VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

DIPLOMOVÁ PRÁCE

Brno, 2020 Bc. Jan Spurný

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ
DEPARTMENT OF TELECOMMUNICATIONS

SBĚR TELEMETRICKÝCH DAT A JEJICH VYHODNOCENÍ
TELEMETRY DATA COLLECTION AND EVALUATION

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. Jan Spurný**
AUTHOR

VEDOUCÍ PRÁCE — **doc. Ing. Petr Číka, Ph.D.**
SUPERVISOR

BRNO 2020

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Diplomová práce

magisterský navazující studijní obor **Telekomunikační a informační technika**

Ústav telekomunikací

Student: Bc. Jan Spurný ***ID:*** 186192
Ročník: 2 ***Akademický rok:*** 2019/20

NÁZEV TÉMATU:

Sběr telemetrických dat a jejich vyhodnocení

POKYNY PRO VYPRACOVÁNÍ:

Seznamte se sběrem telemetrických dat z meteorologických senzorů. Navrhněte vlastní architekturu pro vzdálený sběr dat z meteorologických stanic s vyžitím protokolu Message Queuing Telemetry Transport (MQTT) a implementujte ji. Při návrhu uvažujte modulárnost zdroje dat, vytvořte a popište metodu přidání dalšího zdroje. K ukládání náměrů použijte úložiště dat MySQL/MariaDB nebo SQLite. Systém sběru dat bude dostupný ve formě zdrojových kódů, ale i jako instalační balíček pro Ubuntu. Navrhněte a vytvořte uživatelský portál v podobě interaktivní mapy a grafů, který bude naměřená data vhodně zobrazovat v interaktivní mapě. Dále implementujte metody pro exportování naměřených dat na portál WeatherUnderground.

DOPORUČENÁ LITERATURA:

[1] KODALI, Ravi Kishore a Venkata Sundeep Kumar GORANTLA. Weather tracking system using MQTT and SQLite. In: 2017 3rd International Conference on Applied and Theoretical Computing and Communication Technology (iCATccT) [online]. IEEE, 2017, 2017, s. 205-208 [cit. 2019-09-14]. DOI: 10.1109/ICATCCT.2017.8389134. ISBN 978-1-5386-1144-9. Dostupné z: https://ieeexplore.ieee.org/document/8389134/

[2] FAKHRAZEEV, Anton R., Alexey Yu. ROLICH a Leonid S. VOSKOV. Big telemetry data processing in the scope of modern Internet of Things. In: 2018 Moscow Workshop on Electronic and Networking Technologies (MWENT) [online]. IEEE, 2018, 2018, s. 1-4 [cit. 2019-09-14]. DOI: 10.1109/MWENT.2018.8337256. ISBN 978--5386-3498-1. Dostupné z: https://ieeexplore.ieee.org/document/8337256/

Termín zadání: 3.2.2020 ***Termín odevzdání:*** 1.6.2020

Vedoucí práce: doc. Ing. Petr Číka, Ph.D.

prof. Ing. Jiří Mišurec, CSc.
předseda oborové rady

ABSTRAKT

Práce pojednává o návrhu a vytvoření aplikace pro sběr a vyhodnocení telemetrických dat. Práce obsahuje návrh, realizaci a popis jak tohoto systému, tak i serveru, způsob příjmu a zobrazení dat. Na základě získaných dat z telemetrických stanic (čidel) a zjištění možností pro zobrazení mapy byl vytvořen základ aplikace v podobě uživatelského rozhraní, administračního rozhraní, logiky návaznosti dat a zobrazení mapového podkladu do aplikace. Grafické rozhraní je vytvořeno pomocí jazyků HTML, JavaScript a CSS, přičemž logika zobrazení je tvořena back-end jazykem PHP s použitím frameworku Laravel a s využitím MySQL databáze.

Výsledkem práce je funkční jak uživatelské, tak administrační rozhraní, funkční způsob příjmu dat z dvou zdrojů (MQTT server, API aplikace), vytvoření metod pro zobrazení mapového podkladu a zpracování dat, obsluhu aplikace funkčními prvky pro nastavení zobrazení dat, funkční administrační část aplikace a vytvoření dvou způsobů instalačního balíčku pro systém Ubuntu.

KLÍČOVÁ SLOVA

MQTT, telemetrie, PHP7.3, Laravel, MariaDB, Mapy.cz, Aplikace, Instalační balíček, Composer, NPM, Linoxová služba, Ubuntu

ABSTRACT

This thesis describes design and creation of application which collects and values telemetry data. Thesis consist of design and implementation which describes whole system, server settings as well as approach how to consume any given data. Corner stone of this application is based on given data from telemetry station (sensors) and projecting them on the map background. Application consist of user interface and logic how to project any given data into the map. Graphics interface is created using HTML, JavaScript and CSS programming languages. Logic itself is programmed by using backend language PHP including Laravel Framework in combination with MySQL database. Result of this thesis is working implementation covering administration user interface, working cooperation of MQTT server and APIs. There are also implemented all necessary parts for enablement of user interface with cooperation with map background layer. Implementation creates two possible ways how to install package necessary to run this app on Ubuntu system.

KEYWORDS

MQTT, telemetry, PHP7.3, Laravel, MariaDB, Mapy.cz, Application, Install package, Composer, NPM, Linux service, Ubuntu

SPURNÝ, Jan. *Sběr telemetrických dat*. Brno, Rok, 69 s. Diplomová práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací. Vedoucí práce: doc. Ing. Petr Číka, PhD.

PROHLÁŠENÍ

Prohlašuji, že svou diplomovou práci na téma „Sběr telemetrických dat“ jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené diplomové práce dále prohlašuji, že v souvislosti s vytvořením této diplomové práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Brno ....

podpis autora

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu diplomové práce panu doc. Ing. Petru Číkovi, Ph.D. a Ing. Ondřeji Krajsovi, Ph.D. za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci.

Obsah

Úvod **12**

1 Komunikační protokoly **13**
1.1 Protokol Constrained Application Protocol . . . 13
1.2 Protokol Extensible Messaging and Presence Protocol . . . 14
1.3 Protokol ZeroMQ Message Transport Protocol . . . 14
1.4 Protokol Message Queuing Telemetry Transport . . . 14
1.5 Výběr protokolu pro realizaci aplikace . . . 15

2 Popis komunikačního protokolu MQTT **16**
2.1 Propojení klienta s Brokerem . . . 17
2.2 Komunikace v prostředí MQTT . . . 19
2.3 Struktura témat . . . 20
2.3.1 Standardní témata . . . 20
2.3.2 Témata začínající znakem $ . . . 21
2.4 Publikování zpráv . . . 21
2.5 Přijímání zpráv . . . 21
2.6 Možnosti nastavení kvality služeb . . . 22
2.6.1 QoS 0 - Nejvýše jednou . . . 22
2.6.2 QoS 1 - Alespoň jednou . . . 22
2.6.3 QoS 2 - Přesně jednou . . . 22
2.7 Brokery pro protokol MQTT . . . 23
2.7.1 Mosquitto broker . . . 24

3 Použité technologie pro aplikaci **25**
3.1 HyperText Preprocessor . . . 25
3.1.1 Framework Laravel . . . 26
3.2 Databázový systém MySQL . . . 26
3.3 HyperText Markup Language . . . 27
3.4 JavaScript . . . 27
3.5 Cascading Style Sheets . . . 27
3.6 Mapové podklady . . . 29
3.6.1 Aplikační rozhraní Mapy.cz . . . 29

4 Návrh aplikace **31**
4.1 Návrh veřejné části . . . 31
4.2 Návrh administrační části . . . 31
4.2.1 Přehled . . . 32

4.2.2 Registrace čidel . . . 33
4.2.3 Čidla . . . 34
4.2.4 Správa uživatelů . . . 36
4.2.5 Logy . . . 36
4.2.6 Správa nastavení . . . 36

5 Realizace aplikace 38
5.1 Spuštění serveru . . . 38
5.1.1 Aplikace Adminer . . . 39
5.2 Spuštění frameworku Laravel . . . 39
5.3 Knihovny pro aplikaci . . . 40
5.3.1 Knihovny spravované pomocí Composeru . . . 40
5.3.2 Knihovny spravované pomocí NPM . . . 41
5.3.3 Knihovna pro zpracování grafů . . . 42
5.3.4 Knihovna pro komunikaci s MQTT serverem . . . 42
5.4 Spouštění webové a MQTT aplikace . . . 43
5.4.1 Routování v rámci spuštění aplikace . . . 43
5.4.2 Konvence pro příjem dat z MQTT . . . 43

6 Architektura webové aplikace 45
6.1 Veřejná část aplikace . . . 45
6.2 Administrační část aplikace . . . 48
6.2.1 Přehled . . . 49
6.2.2 Registrace a přihlášení uživatele . . . 49
6.2.3 Správa čidel . . . 50
6.2.4 Správa veličin . . . 52
6.2.5 Správa nastavení Weather Underground . . . 54
6.2.6 Nastavení emailu . . . 55
6.2.7 Aplikační rozhraní aplikace . . . 55
6.2.8 Nastavení témat pro přijímání dat z MQTT serveru . . . 57
6.2.9 Nastavení MQTT serveru . . . 57
6.2.10 Zabezpečení aplikace při příjmu dat . . . 57

7 Instalační balíček 60

Závěr 62

Literatura 63

Seznam symbolů, veličin a zkratek 67

Seznam příloh **68**

A Obsah přiloženého souboru ZIP **69**

Seznam obrázků

2.1 Komunikace mezi klientem a Broker serverem 17
2.2 Struktura paketu MQTT CONNECT 17
2.3 Struktura MQTT CONNACK paketu 18
2.4 Komunikace mezi klienty a Brokerem serverem 19
2.5 Ukázka definice celého tématu 20
2.6 Ukázka definice zástupného znaku hastag 20
2.7 Ukázka definice zástupného znaku plus 20
2.8 Ukázka publikace zprávy MQTT pomocí příkazového řádku 21
2.9 Ukázka přijmutí zprávy MQTT pomocí skriptu 21
2.10 Znázornění komunikace při úrovni QoS 0 22
2.11 Znázornění komunikace při úrovni QoS 1 23
2.12 Znázornění komunikace při úrovni QoS 2 23
3.1 Ukázka mapových podkladů od společnosti Seznam.cz 30
4.1 Grafický návrh veřejné části aplikace 32
4.2 Administrace - Přehled 33
4.3 Administrace - Registrace čidel 34
4.4 Administrace - Čidla základní zobrazení 35
4.5 Administrace - Čidla detailní zobrazení 35
4.6 Administrace - Správa uživatelů 36
4.7 Administrace - Přidání nového uživatele 37
5.1 Ukázka části souboru composer.json 40
5.2 Ukázka proběhlé aktualizace knihoven pomocí Composeru 41
6.1 Výsledná realizace veřejné části aplikace na serveru 45
6.2 Metoda getMapaAStanice() 47
6.3 Výsledná realizace zobrazení statistik v grafu 48
6.4 Administrační část - Přehled 49
6.5 Administrační část - Uživatelé 50
6.6 Administrační část - Uživatelé validace 51
6.7 Administrační část - Čidla 51
6.8 Administrační část - Čidla editace 52
6.9 Administrační část - Čidla nalezení souřadnic 53
6.10 Administrační část - Veličiny 53
6.11 Registrace čidla na portále Weather Underground 54
6.12 Část výpisu služeb pomocí příkazu php artisan list 55
6.13 Administrační část - Nastavení emailu 56
6.14 Administrační část - Nastavení API 56
6.15 Administrační část - Nastavení témat 57

6.16 Administrační část - Nastavení serveru MQTT 58
6.17 Administrační část - Nastavení zabezpečení 59

Seznam tabulek

1.1 Porovnání komunikačních protokolů 13
2.1 Návratové hodnoty v paketu CONNACK 19
2.2 Úrovně QoS v MQTT 22
2.3 Porovnání MQTT dostupných brokerů 23

Úvod

Diplomová práce se zabývá tématem sběru telemetrických dat a jejich vyhodnocením. Jejím cílem je navrhnout systémové získávání dat, jejich zpracování a zobrazení. Telemetrická data z čidel jsou data, která informují o aktuálním stavu počasí v dané lokalitě. Díky uchování těchto hodnot v pravidelných intervalech můžou být vytvářeny statistiky a díky sledovaným veličinám vyhodnocovány změny v počasí v dané lokalitě v průběhu času.

Aplikací podobnými vlastnostmi je například Weather Underground [1], která má ovšem omezené možnosti definic vlastních typů čidel a možnosti nastavení jednotlivých parametrů, které čidlo poskytuje. Další nevýhodou výše uvedené aplikace je, že je možné do ní přispívat pouze jedním způsobem, a to pomocí jejího aplikačního rozhraní.

Na vyvíjenou webovou aplikaci jsou kladeny následující požadavky: možnost zobrazení umístění jednotlivých čidel v mapě, zobrazení posledních přijatých dat ve vizitce čidla, zobrazení historie a vytvoření statistik dat v podobě grafů na bázi několika časových intervalů (hodinové, denní, týdenní, měsíční) nebo případně nastavit odesílání informací emailem o nepřijetí dat z čidla po deklarované době a nastavení. Dále by bylo vhodné, aby aplikace uměla přijímat data alespoň ze dvou zdrojů, tedy pomocí standardizovaného komunikačního protokolu a zároveň i pomocí aplikačního rozhraní. Administrátor aplikace by mohl upravovat definici jednotlivých čidel, nastavovat jejich zobrazení a případné i zakázat přispívání čidla. Výše uvedeným požadavkům neodpovídá žádná realizovaná aplikace.

Diplomová práce popisuje všechny použité technologie, návrh architektury, grafický návrh webové aplikace, realizaci aplikace a tvorbu instalačního balíčku.

1 Komunikační protokoly

Pro přenos zpráv z čidla k serveru s aplikací je nutné využít komunikační protokol. Komunikační protokol zaručuje standardizovanou komunikaci mezi čidlem a aplikací pro sběr dat. Při nedodržení komunikačního protokolu se může stát, že komunikace nebude správná. Mezi využívané protokoly řadíme například COAP, XMPP, MQTT, ZeroMQ. Porovnání jednotlivých zmiňovaných protokolů popisuje Tab. 1.1. [2]

Název protokolu	Transportní protokol	Zabezpečení	Typ spojení
CoAP	UDP	Ano	Tree
XMPP	TCP	Ano	Client-Server
MQTT	TCP	Ano	Client-Server
ZeroMQ	UDP	Ano	P2P

Tab. 1.1: Porovnání komunikačních protokolů

1.1 Protokol Constrained Application Protocol

Protokol Constrained Application Protocol (CoAP) je určený především pro nízko výkonná zařízení. Je definovaný v standardu RFC 7252. Používá se zejména pro komunikaci mezi zařízeními ve stejné síti nebo pokud jsou zařízení v různých sítích spojeny sítí internet. Je určený zejména pro výměnu dokumentů. Tento protokol pracuje na aplikační vrstvě. Pro příjem zpráv využívá systém žádosti a odpovědi nad UDP protokolem. Je možné jej jednoduše přeložit do protokolu HTTP (hypertext transport protocol) pro snadnou integraci v rámci internetu. Využívá se pro zařízení Internet of Things (IoT) a Machine-to-Machine (M2M), kvůli jeho minimálním nárokům a snadné implementaci. Pro komunikaci využívá podobně jako HTTP metody GET, POST, PUT a DELETE. CoAP zprávy jsou zapsány v binárním formátu a mají délku minimálně 4 bajty. Má dva typy zpráv a to odpovědi a požadavky. Vzhledem k závislosti na protokolu UDP může mít celá zpráva velikost maximálně jeden. V protokolu CoAP lze využít také multicastové zprávy. To v konečném důsledku znamená, že namísto posílání zprávy každému jednotlivému klientovi zvlášť, zašleme jednu multicastovou zptávu do sítě, kde jsou připojeni klienti využívající CoAP protokol. [3]

1.2 Protokol Extensible Messaging and Presence Protocol

Extensible Messaging and Presence Protocol (XMPP) je open-source protokol, který je založený na posílání zpráv pomocí XML. Použití struktury XML umožňuje velmi rychlou výměnu dat mezi více zařízeními v síti v reálném čase. Původně byl navržen jako rychlý komunikační protokol pro síť Jabber. Protokol XMPP pro navázání komunikace využívá protokol TCP. Pro komunikaci mezi servery má vyhrazen port 5269, komunikace mezi klientem a serverem je realizována přes port 5222. Lze využít i variantu přes protokol HTTP, kde si můžeme vybrat ze dvou způsobů. První z nich je dotazování pomocí metod GET a POST, druhá možnost je dotazování vazbou, jež je založena na obousměrném toku. Protokol XMPP využívá provoz klient-server Jeho komunikace je obousměrná, má pevně definovanou strukturu dat ve zprávě a nepodporuje nastavení kvality služeb (QoS). Při komunikaci má vysokou režii dat při připojení většího počtu klientů. Každé zařízení připojené pomocí protokolu XMPP má unikátní adresu XMPP. Protokol umožňuje autorizaci i šifrování a je možné použít i šifrování end-to-end, což je zásadní výhoda při použití u aplikací, u nichž požadujeme vysokou míru zabezpečení. [4, 5]

1.3 Protokol ZeroMQ Message Transport Protocol

Protokol ZeroMQ Message Transport Protocol(ZeroMQ) využívá technologii P2P (Peer-to-Peer), což umožňuje komunikaci mezi dvěma klienty bez nutnosti centrálního prvku. Využívá transportní protokol TCP. Zprávy odesílá asynchronní metodou. Pracuje na principu žádost – odpověď, díky nim se propojují klienti. Nebo je možné využít vzor distribuce dat, kdy se propojují vydavatelé a odběratelé. Protokol ZeroMQ implementuje protokol ZMTP, díky němuž jsou definovaná pravidla pro rozšířené bezpečnostní mechanismy, metadata připojení. [6, 7]

1.4 Protokol Message Queuing Telemetry Transport

Protokol Message Queuing Telemetry Transport (MQTT) je jednoduchý a nenáročný protokol pro předání zpráv mezi stanicemi přes centralizovaný bod (broker). Díky své nenáročnosti na procesor je v hodný pro realizaci přenosu zpráv ze zařízení, Které nepotřebují pro svoji činnost výkonný procesor. Přenos probíhá pomocí protokolu TCP a využívá architekturu klient-server. Zprávy v protokolu nejsou vázané na datový typ. Jednotlivé typy zpráv se dělí do témat, které pak lze jednoduše filtrovat. Protokol lze zabezpečit například pomocí TLS, je zde také možné využít řízení

kvality služeb (QoS) nebo implementovat statistická témata pro navázané spojení. Výhodou je, že klient může být zároveň přispěvatel i příjemce zpráv a tím upravovat své chování v závislosti na ostatních klientech. [8]

1.5 Výběr protokolu pro realizaci aplikace

Na základě výše uvedených specifik byl pro realizaci této aplikace vybrán protokol MQTT. Rozhodujícími důvody jsou zejména možnost využití QoS a možnost zabezpečení a navazování spojení pomocí protokolu TCP. Další výhodou je připojení klientů k centrálnímu bodu, kde lze využít ověřování klientů a filtrování zpráv.

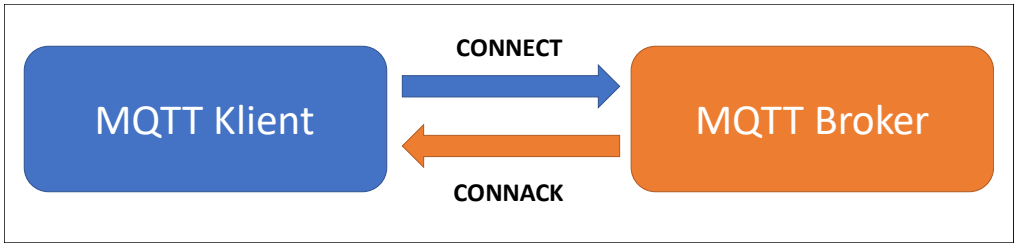
2 Popis komunikačního protokolu MQTT

Protokol MQTT byl navržen Arlenem Nipperem (Arcom, nyní Cirrus Link) a Andy Stanford-Clarkem (IBM) v roce 1999. Jeho účelem bylo řízení ropovodů, ale díky své jednoduchosti, nenáročnosti a snadnému implementování, se začal používat pro menší realizace (domácí). V roce 2013 se tento protokol zařadil do standardu jako open source protokol vhodný pro komunikaci mezi zařízeními.

