VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

DIPLOMOVÁ PRÁCE

Brno, 2019 Bc. Jaroslav Hájek

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

SYSTÉM INTELIGENTNÍ DOMÁCNOSTI
SMART HOME SYSTEM

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. Jaroslav Hájek

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. Ondřej Krajsa, Ph.D.

BRNO 2019

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Diplomová práce

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

Student: Bc. Jaroslav Hájek ***ID:*** 173649
Ročník: 2 ***Akademický rok:*** 2018/19

NÁZEV TÉMATU:

Systém inteligentní domácnosti

POKYNY PRO VYPRACOVÁNÍ:

Navrhněte a realizujte systém inteligentní domácnosti s prvky IoT a možností připojení zařízení pomocí standardu IEEE 802.15.4. Systém bude využívat Raspberry Pi. Pro ovládání a konfiguraci systému se bude využívat dotykový display.

DOPORUČENÁ LITERATURA:

[1] VENKATESWARAN, Sreekrishnan. Essential Linux device drivers. Upper Saddle River: Prentice Hall, c2008,714 s. ISBN 978-0-132-39655-4

[2] KHAN, Ashfaq A. Practical Linux programming: device drivers, embedded systems and the Internet. Hingham: Charles River Media, c2002, xv, 420 s. ISBN 1-58450-096-4.

Termín zadání: 1.2.2019 ***Termín odevzdání:*** 16.5.2019

Vedoucí práce: Ing. Ondřej Krajsa, Ph.D.
Konzultant:

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

ABSTRAKT

Tato práce se zabývá propojováním vzdálených zařízení měřící data různých typů od fyzikálních veličin jako je teplota, vlhkost až po data zobrazující uživateli zatížení procesoru či využití paměti systému. Dále se zabývá kontrolní logikou a závislostmi mezi zařízeními za použití Blockly. V práci je použito mnoho technologií například: MQTT, Websockets, GSM, Lightweight Mesh a další. Systém je založen na mikroslužbě nazvané Flask. Tato služba je aplikační rozhraní rozšiřující základní HTTP služby programovacího jazyka Python. Umožňuje tím prezentovat uživateli naměřená data pomocí grafů či předdefinovaných komponent pro zobrazování dat. Pro ovládání systému byl využit jednodeskový počítač Raspberry Pi spolu se sedmi palcovým dotykovým displejem.

KLÍČOVÁ SLOVA

Chytrá domácnost, MQTT, Websockets, GSM, Lightweight Mesh, Flask, Python, Raspberry pi, ESP8266, ESP32, Blockly, IoT

ABSTRACT

This work deals with connecting remote devices measure data of various types from physical quantities such as a temperature, a humidity to data displaying CPU or a memory usage of the system to the user. The system uses Blockly to controlling logic and dependencies between devices. In the work is used lots of technologies for examples: MQTT, Websockets, GSM, Lightweight Mesh and others. The system is based on microservice which named is Flask. Flask service is an application interface for HTTP services of Python programming language. This can provide measured data by graphs and predefined components for viewing data. For controlling system was used single-board computer Raspberry Pi with a multitouch 7-inch display.

KEYWORDS

Smart Home, MQTT, Websockets, GSM, Lightweight Mesh, Flask, Python, Raspberry pi, ESP8266, ESP32, Blockly, IoT

HÁJEK, Jaroslav. *Systém inteligentní domácnosti*. Brno, 2019, 63 s. Diplomová práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací. Vedoucí práce: Ing. Ondřej Krajsa, Ph.D.

PROHLÁŠENÍ

Prohlašuji, že svou diplomovou práci na téma „Systém inteligentní domácnosti“ 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 Ing. Ondřeji Krajsovi, Ph.D. za odborné vedení a konzultace, dále bych rád poděkoval Bc. Dagmar Stromšíkové za pevné nervy, Bc. Janu Benediktovi za propůjčení sady Turris gadgets a rodičům a sestře za podporu.

Brno ....

podpis autora

Obsah

Seznam obrázků

Seznam tabulek

Seznam výpisů

Úvod

Diplomová práce se zabývá řešením systému chytré domácnosti, napsaném ve velmi univerzálním skriptovacím jazyce Python. V dnešním světe miniaturizace vynikají velmi malé jednočipové obvody, které lze využít pro periodické odesílání dat ze senzorů různých veličin a na základě těchto dat činit určité akce. Tato práce se zabývá vytvořením systému, využívající různá rozhraní jako MQTT (Message Queuing Telemetry Transport), REST (Representational State Transfer), Lightweight mesh pro sběr veličin v chytré domácnosti. Systém je určen jak pro jednodeskový miniaturní počítač Raspberry Pi, tak pro jakýkoliv server umožňující spustit script v jazyce Python. Součástí této práce je napsat knihovny určené pro obsluhu malých WiFi zařízení, které jsou schopny umožnit registraci těchto zařízení do systému a během ní definovat měřené veličiny a to i s jejich vizuálním konceptem. Ten může systém reprezentovat jako výstupní rozhraní pro uživatele. Pro toto zobrazení bylo vytvořeno pár základních prvků, které lze definovat v registraci a to například senzory pro detekci otevřených dveří, rozsvícení žárovky, nebo sepnutí zásuvky. Součástí je také knihovna „*canvas-gauges*“ představující spoustu možností zobrazení teploměrů, grafu apod. Dále zde patří zobrazení polohy na mapě. V systému je zajištěna podpora více členů domácnosti a to včetně omezení jejich pravomocí, kde každý uživatel má možnost přidat svá vlastní zařízení pomocí API (Application programming interface) klíče vygenerovanému při registraci. To umožňuje každému členu domácnosti udělat si svoji vlastní chytrou domácnost. Pokud chceme použít raspberry pi s displejem, tak je možné použít předchystaný obraz paměťové karty, který obsahuje připravené služby detekující server v sítí a zobrazení nástěnky s připravenými měřenými veličinami. Toho bylo docíleno pomocí prohlížeče Chromium a to v jeho kiosek módu. Prohlížeč Chromium byl zvolen, protože jako jediný obsahuje hardwarovou akceleraci obrazu na raspberry pi. Díky tomu je možné v něm použít například CSS (Cascading Style Sheet) animace, které v node-webkit, nebo webkit přes GTK (GIMP Toolkit) či QT (Quality of Service) nelze zobrazit. Po zavedení tohoto systému je načten program umožňující připojení do sítě. Po připojení do sítě systém čeká na ohlášení od serveru. Po přijetí tohoto ohlášení v síti se spustí Chromium v *kiosek* módu s přednastavenou adresou na tento systém v lokální síti. Pokud je server mimo naši lokální sít lze nastavit jeho adresu staticky.

1 Chytrá domácnost

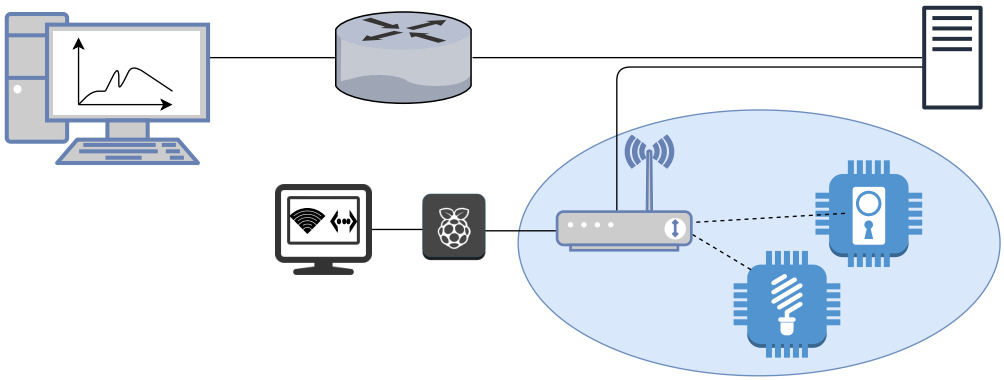
Myšlenka *Smart Home* (chytrá domácnost) zde byla již v polovině 18. století, kdy byla vyvinuta první automatická pračka a lidé se začali zamýšlet nad tím, jak své činnosti automatizovat. Od té doby, díky technologickému pokroku vznikla spousta zařízení, které nám v dnešním světě usnadňují život. Mezi tyto zařízení patří například mobilní telefon, který se již v dnešní době vypracoval do podoby multifunkčního zařízení usnadňující nám práci, jak v osobním, tak v pracovním životě. Dalším takovým zařízením můžeme považovat dnešní automobily, které dnes obsahují mnohem více senzorů než tomu bylo v letech minulých. Chytrá domácnost je tedy souhrn chytrých, mezi sebou komunikujících zařízení, usnadňující naši každodenní rutinu. Dnes se již také za pomocí strojového učení dokážou velice dobře přizpůsobovat našim rutinám a dle nich například nastavovat termostat nebo řídit osvětlení v domě [5].

1.1 IoT (Internet of Things)

Internet of Things je síť navzájem propojených senzorů, čidel, termostatů a spoustu jiných zařízení. Ty spolu mezi sebou komunikují a provádí automatizované činnosti na základě sběru dat. Dnes se do popředí dostávají komunikační technologie pracující na frekvenci 868 MHz, jako je například Lora či Sigfox. Tyto řešení umožňují odesílat menší objem dat na velkou vzdálenost, což je pro IoT typické. Protože není potřeba vždy dosahovat vysokých vzdáleností. Je možné použít i volně dostupnou frekvenci 433 MHz, která se často využívá pro spínání zásuvek, nebo pro lokální meteostanice. Tato frekvence má sice menší dosah, za to má dobrou penetraci a dokáže tedy lépe procházet i skrze překážky jako jsou stěny či nábytek. Další z možných variant je propojení zařízení skrze Bluetooth v jeho BLE (Bluetooth Low Energy) variantě, která dosahuje cca 150 metrů [2]. Ta také obsahuje spoustu profilů umožňujících komunikaci s fitness zařízeními, HID (Human Interface Device) zařízeními, různými zdravotními měřiči hladin látek v krvi, nebo třeba zajišující síťovou komunikaci pomocí profilu IPSP (Internet Protocol Support Profile) [6] [10].

2 Použité technologie

Mezi hlavními cíli práce byla snadná implementace vlastních klientských modulů. Toho bylo dosaženo pomocí REST API umožňující kompletní správu modulů. Dále byla požita technologie MQTT, která zajišťuje čtení hodnot a lokace do systému, nebo jejich zápis. Návrh takového systému je vyobrazen na Obr.2.1.



Obr. 2.1: Diagram komunikace mezi zařízeními.

2.1 MQTT (Message Queuing Telemetry Transport)

MQTT je protokol vytvořen pro komunikaci pomocí zpráv. Byl vyvinut na míru IoT, a proto šetří komunikaci mezi zařízeními. Styl komunikace je typu Publish/Subscribe (publikovavat/odebírat). Toho je dosaženo pomocí dvou typů zařízení a to Brooker a Agent [1].

- Brooker - zařízení sbírající všechna publikovaná data od agentů.
- Aget - zařízení posílající či odebírající zprávy.

Množství senzorů je odděleno pomocí topicu. Jedná se o identifikátor agentů a má následující formát:

```
/CZ/Brno/adresa/pokoj/teplota
```

Kde jsou jednotlivé vrstvy oddělené lomítkem. Pomocí zástupných znaků # a +, kde zástupný + nahrazuje jednu vrstvu topicu.

```
/CZ/Brno/adresa/+/teplota
```

Díky tomu je například možné si zažádat odběr teplot ve všech pokojích na dané adrese. Zástupný znak # nahrazuje jednu vrstvu, + nahrazuje všechny další vrstvy.

```
/CZ/Brno/#
```

Zažádání si odběru o takto definovaný topic zapříčiní příjem všech veličin na všech adresách v Brně [9].

Pokud se zařízení typu klient chce připojit do sítě, musí odeslat serveru (aplikace typu brooker) **Connect Command**. Tento packet obsahuje základní informace pro připojení k brookeru. Jsou to:

- User Name - uživatelské jméno, definované v brookeru určené pro authentizaci agentů.
- Password - heslo, přiřazená fráze k uživatelskému jménu na brookeru, taktéž určené pro authentizaci agentů.
- QoS (Quality of Service) Level
  - **0** Klient odešle brookru zprávu a nečeká na potvrzení. Není tedy zajištěno doručení zprávy.
  - **1** Po odeslání zprávy je zajištěno že zpráva dojde na brooker alespoň jednou. Všichni, kdo budou tento topic odebírat, ale musí počítat s tím, že zpráva může dojít i vícekrát.
  - **2** Zpráva s tímto QoS musí být doručena pouze jednou. Znamená to, že odběratelé topicu nemusejí počítat s tím, že zprávu mohou dostat vícekrát [11].

Brooker následně odpoví **Command Ack**, čímž potvrdí agentovi, že byl připojen do sítě. Během připojení může agent odesílat klientovi zprávy typu **Publish Message**. Ty jsou odeslány k Brookeru ve formě:

- Msg Len obsahuje počet znaků odeslané zprávy.
- Topic Length vyjadřuje počet znaků topicu.
- Topic je topic zprávy.
- Message je vlastní zpráva, kterou brooker přepošle odběratelům vždy ve stejné formě, v jaké přišla od agenta.

.

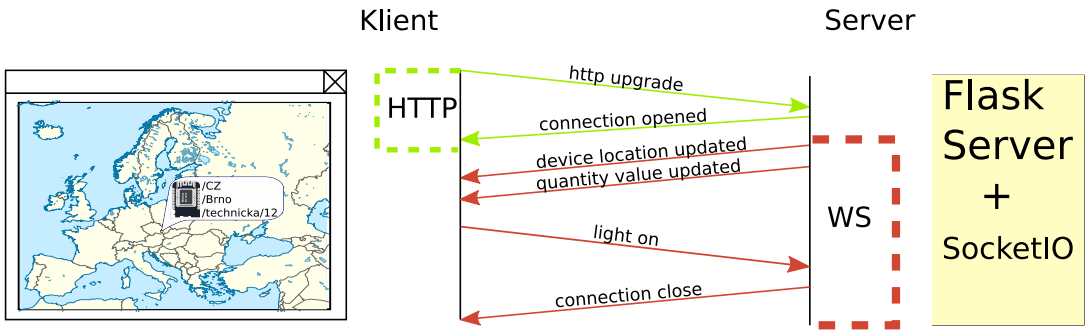
Tyto zprávy jsou následně od brookeru přeposlány všem odběratelům tohoto topicu. Další zprávou, která lze po připojení odeslat je **Subscribe Request**. Ta je použita k požádání brookera o odběr určitého topicu. Její forma vypadá takto:

- Msg Len obsahuje počet znaků odeslané zprávy.
- Message Identifier Identifikační číslo zprávy.
- Topic Len obsahuje počet znaků topicu
- Topic je vlastní topic zprávy.
- Requested QoS obsahuje číslo, které určuje v jakém módu chce agent zprávy přijímat [1].

