VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

SYSTÉM PRO AUTOMATIZACI MARKETINGOVÝCH KAMPANÍ

SYSTEM FOR MARKETING CAMPAIGNS AUTOMATION

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **TOMÁŠ KUNICKIJ**
AUTHOR

VEDOUCÍ PRÁCE **RADEK BURGET, Ing. Ph.D.**
SUPERVISOR

BRNO 2020

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav informačních systémů (UIFS) Akademický rok 2019/2020

Zadání bakalářské práce

Student: **Kunickij Tomáš**
Program: Informační technologie
Název: **Systém pro automatizaci marketingových kampaní**
System for Marketing Campaigns Automation
Kategorie: Web

Zadání:

1. Seznamte se s aplikačním rozhraním marketingových služeb PPC. Zaměřte se na Google Ads, Seznam Sklik a další.
2. Prostudujte technologie pro tvorbu webových aplikací na platformě Python.
3. Navrhněte architekturu aplikace pro správu skriptů automatizujících PPC kampaně.
4. Implementujte navrženou aplikaci.
5. Proveďte testování vytvořené aplikace v reálném prostředí.
6. Zhodnoťte dosažené výsledky.

Literatura:

- Barry, P.: Head-First Python, 2nd edition, O'Reilly, 2016

Pro udělení zápočtu za první semestr je požadováno:

- Body 1 až 3

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Burget Radek, Ing., Ph.D.**
Vedoucí ústavu: Kolář Dušan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2019
Datum odevzdání: 28. května 2020
Datum schválení: 10. prosince 2019

Abstrakt

Tato práce se zabývá vývojem webové aplikace, která má za cíl umožnit PPC specialistům automatizovat a urychlit jejich práci. Aplikace umožní těmto marketérům ovládat na míru předem napsané skripty v Pythonu, které za ně vykonají určitou část práce. Jedná se o webovou aplikaci postavenou pomocí Pythonu a frameworku zvaný Django. Důležitou roli hraje také Celery pro možnost asynchronního zpracování všech skriptů. Systém umožní uživateli, který spravuje PPC účet spouštět a plánovat dodané skripty a tím zefektivní svoji práci.

Abstract

This thesis deals with the development of a web application that aims to enable PPC specialists to automate and make their work more efficient. The application will allow these marketers to control tailored pre-written Python scripts to do some of the work for them. The web application is built by Python and a framework called Django. Moreover, Celery also is important here as it enables the scripts to run asynchronously. The system will allow the user who manages the PPC account to run and schedule the scripts and thus make their work more efficient.

Klíčová slova

webová aplikace, python, django, celery, automatizace, PPC, online marketing, Google Ads, Seznam Sklik, Facebook, API

Keywords

web aplication, python, django, celery, automation, PPC, online marketing, Google Ads, Seznam Sklik, Facebook, API

Citace

KUNICKIJ, Tomáš. *Systém pro automatizaci marketingových kampaní*. Brno, 2020. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Radek Burget, Ing. Ph.D.

Systém pro automatizaci marketingových kampaní

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Radka Burgeta, Ing. Ph.D. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Tomáš Kunickij
15. května 2020

Poděkování

Děkuji panu Radku Burgetovi, Ing. Ph.D. za cenné rady a odborný dohled a vedení mé bakalářské práce. Děkuji také firmě RobertNemec.com, s. r. o., díky které jsem dostal myšlenku vytvořit tuto aplikaci.

Obsah

Kapitola 1

Úvod

Představte si, že jste PPC specialista, tedy člověk, který spravuje a tvoří online kampaně pro své klienty na Internetu. Máte za cíl přivést svému klientovi co nejvíce zákazníků, tedy zvýšit obrat za co nejmenší náklady. Využíváte reklamní platformy Google Ads, Seznam Sklik a Facebook. Tyto platformy nabízejí velké množství možností, jak dosáhnout vašeho cíle. Ovšem existují činnosti, které jako online marketér děláte stále dokola jako rutinu. Tyto činnosti lze automatizovat pomocí skriptů, tím eliminujete chyby a získáte čas navíc, který můžete investovat lépe. Například do činností, které posunou vašeho klienta dále, než tato rutinní, ovšem nutná činnost.

Automatizovat pomocí skriptů přímo uvnitř platformy aktuálně umožňuje pouze Google Ads a to pomocí Javascriptu. Ovšem v tomto případě jste omezení pouze na knihovny, které Google Ads dodává. To znamená, že ztrácíte potenciál psát komplexnější skripty. Navíc vám stále zůstal nástroj Sklik od firmy Seznam a Facebook. Tyto nástroje automatizovat pomocí skriptů přímo uvnitř platformy neumějí.

Všechny tyto platformy poskytují své API, přes které je možné kompletně spravovat celý účet inzerenta. Je tedy zřejmé, že větší potenciál má vytvoření vlastního skriptu, který bude komunikovat s platformami pomocí jejich API, což umožní automatizovat vše.

Tyto skripty píši v Pythonu. Skripty jsou psané na míru pro každého klienta a pro konkrétní činnost, kterou je třeba automatizovat. Dále bylo třeba vymyslet, jak tyto skripty předat marketérům, kteří je budou využívat. Od začátku jejich vývoje bylo zřejmé, že nebude vhodné skripty instalovat marketérům lokálně na jejich počítače. Důvody se dozvíte dále v této bakalářské práci. Další varianta je tedy spouštět skripty na serveru přes nějakou aplikaci. Aplikací, které umí spustit skripty je více, ovšem žádná není určená pro marketéry, kteří nerozumí programování.

Cílem této bakalářské práce je tedy vyvinout webovou aplikaci, skrze kterou mohou PPC specialisti využívat tyto skripty a automatizovat tak jejich práci.

Kapitola 2

Problematika automatizace reklamních online systémů

V této kapitole se dozvíte, jak lze inzerovat na internetu. Jaké jsou výhody a nevýhody této inzerce. Zjistíte, že tuto činnost je možné automatizovat, tedy zefektivnit a urychlit. Také zjistíte, co se za touto automatizací skrývá. Že nejde o nic jiného něž o skripty, které automatizují dílčí činnost a rutiny, které se v reklamních platformách nastavují manuálně. Na konci této kapitoly si stanovíme, jak by měla vypadat webová aplikace, která umožní online marketérům tyto skripty spouštět bez znalosti programování.

2.1 Co je to Pay Per Click (PPC) reklama?

V této části se dozvíte, co to Pay Per Click reklama je. K čemu slouží a také, které PPC nástroje se u nás v České republice využívají. Tedy které platformy nám umožní tato aplikace automatizovat.

2.1.1 Úvod do PPC reklamy

PPC je zkratka z anglického pay-per-click, což znamená platba za kliknutí. Je to tedy forma inzerce kdy vy, jako inzerent většinou zaplatíte za to, že nějaký uživatel Internetu klikl na vaši reklamu. Tím se liší od klasických forem inzerce, kde většinou platíte za počet shlédnutí (televize, tisk a jiné). Jedná se o jednu z nejúčinějších forem inzerce, jelikož zaplatíte opravdu pouze za relevantního zákazníka, který může mít o vaši službu nebo produkt skutečný zájem. Ovšem musíte tuto reklamu správně nastavit, jelikož v opačném případě budete vaše prostředky utrácet zbytečně. Pouhé kliknutí na reklamu samozřejmě není většinou vašim cílem, pokud nechcete zrovna budovat povědomí o značce. Častější situace je ta, kdy chcete, aby uživatel následně nakoupil váš produkt nebo službu.

V České republice jsou tři hlavní PPC systémy, které všichni nepřímo znáte, pouze nevíte, co se za nimi skrývá. Jejich reklamy na internetu potkáváte totiž každý den. Jedná se o Google Ads[1], Seznam Sklik[2] a také o inzerci na Facebooku[3].

[1] https://ads.google.com/
[2] https://www.sklik.cz/
[3] https://www.facebook.com/

Jedna z největších výhod a zároveň důvod, proč je tato forma inzerce nejúčinější je možnost přesného cílení reklamy. Jako inzerent máte možnost zobrazovat reklamu přesně těm uživatelům, kteří se o vaše služby nebo produkt zajímají.

PPC reklama se dělí na dva druhy. A to na reklamu

- ve vyhledávání
- v obsahové síti

Reklamu ve vyhledávací síti naleznete v Google Ads a Seznam Skliku. Tyto reklamy jsou pouze textové a spouští se v případě, kdy zadáte do vyhledávacího pole (ať už v Googlu[13] nebo Seznamu[18]) nějaký vyhledávací dotaz. Tímto dotazem může být například „značkové košile“. Představme si, že vy jako inzerent prodáváte oblečení. Prodáváte také košile a tyto košile máte ve dvou variantách. Levné a dražší. Z analýzy klíčových slov, tedy z vašeho výzkumu, jak lidé ve vašem odvětví zboží vyhledávají jste zjistili, že pokud někdo zadá dotaz „značkové košile“, je vhodné uživateli zobrazit reklamu přímo na vaše dražší košile.

Reklamy ve vyhledávací síti tedy pracují na principu shody klíčových slov s dotazem od uživatele. Jako inzerent máte možnost zadat klíčová slova, jejich kombinace, tvary nebo určité fráze, na které se vaše reklama zobrazí. Uživatel vyhledávače poté zadá nějaký dotaz a algoritmy Googlu nebo Seznamu dotaz porovnají s vašimi klíčovými slovy podle nastavených pravidel shody. Jelikož na daný klíčový výraz neinzerujete většinou sami, následuje aukce. Jde o klasickou aukci kdy vyhrává ten, kdo nabízí více peněz za dané klíčové slovo dané platformě. Tato placená reklama se zobrazuje na horních pozicích ve vyhledávání.

Druhý typ PPC reklamy je reklama v obsahové síti. Toto jsou různé bannery nebo kombinace textů s obrázky, které potkáváte během toho, když procházíte internet. Například někde zahlédnete banner na nějaký zájezd do Francie. Obsahová reklama ze síťe Google se zobrazuje na největším počtu různých webů. Dále následuje Seznam Sklik, který sice své reklamy nezobrazuje na tolika webech, nicméně zahrnuje některé české podstatné weby, jako jsou například novinky.cz[16]. Třetí platforma zobrazuje obsahovou reklamu pouze v rámci této platformy. Jedná se o Facebook. Reklama zde se zobrazuje například formou sponzorovaných příspěvků nebo ve stories. Důležité je zmínit, že pod Facebook spadá také Instagram. Pomocí Facebooku můžete tedy inzerovat také tam.

Velice podstatné je znát vaše zákazníky a vědět, na jaké publikum reklamu chcete zacílit. Cílení reklam dělím na 2 typy. A těmi jsou

- akvizice
- remarketing

Akviziční kampaně jsou kampaně, během kterých oslovujete nové zákazníky. Představme si, že prodáváme longboardy. Ze statistiky našich prodejů víme, že nejvíce zákazníků máme ve věku od 16 do 25 let, jsou to muži i ženy. Naši obsahovou reklamu zacílíme tedy na uživatele ve věku od 16 do 25, na obě pohlaví. Abychom vyloučili publikum (uživatelé na internetu, kterým se reklama zobrazí), které se o sport vůbec nezajímá, přidáme podmínku, že uživatel musí mít zájem o sport. Ano, tyto platformy o vás sbírají mnoho dat a na základě nich odhadují vaše koníčky nebo dokonce přijmy domácností.

Druhý typ kampaní je remarketing. Pokud spouštíte remarketingovou kampaň, již neoslovujete „náhodné“ uživatele internetu. Při remarketingu oslavujete uživatele, kteří již o váš produkt nebo službu projevili nějaký zájem[14]. Můžete tedy zacílit reklamu na všechny uživatele, kteří si u vás na webu vybírali nějaké produkty, ale svůj nákup nedokončili. Jedny z nejefektivnějších remarketingových kampaní jsou ty, které cílí na uživatele, kteří si přidali nějaký produkt to košíku, ale nenakoupili. Reklama cílí poté konkrétně na ně s konkrétním zbožím, které měl potenciální zákazník v košíku.

Pokud tedy kampaň špatně nastavíte, např. tím, že budete cílit na špatné publikum, budete zbytečně zvyšovat své náklady, utrácet více a více peněz a přitom zákazníci nakupovat nebudou. Jelikož necílíte na zákazníky, kteří o vás mají zájem. Přesné cílení tedy zvyšuje pravděpodobnost nákupu a snižuje náklady na samotnou reklamu.

Pokud PPC srovnáme s klasickým bannerem, například u dálnice, můžeme zjistit následující. Banner u silnice vidí všichni, kteří projíždí kolem, tedy pokud si jej všimnou. V tomto případě nemáte možnost účinnost tohoto banneru řádně změřit. Zatímco online banner můžete zobrazit přesně těm uživatelům, kteří mají o inzerovanou službu nebo produkt nejspíše zájem. A současně můžete sledovat, jaký mají výkon, tedy jestli někdo díky tomuto konkrétnímu banneru nakoupil, nebo se jen podíval na e-shop. Díky PPC reklamě jste schopní lidem nabízet produkty zrovna ve chvíli, kdy o ně mají zájem.

Díky tomuto přirovnání online a offline inzerce můžeme pozorovat, že jedna z největších výhod PPC je možnost vyhodnocování, tedy zobrazování detailních statistik. Na základě těchto statistik můžete rozhodovat, jaké strategie u vás fungují a jaké ne.

Další velká výhoda PPC kampaní je ta, že je možné je kdykoliv spustit a kdykoliv vypnout, je to otázka několika málo sekund. Pokud tedy zjistíte, že nějaká kampaň je pro vás nákladná a nepřináší zisk, vypnete ji a tak zamezíte dalším ztrátám.

Kampaně v PPC je také možné snadno měnit. Pokud zjistíte, že vám nějaký konkrétní banner nepřináší konverze, což může být například nákup, jednoduše jej vyměníte za jiný. Pokud zákazník udělá konverzi znamená to, že splnil cíl který vy požadujete, což může být například nákup zboží.

V PPC systémech není reklamní prostor nikdy vyprodán. Zobrazování reklam funguje na principu aukce. Jednoduše, kdo nabídne víc, bude se zobrazovat. Toto tvrzení není ovšem úplně přesné, jelikož platformy se snaží zobrazovat co nejrelevantnější reklamy a tak jsou kvalitní reklamy levnější, než reklamy špatné. Díky tomuto se můžete zobrazovat více, než vaše konkurence a platit méně. Špatná reklama může být reklama s nižší relevancí nebo například se špatnou landing page[4].

Důležité je také zmínit, jaké jsou nevýhody PPC. Aby tato inzerce byla skutečně efektivní, přinášela vám zisky za co nejmenší náklady, je její nastavení složité. To je důvod, proč si řada firem na tuto činnost najímá reklamní agentury. Tato bakalářká práce se zabývá vývojem webové aplikace pro marketéry, kterým bude díky této aplikaci umožněno spouštět skripty, které usnadňují určité PPC úkony. Pokud máte větší eshop, je správa PPC

[4]Landing page je stránka, která se uživateli zobrazí po kliknutí na reklamu.

účtů časově velice náročná, což je samozřejmě nevýhoda. Ovšem díky této aplikaci PPC marketérům umožníme jejich práci zefektivnit.

Každá reklamní platforma má své publikum a vy jako inzerent musíte vědět, kde je pro vás nejvýhodnější inzerovat. Kde tedy inzerovat, pukud potřebujete získat mladší nebo naopak starší zákazníky?