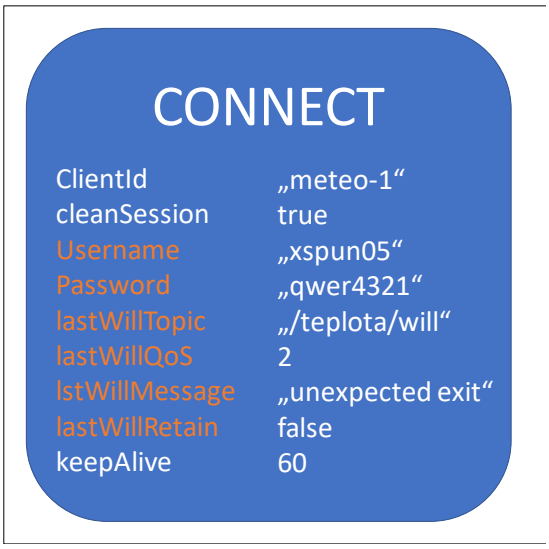
Protokol MQTT pracuje na principu vysílání zpráv od zařízení (klientů) do jednoho centrálního bodu (brokeru). Zprávy jsou s binární hlavičkou a textovým obsahem. V těle zprávy (payload) je možné přenášet textovou zprávu, číselnou hodnotu, nebo datovou strukturu například v podobě JSON pole. Zprávy se nikdy nepřenáší mezi klienty přímo, vždy je nutné, aby mezi nimi byl mezilehlý centrální prvek a to broker. Broker je serverová aplikace, která má za úkol přijímat zprávy MQTT a dále je přeposílat, zpracovávat a filtrovat. Pro odesílání zpráv musí být definovaný takzvaný Publisher (vydavatel) s metodou publish. Publisher má na starosti spojení s brokerem a odesílání zpráv. Zprávy jsou odesílány metodou publish na stanovený broker. Každý klient spadá pod předem definovaný broker. Pro přijímání zpráv z brokeru je nutné mít Subscriber (přijímač) s metodou subscribe. Subscriber má na starosti spojení s brokerem a přihlášení k odběru dat. Pomocí metody subscribe se klientská stanice připojuje k brokeru. Broker se stará o výměnu zpráv mezi Publishery a Subscribery, jejich filtrování a také určuje, kdo je přihlášen k odběru konkrétních témat. Každá zpráva má svoje téma. Tohle téma je zásadní pro zpracovávání zpráv, jelikož pomocí něj může subscriber, který přijímá zprávy z brokeru, přeposlané zprávy filtrovat. Subscriber může přijímat několik témat zároveň, záleží pouze na nadefinování, popřípadě může přijímat všechna témata bez rozdílu. Klient může být současně jak publisher, tak subscriber. [8]

2.1 Propojení klienta s Brokerem

Klienti se v případě nešifrované komunikace pomocí protokolu TCP připojují k brokeru pomocí portu 1883. Pokud se jedná o šifrovanou komunikaci pomocí TLS, broker k připojení využije port 8883. V rámci připojení klienta k brokeru je možné, aby broker vyžadoval autentizaci pomocí kombinace přihlašovacího loginu a hesla. Při navazování komunikace mezi klientem a brokerem se přenáší zpráva od klienta k brokeru. Broker odpovídá po zdárném navázání spojení zprávou a stavovým kódem. V rámci posílá klient cleanSession což znamená, že maže svoje předešlé záznamy. Pokud je klient připojený k brokeru, spojení se stále udržuje, dokud není ukončeno ze strany klienta stavovým příkazem k odpojení nebo zrušení připojení. [9]



Obr. 2.1: Komunikace mezi klientem a Broker serverem



Obr. 2.2: Struktura paketu MQTT CONNECT

ClientId jedná se o identifikátor každého MQTT klienta, který se chystá připojit k brokeru. Je důležitý pro identifikaci a aktuální stav, z tohoto důvodu musí být jedinečný v rámci realizace připojených klientů na jeden broker.

CleanSession nastavuje uchovávání dat pro klienta. Pokud je nastavena hodnota false, dáváme zprávu, aby broker pro tohoto klienta ukládal z předchozích relací. Pokud je nastavena hodnota true, jedná se o zprávu neukládání dat a smaže veškeré předchozí záznamy.

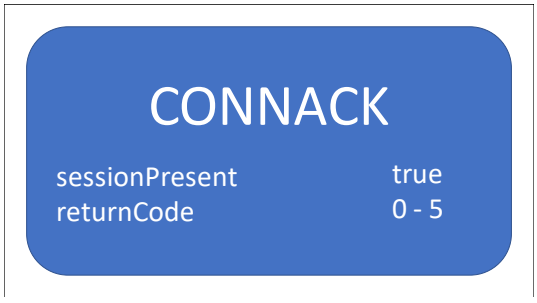
Username a Password je nepovinný parametr a je určený k autentizaci klienta k brokeru. Pokud není spojení šifrováno (využití TLS), je heslo odesíláno jako prostý text.V případě, že požadujeme autentizaci a autorizaci připojených klientů, je vhodné využívat zabezpečený přenos, případně využít autentizační certifikát SSL. Potom už není potřeba využívat přihlašovací jméno a heslo.

Zprávy Will jsou zprávy tzv. „poslední vůle“. Pokud se klient neočekávaně odpojí, aplikují se Will požadavky.

KeepAlive neboli zachování na životě, je časový interval udávaný v sekundách, kterým klient specifikuje nejdelší možnou dobu mezi jednotlivými zprávami od klienta. Klient zasílá brokeru pravidelné zprávy typu Request a broker odpovídá PING Response. Díky těmto zprávám mají obě strany informaci o tom, zda druhá strana je stále k dispozici.

Pokud broker dostane zprávu CONNECT má povinnost odpovědět na ni odpovědět zprávou CONNACK. Zpráva CONNACK obsahuje dva parametry (Obr. 2.3) [10, 11, 12]:

- sessionPresent (příznak současné relace) příznak dává klientovi informaci, zda má broker k dispozici data z jejich předchozího připojení. Pokud klient při žádosti posílá cleanSession true, vrací se hodnota false. Pokud se vrací hodnota true, znamená to, že broker má uložená nějaká data z předchozích spojení.
- returnCode (potvrzovací hodnota)je návratový kód, který dává klientovi zprávu o úspěšnosti či neúspěšnosti připojení.



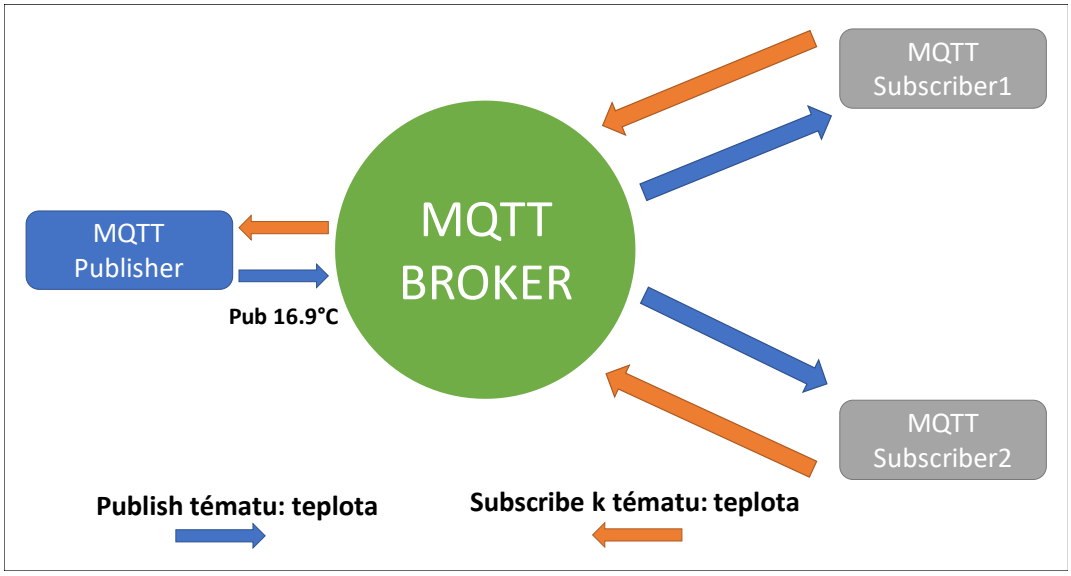
Obr. 2.3: Struktura MQTT CONNACK paketu

Návratový kód	Význam
0	Připojení bylo přijato
1	Připojení odmítnuto, nepřijatelná verze protokolu
2	Připojení odmítnuto, identifikátor odmítnut
3	Připojení odmítnuto, server není k dispozici
4	Připojení odmítnuto, chybné login/heslo
5	Připojení odmítnuto, není autorizováno

Tab. 2.1: Návratové hodnoty v paketu CONNACK

2.2 Komunikace v prostředí MQTT

Architektura pro předávání zpráv vychází z modelu klient-server. Klientem v tomto modelu je zařízení poskytující informace, server je pak broker, který přijímá a dále distribuuje zprávy pro klienty, kteří mají přihlášen tzv. odběr témat (Obr. 2.4).



Obr. 2.4: Komunikace mezi klienty a Brokerem serverem

Klienti, kteří jsou připojeni k brokeru, mohou požádat zprávou o odběr topiců (témat). Proces přihlášení a odhlášení odběru probíhá v následujících krocích:

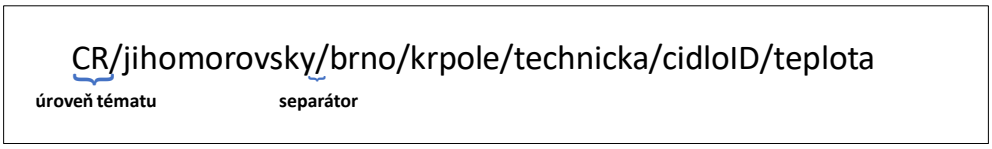
1. Klient zašle zprávu s názvem tématu k brokeru.
2. Broker po úspěšném přijmutí zprávy od klienta a zaregistrování odpovídá zprávou.
3. Pro odregistrování přijímání tématu klient odesílá směrem k brokeru.
4. Broker odpovídá potvrzovací zprávou 0.

2.3 Struktura témat

Témata můžeme dělit na dvě skupiny a to standardní a obslužná. Standardní témata nesou dat, obslužná témata začínají znakem $ a nesou informace o spojení a statistikách.

2.3.1 Standardní témata

Téma v protokolu MQTT se skládá z několika úrovní, které jsou odděleny lomítkem. Díky tomu je jednoduché rozpoznat a rozdělit jednotlivé klienty.



Obr. 2.5: Ukázka definice celého tématu

Na obrázku 2.5 je vytvořené téma, které se skládá ze sedmi úrovní. Přičemž každá úroveň dané téma specifikuje detailněji.

Pro přihlášení ke všem tématům z vyšší úrovně je možné použít zástupný znak „#“ (Obr. 2.6). Z důvodu použití zástupného znaku „#“ budou všechny nižší úrovně zpracovávány brokerem. V praxi to znamená, že pokud budeme mít v rámci tématu krpole více úrovní, všechny budou zpracovávány.

Obr. 2.6: Ukázka definice zástupného znaku hastag

Další zástupný znak je „+“. Tento znak nahrazuje úroveň tématu (Obr. 2.7). Nahrazení úrovně v praxi znamená, že místo znaku „+“ může být jakékoliv téma, na kterém se dotazujeme na dále již specificky určeném tématu.

Obr. 2.7: Ukázka definice zástupného znaku plus

2.3.2 Témata začínající znakem $

Tato témata nejsou součástí Subscribe ani v případě, kdy použijeme zástupný znak #. Jedná se o interní statistiky pro broker. Žádný klient nemůže dané téma publikovat. Tyto zprávy nejsou standardizované a z tohoto důvodu si je každý může upravit dle svých potřeb. Byla vytvořena knihovna, která je pod volnou licencí a tyto zprávy „standardizuje“. [13]

2.4 Publikování zpráv

Klient protokolu MQTT má možnost publikovat zprávy, které mají standardizovanou strukturu. Zpráva Publish může svá data posílat binárně, textově, pomocí pole JSON, nebo ve struktuře XML. V závislosti na požadavcích se volí typ těla zprávy (payload). Každá Publish zpráva v sobě nese typ kvality služeb, téma, ke kterému se váže, hodnotu a svoje unikátní packetId. Odesílání probíhá na broker, který po navázání spojení tyto zprávy přijímá. Pokud je na stanici odkud chceme odesílat nainstalovaný MQTT je možné funkčnost ověřit spuštěním příkazu v příkazovém řádku. Pro tuto realizaci byl vybrán klient Mosquitto. Kde příkaz mosquitto_pub vyvolá publikaci zprávy, parametr -h označuje cílového hosta, kde je provozován broker (v našem případě pro testování byl nainstalovaný na stejném serveru - tedy localhost), parametr -t označuje téma a -m hodnotu (Obr. 2.8).

```
mosquitto_pub -h localhost -t CR/jihomoravsky/teplota -m "16.9"
```

Obr. 2.8: Ukázka publikace zprávy MQTT pomocí příkazového řádku

2.5 Přijímání zpráv

Klient, který je přihlášený k odběru témat od brokeru může přijímat zprávy, které broker poskytuje. V rámci práce byl vytvořen testovací skript, který ověřuje funkčnost připojení na broker a registraci. Vypisuje časovou značku přijaté zprávy, téma na které bylo přijaté a hodnota (Obr. 2.9).

```
Msg Recieved: Sat, 09 Nov 2019 16:52:23 +0100
Topic: CR/jihomoravsky/teplota

        16.9
```

Obr. 2.9: Ukázka přijmutí zprávy MQTT pomocí skriptu

2.6 Možnosti nastavení kvality služeb

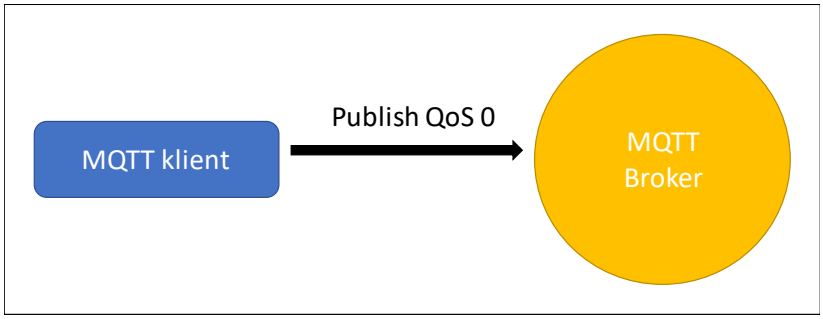
Protokol MQTT umí zaručit kvalitu služeb (QoS). Jsou definovány 3 stupně pro konkrétní zprávu. V tabulce 2.2 jsou jejich definice.

Úroveň	Význam
0	Nejvýše jednou
1	Alespoň jednou
2	Přesně jednou

Tab. 2.2: Úrovně QoS v MQTT

2.6.1 QoS 0 - Nejvýše jednou

Jedná se o nejnižší stupeň kvality služeb. Záznam se nikde neukládá, pouze se odešle od klienta k brokeru. Není tudíž možné jej později odeslat znovu, záruku doručení zabezpečuje pouze protokol TCP (Obr. 2.10).



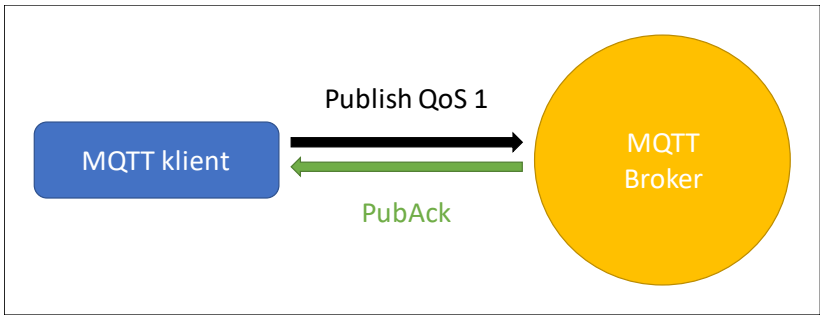
Obr. 2.10: Znázornění komunikace při úrovni QoS 0

2.6.2 QoS 1 - Alespoň jednou

Zaručuje, že zpráva musí být alespoň jednou doručena k brokeru. Klient ukládá zprávu a čeká, až obdrží zpětnou zprávu PubAck od brokeru, který tím stvrzuje doručení (Obr. 2.11).

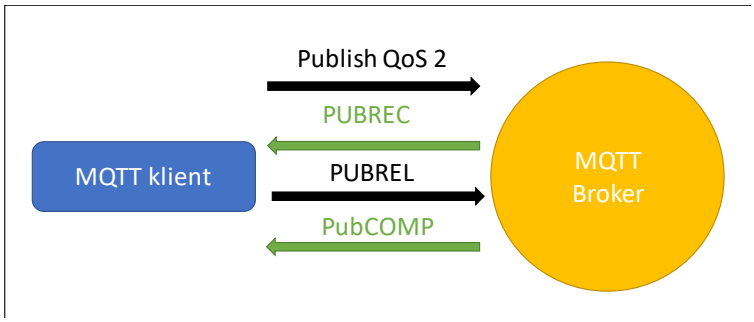
2.6.3 QoS 2 - Přesně jednou

Tato úroveň QoS zabezpečuje, že každá zpráva, která je odeslána, musí být přijata brokerem. Jedná se o nejbezpečnější, ale zároveň také nejpomalejší přenos zpráv. Důvodem je velké zatížení spojení vlivem ověřování. Mezi klientem a brokerem prochází 4 zprávy. V první zprávě jsou všechna data, v dalších třech zprávách si klient



Obr. 2.11: Znázornění komunikace při úrovni QoS 1

s brokerem přeposílají packetId a díky tomu kontrolují, zda byla zpráva doručena (Obr. 2.12). [14]



Obr. 2.12: Znázornění komunikace při úrovni QoS 2

2.7 Brokery pro protokol MQTT

Existují placené, neplacené verze MQTT brokerů a verze neplacené s omezením neplacené verze. Většina placených brokerů nabízí také verzi zdarma s určitými omezeními jako například: maximální počet přispěvatelů, maximální počet zpráv nebo limitovaná podpora oproti placeným licencím.

Název	Placená	Zabezpečení	Limit zpráv
CloudMQTT	Ano	Ano	Ne
IBM Bluemix	Ne	Ano	Ano
Hivemq	Ano	Ano	Ne
Mosquitto	Ne	Ano	Ne
Microsoft Azure IoT	Ne (při účtu Azure)	Ano	Ne

Tab. 2.3: Porovnání MQTT dostupných brokerů

Na základě porovnání byl navržen k použití MQTT broker Mosquitto. Důvody jsou neomezený počet přijímaných zpráv, licence k použití zdarma a možnost zabezpečení komunikace, dále jeho jednoduchá instalace na linuxovou distribuci serverů a použití QoS. Tento broker taktéž není omezen limitem připojených klientů. [15, 16]

2.7.1 Mosquitto broker

Jedná se o broker, který je volný pod licencí EPL/EDL. Tento broker implementuje nejnovější verzi protokolu MQTT 5.0. Mosquitto je součástí nadace Eclipse a je sponzorován společností Cedalo. Instalace je možná na platformy Windows, Linux (Debian/Ubuntu) a Raspberry Pi. Vydavatel poskytuje testovací server, kde je možné jednoduše ověřit funkčnost klienta všemi dostupnými způsoby (MQTT, MQTT s TLS bez klientského certifikátu, MQTT s TLS a klientským certifikátem, MQTT přes WebSocket a MQTT přes WebSocket s TLS). S instalací Mosquitta je pro klienty spojena současně instalace pro publikaci i pro přihlášení k odběru dat. [17]

3 Použité technologie pro aplikaci

Programovací jazyky použité v diplomové práce se dělí na serverové a klientské. K realizaci back-end části byl vybrán jazyk PHP s frameworkem a databázový systém MySql (MariaDB). Pro tento jazyk je vytvořena knihovna pro MQTT, což byl jeden z aspektů pro výběr toho jazyka. Jazyk PHP se řadí mezi nejrozšířenější pro tvorbu webových aplikací. Pro front-endovou část aplikace byla zvolena kombinace HTML a javascriptu. Jedním z důvodů jejich použití byla potřeba využít dostupné webové mapové podklady jako zobrazovací prvek.

3.1 HyperText Preprocessor

HyperText Preprocessor (PHP) se řadí mezi skriptovací interpretované jazyky, pomocí nichž je možné tvořit webové aplikace. Jedná se o multiplatformní jazyk běžící na serverové straně aplikace. Funguje na principu vykonání posloupnosti skriptů, které tvoří aplikaci a odesílání požadovaných dat směrem k uživateli (klientovi).

