TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

Klientská aplikace pro energetický portál

Bakalářská práce

Studijní program: B2646 – Informační technologie
Studijní obor: 1802R007 – Informační technologie

Autor práce: **Michal Samek**
Vedoucí práce: Ing. Tomáš Bedrník

Liberec 2019

TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

Zadání bakalářské práce

Klientská aplikace pro energetický portál

Jméno a příjmení: **Michal Samek**
Osobní číslo: M14000071
Studijní program: B2646 Informační technologie
Studijní obor: Informační technologie
Zadávající katedra: Ústav mechatroniky a technické informatiky
Akademický rok: **2018/2019**

Zásady pro vypracování:

1. Vyberte vhodný framework pro vývoj webové aplikace v jazyce Python.
2. Pomocí vybraného frameworku naprogramujte aplikaci komunikující se serverem poskytujícím data pomocí rozhraní REST API a JSON. Aplikace bude umožňovat i přímou komunikaci s měřicími přístroji pomocí knihovny dodané vedoucím.
3. Aplikace bude umožňovat vyhledávání dat pomocí tagů a času, víceuživatelský přístup s rozdělením rolí, uživatelsky definované obrazovky se zobrazením vybraných veličin, nastavení limitů pro jednotlivé veličiny a generování a zasílání reportů periodicky nebo po překročení limitů.
4. Aplikaci programujte s důrazem na budoucí rozšiřitelnost a důsledně oddělte jednotlivé vrstvy aplikace tak, aby bylo možné rychle reagovat na změny API.

Rozsah grafických prací:	dle potřeby dokumentace
Rozsah pracovní zprávy:	30–40 stran
Forma zpracování práce:	tištěná/elektronická

Seznam odborné literatury:

[1] GUARDIA, Carlos de la, 2016. Python Web Frameworks. B.m.: O'Reilly Media. ISBN 978-1-4920-3787-3.

[2] ALLAMARAJU, Subbu, 2010. RESTful Web Services Cookbook: Solutions for Improving Scalability and Simplicity. 1 edition. Beijing; Sebastopol,CA: Yahoo Press. ISBN 978-0-596-80168-7.

[3] SALE, David, 2014. Testing Python: Applying Unit Testing, TDD, BDD and Acceptance Testing. 1 edition. Chichester: Wiley. ISBN 978-1-118-90122-9.

Vedoucí práce:	Ing. Tomáš Bedrník Ústav mechatroniky a technické informatiky
Datum zadání práce:	10. října 2018
Předpokládaný termín odevzdání:	30. dubna 2019

L. S.

prof. Ing. Zdeněk Plíva, Ph.D.
děkan

doc. Ing. Milan Kolář, CSc.
vedoucí ústavu

V Liberci 10. října 2018

Prohlášení

Byl jsem seznámen s tím, že na mou bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci (TUL) nezasahuje do mých autorských práv užitím mé bakalářské práce pro vnitřní potřebu TUL.

Užiji-li bakalářskou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti TUL; v tomto případě má TUL právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Bakalářskou práci jsem vypracoval samostatně s použitím uvedené literatury a na základě konzultací s vedoucím bakalářské práce a konzultantem.

Současně čestně prohlašuji, že tištěná verze práce se shoduje s elektronickou verzí, vloženou do IS STAG.

Datum: 30.4.2019

Podpis:

Poděkování

Rád bych na tomto místě poděkoval vedoucímu práce panu Ing. Tomáši Bedrníkovi, za odborné vedení, užitečné rady během konzultací a za jeho čas, který mi věnoval.

Abstrakt

Bakalářská práce se zabývá návrhem a tvorbou webové aplikace, která má sloužit jako vzor pro další klientské aplikace komunikující se serverem, který shromažďuje naměřená data z elektrometrů. Vývoj aplikace proběhl v jazyce Python a je využit framework Django, který byl vybrán pro svou jednoduchost a rozšiřitelnost. Komunikace se serverem probíhá skrze HTTP požadavky a odpovědi obsahující data ve formátu JSON. Taktéž jsou využity i doplňující knihovny Django REST framework pro asynchronní dotazy a plotly.js pro vykreslování dat do grafů v Javascriptu. Návrh se odvíjí od již existujících řešení a od plánovaným budoucím využití. Pro realizaci byl použit program PyCharm a operační systémy Windows 7 a 8.1. Aplikace dokáže vykreslovat grafy, ověřovat uživatele, vypisovat alarmy a posílat reporty.

Klíčová slova

Webová aplikace, Python, REST API, JSON, Javascript, AJAX

Abstract

The bachelor thesis deals with the design and creation of a web application, which should serve as a model for other client applications communicating with the server, which collects the measured data from electrometers. Application development was done in Python, and the Django framework has been used for its simplicity and scalability. Communication with the server is through HTTP requests and responses containing JSON data. Also, additional Django REST libraries are used for asynchronous queries and plotly.js for plotting data in Javascript charts. The design is based on already existing solutions and from the planned future use. PyCharm and Windows 7 and 8.1 operating systems were used for implementation. The application can plot graphs, authenticate users, show alarms, and send reports.

Keywords

Web aplication, Python, REST API, JSON, Javascript, AJAX

Obsah

Seznam Tabulek

Seznam Obrázků

Seznam zkratek

HTTP	Hypertext Transfer Protocol
JSON	JavaScript Object Notation
AJAX	Asynchronous JavaScript and XML
REST	Representational State Transfer
API	Application Programming Interface
HVAC	Heating, Ventilation and Air Conditioning
HTML	Hypertext Markup Language
URL	Uniform Resource Locator
URI	Uniform Resource Identifier

Úvod

Bez elektrické energie by se dnešní svět již neobešel. Využívá ji nespočet zařízení s nejrůznějším účelem od důležitých systémů, jako třeba v nemocnicích, až po obyčejné domácí spotřebiče. Během využívání elektrické energie lze pozorovat celou řadu veličin a vlastností, jako je například velikost proudu či napětí nebo spotřeba. K tomu slouží různé druhy inteligentních elektroměrů. Údaje z těchto veličin pak lze využít k úpravám kvality elektrické energie či k regulaci spotřeby. Nejdříve je však nutné data nasbírat, následně pokud možno centralizovaně uložit a nakonec analyzovat. K těmto účelům existují speciální softwary, které ukládají aktuální naměřené hodnoty a ty pak přehledně vypisují a zpětně analyzují.

Na univerzitě je vyvíjen takovýto server pro správu dat z analyzátorů kvality elektrické energie, kompenzátorů jalového výkonu a dalších inteligentních měřáků. Server bude poskytovat možnosti komplexní správy přístrojů a zobrazování dat z připojených zařízení. V budoucnu ho pak bude možné spouštět lokálně pro několik místních zařízení z lokální sítě na průmyslovém ARM mikropočítači, nebo globálně přes internet, čímž bude veřejně dostupný, a tím i přístupný klientským aplikacím.

Cílem této práce pak bylo vyvinout ukázkovou klientskou aplikaci k tomuto serveru, která je schopna pracovat s již funkčními prvky, jako je přihlašování a ověřování uživatelů vůči serveru, výpisy dat do grafů a výpisy alarmů, čímž by měla představovat prototyp pro další aplikace. Při jejím vývoji je tedy nutné počítat s budoucím rozšířením o další funkce.

Práce obsahuje porovnání vybraných stávajících řešení, popis použitých technologií, rešerši dostupných frameworků pro webové aplikace v Pythonu, návrh řešení a nakonec aplikování návrhu a popis jak aplikaci do budoucna rozšířit.

1 Problematika

Na jednom ze školních zařízení běží server, založen na Python frameworku Flask. Server je napojený analyzátory kvality elektrické energie a jeden kompenzátor jalového výkonu, z nich načerpaná data ukládá do místní databáze. Data, je pak server schopen dále poskytovat skrze síť dalším aplikacím. Účelem tohoto zadání je vytvořit jednu z těchto klientských aplikací, která bude se serverem komunikovat skrze posílání požadavků na serverové REST API a získávat zpět data ve formátu JSON.

Plány se serverem a aplikací zde ale nekončí. Server neobsahuje a neposkytuje pouze data z elektroměrů ale umožňuje i přihlášení a ověřování uživatelů, uživatelských skupin a jim přidělovaných práv. Nebo například přístup k alarmům, sloužící k upozornění pokud nastane definovaná situace. Aktuální funkce serveru, a tudíž i aplikace, jsou zatím omezené, neboť celý server je stále ve vývoji. Ovšem návrh aplikace musí počítat nejen s tím, že všechny zadané funkce budou dodělány, ale i s velmi pravděpodobným přidáním nových služeb.

2 Stávající řešení

Stávající řešení jsem analyzoval ze dvou zdrojů, přičemž jsem udělal průnik z prvních 5 položek z obou seznamů seřazených podle nejlepšího hodnocení a vybral tak 3 softwary, které krátce shrnu a v pozdější části práci se jimi i inspiruji z hlediska funkcí a uživatelského rozhraní. [1] [2]

2.1 Entronix EMP

Entronix je software od společnosti Entonix a poskytuje mnohostranný energetický management, který není nijak závislý na případném stávajícím automatizačním systému. Může běžet paralelně s ním, kompletně nad ním ale i bez něj. Naměřená energetická data a nastavení jsou posílány a ukládány do cloudu s přednastavenými sktrukturami pro snadné výpisy.

Platforma je určená pro malé až střední organizace spravující jedno či více míst a hodí se na využití od bytových prostor až po průmyslová místa. Hlavní funkce Entronixu je snadné zavedení a ovládání. Mimo jiné pak monitorování spotřeby, predikce poptávky nebo detekce chyb. Umožňuje také sledovat údaje ze senzorů, regulačních ventilátorů a další zařízení. Umí taktéž hlídat abnormální využití HVAC jednotek, plánovat rozpočty, porovnávat budovy. Pomocí historických dat a jejich okolností je také schopný předpovídat spotřebu a energetickou poptávku v několika následujících dnech.

Další funkcí je alarmový systém umožňující uživatelům zasílat e-mailová upozornění pro celou řadu podmínek jako například blížící se špičky požadavků, ztrát energie nebo aktuální anomálie. V případě, že dostupné moduly nesplňují uživatelské potřeby, software rovněž umožňuje vytváření vlastních funkcí. [3] [2] [1]

2.2 Wattics

Wattics je software pro správu zařízení využívající cloudové uložiště. Je vhodný pro malé až středně velké podniky v různých průmyslových odvětví. Mezi klíčové vlastnosti patří analýza trendů, monitorování energie, měření a ověřování, analýza nákladů nebo plýtvání a plánování úspor.