2.1.2 Google Ads

Google Ads je téměř celosvětově nejrozšířenější PPC platforma. K čemu slouží již víte. Dříve se služba jmenovala Google AdWords. V roce 2020 platforma existuje již 20 let od svého vzniku[1]. Na této platformě dokážete zasáhnout publikum všech věkových kategorií.

2.1.3 Seznam Sklik

Sklik je český PPC nástroj, který zobrazuje reklamy pouze na českých webech. Je vhodnější pro získávání spíše staršího publika, ale nemusí to být pravidlem.

2.1.4 Facebook Ads

Pomocí Facebook Ads je možné vytvářet pouze obsahové kampaně, které se budou zobrazovat ve Facebooku a Instagaramu. Díky této platformě máte možnost získat především mladší publikum zákazníků.

2.2 Možnosti automatizace přímo uvnitř platforem

Vašim cílem je vždy být lepší, než konkurence. Chcete z platforem získat co nejvíce, získat nové zákazníky, zvýšit zisk a obrat. Čím větší máte eshop, tím náročnější práce v PPC účtech je. Automatizace umožňuje, aby se pravidelná rutina, kterou je třeba dělat na denní, týdenní nebo také měsíční bázi dělala sama. PPC specialista se tím pádem může soustředit na důležitější věci v PPC účtech.

PPC platformy umožňují tvořit mnoho druhů reklam. Některé je třeba vytvořit ručně a jiné je možné generovat z feedu. Tyto reklamy tvořené z feedu jsou vždy aktuální, pokud je aktuální také feed. Může to být například inzerce konkrétních produktů na eshopu. Pokud produkt eshop vyprodá, nepošle jej do feedu a reklamní platforma tento produkt přestane izerovat.

Nejjednodušší možností automatizace jsou automatická pravidla. Ta umožňují nastavit jednoduché podmínky jako spouštěče a následné akce, které se mají vykonat. Tato možnost je přímo v uživatelském rozhraní platformy. Automatická pravidla podporuje pouze Google Ads a Seznam Sklik.

Příklad automatického pravidla může být.

- Sniž cenu za proklik o 30% u klíčových slov bez konverze za poslední měsíc.

- Zvyš cenu za proklik u brandové[5] reklamy, pokud je její průměrná pozice je horší, než určitá hranice.

Tato pravidla jsou velký pomocník, ovšem jsou omezena pouze na to, co dané rozhraní dokáže nabídnout.

2.2.1 Automatizace skripty

Zajímavější je možnost automatizace přes skripty. Google Ads jako jedinná platforma umožňuje napsat skript v JavaScriptu, který můžete spouštět přímo v platformě[15]. Rozhraní vám umožňuje jednoduše pracovat s ADS API, nebo také s Google Analytics[6] API. Není třeba řešit složité přihlašování a zabezpečení, jelikož pro spuštění skriptů musí být uživatel do účtu na webu přímo přihlášen.

JavaScriptový kód v Google Ads tedy umožňuje dělat pokročilejší automatizace v této platformě. Nevýhoda tohoto je již nutná znalost základů programování a API Google Ads, zatímco automatická pravidla nastavíte i bez těchto znalostí. V praxi marketéři většinou skripty přejímají a zvládnou pouze drobné úpravy. To znamená, že skripty pouze spouští.

Pokročilejší prací s PPC účtem může být např. pozastavení kampaní, které přesáhly v daném dni svůj denní rozpočet. Nebo také projití všech landing pages[7] v účtu a následné upozornění emailem, kolik z nich aktuálně hlásí chybový kód 404. V tomto případě není dobré inzerovat produkt, na který se nelze prokliknout.

Google Ads skripty jsou nejběžnější a nejvíce používanou formou automatizace PPC účtů. Ovšem pokud bychom chtěli spouštět ještě pokročilejší skripty, je zde ta nevýhoda, že jsme omezení pouze na knihovny, které Google Ads dodává.

Pokud bychom chtěli automatizovat díky Google Ads také Sklik a Facebook, můžeme se v tomto JavaScriptovém kódu připojit na API Skliku a Facebooku. Ovšem nevýhoda je, že Google Ads aktuálně umožňuje, aby skript běžel maximálně 30 minut[15]. Pokud tedy PPC marketér řídí velké účty, skript nezvládne v tomto inzervalu projít celý účet a ještě k tomu na třech platformách. Navíc v případě pokročilejších úprav jste stále omezení na knihovny v Google Ads.

Skripty přímo v Google Ads tedy nabízí ještě více možností, než automatická pravidla. Nicméně pokud bychom chtěli pracovat např. s grafikou, JavaScript v Google Ads nám již nestačí. Tuto situaci je třeba řešit již externími skripty mimo Ads.

2.3 Možnosti automatizace přes API nebo import

Nejpokročilejší forma automatizace všech platforem, tedy Google Ads, Seznam Skliku a Facebooku je správa kampaní a celého účtu přes jejich API. Všechny tři platformy API poskytují.

[5] Brandová reklama je reklama, která se spouští pokud uživatel zadá přímo název dané značky, neboli anglicky brandu.

[6] Google Analytics je nástroj pro sledování činnosti uživatelů na webu.

[7] Landing page je stránka, která se uživateli zobrazí po kliknutí na reklamu.

Pro psaní těchto skriptů je vhodné zvolit univerzální jazyk, který umožní nejen připojit se přes API k platformám, ale například také umožní pracovat s grafikou a tím automatizovat tvorbu bannerů, dále procházet web za účelem parsování dat a další. Jazyk musí mít univerzální potenciál.

Práce s grafikou a parsování webu je pro skripty podstatná, aby skript mohl například získávat obrázky z webu inzerenta. Následně je upraví podle specifikací jaké dané platformy vyžadují a použije v reklamách.

Z tohoto důvodu byl pro tvorbu těchto skriptů zvolen vysokoúrovňový skriptovací programovací jazyk Python, který toto vše umožňuje. Cílem této bakalářské práce je vyvinout webovou aplikaci právě pro správu těchto skriptů tak, aby jej marketéři mohli jednoduše využívat.

2.3.1 Import dat versus práce s API

Při tvorbě nových kampaní a reklam existují dvě možnosti automatické tvorby.

- Tvorba kampaně přímo přes API.
- Import ZIP souboru s grafikou a CSV s informacemi o struktuře kampaně.

První možnost, tedy připojení do API platformy má své výhody a nevýhody.

Mezi výhody patří rychlost, stačí spustit skript, ten přes API vytvoří kampaně a máme vše hotovo. Mezi nevýhody patří nutnost detailního porozumění API a nutnost řešit autentizaci při připojení.

Druhá možnost vychází z možnosti importovat do platforem ZIP soubor, který dle specifikace každé platformy popisuje pomocí CSV souboru strukturu importovaných kampaní[17]. Tento ZIP soubor také obsahuje grafické podklady, které se použijí při tvorbě reklamních inzerátů.

Skripty, které vzniklá aplikace umožní spouštět používají oba dva přístupy. Pro tvorbu nových kampaní se používá import ZIP souborem z důvodu snadnějších změn. Je snadnější změnit strukturu generovaného CSV, než aktualizovat API volání. Pro marketéra je to také přehlednější řešení, jelikož si má možnost CSV otevřít a ověřit si, jaká data se do platformy importují.

Práci s API skripty využívají v případě úprav kampaní. Například pokud nějaká kampaň obsahuje typ inzerátů, který se nedá propojit s feedem, musí být tedy ručně marketérem spouštěn a pozastavován. Pokud tyto reklamy inzerují produkty z webu, skript může projít všechna cílová URL, ověřit zdali je produkt skladem z webu a pokud není, inzerci pozastavit.

2.3.2 Google Ads API

Aplikační programovací rozhraní (API) Google Ads je určeno pro vývojáře, kteří pracují pro velké inzerenty[11]. Tito inzerenti spravují velký počet účtů nebo mnoho kampaní. Pro připojení do API stačí mít aktivní účet Google Ads. Dále je třeba se zaregistrovat jako

vývojář a následně získat přístup do centra Google Ads API. V tomto centru je možné spravovat klíč, který identifikuje vaši aktivitu skrze Ads API[12]. Skripty, které aplikace umožní spouštět využívají pro připojení do tohoto API Python knihovnu s názvem *googleads*.

2.3.3 Seznam Sklik API - Drak

Sklik API je rozhraní pro vývojáře, kteří chtějí automaticky spravovat inzerci na platformě Sklik. API je založeno na XML-RPC protokolu nebo JSON formátu[19]. API umožňuje vytvářet a spravovat kampaně, reklamní sestavy, klíčová slova, reklamy, statistiky a jiné. Aktuální verze se nazývá Drak.

2.3.4 Facebook Marketing API

Pro připojení do Facebook reklamního účtu se využívá tzv. Facebook Marketing API. Pro jeho používání je nutné vytvořit ve Facebook reklamním účtu novou aplikaci, díky které obdržíte přístupový token[10].

2.4 Požadavky na systém pro automatizaci marketingových kampaní

Externí skripty psané v tomto případě v Pythonu mají v oblasti automatizace PPC velký potenciál. Tyto skripty je většinou potřeba psát na míru pro každého klienta, jelikož každý klient má jiný web, jiné požadavky a jinou strukturu reklamního účtu. U každého klienta je tedy třeba nejprve nalézt, co u něj má smysl automatizovat.

Jakmile máme k dispozici samotný funkční skript, je třeba vyřešit jeho distribuci. Skript může používat např. majitel firmy nebo marketér, který spravuje PPC účet pro svého klienta. Tento člověk má na starosti chod PPC účtu a je třeba předpokládat, že nemá znalosti programování.

Z tohoto důvodu není vhodné marketérovi předat pouze samotný skript, jelikož by musel řešit následující záležitosti, ve kterých často nemá dostatečné znalosti.

- instalace interpretu Pythonu
- instalace knihoven
- práce s verzovacím systémem
- specifika kódu podle operačního systému

V tomto případě by si marketér musel instalovat interpret Pythonu a také řešit samotné knihovny, což není reálné. Cílem skriptu je zefektivnit práci i pro člověka, který neumí programovat. Aktualizace skriptu by byla možná řešit přes GIT, nicméně nastává stejná situace. Je potřeba marketérům práci ulehčit a práce s verzovacím systémem je pro ně naopak zátěž. Další nevýhoda tohoto řešení by mohla být ta, že pokud marketér, tedy člověk, který ve většině případů neumí programovat, změní část kódu. Potom nastane situace, kdy skript nebude fungovat správně a marketér nebude vědět proč.

Podstatná nevýhoda tohoto řešení je také ta, že lokální řešení neumožní pravidelné plánování běhů skriptu.

Z tohoto důvodu je tedy vhodnější skripty spouštět z webové aplikace. Ideální je aplikace, která marketérovi umožní následující.

- Přihlásit se pod svým jménem
- Strukturovaně zobrazit přehled skriptů, ke kterým má marketér přístup
- Možnost ihned spustit daný skript
- Sledovat aktuální výstup (log) skriptu
- Zobrazit přehledně historii běhů skriptu s vizualizací, jak doběhly
- Možnost stažení generovaných souborů pro následný import do platforem
- Možnost plánovat pravidelný běh skriptu
- Celkový dashboard s důležitými informacemi o všech bezích skriptů
- Možnost upozorňovat emailem, pokud skript selže
- Zobrazit přehled všech planovaných běhů skriptů

2.5 Existující řešení pro spouštění skriptů na serveru

Existují webové aplikace, které umožňují spouštět Python skripty online. Nicméně žádná z nich nesplňuje všechny požadované vlastnosti. Existují různé CI nástroje, tedy nástroje na tzv. průběžnou intergraci, které umožní spouštět různé kódy na serveru. Ovšem všechny tyto aplikace jsou určené k průběžné integraci a jejich využívání k naším účelům by bylo nepraktické. Služby sice umožní skript spustit, ale neumí pracovat s přidělováním různých práv na úrovni skupin skriptů, neumí ukládat logy z výstupu ve formě, která bude uvedená dále. Také nezobrazí dashboard přesně s těmi informacemi, se kterými je potřeba. Samozřejmě i přidělování klientů v těchto systémech by bylo velice nepraktické, jelikož nic podobného aplikace nepodporují. Dále je problém to, že v těchto CI nástrojích má uživatel přístup k samotnému kódu, čemuž je potřeba v tomto případě zamezit, aby marketér nemohl omylem měnit kód, kterému nerozumí.

Kapitola 3

Návrh aplikace

Jádro aplikace se skládá ze dvou hlavních částí. První z nich se stará o samotné spouštění skriptů. Řeší tedy asynchronní zpracování, ovládání skriptů, automatické periodické spouštění, ukládání výsledků a další. Tato část aplikace také poskytuje API, pomocí kterého lze tyto skripty ovládat. Druhá část aplikace poskytuje uživatelské rozhraní, které umožní klientovi skripty pohodlně spravovat. Toto UI pracuje s právě již zmíněným API. Aplikace je tedy rozdělena do dvou hlavních celků.

- API aplikace, které ovládá samotné skripty
- Uživatelské rozhraní, které komunikuje s API

Toto rozdělení umožní v budoucnu ovládat skripty také bez UI voláním tohoto API externě.

3.1 REST API aplikace

REST API aplikace umožní přes tzv. endpointy ovládat samotné skripty. Všechny endpointy jsou identifikovaný pomocí URL. Endpointy jsou kontrolované, zdali uživatel, který k nim přistupuje je příhlášený a jestli má oprávnění daný endpoint využít. Uživatel tedy například není schopen spustit skript s ID, ke kterému nemá přístup.

Z důvodu přehlednosti se API nachází za endpoitem */api*. Dále kvůli možnému rozšíření funkcionalit v budoucnu se zde nachází uzel */scripts*, za kterým jsou veškeré endpoity, které souvisí s ovládáním skriptů. API komunikuje pomoci formátu JSON.

Přehled těchto endpointů společně s krátkým popisem naleznete v následující tabulce. Pod tabulkou uvádám některé zajímavé aspekty návrhu tohoto API.

/api/scripts/<int:id>/run	Spouští skript s daným ID.
/api/scripts/<int:id>/status	Zobrazuje aktuální informace o posledním běhu skriptu s daným ID.
/api/scripts/<int:id>/logs	Vrací informace o všech bězích skriptu s tímto ID.
/api/scripts/<int:id>/logs/<int:number>	Vrací informace o zvoleném počtu posledních běhů skriptu s tímto ID.
/api/scripts/<int:id>/scheduling/status	Vrací informace o periodickém spouštění skriptu s daným ID.
/api/scripts/<int:id>/scheduling/add/crontab/<int:idcrontab>	Umožňuje přidat ke skriptu s daným ID pravidlo o periodické spouštění na základě plánování konkrétních časů.
/api/scripts/<int:id>/scheduling/add/interval/<int:idinterval>	Umožňuje přidat ke skriptu s daným ID pravidlo o periodickém spouštění na základě plánování intervalem.
/api/scripts/<int:id>/scheduling/delete/<int:idsheduled>	Umožňuje skriptu s daným ID odebrat pravidlo o periodickém spouštění.
/api/scripts/<int:id>/exports/download/<str:filename>	Vrací infromace o tom, na jakých URL se nachází soubory s exporty pro skript s tímto ID.
/api/scripts/dashboard/brief	Vrací statistiky po dnes s informacemi, kolik skriptů jak doběhlo.
/api/scripts/dashboard/logs/all/<int:number>	Vrací informace o všech bězích všech skriptů.
/api/scripts/dashboard/logs/errs/<int:number>	Vrací informace o bězích všech skriptů, které skončily chybou.
/api/scripts/dashboard/numbers	Vrací číselné informace ke kolika objektům má uživatel přístup.
/api/scripts/scheduled	Vrací informace o tom, které skripty mají aktivované jaké periodické plánování.

