat.

Tento způsob definování URL adres je velice pohodlný a přímočarý. Co když ale předem nevíme, jaká bude URL adresa? Například u reportingového systému budeme chtít mít URL adresy pro jednotlivé reporty a předem není vývojáři známo, jaké budou jejich jména, aby mohl nadefinovat URL adresy. I na toto vývojáři Djanga mysleli. Do regulárního výrazu totiž můžeme zakomponovat i parametr, který si pojmenujeme a předáme do parametru pohledu. Tam je pak jednoduše zpracován a z databáze se požádá o objekty, které si uživatel zažádal v rámci URL adresy:

```
url(r'^report/(?P<slug>[-\w]+)/', reportsApp.views.report_view)
```

Na ukázce kódu je uvedena definice URL adresy pro stránku s reportem. Cesta začíná řetězcem *report/,* za ním následuje parametr se jménem *slug*, je tak pojmenovaný proto, že se tak jmenuje atribut modelu *report*, tedy pro zachování přehlednosti. Název slug se využívá pro speciálně upravený řetězec, aby byl použitelný pro URL adresu. Generuje se z uživatelem napsaného názvu reportu, ve kterém můžou být mezery a další pro URL nepřípustné znaky. Za názvem parametru přichází opět regulární výraz. Ten reprezentuje řetězec, který se přenese v parametru do příslušného pohledu a bude využit jako klíč k vyhledání konkrétního reportu v databázi.

4.7.7 Views

Pohledy (views) jsou v Djangu v podstatě funkce, které přijímají jako parametr požadavek na server (request) a vrací odpověď (response). Odpovědí může být např. HTML stránka, přesměrování, 404 error, obrázek atd. V těle pohledu je definována logika potřebná k vygenerování požadovaného výsledku (Django, 2016).

Pohledy jsou deklarovány v souboru views.py v adresáři aplikace. Pro ukázku pohledu byl vybrán pohled *report_view,* který vrací uživateli stránku s vykresleným reportem. V pohledu je poměrně dost řádků kódu, proto budou vybrány jen ty podstatné části pro zachování přehlednosti.

```
def report_view(request, slug):
    r = Report.objects.get(slug=slug)
    sklik_account = SklikSubAccount.objects.get(login=r.sklik_account)
    data = Campaign.objects.filtter(account=sklik_account).order_by('type').
    .prefetch_related('metric_set').all()
    context = {
        "report": r,
        "data": data
    }
    return render(request, "report.html", context)
```

Jak už bylo zmíněno, pohledy přebírají v parametru *request* objekt, tento pohled navíc přijímá parametr *slug,* kterého je využito hned na prvním řádku uvnitř funkce pro získání instance třídy *Report* z databáze. Na druhém řádku se do proměnné *sklik_account* přiřadí příslušný účet, který je asociován s reportem. Na třetím řádku uvnitř pohledu je pokročilejší *queryset,* kde nejprve dotazujeme z databáze kampaně příslušící danému účtu a pak pomocí funkce *prefetch_related* ke každé kampani připojíme metriky. Jde jakoby o reverzní získávání dat, v šabloně chceme renderovat kampaně a pod nimi mít agregované metriky, přičemž v databázi jsou metriky a ty ukazují cizím klíčem na kampaně. V opačném případě by nebylo potřeba funkci *prefetch_related* volat. Na čtvrtém řádku uvnitř funkce přichází proměnná s názvem *context*, tohoto názvu je využito ve všech pohledech, protože jde o kontextovou proměnnou sloužící k tomu, abychom přenesli data do šablony a mohli je vykreslit. Je to proměnná typu, kterému se v Pythonu říká dictionary (slovník), protože obsahuje dvojice tvořené názvem a hodnotou. Na posledním řádku vrací pohled funkci *render()*, která má v parametru request, název šablony, do které se bude vykreslovat a kontextovou proměnnou.

4.7.8 Templates

Šablony v Djangu, podobně jako u multi-view contollerů, slouží k oddělení samotných dat od dokumentu, do kterého se budou data vykreslovat. Šablony využíváme nejčastěji pro generování HTML obsahu, ale můžou být využity i na jiný textově založený formát. Díky šablonám nemusíme tvořit každou HTML stránku webu zvlášť, ale pomocí zástupných symbolů a proměnných můžeme vygenerovat spousty stránek s dynamicky generovaným obsahem a ušetřit si tak spoustu psaní. Velké benefity z toho pak plynou i při údržbě. Změnu nemusíme dělat například na stránce profilu každého uživatele, ale pouze v jedné šabloně (Holovaty, 2014).