Wattics také uživatelům umožňuje spravovat projekty, uživatelské oprávnění, úsporu energie, analýzu a porovnávání spotřebu energie a sní související náklady. Taktéž nabízí integraci s různými měřiči, senzory a datovými systémy. A v neposlední řadě může automaticky posílat oznámení kdy se cena nebo spotřeba liší od očekávaných vzorců. [4] [1] [2]

2.3 EnergyElephant

EnergyElephant, využívající cloudové uložiště, je software pro monitorování energie, který pomáhá uživatelům sledovat a řídit spotřebu energie. Je vhodný pro nejrůznější typy budov a pro všechna průmyslová odvětví. Funkce pro správu účtů automaticky kontroluje jednotkové sazby a pomáhá uživatelům snížit odhadované náklady za energie. Z tímto účelem umožňuje i provádět energetické audity a hodnocení. Uživatelé také mohou porovnávat nabídky od různých dodavatelů energie k nalezení nejvýhodnější kombinace.

Sledováním údajů o využití energie a uhlíkových emisí může EnergyElephant pomoci uživatelům určit zdali jsou pro jejich lokalitu použity vhodné alternativy energie. Ve zprávách o prognózách nákladů lze pak najít potenciální dopady použitých druhů energie na výdajích a emisích. Mezi další funkce pak patří sledování spotřeby energie, nástroje pro vzdělávání zaměstnanců a monitorování a ověřování. [5] [2] [1]

2.4 Srovnání

Tabulka 1: Srovnání stávajících softwarů [6]

	EnergyElephant	Entronix	Wattics
Vhodný pro	Budovu, areál nebo mezinárodní správu energií či financí.	Monitorování energetické správy kancelářských budov, nemocnic, vysokých škol, ..	Energetické poradce, společnosti poskytující energetické služby (1 - 1000+ uživatelů)
Hodnocení	90%	100%	95%
Řízení dodržování předpisů	Ne	Ano	Ano
Plánování zařízení	Ne	Ano	Ano
Řízení zátěže	Ne	Ano	Ne
Prognóza zatížení	Ne	Ano	Ano
Řízení rizik	Ano	Ano	Ne
Platforma	Web/Cloud, iOS, Android	Web/Cloud, iOS, Android	Web/Cloud

3 Technologie

Níže jsou uvedeny a rozebrány použité technologie odvíjející se od zadaní, návrhu a mého řešení. Vytvoření webové aplikace za použití jazyka Python bylo stanoveno zadáním. Volbu vhodného frameworku jsem pak učinil já na základě zvážení důležitých vlastností potřebných pro řešení.

3.1 Webová aplikace

Webová aplikace je software běžící na libovolném serveru, dostupná nejčastěji skrze internet ale i jinou počítačovou síť v závislosti na poloze serveru. K aplikaci lze přistoupit, pokud známe adresu serveru, na kterém běží a port, pod kterým komunikuje. Na takovýto dotaz pak reaguje zasláním dokumentu HTML a přídavnými soubory jako css nebo javascriptový script. Někdy se těmto aplikacím říká dynamické a od statických se liší tím, že obsah, který vrací na požadavek není dopředu, „staticky“, stanoven. Na místo toho je generován kódem v závislosti na akcích a oprávnění uživatele.

Webové aplikace fungují následovně. Nejprve pošle klientské zařízení požadavek na server, skrze webový prohlížeč a adresu serveru. Server požadavek zpracuje na základě, o jaký požadavek se jedná, odkud je poslán, na hodnotách cookies a dalších proměnných. Podle toho pak vygeneruje statický HTML soubor, který je poslán klientovi jako odpověď.

Využití webových aplikací je v dnešní době opravdu hojné. Jediné, co uživatel potřebuje ke komunikaci s aplikací je internetový prohlížeč, který je dnes již před instalován v každém uživatelském operačním systému. To ale není jediná výhoda, kterou tyto aplikace nabízejí. Další jsou například:

- Minimální nároky na výkon – díky tomu, že aplikace běží na serveru.
- Aktuálnost aplikace – po jakékoliv úpravě aplikace, na serveru běží vždy aktuální verze. Není tedy třeba nic instalovat ani si hlídat aktuální verzi.
- Centralizace dat – data uložena aplikací na server jsou přístupná z jakéhokoliv jiného zařízení.

- Online funkce – díky centralizaci a snadnému přístupu, se do těchto aplikací snadno přidávají online služby pro více uživatelů, jako například sdílení.

Webové aplikace mají však i své nevýhody. Jsou to mimo jiné:

- Závislost na internetu – aby uživatel mohl komunikovat s aplikací musí být připojen do téže sítě.
- Závislost na vytížení – vzhledem k vytížení sítě může docházet ke zpomalení komunikace.
- Závislost na poskytovateli – pokud se poskytovatel rozhodne ukončit podporu aplikace, uživatel se k jejím službám již nedostane.
- Bezpečnostní rizika – právě díky centralizaci může dojít k útoku na server a následnému úniku dat. [7] [8]

3.1.1 REST API

Rozhraní REST se používá k centralizaci a k snadnému přístupu dat, skrze HTTP požadavky. Požadavek musí být poslán na předem definovaný identifikátor URI (v podstatě URL adresu), se kterou REST API počítá a podle toho se vrátí klientovi odpověď. Existují čtyři druhy těchto požadavků:

- GET – Nejspíše nejpoužívanější požadavek. Slouží k vyzvednutí potřebných dat.
- POST – Slouží k vytvoření / uložení nového objektu na serveru. Často musí být v hlavičce požadavku přidaná autentizace, nejčastěji v podobě tokenu.
- DELETE – Vymaže konkrétní objekt ze serveru.
- PUT – Funguje podobně jako POST, s tím rozdílem, že upravujeme již existující objekt.

Ať už se pošle jakýkoliv požadavek, příchozí odpověď bude obsahovat kódový status, jestli všechno proběhlo v pořádku nebo došlo k chybě, a případně odpovídající data ve formátu JSON. [9]

3.2 Python

Python je dynamicky interpretovaný (překládá se za běhu) programovací jazyk. Díky čemuž je, oproti většině jazykům, velmi lehké se ho naučit. To je asi jedna z největších výhod, díky které je Python poslední dobou velmi rozšířený. Například v nejnovějším průzkumu od stack overflow, jedné z nejznámějších vývojářských stránek, se umístil na 4. místě jako nejpoužívanější programovací jazyk, čímž porazil dokonce i Javu.

Python je vyvíjen jako open-source a jeho jádro je napsané v jazyce C. Nejaktuálnější verze je nyní Python 3. Někteří tvrdí, že Python, díky své rychlosti, která je slabší než u jiných alternativ, není nejlepší volbou. Ovšem s dnešním výpočetním výkonem není jeho rychlost problémem. Přímo naopak se vlivem tohoto aspektu začínají projevovat výhody Pythonu. Především přehlednost, nucená samotnou syntaxí, a jednoduchost jazyka, dohání ztracený čas na výpočtech mnohem rychlejším vývojem a řešením chyb. To činí Python celkově produktivnější než ostatní jazyky, jelikož dnes již není největší nepřítel vývojáře výpočetní výkon ale doba od nápadu přes vývoj až do nasazení softwaru na trh před konkurencí. [10] [11] [12]

3.3 Webové aplikace v Pythonu

Kromě vědeckých a výpočetních účelů, se Python velmi hojně využívá pro vývoj webových aplikací. K vytvoření takovéto aplikace není sice potřeba použití frameworku nicméně není moc běžné, že by je vývojáři nepoužívali.

Samotný Python není schopen běžet v internetovém prohlížeči a i přesto, že projekty jako pyjs dokáží kompilovat z Pyhonu do Javascriptu, většinou se používá kombinace Pythonu běžícího na serveru a Javascriptu, který je stažen klientovi do prohlížeče a instrukce vykonává tam. [13]

3.4 Frameworky pro webové aplikace v Pythonu

Frameworky slouží ke zjednodušení vývojářské práce. Konkrétně k izolování častých a napříč aplikacemi opakujících se procesů tak, aby se o ně nemusel vývojář starat a mohl se tak soustředit na samotnou aplikaci. O tyto záležitosti se pak automaticky stará sám framework, ať už jde o dodržování návrhových vzorů, zabezpečení nebo prostě jen o

běžnou operaci, jejíž funkce je vždy v podstatě stejná a je využívána skoro každou aplikací.

Níže je uveden seznam v poslední době nejpopulárnějších a nejvyužívanějších webových frameworků pro Python. Celý výběr je rozdělen na full-stack frameworky obsahující velké množství funkcí a zdrojů. Na microframeworky – malé, jednoduché a nenáročné. A na asynchronní frameworky založené především na minimalismu a rychlé odezvě. [14] [15]

3.4.1 Django

Django, původně navržen pro systémy správy obsahu, je volně dostupný full-stack Framework pro python. Jeho princip spočívá ve snaze vytvořit pomyslný „box“ obsahující všechny nezbytné funkce a knihovny už ve výchozím stavu. To z něj činí Framework vhodný pro velké projekty využívající nesčetné množství jakýchkoliv prostředků. Jedny z předností tohoto frameworku je integrovaná autentizace, URL směrování, šablonový engine, object-relational mapper (ORM) a migrace schématu databáze. Django ukládá objekty do databáze přes ORM, tudíž stejný kód funguje i s jinými databázemi a lze i velmi snadno mezi nimi přesouvat. Hlavní využívané databáze jsou PostgreSQL, MySQL, SQLite a Oracle, ale dokáže pracovat i s jinými.

Další funkce: web server pro vývoj, šablonový systém pro formuláře, cechování mnoha různými způsoby, převádění do XML nebo JSON.

Hojně využíván díky mnoha předpřipraveným funkcím. Asi nejoblíbenější Framework. [14] [15]

3.4.2 Pyramid

Pyramid je volně dostupný (open source) web Framework pro python. Jeho cílem je poskytnout prostředí s co nejvíce možnostmi a zároveň s minimální komplexitou. Dokáže velmi dobře pracovat jak s malými jednoduchými aplikacemi tak s těmi rozsáhlými. Jeho výhodou je flexibilita. Jednotlivé knihovny a funkce navíc je nutné přidávat externě. Výsledná aplikace pak tedy obsahuje pouze nezbytně nutné součásti, čímž se šetří jak velikost, tak rychlost ale i přehlednost. Funkce: jedno skriptové aplikace, gene-

rování URL, rozšířitelná konfigurace, celoplošné šablonování, flexibilní autentizace a autorizace, Unit testování, bohatá dokumentace, obsahuje dekorátory funkcí. [14] [15]

3.4.3 TurboGears

TurboGears je volně dostupný data-driven web Framework pro Python. Umožnuje rychle vytvářet rozšiřitelné webové data-driven aplikace. Obsahuje uživatelsky přívětivé šablonování a účinné a přizpůsobivé ORM.