Jedná se o dynamicky typovací jazyk, což v praxi znamená, že programátor není vázaný na datový typ, ale v běhu aplikace lze podle potřeby upravovat datový typ proměnné. PHP podporuje objektově orientovaný styl programování. Pro tento jazyk existuje podpora v podobě knihovny pro dokumenty PDF, dokumenty XLM a nebo pro tabulkové generátory (excel) a mnoho dalších. PHP je obvykle používáno s databázovým systémem MySQL, případně lze využít PostgreSQL, nebo Firebird. Dají se taktéž využít databáze nad Microsoft servery (MSSql databáze). Jedná se o nejvíce rozšířený skriptovací serverový jazyk a využívá se více než na 80 % webech a webových aplikací. [18, 19, 20]

V rámci jazyka PHP lze využít jeho nadstavby v podobě frameworku. Framework má přesně definovanou strukturu, čímž svazuje programátora k programování dle standardů. Framework obsahuje již některé knihovny nebo standardizované návrhové vzory aplikací. Dále řeší například zabezpečení v rámci aplikace, možnosti aktualizace programovacího jazyka a měl by usnadňovat práci programátorů a architektů systému. Architektura frameworků je tvořena tzv. „frozen spots“ a „hot sposts“. Zatímco frozen spots jsou pevně definované struktury a závislosti mezi komponentami aplikace, hot spots jsou tvořeny vlastními kódy vytvořené pro danou aplikaci. Mezi nejznámější frameworky pro PHP se řadí české Nette, Symfony nebo Laravel. Při porovnání frameworků dle oblíbenosti ve světové komunitě vyhrává framework Laravel. V České republice první místo zaujímá framework Nette, který je tvořený českým vývojovým týmem a zároveň se umístil v roce 2015 i jako 3. nejpoužívanější framework na světě. [21, 22]

Vzhledem k výsledkům porovnání a díky skutečnosti, že pro něj existuje modul pro MQTT protokol, se kterým se v práci pracuje, byl vybrán framework Laravel. [23]

3.1.1 Framework Laravel

Laravel je open source PHP framework pro webové aplikace. Byl vydán v roce 2012 a je vydávaný pod licencí MIT. Tento framework využívá architektury MVC (Model-View-Controler). Laravel patří mezi nejoblíbenější frameworky pro PHP. Vychází z frameworku Symfony, ze kterého přebírá moduly jako je FileSystem, Debug nebo Browserkit. Taktéž úzce spolupracuje s komponentou Composer (PHP dependencies manager), který má na starosti zejména udržování závislostí v rámci projektu a usnadňuje migraci celé aplikace. Jako šablonovací systém se zde využívá komponenta Blade, která spravuje vizuální stránku aplikace. Instalace může probíhat několika způsoby a to pomocí instalátoru Laravel, pomocí Composeru a nebo stáhnutím. Nejčastější použití je pomocí Composeru, který je nutný mít nainstalovaný i pokud použijeme samotný instalátor Laravelu. Laravel využívá databázový ORM systém Eloquent. Eloquent je způsob přístupu activerecord, což je způsob vytvoření objektového modelu databázové tabulky a obslužné metody vytváří sám framework, takzvanými magickými metodami. Oproti přístupu například pomocí systému Doctrine, který naopak má způsob datamapper, který striktně odděluje databázové entity od entit využívaných v systému. Je nutné tedy vytvářet mezivrstvu, která mapuje systémovou entitu na databázovou entitu. Eloquent využívá Doctrine, ovšem výše zmíněnými magickými metodami vytváří sám o sobě datamapper. [24]

3.2 Databázový systém MySQL

Jde o multiplatformní relační databázový systém, ve kterém je dotazovací jazyk odvozený ze standardního SQL (Structure Query Languqe) databázového jazyka. Systém MySQL pracuje na principu dotazu klienta na databázový server. Na databázovém serveru MySQL se řeší zejména uložení dat, jejich organizace, vyhledávání a také správa uživatelů nad databázovým systémem. Na straně uživatele musí být spuštěn databázový klient, jehož hlavní funkcí je komunikace s databázovým serverem. Na základě platné autorizace může tento klient žádat data, popřípadě je upravovat. Databázový systém MySQL využívá klasický model relační databáze. Data jsou uložena v tabulkách, kde každý řádek tabulky představuje jeden záznam, sloupce tabulky reprezentují atributy tohoto záznamu, například unikátní identifikátor. Některý z atributů záznamu by měl být nastaven jako primární klíč, jehož hodnota je v rámci tabulky vždy jedinečná. [25, 26]

3.3 HyperText Markup Language

HyperText Markup Language (HTML) je značkovací typ jazyka. Je určený zejména pro tvorbu webových stránek. Pro svoji činnost využívá protokol HTTP (HyperText Transfer Protokol). Nejnovější verze jazyka je HTML5, která opravuje a rozšiřuje funkcionality předešlé verze. Kód pro webové stránky a aplikace se skládá z klasického textu a značek HTML jazyka, tyto značky se nazývají tagy. Jazyk není „case sensitive“, avšak je doporučeno při zápisu URL cesty k dalším souborům dodržovat konvenci ve velikosti jednotlivých znaků. Značky jazyka mohou být párové i nepárové. Párové značky vždy uzavírají obsah a neměly by se křížit. Nepárové značky neuzavírají obsah, ale vykonávají příkaz dle svých atributů. [27]

3.4 JavaScript

Jedná se o klientský, interpretovaný, multiplatformní, objektově orientovaný programovací jazyk, který běží na klientském zařízení. Celý kód skriptu je odesílán v rámci HTML kódu ke klientovi, kde je vykonán v případě, že nemá klientská stanice zakázaný JavaScript.

Pomocí tohoto jazyka lze jednoduše a dynamicky měnit obsah webových aplikací nebo webových stránek a obsluhovat interakci uživatele v rámci aplikace. Použití je omezeno tím, že se jedná o technologii, která je prováděna výhradně na straně klienta, tudíž existuje možnost, že někteří uživatelé budou mít tuto funkcionalitu (skriptování) zakázanou, respektive jejich webový prohlížeč nemusí mít pro tento jazyk podporu. JavaScriptové aplikace využívají výkon a rychlost uživatelského zařízení, tím pádem se celá aplikace mimo jiné odvíjí od výkonu tohoto zařízení.

V rámci klientského zařízení, ve kterém je skript vykonán, je možné jej načíst, stáhnout a uložit přes nástroje, které jsou integrované ve většině internetových prohlížečů. Skript je závislý na internetovém prohlížeči, a proto v některých verzích prohlížečů nemusí být funkční všechny webové komponenty. Zápis syntaxe technologie JavaScript je „case sensitive“, a proto záleží při vytváření kódu na velikosti písmen. Proměnné se nastavuje typ dynamicky což znamená, že proměnná získá svůj datový typ podle typu zapsané hodnoty. K objektům JavaScript přistupuje jako k asociativním polím. Pro tento programovací jazyk existuje mnoho knihoven, nástavb a frameworků, například React, jQuery, nebo Angular JS. [28]

3.5 Cascading Style Sheets

Jazyk Cascading Style Sheets (CSS) byl vytvořen za účelem oddělení definice vzhledu a obsahu webové aplikace. CSS má za úkol popsat vizuální stránku webové apli-

kace, například jak mají vypadat texty v jednotlivých sekcích, jejich velikost, barvu, písmo, tvar kurzoru v různých situacích atd. Nejnovější verze jazyka je CSS4. Styly lze zapisovat několika způsoby:

- inline – styly jsou vepsány přímo k jednotlivým elementům v HTML kódu pomocí CSS atributu style,
- definice stylu v hlavičce HTML kódu – styl nadefinujeme na začátku souboru v HTML kódu a voláme pomocí identifikátoru stylu,
- externím souborem – vytvořením stylového externího souboru, který je připojen v hlavičce HTML souboru.

První dva způsoby se využívají, ale nejsou příliš doporučeny. Důvodem je, že inline styly jsou definované jen pro jeden prvek a k jejich znovupoužití je nutné kód duplikovat. Zápis v hlavičce HTML souboru sice odstraňuje tuto duplicitu, ale její použití je omezeno pouze na daný soubor, a tudíž se nedá použít v rámci celé aplikace, například na jiných view (zobrazení stránky) v rámci moderního a standardizovaného MVC modelu. Nejčastěji použitým způsobem pro tvorbu a volání CSS stylů je vytvoření a připojením externího souboru, ve kterém jsou všechny styly uloženy na jednom místě a jsou dostupné přes celou aplikaci po připojení CSS souboru v hlavičce stránky. Důvodem pro využití poslední varianty je přehlednost kódů jak v souboru HTML, tak v souboru CSS.

Příklad kódu pro nalinkování stylového souboru:

$$< link\ \ href = \ \ 'style\_files/styls\_default.css`\ \ rel \ \ = \ \ 'stylesheet' \ \ / >$$

K webovým elementům se přistupuje pomocí takzvaných selektorů. Existují čtyři druhy selektorů [29, 30]:

- selektor třídy – styl se aplikuje na všechny elementy, které mají v atributech danou třídu,
- ID selektor – styl se aplikuje na všechny elementy, které mají v atributech daný ID prvku,
- typový selektor – styl se aplikuje na všechny prvky s příznakem typu elementu,
- selektor následníka – kde styl se aplikuje na všechny elementy, které se nachází uvnitř daného elementu.

Kaskádové styly disponují tzv. pseudotřídami, které obohacují pouze určenou část elementu, ve které jsou tyto styly aplikovány. V praxi to znamená, že elementům s definovaným stylem a zároveň elementům dle definicí pseudotřídy, přidáme další stylizaci. [31]

3.6 Mapové podklady

Zobrazovací část aplikace má být zasazení jednotlivých klientů (telemetrických čidel) do mapové kompozice. V této kompozici bude pouze informace o geografické poloze klienta. Po kliknutí na příslušného klienta v mapě je požadavek na zobrazení vizitky (karty telemetrického čidla) se zobrazením aktuálních (posledních přijatých) dat, informacích o čidle a dalších základních údajů.

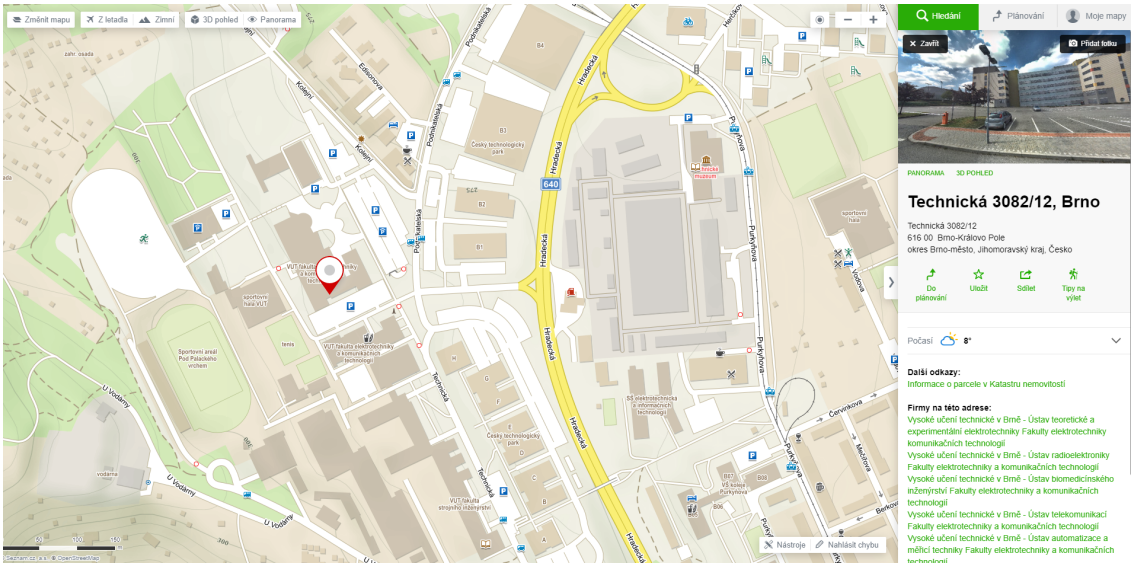
Nejpoužívanějšími mapovými podklady jsou mapy od společnosti Google a Seznam.cz. V rámci porovnání těchto dvou mapových podkladů bylo vybráno aplikační rozhraní(API) od společnosti Seznam.cz z důvodu bezplatné licence a české podpory, která reaguje na požadavky například o doplnění funkcionalit do svého API.

3.6.1 Aplikační rozhraní Mapy.cz

Společnost Seznam.cz, a. s. provozuje aplikační rozhraní (API) s názvem mapy.cz. Umožňuje uživatelům využívat jejich mapové podklady a vytvořené API na vlastních webových stránkách a v rámci aplikací uživatelů. Poskytuje mapovou aplikaci a funkce systému mapy.cz v podobě mapových a obrazových podkladů (letecké mapy).

Jeden z důvodů výběru této služby byl, že nijak neomezuje počet dotazů na webové API. Aktuální verze služby mapy.cz je *4.13 – Neil Armstrong* (k datu 10.11.2019). Použití těchto map i API je zcela zdarma. Je možné ji využít i pro komerční účely, pokud jsou dodrženy licenční podmínky společnosti Seznam.cz, která jsou zejména, že jejich obsah není za placenou bránou.

Funkčnosti API Mapy jsou poskytovány po inicializaci mapového API v hlavičce webové stránky, kde jsou mapy zobrazeny. [32]



Obr. 3.1: Ukázka mapových podkladů od společnosti Seznam.cz

4 Návrh aplikace

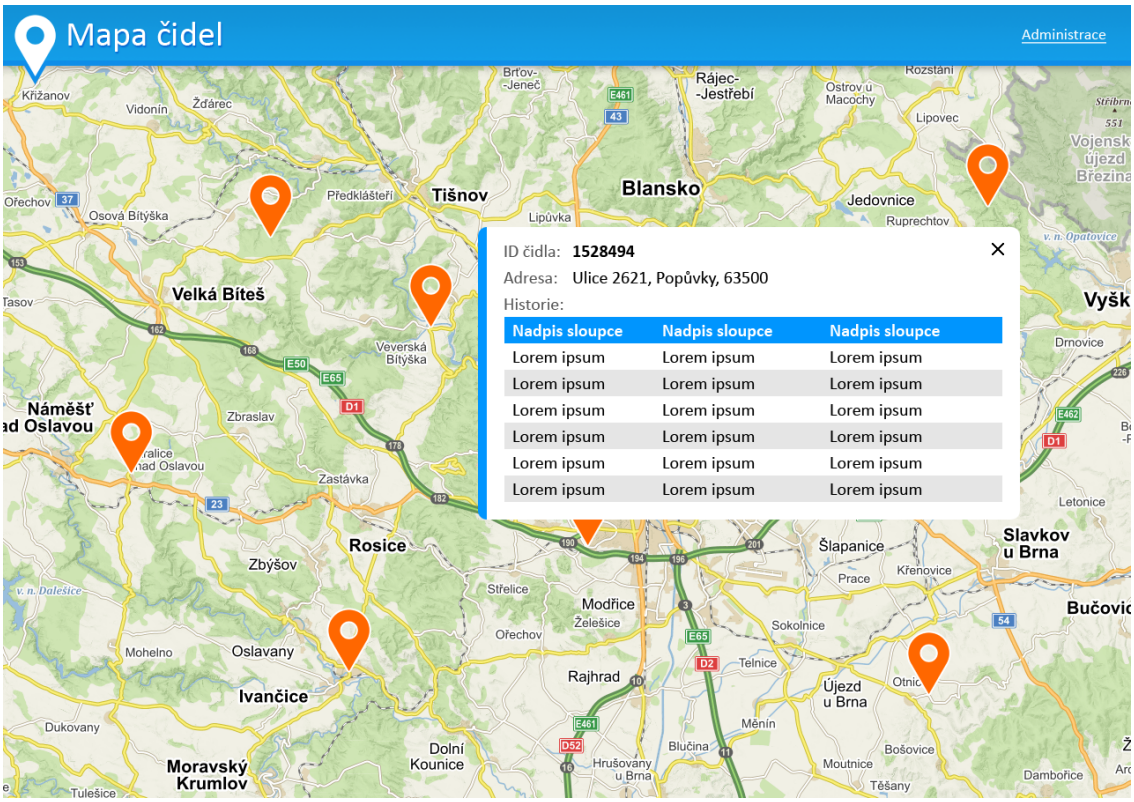
Grafický návrh aplikace je rozdělen na dvě části a to veřejnou a administrační část. Veřejná část bude sloužit pro zobrazení mapy, jednotlivých čidel a po vybrání čidla k zobrazení základních informací. Administrační část aplikace bude sloužit pro správu aplikace, nastavení zobrazení, přidávání uživatelů, schvalování připojených čidel a nastavení exportu dat pro portál Weather Undergroud. Celý návrh je tvořen modulově z důvodu možnosti aplikaci rozšířit a jednoduše upravit případné další prvky. Jako standardní vyplňovací text u grafických návrhů je použit Lorem ipsum. Důvodem je možnost univerzálního využití zobrazovacích modulů, které tak nemusí být vázány na přesnou realizaci.

4.1 Návrh veřejné části

Veřejná část aplikace bude sloužit pro zobrazování provozovaných čidel, informace o tom, kde se nachází a základní informace o hodnotách na nich naměřených. V horní části aplikace je logo s názvem aplikace a možnost přihlášení do administrace. Na zbylé části bude zobrazena mapa, kde budou zobrazena jednotlivé čidla. V rámci kliknutí na čidlo se otevře vizitka čidla. V rámci zobrazení veřejné části bude mapa reflektovat vždy všechna čidla, která budou aktivní a při načtení aplikace se mapa nastaví do stavu zobrazení všech čidel. Mapa bude zobrazovat vždy poslední přijatá data od čidel (Obr. 4.1).

4.2 Návrh administrační části

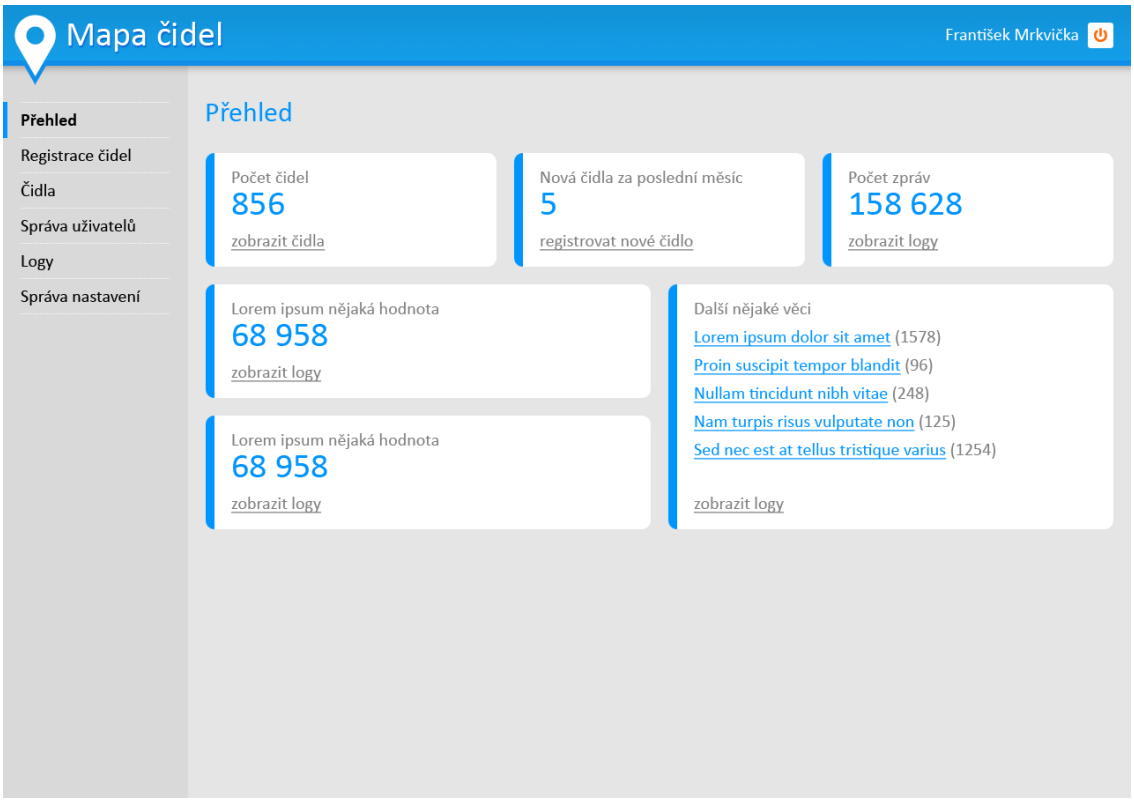
Po přihlášení do administračního rozhraní se zobrazí úvodní stránka administrace. V levé části budou jednotlivé položky administrace s přesnými názvy, jelikož základní funkčnosti známe a mohou se v průběhu vývoje doplňovat a upravovat. Ve střední části bude vždy kontext vázaný k aktuálně vybranému menu.