V reportingovém systému jsou šablony uchovávány ve složce *templates* nacházející se v adresáři aplikace. Šablony nabízejí opravdu široké možnosti, všechny jsou přehledně popsány v dokumentaci Djanga. Ukážeme si však na pár případech,

jak jsou šablony využívány v našem informačním systému. Základní šablonou je šablona *base.html,* na začátku každé šablony musíme uvést krátký kód *{% load staticfiles %},* aby se nám načítal statický obsah (soubory se styly, obrázky, scripty atd.). Na každé stránce ve hlavičce je proměnná s titulkem:

```
<title>{{ template_title }}</title>
```

Obsah této proměnné definujeme v příslušném pohledu a předáváme v kontextové proměnné a v šabloně je volána tímto způsobem.
Statické soubory si můžeme demonstrovat na případu souboru se styly:

```
<link href="{% static "reportingSystem/css/style.css" %}" rel="stylesheet">
```

Šablonovací systém Django automaticky doplní cesty ke statickým souborům, když potom tedy například přeneseme aplikaci na jinou URL nebo server, statické soubory se budou i nadále načítat a nemusíme se o nic starat.
Uvnitř šablon si může vývojář definovat tzv. bloky. Ty jsou užitečné pokud například vytvoříme novou šablonu, kterou pomocí řádku kódu *{% extends "base.html" %}* na začátku šablony nadstavíme na hlavní šablonu. V nové šabloně pak uvnitř bloku může vývojář vytvořit úplně jiný obsah, čímž přepíše obsah z hlavní šablony, zbytek stránky se přitom vygeneruje z šablony *base.html.* V reportingovém systému je této funkcionality využíváno. Na většině stránek se opakují ty stejné prvky (menu, logo) také se načítají stejné styly atd. Proto šablony rozšiřují šablonu *base.html*, ve které jsou nadefinovány tyto prvky a v dílčích šablonách je definován pouze blok s obsahem. Dalším případem využití je blok se skripty. Na jednotlivých podstránkách jsou skripty, které není třeba spouštět nikde jinde, mnohdy jsou také načítány javascriptové knihovny například pro generování grafu atd. Ty zase nejsou potřeba na stránkách, kde nejsou grafy. Proto je v šablonách systému definován i blok *page_script*, ukázka je vidět v následujícím kódu:

```
{% block page_script %}
<script>
   $(function() {
    $( "#datepicker1" ).datepicker({
      dateFormat: "yy-mm-dd",
    });});
</script>
{% endblock %}
```

V tomto případě obsahuje blok *page_script* pouze krátký javascriptový snippet, který inicializuje ve formuláři u pole s datem vyskakovací okno s kalendářem.

V následujícím výtažku z kódu je ukázáno, jakým způsobem je řešeno vypisování kampaní a jim příslušících metrik:

```
{% for c in data %}
{% if c.state == "A" %}
  {% for m in c.metric_set.all %}
    {% if m.name == "ctr" or m.name == "conversionRate" %}
      <td class="metric-cell">{{m.value}} %</td>
     <td class="metric-cell">{{m.value|intcomma}}</td>
    {% endif %}
  {% endfor %}
  {% endif %}
{% endfor %}
```

Pozn.: výtažek z kódu je pro demonstrativní účely značně zjednodušený a očištěný o části kódu, které nejsou nyní podstatné.

4.8 Databáze

Nedílnou součástí systému je také databáze. Ta je využita k ukládání veškerých informacích od uživatelů, přes reklamní systémy až po jednotlivé metriky v reportech. API reklamních systémů mají většinou omezený počet operací nebo přístupů, proto potřebujeme data ukládat ve vlastní databázi a nedotazovat se na ně při každém zobrazení reportu znovu.

Django dokáže spolupracovat se všemi nejznámějšími databázovými systémy. Při návrhu reportingového systému byl zvažován databázový systém PostgreSQL, což je sofistikovaný open-source databázový systém oblíbený v mnoha projektech (Obe, 2012, str. 12).

Služba databázového systému PostgreSQL na cloudu IBM Bluemix naneštěstí neumožňovala vzdálené připojení narozdíl od databáze MySQL, ke které se na Bluemixu dalo připojit odkudkoliv, což přinášelo jistá zjednodušení při vývoji a testování nahrávání dat do databáze. Pro ostrý provoz by se samozřejmě databáze PostgreSQL byla použitelná.

Jelikož Django disponuje nástrojem object-relational mapper (ORM), schéma databáze nebylo navrhováno klasicky, byly však navrženy třídy (modely), ze kterých ORM vygeneroval databázovou strukturu. Schéma databáze vygenerované pomocí nástroje reverse engineering v sofrwaru MySQL Workbench je v příloze dokumentu na obrázku č. 17 na straně 60.

4.9 Sklik API

Data z reklamního systému Sklik od společnosti Seznam.cz, a.s. budou do naší databáze nahrávána přes API Skliku, které se ve své aktuální verzi jmenuje Cipisek (Sklik API, 2016).