Funkce: podpora více databází, architektura MVC (model, view, controller), podpora SQLObject a SQLAlchemy, integrace Kid a Genshi šablonových jazyků, ověřování pomocí FormEncode, integrovaný web server, PasteScript šablony, aplikační knihovna ToscaWidgets pro zjednodušení koordinace mezi frontendem a vývoje serveru, integrované frontend servery založené na WSGI (Paste http server, CherryPyWSGI/http server, …), nástroje příkazového řádku, integrace MochiKit JavaScript knihovny, všechny poskytované možnosti jsou implementovány skrze dekorátory funkcí. [14] [15]

3.4.4 Web2py

Web2py je škálovatelný volně dostupný webový Framework pro Python. Hlavní nevýhoda Web2py je, že nepodporuje Python 3 a vyšší. Ovšem velkou výhodou tohoto frameworku je, že obsahuje svoje vlastní webově založené IDE, které obsahuje editor kódu, debugger a kompilaci i se spuštěním celé aplikace jedním klinutím.

Funkce: není potřeba framework nijak instalovat ani nastavovat, možnost běžet na operačních systémech Windows, Linux/Unix, Google App Engine, AmazonEC2 a poskytuje jakoukoliv webovou službu, která podporuje Python 2.5-2.7 nebo Java+Python, čitelnost více protokolů, integrovaná ochrana proti webovým útokům (cross-site scripting, injection flaws, …), správně použité návrhové vzory, vyhledávání chyb skrze logování, kontrola přístupu dle rolí, podpora pro internacionalizaci [14] [15]

3.4.5 Flask

Flask je mikro framework inspirovaný frameworkem Sinatra Ruby a využívá Werkzeug WSGI toolkit a Jinja2 template. Je vhodný pro menší aplikace s jednoduchými požadavky. Přístupný je pod BSD licencí. Jeho hlavní myšlenkou je vytvoření pevného zá-

kladu pro webové aplikace, z něhož je poté možné použít libovolné rozšíření. Je modulární a přizpůsobivý požadavkům vývojáře. Vhodný pro API. [14] [15]

Funkce:

- Vestavěny vývojářský server a rychlý debugger.
- Vestavěná podpora unit testů.
- Snadné odesílaní RESTFUL požadavků.
- Jinja2 Template.
- Podpora zabezpečení cookies.
- Dodržování normy WSGI 1.0.
- Založen na unicode.
- Možnost přidání jakéhokoliv ORM pluginu.
- Zpracovávání HTTP požadavků.

3.4.6 Falcon

Falcon je webový mikroframework pro malé aplikace, aplikační backendy a vyšší frameworky. Snaží se vývojáře vést k REST konceptu a proto je při práci s ním vhodné mapovat zdroje a stavy do HTTP metod. Falcon je jedním z nejrychlejších webových frameworků pro Python. Není však vůbec vhodný pro službu webových stránek. Jeho použití je na místě spíše při vytváření RESTful APIs. Neobsahuje web server. [14] [15]

3.4.7 Hug

Hug je jeden z nejrychlejších web framewokrků pro Python. Byl vytvořen na základě jiného JSON frameworku a tím je Falcon. Primárně byl navržen k tvorbě API. Podporuje poskytování několika verzí API, automatickou API dokumentaci a ověřování na základě anotací. [14] [15]

3.4.8 Bottle

Bottle je webový mikroframework pro Python. Původně byl navržen pro tvorbu API. Bottle implementuje všechny použité funkce do jednoho zdrojového souboru a nemá

žádné závislosti na standardní knihovně Pythonu. Bottle je vhodný pro vytváření prototypů, učení se organizaci webových frameworků a k budování malých osobních aplikací.

Funkce: směrování – podporuje požadavky na mapování volání funkcí, pro čistší a dynamičtější URL, šablony – rychlý a na Pythonu založený šablonový engine s plnou podporou mako, jinja2 a cheetah, pohodlný přístup k datům z formulářů, cookies, headers, uploadnutým souborům a jiným http metadatům, server – zabudovaný HTTP vývojářský server podporující fapws3, bjoern, GAE, CherryPy a další severy založené na WSGI. [14] [15]

3.4.9 CherryPy

CherryPy je volně dostupný minimalistický webový Framework pro Python. Umožňuje vytváření webových aplikací v Pythonu naprosto stejně jako v kterémkoliv jiném objektově orientovaném programovacím jazyce. V podstatě, aplikace běžící na CherryPy frameworku je samostatná Python aplikace, která si zavádí vlastní více vláknový webový server. CherryPy běží na jakémkoliv operačním systému, který podporuje Python (Windows, Linux, Mac OS, …). Apliakce v CherryPy je možné spustit z libovolného místa stejně jako běžnou aplikaci v Pythonu, není potřeba žádný server jako Apache, Lighttpd nebo IIS. Nicméně je možné aplikaci do takového serveru zasadit. CherryPy nijak neomezuje použité technologie pro šablonování, kontrolu přístupu k datům, …. Ale přesto dokáže zpracovat sessions, statics, cookies, uploady souborů a spousty dalších pro web Framework typických záležitostí.

Funkce: obsahuje HTTP/1.1 v souladu s WSGI webový server se s druženými vlákny, jednoduché spouštění několika HTTP serverů najednou, výkonný konfigurační systém, flexibilní systém pluginů, doplňující moduly pro caching, dekódování, sessions, autentizace, integrovaná podpora pro profilování, pokrytí a testování, možnost aplikace běžet na Python 2.7+, Python 3.1+, PyPy, Jython a Androidu. [14] [15]

3.4.10 Sanic

Sanic je webový Framework pro python založen na uvloop a byl speciálně vytvořen pro rychlé http odpovědi skrze asynchronní vyřizování požadavků. Běží na verzi pythonu

3.5 nebo vyšší a podporuje (asynchronous request handlers) což ho činí kompatibilní s async/wait funkcemi Pythonu verze 3.5, což děla Sanic velmi rychlým. V testech o jednom procesu a 100 připojení byl Sanic schopen vyřídit 33 342 požadavků za vteřinu. [14] [15]

3.4.11 Aiohttp

Aiohttp je webový asynchronní Framework pro python. Využívá, stejně jako Sanic, funkce async/wait v Pythonu od verze 3.5 Aiohttp není jen webovým frameworkem pro server ale i pro klienta, neboť podporuje oba WebSockety, klientské i pro server. Také podporuje integraci šablon Jinja2. [14] [15]

3.4.12 Tornado

Tornado je web Framework pro Python a zároveň asynchronní síťová knihovna. Využívá neblokující síťové vyřizování požadavků a při správném nastavení dokáže obsluhovat více než 10 000 souběžných připojení. Je velmi vhodný pro aplikace vyžadující vysokou výkonnost a vyřizování velkého počtu paralelních požadavků. Tornado oficiálně podporuje pouze Linux a BSD operační systémy. Mac OS X a Microsoft jsou doporučovány pouze pro vývojářské použití.

Funkce: zabudovaná podpora pro autentizaci, služby v reálném čase, vysoká výkonnost, jazyk pro vytváření webových šablon založený na Pythonu, neblokující http klient, implementace schémat pro autentizaci a autorizaci třetích stran (FaceBook, Twitter, ...), podpora lokalizace a překladu [14] [15]

3.5 Výsledné srovnání

Při porovnávání byly důležité především vlastnosti vyvozené ze zadání a požadavků. Zde je menší tabulka shrnující tyto rysy.

Podpora Python 3+: Tato vlastnost je důležitá především pro budoucí využívání aplikace. Pro uchování bezpečnosti a stability je nezbytné, aby využívala nejnovější zdroje, které jsou zrovna vyvíjeny a udržovány.

Databáze: Vlastní databáze nebo práce s ní není nyní až tak důležitá, vzhledem k tomu, že všechna data a uživatelé jsou uloženi v databázi na serveru. Nic méně do budoucna je to aspekt, který se může hodit v případě, že bude potřeba něco ukládat i v rámci aplikace a ne jen serveru. Například při vyšším rozšíření aplikace by mohlo být třeba ukládat uživatelské nastavení.

Jednoduchost a rozšiřitelnost: Tento rys je důležitý pro snazší práci nejen nyní, ale především v budoucím vývoji a rozšiřování aplikace do neznámých nicméně pravděpodobně rozsáhlých mezí.

Github stars & Github forks: Github stars označuje počet uživatelů, kteří momentálně sledují tento projekt na stránce github.com a Github forks značí počet uživatelů, kteří nějakým způsobem přispěli k projektu nebo ho použili jako výchozí bod svého projektu. Tyto dva aspekty jsou důležité především proto, že určují míru oblíbenosti a využívání těchto frameworků a tudíž lze předpokládat, že budou dál vyvíjeny a udržovány.

Knihovna na grafy: Zobrazování grafů je nejdůležitější částí aplikace. V tabulce je porovnání pokud má framework vlastní knihovnu na vytváření grafů, která je ale i zároveň dostačující a splňuje podmínky plynoucí ze zadání. Je to například zobrazení několika os Y, polární graf pro fázorový diagram.

Tabulka 2: Srovnání frameworků

	Python 3	Databáze	Jednoduchost a rozšiřitelnost	Github stars	Github Forks	Knihovna na grafy
Django	Ano	Ano	Ano & Ano	37 409	16 120	Ne
Pyramid	Ano	Ano	Ano & Ano	2 970	828	Ne
TurboGears	Ano	Ano	Ano & Ano	259	59	Ne
Web2py	Ne	Ano	Ano & Ano	1 664	800	Ne
Flask	Ano	Ano	Ano & Ne	39 583	11 534	Ne
Falcon	Ano	Ano	Ano & Ne	5 801	604	Ne
Hug	Ano	Ano	Ano & Ne	5 419	301	Ne
Bottle	Ano	Ano	Ano & Ne	5 790	1 171	Ne
CherryPy	Ano	Ano	Ano & Ano	824	208	Ne
Sanic	Ano	Ano	Ano & Ne	10 580	1 007	Ne
Aiohttp	Ano	Ano	Ano & Ne	6 348	989	Ne
Tornado	Ano	Ano	Ano & Ne	16 741	4 751	Ne

3.6 Finální výběr – Django

Po zvážení všech zmíněných vlastností jsem nakonec vybral framework Django. Plně podporuje Python 3, má vlastní developerskou databázi a zároveň umožňuje snadné připojení kterékoliv jiné, jeho modulárnost poskytuje výborný podklad pro aplikaci, která nyní začíná jako malá ale v budoucnu se pravděpodobně mnohem zvětší a v neposlední řadě je velmi oblíbený a často používaný což zajišťuje určitou podporu do budoucna, ať už ze strany uživatelských řešení či oficiálních vývojářů. V nejnovějším průzkumu od stack overflow, se mezi webovými frameworky umístil na 8. místě a pokud budeme počítat frameworky pouze v Pythonu, pak se umístil dokonce na první příčce. [10]

3.6.1 Knihovna pro grafy

Tuto pasáž ze základu nesplnil žádný z frameworků. Knihovnu, pro vykreslování grafů potřebnou k řešení, buď to nemají vůbec nebo neobsahuje vše potřebné. Byl jsem tedy nucen najít externí knihovnu, která bude co nejlépe splňovat podmínky. Rozhodoval jsem se podle následujících kritérií:

Více druhů grafů: Rozsáhlé možnosti jak vypisovat data je nejen nutností v momentálním návrhu řešení ale také z hlediska budoucího vývoje aplikace a připravenosti k různým druhům výpisů.