Obr. 4.1: Grafický návrh veřejné části aplikace

4.2.1 Přehled

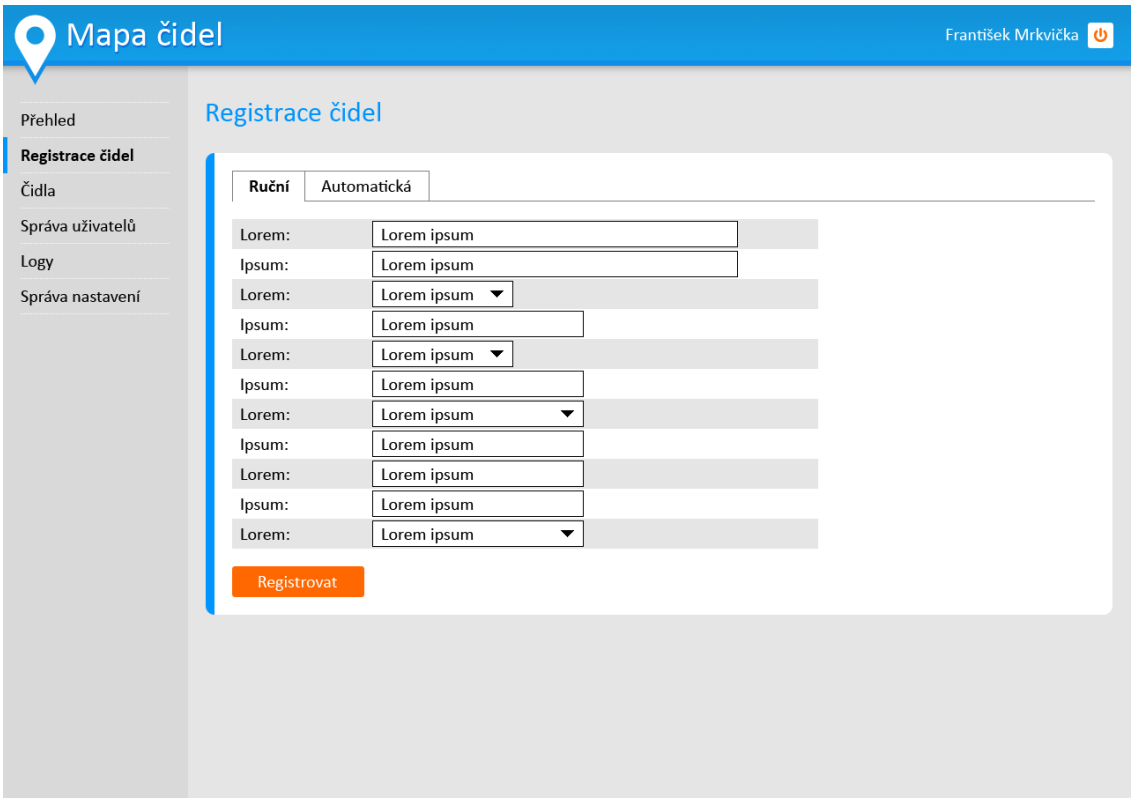
Pro menu přehled, které bude zobrazeno vždy po přihlášení, budou zobrazeny obecné informace o počtu registrovaných čidel, nově přidaných za poslední měsíc, počet přijatých zpráv od jednotlivých čidel (Obr. 4.2).



Obr. 4.2: Administrace - Přehled

4.2.2 Registrace čidel

Pod záložkou Registrace čidel bude možné čidla aktivovat. Registrace čidel bude probíhat automatickou registrací do systému odesláním zprávy. Při registrování čidla bude nutné vyplnit základní informace zejména souřadnice čidla nebo adresu. V automatické registraci nám čidlo předvyplní veškeré údaje, protože je bude poskytovat jako jsou například souřadnice nebo identifikátor. Administrátor pouze potvrdí registraci čidla. Dále v rámci nastavení jednotlivých čidel při registraci bude možné nastavit, které hodnoty chceme ve vizitce v mapě zobrazovat. Je zde počítáno s variantou, že čidla nebudou mít vždy všechny měřící prvky a proto budeme mít možnost nastavovat pro každou vizitku, které údaje budeme zobrazovat (Obr. 4.3).

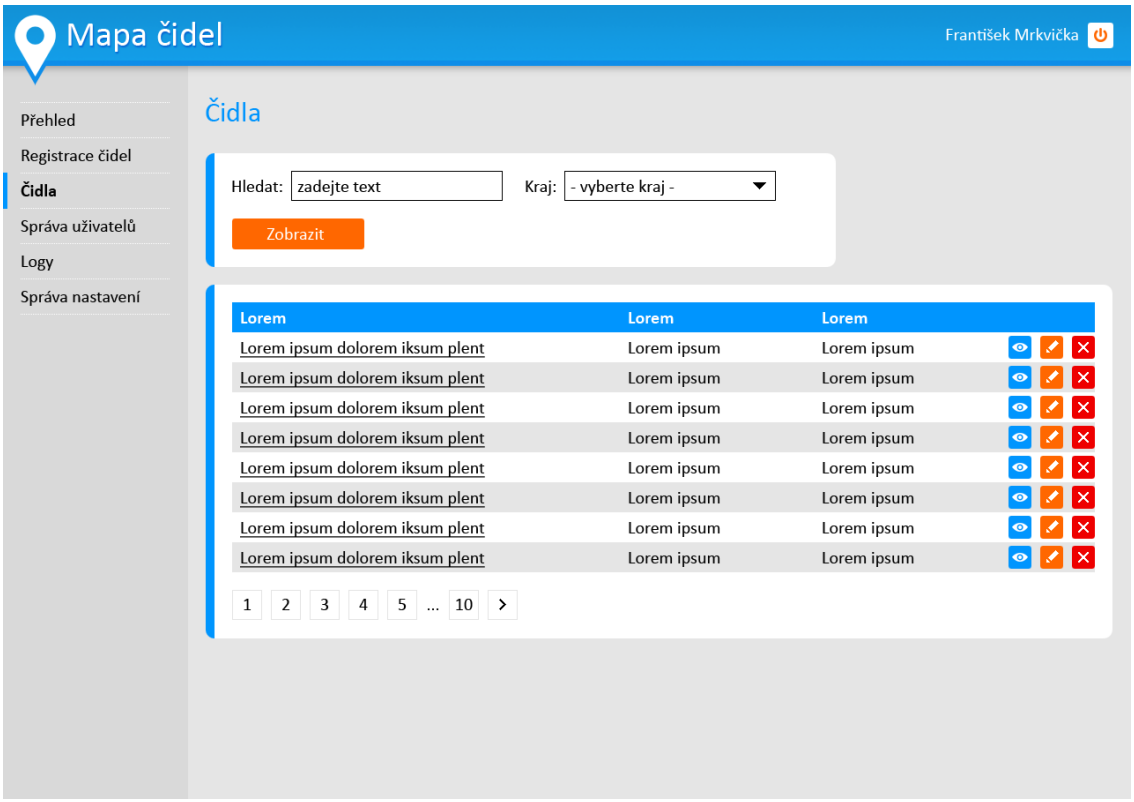


Obr. 4.3: Administrace - Registrace čidel

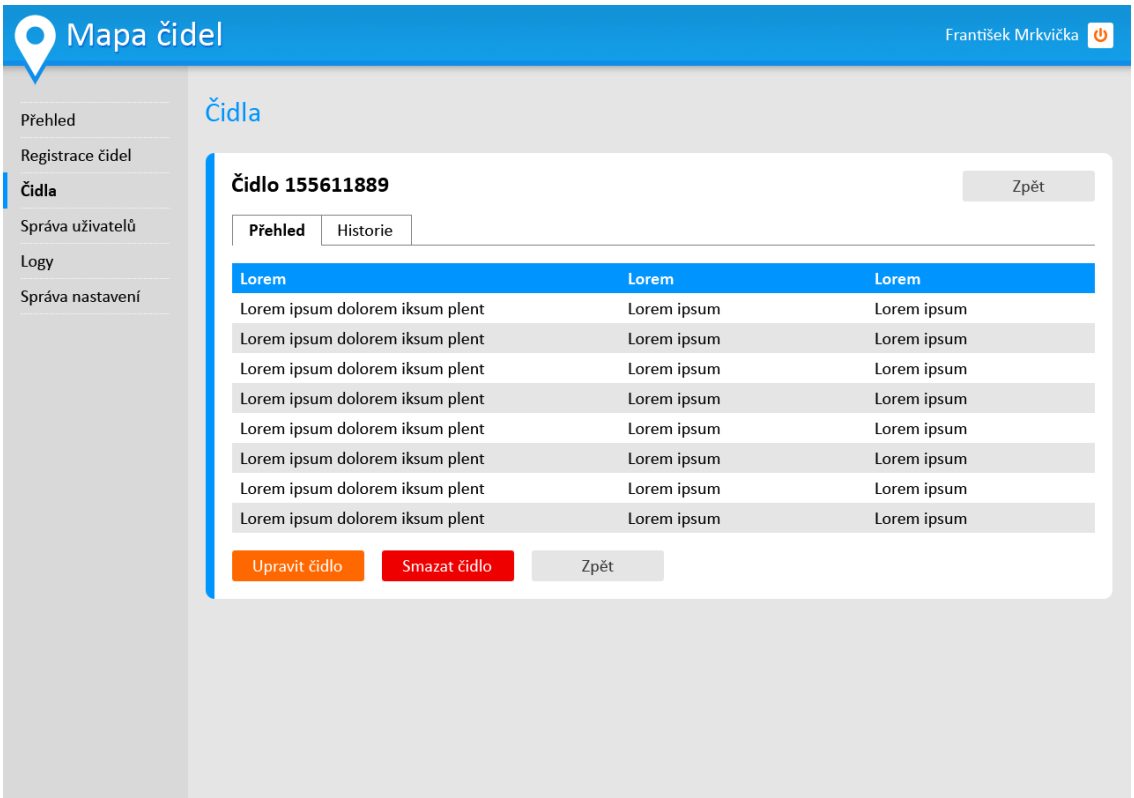
4.2.3 Čidla

V menu čidla bude k dispozici seznam všech aktivních čidel, které lze hledat podle názvu nebo identifikátoru. Bude možné filtrování dle dalšího parametru, v návrhu je použit filtr kraj. Předpoklad je rozšíření filtru na další parametry dle potřeby. V rámci přehledu budou zobrazeny data jako je název, identifikátor, adresa nebo poslední aktualizace dat. U každého čidla budou v pravé části umístěna tlačítka pro detailní náhled, úpravu nastavení nebo skrytí čidla z aplikace (Obr. 4.4).

Po zvolení náhledu čidla se změní kontextové pole na detailní informace o čidle. Zde bude možné zobrazit aktuální hodnoty, aktuální nastavení čidla a po zvolení historie se dívat na historické údaje o datech z čidla. Bude zde možnost i přejít do editace a i čidlo smazat (Obr. 4.5).



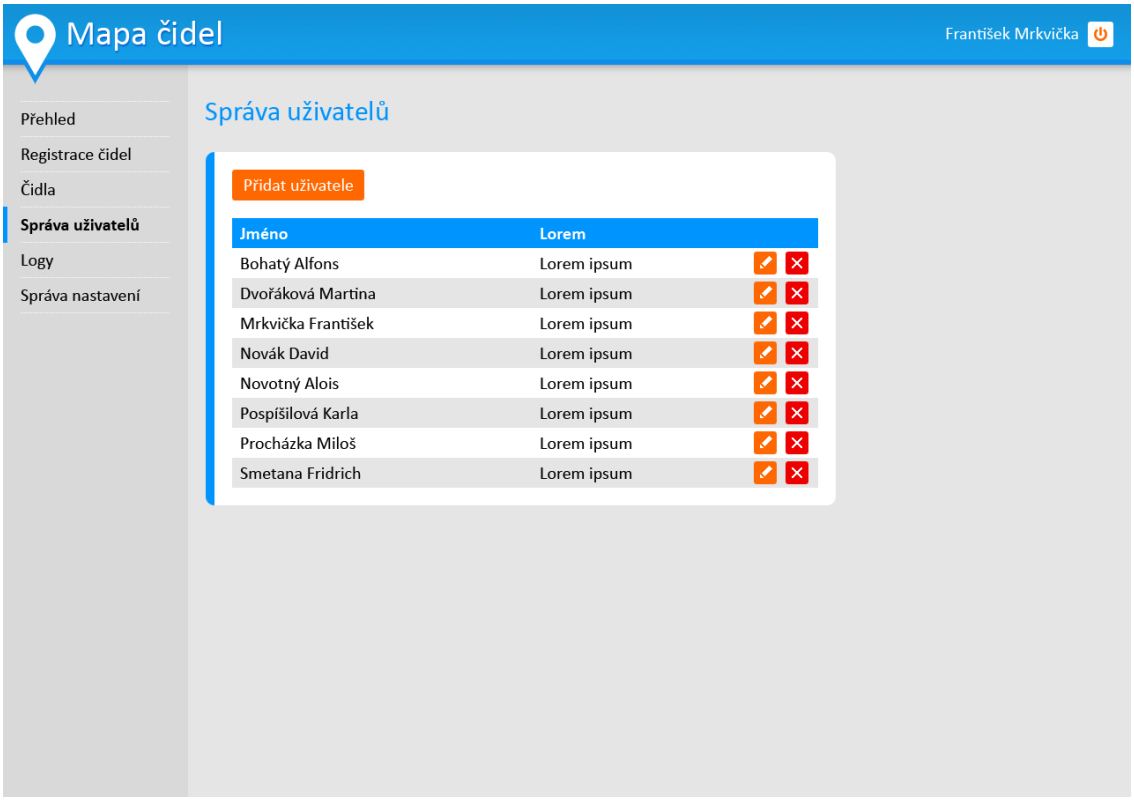
Obr. 4.4: Administrace - Čidla základní zobrazení



Obr. 4.5: Administrace - Čidla detailní zobrazení

4.2.4 Správa uživatelů

V rámci správy uživatelů bude zobrazena tabulka uživatelů, kteří mají přístup do systému a jaké mají oprávnění (Obr. 4.6). Dále bude možné v této části menu vytvářet přístupy pro nové uživatele v rámci systému (Obr. 4.7).



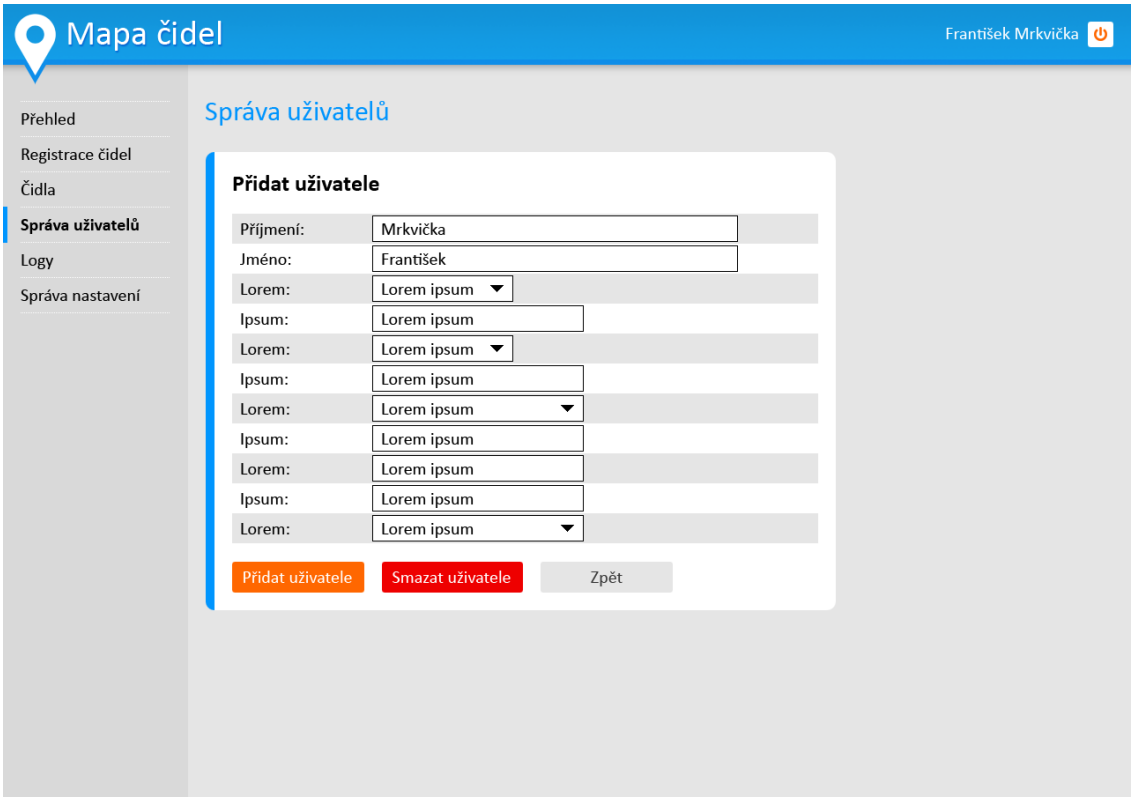
Obr. 4.6: Administrace - Správa uživatelů

4.2.5 Logy

Aplikace bude uchovávat historii změn u důležitých informací. V rámci těchto informací se budou uchovávat data zejména o změnách v administraci, to znamená který uživatel co nastavil na jednotlivých čidlech, komu přidal jaká práva, která čidla byla schválena do systému.

4.2.6 Správa nastavení

Ve správě nastavení bude možné upravovat nastavení uživatelského účtu jako je například změna hesla. Dále zde bude nastavení možnosti upozorňování emailem, pokud se vybrané čidlo například neohlásí po nějakou dobu nebo pokud bude čidlo, které se automaticky přihlašuje čekat na schválení.



Obr. 4.7: Administrace - Přidání nového uživatele

5 Realizace aplikace

Realizace aplikace se do jisté míry liší od jejího návrhu. V průběhu vývoje bylo zjištěno, že některé návrhové funkcionality je vhodnější vytvořit jinak z důvodu uživatelského využití, přehlednosti nebo logické návaznosti. Jedná se zejména o položku Registrace čidla a Čidla, které byly sloučeny a byl vytvořen vhodnější způsob registrace/zobrazení čidel. Dále bylo upraveno názvosloví v jedné části, aby korespondovalo s aplikací třetí strany a budoucí uživatel měl jasné spojení mezi touto apliakcí a aplikací třetí strany. V rámci aplikace má být možnost odeslat data na webový portál Weather Underground, kde se tato meteorologická čidla nazývají stanice.

5.1 Spuštění serveru

Pro realizaci byl zadán a spuštěn Ubunotvý server, jeho verze je Ubuntu 18.04.4 LTS (GNU/Linux 4.15.0-88-generic x86_64. Po instalaci serveru na něm bylo nutné nastavit další prostředí pro spuštění samotného webového serveru. Tudíž byl nainstalovaný Apache2 (Apache 2.4.29 (Ubuntu)), což je HTTP server. Na server bylo nainstalováno PHP ve verzi 7.3 (PHP 7.3.16-1+ubuntu18.04.1+deb.sury.org+1).

Dále byl nainstalovaný databázový systém MySQL. Na začátku vývoje byl na server realizovaný útok, dle dostupných informací se zřejmě jednalo o způsob zkoušení otevřených portů na serveru. Všechna data byla útočníkem smazána a v databázi zůstala vytvořená jediná tabulka, kde byla informace o tom, že pokud si přejeme data vrátit, je možné je odkoupit. Byla zde přiložena cena v Bitcoinech a bitcoinová adresa peněženky. Vzhledem k tomu, že útok byl realizovaný v začátku vývoje aplikace a v databázi byly pouze tabulky, které byly jednoduše obnoveny, a jediná tabulka, kde byla data, byla tabulka s přijatými zprávami z MQTT serveru, tak naštěstí nevznikla velká škoda. Databázový systém byl aktualizován na novější verzi MySQL (mysql Ver 15.1 Distrib 10.4.12-MariaDB, for debian-linux-gnu (x86_64) using readline 5.2) a server byl dále zabezpečen firewallem s možným přístupem jen z určitého adresního rozsahu.

Další instalovaná aplikace je Composer (version 1.9.0 2019-08-02 20:55:32), díky níž je možné instalovat jednotlivé balíčky v rámci frameworku a stará se o závislosti mezi knihovnami a frameworkem. Následně byl naisntalovaný framework Larvel (tehdy verze 6.18). V průběhu vývoje aplikace byla celá aplikace zmigrována na novou verzi Laravelu 7.X, kdy nynější verze frameworku (k 18.4.2020) je 7.6.2. Pro využití grafických balíčků Bootstrap a Node.js bylo nutné nainstalovat Node.js package manager(NPM) ve verzi 6.13.4.

5.1.1 Aplikace Adminer

Aplikace Adminer byla doinstalována později jako serverová aplikace a není nutná ke spuštění serveru. Adminer je aplikace na správu databází. Při vývoji aplikace nebyly využity migrace databází, které umožňuje framework Laravel. Ovšem bylo nutné vytvořit pro instalační balíček tvorbu databázových tabulek a definice některých defaultních hodnot v nich. Proto byla použita aplikace Adminer, která umí vytvořit SQL příkazy z aktuální databáze a uložit je do jednoho souboru, který je poté při instalaci vyvolán a vytvoří se celá databázová struktura. Tato záloha databáze je možná i z vývojového prostředí PHPStorm. Pro jednoduchost zálohy databáze a následné úpravy dat, byla vybrána tedy tato aplikace.