K tomuto API se dá připojit přes XML-RPC (remote procedure call) protokol, který je implementovatelný v několika programovacích jazycích. Zkratka v názvu protokolu RPC v překladu znamená vzdálené volání procedur. Díky tomuto protokolu je tedy možné spojovat různá zařízení, operační systémy a software napříč internetem a zařídit mezi nimi komunikaci, která probíhá přes HTTP protokol (XML-RPC, 2011).

Pro komunikaci s tímto API byly naprogramovány podpůrné funkce v adresáři functions nacházejícím se v adresáři aplikace reportsApp. Implementaci XML-RPC protokolu si ukážeme v následujícím výtažku z kódu s funkcí, která zajišťuje verifikaci uživatelem zadaného Sklik účtu.

```
def sklik_verificate( login, pwd ):
    sklik = xmlrpclib.ServerProxy( sklik_api_url )
    try:
        sessStruct = sklik.client.login(login, pwd)
        session = { 'session': sessStruct["session"]}
        client_status = sklik.client.get(session)["status"]
        if client_status == 200:
            return True
        else:
            return False
    except xmlrpclib.ProtocolError as error:
        print "A protocol error occurred"
        print "Error code: %d" % error.errcode
        print "Error message: %s" % error.errmsg
        return False
```

Funkce *sklik_verificate* přebírá jako parametr přihlašovací jméno Sklik účtu a heslo. Uvnitř funkce se pokusíme navázat spojení s Sklik API. Pokud se povede spojení navázat, získáme session string a požádáme o status klienta. Pokud je status klienta 200, je klient validní a verifikace proběhla úspěšně. V opačném případě funkce vrátí hodnotu False, která je dále zpracována v daném pohledu. V případě, že se vyskytne chyba v protokolu, odchytí se výjimka a vypíše se na standardní výstup chybová hláška, funkce v tomto případě vrátí hodnotu False, protože verifikace se

nezdaří. Podobným způsobem jsou naimplementovány i další funkce, které slouží například k získání podúčtů, kampaní nebo metrik.

4.10 Google API

Stejně jako reklamní systém Sklik tak i Google AdWords má své API. Google má však výrazně striktnější pohled na zabezpečení, protože si je dobře vědom, jak citlivá data uchovává o svých klientech v databázích a nechce dopustit, aby jich kdokoliv zneužil (AdWords API, 2016).

Narozdíl od Sklik API, ke kterému se může připojit v podstatě kdokoliv ze dne na den, u AdWords API to tak není. Jménem agentury je třeba zažádat u Googlu o tzv. developer token. Je zde přísný schvalovací proces. Google chce vidět přesný popis aplikace, k čemu bude sloužit, kdo ji bude používat, jak často atd. Navíc chtějí vidět samotný návrh aplikace a prohlédnout si, jestli nebude využita k nekalým činnostem. Je samozřejmě dobré, že Google dbá na zabezpečení a snaží se tak ochránit své klienty, protože reklamní systém AdWords je jeho hlavním zdrojem příjmu (Jašek, 2014, str. 88). Pro tuto práci to však znamená malou překážku, protože se nepodařilo token před dokončením získat. Pro otestování aplikace to nebude problém, protože můžeme testovat na datech ze systému Sklik. Po přidělení přístupu do AdWords API bude stačit doprogramovat podpůrné funkce pro sběr dat stejně tak, jak bylo ukázáno v předchozí kapitole. Tento postup je aplikovatelný na každý podobný reklamní systém, který by si mohla agentura PROFICIO v budoucnu přát připojit.

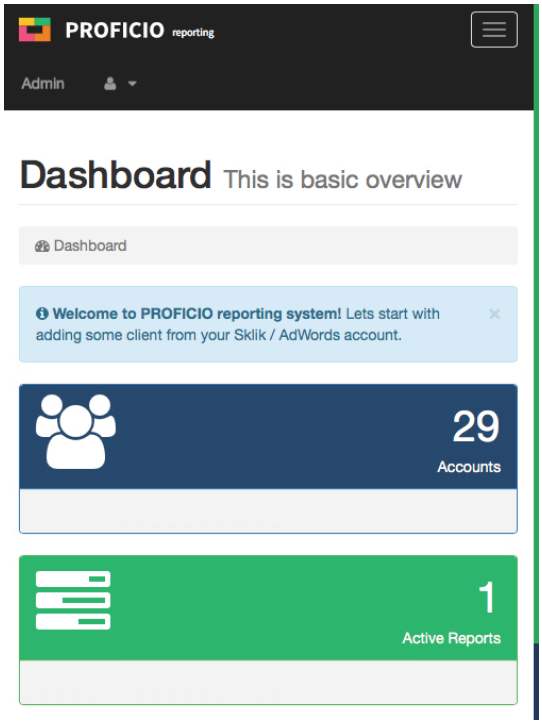
4.11 Frontend

V této kapitole budou uvedeny ukázky nástrojů využitých na uživatelské straně aplikace (frontendu).