Více os Y: Tento aspekt je důležitý při porovnávání různého druhu veličin, které nabývají diametrálně odlišných hodnot.

Licence: Nejen pro aktuální aplikaci ale i do budoucna je vhodné použít free-warovou knihovnu a vyhnout se tak zbytečným finančním nákladům. Je-li to možné samozřejmě.

3.6.2 Plotly js

Na základě definovaných podmínek jednoznačně zvítězila knihovna plotly js, která splňuje všechny výše zmíněná kritéria. Byla vytvořena pro vědecké účely a obsahuje obrovské množství různých grafů, jak ve 2D tak ve 3D. Podporuje libovolný počet os Y. Je velmi oblíbená a často používaná, což znamená podporu a budoucí vývoj. Podporuje velmi silnou uživatelskou škálovatelnost designu grafů. A nejspíše nejdůležitějším

aspektem je, že i se všemi těmito výhodami je šířená pod MIT licencí a její využívání je zcela zdarma, což neplatí u žádné jiné takto rozsáhlé knihovně. Jediná nevýhoda této knihovny je velikost způsobená velkým výběrem grafů. [16]

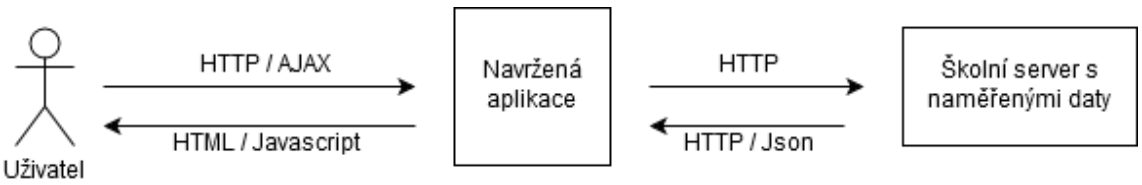
4 Návrh

Následuje mnou navržené řešení. Návrh řešení se odvíjí od zadání a problematiky, aby splňoval co možná nejvíce požadavků a zároveň volil nejvhodnější alternativy. Budu rozebírat rozložení aplikace, návrhy řešení uživatelů, alarmů, grafů a dalších funkcí včetně jejich alternativ.

4.1 Aplikace

Jak už bylo zmíněno výše, pro vývoj aplikace byl vybrán framework Django a to hlavně pro jeho snadné používání a rozšiřitelnost. Celá aplikace pak bude na jedné straně komunikovat s uživatelem a na druhé se serverem. Z uživatelovi strany bude komunikace probíhat HTTP žádostí, na kterou aplikace odpoví vygenerovaným HTML souborem a Javascriptem. Na straně serveru půjde opět o HTTP požadavky a odpovědí na ně budou data ve formátu JSON.

Rozložení samotné aplikace musí splňovat modularitu, kvůli snadnému rozšíření, a obsažení všech důležitých prvků. Mnou navržené a později i aplikované řešení zní následovně. Aplikace bude rozdělena na jednotlivé části, přičemž každá část se věnuje pouze jednomu funkčnímu prvku. Vytvořena bude tedy část pro grafy, obsluhování uživatelů, alarmy, reporty a komunikace se serverem.



Obrázek 1: Schéma funkčního procesu navrhované aplikace

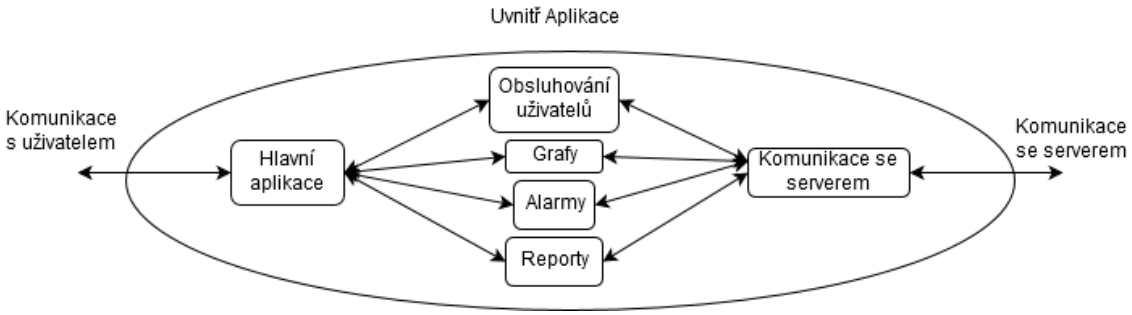
Dalším důležitou technologií, zmíněnou ve výše uvedeným obrázku, je AJAX. Tato technologie umožňuje posílat dynamické HTTP požadavky, skrze Javascript běžící na pozadí bez nutnosti obnovení stránky. Díky této výhodě ho budou i hojně využívat jednotlivé části aplikace, zmíněné níže.

Hlavní aplikace: Hlavní aplikace bude sloužit jako centrální správa nad všemi ostatními moduly. Bude obsahovat úvodní stránku, soubory sdílené mezi moduly a uživatelovi bude nápomocná jako rozcestník do jednotlivých částí.

Komunikace se Serverem: Tuto část je rozhodně nezbytné oddělit od všech ostatních funkcí. Zde bude řešena veškerá komunikace se serverem skrze požadavky na REST API a zpracování JSON odpovědí. K tomuto modulu pak budou mít přístup všechny ostatní, neboť všechny moduly budou využívat data ze serveru. Samotný modul nebude mít žádnou grafickou reprezentaci, avšak dost pravděpodobně bude využíván i skrze AJAX.

Obsluhování uživatelů: Tato část bude zajišťovat přihlášení a odhlášení uživatele. V případě, že se nepřihlášený uživatel pokusí dostat na jinou část než tuto, bude okamžitě vrácen do této části a vyzván, aby se přihlásil. Samotné ověření pak bude probíhat skrze metody v modulu *Komunikace se serverem.* Tato část bude mít vlastní grafické zastoupení v podobě přihlašujícího formuláře.

Grafy: Nejdůležitější část celé aplikace. Bude mít vlastní grafické zastoupení, ve kterém se budou zpracovávat získaná data a ty se pak posléze upraví a vykreslí v podobě různých druhů grafů. Data do grafů bude získávat skrze komunikační modul stejně jako *obsluhování uživatelů*. Na pozadí pak bude využívat i AJAX, zejména pak na aktualizace grafů při monitorování.



Obrázek 2: Vnitřní schéma návrhu aplikace

Alarmy: Modul s alarmy bude sloužit pro správu upozornění. Bude mít vlastní grafickou část a taktéž bude využívat AJAX. Data ohledně alarmů bude čerpat skrze komunikační modul.

Reporty: Možnosti reportů se budou také realizovat odděleně. Nebudou mít grafické zastoupení. AJAX nebude využit přímo ale skrze něj bude možné funkce modulu aktivovat.

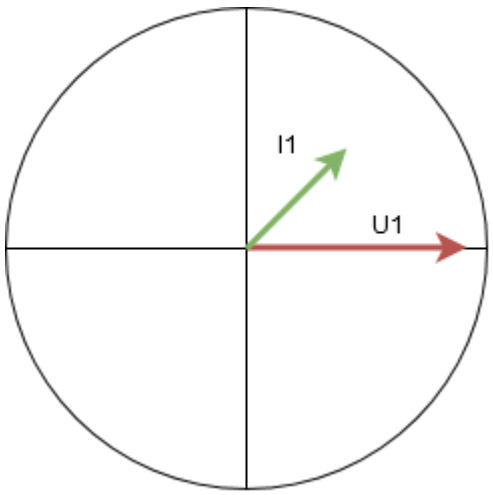
4.2 Návrh výpisů dat

Zpracování a vypisování dat do grafů je pro aplikaci velmi důležitou funkcí. Z dosavadních dat a funkcí, poskytnutých serverem, bude možno vykreslit čtyři grafy. Graf pro výpis dat po minutách, hodinách, fázorový diagram a ukazatel spotřeby s hodinovými záznamy.

Graf po minutách: Tento graf bude vykreslovat jednu nebo více zvolených proměnných z konkrétního zařízení pro vybraný časový úsek. Pro tento účel postačí obyčejný spojnicový graf. Při jeho vytváření bude nutné také počítat s možností vypsání více os Y, v případě, že proměnné budou rozdílného druhu, a tudíž i jednotek.

Graf po hodinách: Zde bude funkcionalita stejná jako u předchozího grafu, s tím rozdílem, že data nebudou v záznamech po minutách ale po hodinách. Také bude použit spojnicový graf a také bude umožňovat více os Y.

Fázorový diagram: Tento graf bude zobrazovat aktuální stav fázových proměnných pro dané zařízení. Vypisovat je pak bude do kruhového grafu s polárními souřadnicemi, kde úhel představuje velikost amplitudy a vzdálenost od středu velikost hodnoty. Ve finále tedy nějak takto:



Obrázek 3: Fázorový diagram

Graf spotřeby: Zde se budou vykreslovat data z proměnné obsahující kompletní spotřebu za určité zařízení. Data jsou evidována a budou i vykreslována po hodinových záznamech s tím, že v grafu budou pouze rozdíly spotřeby za každou hodinu. K těmto účelům bude využit spojnicový graf.

4.3 Alternativy

V této kapitole jsou rozvedeny alternativní návrhy částí mého řešení, objasněn jejich princip, a nakonec odůvodnění jejich zamítnutí.

Alternativa s modulem pro AJAX: Tato alternativa mého návrhu spočívá v myšlence, že by všechny metody zpracovávající asynchronní dotazy byly sdružené do samostatného modulu. Toto řešení by mělo bezpochyby výhodu, že by všechny, v podstatě stejně fungující, metody byly k dispozici na jednom místě. To by však podle mého názoru mohlo způsobovat špatnou orientaci mezi funkcemi, neboť i přes centralizaci metod, by každá sloužila jiným účelům a byla volána jiným modulem. Z tohoto důvodu jsem upřednostnil umístění funkcí, přímo do modulu, jehož účel budou plnit.

Alternativa, kde si každý modul zařizuje komunikaci se serverem sám: Vzhledem k návrhu alternativy s modulem pro zpracování asynchronních dotazů, se nabízí i možnost, že by si komunikaci se serverem zajišťoval každý modul sám, čímž by se dosáhlo seskupení metod se stejným účelem. Tuto alternativu jsem zamítl z důvodu, že server poskytující data aplikace je stále ve vývoji a na případnou změnu v REST API je třeba rychle a co nejsnadněji reagovat, proto je lepší varianta s centralizací veškerých požadavků na server, a nikoliv jejich seskupení dle účelu.

5 Realizace řešení

Zde a v níže zmíněných podkapitolách je uveden postup realizace celého řešení, rozebraného v kapitole předešlé. Týká se samotné aplikace ve frameworku Django, vytvoření jednotlivých částí, jejich napojení, a popisu, jak v budoucnu přidat nové části do aplikace.