5.2 Spuštění frameworku Laravel

Spuštění Laravelu proběhlo pomocí composeru příkazem `composer global require laravel/installer` , který stáhne a nainstaluje Laravel. Poté již pracujeme s příkazy Laravelu pro vytvoření projektu nebo je možné využít i přkazy composeru. Po založení projektu je nutné vygenerovat klíče pro aplikaci. Pokud je potřeba, je nutné vytvořit soubor `.htaccess`, což je soubor pro směrování v rámci serveru (pokud máme jinou adresářovou strukturu než je standardní, popřípadě routing serveru je do jiného adresáře), typicky *var/www/html/public* [33]. Což v rámci aplikace bylo nutné vytvořit, jelikož adresářová struktura je *var/www/html/DP_STD/public*.

Dále byl nastaven přístup na databázi v souboru .env, který je konfiguračním souborem aplikace. Po správném nastavení výše uvedených specifikací se zobrazí úvodní stránka Frameworku. Pro vývoj aplikace dále byl dále zvolen přístup s automatickým uploadem změn na server. Tudíž lokální prostředí vůbec není nutné spouštět (ale je možné). Všechny změny jsou okamžitě odeslány na server, kde se projeví. Úskalí tohoto typu vývoje je, že je nutné spouštět příkazy pro NPM a Composer jak lokálně (kdyby bylo potřeba spustit lokálně aplikaci), tak separé na serveru. Jelikož složka vendor (závislosti a knihovny) se může lišit v rámci konfiguračních souborů a cest v nich nadefinovaných v závislosti na prostředí.

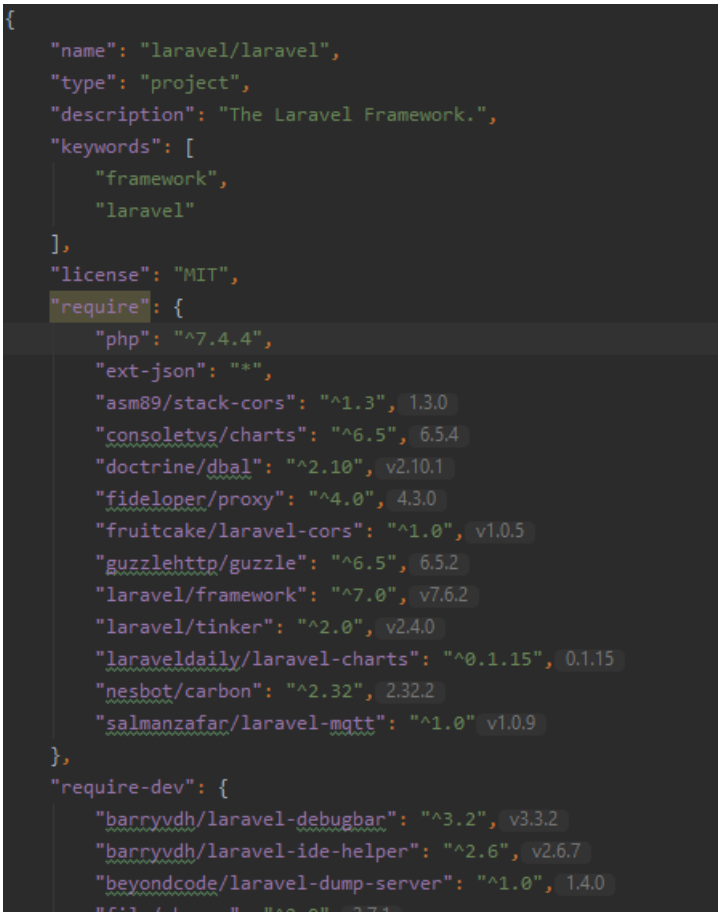
Jedním z důvodů tohoto typu vývoje byl okamžitý náhled pro vedoucího práce, jednoduché řešení připojení databáze za firewallem a to, že není potřeba bez nutnosti řešit tunelování přes SSH přístup nebo možnost souběhu služeb pro příjem dat z MQTT služby a odesílání dat na portál Weather Underground.

5.3 Knihovny pro aplikaci

Knihovny aplikace můžeme rozdělit na knihovny nutné pro framework a volitelné knihovny instalované pomocí aplikace Composer nebo NPM a knihovny pouze importované například pomocí inline zápisu. Některé knihovny jsou navzájem provázané například knihovna `laravel/ui`, která vkládá závislosti pro grafickou stylizaci aplikace. Je nutné ji nainstalovat pomocí Composeru a je přímo využívána NPM.

5.3.1 Knihovny spravované pomocí Composeru

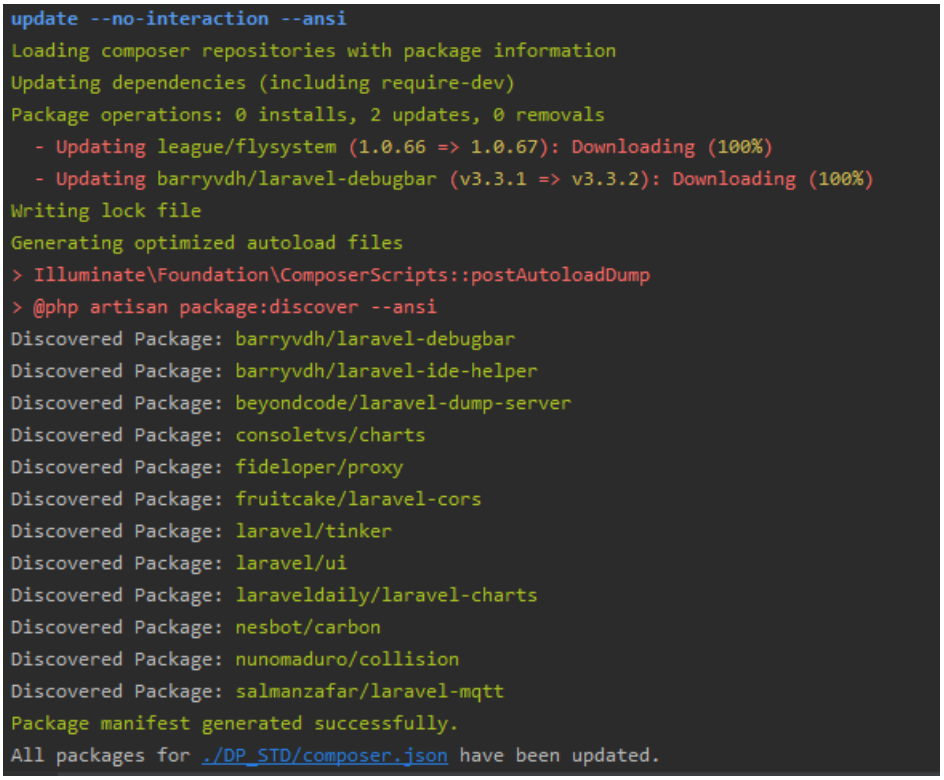
Knihovny, které jsou součástí aplikace jsou definované v souboru composer.json. Tento soubor obsahuje strukturu pro Composer (Obr. 5.1). Jak je vidět na obrázku (Obr. 5.1), tak prostředí PHPStorm samo vypisuje aktuální verze instalovaných knihoven a zároveň upozorňuje na to, že v některé knihovně byla vydaná aktualizace a nabídne možnost, jak projekt udržovat aktualizovaný ihned po vydání aktualizace. Po zadání příkazů `composer install` a `composer update` se knihovny definované



Obr. 5.1: Ukázka části souboru composer.json

v souboru nainstalují, zaktualizují se ostatní již nainstalované knihovny a vytvoří se

v rámci projektu správné závislosti (Obr. 5.2). V souboru je taktéž možné definovat



Obr. 5.2: Ukázka proběhlé aktualizace knihoven pomocí Composeru

verze knihovny, zda se mají aktualizovat, případně nastavit přesnou verzi knihovny, kterou chceme využívat. Pro aplikaci sběru telemetrických dat a jejich zpracování bylo potřebné zejména využít knihovnu pro MQTT komunikaci, vykreslování grafů nebo práci s časem.

Výpis některých klíčových knihoven pro vývoj:

- salmanzafar/laravel-mqtt – knihovna pro komunikaci s MQTT serverem,
- nesbot/carbon – knihovna pro práci s daty a časem,
- barryvdh/laravel-debugbar – knihovna pro zobrazení debugování aplikace,
- barryvdh/laravel-ide-helper – knihovna pro vypsání properties modelu a magických metod nad modelem v rámci Eloquentu.

5.3.2 Knihovny spravované pomocí NPM

Node.js package manager má na starosti grafické knihovny. Zde je nutné vytvořit závislosti v rámci Laravel projektu a to příkazy:

- `php artisan ui bootstrap -auth`
- `php artisan ui vue -auth`
- `php artisan ui react -auth`

Po nastavení závislostí je nutné zkompilovat knihovnu a to příkazem: `npm run dev`. Díky tomuto můžeme využívat bootstrap, vue a react. [34]

5.3.3 Knihovna pro zpracování grafů

Pro zpracování grafů byly vyzkoušeny 3 knihovny, z toho dvě, které jsou přímo pro PHP a jedna pro JavaScript. PHP knihovny byly `consoletvs/charts` [35] a `LaravelDaily/laravel-charts` [36] a JavaScriptová knihovna `Highcharts` [37].

Knihovna LaravelDaily/laravel-charts byla zkoušena jako první, po nastavení bylo zjištěno, že umožňuje zobrazení dat pouze z databázového modelu, což je pro účely aplikace nevhodné. Byla tedy nalezena knihovna consoletvs/charts, která umí definovat pole dat i jinak než pouze z modelu. Zde se ovšem nepodařilo nalézt řešení zobrazení grafů dle architektury aplikace. Musel se využít vložený iFrame a tím nastal problém v ovládání grafu. Proto byla zvolena varianta JavaScriptové knihovny Highcharts.

Výhodou je, že graf funguje pomocí zavolání knihovny až na fronendové straně aplikace. V rámci aplikace je pomocí technologie AJAX volána třída ModalController, která sestaví pouze JSON pole se všemi daty a ty jsou odeslané na frontend aplikace, kde se volá příslušná knihovna s těmito parametry a graf se vykresluje. Další zásadní věc je ta,že ke knihovně je výborná technická dokumentace. Následující zásadní výhoda této knihovny je, že není potřeba v rámci aplikace vytvářet závislosti. Tato knihovna je pod licencí volného užití za splnění některé z následujících podmínek [38]:

- vládou financované školy
- univerzity,
- neziskové organizace,
- student.

5.3.4 Knihovna pro komunikaci s MQTT serverem

Komunikace s MQTT serverem má na starosti knihovna MQTT-Laravel [39]. Knihovna se instaluje pomocí composeru `composer require salmanzafar/laravel-mqtt` , dále je nutné ji inicializovat v rámci projektu v souboru app.php a to v sekci providers a aliases.

Výpis 5.1: Přidání knihovny

```
'providers' => [Salman\Mqtt\MqttServiceProvider::class,];

'aliases' => ['Mqtt' => \Salman\Mqtt\Facades\Mqtt::class,];
```

po nastavení je nutné knihovnu tzv. vypublikovat v aplikaci pomocí příkazu: `php artisan vendor:publish -provider='NAZEV PROVIDERU'`.O chod MQTT subscriberu se

stará kontroler `MQTTSubscribeController`, který extenduje třídu knihovny Mqtt. V rámci kontroleru `MQTTSubscribeController` je volána metoda `subscribeToTopic` , která má za úkol vytvořit připojení k mqtt serveru a po přijmutí zprávy s daným tématem tato data dále zpracovávat.

5.4 Spouštění webové a MQTT aplikace

Architektura aplikace je vázaná na standardy MVC modelu a Laravelu. Tudíž v rámci aplikace je rozdělen Model, View a Controller. Model představuje obraz jednotlivých databázových tabulek a příslušných operací nad zpracovávanými daty. Laravel přistupuje k databázovému modelu pomocí metody Active Records, což znamená, že z databázové tabulky vytváří objektový model a operace nad databázemi řeší pomocí definovaných metod (magických), které není nutné definovat, jelikož jsou součásti frameworku. To je největší rozdíl od druhého přístupu typu datamapper. View je soubor s kombinací HTML, javascriptu a css pro vykreslování uživatelského rozhraní, přičemž byl využit šablonovací systém blade. Controller je třída pro zpracování dat a jejich úprava. Jedná se o vrstvu, která spojuje model a view.

5.4.1 Routování v rámci spuštění aplikace

Bylo nutné najít způsob, jak zabezpečit, aby jel stále skript, který má na starosti přijímání dat z MQTT serveru. Tady nastal problém s tím, že bylo nutné nastartovat celý web a veškeré obslužné metody, aby bylo možné využít PHP pro komunikaci do databáze. Problém byl vyřešen použitím linuxového serveru, kde je vytvořena služba, která spouští skript, popřípadě pokud se vyskytne chyba tato služba jej restartuje. Tato služba spouští `indexSubscribe.php`, která následně spouští celou aplikaci. V routování webu je nastaveno spouštění přes základní kontroler s názvem `SwitcherIndexController`. Tento kontroler má na starosti zjištění, jestli se jedná o dotaz ze služby nebo o přístup na webovou aplikaci. Pokud se jedná o požadavek ze služby, tak se vytváří spojení, které se připojí na definovaný MQTT server a začne přijímat data, zpracovávat je a ukládat do databázové struktury. Pokud se jedná o požadavek jiný, spouští se aplikace a zobrazuje se webové rozhraní.

5.4.2 Konvence pro příjem dat z MQTT

V rámci návrhu příjmu dat bylo s kolegou Bc. Alešem Musilem, který má jako diplomovou práci tvorbu čidel/stanic, navržen komunikační standard. Tento standard pro hodnoty je nutné dodržet, jinak data nebudou zpracována aplikací. Podoba vyplněných dat je následující:

- v tématu zprávy (topic) je možné použít jakékoliv téma, ovšem poslední položka musí být ID čidla, nyní vutbr/xmusil/223202,
- hodnota tohoto tématu se skládá z JSON pole ve kterém každá hodnota má své vlastní JSON pole.

Výpis 5.2: Model dat pro zpracování aplikací

```
[
  {
    "name":"Baterie",
    "value":"3.069322344322344"
  },
  {
    "name":"Teplota",
    "value":"21.85"
  },
  {
    "name":"Tlak",
    "value":"952.61"
  },
  {
    "name":"Vlhkost",
    "value":"36"
  }
]
```

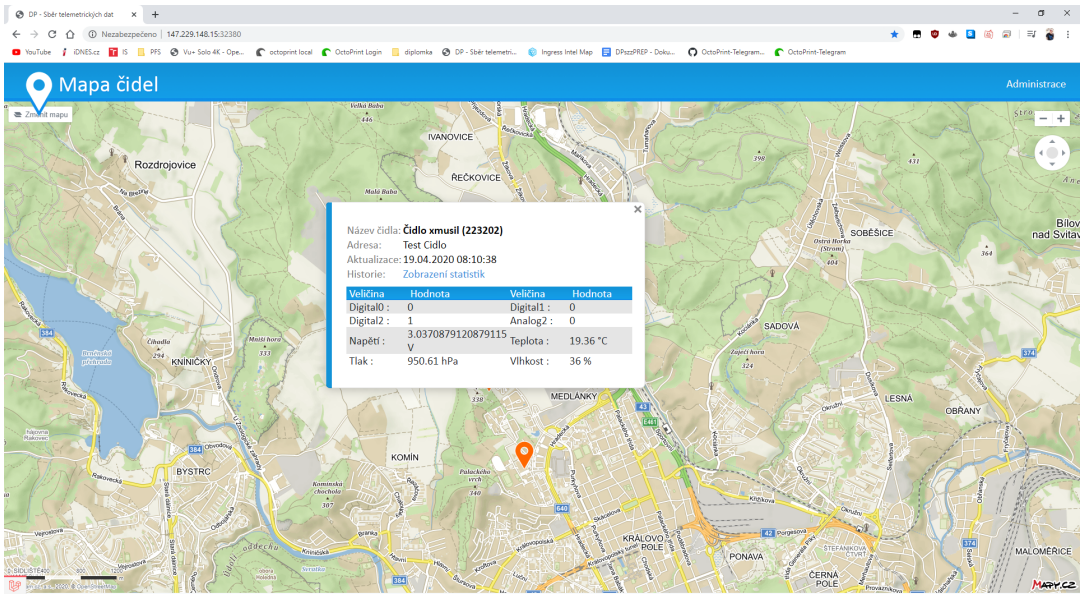
Aplikace dále vždy zobrazuje poslední přijatou zprávu jako nejaktuálnější. V rámci vývoje než byl domluven komunikační standard aplikace. Ovšem z počátku bylo nutné zahájit vývoj na aplikaci a proto byl připojený MQTT server, na který jsou odesílány informace z iZastávek. Tyto zastávky využívají model zasílání zpráv, kdy ke každému tématu byla přijata jen jedna hodnota v strinogvé podobě. Navíc téma mělo strukturu iZastavka/VELICINA. Aplikace v první části vývoje tedy pracovala s tímto způsobem příjmu a uložení dat, který po definování standardu s kolegou Bc. Alešem Musilem, který v rácmi své diplomové práce vytvářel tyto čidla, byl definován.

6 Architektura webové aplikace

Architektura webové aplikace lze rozdělit do dvou částí a to do veřejné a do administrační sekce. Každá sekce má svou vlastní funkci, která je popsána níže.

6.1 Veřejná část aplikace

Po přístupu na webové rozhraní aplikace se zobrazí mapa s aktuálně zobrazenými stanicemi (čidly) (Obr. 6.1).



Obr. 6.1: Výsledná realizace veřejné části aplikace na serveru

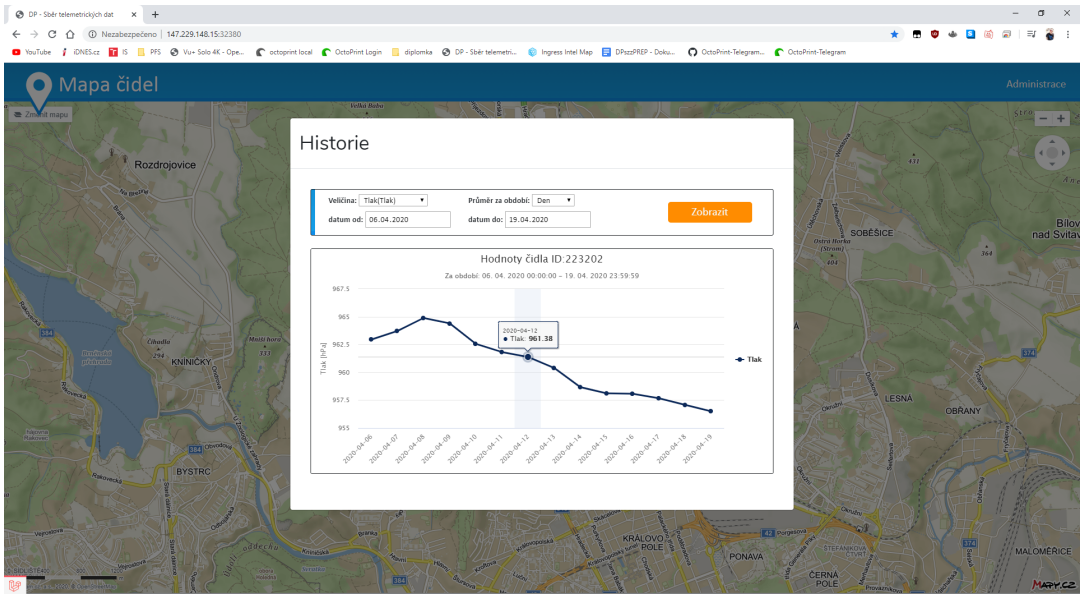
Vykreslení obsluhuje `StaniceController`, kde je volána metoda `getMapaAStanice()`, tato metoda sestavuje kompletní javascript pro zobrazení a ovládání mapy, markerů, které představují jednotlivé stanice a vizitky stanic (Obr. 6.2). Jednotlivé segmenty aplikace jsou členěny do příslušných tříd kontrolerů a mají své metody pro získání, zpracování a vrácení dat. V první části se vytváří základní mapa, zde se volá `MapyController` a je nutné předat parametr, ve které sekci (divu) v uživatelském rozhraní se bude mapa zobrazovat. Pro aplikaci byl zvolen parametr K. Dále nastavujeme, jak se má mapa chovat, pokud v ní budou nějaké markery. Pokud v mapě budou markery, bude se centrovat na ně, pokud markery nebudou, zobrazí se mapa celé republiky. Pro nastavení základní mapy je nutné přidat mapovou vrstvu, kam se budou jednotlivé markery vkládat. Na toto slouží `VrstvaController`. Zásadní je nastavení, zda se jedná o vrstvu pro markery anebo pro zobrazení grafických prvků, jako jsou linie a polygony. Jako poslední vkládáme markery. Jednotlivé markery potřebují znát informace o mapě a o vrstvě. Jelikož

v konstruktorech jednotlivých tříd výše předáváme vždy objekt prvku, který je potřeba, tak i tady při vytváření instance třídy markeru `MarkerController` předáváme objekt vrstvy, který zároveň nese objekt mapy. Nad objektem `MarkerController` voláme metodu pro získání dat o stanicích, jako nepovinný parametr zde vstupuje `idStanice`, který pokud je vyplněný, tak vrací pouze data pro jeden konkrétní marker. Data, která jsou získána z `MarkerController`, jsou vložena do vizitkové karty jednotlivých stanic pro zobrazení v mapě. Po nastavení všech objektů a vytvoření JavaScriptů se celý tento balík vrací do mapy, kde se zpracuje a vykresluje se mapa s markery (Obr. 6.2).

```php
public function getMapaAStanice($idStanice = null): string
{
    /** @var  $mapa */
    $mapa = new MapyController(env('parametr_mapy'));
    $mapa->setCentrovani( parametr: true);

    /** @var  $vrstva */
    $vrstva = new VrstvaController($mapa);
    $vrstva->nastavTypVrstvy(VrstvaController::$VRSTVA_MARKERU);
    $vrstva->setIdVrstva( parametr: VrstvaController::$VRSTVA_MARKERU. 's_layer');
    $vrstva->vytvorJavaScriptProVrstvu();

    /** @var  $markery */
    $markery = new MarkerController($vrstva);

    $stanice = new MarkerDataController();
    $stanice->getDataFromDB($idStanice);
    $JSpush = '';
    foreach ($stanice->allData as $id => $data)
    {
        $markery->setMarkerID( markerID: 'Stanice_' .$id);
        $markery->setUrlMarker(URL::asset( path: '/resources/images/ico-spot_mensi.png'));
        $markery->setTitle( title: 'Stanice ' .$id);
        $markery->vytvorMarkerJS($data);
        if($mapa->isCentrovani())
        {
            $JSpush.= $mapa->VytvorJavaScriptProPridaniSouradniceDoAutoZoomu
            ($data['info']->souradnice_x,$data['info']->souradnice_y);
        }
    }

    /** @var  $JS */
    $JS =  $mapa->vytvorMapu();
    if($mapa->isCentrovani())
    {
        $JS .= $JSpush;
        $JS .= $mapa->centrovaniMapyVypocet();
    }
    $JS .= $vrstva->VrstvaJS;
    $JS .= $markery->MarkerJS;

    return $JS;
}
```

Obr. 6.2: Metoda getMapaAStanice()

Po zobrazení historie ve vizitce se vyobrazuje grafické zobrazení hodnot v historii. (Obr. 6.3).