2.2 WebSockets

HTML (Hypertext Markup Language) verze 5 přišlo s možností jak navazovat trvalé spojení mezi serverem a klientem a to pomocí WebSockets. Tato technologie vylepšuje standardní komunikaci protokolu HTTP (Hypertext Transfer Protocol) typu dotaz odpověď na možnost udržení trvalého spojení. Povýšení na WebSocket spojení je vyobrazeno na Obr.2.2. Podpora této funkcionality musí být zaručena na obou stranách, jinak se spojení nenaváže. Proto vznikly různé knihovny třetích stran, které ověřují zda-li je spojení mezi klientem a serverem aktivní. Pokud ne, tak v rámci zpětné kompatibility prohlížečů knihovny třetích stran přepnou na dříve velice používaný **long polling** [12], tedy způsob při kterém klient odesílal v krátkých intervalech dotaz na server. Zde se pokouší zjistit, zda nedošlo k nějaké změně. Tento způsob kladl zbytečně spousty požadavků, při kterých se muselo pořád dokola otvírat a zavírat spojení. Mezi takové knihovny třetích stran patří například SocketIO, který je použitý v tomto systému. SocketIO má výhodu ve stylu odesílání zpráv a to tak, že využívá podobně jako MQTT publikace a odběry. Tento systém vyžaduje spoustu interaktivity ze stran klientů, proto bylo vhodné využít WebSockets pro komunikaci ze stran serveru ke klientům. Jedná se o několik typu zpráv odesílaných klientovi [15]:

- **změna hodnoty u veličiny zařízení**, kde jsou odeslány informace o celém zařízení. Využívá se u zobrazovaní komunikace v reálném čase na mapě. **pro tuto možnost je nutné mít registrovaný odběr hodnot u zařízení**
- **změna hodnoty u veličiny**, kde jsou odeslány informace o konkrétní veličině. Používá se u zobrazování jednotlivých veličin na nástěnce aplikace. **pro tuto možnost je nutné mít registrovaný odběr hodnot u jednotlivých veličin.**.
- **změna lokace u zařízení**, kde jsou odeslány informace o lokaci zařízení. Využívá se pro změnu polohy zařízení na mapě. **pro tuto možnost je nutné mít registrovaný odběr lokace u zařízení**



Obr. 2.2: Navázání WebSockets spojení.

2.3 REST (Representational State Transfer)

Rozhraní REST API definuje přístup ke zdrojům (Resources). Zdroje jsou poskytovány jako odpovědi na požadavky směřující na konkrétní koncové body. Odpověď může být buď samostatný objekt, nebo kolekce objektů. Stejně jako protokol HTTP je založen na komunikaci typu požadavek - odpověď. Pro přístup k Resources jsou definovány 4 typy požadavku [8]:

- **GET (Retrieve)** - je metoda HTTP požadavku, kterou REST definuje pro přijetí zdroje. Příklad použití je za pomocí programu *curl*.

  ```
  curl -i -X GET https://sm.cz/API/1/uzivatele/12345.json
  ```

  Kdy je dotazujícímu se klientovi vrácena ze serveru odpověď ve tvaru JSON (JavaScript Object Notation), která může vypadat například takto:2.1

  ```
  {
      "id": 12345,
      "jmeno": "Dagmar",
      "prijmeni":"Stromšíková",
      "administrator":true
  }
  ```

  Code 2.1: Odpověď na požadavek GET.

- **POST (Create)** - je dotaz na server, aby vytvořil zdroj. Tento zdroj ještě neexistuje a tak, musí být použita adresa tzv. „endpointu“, jehož tvar bude v našem případě:2.2

```
curl -i -X POST
        --header "Content-Type: application/json"
        --data '{"jmeno": "Dagmar",
                 "prijmeni":"Stromšíková"}'
        https://sm.cz/API/1/uzivatele
```

Code 2.2: Požadavek typu POST na vytvoření uživatele.

- **DELETE** - pomocí metody DELETE je serveru oznámeno, že má smazat příslušný zdroj.

```
curl -i -X DELETE https://sm.cz/API/1/uzivatele/12345
```

- **PUT (Update)** - metoda PUT slouží k úpravě zdroje. Předávají se pouze parametry, které chceme přepsat. Příklad je velmi podobný metodě PUT2.3 [14].

```
curl -i -X PUT
        --header "Content-Type: application/json"
        --data '{"prijmeni":"Hájková"}'
        https://sm.cz/API/1/uzivatele/12345
```

Code 2.3: PUT požadavek na úpravu příjmení.

2.4 Python

Python je vysokoúrovňový programovací (skriptovací) jazyk, který používá dynamickou kontrolu datových typů. Interpret jazyka Python je napsán v několika jazycích, kde nejběžnější je tzv. „cpython“ napsaný v jazyce C. Existují ale i varianty napsané v jazyce Java (Jython), nebo interpret používající .NET nazvaný jako „IronPython“. Jazyk Python vznikal až do verze 3.0 ne úplně podle návrhu a tak byla vytvořena nová verze 3+ používající standardní návrh. Verze 2 je stále ve vývoji, kde přebírá některé funkce z verzí 3+. Podpora druhé verze je plánována do roku 2020. Následně je očekáván rychlejší přestup k verzi 3. Python je víceparadigmatický dynamicky interpretovaný jazyk. Znamená to, že ho lze směřovat pro používaní jak objektově orientovaných paradigmat, tak procedurálních či funkcionálních paradigmat. Záleží to na typu projektu a privilegii vývojáře. Python je jazyk vhodný pro učení a to díky jeho krátkého a jednoduchého zápisu. Inspirací tohoto jazyka byl jazyk ABC, který byl vyvinut hlavně pro výuku. V Python u jdou ale i přes jeho jednoduchost vytvářet i velice rozsáhlé aplikace. Jazyk Python lze pomocí překladače Cython přeložit do čistého C, což má výhody hlavně v rychlosti aplikace. Pokud v aplikaci otypujeme proměnné pomocí pomocných deklarací „**cdef**“, výsledný program může být až mnohonásobně rychlejší. Python má velkou podporu knihoven, kde velkou roli hraje její webový repozitář „pypi.org“. Tam může jakýkoliv zaregistrovaný uživatel nahrát své knihovny a prezentovat je tak komunitě, která je u Python u opravdu početná [18]. Základní datové typy:

- **int** = Číslo neomezené velikosti (limitem je paměť).
- **float** = desetinné číslo.
- **bool** = Může nabývat pouze hodnot True/False (Pravda/Nepravda).
- **string** = řetězec znaků (v Python u je i jeden znak typu string).
- **list** = Pole objektů (není vázáno typem, může obsahovat různé typy).
- **dict** = Slovník obsahuje páry klíč:hodnota (spolu s listem je syntaxe velice podobná jazyku JSON).
- **touple** = Množina chová se podobně jako list jenže je neměnný.

Tento systém potřebuje pro obsluhu všech požadavků pracovat asynchronně. Toho lze docílit několika způsoby. Mezi nejčastější patří rozdělení aplikace do vláken pomocí standardní knihovny **threading**. Jedná se o preemptivní multitasking (o přepínání vláken rozhoduje operační systém), nebo použití knihovny asyncio či greenlet. Asyncio je již implementován ve standardních knihovnách Python u, zde je možné pomocí speciální definice funkce **async def** pracovat ve smyslu tzv. coroutin a úloh. Pomocí funkce asyncio.run() lze spustit kontext, který dokáže přepínat mezi úlohami a corutinami v časech, kdy je funkce uspána. Díky tomu je možné obsluhovat více požadavků ve stejný okamžik. Flask má nejlepší implementaci pomocí knihovny

greenlet. Ta pracuje v tzv. „mikrovláknech“ jako nadstavba nad knihovnou gevent.

2.5 FLASK

Flask je microframework napsán v jazyce Python . Vznik toho frameworku byl zapříčiněn spojením dvou knihoven a to knihovny werkzeug a jinja2. Werkzeug je sada webových nástrojů umožňující zpracovat požadavek klienta a odpovědět na něj pomocí vygenerované odpovědi. Pomocí knihovny Jinja2 bylo následně umožněno vytvářet složitější odpovědi. Použití knihovny Flask je možné u jednoduchých, ale i u velmi složitých aplikací. Flask používá architekturu MVC (Model View Controller). Ta je definovaná jako architektura rozdělená na 3 odlišné části tak, aby šla každá část upravovat samostatně bez většího dopadu na ostatní. Jedná se o části **Model View** a **Controller**, kde modelová část se zabývá prací s daty. K tomu je většinou použit nějaký externí zdroj jako databáze či souborový systém. Flask nemá žádné pevně definované ORM (Object-relational mapping), neboli možnost relačního propojení mezi databází a objektově orientovaným programovacím jazykem. Tím je vývojáři umožněno, použití vlastního řešení či knihoven třetích stran. View neboli pohledová část architektury se zabývá samotnou prezentací dat klientům. Ve frameworku Flask je k tomu použit šablonovací systém Jinja2. Controller je určen k tomu, aby prováděl veškeré řídící záležitosti. Jednou z nich může být ta, kde po přijetí requestu správně vyhodnotí, kterou pohledovou funkci zvolit a jaké jí předá data získaná například z modelové vrstvy [4].

2.5.1 Šablony (Templates)

Flask používá pro vkládání dat do obsahu stránek šablonovací jazyk Jinja2, ten lze vidět například u DevOps nástrojů jako je Salt, nebo Ansible. Šablonovací jazyk Jinja2 rozlišuje tři základní značky pro úpravu šablony a to:

- **{{ název_proměnné }}** - používá se pro výpis proměnných či listů,
- **{% %}, {% %}** - je tag párový a používá se pro řídící struktury.
  - podmínka :2.4

```
1    {% if castka>2 %}
2        ano {{ castka }} je vetsi nez 2
3    {% else %}
4        ne {{ castka }} není vetsi nez 2
5    {% endif %}
```

Code 2.4: Podmínka v šablonovacím jazyku Jinja.

  - cyklus :2.5

```
    {% for polozka in polozky %}
        toto je {{ polozka }}
    {% endfor %}
```

Code 2.5: Cyklus v šablonovacím jazyku Jinja.

item **{# TODO: opravit chybu #}** - tento blok Jinja2 odstraní z výsledku. Je to komentář, který slouží pro orientaci autora kódu.

2.5.2 Cesty (Routes)

Definice cesty je dosaženo pomocí dekorátoru (to je funkce, která dokáže obalit jinou funkci) ve Flask projektech se cesta přiřazuje k aktuální aplikaci a vypadá takto:2.6

```
@app.route('/uzivatel/<str:uzivatel_jmeno>/profile')
    def uzivatel(uzivatel_jmeno):
        return "profil uživatele "+uzivatel_jmeno


```

Code 2.6: Přiřazení routy k pohledové funkci.

Pomocí proměnných v URL (Uniform Resource Locator) lze vytvářet dynamické URL. Není tedy potřeba vytvářet pro každého uživatele stránku s profilem. Stačí v URL předat uživatelské jméno. Stránka se přizpůsobí podle šablony.

2.5.3 Pohledy (Views)

Pohledy vytváří logickou mezivrstvu mezi cestami a šablonami, zde se přistupuje k modelům, zabezpečuje přístup a připravuje se vše, co v šabloně má být použito. Na této vrstvě máme přístup, jak k samotným požadavkům, tak i k prozatímním odpovědím. Kde můžeme například nastavovat cookies, nebo přidávat hlavičky, které mají být s odpovědí vráceny. U požadavků na stránku máme přístup ke čtení cookies a také k hlavičkám, kde lze zjistit například, který prohlížeč se na stránku dotazuje, dále k parametrům předané metodou GET (součást URL požadavku), nebo POST (přenesené v těle požadavku). Někdy není potřeba, aby pohled směřoval svůj výstup přes šablonu, lze tedy vrátit odpověď přímo z pohledu. Díky tomu lze vracet odpovědi ve tvaru: JSON, XML (Extensible Markup Language), nebo třeba datový proud, reprezentující obrázek či jiný typ. Pokud je aplikace psána tak, aby nepotřebovala žádné ostatní soubory, je možnost i obsah šablonových souboru vrátit přímo

z pohledové funkce a to pomocí funkce `render_template_string()`, kde prvním argumentem je parametr **source**. Ten představuje text reprezentující šablonu. Zbývající parametr ****context** je dynamický. To znamená, že přebírá všechny ostatní parametry a vrací je jako pole. Tento parametr slouží pro předání proměnných pro uvedený source.2.7

```
1    def uzivatelsky_profil(uzivatel_id):
2        return render_template("profile.html",
3                               user={"jmeno":"Jaroslav",
4                                     "prijmeni":"Hájek"})
```

Code 2.7: Základní pohledová funkce.

2.5.4 Modely (Models)

Vrstva modelů reprezentuje čtení, zápis nebo jakoukoliv jinou manipulaci s datovými zdroji. Nejčastěji bývá k jejich manipulaci použita knihovna **SQLALCHEMY**, která zprostředkovává veškerá data databáze ve formě jednoduchých Python objektů. Může však být použita například jako překladová vrstva pro ukládání dat do souboru formátu YAML (Ain't Markup Language), JSON nebo XML).