5.1 Aplikace v Django frameworku

Realizace byla prováděna na operačních systémech Windows 7 a Windows 8.1. Ale vzhledem k tomu, že Python lze stáhnout a nainstalovat na libovolný operační systém, tak se tento aspekt projeví pouze syntaxí při ovládání frameworku skrze Příkazový řádek.

Při realizaci bylo jako první třeba stáhnout a nainstalovat nejnovější verzi Pythonu (3.7.2) a poté i frameworku Django (2.1.5). Pak jsem po studii oficiálního tutoriálu a dokumentace vytvořil projekt. Tvorba webové aplikace ve frameworku Django je velmi jednoduchá. Po nainstalování stačí pouze otevřít Příkazový řádek a napsat následující příkaz.

```
...\> django-admin startproject mysite
```

Tento příkaz vytvoří do aktuální složky sktrukturu adresářů a souborů zvanou jako *projekt*. Kořenová složka je pak pojmenována *mysite*, jak je patrné v příkazu. *Projekt* pak obsahuje konfigurační soubory, například pro databázi, a soubor manage.py, přes který se, mimo jiné, spouští testovací server. Dále pak osahuje složky jednotlivých *aplikací*, tedy při vytvoření pouze jednu, pojmenovanou stejně jako *projekt*. [17]

Django rozlišuje *projekty* a *aplikace*, přičemž *projektem* se rozumí jeden celek zahrnující libovolné množství menších *aplikací* pod jednou správou. Například tedy vytvořená klientská aplikace. *Aplikace* pak v Djangu znamená část, která se stará o konkrétní řešení. V návrhu jsem tuto část pojmenoval modul. V praxi to pak například znamená správa uživatelů nebo výpis alarmů či jiná konkrétní činnost. Tyto *aplikace* pak nejsou svázány s jedním *projektem*, ale lze je importovat a využívat i jinými *projekty*.

Po přesunutí se v Příkazové řádce do složky s *projektem*, tedy na stejné úrovni jako jednotlivé *aplikace*, pak můžeme zadat následující příkaz pro vytvoření nové *aplikace*. Ukázkově tedy například *aplikaci* pro řízení alarmů.

```
...\> py manage.py startapp alarms
```

Po zadání tohoto příkazu se tedy vytvoří další struktura adresářů a souborů pro konkrétní *aplikaci*, kterou bude projekt využívat. Nejdůležitější jsou soubory *views.py*, *urls.py* a složky *templates* a *static*, přičemž soubor *urls.py* a složku *static* je nutné vytvořit ručně. Tyto položky nejsou povinné, nicméně Django s nimi počítá. [17]

Výchozí aplikace, která se založí ihned po vytvoření projektu, obsahuje soubory *_init_.py*, *settings.py*, *urls.py* a *wsgi.py*. Pro účely aplikace jsou důležité však pouze dva.

Urls.py: Tento soubor obsahuje hlavní seznam veškerých url cest, se kterými bude aplikace počítat a pracovat. Pokud uživatel zadá url adresu, která se neshoduje s žádnou zde uvedenou, bude mu vrácena chyba 404. Nachází se zde jak cesty pro jednotlivé asynchronní požadavky, tak i cesty k jednotlivým aplikacím. Jeden takový záznam z listu url adres může vypadat takto:

```
path('alarms/', include('alarms.urls'))
```

Řádek reprezentuje jednu z url cest poskytnutou aplikací uživateli, kde první atribut „alarms/“ značí část url adresy a druhý atribut pak odkazuje na soubor jedné z *aplikací*, ve které bude Django hledat, pokud bude uživatelem zadaná adresa obsahovat „alarms/“.

Další důležitou funkcí je možnost, pojmenovat si cestu libovolným, avšak doporučuji unikátním, identifikátorem, na který je pak možno se v průběhu psaní aplikace odkazovat, aniž by bylo nutné pamatovat si nebo dokonce vědět celé znění url adresy.

```
path('api/alarms_per_device/$', AlarmsPerDevice.as_view(),
name='alarms-per-device')
```

Ukázka cesty pro asynchronní dotaz využívající pojmenování „alarms-per-device“, na které je možné se později odkazovat v šablonách.

Setting.py: Tento skript obsahuje globální nastavení *projektu,* včetně globálních proměnných, seznamu *aplikací* připojených k *projektu*, nastavení databáze a tak dále. Jedna ze systémových proměnných je například cesta, kde má Django hledat statické soubory nebo proměnná pro vypnutí či zapnutí módu pro odlaďování chyb.

5.1.1 Seznam připojených aplikací

Kromě již předem připojených aplikací jako je administrace, autentizace nebo třeba relace, se do seznamu přidávají i záznamy informující *projekt* o vývojářem vytvořenou *aplikaci*. Záznam je opět jen položka v listu s odkazem na metodu „Config“ ve skriptu „apps“ pod konkrétní aplikací. V praxi tedy takto: „alarms.apps.AlarmsConfig“. Díky tomuto řádku Django ví, že je k *projektu* připnuta aplikace pod složkou „alarms“ a její skript „apps.py“ obsahuje, při vytvoření automaticky vygenerovanou, metodu „AlarmsConfig“, ve které je pak nastavení aplikace.

Kromě těch již automaticky přidaných, seznam realizovaného *projektu* vypadá takto:

Myloginer: Tato *aplikace* reprezentuje část „obsluhování uživatelů“ starající se o správu uživatelů, jak již bylo zmíněno v návrhu řešení.

Alarms: Tato aplikace reprezentuje část z návrhu, která má na starosti alarmy, tedy „alarmy“.

Maingraph: Tato aplikace zastupuje návrhovou část „grafy“.

Rest_framework: Tato aplikace byla stažena a implementována extra pro účely asynchronních dotazů. [18]

5.1.2 Výchozí šablony

Stejně jako většina webových frameworku nejen pro Python, i Django má možnost definování výchozí HTML šablony. Jsou uloženy pod složkou templates, která je na stejné úrovni jako *aplikace* vnořené v *projektu*. A definují hlavní rozložení HTML dokumentu obsahující rozložení stránky, kontrolní nabídku, importované styly a Javascriptové knihovny. Do hlavní šablony lze také na libovolné místa vkládat zarážky, v Djangu nazvané jako bloky. Všechny ostatní, již konkrétní, šablony dokáží tuto hlavní šablonu

rozšířit a doplnit o další prvky, s tím, že se odkazují na výše zmíněné bloky, do kterých je pak, během generování výsledného HTML souboru, konkrétní obsah vložen.

Šablony také umožňují předávat dynamická data z Pythonu do Javascriptu a HTML nebo poskytují základní programové funkce jako jsou cykly nebo podmínky. Pro snazší tvorbu celého rozložení jak základní šablony, tak ostatních zobrazení, byl použit bootstrap importovaný přes internet. [19]

5.1.3 Další funkce

Mezi další využitou funkci patří možnost spustit malý vývojářský server, na kterém pak aplikace běží. Celý Django *projekt* lze umístit na libovolný server a připojit k libovolné databázi, a však pro vývojářské potřeby naprosto stačí integrované nástroje. Server se spustí příkazem:

```
...\> py manage.py runserver
```

K úspěšnému provedení tohoto příkazu je nutné nacházet se ve stejném adresáři jako soubor „manage.py“, neboli na úrovni vnořených aplikací. Ve výchozím stavu bude server dostupný na adrese 127.0.0.1 a portu 8000, pokud je třeba jiné nastavení stačí za příkaz dopsat adresu a port. [17]

Django obsahuje velké množství prvků ulehčující jeho používání, ne všechny se hodí a jsou využity v řešení, ale při učení se s Djangem jsem si je vyzkoušel. Například, v jedné z již před připojených *aplikací* je obsaženo celé administrátorské grafické rozhraní, které má přístup nejen ke správě uživatelů ale i k jednotlivým modelům z *aplikací*. Po správném nastavení *aplikačního* skriptu „models.py“, ve kterém se objektově definují podoby entit uložených do databáze, a po nastavení skriptu „admin.py“, je pak možné přímo v grafickém administrativním rozhraní upravovat záznamy z databáze. A samozřejmě i celé rozhraní je možné si na míru upravit.

5.2 Uživatelé

Tato Django *aplikace* má za úkol řešit problematiku uživatelů. V hlavním souboru celého *projektu* pro evidenci url adres *urls.py* je záznam odkazující na místní soubor *urls.py*, spadající pod tuto aplikaci. V něm jsou pak cesty na výchozí přihlašovací stránku, na

přihlášení a na odhlášení. Tyto cesty pak vedou ke konkrétním metodám v souboru *views.py*, kde je požadavek zpracován a uživateli je pak vrácen HTML dokument *loginer.html*, který dle šablony mění obsah podle stavu uživatele.

Login_index: Tato metoda je zavolána, pokud se uživatel snaží dostat k přihlašovacímu formuláři. K tomu se dá dostat buď odkazem „login“ vpravo nahoře hlavního menu nebo pokusem dostat se na jakoukoliv jinou funkci aplikace, v tom případě bude uživatel n formulář odkázán automaticky. Účelem této metody je pouze vypsat formulář.

Log_user: K této metodě se aplikace dostane pouze při stisknutí přihlašovacího tlačítka formuláře a metoda má pak za úkol autentizaci uživatele vůči serveru. Postup je následující: Algoritmus nejprve převezme přihlašovací jméno a heslo uživatele z formuláře. Pak zkontroluje, zdali jsou obě převzaté položky vyplněné. Pokud nejsou, vygeneruje HTML dokument *loginer.html* a do šablony přidá atribut s chybou. V opačném případě se postupuje dále napojením na modul zajišťující komunikaci se serverem. Zavolá se jeho metoda *user_log*, která v případě úspěšného přihlášení vrátí ověřovací token a v případě neúspěchu prázdný textový řetězec. Pokud je vrácen token, jsou pak do relací uloženy záznamy o jménu, přihlášení a tokenu. Jinak je opět vrácen soubor s chybou.

Logout_user: Tato metoda je zavolána odkazem „logout“ vpravo nahoře hlavního menu, pokud je již uživatel přihlášen. Metoda vymaže všechny relační záznamy s uživatelem a vrátí přihlašovací formulář.

5.3 Komunikace se serverem

Jak již bylo řečeno v návrhu, takto sekce musí být oddělena od všech ostatních, protože se skrze ni vede veškerá komunikace se serverem. Skript, ve kterém je obsažena veškerá komunikace, se nachází mezi skripty v prvotní automaticky vygenerované *aplikaci* a pojmenoval jsem ho *datareceiver.py*. Odtud je pak dostupný všem ostatním *aplikacím*, které ho využívají. Jeho metody jsou:

User_create: Tato metoda posloužila hlavně k vytvoření testovacího uživatele, na kterém probíhal veškerý vývoj. V budoucnu pak může být využita k registračním účelům. Vytvoření uživatele se provede skrze POST požadavek na serverové REST API, ke kte-

rému jsou připojena data ve formátu JSON, obsahující přihlašovací jméno a heslo. Nazpět pak dorazí odpověď, jestli k vytvoření úspěšně došlo nebo ne.