4.11.1 Bootstrap

Bootstrap je hodně populární framework mezi frontend vývojáři. Dělá vývoj webu rychlejším a jednodušším. Je použitelný u projektů většího měřítka i u mikro webů a funguje dobře na naprosté většině dostupných zařízení (Bootstrap, 2016).

Pro informační systém tvořený v této práci je využito Bootstrapu hlavně pro jednoduché zavedení responzivního designu. Ukázka je vidět na následujícím obrázku.



Obr. 15 Náhled responzivního designu aplikace

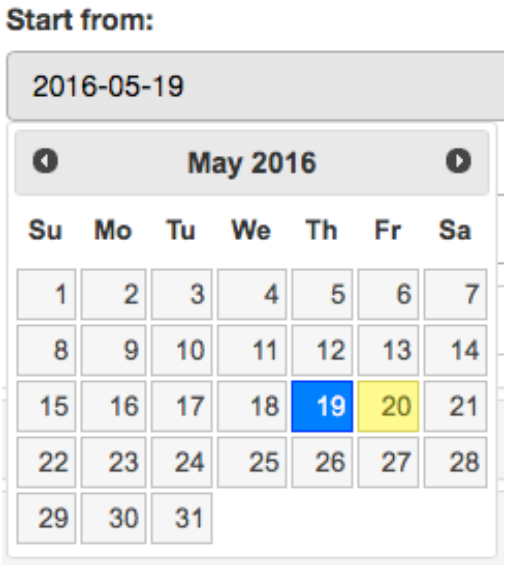
Pro responzivní design je v aplikaci využito tzv. grid systému, kdy webová stránka je pomyslně rozdělena do 12 sloupců, které se automaticky přizpůsobují velikosti okna prohlížeče. Framework sebou zároveň nese podpůrné skripty, které zajistí, aby se boční menu složilo do horní lišty a umožnilo tak lepší manipulaci na mobilních zařízeních.

4.11.2 jQuery UI

jQuery UI je knihovna napsaná v jazyce Javascript. Pomáhá webovým vývojářům usnadnit práci při tvorbě určitých prvků uživatelského rozhraní (JQuery UI , 2016).

V této práci bylo využito knihovny jQuery UI pro kalendáře (datepickery), které umožní uživateli po kliknutí na formulářové políčko zobrazit kalendář a zvolit datum v předem definovaném formátu. Toto datum se automaticky po výběru zapíše do pole formuláře, což usnadní práci nejen vývojáři ale i samotnému uživateli aplikace. Uživatel nemusí přemýšlet nad tím, v jakém formátu zadat datum do pole a vývojář zase nemá tolik práce s ošetřováním vstupu.

Ukázka okna s kalendářem u formuláře pro vytvoření reportu je k vidění na následujícím obrázku.



Obr. 16 Ukázka kalendáře pro výběr data z knihovny jQuery UI

Vstupní pole stačí v kódu označit identifikátorem (třídou nebo id) a potom krátkým javascriptovým snippetem iniciovat datepicker. Dále bylo knihovny využito pro vyhledávání ve výběrových polích formuláře, u kterých se předpokládá dlouhý seznam možností, z kterých se bude vybírat (například u Sklik a AdWords účtů).

4.11.3 Morris.js

Morris.js je také knihovna napsaná v jazyce Javascript. Slouží však k usnadnění vizualizace dat v grafech. Vykreslovat vektorové grafy v prohlížeči není jednoduchou záležitostí a proto existují knihovny tohoto typu, které to vývojářům usnadní (Morris.js, 2013).

V systému pro tvorbu reportů jsou grafy základním kamenem, bylo tedy nezbytné vyhledat takovou knihovnu, která by umožnila jednotným způsobem zobrazovat grafy a zároveň nebyla příliš komplikovaná na implementaci. Knihovna Morris.js těmto kritériím vyhovuje.

4.12 Testování

Předtím než se aplikace nasadí do ostrého provozu, je třeba ji pořádně otestovat. Testovat můžeme několika způsoby – uživatelsky, automaticky v kódu nebo za pomocí nějakého nástroje. Aby mělo uživatelské testování smysl, potřebovali bychom nasbírat větší vzorek dat, což může být časově i finančně nákladné. Proto naši webovou aplikaci pro tvorbu reportů otestujeme pomocí tzv. automatického testování a pomocí nástroje Selenium.

4.12.1 Automatické testování

Automatické testování je užitečný nástroj pro identifikaci chyb v kódu a je frekventovaně využíván v moderním programování. Princip spočívá v tom, že při psaní kódu můžeme ihned ověřovat, zda funguje přesně podle očekávání (Django, 2016).