SQLALCHEMY

Základním prvkem SQLALCHEMY je to, že se chová jako překladová vrstva mezi databází a samotným jazykem Python. Pokud tedy napsána třída dědí ze třídy **db.Model**, chová se pro Python jako objekt a pro databázi jako tabulka. Ve třídě se definují jednotlivé objekty, které jsou reprezentovány jako sloupce databáze. Vytvoříme-li tedy model reprezentující uživatele, vypadal by například takto: 2.8

```
1 class User(db.Model):
2     id = db.Column(db.Integer, primary_key=True)
3     username = db.Column('username', db.String(20), unique=True)
4     password = db.Column('password', db.String(300))
5     email = db.Column('email', db.String(50), unique=True)
```

Code 2.8: Definice modelu knihovny sqlalchemy.

V této třídě definujeme u **id** uživatele primární klíč na hodnotu True. To je pro databázi označení, že má tento sloupec brát jako identifikační prvek uživatele. Pokud je primární klíč nadefinován a není řečeno jinak, nadefinuje se tento sloupec i jako auto-increment. To způsobí inkrementaci tohoto pole při každém vložení nového řádku. U dalších sloupců je datový typ definován z typů sqlalchemy reprezentující databázové datové typy. Dále obsahuje parametr **unique** značící zda má být sloupec unikátní či nikoliv. Pokud bychom se snažili zapsat nový objekt do databáze a při vkládali použili stejná data ve sloupci s parametrem **unique=True**, databáze by zapsání odmítla s chybou [3].

Pokud na základě této třídy vytvoříme objekt pomocí Python scriptu:

```
administrator = User(
    username = "administrator",
    password = hashlib.sha256(b"Secret_Passowrd").hexdigest(),
    email="hajda14@gmail.com")
```

Tak za pomocí funkce **db.session.add(administrator)** přidáme vytvoření objektu na seznam akcí pro databázi a pomocí **db.commit()** tyto akce nad databází provedeme. Reprezentace uživatele administrátor bude v databázi zapsána jako na Tab.2.1.

Následně lze model použít k operacím s databází a to pomocí pohledových funkcí. Pokud chceme vyhledat v databázi uživatele administrátor, uděláme to pomocí statických funkcí. Zde je ukázka takového pohledu: 2.9

```
@app.route("/users/<int:id>")
def user(id):
```

id	username	password	email
1	administrator	483d1b9a53651ef1bdce3...	hajda14@gmail.com

Tab. 2.1: Model v databázi.

```
person = User.filter_by(id=id).first()
return render_template_string("""
        <table>
        <tr><td> uzivatel </td><td> {{ user.username }}</td></tr>
        <tr><td>    email </td><td>    {{ user.email }}</td></tr>
        </table>
                              """,user=person)
```

Code 2.9: Pohledová funkce pracující s modelem.

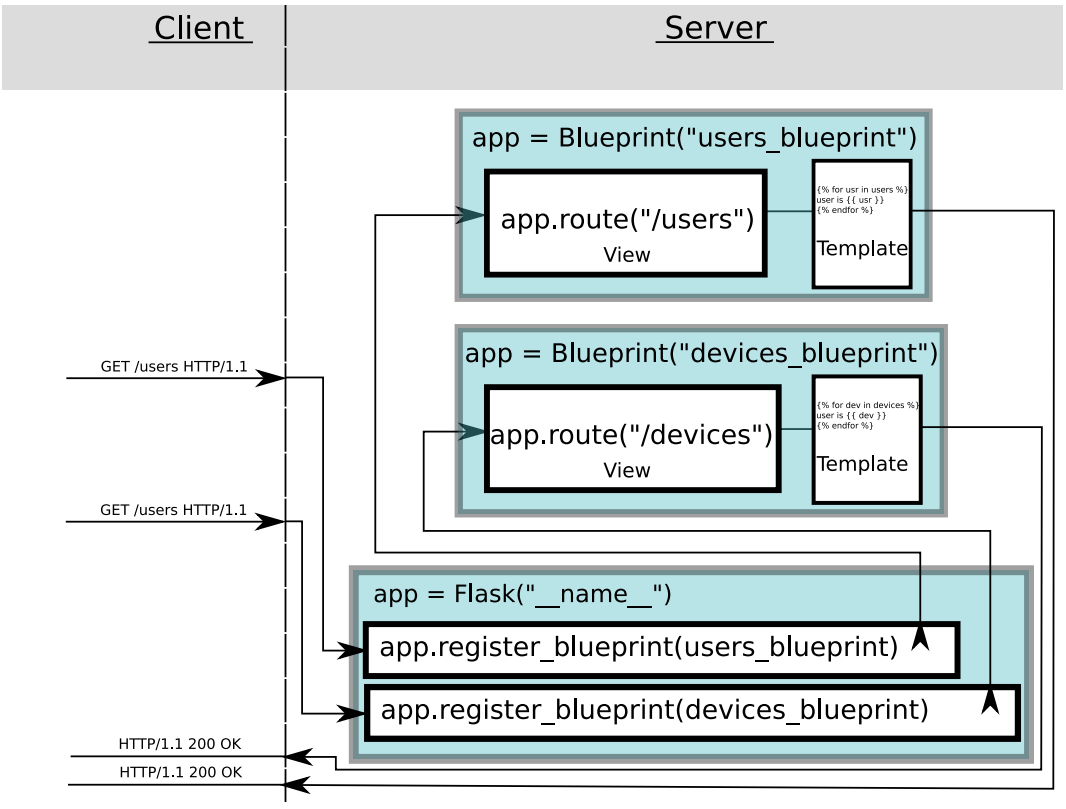
Pokud uživatel pomocí prohlížeče odešle požadavek na URL aplikace "**http://adresa:port/users/1**", tak mu aplikace vrátí informace o uživateli v tabulce jako na Tab.2.2

uzivatel	administrator
email	administrator@gmail.com

Tab. 2.2: Výsledek z pohledové funkce.

2.5.5 Blueprints

Blueprints byly vytvořeny, aby nedocházelo ve Flask aplikacích k cyklickým importům. Ty vznikají tím, že jeden Python script importuje druhý, který má v sobě import opět na první. Bez blueprintů bychom se těmto situacím nevyhnuli. Blueprint odděluje části aplikace na díly, které inicializační soubor sjednotí. V zásadě jde o to, že vytvoříme modul a v jeho inicializačním souboru `__init__.py` definujeme blueprint a za něj naimportujeme soubor obsahující pohledy. V tomto souboru však budeme pracovat s blueprintem jako by to byla celá aplikace, proto zde pomocí dekorátoru budeme routy registrovat do blueprintu. V hlavním vlákně je potřeba naimportovat blueprint a zaregistrovat jej do hlavní aplikace. Tím se zaregistrují všechny vnitřní pohledy do hlavní aplikace [13]. Jednou z velkých výhod je prefix, můžeme tedy jednotlivé aplikace oddělovat na základě prefixu před URL. Mezí další výhody patří například škálovatelnost aplikací, kdy můžeme již hotové blueprinty přidávat bez zásahu do hlavní aplikace, nebo samotná přehlednost projektu. Vyobrazeno na Obr.2.3 [17].



Obr. 2.3: Práce blueprintů.

2.5.6 Flask moduly

Flask byl navržen tak, aby byl jednoduše rozšiřitelný. Toho bylo docíleno tím, že celá Flask aplikace lze předat do rozšíření, které ji rozšíří a navrátí zpět. viz. 2.10

```
1    app = Flask(__name__)
2    app = SimpleLogin(app)
3    app = SocketIO(app)
```

Code 2.10: Rozšíření aplikace.

Flask rozšíření existuje celá řada mezi hlavní třeba **Flask-restful**, jež poskytuje snadnou implementaci REST API díky přidání Resources do pohledové vrstvy aplikace. Dalším modulem **Flask-login** implementující správu uživatelů a jejich autentizaci. Přístup k jednotlivým pohledům lze omezit pomocí přihlašovacího dekorátoru „**@login_reuired**“, jež zpřístupní pohledovou vrstvu a další za ní, pouze přihlášenému uživateli. Přihlášení uživatele se provádí za pomocí autentizační funkce „**"login_user(user)"**“, kde je prvním argumentem uživatel s ověřenými přístupovými údaji. Parametry jsou:

- **přihlašovací jméno** - identifikační prvek uživatele na základě jeho přihlašovacího jména, lze uživatele získat z modelové vrstvy aplikace,
- **heslo** - heslo je ověřovací prvek a nemělo by být nikdy uloženo v čisté podobě. Vždy je nutné na něj použít silnou hashovací funkci, protože samotný hash je nerekonstruovatelný a je zabráněno tomu, aby útočník dostal z hashe čistou podobu hesla.

Pro zajištění, aby bylo uživatelské heslo správné, je nutné jeho vstupní heslo při přihlašování opět zahashovat stejnou funkcí, která byla použita při registraci uživatele k uložení jeho hesla do modelu. Pokud tedy uživatel vyplní přihlašovací formulář a odešle ho na server tak je potřeba:

1. Dostat z požadavku přihlašovací informace.
2. Najít podle přihlašovacího jména uživatele z modelové vrstvy.
3. Pomocí hashovací funkce zahashovat heslo z požadavku.
4. Porovnat hash uživatele z modelové vrstvy a hash z požadavku.
5. Pokud se hashe shodují zavolat funkci „login_user(user)“ s parametrem ověřeného uživatele [4].

2.6 Kivy

Python framework Kivy umožňuje vytvářet grafické multiplatformní aplikace s podporou více dotyků. Tento framework byl vybrán, jako vstupní rozhraní pro raspberry pi sloužící jako zobrazovač nástěnek. Tato volba byla zvolena z důvodu podpory oficiálního 7"palcového dotykového displeje. Framework umožňuje psát aplikace dvěma způsoby: první z nich je práce a rozmístění prvků přímo v hlavní části Python kódu. To je značně nepraktické při psaní složitějších projektů. Druhou možností je rozmísťování prvků pomocí jazyka KV. Jazyk KV lze psát jak v hlavní části kódu, tak odděleně v souboru s příponou „.kv“. Hlavní kód tento soubor zpracuje a předá všechny prvky hlavnímu souboru, který je přiřadí k objektům pro možnost pozdější manipulaci s prvky [20].

2.6.1 Rozvržení

Kivy podporuje 8 druhů rozvržení, které je možné vnořovat. Jsou to tato rozvržení:

- **AnchorLayout** - v tomto rozvržení mohou být prvky umístěny,
  - pozice: nahoře, hodnota: „top“,
  - pozice: dole, hodnota: „bottom“,
  - pozice: vlevo, hodnota: „left“,
  - pozice: vpravo, hodnota: „right“,
  - pozice: uprostřed, hodnota: „center“.
- **BoxLayout** - toto rozvržení skládá prvky pod sebe, nebo vedle sebe podle nastavení zda je rozvržení vertikální či horizontální.
- **FloatLayout**: rozvržení s absolutním pozicováním vůči oknu.
- **RelativeLayout**: je pozicován vůči rodičovskému rozvržení.
- **GridLayout**: pracuje jako tabulka nastavuje se počtem sloupců a řádků.
- **PageLayout**: má více tabů je vhodný pro mobilní zařízení.
- **ScatterLayout**: pozicování stejné jako u RelativeLayout lze však otáčet zvětšovat a přemísťovat pomocí gest.
- **StackLayout**: jako BoxLayout s určitelným směrem plnění tohoto layoutu.

2.6.2 Prvky grafického rozhraní (Widgets)

Knihovna Kivy obsahuje spoustu užitečných interaktivních prvků, které lze umísťovat do rozvržení. Mezi základní patří tlačítko „Button“, textové pole „Text Input“, obrázek „Image“, nebo třeba popisek „Label“. Všechny tyto prvky se nacházejí v modulu **kivy.uix**. Každý z těchto prvků má ovšem své speciální parametry podle toho o jaký typ prvku se jedná. Mezi ty společné patří výška, šířka nebo text. Každý

z těchto prvků je naprogramován tak, aby fungoval na všech aktuálně používaných platformách včetně těch mobilních.

2.6.3 Manažér scén

Manažer scén lze naimportovat z modulu **kivy.uix**. Jeho úkolem je spravovat více obrazovek a umožnit uživateli přepínání mezi nimi. Nejdříve je potřeba nadefinovat novou obrazovku, dále je zapotřebí pomocí funkce „add_widget“ přidat všechny obrazovky do manažeru obrazovek. Manažer dokáže přepínat obrazovky s různými efekty, u kterých jde nastavit směr a dobu trvání přechodu. Mezi obrazovkami lze přepínat pomocí změny proměnné **sensormanager.current**.

2.6.4 Balíkování

Balíkování neboli packaging je možnost, jak dostat aplikaci na více platforem. Pro zabalení aplikace do APK (balík platformy android) je potřeba využít buď projektu **python-for-android**, nebo automatizačního nástroje **Buildozer**. Tento nástroj je také doporučený jako jednodušší varianta k vytvoření plnohodnotného APK balíku a k jeho fungování je potřeba mít stažené android SDK verze, pro kterou chceme aplikaci sestavit. K vytvoření aplikace pro platformu IOS je důležité mít nainstalované závislosti autoconf, automake, libtool, pkg-config a cython. Následně je potřeba stáhnout zdrojové soubory projektu **kivy-ios** a pomocí toolchain.py ve zdrojových souborech sestavit Xcode projekt.

3 Řešení Serverová část

3.1 Komunikace

Práce byla navržena tak, aby každý modul, jenž chce komunikovat se serverem o sobě při registraci řekl, co nejvíce. Z toho důvodu, aby při běžné komunikaci nedocházelo k výměně zbytečných dat. Dalším důvodem je, aby uživatel nemusel jakkoliv zasahovat do serveru. Při registraci je tedy potřeba serveru předat informaci o tom, jak mají jednotlivé měřené veličiny (quantity) vypadat. Definice je předána pomocí JSON, kde pomocí klíče „**gauge**“ můžeme definovat, o jaký typ grafu se jedná. Klíč „**gauge-style**“ definuje vzhled této veličiny v systému. Server má definované API. Pro práci s API bylo vytvořeno několik handlerů, jež jsou využity jako převodníky mezi API a různými technologiemi. Tato část je modulární a lze dopsat handler pro jakékoliv další technologie.