User_log: Tato funkce slouží k přihlášení uživatele. Přijímá dva vstupní parametry, email, který je rovněž přihlašovacím jménem a heslo. Obě informace pak zapouzdří do hlavičky POST dotazu ve formátu JSON. V případě úspěšného přihlášení metoda vrátí ověřovací token.

User_new_token: Vzhledem k tomu, že ověřovací token má omezenou časovou platnost, je třeba ho pravidelně obměňovat. Metoda se tedy pokusí GET požadavkem dostat nový token, přičemž do hlavičky dotazu doplní pole „authorization“ s aktuálním tokenem. Pokud úspěšně obdrží nový token vrátí ho. Pokud ne, zavolá metodu *user_log* a vrátí výsledek této metody.

Get_list_of_tags: Tato metoda pošle GET požadavek doplněný o autentifikační token a vrátí seznam všech tagů napříč serverem.

Get_list_of_devices: Tato funkce vybere seznam všech zařízení připojených k serveru. Celý seznam poté probere a o každém zařízení si vezme jen důležité informace jako je název a identifikátor. Nasbírané data pak poskládá do pole slovníků a to vrátí.

Get_list_of_variables: Tato metoda na vstupu přijme identifikátor připojeného zařízení, jehož proměnné je potřeba zjistit. Pak se vyšle GET požadavek i s tokenem a obdržený seznam proměnných pro konkrétní zařízení se opět projde a vezmou se jen důležité informace, které jsou zase ve stejném formátu vráceny.

Get_tag_for_device: Na začátku přijme identifikátor zařízení, jehož chceme tag. Ten je pak obdržen skrze GET požadavek a na konci metodou vrácen.

Get_list_of_devices_for_phase_diagram: Tato funkce použije nejprve již zmíněnou metodu „get_list_of_devices“ a pro každé zařízení pak metodu „get_list_of_variables“. Ve výsledku pak vrátí seznam zařízení, které mají proměnné potřebné pro výpis fázorového diagramu.

Get_alarms_per_device: Jako vstupní parametr obdrží identifikátor zařízení. Pro něj pak přes GET požadavek vybere seznam alarmů a z nich opět vytáhne pouze důležitá data jako třeba časovou známku poslední změny.

Get_consumption: Metoda, která skrze GET požadavek spolu s tokenem obdrží informace o spotřebě. Z nich jsou pak vytažena jen užitečná data jako naměřené hodnoty, časová známka nebo jednotka měřené veličiny.

Phase_diagram: Na vstupu metoda očekává identifikátor zařízení a dva seznamy proměnných, proudy a napětí. Nejprve získá tag zařízení metodou „get_tag_for_device“. Pak pro zvolené zařízení a jednotlivé proměnné získá hodnoty velikostí a úhlů. Tyto hodnoty poté uloží a vrátí do dvou listů pro proudy a pro napětí.

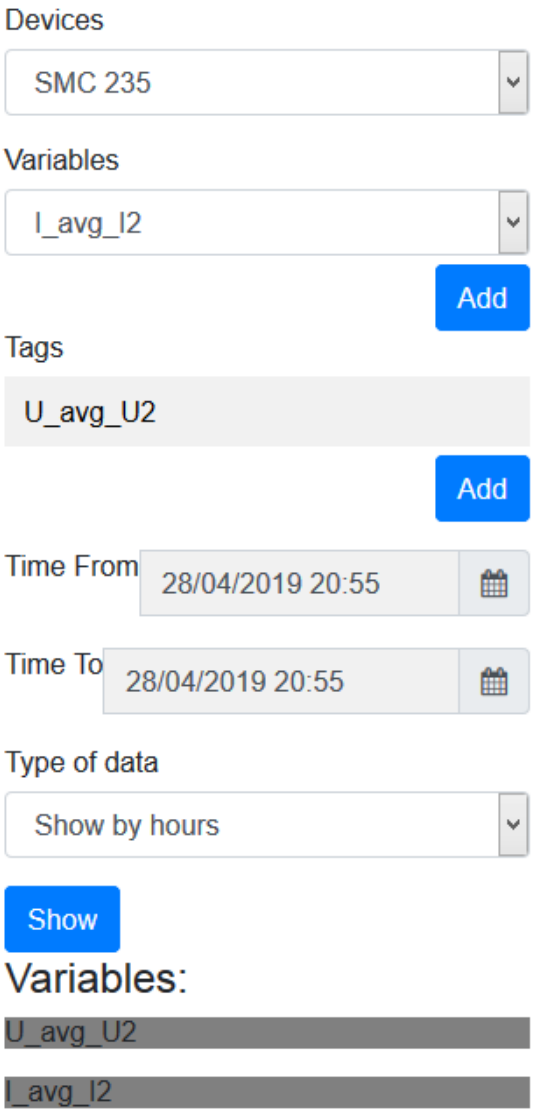
Get_measured_data_minutes: Metoda na vstupu převezme identifikátor zařízení, přes který je ihned na začátku vyslán požadavek pro získání tagu zařízení již umíněnou metodou, dále pak časovou známku pro začátek intervalu, časovou známku pro konec intervalu a seznam proměnných. Poté se projede seznam proměnných a pro každou z nich je poslán GET požadavek s parametry časového intervalu, ve kterém nás data zajímají, tagem zařízení a identifikátorem proměnné. Obdržené hodnoty se pak vyfiltrují a přetransformují do pole slovníků.

Get_measured_data_hours: Funguje na podobném principu jako předchozí metoda, ale slouží k získání dat po hodinách nikoliv minutách. Jediným prvkem navíc je, že vrácené informace serverem o jednotlivých proměnných, můžou obsahovat kromě naměřených zprůměrovaných hodnot také hodnoty maximální a minimální za měřenou hodinu. O tyto dva parametry je tedy i obohacen výstup metody.

5.4 Grafy

Tato *aplikace* má za úkol obsluhovat a vypisovat grafy. V *projektu* je pojmenována jako „maingraph“ a z hlavního url skriptu vede odkaz do místního skriptu *urls.py*, který dle url adresy spouští metody „monitoring“ pro grafy po minutách nebo hodinách, „phase_diagram“ pro výpis fázorového diagramu a „consumption“ pro výpis spotřeby. *Aplikace* také hojně využívá asynchronních dotazů, které jsou mají definovanou cestu v hlavním *urls.py* skriptu a algoritmy pak v místním *views.py* spolu s výše zmíněnými metodami. Pro každý druh grafu pak existuje jeden, který je napojen na komunikační modul se serverem a využívá již zmíněné metody k získání potřebných dat.

Každá z metod „monitoring“, „phase_diagram“ a „consumption“ obsahuje vlastní šablonu pro grafický výpis.



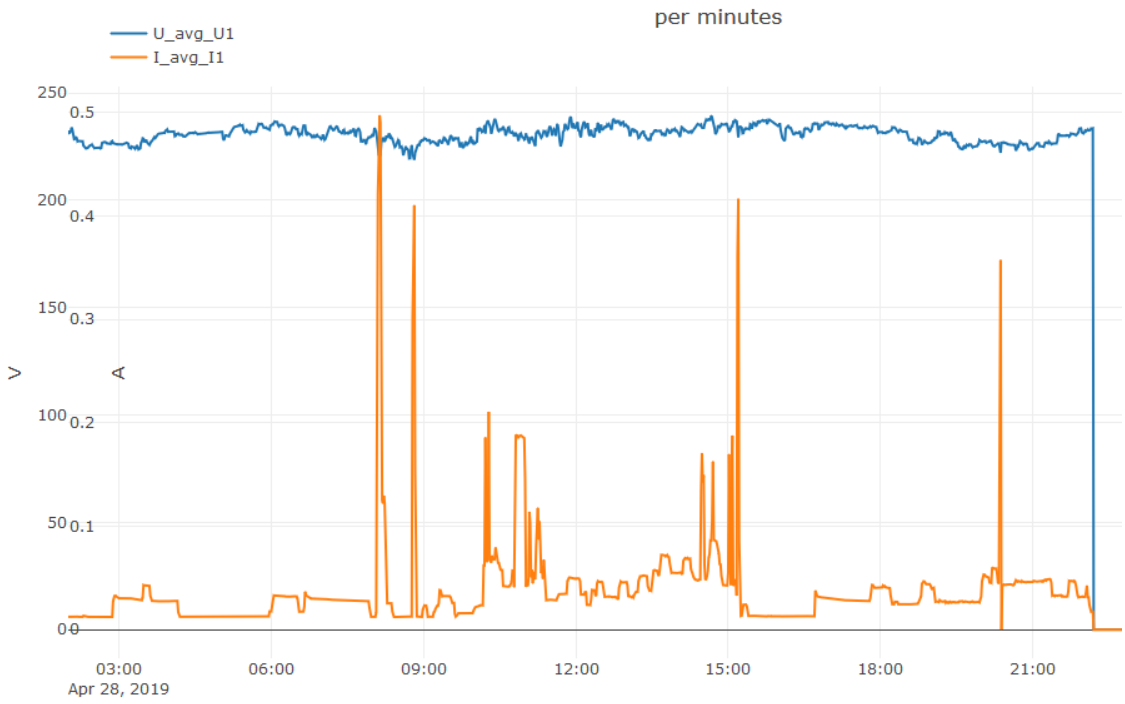
Obrázek 4: Menu pro výpis minutového a hodinového grafu

Na obrázku nahoře je ukázka menu pro výpis dat v grafu po minutách nebo po hodinách. Do výběru se zařízeními („Devices“) je vložen list z metody, která vrací seznam zařízení. Po zvolení zařízení je skrze Javascript a asynchronní dotaz aktualizován výběr s proměnnými („Variables“). V položce „tags“ je seznam veškerých tagů spolu s našeptávačem. [20] Položky „Time From“ a „Time To“ obsahují objekty pro výběr cíleného časové intervalu. [21] Výběr „Type of data“ nabízí možnost vypsat data po minutách nebo hodinách. Tlačítky „Add“ se přidá proměnná do spodního seznamu a ten je

pak odeslán asynchronním dotazem po stisku tlačítka „Show“. Na pozadí je pak aktivována jedna z metod z modulu pro komunikaci se serverem a její výsledek je pak poslán Javascriptovým funkcím, aby připravili data pro výpis do grafu.