Jak již bylo zmíněno, API komunikuje pomocí formátu JSON. Níže pro ukázku odpovědi API popisuji některé zajímavé detaily tohoto návrhu.

Endpoint s adresou */api/scripts/<int:id>/run* v případě úspěšného spuštění skriptu vrací následující JSON s kódem 200. Pracuje tedy s návratovými kódy, jako všechny ostatní endpointy.

```
1 {
2     "INFO": "script_started"
3 }
```

V případě neúspěchu uživatel dostane následující odpověď s kódem 406.

```
1 {
2     "INFO": "already_running"
3 }
```

Endpoint */api/scripts/<int:id>/status* je důležitý pro získávání aktuálních informací o stavu skriptu. Zde uvádím jeho možný výstup, jako ukázku struktury odpovědi. Jakým způsobem se data získávájí je uvedené dále v kapitole *Implementace*.

```
{
    "STATUS": "SUCCESS",
    "ID": "1cd74573-ce28-483c-a20b-958f0566d02b",
    "DATE_DONE": "2020-02-13 20:27:04",
    "RESULT": "null",
    "LOG": {
        "0": "Zde je prvni log tohoto skriptu.",
    },
    "EXPORTS": {
        "ZIPs": []
    }
}
```

Jednotlivé atributy znamenají následující.

STATUS obsahuje informaci o aktuálním stavu skriptu, zdali běží, doběhl úspěšně, neúspešně nebo čeká na spuštění.

ID obsahuje unikátní řetězec identifikující konkrétní běh skriptu.

DATE_DONE udává časovou informaci o tom, kdy daný skript doběhl.

RESULT udává výsledek, pokud skript nějaký vrací.

LOG udává veškerý log, tedy výstup skriptu pro uživatele.

EXPORTS obsahuje cestu k exportům, pokud byly nějaké vytvořeny.

Endpoint */api/scripts/<int:id>/scheduling/status* vrací informace o tom, zdali má tento skript nějaký záznam v plánovači. Pokud ano, tak jaký. Tento uzel umožní tedy zjistit například to, že tento skript bude běžet pravidelně každé tři hodiny. Plánování skriptu v aplikaci je dvojího druhu a to buď intervalem nebo časovou informací, kdy se má skript spouštět.

Endpointy */api/scripts/<int:id>/scheduling/add/crontab/<int:idcrontab>* a */api/scripts/<int:id>/scheduling/add/interval/<int:idinterval>* slouží k nastavení nového záznamu do plánovače. V případě využití *<idcrontab>*, je třeba použít ID *crontab* záznamu. Například *id* 30 znamená „Každý den v 20:00“. Tuto informaci naleznete při volání */api/scripts/<int:id>/scheduling/status*. V případě využití proměnné *<idinterval>* je třeba udat ID intervalu, které nalezneme ve stejném endpointu, jako v předchozím případě. Níže vidíte ukázku jedné možnosti plánování pomocí *crontab* s ID 30, která znamená „Každý den v 20:00“. Tato informace pochází právě z */api/scripts/<int:id>/scheduling/status*.

```
{
    "crontabs": [
        ...
        {
            "id": 30,
            "minute": "0",
            "hour": "20",
            "day_of_week": "*",
            "day_of_month": "*",
            "month_of_year": "*",
            "czech_desc": "Kazdy den v 20:00"
        },
        ...
    ]
}
```

Endpoint */api/scripts/<int:id>/scheduling/delete/<int:idsheduled>* umožní odstranit záznam z plánovače pomoci ID tohoto záznamu.
Informaci o tomto ID nalezneme v endpointu */api/scripts/<int:id>/scheduling/status*.

Endpoint */api/scripts/dashboard/brief* se týká celkového přehledu, tedy dashboardu. Vrací informace po dnech o tom, jaký počet skriptů daný den doběhl v pořádku a jaký s chybou. Tato data se potom v UI využívají k vykreslení grafu v dashboardu.

Endpoint */api/scripts/dashboard/logs/all/<int:number>* vrací informace o všech běžích skriptů, ke kterým má uživatel přístup. Počet vrácených běhů je třeba omezit v parametru *number*.

Endpoint */api/scripts/dashboard/logs/errs/<int:number>* vrací informace o všech běžích skriptů, ke kterým má uživatel přístup a které skončily chybou. Opět je třeba omezit jejich počet.

Endpoint */api/scripts/dashboard/numbers* vrací informace o tom, ke kolika skriptům má daný uživatel přístup. Dále ke kolika klientům a také ke kolika skupinám skriptů. Skupiny skriptů budou vysvětleny dále.

Endpoint */api/scripts/scheduled* vrací obecný přehled o tom, jaké skripty mají nějaký záznam v plánovači. Uživatel je tedy schopen vidět na jeden API dotaz, které skripty se samy spouští a kdy.

3.2 Uživatelské rozhraní

Je třeba navrhnout uživatelské rozhraní, které bude splňovat požadavky uvedené v kapitole *2.4*. Marketér musí bez dlouhého hledání snadně nalézt všechny prvky, které pro ovládání aplikace potřebuje.

Po spuštění aplikace ve webovém prohlížeči je uživatel vyzván k zadání přihlašovacího jména a hesla. Jelikož jsou skripty psané na míru, tento účet bude na míru také vytvořen.

Po přihlášení se uživatel dostane do uživatelského prostředí, které je rozděleno na dvě části. Na levé straně nalezneme menu a na pravé samotný obsah.

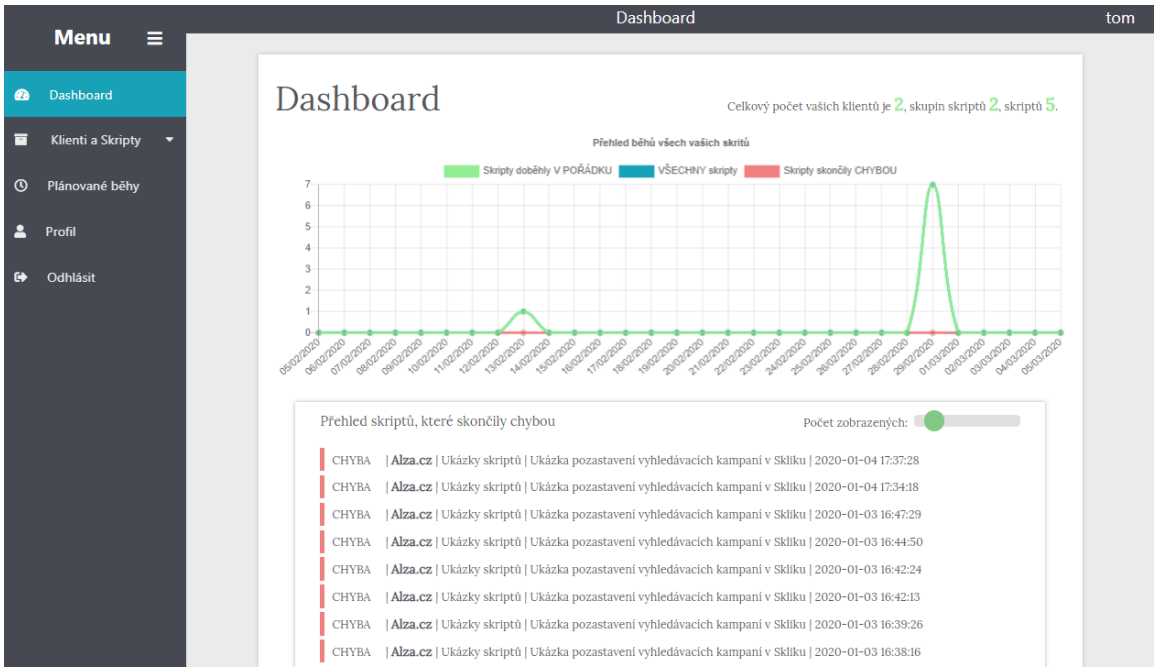
Celé uživatelské rozhraní je responzivní, pracuje tedy s různými rozlišeními. Prvky na pravé straně mají proměnlivou velikost a v případě rozlišení, které má menší šířku, je panel s menu na levé straně automaticky skryt a uživatel má možnost jej zobrazit a opět skrýt.

Menu aplikace obsahuje položky

- Dashboard
- Klienti a Skripty
- Plánované běhy
- Profil
- Odhlásit

Po přihlášení je třeba, aby uživatel viděl na první pohled, zdali nějaké skripty doběhly neúspěšně. Proto je uživatel ihned po přihlášení nasměrován do Dashboardu.

Dashbard má za cíl informovat uživatele na první pohled, zdali je vše v pořádku. Tedy, zdali všechny skripty doběhly bez chyby.



Obrázek 3.1: Na tomto obrázku vidíte, jak Dashboard vypadá.

V horní části se nachází graf, který přehledně znázorňuje na ose x data za poslední měsíc. Na ose y se nachází počet skriptů. Zelená barva udává, kolik skriptů daný den doběhlo v pořádku, červená barva udává, jaký počet skriptů doběhl s chybou. Modrá barva znázorňuje součet těchto skriptů, tedy kolik skriptů daný den běželo nebo stále běží.

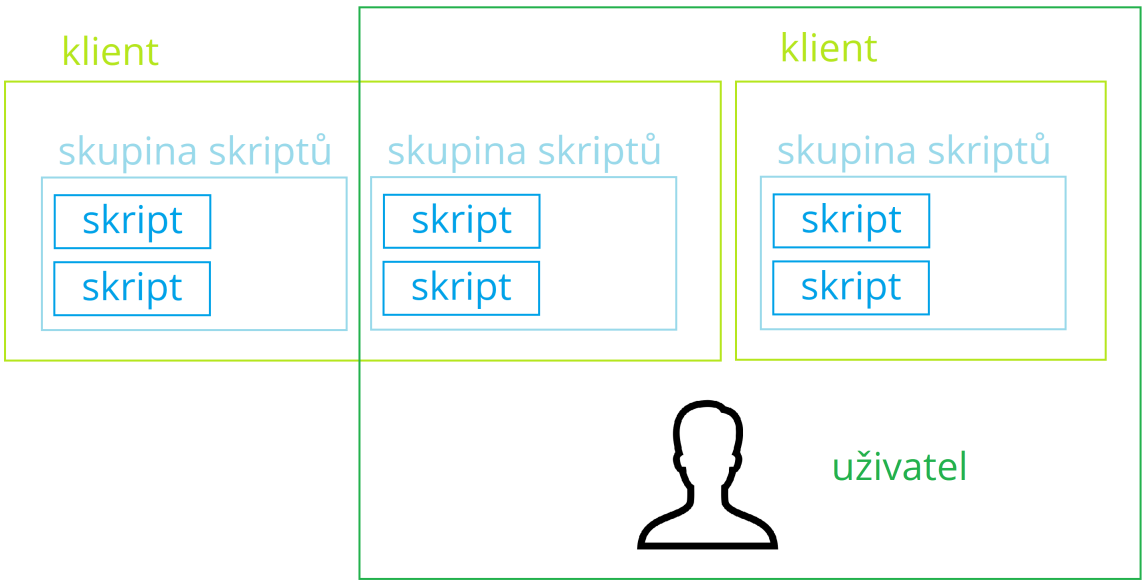
Pod tímto grafem se nachází dva přehledy. První se skripty, které jako poslední skončily chybou. Jejich počet si uživatel může nastavit. U každého záznamu je vizuálně znázorněno,

že se jedná o chybu, dále o jakého klienta se jedná, v jaké skupině skriptů daný skript je, název samotného skriptu a informace o tom, kdy skript doběhl. Je zde možné se prokliknout do podrobnějších informací, jako je například log nebo popis dané chyby.

Druhý obsahuje ty samé informace ovšem ne pouze se skripty, které skončily s chybou, ale se všemi běhy skriptů, ke kterým má daný uživatel přístup.

Nejdůležitější je pro uživatele záložka *Klienti a Skripty*. Zde se nachází to nejdůležitější, proč je tato aplikace vyvíjena.

Zde je důležité pochopit strukturu skriptů, skupin skriptů, klientů a udělení přístupu ke skupinám skriptů. K pochopení pomůže následující obrázek. Základ je samozřejmě samotný skript. Ovšem může nastat situace, kdy pro dokončení jednoho typu úkolu je třeba spustit více skriptů a mezi tím udělat v platformách ručně nějaké kroky. Z tohoto důvodu v aplikaci existuje tzv. *skupina skriptů*. Tato skupina obsahuje skripty, které spolu souvisejí. Pro lepší pochopení uvedu příklad. Marketér automatizuje tvorbu remarketingových reklam. V rámci vytvoření těchto reklam musí vytvořit v platformách remarketingová publika. Toto má na starosti první skript. Druhý skript vytvoří ZIP soubory, které marketér nahraje do systému. Tyto soubory vyžadují již při uploadu vytvořená publika v systémech. Tyto skripty je tedy třeba spustit po sobě a souvisejí spolu. Proto umístíme tyto skripty do stejné *skupiny skriptů*. Tato skupina skriptů automatizuje PPC činnost pro nějakého konkrétního klienta. Z tohoto důvodu je na této úrovni přiřazen klient.



Obrázek 3.2: Na tomto obrázku vidíte, jakým způsobem sposu souvisí skripty, skupiny skriptů, klienti a jakým způsobem se přidělují práva přístupu uživatelům.

Představme si tedy konkrétní případ. Máme skript *skript_A* a skript *skript_B*. Jelikož tyto skripty se doplňují a řeší automatizaci jedné činnosti nachází se ve stejné skupině *skupina_A*. Ke skupině s názvem *skupina_A* přiřadíme klienta s názvem *klient_A*. Dále je třeba, aby tyto skripty někdo využíval. Mějme tedy marketéra se jménem *marketer_A*.

Tomuto marketérovi nastavíme přístup ke skupině *skupina_A*. Přístup se řeší na této úrovni z toho důvodu, že skripty ve skupině spolu souvisejí. Marketér nikdy nemá mít tedy přístup pouze k nějakému skriptu ze skupiny. Systém tedy umožní, aby klient měl více skupin skriptů, ovšem marketér nemusí mít přístup ke všem skupinám u tohoto klienta. Každý marketér může mít na starosti jiné věci a potřebuje tedy jiné skupiny skriptů. V uživatelském rozhraní to tedy znamená, že po kliknutí na *Klienti a Skripty* se marketérovi zobrazí přehled všech klientů, u kterých má přístup do alespoň jedné skupiny skriptů. Marketér si tedy vybere klienta, kterého potřebuje. Dále se uživateli zobrazí přehled všech skupin skriptů, ke kterým má u daného klienta přístup spolu s krátkým popisem, k čemu daná skupina slouží.

Po vybrání dané skupiny vidí marketér samotné skripty, které do skupiny patří. Vidí také detailní popis toho, kdy jaký skript pro dokončení této činnosti spustit. Marketér si tedy vybere skript, který je třeba spustit jako první.