3.2 REST handler

Pomocí tohoto handleru se provádí registrace, nebo nastavují hodnoty u quantit. Protože REST funguje za pomocí HTTP požadavků, stává se tento kanál jednosměrný. Druhá strana totiž komunikuje pouze na základě té první, což neumožňuje periodickou stabilní komunikaci ze strany serveru ke klientovi. Z toho důvodu byla ke každému zaregistrovanému klientovi přiřazena fronta. Pokud chce server odeslat příkaz klientovi, přidá ho do jeho fronty. Při dalším požadavku klienta na server je fronta zkontrolována a příkazy odeslány ke klientovi. Využití této možnosti není dobré u spínačů, které musí reagovat okamžitě, protože by reakce spínače byla buď hodně opožděná, nebo by se mohlo více příkazů nahromadit ve frontě. Tato situace by při převzetí příkazů mohla vést k nežádoucím situacím. Pro komunikaci tohoto typu je lepší použít jiný kanál například MQTT, jenž nám zajistí obousměrnou komunikaci [14].

3.2.1 Bezpečnost

U REST je bezpečnost zajištěna pomocí JWT (JSON Web Token) tokenů. Řešení obsahuje dva tokeny "**session-key**"a "**identify-key**". O první klíč se žádá pomocí Basic Auth, kde jako uživatelské jméno použijeme jméno registrovaného uživatele a jako heslo použijeme API klíč vygenerovaný při registraci. Session-key nám umožní práci se systémem, například registraci zařízení. Co nám ale neposkytne, je práce s konkrétním zařízením, jako například mazat, nebo měnit hodnoty quantit. K tomu slouží Identify-key [7].

3.3 MQTT handler

MQTT handler slouží pro publish/subscribe hodnoty, nebo lokace quantity. MQTT při startu systému zažádá o subscribe na adrese /#, což způsobí naslouchání všech možných topiců. Pokud nějaká data přijdou na topic ve tvaru `/CZ/adresa/cislo/quantita`/value, systém najde zařízení v DB a nastaví hodnotu jeho quantity na odeslaná data. Pokud přijdou data na topic `/CZ/adresa/cislo/quantita`/location ve tvaru, [**latitude,longitude**], nebo **latitude,longitude** dojde k nastavení lokace u daného zařízení.

3.3.1 Bezpečnost

Bezpečnost u MQTT je dána samotným protokolem. Lze zde využít přihlašovacích jmen a hesel [19]. MQTT má několik možností zabezpečení kanálů používané na portech:

- **1883**: nezabezpečená verze,
- **8883**: provoz MQTT přes TLS (Transport Layer Security), proto je potřeba vygenerovat certifikáty.

3.4 HTTP handler

HTTP handler je mezičlánek mezi webovou aplikací a API. K sjednocení používá web také handler. Umožňuje to jednoduchým přepsáním API upravovat všechny možnosti komunikace najednou. K použití HTTP handleru lze používat i AJAX (Asynchronous JavaScript and XML) a tím odesílat požadavky na server i bez opětovného načtení stránky.

3.4.1 Bezpečnost

Zde je bezpečnost zajištěna pomocí knihovny Flask-login, která pomocí dekorátoru **@login-required** zabezpečí pohledovou funkci tím, že musí byt uživatel aktuálně přihlášen ve webové aplikaci. To znamená mít otevřenou session a její ID uložené v cookies.

3.5 Broadcaster

Broadcaster je modul běžící ve vlastním vlákně. Jeho hlavní úlohou je rozesílat do sítě zprávu o tom, na které adrese se server nachází. Broadcaster se spouští v inicializační části aplikace a perioda odesílaní jeho zpráv je nastavitelná v souboru

config.py. Broadcaster pro ohlašování do sítě využívá standardní knihovnu Python u nazvanou socket. Ta poskytuje jednoduché rozhraní pro komunikaci více zařízení. Broadcaster díky knihovně socket vytváří UDP (User Datagram Protocol) datagram, který je v daných intervalech odesílán na adresu 255.255.255.255. Tato adresa je poslední v sítí a značí tzv. Broadcast neboli adresu určenou k odesílání dat všem zařízením v síti.

3.6 Konfigurace

Konfigurace serveru zajišťuje více profilů spuštění. To je velice důležité proto, aby nebyly spuštěny vývojářské funkce, nebo zobrazovány detaily chybových hlášení klientům v ostrém nasazení. Spouštění těchto profilů je umožněno pomocí argumentu při spuštění serverové aplikace, nebo pomocí enviromentálních proměnných, kde nastavení proměnné „SENSORMANAGER_PROFILE“ na každém zařízení umožňuje spuštění serveru s jinou konfigurací. Priorita u načítání konfigurace je dána tak, že argument při spuštění serveru má větší prioritu než enviromentální proměnná. Pokud není definován ani argument ani enviromentální proměnná je použit profil „**production**“, tedy ten jež nespouští vývojářský režim serveru. V základu byly profily vytvořeny dva a to:

- **developement** profil je potřeba zavést přidáním názvu profilu do argumentu za spuštění serverové aplikace,
- **production** není potřeba zadávat do argumentu je zvolen jako výchozí, aby nedošlo k mylnému spuštění developement módu na produkčním serveru.

3.7 Životní cyklus serverové aplikace

Životní cyklus aplikace začíná načtením konfigurace. Pokud konfigurace obsahuje debug mode, načte se konfigurace pro debug. Následně dojde k inicializaci aplikací jako je SocketIO, Flask-Login nebo MQTT. V dalším kroku systém naimportuje veškeré doplňky v adresáři plugins a pokusí se spustit inicializační funkce jednotlivých doplňků. Pokud v doplňku nastane chyba zaznamená se a doplněk je zablokován. Po té proběhne kontrola, zda je v databázi administrátor. Pokud žádný administrátor v databázi není, zaregistruje blueprinty nezbytné pro instalaci. Pokud zde je nějaký uživatel, tak aplikace zaregistruje všechny ostatní blueprinty. Samozřejmě nám budou fungovat pouze ty pohledové funkce, které nemají dekorátor login-required. Ty pohledy, které je mají tento dekorátor přesměrují uživatele na přihlašovací stránku s GET parametrem "next". Ten určuje, na kterou stránku se uživatel pokusil dostat. Po přihlášení na ni bude přesměrován.

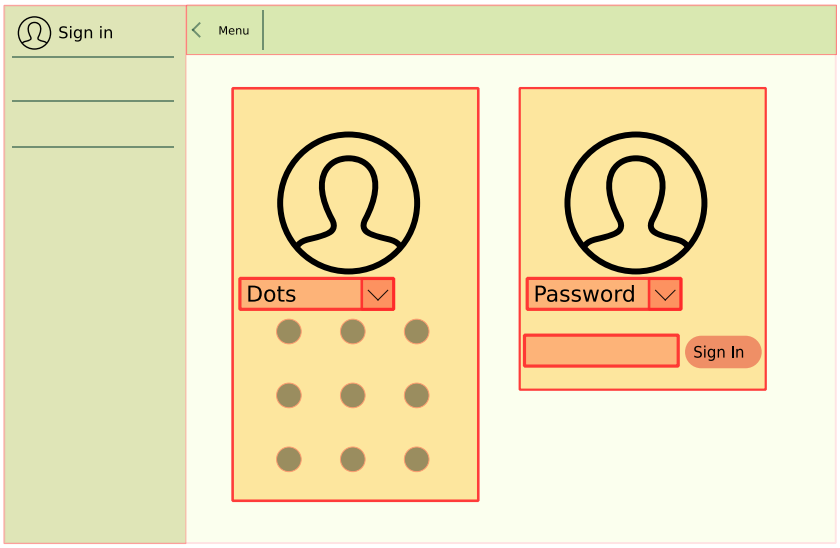
3.8 Webová aplikace

Webová aplikace je rozdělena do několika oddělených aplikací a to:

- Users (Uživatelé).
- Devices (Zařízení).
- Dashboard (Nástěnka).
- Scenes (Scény).
- Instalation (Instalace).
- Settings (Nastavení).

3.8.1 Users

Obsahuje 3 šablony jsou to **login**, **logout** a **register**. Šablona login umožňuje dva typy autentizace a to pomocí devíti stavového patternu, nebo pomocí hesla. Pattern byl zvolen z důvodu přihlašování i na malých dotykových obrazovkách, kde by zobrazení klávesnice, zabralo celou obrazovku displeje. Rozložení lze vidět na Obr.3.1.



Obr. 3.1: Druhy přihlášení v šabloně login.py.

Šablona logout zobrazuje po ukončení jeho sezení zprávu o tom, že byl odhlášen. Následně spustí v JavaScriptu odpočet, který po uplynutí přesměruje uživatele na přihlašovací obrazovku. Registrační šablona je zobrazena na základě povolení administrátora, ten má možnost ve svém nastavení vygenerovat odkaz zpřístupňující registraci uživatele v následující hodině.

3.8.2 Devices

Aplikace devices se stará o čtyři části:

- Správa zařízení.
- Skupiny zařízení.
- Mapa zařízení.
- Typy zařízení.

Správa zařízení

Správa zařízení na své hlavní stránce zobrazuje podrobný seznam registrovaných zařízení. V detailu jsou zobrazeny důležité informace jako **ID zařízení**, **typ zařízení** **topic** či fyzickou adresu zařízení MAC (Media Access Control). Je zde i informace, která zobrazuje aktuální stav zařízení a to pěti odlišnými barvami a stavovou informací.

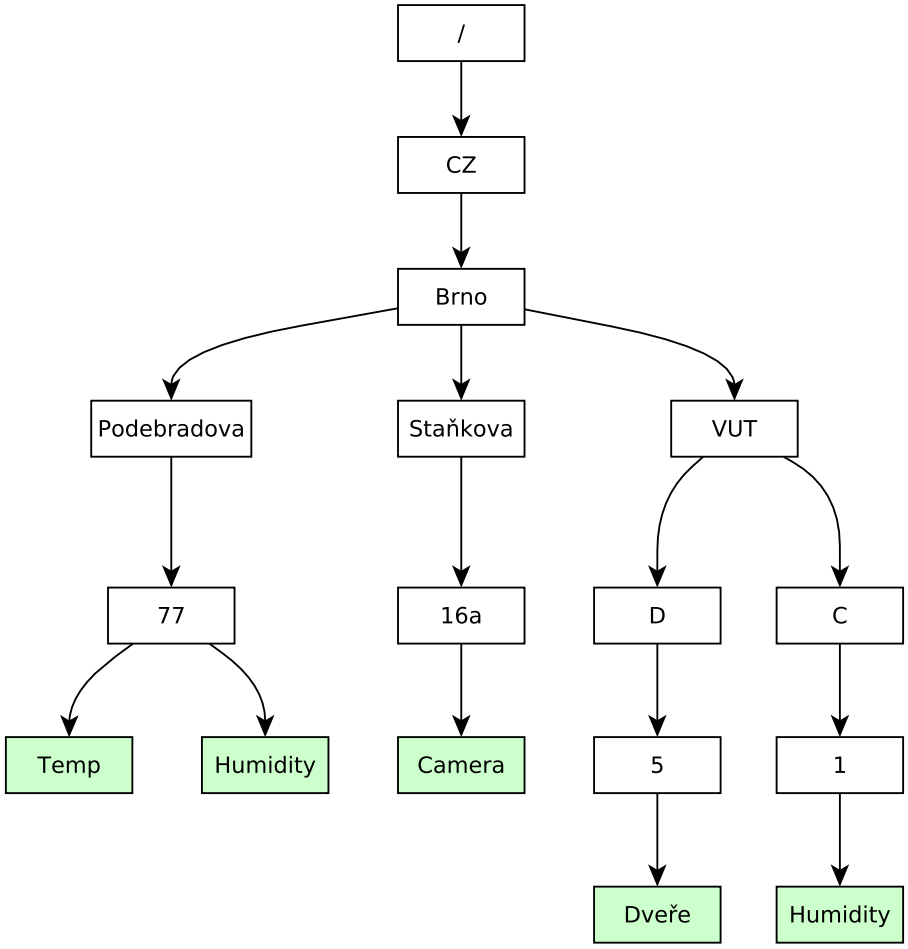
- **Zelená**, zařízení je online. Stavová informace zobrazuje odpočet pěti minut. Každá nová komunikace se zařízením vrátí odpočet na pět minut.
- **Žlutá**, zařízení nekomunikuje déle než čtyři minuty a zbývá tedy minuta než bude offline.
- **Červená**, zařízení nekomunikuje déle než pět minut a je tedy ve stavu offline.
- **Modrá**, zobrazuje virtuální zařízení registrované přes webovou aplikaci.

Z ovládacího panelu na přehledové stránce zařízení, je možnost dostat se na stránku přidání virtuálního zařízení. Tato stránka obsahuje tři pole nutné pro registraci. V prvním poli jsou tři textové prvky, očekávající od uživatele **topic**, **zeměpisnou šířku** a **zeměpisnou délku**. V druhém poli je mapa, na kterou lze zařízení umístit. Po umístění nám přepíše zeměpisnou šířku a délku na aktuální polohu z mapy. Jako třetí a zároveň nejdůležitější je JSON editor v třetím poli. Ten obsahuje veškeré zadané informace. Dále obsahuje informace o jednotlivých quantitách. Všechny tyto informace lze editovat, nebo dopsat. Zde je tedy místo, kde je potřeba definovat, jak se quantita jmenuje, v jakých jednotkách je měřená, nebo jak má vypadat její graf v systému. Protože by bylo nebezpečné dát senzoru možnost odeslat serveru HTML či JavaScriptový soubor pro definici vzhledu grafu a to kvůli zranitelnosti typu XSS (Cross-Site Scripting), kdy by útočník měl možnost vkládat do stránky nebezpečný kód.

Skupiny zařízení

Skupiny zařízení vyobrazují stromovou strukturu zařízení na základě topicu. Vše začíná kořenovým prvkem /. Pokud přijde serveru požadavek na tuto stránku, pohledová funkce vyčte z databáze všechny zařízení a postupně prochází jejich topic. Rozdělí ho pomocí regexu na jednotlivé vrstvy. Následně začne od kořene stromu

(pokud již neexistují) řetězit prvky za sebe. Protože zařízení může mít více quantit, je použit tento topic vícekrát. S tím, že je zobrazen jako poslední prvek stromu. Jak lze vidět na Obr.3.2.