5.4.1 Graf pro data po minutách nebo hodinách

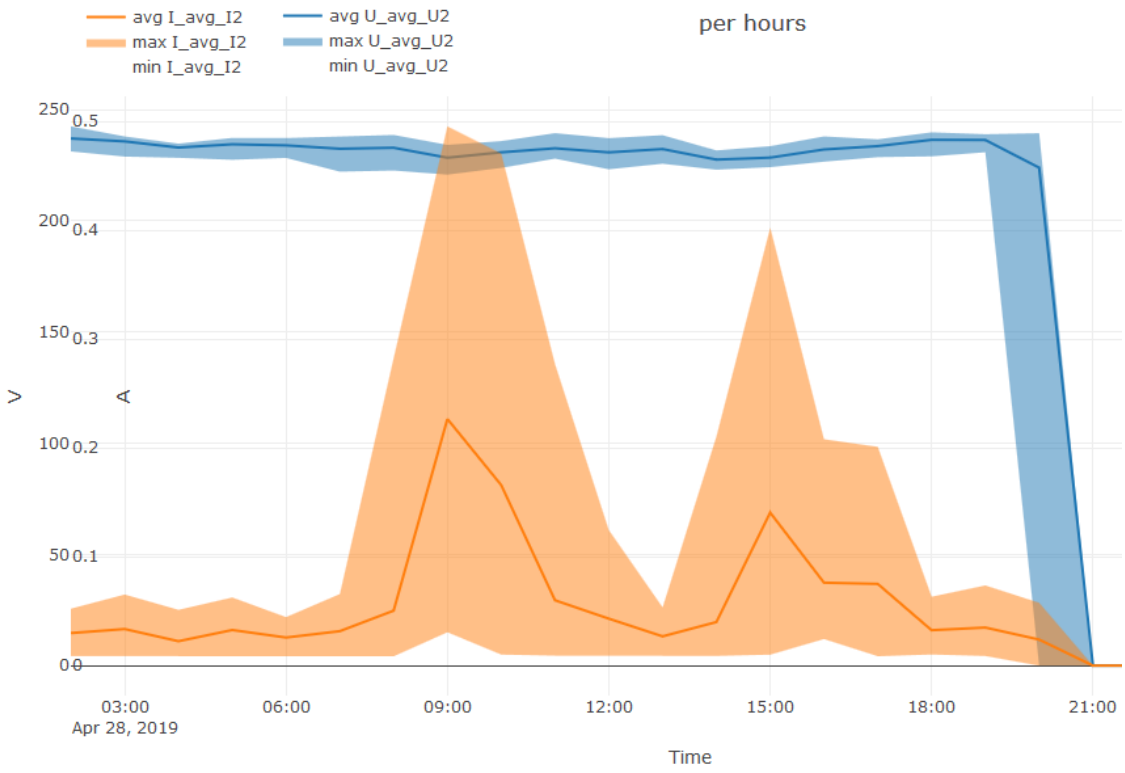
Javascript na pozadí této stránky rozhodne, zdali má aktivovat metodu pro data po minutách či hodinách dle nastavení. Při výpisu dat po minutách se připraví časová osa X podle počínající časové známky, od které se pak generují časové skoky po minutách pro celou délku pole s hodnotami. Osy Y se připraví pro každou proměnnou zvlášť přímým přidělením hodnot obdržených asynchronním dotazem. Při tvorbě os Y se také eviduje seznam jednotek a pro každou z nich se pak vykreslí speciální hodnotová osa Y. Vše za pomocí knihovny plotly.js.



Obrázek 5: Výpis dat po minutách

Při výpisu dat po hodinách je postup stejný jako při minutách akorát s jedním rozdílem. Při přípravě os Y se jednotlivé proměnné vypisují po trojicích: průměrná hodnota, maximum a minimum, přičemž průměrná hodnota se vykreslí jako čára a maximum

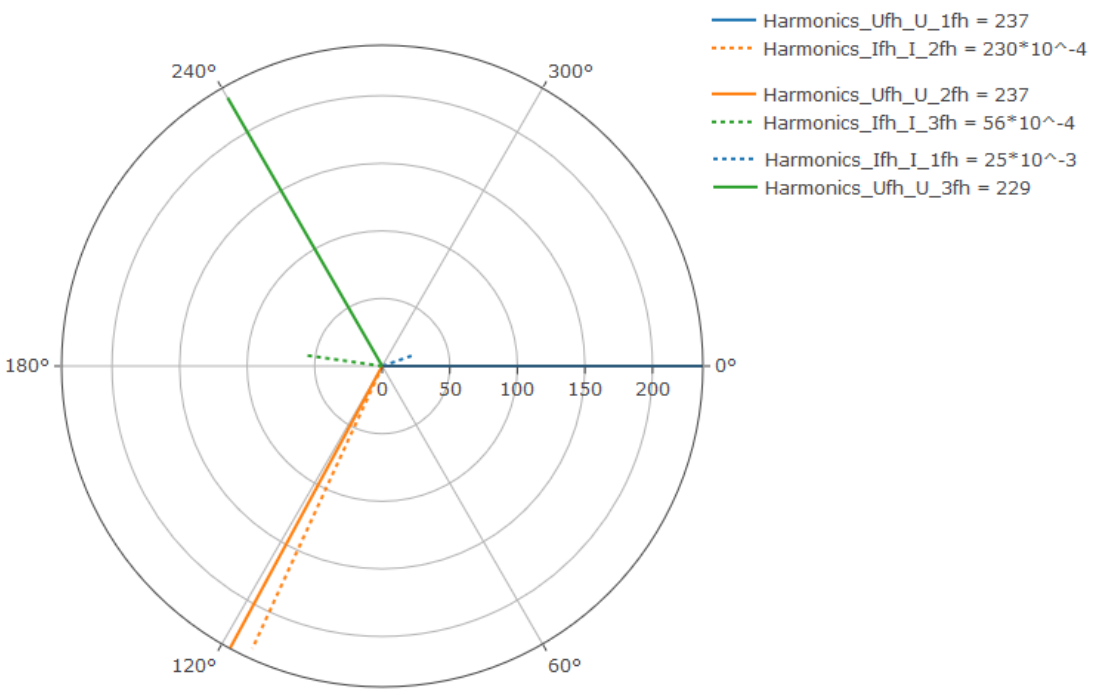
s minimem vykreslí kolem průměru plochu. A časová osa je samozřejmě realizována po hodinových krocích.



Obrázek 6: Výpis dat po hodinách

5.4.2 Fázorový diagram

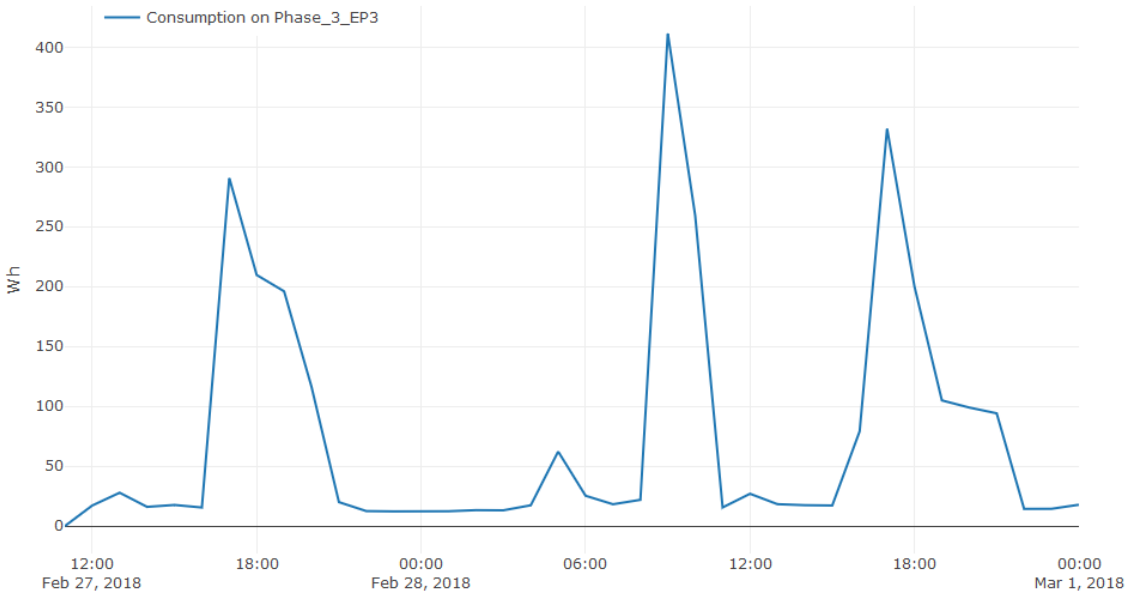
Při vykreslení fázorového diagramu je uživateli nabídnut seznam zařízení, které mají potřebné proměnné pro výpis diagramu. Po zvolení se vyšle asynchronní dotaz, který získá potřebná data a předá je Javascriptu. Ten pak pomocí knihovny plotly.js vykreslí speciální polární graf.



Obrázek 7: Fázorový diagram v polárním grafu

5.4.3 Graf spotřeby

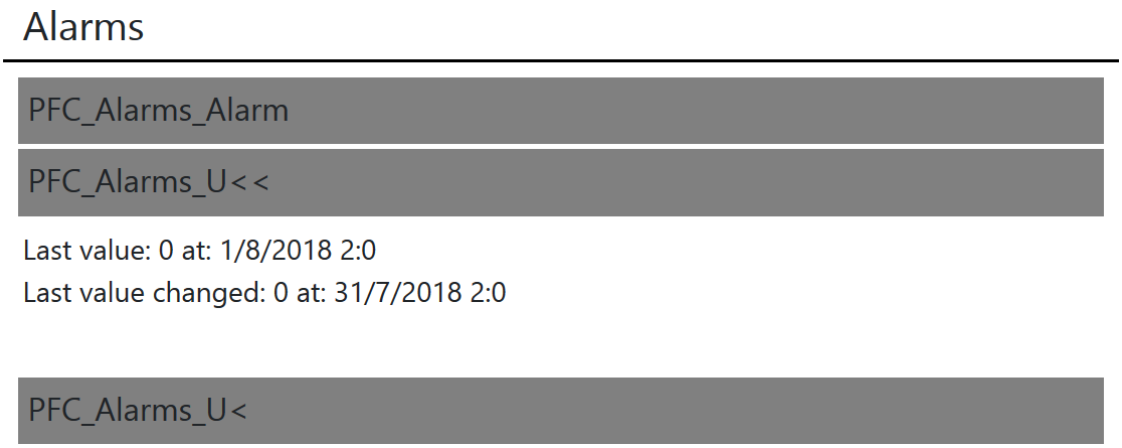
Vykreslení grafu pro spotřebu je realizováno z jedné proměnné a stejným způsobem jako u grafu po hodinách. Rozdíl je ve zpracování dat v Javascriptu, do grafu se totiž nedávají přímo hodnoty, ale jejich rozdíly, aby byla patrná spotřeba za hodinu.