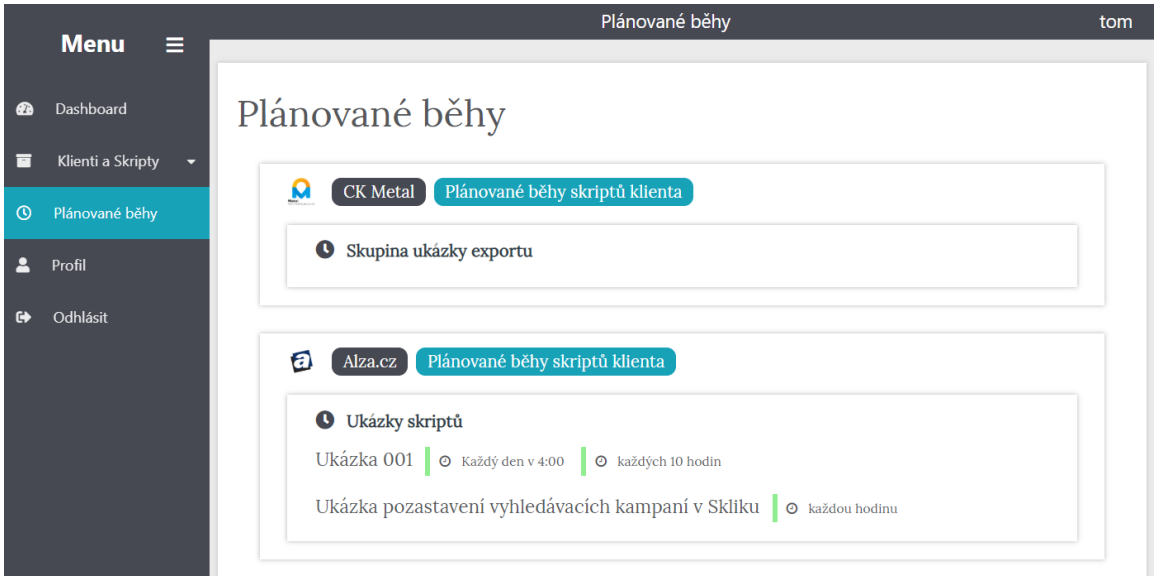
Nyní jsme v nejpodstatnější části aplikace. Části, kde marketér ovládá spouštění samotného skriptu. V horní části rozhraní vidíme název samotného skriptu. Vpravo vedle názvu vidíme vizuálně znázorněný stav skriptu, zdali jeho poslední běh doběhl úspěšně, neúspěšně nebo zdali skript nyní běží. Vedle tohoto stavu nalezneme tlačítko pro spuštění skriptu.

Uživatelské rozhraní je dále rozdělení do několika sekcí umístěných pod sebe. Jako první nalezneme sekci s popisem daného skriptu. Druhá je sekce a návodem, jak tento konkrétní skript používat. Jako třetí následuje sekce s logem. Zde uživatel vidí aktuální výstup skriptu. Skript zde informuje uživatele o tom, co se právě děje. Následuje sekce s exporty. Pokud skript generuje nějaký export, se kterým marketér následně dále pracuje, právě v této sekci je možnost daný soubor stáhnout. Pátá sekce zobrazuje informace o několika posledních bězích. Množství těchto běhů si může uživatel nastavit a data se pomocí volání API JavaScriptem aktualizují na daný počet. Vidíme zde informace, zdali skript doběhl v pořádku nebo ne. Každý záznam je možné otevřít pro získání konkrétnějších informací, jako je log nebo chybová hláška. Poslední sekce umožňuje uživateli nastavit pravidelný běh skriptu. Časové plánování je dvojího druhu, z tohoto důvodu jsou v uživatelském rozhraní v této sekci dvě podsekce. Jedna umožňuje nastavit interval, po kterém se skript znovu spouští, zatímco druhá umožňuje nastavit pevný čas, například každý den ve 2 hodiny ráno.

V případě spuštění skriptu skript pomocí tlačítka spustit, rozhraní vizualizuje běh skriptu. Tlačítko se změní z modré na oranžovou barvu a kromě odlišného textu uvnitř také během běhu skriptu obsahuje ikonku s animací běhu. Vlevo od tohoto tlačítka se nachází informační box, ve kterém uživatel po doběhnutí uvidí zdali skript skončil v pořádku nebo s chybou. Box je vhodně zeleně nebo červeně podbarvený. Během samotného běhu aplikace zobrazuje aktuální log, tedy zprávy o tom, co skript dělá.

Další sekce z menu s názvem *Plánované běhy* umožňuje uživateli zobrazit přehledně na jednom místě, jaké skripty využívají časové plánování. Toto je praktická funkce, jelikož nyní není třeba procházet skripty po jednom k získání této informace. Tento přehled je rozdělen dle klientů, dále dle skupin skriptů a nakonec podle skriptů samotných.

V části *Profil* má uživatel možnost zvolit si, zdali si přeje zasílat emaily, pokud nějaký skript selže, nebo ne. Dále má možnost změnit si přístupové heslo ke svému účtu.



Obrázek 3.3: Na tomto obrázku vidíte u jakého klienta je naplánovaný jaký skript. Je zde také vidět v jaké skupině skriptů tento skript je a také na kdy je naplánovaný.

3.3 Propojení uživatelského rozhraní a API

Uživatelské rozhraní při práci se skripty nepřistupuje ke skriptům napřímo, ale využívá API, které aplikace nabízí. Tím je oddělena logika spouštění skriptů od uživatelského rozhraní. Ostatní přístup k datům, který se netýká přímo spouštění skriptů probíhá napřímo skrze dotaz na databázi, tedy bez využití tohoto API.

Uživatelské rozhraní k API přistupuje díky JavaScriptu, ten zasílá na API dotazy.

API je možné do budoucna využívat i mimo tuto aplikaci, je tedy možné skript v této aplikaci spustit voláním API této aplikace aplikací jinou.

Více detailů o konkrétním propojení těchto dvou částí naleznete v kapitole *Implementace*.

Kapitola 4

Použité technologie

V této kapitole naleznete popsané technologie, které jsem při vývoji aplikace využil. Také zde naleznete důvody, proč byla daná technologie využita.

4.1 Python

Python je vysokoúrovňový skriptovací programovací jazyk, který podporuje různá programovací paradigmata jako je objektově orientované, imperativní, procedurální nebo funkcionální programování[20]. Je to univerzální jazyk, který se využívá v široké škále různých oblastí a proto je také velice oblíbený.

Podle mnoha průzkumů se Python drží na prvních pozicích, co se týče oblíbenosti[2]. Je také jeden z nejvíce doporučovaných jazyků, které se učit. Jeho univerzálnost, čistota kódu a snadné pochopení jsou důvody, proč jsou nejen skripty, ale také samotná aplikace napsané právě v tomto jazyce.

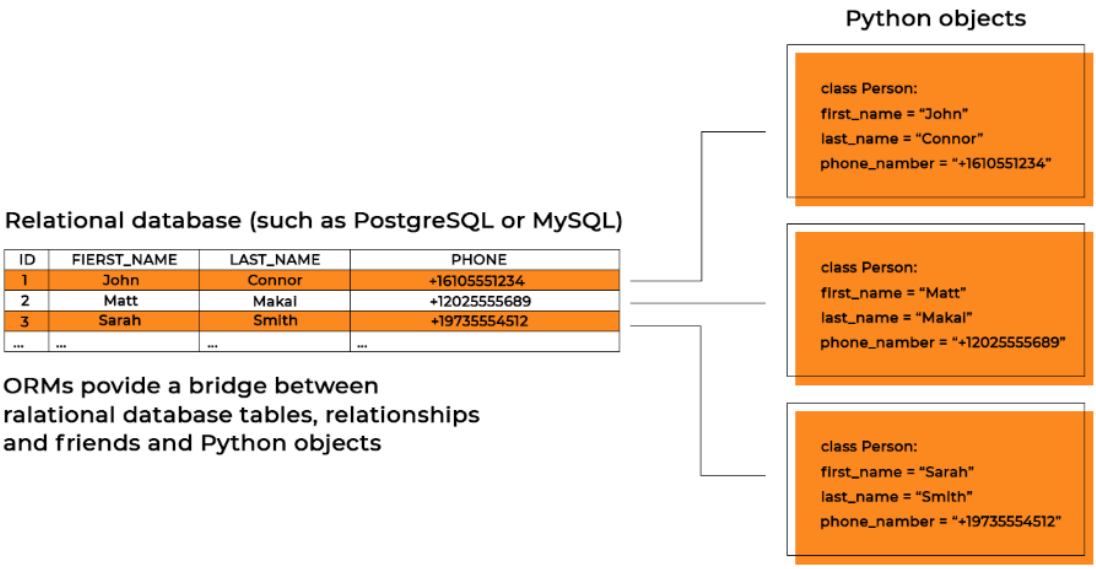
V současné době má Python dvě hlavní verze. Python 2 a Python 3. Samotné skripty, které aplikace spouští jsou psané v Pythonu verze 3.7.2, stejně jako samotná aplikace. Verze 2 se postupně přestává používat a čím dál více vývojářů se zaměřuje na verzi 3. To je tedy důvod, proč skripty i aplikace běží právě na této verzi.

4.2 Framework Django

Po volbě programovacího jazyka bylo nutné vybrat framework nebo knihovnu, pomocí které aplikaci vyvíjet. Frameworků nebo knihoven se nabízelo velké množství, jako je například CubicWeb, Dash, Django, Bottle, Falcon, Flask a další. Rozhodl jsem se zaměřit na dvě nejvíce populární řešení a tím je framework Django a knihovna Flask. Jak Django, tak i Flask mají velkou komunitní podporu a pomáhají vývojáři soustředit se na samotnou webovou aplikaci a usnadňují vývoj. Oba jsou určeny k vývoji aplikací, rozdíl je pouze v tom, jakým způsobem je aplikace postavená.

Django i Flask jsou open-souce. Jelikož je Django framework, nabízí větší funkcionalitu, než samotná Flask knihovna[3].

Django je možné napojit na relační databázi, jako je například SQLite, PostgreSQL, MySQL nebo Oracle. Poskytuje tzv. ORM (Object Relational Mapping), díky kterému přistupujeme k databázi skrze Django objekty. Databázi můžeme tedy v průběhu vývoje změnit, zatím co kód může zůstat stejný. ORM se automaticky postará o zapisování a čtení z jiné databáze.



Obrázek 4.1: Na tomto obrázku vidíte, jak spolu souvisí model ve frameworku Django a tabulka v relační databázi. Převzato z [21].

Jelikož většina aplikací vyžaduje možnost přihlášení a autorizace, Django poskytuje tuto funkcionalitu společně s možností správy těchto účtů. Flask poskytuje pouze sezení pomocí cookie, ale správu uživatelů, autentizaci a autorizaci si vývojář musí napsat sám.

Django poskytuje uživatelské rozhraní pro přístup k databázi. Jedná se tedy o admin panel, ve kterém je možné spravovat data vytvořené na základě modelů v aplikaci. Model je popis, jak má daná tabulka v databázi vypadat. Flask neposkytuje nic podobného.

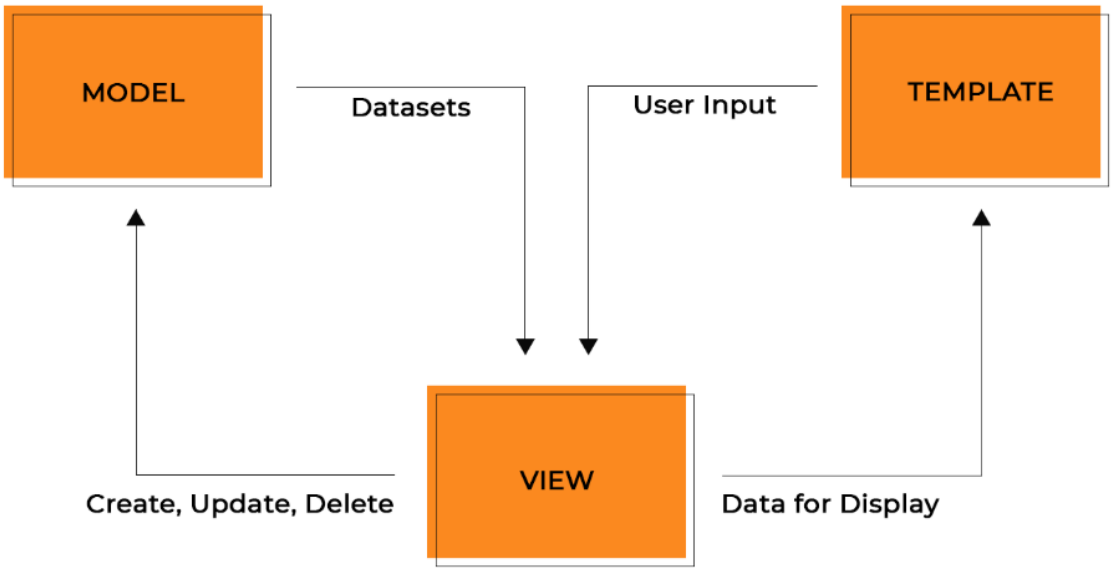
Dále je třeba zajistit, aby po zadání adresy do prohlížeče byla vrácena správná stránka. To zajišťuje routování. Při zadání URL Django páruje zadanou URL s URL v aplikaci a hledá, jakou funkci zodpovědnou za vykreslení dané stránky zavolá. Routování se nastavuje v každé aplikaci v souboru *urls.py*. Je možné tedy vložit novou aplikaci například za endpoint */blog* a daná aplikace sama vyřeší routování za tímto endpointem podle svého *urls.py*. Flask podporuje routování také.

Django umožňuje psát aplikace tak, aby se její části daly znovu využít. Proto je dobré projekt rozdělit do tzv. aplikací. Tyto aplikace fungují samostatně, mají vlastní modely (tedy tabulky v databázi), vlastní pravidla pro routování URL, vlastní funkce pro vykreslování stránky a další.

Jak bylo zmíněno, Django je plnohodnotný framework, řeší tedy také zabezpečení. Má implementované kódy, které zamezují běžným útokům jako je SQL injekce, cross-site scripting a další.

Django funguje na principu MVC (Model View Controller)[22], nicméně tento princip používá v mírně upravené verzi MVT (Model View Template). Jeho základní logikou je oddělení logiky od výstupu. Celá aplikace je rozdělena na komponenty 3 typů.

- Model (MVC) Django nazývá stejně, tedy i v MVT se jedná o **Model**. Model obsahuje logiku, jeho práce spočívá v přijetí parametru, vypočítání dat a jejich vrácení. Model neřeší podobu nebo formát dat na výstupu.
- Views (MVC) Django nazývá v MVT **Templates**, v překladu šablony. Šablona se stará o zobrazení výstupu uživateli, jedná se tedy o HTML šablonu rozšířenou o tagy, pomoci kterých Django propisuje data na výstup.
- Controllers (MVC) Django v MVT bohužel nazývá **Views**. View tedy v MVT neznamená HTML šablonu, ale komponentu, která propojuje právě Modely a Šablony. Jedná se tedy o onoho chybějícího prostředníka.



Obrázek 4.2: Na tomto obrázku vidíte model MVT graficky znázorněný. Převzato z [21].

Django používají velké projekty jako je například Bitbucket, Instagram, Pinterest, NASA a další. Z důvodu uvedených výše jsem si vybral právě tento framework.

4.3 Celery

Jelikož skripty mohou běžet několik minut, je klíčové, aby běžely asynchronně. Není žádoucí, aby uživatel po kliknutí na spuštění skriptu musel během jeho běhu čekat u zamrzlého okna, ze kterého bude mít pocit, že daná aplikace nefunguje.