Obr. 3.2: Stromová struktura zařízení.

Mapa zařízení

Mapa zařízení vyobrazuje pozici zařízení v závislosti na poloze. Ta je inicializována při registraci zařízení, ale může být kdykoliv přenastavena a to buď pomocí MQTT s topicem ve tvaru `/CZ/adresa/cislo/`**quantita**`/`**location**, nebo pomocí REST API. Pokud k přepsání polohy dojde, API se postará a oznámení této skutečnosti prohlížeči a to pomocí WebSockets, který pomocí SocketIO funkce:

`socket.on('change_location_{{ device.id}}', function (data) {}`.

Přijímá zprávu o změně lokace zařízení, které vlastní jeho ID a následně spustí anonymní funkci, která spustí funkci, což změní pozici zařízení na mapě. Na mapě je

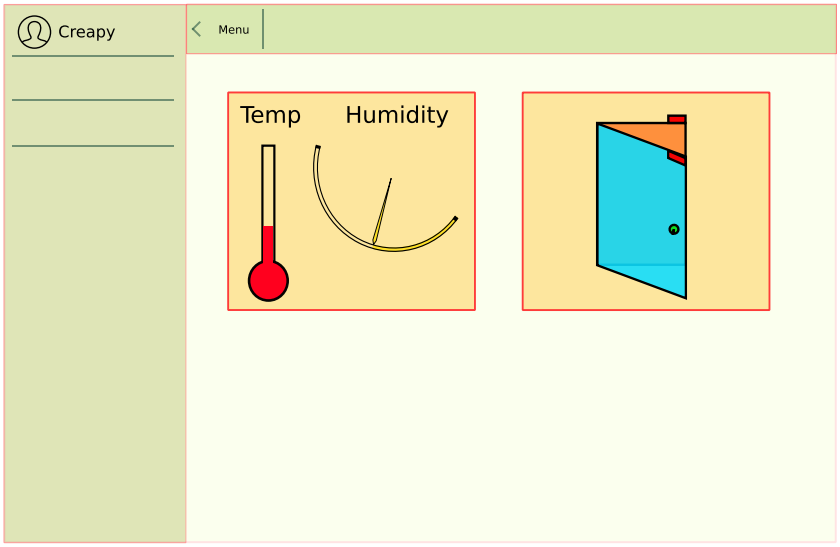
také zobrazen komunikační tok a to od zařízení směrem na server a ze serveru k aktuální pozici prohlížeče. Ten svoji lokaci dostane z location API prohlížeče. Z každého zařízení lze kliknutím zjistit jeho základní informace a také odkaz na stránku detailu zařízení.

Typy zařízení

Tato stránka zobrazuje seznam typů zařízení a to konkrétně ID, obrázek prezentující typ a popis. Typ zařízení lze přidat přes tlačítko v kontrolní oblasti. Ten stránku načte znovu s tím, že v GET parametru předá klíč "**new**"s hodnotu "**true**". Díky tomu JavaScript detekuje, že uživatel má v plánu přidat typ a je mu připraven formulář pro jeho přidání. Typ zařízení je potřeba uvést v registraci, a proto pokud typ ještě neexistuje, je potřeba jej vytvořit buď manuálně přes webový formulář, nebo přes REST API.

3.8.3 Nástěnka

Nástěnka zobrazuje připnuté qunatity s vypsaným topicem pro snadnou identifikaci. Aby tyto qunatity byly zobrazeny, je nutné je připnout v detailech zařízení. Tento web lze spustit v režimu celé obrazovky, bez menu a postranních lišt a to pomocí tlačítka „fullscreen“ v kontrolní oblasti, nebo pomocí GET parametru s klíčem "**full**"a hodnotu "**true**". Tento mód je určen pro dotyková zařízení v domácnosti, zobrazující stav. Nástěnka má vlastní topic systém, lze tak díky němu zobrazit na displeji například v kuchyni jiné veličiny než v obývacím pokoji. Stejně jako na Obr.3.3. Každá nástěnka je v databázi provázána s takzvanými piny (připnutá veličina). Připnutí veličiny je možné pomocí přepínače umístěného nad každou veličinou v detailu zařízení. Přepínač zobrazí dialog s výběrem na jaký dashboard chceme pin připnout. Odstranění probíhá podobně. Přepínač pro odstranění pinu z nástěnky nebylo umístěno přímo na nástěnce z důvodu očekávání, že na piny v oblasti nástěnky budou často prováděny mouse či touch eventy, které by mohli vest na menších zařízení k častému odepínaní pinu z aktuální nástěnky.



Obr. 3.3: Ukázka zobrazení nástěnky.

3.8.4 Scény

Scény byly vytvořeny jako řídící část, umožňující vzájemně logicky propojit prvky domácností pomocí Blockly, který je vytvořen společností Google pro interaktivní programovaní pomocí puzzle bloků. Lze tedy vytvářet scény, kde po otevření dveří odešleme SMS (Short Message Service) například o vloupání, přímo vlastníkovi objektu. Tyto scény jsou spouštěny pomocí plánovače. Jedná se o interní plánovač

procesů, který spouští scény a různé systémové funkce v daných intervalech. Při použití bloku, který je spuštěn externí akcí jako je přijmutí SMS, hovoru či přijmutí nové hodnoty z čidla, spouští plánovač Scénu pouze jednou. Tato scéna si zajistí odběr na určitém řetězci a zaregistruje zpětně volající funkce, poté je scéna ukončena [16]. Návrh takové scény je zobrazen na Obr.3.4.

návrh scény

Systém blockly se sestává z dvou druhů puzzle bloků generující kód. Jsou to bloky hodnot očekávající hodnotu ze své pravé strany a potom bloky, které umožňují vnoření více bloků. Tyto dva styly bloků stačí k tomu, aby šli v kódu vytvářet podmínky a cykly. Tyto bloky jsou předpřipravené pro spoustu programovacích jazyků včetně jazyka Python. Tyto základní bloky jsou rozděleny do kategorii umístěné pod navrhovací plochou, kde je lze pomocí myši nebo tažením prstu přesunout a skládat do sebe. Scénu lze spustit či uložit pomocí tlačítek „save“/„play“ v nástrojové liště.

pozadí

Každá scéna je záznam v databázi, který obsahuje identifikátor, název scény a její popis. Ke každé scéně je vytvořen spustitelný soubor, jehož obsahem se stává export kódu generovaného pomocí Blockly. Tento soubor může být spuštěn pomocí metody **run** definované u modelu Scene.

plánovač

Spouštění scén není závislé pouze na akci uživatele, ale mohou být spouštěny i v časových intervalech, nebo přesných časech. Byl kvůli tomu vyvinut interní plánovač procesů, který vyčítá časy spuštění ze záznamů v databázi a vyhodnocuje, kdy má danou scénu spustit. Záznam plánovače je relačně vázán ke scéně a každý takový záznam obsahuje informace o čase o opakování a o jednotce. Čas definuje, v kolik hodin se má scéna spustit, opakovaní definuje, jak dlouhá má být pauza mezi jednotlivými spuštěními. Jednotka definuje typ pauzy, například hodnoty: čas=15:00, opakování=2, jednotka=day způsobí to, že scéna bude spuštěna každý druhý den ve tři hodiny odpoledne. Výjimka nastává u scén, které je potřeba spouštět na základě přijaté hodnoty. Tyto scény musejí být spuštěny pouze jednou k registraci zpětných funkcí.

3.8.5 Budovy

Z důvodu přehledného rozmisťování více zařízení na mapu. Byla vytvořena sekce „Budovy“. Ta zajišťuje správu budov a organizaci zařízení v nich. Sekce budovy

Obr. 3.4: Provedení akce po příjetí zprávy ve službě telegram.

využívá možnosti prohlížečů pracovat s vektorovou grafikou pro navrhování jednotlivých podlaží budovy. Uživatelské rozhraní je tvořeno seznamem všech budov a detailem každé z nich. Detail zobrazuje uskupení všech podlaží nad sebou pomocí technologie WebGL (WEB Graphics Library) a zobrazuje nám veškeré informace, které systém o budově obsahuje.

Plánovač podlaží

Plánovač podlaží je hlavní částí této sekce. Umožňuje pomocí základních grafických obrazců různých barev navrhovat podlaží. Plánovač se skládá ze dvou hlavních sekcí a to panel nástrojů a samotná plocha návrhu. Ta je tvořena mřížkou pro usnadnění kreslení a posuvníkem přepínající aktuální podlaží budovy. Panel nástrojů se skládá ze tří částí: 1. část obsahuje tato tlačítka: pro uložení, export , import , přepnutí do perspektivy a zámek plochy.

- **uložení** - Uloží aktuální podlaží na server.
- **import** - Otevře dialog pro uložení aktuálního podlaží do svg souboru.
- **export** - Otevře dialog pro nahrání svg souboru do aktuálního podlaží.
- **perspektiva** - Zobrazí podlaží nad sebou z perspektivy.
- **zámek** plochy - Umožňuje či zakazuje manipulaci objektů na ploše návrhu.

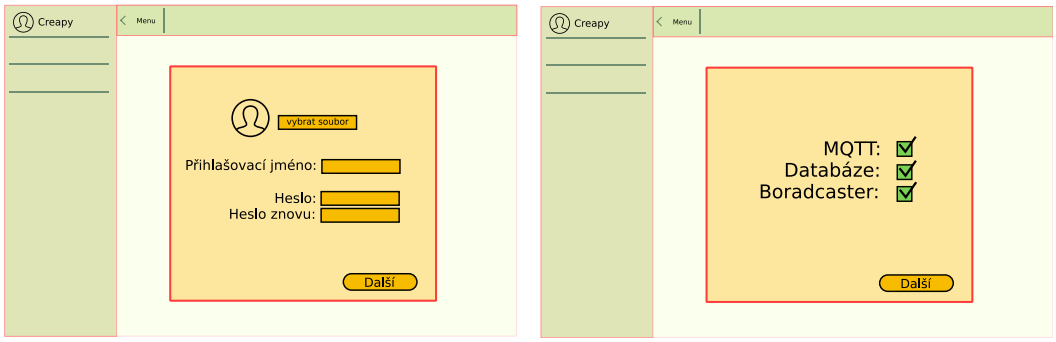
Druhou skupinou jsou tvary, které lze na navrhovací plochu umísťovat. Mezi tvary patří také položka **Text** umožňující vkládat na plochu popisky. Do těchto popisků je uživateli umožněno vkládat hodnoty měřených veličin a to pomocí textového formátu `$(/CZ/Brno/Podebradova/77/balkon/Temp/value)`. Poslední skupina obsahuje několik základních barev pro odlišení místností od sebe.

3.8.6 Installation

Instalace je inicializační proces systému. Slouží k vytvoření administrátorského účtu, kontrolu vedlejších potřebných služeb jako je připojení do databáze, nebo k MQTT

Brokeru. Pokud některá z těchto služeb nebude mít ověřené připojení, instalační proces zobrazí konfigurační formulář pro nastavení nových hodnot. Tuto konfiguraci následně uloží a pokusí se o opětovné připojení. Pokud všechny služby běží, instalace je ukončena a je vrácena výzva k restartování serveru. Pokud server restartujeme, po načtení odešle přes WebSockets zprávu:
`sio.emit('restarted',{"ok":"ok"}, broadcast=True)`,

která prohlížeč informuje o tom, že instalační proces byl úspěšný. Ten následně přesměruje prohlížeč na přihlašovací stránku. Jako na Obr.3.5.



Obr. 3.5: Ukázka instalace.

3.9 Doplňky

Architektura doplňků byla navržena tak, aby co nejuniverzálnějším způsobem dokázala doplnit funkcionalitu této práce. Každý doplněk je tvořen adresářovou strukturou skládající se ze složek **blocks**, **pages**, **gauges** a ze dvou souborů nazvaných: „info.json“ a „data.json“. Složka blocks obsahuje jednotilivé Python í komponenty reprezentující puzzle bloky použitelné v editoru scén. Příklad jednoduchého bloku je znázorněný na ukázce 3.1.

```
from server.block import Block, Function
name = "device_block"

data={
  "type": name,
  "message0": "%1",
  "args0": [
    {
      "type": "field_dropdown",
      "name": "NAME",
```

```
      "options": [
                    ["/CZ/Brno/technicka/12","skola12"],
                    ["/CZ/Brno/technicka/10","skola10"]
                  ]
    }
  ],
  "output": None, "colour": 230,
  "tooltip": "",  "helpUrl": ""
}
function = Function(name, "Python", """
  var dropdown_name = block.getFieldValue('NAME');
  var code = dropdown_name
  return [code, Blockly.Python.ORDER_NONE];
      """)
block = Block(data, function)
```

Code 3.1: Ukázka struktury puzzle bloku

Takto napsaný blok je tvořen dvěma částmi. Jedna popisuje. jak blok vypadá a jak se chová ke svému okolí. Jestli jej de napojovat z hora, nebo ze spodní části či jej zamknout k použití pouze v jiném bloku. Dle popisu bloku z příkladu lze usoudit, že se jedná o blok, který bude napojitelný zleva (vychází možnost pokud není uvedeno jinak). Blok bude obsahovat jedno vstupní pole typu „dropdown“ (rozbalovací list možností). Dále to, že název tohoto bloku je „**NAME**“ a možnosti umístěné v listu jsou „**skola12**“ a „**skola10**“, kde jejich hodnoty představují topicy zařízení. Druhá část představuje jeho výstup a chování v rámci exportovaného kódu. Z ukázky opět vyplývá, že do bloku, na který se tento blok napojuje, bude předána pouze hodnota z rozbalovacího seznamu. Každý takto napsaný blok musí být uveden v souboru „info.json“. Zde je puzzle bloku určena kategorie, do které má být zařazen. Soubor „info.json“ dále obsahuje základní informace o doplňku jako jeho název, jméno autora, nebo například textové pole položky v hlavním menu v sekci „plugins“. Adresář „pages“ obsahuje blueprint, tvořící skupinu handlerů, kde každý z nich začíná prefixem „/nazevPluginu-plugin/“. Odkaz směřující z hlavního menu vždy směřuje na adresu index. Globální handler „plugins“ směřuje na stránku zobrazující seznam všech doplňků v systému. V postranním menu jsou zobrazeny jen ty doplňky, které prošly inicializační funkcí. Pokud inicializační funkce neproběhne korektně, systém načte pouze stránku info, která by měla zobrazit, z jakého důvodu nebyl doplněk načten. Sekce „Pages“ je taky ta část doplňku, která by měla spouštět externí kód, nebo si vytvářet handlery na komunikaci s externímy či internímy

procesy. Mezi ně může patřit například komunikace přes sériový port. Poslední složkou v adresářové struktuře je složka „gagues“, ta obsahuje jednotlivé zobrazovače měřených veličin, jako například otevřené či zavřené dveře, zapnutou nebo vypnutou žárovku apod. Tyto zobrazovače jsou popisovány pomocí Javascriptu, kde lze i definovat jak bude zobrazovač reagovat na přijetí nové hodnoty systémem. Pro rychlé akce lze u zobrazovače nastavit handlery na akce typu „click“, „mousemove“ apod. a v těchto akcích odesílat do zařízení různé příkazy. Doplňky mají přístup k tzv. „Accounts-service“ objektům. Jedná se o modely v databázi, které lze použít pro ukládání autentifikačních klíčů a dalších bezpečnostních prvků do databáze k pozdějšímu využití. Této služby využívá například doplněk hudební služby Spotify, který pro přístup k API používá bezpečnostní řešení OAuth.