Obr. 6.3: Výsledná realizace zobrazení statistik v grafu

Zobrazení historie hodnot je realizováno pomocí modálního okna. V rámci otevření modálního okna je volána metoda pomocí technologie AJAX k vyplnění příslušnými daty. Zde jsou vytvořené dvě volání metodou GET a metodou POST na `ModalController`. Metoda GET volá metodu `getDataOvladani`, která má na starosti nastavení ovládání grafu. V nastavovacím rozhraní grafu lze vybrat veličinu, jakou četností ji chceme zobrazit a datum od a do. Po vytvoření těchto dat se volá na stejnou routu ovšem metodou POST metoda `getRefreshGraf`, který obstarává výčet dat a vrací výsledné pole pro knihovnu Highcharts. Metoda `getRefreshGraf` má nastarosti zpracování request z modálního okna a volá metodu `vytvorDataProGraf`, která je zodpovědná za správné vyplnění JSON pole pro vykreslení grafu. V této metodě se tedy z databáze získávají data pro vytvoření statistik a jsou zde volány metody pro průměrování hodnot, tak jak jsou vybrány (hodinové, denní, týdenní, měsíční) nebo metody pro výběr všech dostupných hodnot.

6.2 Administrační část aplikace

Po přihlášení do administrační sekce pomocí emailu a hesla je možné definovat a upravovat definice aplikace. V rámci administrační části je například přehled, správa čidel, nastavení odesílání upozornění a jiné.

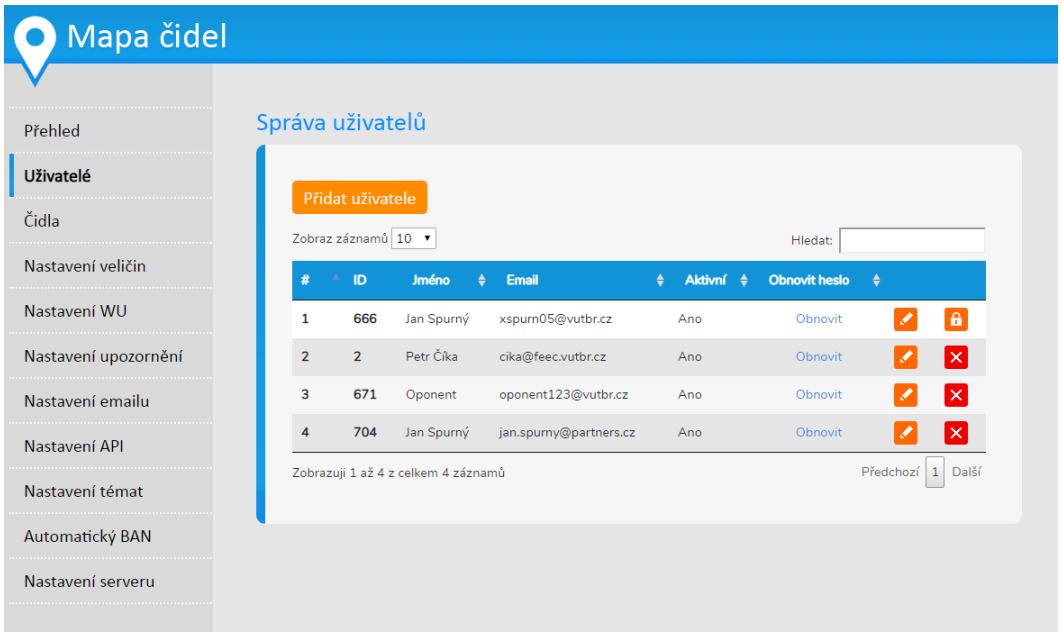
6.2.1 Přehled

Po přihlášení do administrace se zobrazí první část menu se stručným přehledem a základními informacemi o celé aplikaci. Jsou zde informace o počtu administrátorů, počtu registrovaných stanic, počtu aktivních a neaktivních (zobrazených v mapě) stanic, celkový počet přijatých zpráv ze všech vstupů a počet nastavení exportních přístupů pro portál WeatherUnderground 6.4.

Obr. 6.4: Administrační část - Přehled

6.2.2 Registrace a přihlášení uživatele

Registrace uživatelů byla zvolena pouze z administrace. Nového administrátora smí přidat pouze registrovaný uživatel. Registrace probíhá v záložce Uživatelé - Přidat uživatele. V tomto formuláři je nutné vyplnit Jméno a Přihlašovací email, na který je odeslán email pro vytvoření hesla. Heslo si může každý uživatel po přihlášení změnit. V záložce uživatelé je taktéž přehled všech uživatelů. Aktuálně přihlášený uživatel se vždy zobrazuje jako první. V rámci menu jsou zde prvky pro smazání a editaci uživatele. Pro registrace uživatelů byla využita základní komponenta frameworku Laravel a byla upravena k potřebám aplikace. Byly odstraněny některé prvky nutné pro registraci a byla upravena validace (Obr. 6.5). Validuje se vyplnění jména, vyplnění a unikátnost přihlašovacího emailu a délka hesla (Obr. 6.6). Pro zabezpečení hesla byla použita metoda `Hash::make()`. Bylo zde možné vytvořit vlastní



Obr. 6.5: Administrační část - Uživatelé

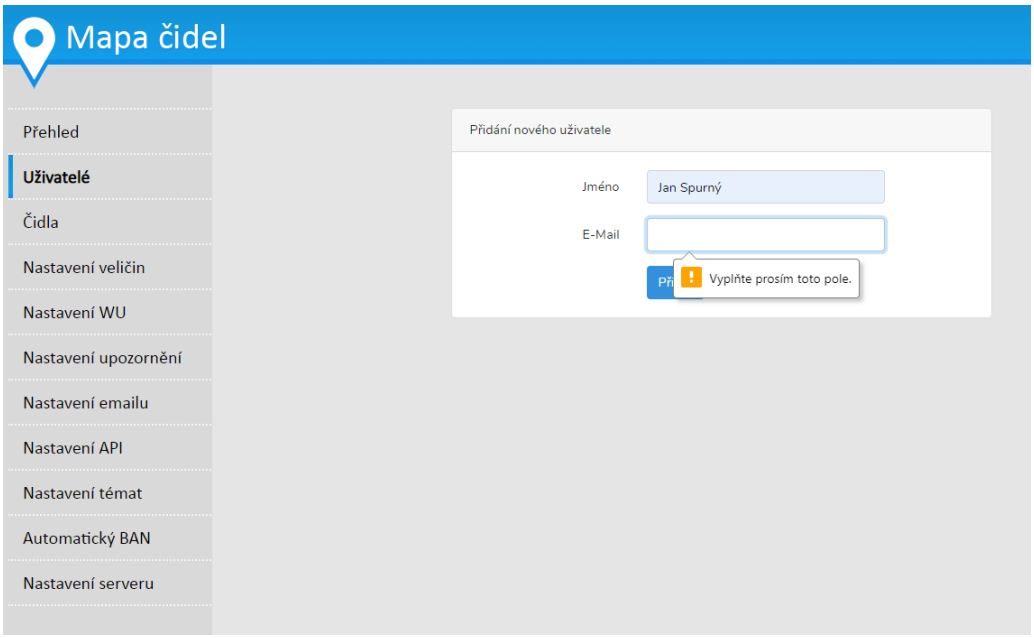
hašovací algoritmus, s využitím papperu (statická sada znaků například ze souboru .env) a saltu (dynamické ukládání sady znaků unikátních pro každého uživatele), vícenásobné hašhování, přeskládávání jednotlivých iterací hašovaných znaků popřípadě dalších kryptografických metod. Ovšem aplikace nepracuje s tak citlivými daty, že tyto metody, využívané zejména v bankovnictví, nejsou nutné k implementaci.

6.2.3 Správa čidel

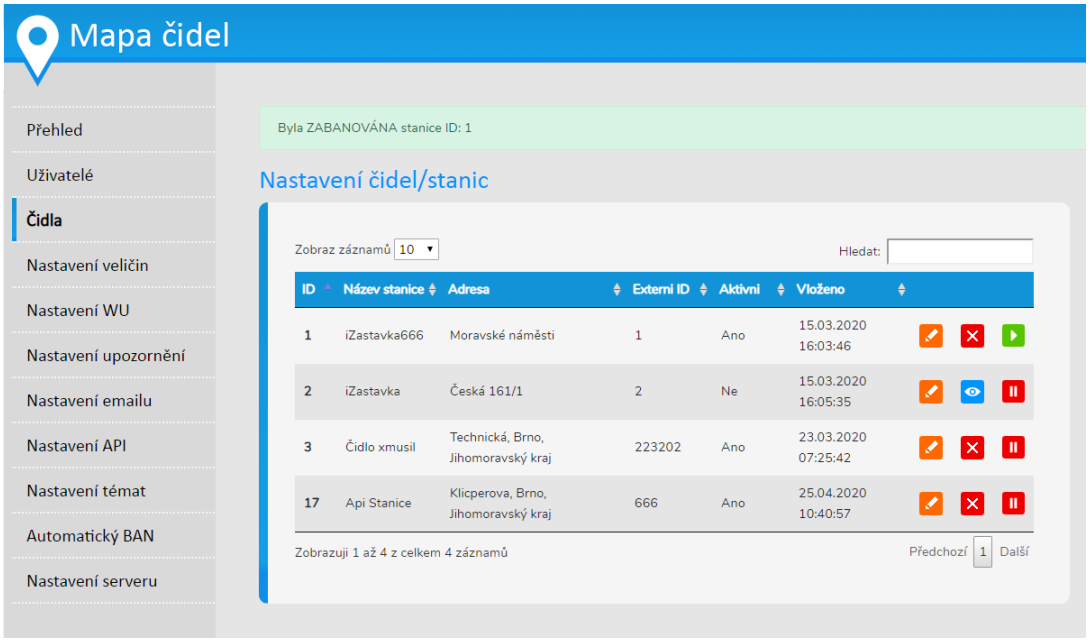
V menu Čidla je výpis všech čidel/stanic, je zde vidět jejich název, adresa, externí ID a jestli je čidlo aktivní (zobrazené v mapě). Dále je zde možné přejít do nabídky editace pomocí tlačítka tužky. Zneaktivnit čidlo lze pomocí tlačítka křížku, pokud je stanice neaktivní, zobrazuje se ikona oka pro aktivaci (zobrazení) nebo lze nastavit zabanování příjmu zpráv od čidla (Obr. 6.7).

V editaci jednotlivých stanic můžeme vidět základní nastavení stanice, jako je název, adresa, souřadnice ve formátu JTSK (Systém jednotné trigonometrické sítě katastrální), zda má být čidlo zobrazeno v mapě (aktivní), přiřazování veličin a náhled vizitky, která se zobrazuje v mapě (Obr.6.8). V rámci editace umí aplikace na základě adresy i odpočítat souřadnice, které lze ručním zásahem jednoduše doupravit. Jelikož jsou souřadnice ve formátu JTSK, kde jednotlivá čísla určují počet metrů od počátku souřadného systému.

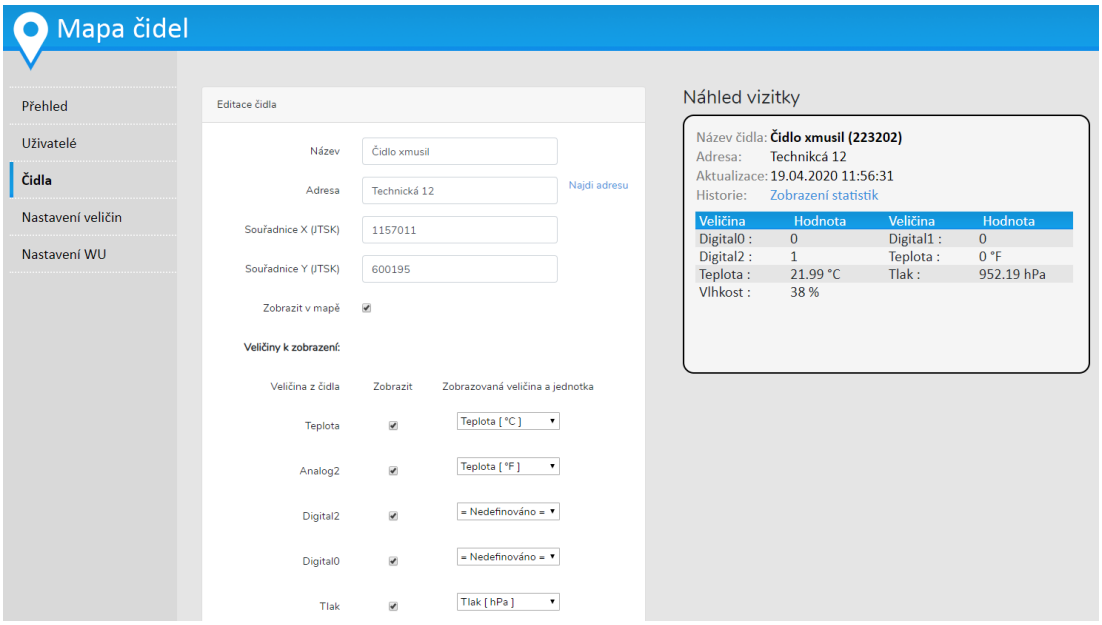
Přiřazování veličin je důležité z důvodu toho, že čidla/stanice mohou posílat údaje mimo konvenci. Tudíž bylo potřeba vytvořit překladovou tabulku. V editaci můžeme přiřadit k výstupu z čidla (Digital0, Digital1,...) jednotlivé veličiny včetně



Obr. 6.6: Administrační část - Uživatelé validace



Obr. 6.7: Administrační část - Čidla



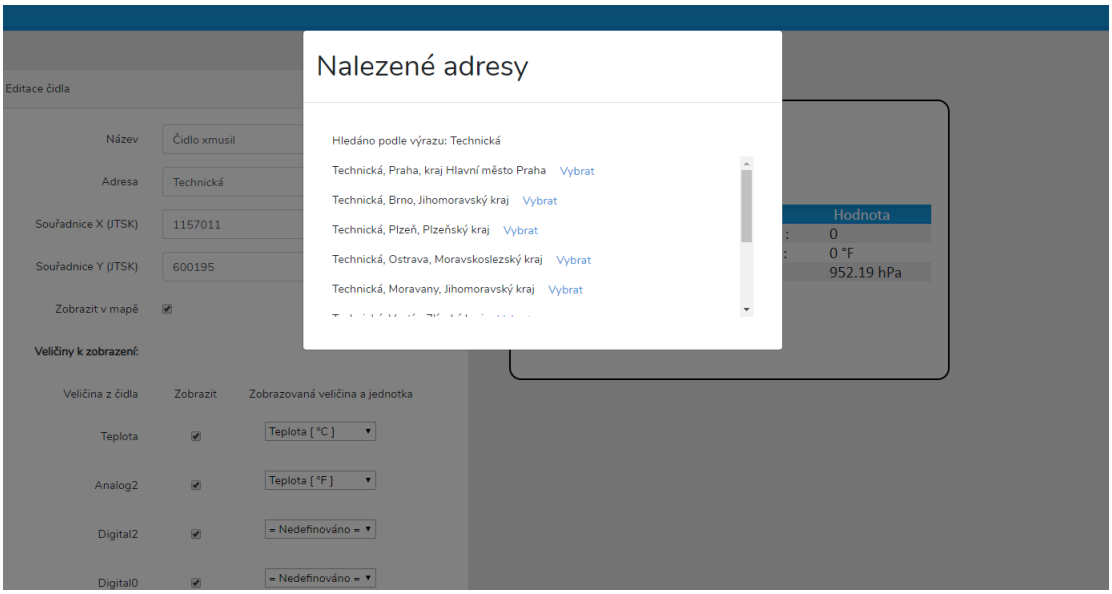
Obr. 6.8: Administrační část - Čidla editace

jednotek. Způsob, který jsem popsal výše, byl vymyšlen na základě sdělení kolegou Bc. Alešem Musilem, který nevěděl, co se připojí na jednotlivé piny na desce čidla a tudíž nebyl schopen naprogramovat jednotlivé názvy veličin. Administrátor tak může přiřadit k jednotlivým výstupům z čidla veličinu a jednotku, kterou představují. Tento překlad je pak reflektován ve vizitce nebo v zobrazených grafech. Dále je možné individuálně u každého čidla vybrat, které hodnoty zobrazovat. Na základě vyplněné adresy lze vypočítat souřadnice stanice. Vypočítání souřadnice stanice se provede po kliknutí na *Najdi adresu*, poté se otevře modální okno s výběrem přesně nalezených adres dle vyplnění hledané adresy (Obr. 6.9).

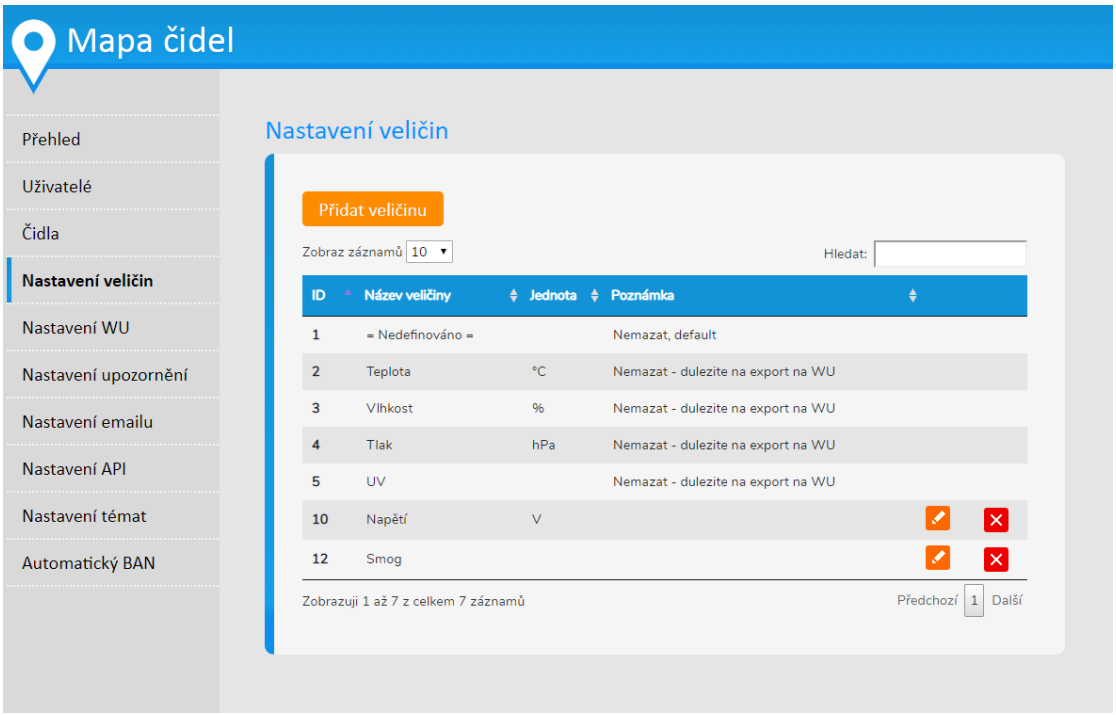
Po uložení se údaje ihned propíší a změny je možné ihned vidět v nastavení (zejména ve vizitce) v editaci čidla/stanice. Není tedy nutné přecházet do mapy a tam upravenou stanici hledat a kontrolovat, zda je vše nadefinované dle představ administrátora.