Z tohoto důvodu bylo třeba vybrat knihovnu, které umožní asynchonní spouštění skriptů. Nejvíce používaná knihovna, která umožní detailní konfiguraci a také automatické spouštění skriptů v Pythonu podle plánovače se nazývá Celery[1].

Celery je systém, který implementuje asynchronní fronty úloh pro Python. Pro využívání Celery je nutné nastavit také tzv. brokera. Broker je služba, která zajišťuje zasílání zpráv o tom, jaký skript se má spustit. Celery podporuje následující 4 brokery

- RabbitMQ
- Redis
- Amazon SQS
- Apache Zookeeper

U Amazon SQS a Apache Zookeeper Celery nepodporuje možnost monitoringu a vzdáleného ovládání skriptů[4]. Reddis je využívá často, nicméně umožňuje méně nastavení, než RabbitMQ. RabbitMQ navíc umožňuje zasílání zpráv, které ukončují běh skriptů. Celery navíc v dokumentaci uvádí, že RabbitMQ je kompletní a stabilní broker, který je snadné nastavit. Z tohoto důvodu jsem vybral jako brokera právě RabbitMQ.

Celery pracuje s tzv. workery. Worker si z fronty vyzvedne danou úlohu, zpracuje ji a uloží případný výsledek. Samotná úloha není nic jiného, než funkce, která se má vykonat. V našem případě tato funkce volá skript, který si marketér přeje spustit. Po workerech většinou vyžadujeme spuštění náročnějších úloh, jako je například poslání emailu. Tedy úloh, které trvají déle. Samotné úlohy jsou potom uloženy v souboru *tasks.py*. Pro jejich zavolání pomoci Celery je nutné funkci zavolat s pomocí *delay()* metody následujícím způsobem.

```
1 from tasks import add
2 add.delay()
```

Celery umožňuje také ukládat výsledek úlohy, nicméně toto není výchozí chování. Pro ukládání výsledku je třeba nastavit backend, kam má Celery výsledky ukládat pomocí knihovny *django-celery-results*[2]. Konkrétní implementace je uvedená v kapitole Implementace.

Pro běh aplikace je tedy nutné spustit Celery server. Dále frontu pro zasílání zpráv do Celery, tedy RabbitMQ a samožřejmě také samotnou aplikaci. Beží tedy tři služby.

Jak již bylo zmíněno, Celery podporuje periodické úlohy, tedy automaticky a opakovaně spouštět úlohy. Tzv. *celery beat* je plánovač, který zašle brokerovi ve správný čas informaci o tom, že má být nějaká úloha spuštěna. Worker si následně úlohu vyzvedne a spustí ji. Výchozí nastavení funguje tak, že záznamy o tomto plánování jsou naprogramované v kódu, nicméně je možné také získávat záznamy z databáze, což je případ, který využívá naše aplikace. Je to z toho důvodu, že toto plánování se mění podle toho, kdy uživatel tento pravidelný běh skriptu nastaví.

[1] http://www.celeryproject.org/
[2] https://pypi.org/project/django-celery-results/

4.4 SQLite

SQLite je relační databáze, tedy databáze založená na tabulkách. Můžeme mít například tabulku s názvem uživatel, ve které budeme uchovávat informace o všech uživatelích, jako je například jejich jméno, email nebo heslo v zašifrované formě. Položky (tedy uživatelé) se ukládají jako jednotlivé řádky. Sloupce tabulky poté označují atributy jako je právě například jméno.

SQLite na rozdíl od databází jako je MySQL nebo PostgreSQL neběží jako služba. Jedná se pouze o knihovnu nástrojů, kterou některé jazyky mají zabudovanou. Každá databáze se ukládá jako soubor na disk. Tato databáze je tedy vhodná pro malé projekty. V naší aplikaci se využívá, ovšem v budoucnu není problém přejít na MySQL, jelikož stačí v Django konfiguraci připojit odlišnou databázi.

4.5 HTML a CSS

Front end aplikace je vykreslován pomocí HTML *(HyperText Markup Language)* šablon, které Django naplní daty a následně zašle do prohlížeče uživatele.

HTML je značkovací jazyk používaný pro vytváření webových stránek. Využívá značky (tzv. tagy) a jejich vlastnosti (atributy). Pomocí těchto značek určuje strukturu webu. Pro určení vzhledu se nyní využívá CSS.

CSS, tedy *Cascading Style Sheets* je moderní jazyk, který popisuje informace jako je barva, písmo, umístění prvku a další formátovací vlastnosti. Udává tedy vzhled webu. V praxi se většinou vytvoří několik CSS souborů, které udávají grafický styl nějakého webu a tento soubor se následně využívá ve všech HTML souborech.

4.6 JavaScript

HTML společně s CSS zajišťují statické zobrazování webu. I když dnes již CSS podporuje různé animace. Ovšem k dynamickým interakcím se stránkou je potřeba JavaScript.

JavaScript je multiplatformní, objektově orientovaný skriptovací jazyk. Běží na straně klienta a nejčastěji se využívá pro zobrazení interaktivních prvků na stránce jako jsou například tlačítka, animace, upozornění na chybějící prvky při vyplňování formulářů a další.

JavaScript může běžet také na straně serveru, nicméně toto v naší aplikaci není využito.

4.7 Bootstrap

Bootstrap je CSS knihovna, která umožňuje aplikovat CSS styly, které byly předem vytvořené a často se opakují. Bootstrap tedy usnadňuje design webu pomocí předdefinovaných stylů. Díky jeho popularitě dnes Bootstrap poskytuje také různé návrhářské šablony založené na HTML a CSS, ale také například různá rozšíření v JavaScriptu.

Kapitola 5

Implementace

Implementace byla provedena v jazyce Python 3.7.2 s využitím technologií popsaných výše v kapitole *Použité technologie*. V této kapitole bude popsána konkrétní implementace předchozího návrhu aplikace.

5.1 Struktura vytvořeného řešení

Jak jsem psal již dříve, Django pracuje s tzv. aplikacemi. Celý tento projekt se tedy skládá ze tří aplikací a těmi jsou

- aplikace API
- aplikace UI
- aplikace USERS

Každá z těchto aplikací je zodpovědná za část celkové aplikace. Aplikace *API* poskytuje funkcionalitu REST API. Aplikace *UI* se stará o uživatelské rozhraní. Aplikace *USERS* má na starost správu uživatelů.

Každá z těchto aplikací má následující adresářovou strukturu, přičemž jednotlivé aplikace mají některé části lehce odlišné. Části (soubory), které obsahuje každá z aplikací jsou následující

- admin.py
- models.py
- urls.py
- views.py

Soubor *admin.py* slouží v každé aplikaci k přidání tzv. *modelů* ze souboru *models.py* do admin části. To znamená, že pokud registrujeme daný *model* do admin *admin.site*, budeme moci ve webovém rozhraní v části pro administrátory vidět a spravovat tento model (model, je tabulka v databázi).

Soubor *models.py*[7] slouží k definování tabulek v databázi. Každá aplikace má své tabulky, se kterými dále pracuje. V modelu určujeme jaké sloupce jakých typů má daná tabulka mít. Django se poté postará o tvorbu tabulky v databázi, kterou používáme.

Soubor *urls.py*[8] slouží k routování cest v rámci dané aplikace. Zde tedy určujeme, že například po zadání *.../scripts/<int:id>/run* se spustí *view*, který má tuto URL na starosti. Tři tečky před touto částí URL značí, že může předcházet ještě nějaká cesta, která je definovaná ještě před touto aplikací.

Soubor *views.py*[9] slouží k propojení modelů a šablon. Získává tedy data z databáze a vrací je pomocí šablon uživateli. Je dobré oddělit logiku získávání dat z databaze do zvláštního souboru z důvodu častého dotazovaní na stejná data v různých *views*.

Mimo již tyto tři zmíněné adresáře projekt také obsahuje složku s názvem samotného projektu. Tato složka obsahuje celkové nastavení aplikace. Obsahuje vlastní soubory *urls.py*, *views.py*. Soubor *urls.py* obsahuje následující.

```
1 urlpatterns = [
2     path('admin/', admin.site.urls),
3     path('api/', include('api.urls')),
4     path('', include('ui.urls')),
5 ] + static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

Toto nastavení říká, že za veškeré URL za *admin/* je zodpovědný jiný *urls.py* soubor. Většinou se jedná o soubor v aplikaci, která je za daný blok zodpovědná. V tomto případě existuje Djangem definovaná aplikace *admin*, které říkáme, že za *admin/* může převzít kontrolu routování. Dále zde vidíme *api/*, tímto definujeme, že za veškeré operace za *api/* je zodpovědný soubor *urls.py* z aplikace *API*. Další řádek říká, že aplikace *UI*, tedy uživatelské rozhraní je dostupná přímo z rootu domény.

Ve složce s nastavením celého projektu najdeme také soubor *settings.py* a *celery.py*.

Soubor *settings.py*[6] obshuje důležitá nastavení celého projektu jako je

- přidání všech aplikací do projektu
- definování, kde se nachází šablony
- nastavení databáze
- nastavení časové zóny
- nastavení, kde se nachází statické soubory
- nastavení přístupu do emailu, ze kterého se zasílají zprávy
- nastavení celery
- a další

Soubor *celery.py* vytváří instanci Celery, se kterou se následně v projektu pracuje. Také zde nastavujeme, aby Celery automaticky hledalo úlohy (tasky), které má spouštět v souboru *tasks.py*.

Také se zde nachází složka *static*, do které se umísťují soubory se, kterými aplikace následně pracuje. Tato složka obsahuje dva podadresáře. Adresář *media* obsahuje veškeré soubory, které jsou veřejně přístupné i bez přihlášení. Jedná se například o loga klientů, faviconu nebo prvky uživatelského rozhraní. Zde není třeba omezovat práva přístupu. Druhá složka s názvem *private_media* obsahuje exporty pro marketéry. Každý skript zde má vlastní složku. Soubory ve složce *private_media* mají poté uživatelé možnost stáhnout u jednotlivých skriptů. Samozřejmě zde probíhá ve views kontrola, zdali má uživatel k danému souboru přístup. Není tedy možné neoprávněně stáhnout soubor pomocí úpravy URL.

5.2 Aplikace API

Tato aplikace má na starosti REST API, které poskytuje pomocí *Django REST Framework*[1]. Mimo zmíněné soubory *models.py*, *views.py* a *urls.py*, také obsahuje soubory *helpers.py*, *task_helpers.py* a *tasks.py*. Obsahuje také adresář *scripts*, který obsahuje veškeré skripty, které do aplikace dodáváme.

Soubor *urls.py*, směruje dané URL ke správným views. Jaké jsou jednotlivé endpointy API naleznete výše v návrhu aplikace.

5.2.1 Struktura databáze

Soubor *models.py* definuje pět různých modelů tedy tříd, ze kterých Django poté staví tabulky v databázi. Jedná se o

- TasksLogger
- ScriptLastTaskID
- Clients
- ScriptsGroup
- ScriptsInfo

Model *TasksLogger* slouží k zaznamenávání logů ze skriptů. Obsahuje proměnné (tedy sloupce v tabulce v databázi) *task_id*, *order*, *log_message*. Pokaždé, když některý ze skriptů zapisuje nějakou zprávu pro uživatele, uloží se to této tabulky ID dané úlohy ze Celery do sloupce *task_id*. Do sloupce *order* se uloží číselná hodnota s informací, kolikátá v pořadí tato zpráva pro danou úlohu je. A v poslední řadě se samozřejmě uloží samotná zpráva do *log_message*. Zápis probíhá pomocí funkce *log* ve třídě textitLogger, jejíž instanci dostává každý skript při spuštění.

[1] https://www.django-rest-framework.org/

Model *ScriptLastTaskID* uchovává pro každý skript jeho Celery *task_id* z posledního spuštění. Model používá tři sloupce. Jedná se o *script_id*, tedy ID skriptu, který spouštíme. Dále *last_celery_task_id* uchovává právě ID úlohy z Celery, které se uloží při spuštění skriptu. Jelikož nás zajímá, kdy daný skript byl spuštěn nebo ukončen existuje zde sloupec *updated_at*.

Model *Clients* uchovává jméno klienta v poli *client_name* a také jeho logo v *picture*. V případě vytvoření nového klienta se do loga přiřadí výchozí obrázek.

Model *ScriptsGroup* slouží k vytváření skupin skriptů, které spolu souvisejí například tím, že je třeba je spustit po sobě k dokončení daného úkolu. Tento model obsahuje sloupce *name_group*, který slouží k pojmenování dané skupiny. Dále *group_description* a *how_to_use* slouží k uchování popisu skupiny skriptů a k návodu, jak tuto skupinu používat. Samotná skupina skriptů je poté přiřazena nějakému klientovi pomocí *client*, jedná se o cizí klíč. Poslední sloupec s názvem *userAccess* slouží k nastavení práv přístupu k této skupině skriptů. Uživatel poté automaticky může přistoupit ke všem skriptům v této skupině.

Model *ScriptsInfo* definuje tabulku v databázi, ve které nalezneme informace o samotných skriptech. Najdeme zde sloupce *script_id*, který slouží k zaznamenání unikátního ID skriptu. Dále *name*, tedy název skriptu. Další informace nalezneme v *about_script* a *how_to_use*. Jsou to informace s popisem, k čemu skript slouží a s návodem, jak jej použít. Poslední sloupec s názvem *name_group* slouží jako cizí klíč k propojení se skupinou skriptů.

5.2.2 Views

Soubor *views.py* poskytuje views, které se spouští zadáním vhodné URL do prohlížeče. Tyto funkce propojují modely a šablony. Vrací uživateli tedy výstup do prohlížeče. Pro zpřehlednění kódu mají často opakované části vlastní funkci v souboru *helpers.py*. Soubor *views.py* tedy obsahuje funkce

- runScript
- scriptStatus
- getPreviousLogs
- schedulingStatus
- schedulingCreate
- schedulingDelete
- exportDownload
- dashboardBrief
- dashboardAllLogs
- dashboardErrLogs

- scheduled
- numbers

Funkce (view) *runScript* dostane na vstupu parametr *request*, stejně jako všechny následující funkce. Tento parametr obsahuje metadata o požadavku na server. Dále tato funkce dostane na vstupu ID skriptu, se kterým pracujeme. Toto ID se získá z URL, kdy se v souboru *urls.py* předá jako parametr právě funkci *runScript*. Všechny funkce v API aplikaci využívají Django *rest_framework*, který je určený pro práci s REST API. Při zavolání jakékoliv ze zmíněných funkcí výše se nejprve ověří, zdali má daný uživatel právo tuto operaci vykonat. Tato kontrola probíhá na třech úrovních. Nejprve se ověří, zdali uživatel, který volá přes API tuto URL je přihlášený do aplikace. Poté se ověří, zdali toto ID skriptu skutečně existuje. V poslední řadě probíhá kontrola, zdali uživatel má právo přístupu k tomuto skriptu. Funkce slouží ke spuštění skriptu. Jako další tedy proběhne kontrola zavoláním pomocné funkce z *helpers.py*, konkrétně *isRunningOrPending*. Pokud skript již běží, uživatel dostane JSON odpověď s informací, že skript již běží. Pokud skript neběží, zavolá se opět funkce z *helpers.py*, tentokrát *runScript* a daný skript se spustí.

Funkce (view) *scriptStatus* přijímá stejné parametry, jako bylo zmíněno výše. Kontrola přístupu probíhá také u všech views stejně. Z důvodu přehledného kódu tato funkce pouze definuje strukturu vraceného JSONu na nejvyšší úrovni. Tzn. vracený JSON obsahuje několik atributů, jako je například *STATUS*. Data do tohoto atributu se získávají opět pomocí volání pomocné funkce z *helpers.py*, konkrétně *getScriptStatus()*. Pro přehlednost uvádím které funkce z *helpers.py* obstarávájí získávání daných dat.

- Atribut *STATUS* získává hodnoty pomocí funkce *getScriptStatus()*
- Atribut *ID* získává hodnoty pomocí funkce *getLastCeleryTaskID()*
- Atribut *DATE_DONE* získává hodnoty pomocí funkce *getLastCeleryTaskDateTime()*
- Atribut *RESULT* získává hodnoty pomocí funkce *getLastCeleryTaskResult()*
- Atribut *LOG* získává hodnoty pomocí funkce *getCurrentLog()*
- Atribut *EXPORTS* získává hodnoty pomocí funkce *getExportFiles()*

Jak získávají tyto funkce data z databáze bude popsáno dále.

Funkce (view) *getPreviousLogs* vrací v atributu *DATA* logy, z několika posledních běhů skriptů. Data jsou získávaná pomocí *getPreviousLogs*.

Funkce (view) *schedulingStatus* vrací o každém skriptu informace o tom, kdy se má tento skript automaticky spouštět. Funkce nepoužívá pomocné funkce z *helpers.py* a získává data z databáze přímo. Periodické spouštění funguje díky *Celery Beat*, které je vysvětleno v kapitole o *Celery*. *Celery* je napojené na databázi, odkud získává informace o plánování. V aplikaci existuje několik předdefinovaných možností plánování. Víme, že možnosti periodických úkolů jsou dvojího druhu. První je možnost plánování intervalem. Aplikace tedy poskytuje několik předdefinovaných intervalů, které je možné editovat v databázi. Každý

tento interval má své ID. Funkce opět vrací JSON se třemi atributy. První atribut je *intervals*, který poskytuje přehled všech intervalů s popisem a jeho ID. Zde zjistíme, že například spuštění každou hodinu má ID 7. Další atribut *crontabs* poskytuje přehled předdefinovaných možností plánování typu spuštění každé ráno ve 3 hodiny. Opět včetně crontab ID. Tato ID jsou dvě nezávislá číslování. Poslední nejdůležitější atribut je *scheduled*. Tento atribut pro daný skript říká, jaké plánování, ať už pomocí intervalu nebo pomocí crontab pro tento skript platí. Zde je možné demonstrovat práci s Django modely. Pro získání atributu *scheduled* voláme následující funkci.