Testy jsou napsané v souboru tests.py ležícím v adresáři aplikace reportsApp. Jako ukázku uvedeme otestování přihlašovací funkce do Sklik API:

```
class SklikLoginTest(TestCase):

  def test_sklik_login_with_wrong_details(self):
    """
    sklik_login() should return False when wrong account/password given
    """
    login = "retezec123"
    pwd = "123adsf"
    self.assertEqual(sklik_login(login, pwd), False)
```

Automatkcé testy v Djangu spustíme v terminálu pomocí příkazu *./manage.py test.* Na standardní výstup se vypíše podobná hláška:

```
Creating test database for alias 'default'...
.
----------------------------------------------------------------------
Ran 3 test in 0.277s

OK
Destroying test database for alias 'default'...
```

Výstupem je informace o tom, kolik testů proběhlo. Testovací systém Django zároveň vytvoří pro test databázi, kterou po skončení testu smaže. Pokud bychom chtěli databázi zachovat, abychom například ověřili, co se nám do databáze ukládá, přidáme za příkaz spouštěcí test volbu *--keepdb*.

4.12.2 Selenium

Na rozdíl od automatického testování, které probíhá uvnitř Djanga, webovou aplikaci můžeme otestovat i externě pomocí nástroje Seleniu. Potenciál Selenia tkví v tom, že dokáže automatizovat činnost webového prohlížeče. Selenium podporuje většinu nejpoužívanějších prohlížečů a nabízí tak široké možnosti testování i napříč platformami (Selenium, 2016).

V následující ukázce kódu je skript, který testuje přihlašovací stránku naší webové aplikace:

```
#!/usr/bin/env python
from selenium import webdriver
driver = webdriver.Firefox()
driver.get("http://reportingsystem.mybluemix.net/login")

username = driver.find_element_by_id("username")
password = driver.find_element_by_id("password")
username.send_keys("admin")
password.send_keys("admin")

driver.find_element_by_name("submit").click()

driver.quit()
```

Podobným způsobem si lze naskriptovat různé scénáře průchodem aplikací a pomocí jednoho skriptu je nechat automatizovaně otestovat. To ušetří spoustu času vývojářům při opakovaném testování aplikace z uživatelské strany.

5 Závěr

V této práci byl navržen a implementován informační systém, který automatizuje rutinní tvorby reportů, opakující se každý měsíc. Nejprve byly identifikovány požadavky prodiskutováním se specialisty z marketingové agentury PROFICIO s.r.o. Na základě těchto požadavků byl proveden průzkum na trhu a srovnány existující řešení. Žádný z existujících systémů plně nevyhovoval požadavkům, navíc sebou nesly finanční náklady. Proto se přistoupilo k návrhu vlastního řešení.

Pro implementování reportingového systému byl zvolen programovací jazyk Python a webový framework Django. Při vývoji aplikace bylo využíváno funkcí a nástrojů, které Django přináší. Součástí práce bylo také seznámit se s cloud systémem IBM Bluemix, nastudovat jeho principy a nasadit na tento cloud navrženou aplikaci pro tvorbu reportů. Aplikace byla i v průběhu konzultována se zástupci agentury a díky tomu bylo možné ladit systém již v průběhu vývoje a dosáhnout tak požadovaného výsledku.

Výsledkem je aplikace, která vyhovuje všem zadaným kritériím marketingové agentury a zároveň sebou nenese takovou finanční zátěž jako systémy třetích stran. V aplikaci se dá jednoduše na pár kliknutí přihlásit reklamní účet a poté vytvořit report na základě zvoleného období a zvolení účtu. Automaticky je vygenerována unikátní URL adresa pro klienta, kterou mu stačí zaslat. Netřeba již vytvářet dokumenty v excelu, posílat je každý měsíc v příloze atd. Díky tomuto novému řešení se očekává finanční úspora 16 850 Kč za měsíc.

V budoucnu by bylo dobré rozšířit systém o další možnosti a funkcionalitu jako může být například přidání historických dat do reportu, aby mohl klient vidět vývoj nejen za aktuální zvolené období, ale srovnat třeba s minulým rokem. Dále by bylo dobré obohatit uživatelské rozhraní například o možnosti filtrování reportů, řazení v tabulkách, možnost sloučit kampaně v reportu, možnost exportovat report do pdf atd.

6 Literatura

AdWords API*: Signing up for the AdWords API* [online]. 2016 [cit. 2016-05-21]. Dostupné z: https://developers.google.com/adwords/api/docs/signingup#step3a

BAXTER, RYAN J. *Accessing Application Logs In Bluemix* [online]. 2014, , 2 [cit. 2016-05-17]. Dostupné z: https://developer.ibm.com/bluemix/2014/10/29/accessing-application-logs-bluemix/

Bootstrap [online]. 2016 [cit. 2016-05-20]. Dostupné z: http://getbootstrap.com

Bootstrap: *Components* [online]. 2015 [cit. 2016-05-20]. Dostupné z: https://console.ng.bluemix.net/pricing/

Cloud Foundry Docs: *Buildpacks* [online]. [cit. 2016-05-16]. Dostupné z: http://docs.cloudfoundry.org/buildpacks/