6.2.4 Správa veličin

V menu Nastavení veličin je možné přidat, upravovat anebo mazat jednotlivé definované veličiny. Jsou tu i defaultně definované veličiny pro export hodnot na portál Weather Underground (Obr. 6.10).



Obr. 6.9: Administrační část - Čidla nalezení souřadnic



Obr. 6.10: Administrační část - Veličiny

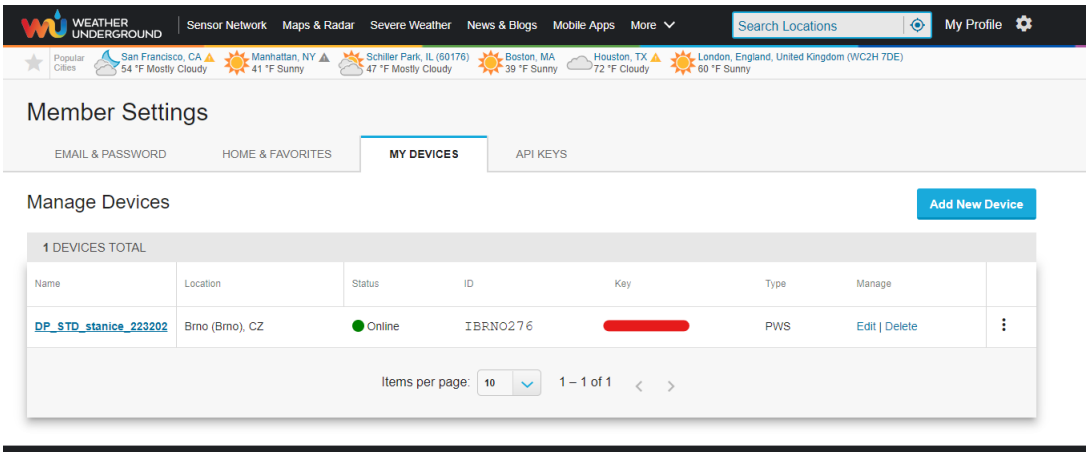
6.2.5 Správa nastavení Weather Underground

Nastavení pro portál Weather Undeground má vlastní část v menu, jelikož je vhodné mít oddělené toto separátní nastavení, i když se přímo váže na nastavení/editaci čidel. V menu je přehled nastavených čidel, kterých se export týká. Přidání nového nastavení probíhá způsobem vyplnění přihlašovacích údajů z portálu WeatherUnderground.com a vybrání čidla, které chceme k příslušným přihlašovacím údajům přidat.

Přidání nového čidla na portále WU probíhá v několika krocích:

- registrace uživatele na portál,
- registrace stanice.

Po registraci uživatele je nutné v registraci stanice vyplnit informace o stanici. Jedná se zejména, jestli se jedná o komerční stanici nebo jinou. Po vyplnění údajů jako je adresa, typ zařízení, výška nad zemí a potvrzení registrace, jsou vygenerovány přístupové údaje k čidlu (Obr. 6.11). Tyto údaje pak vložíme do administrace aplikace a tím provážeme aplikaci s webovou službou Weather Underground.



Obr. 6.11: Registrace čidla na portále Weather Underground

Odesílání dat na portál Weather Underground probíhá pomocí HTTP Requestu GET. Bohužel jsem nenašel technickou dokumentaci, jak sestavit tento request na oficiálních stránkach Weather Underground, ale na webových fórech pro arduina nebo Raspberry pi. [40]

O odesílání dat na portál se stará kontroler `ExportToWUController`. Ten má za úkol získat data od posledního exportu, sestavit request a ten odeslat. Jelikož každý request trvá cca 1 sec bylo nutné najít způsob, jak tento proces automatizovat. Proto byl vytvořen cron, který každou hodinu zavolá právě tento kontroler.

Vytvoření cronu je možné přímo ve frameworku Laravel.
Příkazem $php\ \ artisan\ \ make:command\ \ NÁZEV\_COMMANDU$, v případě apli-

kace `uploadData:Wu`(Obr. 6.12) , vytvoříme command, který bude spouštěn serverem. V třídě commandu poté definujeme název, popis a jaká metoda se má volat. Registraci commandu je nutné poté provést v souboru *app/Console/Kernel.php*, kde mimo jiné i definujeme, v jakých časových okamžicích se má cron spustit. Pro tuto aplikaci byl zvolen hodinový spouštěcí interval.

```
uploadData
 uploadData:WU        Upload all data from mqtt_data_request to WU
vendor
 vendor:publish       Publish any publishable assets from vendor packages
view
 view:cache           Compile all of the application's Blade templates
 view:clear           Clear all compiled view files
spurny@morana:/var/www/html/DP_STD$
```

Obr. 6.12: Část výpisu služeb pomocí příkazu php artisan list

Poslední část nastavení cronu spočívá v přidání commandu do crontable serveru. Po přihlášení na server do commandline zadáme příkaz `crontab -e`. Potvrdíme otevření administrátorským heslem a vybereme editor k úpravě. Pro přidání Laravelového cronu je potřeba přidat řádek:

$* \ * \ * \ * \ * \ cd \ /ADRESAR \ \&\& \ php \ artisan \ schedule:run \ >> \ /dev/null \ 2>\&1$

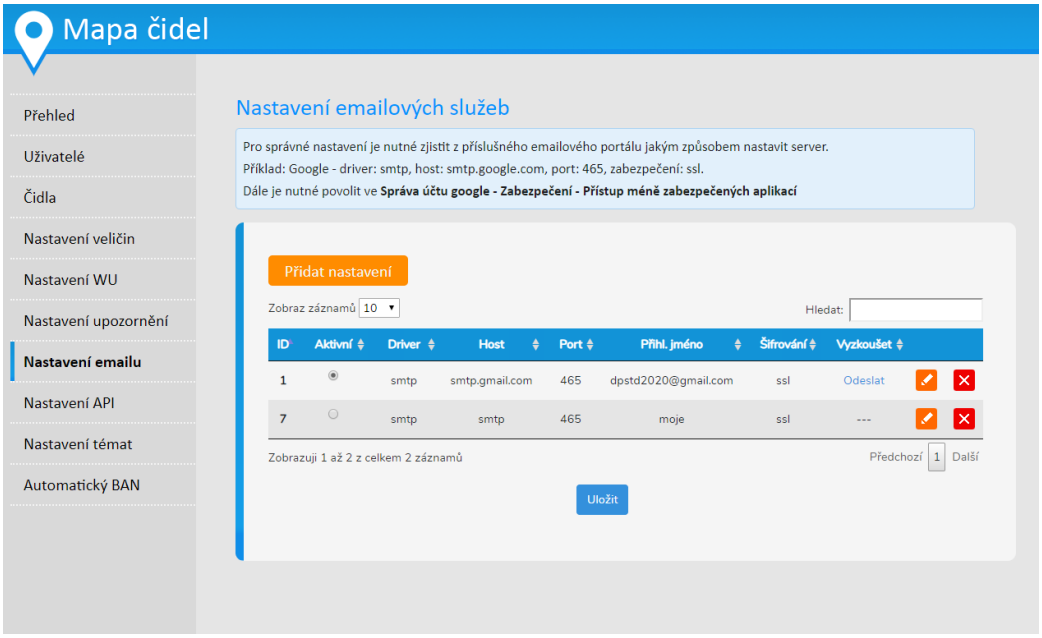
hvězdičky na začátku znamenají, že cron se spouští v závislosti na rozvrhu spuštění definovaný v commandu. [41]

6.2.6 Nastavení emailu

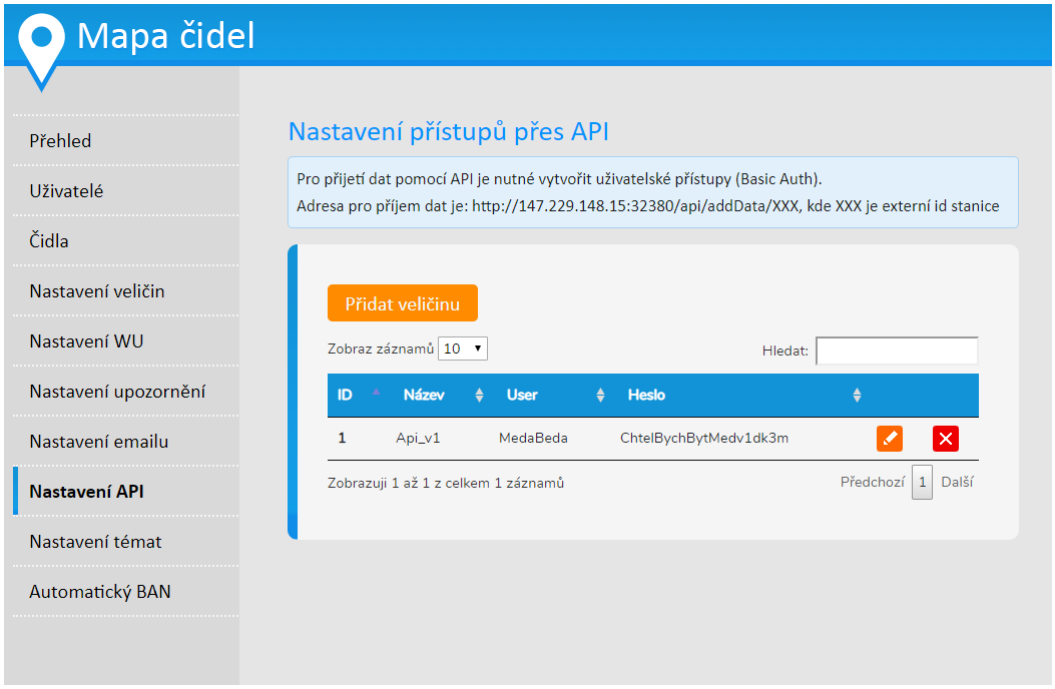
Na aplikaci byl kladen požadavek, aby v aplikaci bylo možné uživatelem nastavit odesílání emailů. Tudíž uživatel může nastavit vlastní odesílací server. Pro testy byl využit SMTP server a emailový klient společnosti Google (Obr. 6.13). Zde je po vyplnění možnost odeslat testovací email. Pokud je emailový server správně nastaven, email odejde přihlášenému uživateli.

6.2.7 Aplikační rozhraní aplikace

V rámci zadání bylo nutné vytvořit další způsob příjmu dat aplikací. Bylo tedy vytvořeno aplikační rozhraní (API), které je vystaveno na adrese `DOMENA/api/addData/XXX`. Pro API byla použita stejná komunikační definice zpráv, jako je při příjmu dat z MQTT serveru. V rámci adresy je odesláno identifikační číslo čidla/stanice a v datech jsou odeslána data v JSON struktuře. Pro možnost příjmu dat je nutné vytvořit přístupové údaje pro API. Je zde využita metoda basic auth, kde je definováno přihlašovací jméno a heslo. Správa těchto přihlašovacích údajů je v menu Nastavení API (Obr. 6.14).



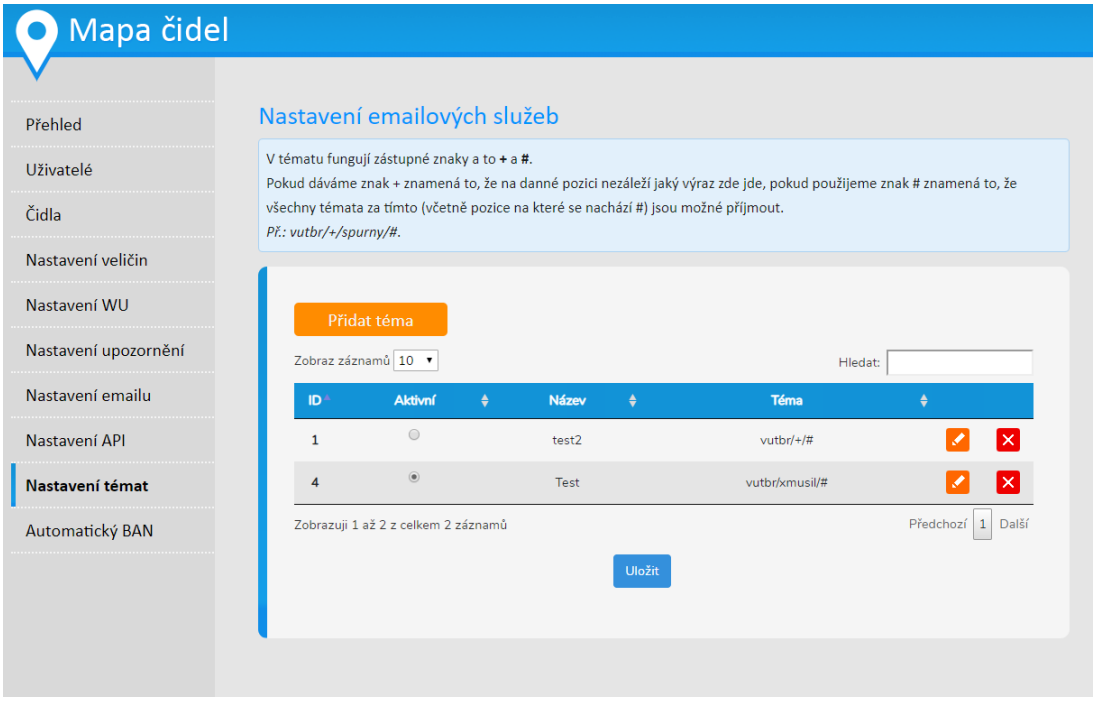
Obr. 6.13: Administrační část - Nastavení emailu



Obr. 6.14: Administrační část - Nastavení API

6.2.8 Nastavení témat pro přijímání dat z MQTT serveru

Definici nastavení témat, které jsou přijímány v rámci aplikace, je možné provést v menu Nastavení témat. V tomto menu definujeme, jak má téma vypadat a také zde vybíráme aktuální použité téma. Pokud není vybrané žádné téma, aplikace přijímá všechny zprávy (použit zástupný znak #)(Obr. 6.15).



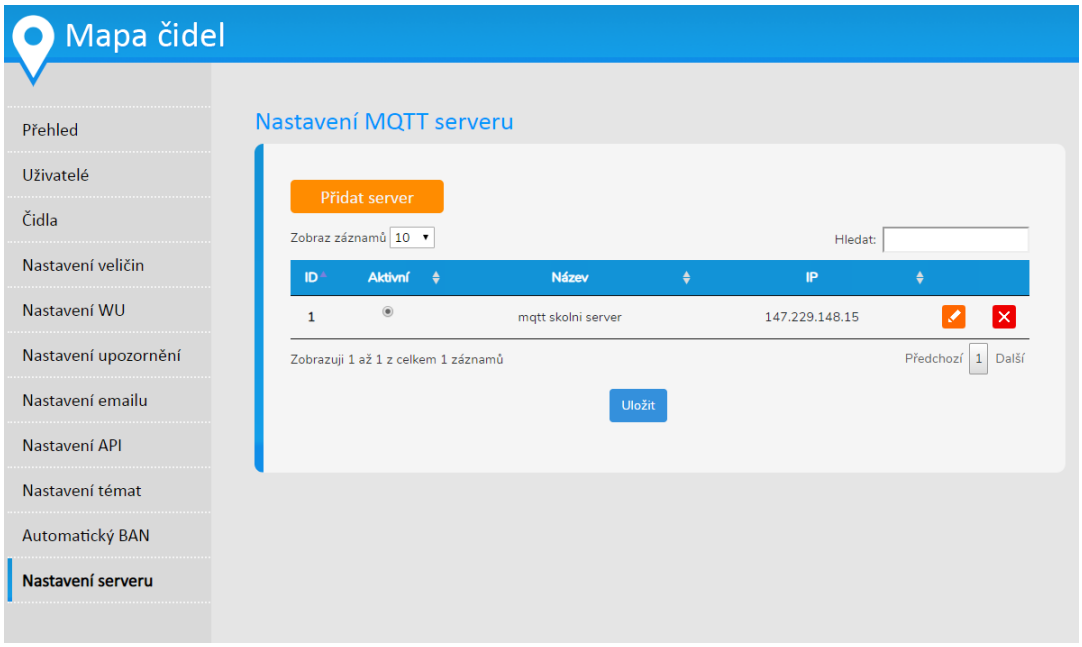
Obr. 6.15: Administrační část - Nastavení témat

6.2.9 Nastavení MQTT serveru

V tomto menu může uživatel nastavit MQTT server pro příjem dat. Tato IP adresa se propíše do databáze a z té následně služba při inicializaci příjmu dat vezme aktuální aktivní server. (Obr. 6.16).

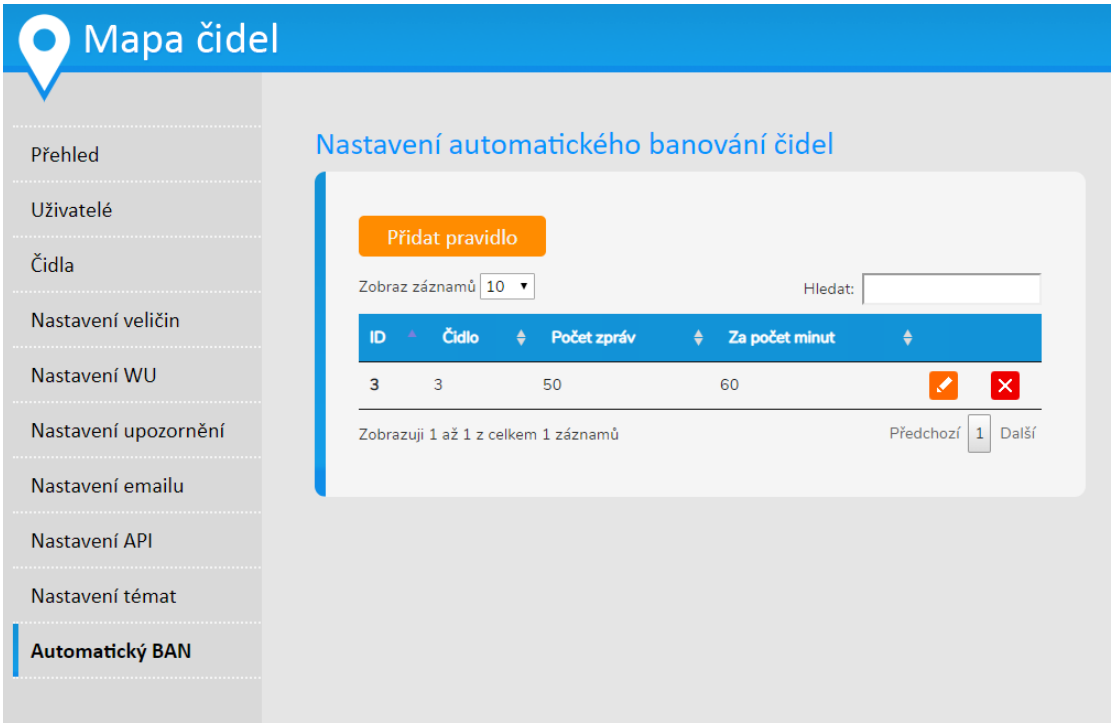
6.2.10 Zabezpečení aplikace při příjmu dat

V rámci příjmu dat aplikace není možné vytvořit mezi aplikací a serverem ověření autenticity zpráv (například unikátním tokenem každého čidla, který by se měnil). Zprávy, které aplikace přijímá jsou přímo z MQTT serveru, který by měl zajišťovat správné přijmutí dat a jejich ověření. Zprávy, které MQTT server odesílá, není možné nijak validovat kromě identifikačního čísla. Proto byla realizována pravidla pro zabezpečení aplikace jako takové.