```
1 PeriodicTask.objects.filter(
2                 task=getTaskName(id)).values(
3                     "id", "task", "interval_id", "crontab_id", "enabled"
4                 ))
```

V této ukázce vidíme práci s modelem *PeriodicTask*. Přistupujeme k jeho objektům a z těch filtrujeme ty, které se týkají skriptu, který nás zajímá. Vracíme hodnoty *id*, což je ID tohoto záznamu, slouží poté k odstranění tohoto záznamu. Dále *task*, což je identifikátor úlohy, které Celery spouští. Dále hodnoty *interval_id* a *crontab_id*. Právě jedna z nich vždy obsahuje nějaké ID. Poslední hodnota je *enabled*, určuje zdali je toto pravidlo aktivní.

Funkce (view) *schedulingCreate* přijímá také parametry *id_cron_or_interval* a *type*. Druhý parametr poskytuje crontab IS nebo ID intervalu. Třetí určuje zdali se jedná právě o interval, nebo o crontab. Funkce vytváří záznam v databázi pomocí modelu *PeriodicTask*. Fukce uloží záznam do databáze s informacemi o skriptu, který nyní plánujeme a s crontab ID nebo ID intervalu. Níže je ukázka uložení intervalu.

```
1 PeriodicTask.objects.create(
2                 interval_id=id_cron_or_interval,
3                 task=getTaskName(id),
4             )
```

Tento kód je obalený v bloku *try*. V případě, že operace selže, což znamená, že dané ID je již přidané, v bloku *catch* se vrací JSON s chybovým kódem.

Funkce (view) *schedulingDelete* se stará o smazání informace o periodickém vykonávání některého ze skriptů. Funkce nepoužívá žádné pomocné funkce z *helpers.py*. V obloku *try* probíhá nejprve nalezení, ke kterému skriptu (ID skriptu) patří zvolené scheduled nebo interval ID. Následně probíhá kontrola, zdali uživatel nezadal kolizní případ. Tedy scheduled nebo interval ID, které nepatří zvolenému skriptu v URL. V případě, že je toto v pořádku proběhne smazání záznamu z tabulky *PeriodicTask*.

Funkce (view) *exportDownload* zajišťuje stažení exportu, který skript vygeneroval. Uživatel nejprve potřebuje zjistit URL tohoto souboru. Tu získá pomocí volání *scriptStatus*. V JSONu, který se vrátí nalezneme v atributu *EXPORTS* například následující data.

```
{
    ...
    "EXPORTS": {
        "ZIPs": [
            "/api/scripts/4/exports/download/export1.zip"
        ]
    }
}
```

To znamená, že při zadání této URL funkce *exportDownload* vrátí uživateli daný soubor s exportem. Funkce tedy, jako všechny ostatní nejprve ověří, zdali má uživatel k tomuto skriptu přístup.

Dále vyhledá cestu k souboru pomocí globální proměnné *PRIVATE_MEDIA_ROOT*, která je uložená v souboru *settings.py*. *PRIVATE_MEDIA_ROOT* obsahuje cestu k soukromým souborům. Tento soubor, stejně jako exporty jsou umístěné ve složce s názvem projektu po boku tří složek s jednotlivými aplikacemi. Pokud daný soubor existuje je vrácen uživateli. V opačném případě je samozřejmě vrácena chyba.

Funkce (view) *dashboardBrief* pracuje s *helpers.py*. Vrací pouze jeden atribut, tím je *BRIEF*. Data do tohoto atributu se získávají pomocí funkce *getDashborad*.

Funkce (view) *dashboardAllLogs* využívá také pomocné funkce. Konkrétně *getPreviousLogs*. Vrací pouze takto získaná data v atributu *LAST_ALL*.

Funkce (view) *dashboardErrLogs* pracuje podobně jako *dashboardAllLogs*. Ovšem volá funkci *getPreviousLogs* s atributem *onlyErrors=True* a data vrací pomocí *LAST_ERRs*.

Dvě poslední jednoduché funkce (views) *scheduled* a *numbers* opět pracují s pomocnými funkcemi a tato data jednoduše vrací v jediném atributu. V případě *scheduled* se jedná o atribut *SHEDULED* a volání funkce *getSheduledDashboard*. V druhém případě, funkce *numbers* obsahuje atribut *NUMBERS* a volá pomocnou funkci *getDashboardNumbers*.

5.2.3 Pomocné funkce

Jelikož většina funkcí ze souboru *views.py* pracuje s pomocnými funkcemi v *helpers.py* z důvodu přehlednosti a čistého kódu, je třeba vysvětlit, jak i tyto funkce fungují. Tento soubor obsahuje něliko funkcí, které zastřešují operace nad databází. Jde o často potřebné operace.

První z funkcí je *userHasAccessToScript*. Přijímá na vstupu ID skriptu, který kontrolujeme. Využívá se při kontrole, jestli daný uživatel má oprávnění pracovat s tímto skriptem. Funkce přistupuje přes model *ScriptsGroup* k objektům v databázi, filtruje všechny skripty, ke kterým má uživatel přístup. Toto se děje, jelikož právo udělení přístupu ke skriptům se určuje na úrovni skupiny skriptů. Poté ověřuje, jestli je mezi těmito skripty skript s dotazovaným ID. Vrací *True* nebo *False*.

Další pomocná funkce je *userHasAccessToGroup*. Tato funkce ověřuje, zdali uživatel má přístup k samotné skupině sktriptů. Opět vrací *True* nebo *False*.

Funkce *getCanAccessGroups* vrací objekty z databáze. Vrací záznamy podle modelu *ScriptsGroup*, ke kterým má uživatel přístup.

Funkce *getClients* může a nemusí přijmout argument *id*. V případě, že jej nedostane, tedy jeho hodnota je *None*, funkce pomocí *getCanAccessGroups* zjistí, ke kterým skupinám skriptů má uživatel přístup. Dále zjistí, pro jaké klienty jsou tyto skupiny skriptů určeny. Vrací seznam klientů, u kterým má tedy uživatel právo přístupu do nějaké skupiny skriptů. Toto se využívá například při vykreslení seznamu klientů v menu. Druhý případ, kdy v argumentu *id* funkce dostane nějaké ID, funkce vrací klienta s tímto ID.

Následující funkce *getClientOfScriptGroup* přijímá argument *scriptGroupID*. Podle tohoto ID skupiny skriptů danou skupinu nalezne a vrací klienta, který je k této skupině přiřazený.

Funkce *getScripts* přijímá volitelný argument *groupID*. Pokud tento argument dostane, vrací objekty podle modelu *ScriptsInfo*. Ovšem filtrované tak, aby na výstupu byly pouze ty, ke kterým má uživatel přístup a také byly ve skupině skripů dané argumentem *groupID*. V druhém případě, kdy funkce na vstupu toto ID nedostane, funkce vrací objekty podle modelu *ScriptsInfo*, ke kterým má uživatel přístup, přičemž nezáleží do jaké skupiny skriptů skript spadá.

Funkce *getScriptsTasksNames* nejprve prvním dotazem na databázi zjistí k jakým ID skriptů má uživatel přístup. A následně hledá v tabulce *TaskResult* všechny výsledky těchto skriptů, které vrací seřazené od nejnovějšího.

Další z pomocných funkcí se jmenuje *getGroups*. Přijímá volitelný parametr *clientID*. V případě, že není zadán, funkce vrací skupiny skriptů, ke kterým má uživatel přístup. V případě, že zadán je, funkce vrací pouze skupiny skriptů od klienta s tímto ID.

Následující funkce s identifikátorem *getGroupByScriptID* vyžaduje na vstupu argument *scriptID*. Vrací jméno skupiny skriptů, pod kterou daný skript spadá. Tedy hodnotu ze sloupce *name_group* z tabulky definovanou modelem *ScriptsInfo*.

Funkce *getClientByScriptID* vyžaduje na vstupu také *scriptID* a následně vrací objekt klienta, který je k tomuto skriptu přiřazený přes skupinu skriptů.

Funkce *getGroupData* vyžaduje na vstupu *groupID*. Vrací objekt definovaný modelem *ScriptsGroup*, který má zmíněné ID.

Funkce *getTaskID* a *getTaskName* překládají jméno Celery úlohy na ID úlohy a naopak. Pokud tedy potřebujeme získat z názvu úlohy *task_1* její ID *1* využijeme funkci *getTaskID*.

Jako další soubor *helpers.py* obsahuje několik tříd. První z nich je třída *Logger*. Tato třída se využívá při zapisování logu, tedy výstupu skriptu do databáze. Před spuštěním skriptu se vytvoří instance této třídy. Při tvorbě instance tento objekt dostane na vstupu v argumentu *celery_task_id* ID úlohy od Celery a nastaví si počítadlo zpráv na hodnotu 0. Následně tato třída obsahuje pouze jednu funkci a to funkci *log*. Tato funkce má jeden

povinný argument s názvem *message*. Pokud skript potřebuje zapsat zprávu (log), volá právě tuto metodu. Funkce poté zapisuje data skrze model *TasksLogger*. Při každém zápisu zapíše tři proměnné, ID Celery úlohy, pořadí zprávy, které je na začátku 0 a samotnou zprávu.

Další třída definovaná v tomto souboru se nazývá *CeleryAndDtbInterface*. Třída zastřešuje operace nad Celery, tedy nad asynchronním spouštěním skriptu. Její metody jsou popsané níže.

První z metod se nazývá *runScript*. Tato metoda spouští samotné skripty. Na vstupu dostává ID skriptu, který má spustit. Následně skript spustí a v tabulce v databázi definovanou modelem *ScriptLastTaskID* aktualizuje informace. Uloží jaké je *last_celery_task_id*, tedy jaké je ID Celery úlohy z posledního běhu skriptu. Také se aktualizuje informace o tom, kdy byl daný skript naposledy spuštěn pomocí sloupce *updated_at*.

Další metoda s názvem *getLastCeleryTaskID*. Tato funkce na svém vstupu vyžaduje ID skriptu. Vrací identifikátor tedy ID Celery úlohy z posledního běhu skriptu. O tuto operaci se pokouší v bloku *try*, pokud skript tedy ještě nebyl nikdy spuštěn, je spuštěna výjímka a funkce vrátí hodnotu *None*.

Metoda *getScriptStatus* vrací status v jakém se nachází daný skript. Data získává z tabulky *TaskResult*. Tento model se nachází přímo v Celery. Obsahuje sloupce jako je *task_id*, *task_name*, *status*, *result*, *getScriptStatus*, *date_done* a další. Pro získání záznamu o posledním běhu skriptu, potřebujeme znát jeho posledni Celery task ID, to získáme pomocí funkce *getLastCeleryTaskID*. Následně z tabulky definovanou modelem *TaskResult* získáme daný řádek a vracíme sloupec *status*. Pokud o tomto v tabulce není záznam, znamená to, že skript ještě nebyl spuštěn z důvodu čekání ve frontě nebo neběžícího Celery serveru. Toto je ošetřeno výjímkou a metoda v tomto případě vrací hodnotu *PENDING*.

Metoda *isRunningOrPending* vrací informaci o tom, zdali skript běží nebo čeká na spuštění. Metoda se ptá na stav pomocí *getScriptStatus*, pokud je stav *PENDING* nebo *STARTED*, vraci se *True*.

Další z metod s názvem *getCurrentLog* vrací log z posledního běhu skriptu. Nejprve se získá ID Celery úlohy z posledního běhu pomocí *getLastCeleryTaskID*. Dále se pomocí tohoto ID naleznou záznamy v tabulce definované pomocí modelu *TasksLogger*. Tyto záznamy se poté vrací jako slovník, kdy klíč každého záznamu je jeho pořadí.

Metoda *getLastCeleryTaskDateTime* vrací z modelu *TaskResult* pole *date_done*. Vrací tedy informaci o tom, kdy daný skript doběhl. Nejprve se opět zjišťuje Celery ID posledního běhu pomocí *getLastCeleryTaskID*. Poté po mocí tohoto ID hledá záznam v tabulce *TaskResult*.

Metoda *getLastCeleryTaskTraceback* a *getLastCeleryTaskResult* fungují stejně jako předchozí zmíněná metoda, pouze s tím rozdílem, že první z nich vrací traceback z posledního běhu, tedy chybový výstup. Druhá výsledek, pokud skript nějaký vrací.

Další z metod se nazývá *getPreviousLogs*. Na vstupu přijimá mimo ID skriptu také *number*, které určuje z kolika posledních běhů uživatel požaduje logy. Dále na vstupu existuje volitelný argument *onlyErrors*, který má výchozí hodnotu *False*. Funkce nejprve získává data z tabulky definované modelem *TaskResult*. Toto se ovšem děje z dvou různých situací, pokud je ID skriptu na vstupu *None*, funkce pracuje se všemi skripty. Pokud se jedná o ID konkrétního skriptu, funkce pracuje pouze s tímto skriptem. První případ se využívá pro získávání dat do dashboardu, kdy uživatele zajímá, zdali selhal jakýkoliv ze skriptů. Dále funkce nabízí možnost výběru zdali vyžadujeme filtrovat pouze chybné stavy skriptů pomocí argumentu *onlyErrors*. Toto se opět využívá v dashboardu, kdy uživatel vidí přehled všech skriptů, které skončily jako poslední s nějakou chybou. Funkce poté definuje strukturu vraceného JSONu na nejvyšší úrovni a pro získání dat do jednotlivých atributů využívá funkce z tohoto *helper.py* souboru.