Cloud Foundry: Features. *CloudFoundry.org* [online]. 2015 [cit. 2016-05-06]. Dostupné z: https://www.cloudfoundry.org/learn/features/

Digital Ocean: How To Use MySQL or MariaDB with your Django Application on Ubuntu 14.04. *Python* [online]. 2015 [cit. 2016-05-16]. Dostupné z: https://www.digitalocean.com/community/tutorials/how-to-use-mysql-or-mariadb-with-your-django-application-on-ubuntu-14-04

Django Girls: *Django urls* [online]. 2015 [cit. 2016-05-18]. Dostupné z: http://tutorial.djangogirls.org/en/django_urls

Django: *Cross Site Request Forgery protection* [online]. 2016 [cit. 2016-05-18]. Dostupné z: https://docs.djangoproject.com/en/1.9/ref/csrf/

Django: *Django admin site* [online]. 2016 [cit. 2016-05-18]. Dostupné z: https://docs.djangoproject.com/en/1.9/ref/contrib/admin/

Django: *Migrations* [online]. 2016 [cit. 2016-05-18]. Dostupné z: https://docs.djangoproject.com/en/1.9/topics/migrations/

Django: *Testing in Django* [online]. 2016 [cit. 2016-05-21]. Dostupné z: https://docs.djangoproject.com/en/1.9/topics/testing/

Django: Why Django? *Django: The web framework for perfectionists with deadlines.* [online]. 2016 [cit. 2016-05-06]. Dostupné z: https://www.djangoproject.com/start/overview/

Django: *Writing views* [online]. 2016 [cit. 2016-05-18]. Dostupné z: https://docs.djangoproject.com/en/1.9/topics/http/views/

Effective Django*: Form Basics* [online]. 2013 [cit. 2016-05-18]. Dostupné z: http://www.effectivedjango.com/tutorial/forms.html

FERREIRA, Carlos. *Django'ing with Bluemix* [online]. 2014, s. 3 [cit. 2016-05-16]. Dostupné z: https://bluemixstirred.wordpress.com/2014/05/29/almost-djangoing-with-bluemix/

GEORGE, Nigel. *Mastering django: Core*. 1. MasteringDjango.com, 2015.

Google Trends*: Explore* [online]. 2016 [cit. 2016-05-04]. Dostupné z: https://www.google.com/trends/

Gunicorn: Python WSGI HTTP Server. *Python* [online]. 2015 [cit. 2016-05-16]. Dostupné z: http://gunicorn.org

HOLOVATY, Adrian. *The Django Book* [online]. 1. 2014 [cit. 2016-05-19]. Dostupné z: http://www.djangobook.com/en/2.0/chapter04.html

IBM Bluemix DevOps Services: DevOps Made Easy [online]. 2016 [cit. 2016-05-17]. Dostupné z: https://hub.jazz.net

IBM Bluemix: Pricing [online]. 2016 [cit. 2016-05-20]. Dostupné z: https://console.ng.bluemix.net/pricing/

JAISWAL, S. -- KUMAR, R. *Learning Django Web Development: From idea to prototype, a learner's guide for web development with the Django application framework.* Birmingham B3 2PB, UK: Packt Publishing Ltd, 2015. 336 s. ISBN 978-1-78398-440-4.

JAŠEK, P. *Online marketing.* 1. vyd. Brno: Computer Press, 2014. 212 s. ISBN 978-80-251-4155-7.

JQuery UI [online]. 2016 [cit. 2016-05-20]. Dostupné z: https://jqueryui.com

KATZEN, Erik. *3 Different Ways to Deploy on Bluemix* [online]. 2015, , 1 [cit. 2016-05-17]. Dostupné z: https://bluemixhub.wordpress.com/2015/02/10/3-different-ways-to-deploy-on-bluemix/

LENNON, Joe. Deploying Django applications to a production server. *IBM developerWorks* [online]. 2009, 2009, 5 [cit. 2016-05-06]. Dostupné z: http://www.ibm.com/developerworks/library/os-django/

MEASUREFUL: *PRICING. MEASUREFUL* [ONLINE]. 2016 [CIT. 2016-05-03]. DOSTUPNÉ Z: http://measureful.com/

Morris.js [online]. 2013 [cit. 2016-05-20]. Dostupné z: http://morrisjs.github.io/morris.js/

Ninjacat Inc*.: How can NinjaCat's all-in-one reporting platform help you? Ninjacat* [online]. 2016 [cit. 2016-05-03]. Dostupné z: https://www.ninjacat.io/platform

OBE, Regina O. a Leo S. HSU. *PostgreSQL: up and running*. Second edition. ISBN 9781449373160.