3.9.1 doplněk SM_plugin

Z důvodu zachovaní struktury doplňování prvků do systému výhradně pomocí doplňků, byl vytvořen doplněk „SM_plugin“. Cílem tohoto doplňku bylo dostat do systémů základní set zobrazovačů dat, jako například: žárovka, zásuvka, dveře. Všechny tyto prvky jsou si v části kódu velice podobné jedná se o následující strukturu. 3.2

```
1 from server.gauge import Gauge
2 name = "bulb"
3 content = """console.log("content of gauge")"""
4 gauge = Gauge(content=content, name=name)
```

Code 3.2: Pythoní část zobrazovače dat v doplňku.

Takto definovaný zobrazovač nekontroluje, zda aktuálně testovaný zobrazovač má definici **gauge** shodnou s názvem tohoto zobrazovače a script použitý v části „content“ se provede při každém testování doplňku. Správně definovaný zobrazovač nejdříve vždy kontroluje, zda se jedná o testovaný zobrazovač a to použitím proměnné `quantities[i]`, která obsahuje veškeré informace o zobrazovači takže nejlepší variantou jak kontrolovat zda se jedná právě o aktuální zobrazovač je použití podmínky 3.3:

```
1 if quantities[i].gauge == "bulb"{
2     let div = document.createElement("div")
3         div.id = 'bulb_'+topicOfQuantity;
4         div.style.width="200px";
5         if (quantities[i].value == "1" ||  quantities[i].value == "True"){
6             div.innerHTML="Svítící Žárovka"
7         }else{
```

```
8               div.innerHTML="zhasnutá Žárovka"
9           }
10      doucument.appendChild(div);
11  }
```

Code 3.3: Příklad content části nereagující na změny v systému.

Takový kód je sice funkční ale ukáže nám, zda žárovka svítí pouze při načtení stránky. Pokud se data v systému změní, měl by na tuto skutečnost reagovat i zobrazovač a to za použití technologie WebSockets, jejichž rozhraní využívá knihovna SocketIO, použita v tomto systému. Pro detekci změny hodnot u quantity je potřeba vytvořit handler knihovny SocketIO, naslouchající na topicu měřené veličiny. Handler lze napsat takto:3.4

```
1   socket.on('change_data_'+topicOfQuantity,
2       function (data,qtopic=topicOfQuantity) {
3           let quantity = data.quantity;
4           let bulb_div = document.getElementById('bulb_'+qtopic)
5           if (quantity.value == "1" ||  quantity.value == "True"){
6               bulb_div.innerHTML = "Svítící Žárovka"
7           }
8           else{
9               bulb_div.innerHTML = "zhasnutá Žárovka"
10          }
11  });
```

Code 3.4: SocketIO handler reagující na změnu hodnoty

Takto napsaný kód zobrazí v detailu zařízení měřenou veličinu prezentovanou nápisem: „Svítící Žárovka“ za předpokladu, že poslední naměřená hodnota veličiny je „1“, nebo „True“. Pokud nebude mít ani jednu z těchto hodnot, bude zobrazovač vracet nápis: „zhasnutá Žárovka“. Díky SocketIO handleru bude se tento nápis měnit v čase podle aktuální hodnoty bez obnovení stránky prohlížeče. Díky proměnné „**topicOfQuantit**“, která je spolu s proměnnou „**topicOfDevice**“ předána každému doplňku lze identifikovat jednotlivou veličinu a tedy zamezit změnu hodnot u jiných veličin, nebo naopak lze na základě změny hodnoty na stejném zařízení změnu provést.

Další částí tohoto doplňku je kategorie „Sensors“, která je viditelná v editoru scén. V této kategorii se nacházejí bloky vracející topic registrovaných zařízení, nebo jejich quantity. Tento doplněk neobsahuje žádné doplňující nástroje, které by byly přístupné z hlavního navigačního menu.

3.9.2 doplněk Turris Gadgets

Turris gadgets byl vyvinut spoluprací společnosti Jablotron a sdružení CZ.NIC jako speciální doplněk projektu Turris. Cílem spolupráce bylo přinést majitelům směrovače TURRIS možnost domácí automatizace spojením bezpečnostních prvků společnosti Jablotron s jejich směrovači. Komunikace s bezpečnostními prvky byla umožněna skrze zařízení nazvané jako „Turris dongle“ viz Obr3.6. komunikující pomocí

Obr. 3.6: Turris gadgets dongle.

sériové komunikace s operačním systémem směrovače tedy OpenWRT. Sériová komunikace pracuje na těchto parametrech:

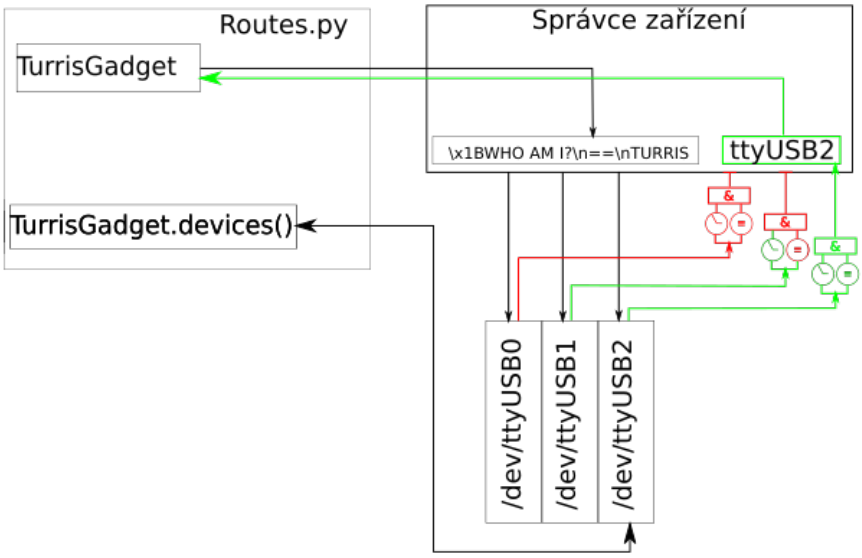
komunikační rychlost	57600 Bd
šířka slova	8 bitů
parita	Žádná
stop bit	1

Komunikace mezi modulem a směrovačem má tři režimy:

- **Příkaz** (směrovač → dongle) asynchroně.
- **Zpráva** (směrovač → dongle) asynchroně.
- **Odpověd** (dongle → směrovač) synchroně.

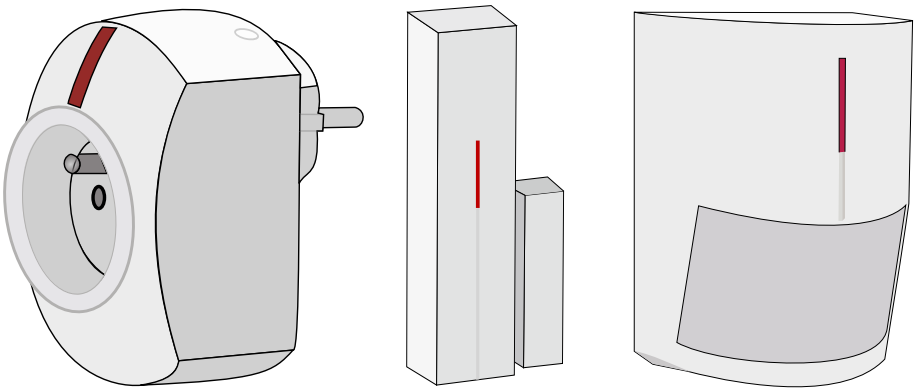
Kde příkaz musí mít formát „\x1B|**PŘÍKAZ**\n“ na něj je vždy vrácena odpověď ve tvaru „\n**ODPOVĚĎ**\n“. Asynchronní zprávy odesílá dongle v případě, že přijal zprávu z registrovaného bezdrátového senzoru. Příklad několika senzorů lze vidět na Obr.3.8. Formát přijaté zprávy se liší podle typu senzoru. Modul je opatřen pamětí s možností registrovat zde až 32 bezdrátových zařízení. Prvním problémem po zapojení Turris donglu do systému byla samotná detekce, na kterém portu se Turris dongle nachází. Proto byl vytvořen správce zařízení, který v souboru „**routes.py**“ zahájí hledání zařízení v systému a to pomocí očekávané odpovědi. Správce zařízení začne testovat každý port v systému odesláním příkazu „\x1B**WHO AM I?**\n“ a očekává od něj odpověď začínající sekvencí „\nTURRIS“. Pokud zařízení neodpoví než vyprší stanovený čas, pokračuje správce testováním dalšího portu viz Obr.3.7.

Pokud správce dostane správnou odpověď než vyprší čas, označí tento port jako používaný a odstraní jej ze seznamu testovaných zařízení. Na základě odstranění portu z testovaných se při použití více zařízení zrychlí jejich detekce. Po úspěšně



Obr. 3.7: Správce zařízení.

detekovaném zařízení si doplněk vyžádá seznam dostupných senzorů. Ten lze vypsat pomocí webové stránky, kterou doplněk poskytuje. Odkaz na tuto stránku je umístěn v navigačním menu v sekci „plugins“. Zde lze také zjistit, na kterém sériovém portu se Turris dongle nachází. Pro detekci nových zpráv ze senzorů je použito zpětného volání funkce přepsané tak, aby při přijetí zprávy byl její obsah odeslán na definovaný topic technologie MQTT. Tím je zajištěno přijetí zpráv do scénářů očekávající data ze senzorů.



Obr. 3.8: Senzory od společnosti Jablotorn.

3.9.3 doplněk GSM plugin

Doplněk GSM plugin přináší možnost odesílání a přijímání SMS, vytáčeni a detekci telefonních hovorů. Doplněk byl napsán pro modul „SIM800L“ používající sériovou komunikaci a to pomocí FTDI (Future Technology Devices International) obvodu umístěném na desce modulu. Základní konfigurace má nastaveny tyto parametry.

komunikační rychlost	9600 Bd
šířka slova	8 bitů
parita	Žádná
stop bit	1

Modul „SIM800L“ potřebuje napájení od 3,6V do 4,2V za použití základního modulu „sim800L“, je tedy potřeba snížit úroveň napětí z 5V USB (Universal Serial Bus) na napětí v rozsahu. Toho lze docílit pomocí přidání dvou usměrňovacích diod např.: „**1N5871**“, kde každá z nich způsobí úbytek 0,6V. Dohromady tedy usměrní napětí na 3,8V, což je v mezi. Problém ale nastává u proudu. Proud z USB raspberry pi není dostatečný na to, aby se modul zaregistroval do sítě a je tedy zapotřebí použití externího zdroje. V případě této práce je použito externí napájení a FTDI („FT232RL“) převodníku USB. Do modulu „SIM800L“ lze odesílat příkazy pomocí tzv. AT příkazů, na které odesílá vždy odpověď ve formátu „\nodpověď\nOK“, pokud na dotaz není vrácena žádná odpověď, je uživateli odeslána pouze odpověď OK. Modul „SIM800L“ má více druhů komunikace. Pro tento případ byl zvolen režim, kdy SMS zprávy, které přijdou na modul jsou asynchronně odeslány na hostitele. Existuje ovšem také režim, kdy jsou zprávy dostupné pouze na vyžádání. Stejně tak i volání je ihned po navázání automaticky ohlašováno hostiteli pomocí stavu „+CLIP: "420111222333",145,"",0,"",0“ pro informaci o volaném, stavu „RING“ oznamující zvonění a po ukončení hovoru oznámení „NO CARRIER OK“. Tento mód byl zvolen proto, aby systém nemusel v krátkých intervalech kontrolovat přijaté zprávy, ale pouze očekával, zda na sériový port nepřišel nový vstup. Pro základní práci s modulem stačí znát pár AT příkazu, mezi ně můžou patřit například:

Doplněk stejně jako Turris gadgets používá zpětné volání pro emitování zpráv na určitý topic technologie MQTT. Díky tomu, lze nové zprávy či příchozí hovory detekovat kdekoliv v systému pomocí zažádání odběru na tomto topicu. Samotný modul při registraci zažádá odběr a následně při odeslání příkazu na topic může odesílat SMS a vytáčet telefonní hovory. Pro inicializaci sériového portu je opět použit správce zařízení, který nalezne tento modul pomocí očekávané odpovědí „ATI\r\nSIM800“ na požadavek „ATI\n“. Modul v navigačním menu poskytuje