```
...
reportOneSession = {
    "STATUS": taskResultsFiltered.status,
    "ID": taskResultsFiltered.task_id,
    "CLIENT": self.getClientName(taskResultsFiltered.task_id),
    "SCRIPT_GROUP": self.getGroupName(taskResultsFiltered.task_id),
    "SCRIPT_NAME": self.getScriptName(taskResultsFiltered.task_id),
    "DATE_DONE": taskResultsFiltered.date_done.strftime("%Y-%m-%d %H:%M:%S"),
    "RESULT": taskResultsFiltered.result,
    "LOG": logOneSession
}
...
```

Metoda *getExportFiles* se používá k vracení URL, kde uživatel nalezne export ze skriptu. Metoda využívá proměnnou *settings.PRIVATE_MEDIA_ROOT*, která určuje umístění všech privátních souborů v projektu. Následně se k této cestě přidá následující *"/exports/script_"+ str(id)* abychom získali umístění exportů pro daný skript. Dále tato metoda prochází, zdali v tomto umístění existuje nějaký *ZIP* soubor. Pokud ano, vytváří pro něj URL, odkud jej lze pomocí API stáhnout.

Metoda *getDashboard* vyžaduje na vstupu informaci o tom, kolik z kolika dnů zpětně data požadujeme. Toto uvadíme v argumentu *days_back*. Funkce poté iteruje skrze data těchto dnů a u každého dne získává počet, kolik skriptů doběhlo celkově, kolik s chybou a kolik v pořádku.

Následující metoda se nazývá *getSheduledDashboard*. Jedná se implementačně o nejdelší metodu. Vrací obecný přehled o tom, které skripty jsou plánované a na kdy mají nastavené periodické spouštění. Nejprve si metoda získá přehled všech klientů, ke kterým má uživatel přístup. Následně iteruje skrze všechny klienty. U každého klienta získá všechny skupiny skriptů. Následně iteruje skrze tyto skupiny skriptů a prochází jednotlivé skripty. U těchto skriptů získáváme informace z tabulky, kterou definuje model *PeriodicTask* pomocí filtru s ID daného skriptu. Data o těchto záznamech s periodickým spouštěním se skládají do listů a slovníků, na konci jsou vrácena v jednom listu.

Poslední metoda z tohoto souboru se nazývá *getDashboardNumbers*, vrací pouze informativní hodnoty o tom, ke kolika skriptům, skupinám skriptů a klientů daný uživatel má právo přistoupit.

Další důležitý soubor této aplikace se nazývá *task_helpers.py*. Ten obsahuje funkci, která zajišťuje inicializaci daného skriptu. Je spuštěná před spuštěním samotného skriptu. Soubor také poskytuje funkci, pomocí které skript může exportovat soubory.

5.2.4 Struktura spouštěných skriptů

Všechny úlohy, které Celery spouští se nacházejí v souboru *tasks.py*. Při přidání nového skriptu je nutné skript přidat do složky *scripts*, následně do podsložky *script_ID*. Dále je třeba dodržet strukturu skriptu pomocí následující šablony.

```
1  class class_script_3:
2    def __init__(self, logger):
3      self.logger = logger
4
5    def run(self):
6      self.logger.log("Ukazka logu.")
7      ...
8
9    def dalsi_pomocna_funkce(self):
10     ...
```

V části inicializace je nutné inicializovat *logger*. Dále je nutné, aby tato třída obsahovala metodu s názvem *run*, která je volána při spuštění skriptu. Následně je třeba pro tento skript vytvořit v soboru *tasks.py* úlohu následujícím způsobem.

```
1  @shared_task(bind=True)
2  def task_3(self):
3      lg = initTask(self.request.id.__str__(), 3)
4      script = class_script_3(lg)
5      script.run()
```

Skripty jsou spouštěné samotnou instancí Celery. Ta pomocí funkce *autodiscover_tasks* nalezne veškeré skripty, které bude schopná spustit. Zároveň s aplikací je třeba ještě spustit *Celery worker server*, který bude samotné skripty spouštět.

5.3 Aplikace UI

Aplikace UI má na starosti zobrazování uživatelského rozhraní uživateli. Struktura této aplikace je podobná, jako v případě aplikace API.

Aplikace obsahuje opět soubory *urls.py*, *models.py* a *views.py*. Dále také adresáře *static*, *templates* a *templatetags*.

Adresář *static* slouží k uložení statických souborů, se kterými toto uživatelské rozhraní dále pracuje. Obsahuje čtyři podadresáře s názvy *bootstrap*, *css*, *fonts* a *js*.

- Adresář *bootstrap* obsahuje CSS a JavaScriptové soubory z knihovny Bootstrap, které se v uživatelském rozhraní využívají.
- Adresář *css* obsahuje soubor *styles.min.css*, který definuje kaskádové styly v celé aplikaci.
- Adresář *fonts* obsahuje fonty, které se v rámci aplikace využívají.
- Adresář *js* obsahuje několik JavaScriptových souborů, které zajišťují dynamické zobrazování dat v reálném čase. Tyto jednotlivé soubory budou popsány níže v návaznosti na tom, kde se využívají.

Adresář *templates* obsahuje HTML šablony. Tyto šablony se využívájí ve views, určují, jak bude vypadat klientská část. Je zde umístěn soubor *menu_base.html*, který obsahuje hlavní šablonu uživatelského rozhraní. To znamená boční menu, horní lištu a blok pro obsah. Strukturu tohoto souboru poté dědí ostatní šablony, které definují konkrétní stránky. Konkrétní šablony budou popsány dále.

Adresář *templatetags* obsahuje soubor *ui_extras.py*, který definuje vlastní tagy pro HTML šablonu. Tento soubor obsahuje pouze jednu funkci, tedy jeden vlasntí tag s názvem *get_client_logo_url*, která přijímá ID klienta a vrací URL adresu s jeho logem. Tento tag se používá ve view, který zobrazuje shrnutí všech plánovaných skriptů.

Jelikož veškeré operace se skripty zastřešuje aplikace API, aplikace UI nepotřebuje žádné tabulky v databázi a tudiž je soubor *models.py* prázdný.

Struktura *urls.py* souboru je v této aplikaci následující.

```
1 urlpatterns = [
2 path('', views.redirectRoot, name="ui_root"),
3 path('dashboard/', views.dashboard, name="dashboard"),
4 path('script-groups/client/<int:id>/', views.scriptGroups, name="script_groups"),
5 path('script-groups/<int:id>/', views.scriptGroup, name="script_group"),
6 path('scripts/<int:id>/', views.scriptControls, name="scripts_controls"),
7 path('sheduled/', views.sheduled, name="sheduled"),
8 path('users/', include("users.urls")),
9 ]
```

Na tomto příkladu lze popsat URL strukturu webového rozhraní aplikace. Pokud uživatel zadá přímo homepage aplikace, je přesměrován pomocí view *redirectRoot* na *dashboard/*. Jak tyto jednotlivé views fungují bude popsáno dále. Mimo *dashboard/* uživatel může navštívit *script-groups/client/<int:id>/*. Zde se dostává na přehled skupiny skriptů pod klientem, jehož ID je uvedeno v URL. Dále při zadání *script-groups/<int:id>/* se dostáváme na zobrazení konkrétní skupiny skriptů, jejíž ID je uvedena v URL. Přístup k samotnému ovládání skriptu má uživatel přes *scripts/<int:id>/*. Opět, ID zde značí ID daného skriptu. Přes zadání *sheduled/* má uživatel zobrazit si všechny plánované skripty na jednom místě. Poslední důležité směrování se týká veškerých operací ohledně uživatele, tedy přihlašování, odhlašování nebo například změna hesla. Veškeré tyto operace má na starosti aplikace *USERS*, proto je tato URL směřována do dalšího *urls.py* souboru právě v této aplikaci.

5.3.1 Views, Šablony a JavaScript

V následujících částech budou popsány jednotlivé views. U každého view bude popsána také šablona, se kterou tento view pracuje společně s JavaScriptovým souborem, který zajišťuje dynamické zobrazování dat dané stránky.

Všechny views, které se starají o uživatelské rozhraní kontrolují, zdali uživatel přistupuje pouze tam, kam může. Kontrola tedy probíhá na dou úrovních, jak v UI, tak v API. První view s názvem *redirectRoot* byl popsán výše. Další z nich se jmenuje *dashboard*. Je spuštěn pokud uživatel zadá v URL *dashboard/*. Tento view vrací pomocí funkce *render* HTML šablonu s názvem *dashboard.html*. Tato šablona, stejně jako všechny ostatní, rozšiřuje základní šablonu s názvem *menu_base.html*. Šablona *dashboard.html* obsahuje čtyři prvky, které pracují s API aplikace. První z nich je přehled dat o tom, kolik skriptů (a jiných podobných informací) může daný uživatel spravovat. Následně se zde nachází graf, který vykresluje počet spouštěných skriptů za poslední měsíc a jejich stavy. Poslední dva prvky se týkají zobrazení všech skriptů, které doběhly jako poslední. První z nich zobrazuje všechny skripty a druhé ty, které doběhly s chybou. Všechny čtyři prvky obsahují v HTML unikátní ID, pomocí kterého se poté naplňují skrze volání API JavaScriptem. JavaScriptový soubor, který se stará o získávání dat na této stránce se jmenuje *dashboard.js*.

Soubor *dashboard.js* obsahuje několik funkcí, z nichž nejdůležitější jsou *renderTopNumbers*, která posílá dotaz na API, odkud získává data ohledně informací ke kolika skriptům atd. má uživatel přístup. Následně tato data vepíše do HTML pomoci ID prvku.

```
1  function renderTopNumbers(){
2      var r = new XMLHttpRequest();
3      r.onload = function(){
4          var responce = JSON.parse(this.responseText);
5          let {NUMBERS: numbers} = responce;
6          document.getElementById("sdh-top-clients-count").innerHTML =
7          numbers.NUMBER_YOUR_CLIENTS;
8          document.getElementById("sdh-top-groups-count").innerHTML =
9          numbers.NUMBER_GROUPS;
10         document.getElementById("sdh-top-scripts-count").innerHTML =
11         numbers.NUMBER_SCRIPTS;
12     }
13
14     r.open('GET', url + '/api/scripts/dashboard/numbers?format=json', true);
15     r.send();
16 }
```

Další důležitá funkce je *renderBriefChart*, která získává data do grafu. Následně s těmito daty volá funkci *drawChart*, která pomocí knihovny *Chart.min.js* daný graf vykresluje.

Funkce *renderErrsLogs* a *renderAllLogs* získávají data do dvou posledních bloků na stránce. Následně pro jejich vykreslení využívají funkci *loadLogs*, která daná data přidá do DOM struktury HTML stránky. Data je třeba přidat tímto způsobem, jelikož se jedná o interaktivní prvky, které zobrazují detailnější informace po rozkliknutí.

Funkce *renderErrsLogs*, *renderAllLogs*, *renderBriefChart* a *renderTopNumbers* jsou při načtení stránky automaticky zavolány. Jelikož data na této stránce nevyžadují pravidelné aktualizace, jako například stránka zobrazující běh skriptu, jsou data načtena pouze při načtení stránky s výjímkou toho, pokud uživatel využije možnost zobrazení více nebo méně běhu do historie. V tomto případě se znovu vykoná dotaz přes API a data se aktualizují.

Druhý view s názvem *scriptGroups* při zpracování kontroluje, zdali má uživatel k danému klientovi vůbec přístup. ID klienta se předává v URL. Pokud ne, stránka vrací chybový kód 401 společně s chybovou hláškou. Pokud uživatel právo k zobrazení skriptů pod tímto klientem má renderuje HTML šablonu s názvem *groups.html*. Tento view při získávání skupin skriptů nepoužívá API, ale data získá pomocní funkcí z *helpers.py* a předává je dále pomocí proměnné *context*, jedná se o slovník v Pythonu.

Šablona *groups.html* zobrazuje všechny skupiny skriptů, ke kterým má uživatel přístup. Propisování informací do šablony z kontextu Django umožňuje skrze speciální tagy. Níže naleznete ukázku s částí kódu, ve které je vidět, jak django pracuje s daty z kontextu.