PETERS, Tim. *The Zen of Python* [online]. 2004 [cit. 2016-05-05]. Dostupné z: https://www.python.org/dev/peps/pep-0020/

Pip: *Instalation* [online]. 2014 [cit. 2016-05-20]. Dostupné z: https://pip.pypa.io/en/latest/installing/

Python: dj-database-url. *Python* [online]. 2015 [cit. 2016-05-16]. Dostupné z: https://pypi.python.org/pypi/dj-database-url

Raventools: Afordable plans for busy marketers. *Raventools* [online]. 2016 [cit. 2016-05-03]. Dostupné z: https://raventools.com/pricing/

Real Python: *Django Migrations - a Primer* [online]. 2014 [cit. 2016-05-18]. Dostupné z: https://realpython.com/blog/python/django-migrations-a-primer/

Reportgarden: Project management software. *Reportgarden* [online]. 2016 [cit. 2016-05-03]. Dostupné z: http://reportgarden.com/ad-agency-project-management-software/

Selenium: *Browser automation* [online]. 2016 [cit. 2016-05-21]. Dostupné z: http://www.seleniumhq.org

Sklik API: *Cipisek* [online]. 2016 [cit. 2016-05-21]. Dostupné z: https://api.sklik.cz/cipisek/

SWEIGART, Al. *Automate the boring stuff with python: practical programming for total beginners*. San Francisco, CA: No Starch Press, 2014. ISBN 1593275994.

TOMALA-REYES, Angel. *What is IBM Bluemix: IBM's Open Cloud Architecture implementation based on the Cloud Foundry project* [online]. , 3 [cit. 2016-05-06].

Ultimate Django: Serving Static Files. *Python* [online]. 2015 [cit. 2016-05-16]. Dostupné z: https://ultimatedjango.com/learn-django/lessons/serving-static-files/

XML-RPC: *What is XML-RPC?* [online]. 2011 [cit. 2016-05-21]. Dostupné z: http://xmlrpc.scripting.com

Zoho Reports: Connect any data source. *Zoho Reports* [online]. 2016 [cit. 2016-05-03]. Dostupné z: https://www.zoho.com/reports/features.html