Obrázek 8: Graf spotřeby

5.5 Alarmy

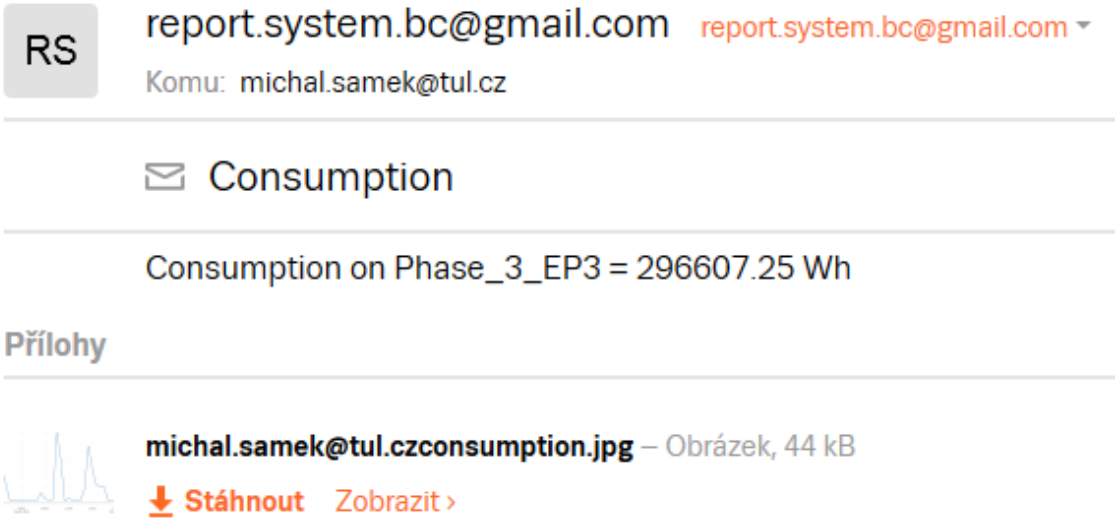
Tato *aplikace* se stará o výpis alarmů. Server zatím jejich správu neposkytuje, takže aktuálně je možné je pouze vypisovat. Při vstupu do aplikace je uživatelovi poskytnut seznam zařízení. Po zvolení jedno z nich se asynchronním dotazem naplní tabulka s jednotlivými alarmy. Pokud zařízení nemá žádný nastavený alarm tak se vyhodí hláška s informačním textem. Jednotlivé alarmy je pak možno rozkliknout a zobrazit jejich nejdůležitější hodnoty.



Obrázek 9: Seznam alarmů

5.6 Reporty

Reporty slouží k poslání výsledku či upozornění konkrétnímu uživateli na email. Pro jejich realizaci nebylo třeba vytvářet celou *aplikaci*, ale postačil pouze skript. Skript jsem pojmenoval „reporter.py“ a je uložený v hlavní automaticky vygenerované *aplikaci*. Obsahuje jednu metodu s názvem „send_report“, kterou aktivuje tlačítko na stránce kde je vypsán graf spotřeby. Asynchronní dotaz pošle obrázek grafu kódovaný v base64 této metodě a ta ho pošle prostřednictvím, pro tyto účely speciálně vytvořeným, emailovým účtem.



Obrázek 10: Ukázka příchozího emailu

5.7 Rozšíření

Jak již bylo několikrát zmíněno, celá aplikace byla vyvíjena s přípravou na budoucí rozšíření. Všechny *aplikace* jsou navrženy modulárně a funkční prvky rozděleny do patřičných metod, které lze snadno modifikovat nebo přidat nové. Částečně je to i zásluha frameworku Django, který svými funkcemi vynucuje dodržování standartních softwarových schémat jako je například model view controller.

V případě přidaní nového grafu je potřeba přidat do hlavního skriptu „urls.py“ novou cestu odkazující na „maingraph\urls.py“ a zde pak na novou metodu v „maingraph\views.py“, tam by se museli přidat i případné metody k asynchronním dotazům. Dále by bylo třeba přidat novou šablonu pro HTML soubor, který by rozšiřoval ten hlavní. V „datareceiver.py“ by musela přibýt nová metoda pro výběr nových dat ze serveru, nebo popřípadě nakombinovat už stávající. A na konec upravit Javascript, uživatelovi vrácenému HTML dokumentu, který by na míru upravil data pro graf a pak ho i vykreslil.

Při obohacení o novou funkci, by musela být vytvořena nová Django *aplikace,* nebo popřípadě skript. A do ní pak postupně přidat již zmíněné soubory a napojení na *projekt*.

6 Závěr

Cílem bakalářské práce bylo vytvořit prototypní klientskou webovou aplikaci v Pythonu, která bude komunikovat s univerzitním serverem, jenž shromažďuje data z elektroměrů. Od něj získaná data pak bude vypisovat do grafů, bude kontrolovat autentizaci uživatelů, vypisovat alarmy a umožňovat posílat reporty. Celá aplikace pak musí být vyvíjena s důrazem na budoucí rozšíření.

Pro tvorbu aplikace byl vybrán framework Django a skrze něj pak byly splněny jednotlivé prvky zadání. Aplikace je schopna komunikovat se serverem pomocí HTTP požadavků a data zpátky přijímá a zpracovává ve formátu JSON. Při komunikaci je taktéž schopná ověřovat uživatele, udržováním autentizačního tokenu, který je uživateli přidělen při úspěšném přihlášení. Přihlášeným uživatelům pak umožňuje vypisovat stavy jednotlivých alarmů ke každému zařízení, které je připojené k serveru a má alarmy nastavené. Dále poskytuje zobrazení dat ze serveru v podobě několika druhů grafů využitím Javascriptové knihovny plotly.js. Uživatel si zvolí, z jakého zařízení chce data sledovat a jaké proměnné chce vypsat do grafu. Ty se pak dle volby vykreslí do grafu po minutách či hodinách pro zvolený časový úsek. Další poskytovaný druh grafu je fázorový diagram, který zobrazuje poslední hodnoty na jednotlivých fázích konkrétního zařízení. Posledním je pak graf znázorňující spotřebu elektrické energie pomocí rozdílů spotřeby za konkrétní hodiny. Z tohoto grafu je pak možné vytvořit report, který pošle hodnotu aktuální spotřeby a obrázek grafu na email uživatele, jenž je i jeho přihlašovacím jménem. V poslední řadě je aplikace, i díky zvolenému frameworku, snadno rozšiřitelná a díky modulárnímu návrhu schopna rychle reagovat na změny REST API.

Vytvořená aplikace se pochopitelně nemůže rovnat komerčním softwarům zmíněných v rešerši, nicméně dle mého názoru splňuje zadání a představuje možný prototyp jedné z budoucích aplikací.

Seznam literatury

[1] Energy Management Software. *softwareadvice.com.* [Online] © 2006-2019 Software Advice, Inc. [Citace: 16. 4 2019.] https://www.softwareadvice.com/cafm/energy-management-comparison/?segments=507&sizes=924&avg_rating=4.

[2] Energy Management Software. *capterra.com.* [Online] Capterra Inc. [Citace: 16. 4 2019.] https://www.capterra.com/energy-management-software/?utf8=%E2%9C%93&sort_options=Highest+Rated.

[3] Software. *Entronix.io.* [Online] Entronix. [Citace: 16. 4 2019.] https://entronix.io/index.php/general/.

[4] *wattics.com.* [Online] Wattics Ltd. [Citace: 16. 4 2019.] https://www.wattics.com/analytics/.

[5] *energyelephant.com.* [Online] EnergyElephant. [Citace: 16. 4 2019.] https://energyelephant.com/.

[6] *Capterra.com.* [Online] Capterra. [Citace: 16. 4 2019.] https://www.capterra.com/energy-management-software/compare/150272-146101-143877/Energy-Elephant-vs-Entronix-EMP-vs-Wattics.

[7] Webová aplikace. *managementmania.com.* [Online] ManagementMania.com, 18. 10 2018. [Citace: 16. 4 2019.] https://managementmania.com/cs/webova-aplikace-web-application.

[8] O webových aplikacích. *helpx.adobe.com.* [Online] Adobe. [Citace: 16. 4 2019.] https://helpx.adobe.com/cz/dreamweaver/using/web-applications.html.

[9] Martin Malý. REST: architektura pro webové API. *zdrojak.cz.* [Online] Devel.cz Lab s.r.o., 3. 8 2009. [Citace: 16. 4 2019.] https://www.zdrojak.cz/clanky/rest-architektura-pro-webove-api/.

[10] Developer Survey Results. *stackoverflow.com.* [Online] Stack Exchange Inc. [Citace: 16. 4 2019.] https://insights.stackoverflow.com/survey/2019#technology.

[11] Humrich, Nick. Yes, Python is Slow, and I Don’t Care. *hackernoon.com.* [Online] Hacker Noon. [Citace: 16. 4 2019.] https://hackernoon.com/yes-python-is-slow-and-i-dont-care-13763980b5a1.

[12] A Speed Comparison Of C, Julia, Python, Numba, and Cython on LU Factorization. *ibm.com.* [Online] IBM, 16. 1 2016. [Citace: 16. 4 2019.] https://www.ibm.com/developerworks/community/blogs/jfp/entry/A_Comparison_Of_C_Julia_Python_Numba_Cython_Scipy_and_BLAS_on_LU_Factorization?lang=en.

[13] Makai, Matt. Web Development. *fullstackpython.com.* [Online] [Citace: 16. 4 2019.] https://www.fullstackpython.com/web-development.html.

[14] Inc., SteelKiwi. Top 10 Python Web Frameworks to Learn in 2018. *hackernoon.com.* [Online] Hacker Moon, 6. 3 2018. [Citace: 16. 4 2019.] https://hackernoon.com/top-10-python-web-frameworks-to-learn-in-2018-b2ebab969d1a.

[15] Primakovski, Bogdan. 17 Best Python Web Frameworks to Learn in 2017. *steelkiwi.com.* [Online] Steel Kiwi. [Citace: 16. 4 2019.] https://steelkiwi.com/blog/best-python-web-frameworks-to-learn/#Flask.

[16] Baaj, Adil. Compare the Best Javascript Chart Libraries. *blog.sicara.com.* [Online] Sicara's blog, 27. 6 2017. [Citace: 16. 4 2019.] https://blog.sicara.com/compare-best-javascript-chart-libraries-2017-89fbe8cb112d.

[17] Writing your first Django app. *djangoproject.com.* [Online] Django Software Foundation. [Citace: 16. 4 2019.] https://docs.djangoproject.com/en/2.2/intro/tutorial01/.

[18] Django REST framework. *django-rest-framework.org.* [Online] Encode OSS Ltd. [Citace: 16. 4 2019.] https://www.django-rest-framework.org/.

[19] Bootstrap. *getbootstrap.com.* [Online] [Citace: 16. 4 2019.] https://getbootstrap.com/.

[20] How TO - Autocomplete. *w3schools.com.* [Online] Refsnes Data. [Citace: 16. 4 2019.] https://www.w3schools.com/howto/howto_js_autocomplete.asp.

[21] Tempus Dominus. *tempusdominus.github.io.* [Online] [Citace: 16. 4 2019.] https://tempusdominus.github.io/bootstrap-4/.