```
1  {% for group in canAccessGroups %}
2      <div class="col-sm-6 col-md-5 col-lg-4 item">
3          <div class="box shadow-1 rounded"><i class="fa fa-cogs icon"></i>
4              <h3 class="name">{{ group.name_group}}</h3>
5              <p class="description">{{ group.group_description }}</p>
6              <a class="learn-more" href="{% url 'script_group' group.id %}">
7                  Skripty
8              </a>
9          </div>
10     </div>
11 {% endfor %}
```

Třetí view s názvem *scriptGroup* zobrazuje uživateli konkrétní skripty v rámci skupiny skriptů. Přijímá ID skupiny skriptů. Opět probíhá kontrola na práva přístupu do této skupiny. Tento view také nepracuje s API, jelikož pouze naviguje uživatele ke skriptu, který si následně přeje spouštět. Data se tedy získávají opět pomocných funkcí a předávají pomocí proměnné *context* funkci *render*, která vrací šablonu *group.html*. Tato šablona vykresluje přehled skriptů v dané skupině skriptů.

Další uživatelská stránka je samožná stránka ovládání konkrétního skriptu. Jedná se o nejsložitější část uživatelského rozhraní. O tuto část se stará view s názvem *scriptControls*. Opět kontroluje uživatelská prává přístupu. Tato část aplikace kombinuje předávání dat pomocí kontextu přímo uvnitř view a volání API pomocí JavaScriptu. Předání konktextem se používá pro předání dat jako jsou informace o klientovi nebo o daném skriptu. Jou to tedy data, která se v průběhu práce se skriptem nemění a není třeba je dynamicky načítat. Funkce renderuje šablonu s názvem *script_controls.html*.

Tato šablona je rozdělená do bloků. Pokud daný blok obsahuje část, do které je dynamicky třeba propisovat data, daný HTML element obsahuje ID, se kterým poté pracuje JavaScript, který daná data doplňuje. První dva bloky obsahují pouze informace o daném skriptu, data zde se propisují pomocí kontextu přímo z databáze. Následující prvek zobrazuje aktuální log skriptu. Jelikož log se mění během toho, co skript běží jsou tato

data získávána dotazem na API. Veškeré JavaScriptové funkce, které zajišťují získávání aktuálních dat na této stránce se nacházejí v souboru *script_controls.js*. Následující prvek umožňuje uživateli stáhnout exporty pokud skript nějaké vygeneruje. Toto je také prováděno přes JavaScript, jelikož exporty jsou dostupné po skončení skriptu. Jak fungují dané JavaScriptové funkce bude popsáno dále. Následuje velice důležitá část, ve které uživatel vidí přehled několika posledních běhů. Na první pohled zdali skript doběhl úspěšně nebo ne. Data se získávají pomocí volání API, jelikož po každém spuštění je třeba tuto část aktualizovat. Uživatel má současně možnost si daný běh otevřít a zjistit více podrobností. Poslední blok, který také pracuje dynamicky pomocí získávání aktuálních dat a také jejich zasílání na server je část, která umožňuje nastavování plánování periodického běhu skriptů. V horní části stánky se nachází také ovládací tlačítko, které umožňuje skript spustit. Po stisknutí se odešle dotaz na API aplikace.

Soubor s názvem *script_controls.js* obsahuje několik funkcí, která zajišťují dynamické získávání a odesílání dat z API. Při načtení stránky se spustí několik JavaScriptových funkcí a těmi jsou *statusScript.js*, *loadLogs.js*, *loadSheduling.js*. Dále se nastaví pravidelné volání funkce *statusScript.js*.

Funkce *statusScript.js* volá API aplikace a získává aktuální data o skriptu, se kterými dále pracuje. V případě, že je tato funkce volána ihned při načtení stánky, jako první vykreslí log z posledního běhu skriptu. Následně v případě, že skript právě doběhl vykreslí opět jeho celý log z posledního běhu, dále načte exporty, které tím pádem umožní uživateli stáhnout. Následně aktualizuje přehled s posledními běhy skriptu. Tato funkce také ovládá vzhled tlačítka, které spouští daný skript. V případě, že skript čeká ve frontě na spuštění, tlačítko tuto situaci vizuálně znázorní. Stejně tak v případě, kdy daný skript běží. Vedle tlačítka pro spuštění skriptu se nachází informační rámeček, ve kterém se po doběhnutí skriptu zobrazí zdali skript skončil v pořádku nebo s chybou.

Funkce *loadLogs.js* získává z volání API informace o posledních bězích skriptu. Aby tato data mohla zobrazit v interaktivním boxu, je třeba je přidat do DOM struktury stánky. Data jsou potom zobrazena a uživatel má možnost po kliknutí na jednotlivé běhy získat podrobnější informace.

Funkce *loadSheduling.js* volá API aplikace k získání informací o tom, zdali daný skript má nějaké periodické plánování. Tyto informace poté propisuje do uživatelského rozhraní. Uživatel má možnost přidávat další periodické plánování kliknutím na předdefinované možnosti. Toto zajišťuje funkce *addSchedule*. V takovém případě probíhá další volání API s požadavkem na přidání tohoto plánování. Záznamy se poté v uživatelském rozhraní aktualizují. JavaScript také na místech aplikace, kde se zobrazuje přehled historických běhů skriptů umožňuje uživateli pomocí slideru nastavit, kolik těchto záznamu požaduje vidět. Následně JavaScript provede volání API a daný počet informací zobrazí.

Následující view s názvem *sheduled* získává data přímo z databáze a nepracuje s API. Uživateli slouží k celkovému přehledu, aby na jednom místě viděl, u kterých skriptů má aktivované periodické spouštění a jaké. Strukturovaná data poskytuje již zmíněná funkce s názvem *getSheduledDashboard*. Tato data následně předává v proměnné *context* funkci *render*, která pracuje s šablonou *sheduled.html*.

Šablona *sheduled.html* zobrazuje data pomocí speciálních Django značek, které do HTML kódu vkládají data z proměnné *context*. Tyto tagy jsou dvojího typu[5], první z nich má podobu *{{ variable }}*. Takto Django dokáže do HTML šablony vepsat data, se kterými šablona může pracovat. Druhý typ může vypadat například takto *{% for o in some_list %}*. Jedná se o začátek iterace v listu. V šabloně je tedy možné zpracovávat listy a iterovat v nich.

5.4 Aplikace USERS

Aplikace USERS zajišťuje správu uživatelů. Opět má podobnou strukturu jako ostatní dvě zmíněné aplikace.

Soubor *models.py* obsahuje pouze jeden model (třídu) s názvem *Profile*. Django samo o sobě poskytuje třídu pro uchování informace o uživatelích, ovšem pokud potřebujete rozšířit sloupce v této tabulce, je vhodné vytvořit každému uživateli profil a vazbou 1:1 je propojit, jako je tomu v tomto případě. Důvod vytváření profilu je pole *sent_fails_email*, které uchovává pravdivostní hodnotu, zdali uživatel vyžaduje zasílání emailu, pokud některý ze skriptů skončí s chybou. Tento soubor také obsahuje dvě speciální funkce s dekorátorem *@receiver*. První z nich se nazývá *create_user_profile*. Zajišťuje, aby se při vytvoření objektu uživatele vytvořil také objekt s profilem tohoto uživatele. Právě receiver se stará o to, aby toto bylo automatické a nebylo třeba uživateli vytvořit účet a profil ve dvou krocích. Druhá funkce s názvem *save_user_profile* zajišťuje aktualizaci profilu pokaždé, když se aktualizuje uživatelský účet.

Soubor *urls.py* obsahuje následující URL

- *login/*
- *logout/*
- *password-reset/*
- *password-reset/done/*
- *password-reset-confirm/<uidb64>/<token>/*
- *password-reset/complete/*
- *password-change/*
- *password-change/done/*
- *profile/*

pro připomenutí, uvádím, že tyto URL musí být za *users/*. Tedy například *users/password-reset/complete/*.

Ve frameworku Django existují views (třídy), které lze využít k věcem, které se ve vývoji aplikací často opakují. Z tohoto důvodu v Djangu nalezneme několik tříd, které se starají o přihlašování, odhlašování, změnu hesla a podobně. Programátor se tedy může soustředit

na vývoj samotné aplikace a tyto běžné věci za něj Django vyřeší. URL po přihlášení tedy směruje do view s názvem *LoginView*, kterému pouze předáme HTML šablonu s názvem *login_page.html*. Tato šablona obsahuje přihlašovací stránku, která využívá tzv. *cripsy forms*, což jsou formuláře, které zajistí pole pro vyplnění uživatelského jména a hesla. Také se starají například o vykreslení chybové hlášky javaScriptem nad daným polem, pokud nastane nějaká chyba. Další z views se nazývá *LogoutView*, který dostane v argumentu šablonu *logged_out.html*.

Změna hesla, pokud jej uživatel zapomene probíhá následujícím způsobem. Uživatel se nejprve dostane na stránku *password-reset/*, pomocí view *PasswordResetView* s šablonou *password_reset.html*. Na této stránce uživatel vyplní svůj email, který má vyplněný v účtu. Aplikace uživateli odešle email a uživatele pošlě na *password-reset/done/*. Email, který uživatel obdrží obsahuje speciální na míru generovanou URL adresu, která může vypadat například takto *users/password-reset-confirm/MQ/5ej-ab9ae68a55d5ce6b44e8/*. Jde tedy o URL *password-reset-confirm/<uidb64>/<token>/* ze souboru *urls.py*. Tento link je na jedno použití a umožní uživateli změnit jeho heslo. Parametr *uidb64* obsahuje primární klíč daného uživatele zakódovaný v base64. Parametr *token* slouží ke kontrole, zdali je odkaz pro obnovu hesla validní. Po kliknutí na daný link se uživatel dostane tedy díky třídě *PasswordResetConfirmView* na stránku, kde má možnost zadat nové heslo. Jakmile toto potvrdí, díky view *PasswordResetCompleteView* se jeho heslo změní a uživatel se dostane na stránku *password-reset/complete*.

Soubor *urls.py* obsahuje také URL pro změnu hesla, těmi jsou *password-change/* a *password-change/done/*. První z nich využívá view s názvem *PasswordChangeView* a možní uživateli změnit si své heslo. Druhý využívá třídu *PasswordChangeDoneView* a uživateli pouze potvrzuje změnu hesla.

Soubor *urls.py* také obsahuje záznam *profile/*, který zako jedninný z tohoto souboru používá nepředddefinovanou třídu. Jedná se o view s názvem *UserProfile*. Ten využívá šablonu *user_profile.html* a zobrazuje uživateli základní informace o jeho účtu.

Kapitola 6

Testování

Testování aplikace proběhlo na několika testovacích skriptech, které demonstrovaly funkčnost všech požadavků na aplikaci, které byly uvedené na začátku. Během testů jsem testoval aplikaci pomoci několika různých účtů s různými kombinacemi práv ke skupinám skriptů.

Testování se účastnili dva marketéři, kteří si nejprve prošli uživatelské prostředí. Následně dostávali úkoly, které měly vykonat. Ukázalo se, že aplikace je správně navržená a nebyl problém dané úkoly splnit.

Samotné skripty, které byly testované spadaly pod dva různé klienty. První z nich obsahoval pouze jednu skupinu skriptů, tato skupina obsahovala jeden skript, který demonstroval ukázku exportu souborů. Skript tedy chvíli běží a po jeho doběhnutí jsou vygenerované soubory, které uživatel může stáhnout.

Druhý klient obsahoval také jednu skupinu skriptů, tato skupina ovšem obsahovala čtyři různé skripty. První z nich simuluje činnost 10 sekund, poté úspěšně skončí. Druhý skript ukazuje schopnost aplikace zasílat emaily, skript tedy zasílá email na testovací emailovou schránku. Třetí skript zobrazuje, jak se aplikace chová, pokud skript skončí chybou. Kód tohoto skriptu tedy úmyslně obsahuje chybu a skript skončí neúspěšně. Poslední skript demonstruje schopnost aplikace připojit se do API PPC nástrojů, konkrétně na příkladu Skliku. Skript se připojí do testovacího Sklik účtu a pozastaví veškeré vyhledávací kampaně.

Během testováni byla provedena kontrola autentizace. Uživatelé tedy testovali, jestli skutečně nemohou přistoupit do části aplikace, ke které nemají dostatečná oprávnění. Mimo testování oprávnění a spouštění samotných skriptů byly provedeny testy na automatické periodické spouštění aplikace. Otestován byl také dashboard, který správně a přehledně reportoval aktuální stav skriptů.

Uživatelé během testů sledovali chování aplikace a dle jejich slov jim aplikace vizuálně správně zobrazovala, co se právě děje a vše bylo přehledná. To stejné pokud marketér potřeboval zjistit nějakou historickou informaci, zdali nějaký skript doběh v pořádku. Intuitivní a jednoduché ovládání ocenili marketéři i při nastavování periodického běhu skriptu a zobrazení přehledu o celkovém periodickém plánování všech skriptů.

Kapitola 7

Závěr

Cílem této bakalářské práce bylo vytvořit webovou aplikaci pro marketéry, kteří využívají různé PPC nástroje. Správu kampaní v těchto nástrojích lze automatizovat pomocí různých skriptů. Aby tito marketéři mohli dané skripty bez znalosti programování jednoduše ovládat, vznikla aplikace, která jim to umožní.

Základní motivace bylo zjistit, jak dostat tyto skripty k marketérům, tak aby je mohli snadnou využívat. Z tohoto důvodu jsem se rozhodl o webovou aplikaci, ke které může uživatel přistoupit odkudkoliv a potřebuje k tomu pouze webový prohlížeč. Aplikace musí splňovat mnoho různých požadavků jako je například možnost generování souborů a následné stažení nebo automatické periodické spouštění skriptů. Jelikož žádný existující nástroj nepodporoval kombinaci všech požadavků zadání, bylo třeba vytvořit aplikaci vlastní.

Na základě těchto požadavků byla implementována webová aplikace v jazyce Python ve frameworku Django s použitím knihovny Celery pro podporu asynchronního běhů skriptů.

Aplikace byla otestována samotnými marketéry, kteří ji ocenili a rádi by ji v budoucnu využívali.

Literatura

[1] *Google AdWords Turns 15: A Look Back At The Origins Of A $60 Billion Business* [online]. 2015 [cit. 2020-03-08]. Dostupné z: https://searchengineland.com/google-adwords-turns-15-a-look-back-at-the-origins-of-a-60-billion-business-234579.

[2] *The 10 most popular programming languages, according to the Microsoft-owned GitHub* [online]. 2019 [cit. 2020-03-11]. Dostupné z: https://www.businessinsider.com/most-popular-programming-languages-github-2019-11.

[3] *Django vs. Flask in 2019: Which Framework to Choose* [online]. 2019 [cit. 2020-03-08]. Dostupné z: https://testdriven.io/blog/django-vs-flask/.

[4] *Celery Broker Overview* [online]. 2020 [cit. 2020-03-11]. Dostupné z: https://docs.celeryproject.org/en/latest/getting-started/brokers/.

[5] *Django Documentation: Built-in template tags and filters* [online]. 2020 [cit. 2020-03-11]. Dostupné z: https://docs.djangoproject.com/en/3.0/ref/templates/builtins/.

[6] *Django Documentation: Django settings* [online]. 2020 [cit. 2020-03-11]. Dostupné z: https://docs.djangoproject.com/en/3.0/topics/settings/.

[7] *Django Documentation: Models* [online]. 2020 [cit. 2020-03-11]. Dostupné z: https://docs.djangoproject.com/en/3.0/topics/db/models/.

[8] *Django Documentation: URL dispatcher* [online]. 2020 [cit. 2020-03-11]. Dostupné z: https://docs.djangoproject.com/en/3.0/topics/http/urls/.

[9] *Django Documentation: Writing views* [online]. 2020 [cit. 2020-03-11]. Dostupné z: https://docs.djangoproject.com/en/3.0/topics/http/views/.

[10] *Facebook Developers: Access and Authentication* [online]. 2020 [cit. 2020-03-08]. Dostupné z: https://developers.facebook.com/docs/marketing-api/access.

[11] *Google Developers: AdWords API* [online]. 2020 [cit. 2020-03-08]. Dostupné z: https://developers.google.com/adwords/api/docs/guides/start.

[12] *Google Developers: Make Your First API Call* [online]. 2020 [cit. 2020-03-08]. Dostupné z: https://developers.google.com/adwords/api/docs/guides/first-api-call#request_a_developer_token.

[13] *Google Support: About the Google Search Network* [online]. 2020 [cit. 2020-03-08]. Dostupné z: https://support.google.com/google-ads/answer/1722047?hl=en.

[14] *Google Support: Remarketing* [online]. 2020 [cit. 2020-03-08]. Dostupné z: https://support.google.com/google-ads/answer/2453998?hl=cs.

[15] *Google Support: Using scripts to make automated changes* [online]. 2020 [cit. 2020-03-08]. Dostupné z: https://support.google.com/google-ads/answer/188712?hl=en.

[16] *Nápověda Sklik: Obsahová síť* [online]. 2020 [cit. 2020-03-08]. Dostupné z: https://napoveda.sklik.cz/zaciname-inzerovat/obsahova-sit/#partnerske-weby.

[17] *Nápověda Sklik: Specifikace CSV souboru* [online]. 2020 [cit. 2020-03-08]. Dostupné z: https://napoveda.sklik.cz/pokrocila-prace-s-daty/import-a-export/specifikace-csv-souboru/.

[18] *Nápověda Sklik: Vyhledávací síť* [online]. 2020 [cit. 2020-03-08]. Dostupné z: https://napoveda.sklik.cz/zaciname-inzerovat/vyhledavaci-sit/.

[19] *Sklik API Drak* [online]. 2020 [cit. 2020-03-08]. Dostupné z: https://api.sklik.cz/drak/.

[20] *What is Python? Executive Summary* [online]. 2020 [cit. 2020-03-08]. Dostupné z: https://www.python.org/doc/essays/blurb/.

[21] Zublenko, E. *Why Django is the Best Web Framework for Your Project* [online]. 2020 [cit. 2020-03-13]. Dostupné z: https://steelkiwi.com/blog/why-django-best-web-framework-your-project/.

[22] Čápka, D. *Představení MVC a MVT architektury v Django* [online]. 2020 [cit. 2020-03-11]. Dostupné z: https://www.itnetwork.cz/python/django/predstaveni-mvc-a-mvt-architektury-v-django.