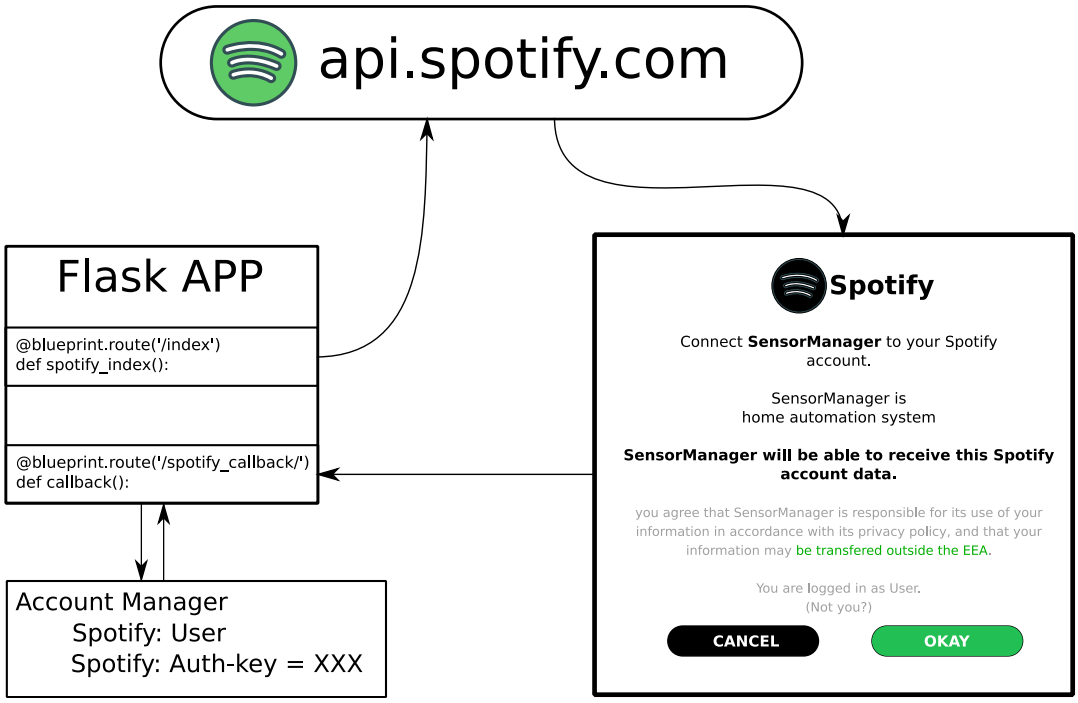
Příkaz	Odpověď
AT+CREG?	**+CREG: 0,1 OK** = domácí síť **+CREG: 0,3 OK** = roaming
AT+CSQ	**+CSQ: 25,0 OK** (číslo 25 udává sílu kvality signálu v rozmezí -110 až -54dBm)
AT+COPS?	**+COPS: 0,0,"T-Mobile CZ"OK** (Vrací název mobilního operátora)
AT+CLIP=1	**OK** (Povolí režim kdy modul asynchronně odesílá kdo na modul volá)
AT+CMGR=1	**OK** pokud je paměť prázdná pokud není, vypíšou se informace o SMS na prvním slotu v paměti a následně **OK**
ATD 111222333;	**OK** (zahájí vytáčení telefonního čísla)
AT+CMGS="111222333"	spustí režim psaní SMS po přijetí znaku CTRL+Z zprávu odešle a následně vrátí **OK**

utilitu zobrazující interaktivní mobilní telefon, na kterém lze ověřit funkčnost modulu. Telefon je vyobrazen na Obr.3.9. Do sekce editoru scén přidává doplněk sekci GSM, která obsahuje puzzle bloky pro vytáčení hovorů či jejich detekci. Stejně tak doplňuje do systému možnost odesílat a detekovat příjem SMS.

Obr. 3.9: Nástroj GSM.

3.9.4 Doplněk spotify_plugin

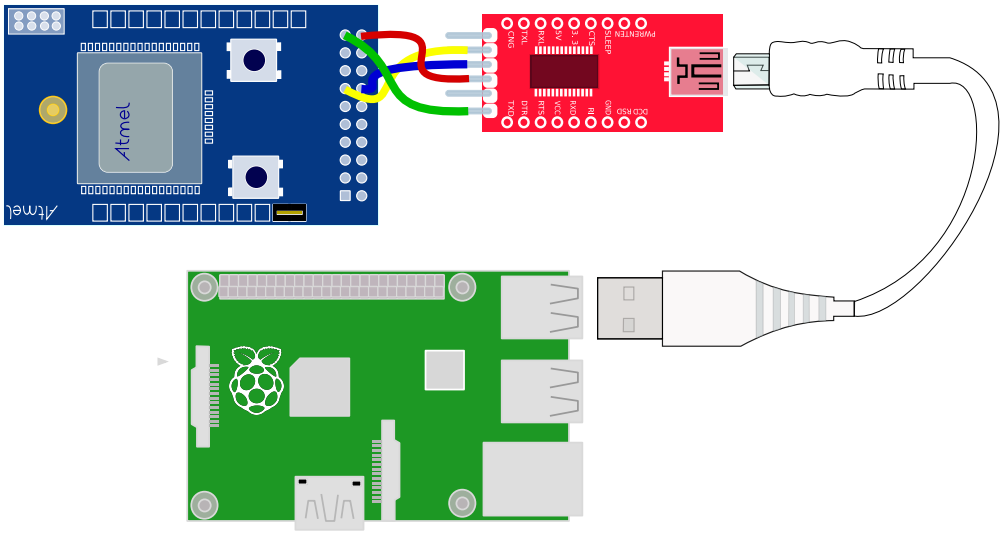
Doplněk Spotify využívá veřejného API služby Spotify k tomu, aby pomocí puzzle blocků bylo uživatelům na základě podmíněných akcí ve scénářích umožněno přepínat aktuálně přehrávaný playlist vpřed či vzad, pozastavovat čí spustit přehrávání na aktuálně přehrávaném, nebo jiném zařízení, které má spuštěnou aplikaci. Dále nabízí jednoduchý nástroj v podobě webové stránky, který umožňuje stejné funkce jako puzzle bloky a navíc zobrazuje uživateli aktuálně přehrávanou skladbu včetně obrázku alba, z kterého hudba pochází. Nástroj obsahuje vyhledávač hudby, kde u každé jednotlivé skladby je uveden identifikátor který lze uvést ve scénáři pro přehrání předem definované skladby. Doplněk využívá interního správce účtů pro uchovávání autentizačních klíčů pro každého uživatele zvlášť. pokud tento klíč vyprší je uživatel přesměrován na auntentizační stránku služby Spotify, která po přihlášení přesměruje uživatele zpět do systému. Díky tomu je dodán autentizační klíč umístěný v url přesměrování. viz Obr.3.10.



Obr. 3.10: Graf autentizace Spotify.

3.9.5 Doplněk lightweight_mesh_plugin

Lightweight mesh je velice odlehčená senzorová síť krátkého dosahu typu WPAN (Wireless Personal Area Network), obstarávající základní vrstvy komunikačního modelu a to fyzickou a přístupovou. Rychlost takové sítě je pouze několik desítek až stovek bitů za sekundu a splňuje pouze základní požadavky sítě, jako je bezpečnost, shromažďování dat či přeposílání dát dalším uzlům. Rámec používaný touto sítí je definován standardem IEEE 802.15.4 stejně jako samotná síť. Doplněk pro systém dodává možnost komunikovat s uzly sítě pomocí puzzle bloků. Stejně jako u ostatních komunikačních doplňků, využívá tento systém pro komunikaci sériový port komunikující s bránou. Pro detekci zařízení pomocí správce zařízení, byl do brány přidán příkaz, jež odpoví na dotaz „**WHO ARE YOU ?**“ odpovědí „**lightweight mesh gateway**“. Brána byla sestavena pomocí vývojového modulu „**ATmega256RFR2 ZigBit Xplained Pro Extension**“ viz Obr.3.11. Naprogramována tak, aby do sériové linky odesílala řetězec ve formátu JSON obsahující adresu příjemce a obsah zprávy. Pro odesílání je opět využito formátu JSON, kdy systém bráně odesílá adresu příjemce a text zprávy. Brána po převzetí takové zprávy ze sériové komunikace sestaví strukturu rámce a pokusí se jej odeslat. Veškerá komunikace využívá bezlicenčního pásma 868.0–868.6 MHz (pro Evropu). Senzorová síť Lightweight mesh nemá vrstvu, která by distribuovala adresy zařízením. Adresa každého zařízení je pevně daná. Síť lze rozdělit do menších podsítí pomocí PAN ID. Tato hodnota určuje, o kterou podsíť se jedná.



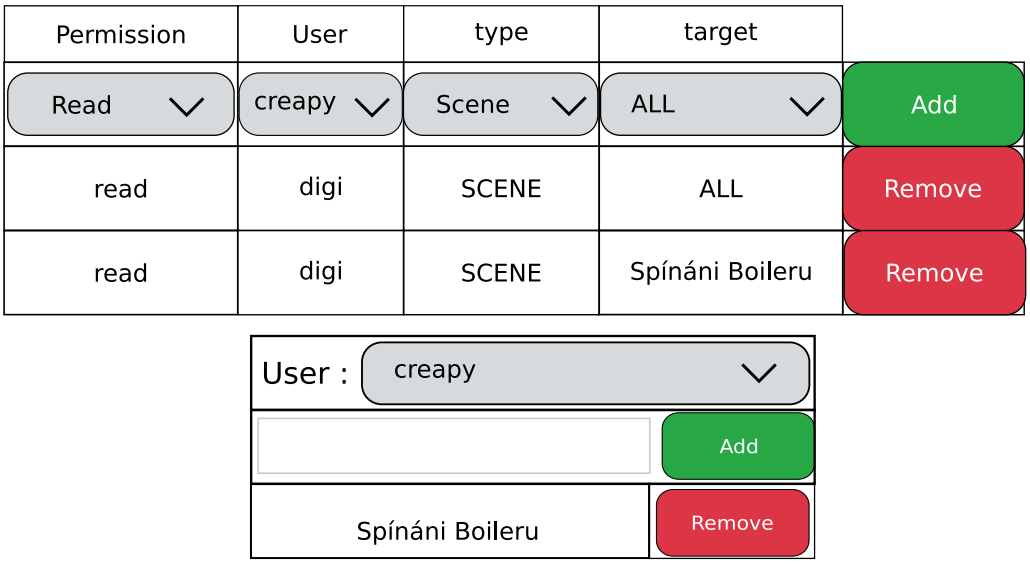
Obr. 3.11: Lightweight mesh gateway.

3.10 Settings

Nastavení dovoluje uživateli měnit své osobní informace a zároveň zobrazit API klíč, který je do prohlížeče přenesen ve formě kryptogramu a zašifrován pomocí šifrovací funkce **SHA256**. Pokud uživatel bude chtít zobrazit API klíč, klikne na tlačítko, které zobrazí dialog pro zadání svého hesla. Heslo je klíč pro rozšifrování kryptogramu. Po jeho zadání se API klíč zobrazí. Pokud je účet typu administrátor, umožňuje vygenerovat registrační odkaz s tokenem, který umožní registraci účtu na administrátorem stanovený čas. Dále obsahuje grafy, které znázorňují počty dotazů na server nebo požadavky, které nekončí stavovým kódem "200 OK". Také nabízí možnost definovat, na kterou stránku je uživatel přesměrován po přihlášení.

3.11 Oprávnění

Sekce oprávnění byla vytvořena z prostého důvodu a to omezit možnosti manipulace se systémem uživatelům, kteří by neměli zasahovat do oblastí, které jim nepřísluší. Takové omezovaní umožňuje spravovat jedním systémem více pozičně či virtuálně oddělené domácnosti. Nástroj umožňující změnu oprávnění se nachází v hlavním navigačním menu pod položkou „settings“. Aby se do této sekce uživatel dostal a bylo mu umožněno s oprávněním manipulovat, musí vlastnit speciální oprávnění „**can_edit_permissions**“ je rozdělen na dvě sekce. V první sekci lze používat oprávnění typu Read/Write, které je přiřazeno uživatelům k jednotlivým scénám, zařízením a budovám. nebo k celým kategoriím. Druha sekce obsahuje oprávnění dle určitého řetězce. Použití tohoto oprávnění je pro speciální případy, jako například samotný přístup do sekce nastavení a oprávnění. Oprávnění je možné ověřit jak na ovládací části **routes**, tak v rámci šablon a to díky přidání funkce „has_perm“ do modelu uživatele. Ta umožňuje ověřovat oprávnění kdekoliv, kde je přístup k objektu uživatele a to včetně objektu current_user, který prezentuje aktuálně přihlášeného uživatele do kontrolní i pohledové části aplikace. Model ověřuje oprávnění vrstveně. Nejdříve je zjištěno zda nemá uživatel administrátorské oprávnění (pokud ano funkce „has_perm“ pomocí návratové funkce vrátí „True“, čímž udělí uživateli oprávnění), pokud ne server se snaží zjistit zda uživatel není vlastníkem ověřovaného objektu. Pokud ani toto oprávnění uživatel nedostane tak se nakonec provede kontrola zda je v globálních záznamech oprávnění záznam o tom, že by měl uživatel práva k objektu. Pokud ani zde oprávnění není, tak funkce „has_perm“ vrátí hodnotu „False“ a zamezí, tak uživateli editovat či číst skupinu nebo konkrétní objekt.



Obr. 3.12: Rozhraní pro změnu oprávnění.