Obr. 6.16: Administrační část - Nastavení serveru MQTT

Zprávy, které jsou přijaty z čidla, musí splňovat podmínky komunikačního protokolu, identifikátor čidla musí být zaregistrovaný v aplikaci a nesmí být zabanovaný. Při prvotní zprávě od čidla (unikátní ID) je čidlo zaregistrováno, skryto v zobrazení mapy a další příjem dat je zabanovaný. Potom, co administrátor ověří, zda-li je čidlo správné, mu doplní informace o adrese, odblokuje příjem dat a zobrazí jej v mapě. Pod pojmem zabanování příjmu dat se rozumí, že na rozhraní aplikace je kontrola, zda je možné přijímat data od tohoto ID, pokud ne data jsou zahozena. Tudíž nyní není možné server zahltit registrací čidel a následným útokem pomocí zahlcení posílanými zprávami. Dále je možné v administračním rozhraní nastavit maximální počet přijatých zpráv u jednotlivých čidel za určitý časový úsek. Tato ochrana spočívá v tom, že pokud útočník odhalí ID, které je zaregistrované a povolené v aplikaci a zahltí MQTT server zprávami, aplikace díky těmto definicím odpojí čidlo a nebudou přijímána žádná data. Tudíž je to ochrana proti zahlcení aplikace zprávami (Obr. 6.17).



Obr. 6.17: Administrační část - Nastavení zabezpečení

7 Instalační balíček

V rámci výstupu diplomové práce bylo zadáno aplikaci odevzdat ve formě zdrojových kódů i instalačního balíčku pro systém Linux ve verzi Ubuntu. Instalaci můžeme rozdělit do dvou částí a to instalaci prostředí (linuxového serveru) a samotné aplikace. V rámci vývoje bylo zkoušeno vytvořit instalační balíček celého linuxového serveru. To znamená, že na nainstalovaný linuxový server pouze dodáme instalační debianový balíček, který obsahuje, jak instalaci všech nutných softwarů, tak zároveň i instalaci aplikace. Aplikace je možné instalovat na server s nainstalovaným systémem Ubuntu 18.04.4 LTS.

Během zpracovávání diplomové práce bylo vyzkoušeno několik způsobů tvorby deb a rpm balíků. Byly vyzkoušeny jak grafické programy, které mají vytvořit debianový balíček, tak pouze terminálové. Pro tvorbu debianového balíčku byl nakonec zvolen terminálový způsob. Bylo nutné nainstalovat aplikace:

- build−essential,
- devscripts,
- debhelper.

Po instalaci byla vytvořena adresářová struktura. Hlavní adresář je název budoucího balíku. Tento balíček je potřeba pojmenovat s číselnou verzí aplikace. V případě diplomové práce byl tedy hlavní adresář pojmenován dp−1.0. Do toho adresáře je potřebné vytvořit dva adresáře a to DEBIAN a opt. Do adresáře DEBIAN byly vytvořeny dva soubory a to control a postinst. V souboru control jsou základní údaje o aplikaci, jako je například název, jméno a email autora. V souboru postinst je definované, co se má při instalaci stát. Tudíž v něm jsou definované příkazy na kopírování souborů k nastavení apache2, k vytvoření služeb a nebo samotné aplikace. V adresáři opt jsou pak nutné soubory, které jsou potřebné k instalaci. Po nastavení konfiguračních souborů a nastavení správných práv pro jednotlivé soubory (doporučuje se nastavit práva pomocí chmod 0755) se nechá vytvořit balíček. Příkaz pro vytvoření balíčku je `dpkg−deb −−build dp-1.0/`. Soubor dp−1.0.deb se vytvoří do adresáře, kde se nachází výchozí hlavní adresář.

Systém linux neumožňuje dělat v rámci instalace další instalaci, tudíž nebylo možné vytvořit instalační balíček, který obsahuje další instalace v podobě apache2, mariadb serveru, php 7.3 a dalších. Proto byl vytvořen instalační bashový skript, který je možné spustit příkazem `sudo sh install_prostredi.sh`. Tento skript nainstaluje veškeré potřebné softwary. [42, 43]

Pokud by nebylo možné využít tento skript je nutné, aby na serveru byly nainstalované aplikace pomocí příkazu: `apt−get install NÁZEV`.

Jedná se o následující aplikace:

- apache2
- mariadb−server
- composer
- cron
- php7.3
- npm
- php7.3−mbstring
- php7.3−xml
- php7.3−pdo−mysql
- git

Po připravení serverových aplikací je možné instalovat debianový balíček s webovou aplikací. Je vhodné ho pustit přes terminál pomocí příkazu
`sudo dpkg -i dp-1.0.deb`, tento způsob se doporučuje, obzvláště pokud není striktně dodržen celý postup instalace, tak lze díky terminálu odhalit chyby, které vznikly při instalaci (například na systému probíhá jiná instalace). V rámci této instalace se nastaví Apache2, nastaví se defaultní nastavení php7.3, pokud není, vytvoří se databáze, nahrají se databázové struktury, nastaví se služba pro přijímání MQTT zpráv ze serveru, nainstaluje se pomocí aplikace composer celá webová aplikace, nastaví se cron pro odesílání informací na portál Weather Underground a v posledním kroku se restartují, popřípadě se spustí příslušné služby. V tuto chvíli je celá aplikace dostupná v prohlížeči na adrese localhost.

Pokud nechceme využít instalaci pomocí skriptu install_prostredi.sh a debianového balíčku, je možné celou aplikaci nainstalovat pomocí jednoho bashového skriptu, který se spouští stejným způsobem, jako je výše zmíněný, a musí být spuštěn skript install.sh.

Po instalaci je možné se do aplikace příhlásit pomocí přihlašovacích údajů a to emailu `dpstd2020@gmail.com` a hesla `testtest`. Které je vhodné po přihlášení změnit, protože je to defaultně známé heslo. V rámci aplikace, pokud se změní MQTT server, je nutné restartovat službu pro příjem dat z MQTT serveru. Tuhle část může udělat pouze administrátor serveru, který se přihlásí na server a pomocí příkazu `sudo systemctl restart mqtt_subscribe.service` ji restartuje.

Závěr

Diplomová práce se věnuje návrhu a implementaci aplikace pro sběr telemetrických dat.

V úvodní teoretické části diplomové práce je popsán výběr komunikačního protokolu a porovnání s ostatními protokoly, které jsou využívány pro podobnou komunikaci. Na základě porovnání protokolů byl zvolen a podrobně nastudován komunikační protokol MQTT. V rámci teoretické části následuje výběr a popsání technologií, které jsou nutné pro vývoj a samotný návrh aplikace. Praktická část se zabývá vývojem a spuštěním jak webové aplikace, tak služeb pro linuxový server.

Praktická část je definována ve dvou vrstvách: 1. správa a konfigurace linuxového serveru s instalací všech potřebných aplikací, 2. samotnou webovou aplikaci pro zobrazení dat.

Pro vývoj aplikace bylo nutné nakonfigurovat linuxový server. Zejména se jednalo o správné nastavení služeb pro aplikaci, mezi něž patřily služby instalované z repositářů (apache2, mysql,...) a vlastní vyvinutá služba pro příjem dat z MQTT serveru. Dále byl nastaven automat (cronjob), který periodicky kontroluje a odesílá varovné emaily dle definice v aplikaci.

Webová aplikace je rozdělena do dvou částí, a to do veřejné a administrační. Veřejná část obsahuje mapu od společnosti Seznam.cz, ve které jsou zobrazena aktivní čidla informující uživatele o posledních přijatých a historických telemetrických hodnotách. V rámci administrační části je možné definovat jednotlivé prvky aplikace, jako je například definice zobrazených hodnot ve vizitce čidla, přiřazení veličin k přijatým hodnotám, možnost registrace, zobrazení v mapě či případné zakázání příjmu dat od konkrétního čidla.

Aplikace je psána v jazyce PHP s využitím frameworku Laravel, jako databázový systém byl použit MySQL (MariaDB). Dále byl použit značkovací jazyk HTML a skriptovací jazyk JavaScript s využitím technologie AJAX pro interaktivní ovládání. Taktéž podkladová mapa od společnosti Seznam.cz a jejich API je tvořena JavaScriptem.

K diplomové práci jsou dostupné zdrojové kódy, instalační debianový balíček aplikace pro linuxovou distribuci Ubuntu a bashový skript pro instalaci prostředí serveru i celé aplikace včetně nutných aplikací pro server.

Všechny cíle diplomové práce byly splněny a byl spuštěn pilotní provoz celé aplikace, jak s daty z reálného čidla, tak i s testovacími daty vytvořenými pro API.

Literatura

[1] Weather Underground [online]. 2014 [cit. 2020-05-28]. Dostupné z: <https://www.wunderground.com/>

[2] VOJÁČEK, Antonín. Základní úvod do oblasti internetu věcí (IoT). *Automatizace hw* [online]. 2016, 2016-09-16 [cit. 2019-11-30]. Dostupné z: <https://automatizace.hw.cz/zakladni-uvod-do-oblasti-internetu-veci-iot.html>

[3] *CoAP* [online]. 2014 [cit. 2019-11-30]. Dostupné z: <https://coap.technology/>

[4] XMPP: A Communication Protocol for the IoT. *Opensource foru* [online]. 2019, 2019-10-03 [cit. 2019-11-30]. Dostupné z: <https://opensourceforu.com/2019/10/xmpp-a-communication-protocol-for-the-iot/>

[5] EARLS, Alan R. XMPP: IoT protocol winner, or second place to MQTT? *IoT Agenda* [online]. 2016, 2016-11-14 [cit. 2019-11-30]. Dostupné z: <https://internetofthingsagenda.techtarget.com/feature/XMPP-IoT-protocol-winner-or-second-place-to-MQTT>

[6] POWELL, Hugh. A quick and dirty introduction to ZeroMQ. *Scott Logic* [online]. 2015, 2015-04-15 [cit. 2019-11-30]. Dostupné z: <https://blog.scottlogic.com/2015/03/20/ZeroMQ-Quick-Intro.html>

[7] ZeroMQ Feature List. *ZeroMQ* [online]. 2014, 2016-04-29 [cit. 2019-11-30]. Dostupné z: <http://wiki.zeromq.org/docs:features>

[8] Introducing the MQTT Protocol - MQTT Essentials: Part 1. *Hivemq* [online]. 2015, 2015-01-12 [cit. 2019-11-10]. Dostupné z: <https://www.hivemq.com/blog/mqtt-essentials-part-1-introducing-mqtt/>

[9] Client, Broker / Server and Connection Establishment - MQTT Essentials: Part 3. *Hivemq* [online]. 2019, 2019-01-17 [cit. 2019-11-10]. Dostupné z: <https://www.hivemq.com/blog/mqtt-essentials-part-3-client-broker-connection-establishment/>

[10] Publish & Subscribe - MQTT Essentials: Part 2. *Hivemq* [online]. 2015, 2015-01-19 [cit. 2019-11-10]. Dostupné z: <https://www.hivemq.com/blog/mqtt-essentials-part2-publish-subscribe/>

[11] Last Will and Testament - MQTT Essentials: Part 9. *Hivemq* [online]. 2015-03-09 [cit. 2019-11-10]. Dostupné z: <https://www.hivemq.com/blog/mqtt-essentials-part-9-last-will-and-testament/>

[12] MQTT Publish, Subscribe & Unsubscribe - MQTT Essentials: Part 4. *Hivemq* [online]. 2015, 2015 [cit. 2019-11-10]. Dostupné z: <https://www.hivemq.com/blog/mqtt-essentials-part-4-mqtt-publish-subscribe-unsubscribe/>

[13] MQTT Topics & Best Practices - MQTT Essentials: Part 5. *Hivemq* [online]. 2019, 2019-08-20 [cit. 2019-11-10]. Dostupné z: <https://www.hivemq.com/blog/mqtt-essentials-part-5-mqtt-topics-best-practices/>

[14] Quality of Service 0,1 & 2 - MQTT Essentials: Part 6. *Hivemq* [online]. 2015, 2015-02-16 [cit. 2019-11-10]. Dostupné z: <https://www.hivemq.com/blog/mqtt-essentials-part-6-mqtt-quality-of-service-levels/>

[15] 8 Online MQTT Brokers: Your IoT (Connected Objects) in the Cloud. *DIY Projects* [online]. 2017, 2017-02-14 [cit. 2019-11-10]. Dostupné z: <https://diyprojects.io/8-online-mqtt-brokers-iot-connected-objects-cloud/>

[16] *MQTT Brokers/Servers and Cloud Hosting Guide* [online]. 2019 [cit. 2019-11-10]. Dostupné z: <http://www.steves-internet-guide.com/mqtt-hosting-brokers-and-servers/>

[17] R. A. Light, "Mosquitto: server and client implementation of the MQTT protocol,"The Journal of Open Source Software, vol. 2, no. 13, May 2017, DOI: <https://joss.theoj.org/papers/10.21105/joss.00265>

[18] *Trends in the usage of server-side programming languages* [online]. [cit. 2019-11-1]. Dostupné z: <https://w3techs.com/technologies/history_overview/programming_language>

[19] ČÁPKA, David. 1. díl - Úvod do objektově orientovaného programování v PHP. In: *Itnetwork* [online]. 2013 [cit. 2017-10-23]. Dostupné z: <https://www.itnetwork.cz/php/oop/php-tutorial-uvod-do-objektove-orientovaneho-programovani>

[20] KULHAN, Jakub. OOP v PHP. In: *Programujte* [online]. 2009 [cit. 2019-10-26]. Dostupné z: <http://programujte.com/clanek/2009113001-oop-v-php/>

[21] MARKIEWICZ, Marcus Eduardo a Carlos. J.P. LUCENA. *Object Oriented Framework Development* [online]. 2010-02-25, , 12 [cit. 2019-11-10]. Dostupné

z: <https://www.cos.ufrj.br/~toacy/material/Object-Oriented%20Framework%20Development.pdf>

[22] The State of Developer Ecosystem 2019. *Jetbrains* [online]. 2019 [cit. 2019-11-10]. Dostupné z: <https://www.jetbrains.com/lp/devecosystem-2019/php/>

[23] ZAFAR, Salman. A simple Laravel 5 and 6 Library to connect/publish/subscribe to MQTT broker. *Packagist* [online]. 2019, 2019-10-29 [cit. 2019-11-10]. Dostupné z: <https://packagist.org/packages/salmanzafar/laravel-mqtt>

[24] Laravel. *Laravel* [online]. [cit. 2019-11-24]. Dostupné z: <https://laravel.com/>

[25] *MySQL databáze - český manuál* [online]. [cit. 2019-10-27]. Dostupné z: <http://www.junext.net/mysql/>

[26] ZAJÍC, Petr. MySQL (4) - něco terminologie. In: *Linuxsoft.cz* [online]. 2005 [cit. 2017-10-30]. Dostupné z: <http://www.linuxsoft.cz/article.php?id_article=744>

[27] Syntaxe HTML. *Jak Psát Web* [online]. [cit. 0179n. l.]. Dostupné z: <https://www.jakpsatweb.cz/html/syntaxe.html>

[28] ČÁPKA, David. Úvod do JavaScriptu. In: *Itnetwork* [online]. 2013 [cit. 2019-11-2]. Dostupné z: <https://www.itnetwork.cz/javascript/zaklady/javascript-tutorial-uvod-do-javascriptu-nepochopeny-jazyk/>

[29] HTML & CSS. *W3C* [online]. [cit. 2019-11-05]. Dostupné z: <https://www.w3.org/standards/webdesign/htmlcss>

[30] Zápisy CSS. *Jak Psát Web* [online]. [cit. 2019-11-05]. Dostupné z: <http://polopate.jakpsatweb.cz/index.php?page=zapis-css#externi>

[31] CSS selektory, pseudotřídy a pseudoelementy. *Interval.cz* [online]. 2002 [cit. 2019-11-05]. Dostupné z: <https://www.interval.cz/clanky/css2-selektory-pseudotridy-a-pseudoelementy/>

[32] *API mapy dokumentace* [online]. Praha, 2018 [cit. 2019-11-11]. Dostupné z: <https://api.mapy.cz/>

[33] Laravel dokumentace *Laravel.com* [online]. [cit. 2020-04-10]. Dostupné z: <https://laravel.com/docs/7.x/installation>

[34] JavaScript & CSS Scaffolding. *Laravel* [online]. 2020 [cit. 2020-04-18]. Dostupné z: <https://laravel.com/docs/7.x/frontend>

[35] ConsoleTVs/Charts. *GitHub* [online]. 2019 [cit. 2020-04-18]. Dostupné z: <https://github.com/ConsoleTVs/Charts>

[36] LaravelDaily/laravel-charts. *GitHub* [online] [cit. 2020-04-18]. Dostupné z: <https://github.com/LaravelDaily/laravel-charts>

[37] Highcharts. *highcharts* [online]. [cit. 2020-04-18]. Dostupné z: <https://www.highcharts.com/docs/index>

[38] Highcharts License. *highcharts* [online] [cit. 2020-04-18]. Dostupné z: <https://shop.highsoft.com/faq#Non-Commercial-0>

[39] Salmazafar949. *MQTT-Laravel*. GitHub [online]. [cit. 2020-04-18]. Dostupné z: <https://github.com/salmanzafar949/MQTT-Laravel>

[40] *Weather Underground Upload*. Campbellsci [online]. 2008, 2008-09-24 [cit. 2020-04-19]. Dostupné z: <https://www.campbellsci.com/forum?forum=1&l=thread&tid=9>

[41] HAMZA, Ali. *How To Set Up Task Scheduling With Cron Job In Laravel*. Tutsforweb [online]. 2019, 2019-10-09 [cit. 2020-04-19]. Dostupné z:<https://tutsforweb.com/how-to-set-up-task-scheduling-cron-job-in-laravel/>

[42] MUKHAMETSHIN, Marath, ed. Create .deb Package for Femhub. GitHub [online]. 2013, 2013-10-13 [cit. 2020-05-01]. Dostupné z: <https://github.com/hpfem/hermes/wiki/Create-.deb-Package-for-Femhub/>

[43] SANANTHANA, Kosala. Introduction to Debian Maintainer Script Flow Charts. The Startup [online]. 2019-03-19 [cit. 2020-05-01]. Dostupné z:<https://www.leaseweb.com/labs/2013/06/creating-custom-debian-packages/>

Seznam symbolů, veličin a zkratek

API	Application Programming Interface
PHP	Hypertext Preprocessor
HTML	HyperText Markup Language
CSS	Cascading Style Sheets
HTTP	Hypertext Transfer Protocol
MQTT	Message Queuing Telemetry Transport
ZeroMQ	ZeroMQ Message Transport Protokol
XMPP	Extensible Messaging and Presence Protocol
CoAP	Constrained Application Protocol
P2P	Peer-to-peer
M2M	Machine-to-machine
UDP	User Datagram Protocol
TCP	Transmission Control Protocol
RFC	Request for Comments
IoT	Internet of Things
QoS	Quality of Service
XML	Extensible Markup Language
TLS	Transport Layer Security
PING	Packet InterNet Groper
EPL	Eclipse Public License
PDF	Portable Document Format
MIT	Massachusetts Institute of Technology
SQL	Structured Query Language
JS	JavaScript
WU	Weather Underground
NPM	Node.js package manager
SSH	Secure Shell
AJAX	Asynchronous JavaScript and XML
JSON	JavaScript Object Notation
JTSK	Systém jednotné trigonometrické sítě katastrální

Seznam příloh

A Obsah přiloženého souboru ZIP **69**

A Obsah přiloženého souboru ZIP

Adresář DP_STD obsahuje veškeré zdrojové kódy aplikace. Soubor mqttstd.sql obsahuje vyexportované data databáze, které jsou nutné pro spuštění aplikace, nachází se zde defaultní hodnoty. Soubory mqtt_subscribe.service a mqtt_subscribe.sh jsou spouštěcími soubory pro službu, která se připojí na MQTT server a dále data přijatá ze serveru zpracovává. Soubor install_prostredi.sh je určený pro instalaci všech nutných prostředí na serveru. Soubor install.sh provede kompletní instalaci prostředí serveru a celé aplikace. Debianový balíček dp-1.0.deb je instalačním balíčkem aplikace. Konfigurační soubory apache2.conf a 000-default.conf jsou soubory pro nastavení http serveru pod službou apache2.

- `/` ..... Kořenový adresář souboru ZIP
  - `install_DP_STD+deb+readme` ..... Složka se zdrojovými soubory
    - `DP_STD` ..... Zdrojové kódy PHP aplikace
    - `mqttstd.sql` ..... MySql dump databáze
    - `mqtt_subscribe.service` ..... Služba pro připojení na MQTT server
    - `mqtt_subscribe.sh` ..... Bash pro ukládání dat z MQTT serveru
    - `install_prostredi.sh` ..... Instalace prostředí pro aplikaci
    - `install.sh` ..... Kompletní instalace prostředí a aplikace
    - `dp-1.0.deb` ..... Instalační balíček aplikace
    - `apache2.conf` ..... Konfigurační soubor pro Apache2
    - `000-default.conf` ..... Konfigurační soubor pro Apache2
  - `readme.txt` ..... Soubor s instrukcemi pro instalaci a příhlášení