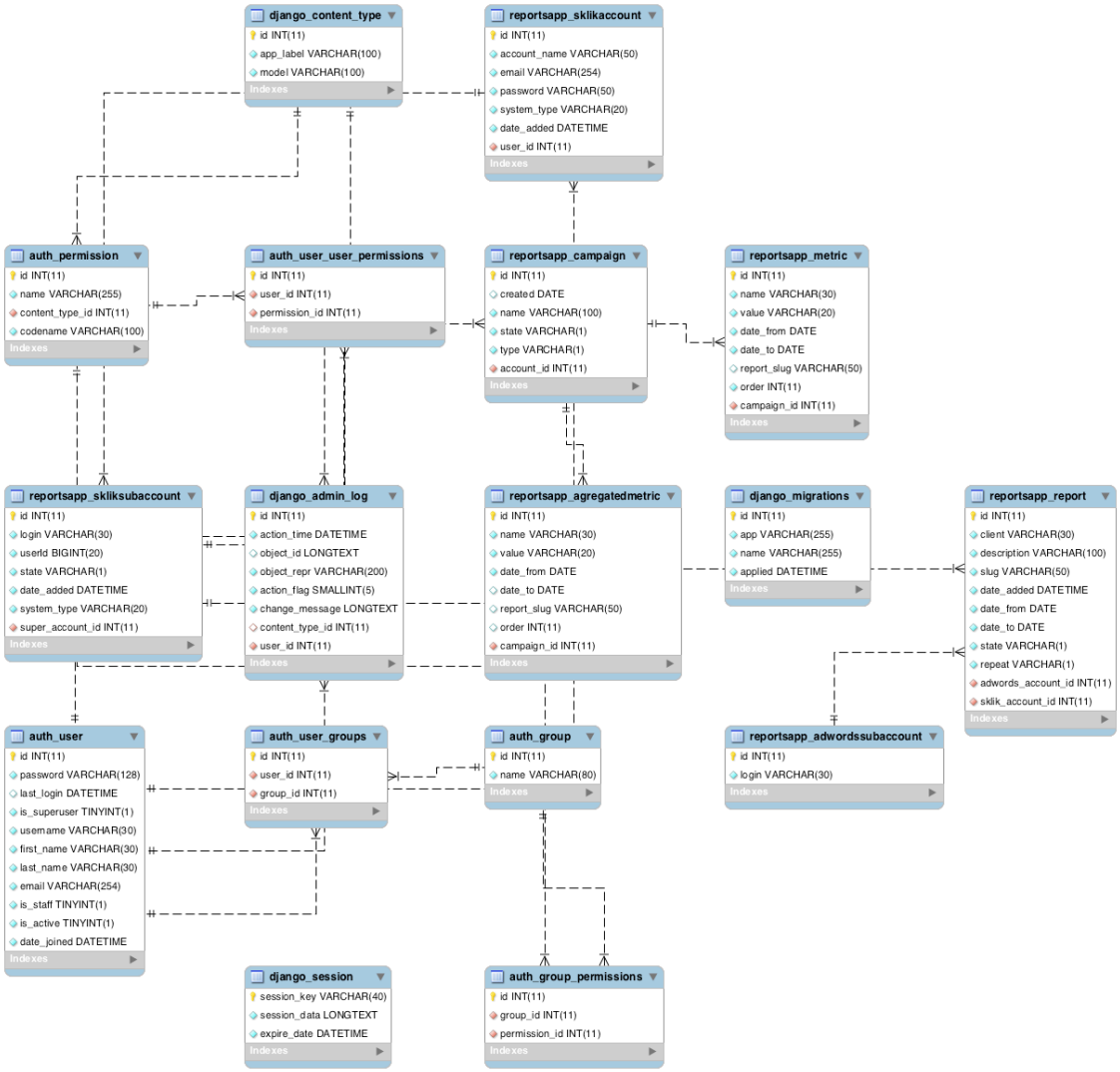
7 Seznam obrázků

8 Seznam tabulek

Přílohy

A Schéma databáze

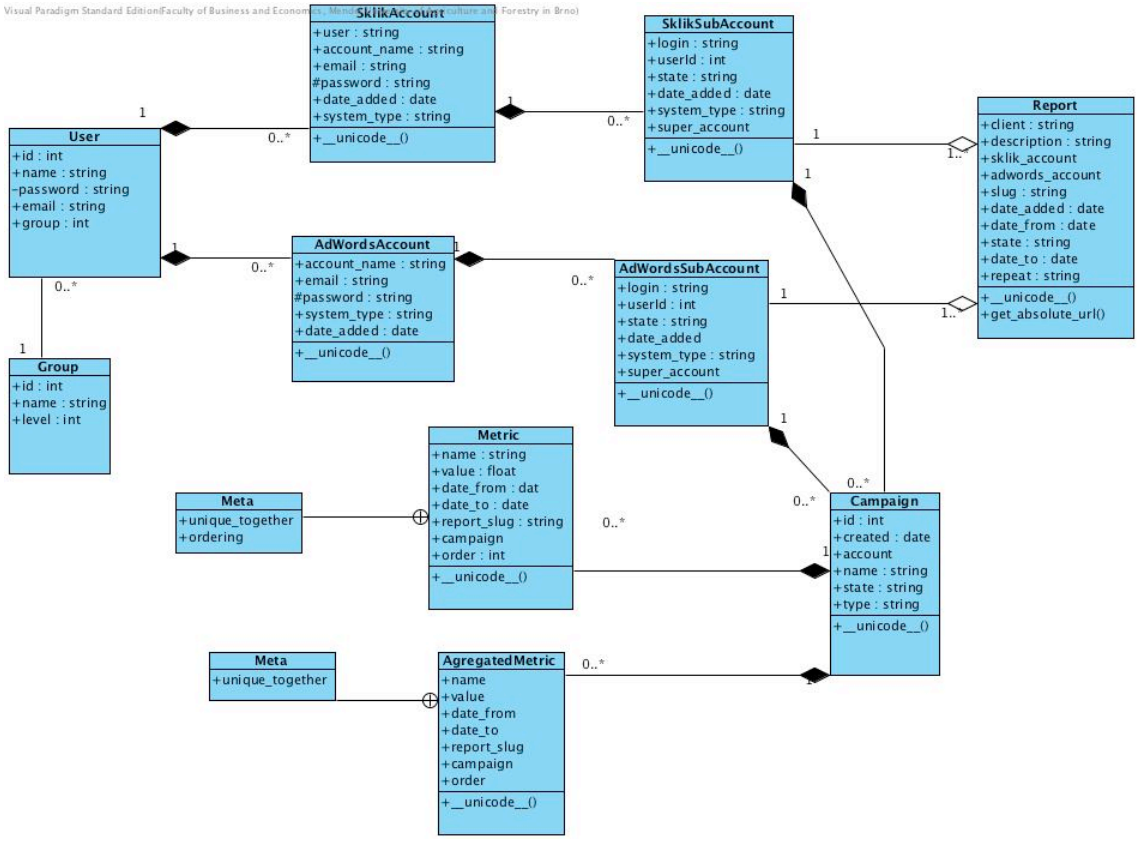
Na následujícím obrázku je schéma databáze informačního systému. Toto schéma je vytvořeno pomocí nástroje reverse engineering v softwaru MySQL Workbench. Databáze byla vygenerována object relational mapperem v Djangu z nadefinovaných modelů. Tabulky s prefixem reportsApp (název aplikace v Djangu) jsou vytvořeny na základě navržených modelů.



Obr. 17 Schéma databáze

B Diagram tříd

Na následujícím obrázku je diagram tříd, na základě kterého byly navrženy modely v informačním systému.



Obr. 18 Diagram tříd