3.12 Statické soubory

Statické soubory jsou například: obrázky, JavaScriptové knihovny či soubory kaskádových stylů. Protože tyto soubory za celou dobu běhu serveru nezmění svůj obsah, není nutné je k uživateli odesílat pomocí Flask aplikace. Jsou ale chvíle, kdy je tato možnost vhodnější variantou, a proto mohou být k uživateli dopraveny dvěma způsoby. Ve vývoji na lokálním serveru bývá zpravidla použita interní pohledová funkce, která všechny statické soubory ze složky „**static**“ předá prohlížeči. Tento způsob doručování statických souborů není efektivní pro produkční server a to zejména díky tomu, že zbytečně využívá prostředky, které Flask aplikace má. Pro produkční server je výhodné použít například **nginx**, který se pomocí reverzní proxy postará o doručení statických souborů a nebude požadavky zbytečně předávat Flask aplikaci. Zároveň lze pomocí reverzní proxy šifrovat spojení ke klientovi a využít tak protokolu https na portu 443. Statické soubory v doplňcích nelze odesílat skrze reverse proxy protože doplňky jsou dynamicky inicializovány a tak je nutné tyto soubory odesílat pomocí statických funkcí do klientského prohlížeče. Jednotlivé funkce odesílající statické soubory klientům jsou vyobrazeny v kódu3.5. Tyto funkce směrující požadavky pro soubory skriptů a kaskádových stylů a jejich odpovědí je samotný soubor z cílové složky. Funkce s názvem „**get_avatar**“ vrací profilový obrázek na základě id uživatele. Funkce počítá i s tím, že by uživatel použil vektorový obrázek typu svg.

```
1 @static_blueprint.route('/js/<path:path>')
2 def send_js(path):
3     return send_from_directory(static_blueprint.static_folder+'/js', path)
```

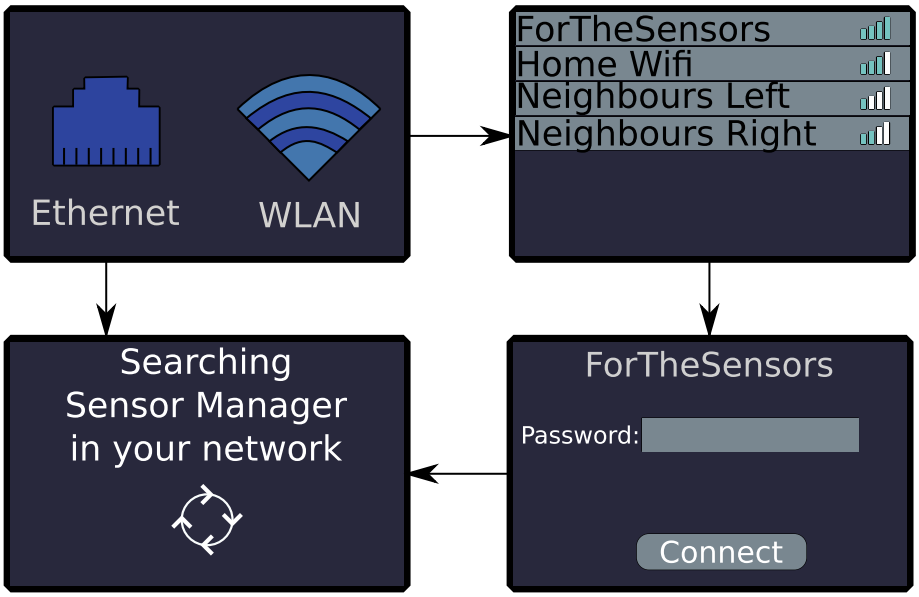
```
4
5
6  @static_blueprint.route('/css/<path:path>')
7  def send_css(path):
8      return send_from_directory(static_blueprint.static_folder+'/css', path)
9
10
11 @static_blueprint.route('/profilovky/<int:idecko>')
12 def get_avatar(idecko):
13     uzivatel = User.query.filter_by(id=idecko).first()
14     if uzivatel.avatar_ext == ".svg":
15         buffer = uzivatel.avatar
16     else:
17         buffer = BytesIO(uzivatel.avatar)
18     return send_made_file(buffer,
19                           mimetype=uzivatel.avatar_ext.replace(".", "image/"),
20                           attachment_filename='avatar.' + uzivatel.avatar_ext,
21                           as_attachment=True)
```

Code 3.5: Servírování statických souborů ve vývoji.

3.13 Zobrazovače dat

Výstupem dat ze systému může být jakékoliv zařízení reprezentující nám ať už zvuková čí obrazová data na základě nějaké akce. Takovým výstupem může být například světlo emitované žárovkou, led páskem, nebo zvuk vycházející z reproduktorů. Mezi speciální zobrazovače dat patří displeje, u kterých je potřeba v různých částech objektu a v různých časech dne zobrazovat odlišné informace. Mezi takový zobrazovač může patřit displej na nočním stolku, který v ranních hodinách nabízí možnost spuštění světla či vyjetí rolet pro vpuštění venkovního světla do místnosti. Během dne ukazuje možnosti spouštění spotřebičů, aktuální čas, nebo aktuální venkovní a vnitřní teplotu. Na večer zobrazuje možnosti zhasnutí světel a zajetí všech rolet v domě a výběr budíku na další ráno. V této práci byl jako zobrazovač použit jednodeskový počítač raspberry pi s oficiálním 7"palcovým displejem. Ten po zapnutí spustí aplikace napsanou v Kivy. Aplikace slouží k tomu aby bylo možné připojit raspberry do sítě a to bezdrátově i kabelem. Po spuštění raspberry se zobrazí dvě možnosti a to bezdrátová a drátová síť. Po výběru bezdrátové sítě se zobrazí seznam zobrazující dostupné WiFi sítě. seznam je získán pomocí nástroje **nmcli**, jež využívá jednoho z nejpoužívanějších linuxových správců sítě **Network Manager**. Výběrem bezdrátové sítě a vyplnění formuláře dotazující se na heslo pokusí se aplikace opět

pomocí nástroje **nmcli** přihlásit do sítě. Po přihlášení spustí UDP server naslouchající na portu 12345. Na tento port se Broadcaster ohlašuje do sítě. Pokud přijde na port zpráva od Broadcasteru, klient spustí prohlížeč Chromium v kiosek modu, kde jako url je použita adresu ohlašovatele.



Obr. 3.13: Zobrazovač Raspberry PI.

4 Řešení klientská část

4.1 Komunikace

Komunikace ze strany klienta může být velice rozdílná a to hlavně kvůli možnosti odlišných platforem, programovacích jazyků a přenosových médií. Problém platforem byl vyřešen díky univerzálnosti jazyka Python a to sepsáním knihovny, jež zaregistruje zařízení v systému a na základě vygenerovaného klíče komunikuje se systémem pomocí MQTT. Tím byl vyřešen problém u zařízení s chybějící veřejnou adresou. Zařízení bez veřejné adresy mohlo komunikovat pouze synchronním systémem, kdy bylo možné do zařízení odeslat data pouze na základě jeho požadavku. Tento způsob byl pomalý a nevyhovující pro tento systém, kdy v jeho prvotní fázi tvořil fronty úkolů, které odesílal do zařízení v odpovědí na jeho požadavek. To způsobovalo opožděné rychlé sekvence spuštěných příkazů. V druhé fázi byla implementována možnost komunikovat skrze MQTT a to za pomocí MQTT handleru. Komunikace pomocí jiných přenosových médii je způsobena možností dopsání vlastního doplňku komunikující se systémovými prostředky.

4.2 Registrace

Pro možnost využití systému je potřeba nejdříve zařízení zaregistrovat. Tuto registraci lze provést pomocí REST nebo MQTT handleru. Kde v registračním payloadu je potřeba přenést JSON obsahující informace o poloze zařízení, „topic“, „device_type“ určující o jaký typ zařízení se jedná (lze využít pro sepsání doplňku pracující s určitým zařízením.) a jako poslední hodnotou je pole měřených veličin „quantities“. každá taková veličina je reprezentována: hodnotou „**value**“, jménem „**name**“, jednotkou „**unit**“, proměnnou „**tograph**“, určující zda se má daná veličina zapisovat do grafu, klíč „**gauge**“ umožňující výběr zobrazovače v detailu zařízení nebo nástěnce a klič „**gauge_style**“ jako dodatečné nastavení zobrazovače. Odpovědí na takto definovaný registrační JSON je klíč, umožnující další komunikaci ze serverem. Díky tomuto klíčí je možná obnova lokace či hodnoty v tomto systému.

4.3 Obnovení hodnoty a lokace zařízení

Pro obnovení hodnoty již není potřeba odesílat na server přebytečné informace o zařízení. Jediné, co je potřeba odeslat, je JSON obsahující klíče „topic“, „value“ a „location“. Hodnotou „value“ oznamujeme systému, že chceme hodnotu na topicu

„/topic zařízení/název veličiny“ změnit. Při předání klíče location provádíme změnu lokace u zařízení protože měřené veličiny svou vlastní lokaci nemají.

4.4 Programování zařízení

4.4.1 Python a MicroPython

zařízení umožňující běh skriptů vytvořených v jazyce Python či MicroPython mají možnost komunikovat ze systémem pomocí knihovny Sensormanager. Tato knihovna usnadňuje registraci do systému a následného používání funkcí pro změnu hodnoty a lokace. Knihovna používá kombinaci REST API a MQTT. Pro registraci je použito REST API a pro následnou změnu či odběr hodnot ze systému je použito technologie MQTT. Tato knihovna byla vytvořena hlavně pro zařízení od společnosti Espressif systém, umožňující běh interpretru jazyka MicroPython nebo pro jednodeskový počítač raspberry pi.

4.4.2 Wiring

Jazyk Wiring byl vytvořen jako nadstavba nad jazykem C pro zařízení označovaná jako Arduino kompatibilní. Arduino je opensource platforma pro programování vývojových desek, používající převážně mikrokontroléry od společnosti **AVR**, využívající dostupných knihoven „**ArduinoHttpClient**“ umožňující komunikaci pomocí http protokolu a „**PubSubClient**“ pracující s technologií MQTT.

5 Závěr

Cílem diplomové práce bylo vytvořit univerzální řešení chytré domácnosti. Všechny kroky zadání byly splněny. Systém umožňuje základní požadavky chytré domácnosti a je to zejména registrace zařízení. U těchto zařízení možnost sběru dat a to pomocí síťové vrstvy s připojenými zařízeními pomocí bezdrátové technologie WiFi, například: na modulech „ESP8266“, nebo „ESP32“ od společnosti Espressive, tak pomocí proprietárních řešení, jako jsou GSM, nebo Lightweight Mesh. Tento systém lze využít i jako monitorovací či dohlížecí systém pro různé síťové prvky. Jakékoliv zařízení umožňující připojení do sítě vlastnící interpret Pythonu má možnost registrace. Například mobilní telefon má možnost pomocí aplikace Qpython či jiného interpreteru jazyka Python odesílat data ze svých senzorů do systému. Pokud zařízení nevlastní interpret jazyka Python, je možnost zařízení zaregistrovat v systému ručně a k odesílaní dat na server využívat pouze protokol MQTT. V práci byl vytvořen management uživatelů včetně definování jejich oprávnění. Díky tomu je tedy možné přihlašovaní více účastníků rodiny. Hlavním přínosem práce je možnost při registraci čidel definovat typ a vzhled zobrazovaných grafů. Díky tomu není potřeba zasahovat do serverové části aplikace při každé registraci senzoru. Zařízení, jež jsou registrovaná v tomto systému lze vzájemně propojovat pomocí scén. Ty umožňují uživatelům tvořit posloupnosti akcí, které můžou vznikat podmíněně, nebo spouštěny v určitých časech či intervalech pomocí interního plánovače. Práce dále obsahuje systém umožňující správu budov a jejich jednotlivých podlaží, které je možné navrhnout a umístit zde měřené hodnoty jednotlivých zařízení. Veškeré zařízení a budovy jsou umístěny v přehledné mapě zobrazující komunikaci čidel v reálném čase. Tato práce byla představena na soutěži studentských prací EEICT, kterou pořádá Fakulta elektrotechniky a komunikačních technologií (FEKT) VUT v Brně.

Literatura

[1] Amaran, M. H.; Noh, N. A. M.; Rohmad, M. S.; aj.: A comparison of lightweight communication protocols in robotic applications. *Procedia Computer Science*, roèník 76, 2015: s. 400–405, [cit. 05.05.2019].

[2] Decuir, J.; aj.: Bluetooth 4.0: low energy. *Cambridge, UK: Cambridge Silicon Radio SR plc*, roèník 16, 2010, [cit. 08.05.2019].

[3] Fredstam, M.; Johansson, G.: Comparing database management systems with SQLAlchemy: A quantitative study on database management systems. 2019, [cit. 08.05.2019].

[4] Grinberg, M.: *Flask web development: developing web applications with python*. "O'Reilly Media, Inc.", 2018, [cit. 08.05.2019].

[5] Harper, R.: *Inside the smart home*. Springer Science & Business Media, 2006, [cit. 01.05.2019].

[6] Johnson, A.: Extended Bluetooth Profiles on CCpilot displays. 2017, [cit. 01.05.2019].

[7] Jones, M.; Campbell, B.; Mortimore, C.: JSON Web Token (JWT) profile for OAuth 2.0 client authentication and authorization Grants. Technická zpráva, 2015, [cit. 01.05.2019].

[8] Kaplinger, T. E.; Moore, V. S.; Nusbickel, W. L.: Self-documentation for representational state transfer (REST) application programming interface (API). Kvìten 1 2018, uS Patent 9,959,363.

[9] Lampkin, V.; Leong, W. T.; Olivera, L.; aj.: *Building smarter planet solutions with mqtt and ibm websphere mq telemetry*. IBM Redbooks, 2012, [cit. 08.05.2019].

[10] Lauridsen, M.; Vejlgaard, B.; Kovacs, I. Z.; aj.: Interference measurements in the European 868 MHz ISM band with focus on LoRa and SigFox. In *Wireless Communications and Networking Conference (WCNC), 2017 IEEE*, IEEE, 2017, s. 1–6, [cit. 08.05.2019].

[11] Lee, S.; Kim, H.; Hong, D.-k.; aj.: Correlation analysis of MQTT loss and delay according to QoS level. In *Information Networking (ICOIN), 2013 International Conference on*, IEEE, 2013, s. 714–717, [cit. 01.05.2019].

[12] Liu, X.; Liu, J.; Liao, B.; aj.: Design of IoT Web Server Communication Platform based on Netty and WebSocket. In *3rd International Conference on Mechatronics Engineering and Information Technology (ICMEIT 2019)*, Atlantis Press, 2019, [cit. 05.05.2019].

[13] Maia, I.: *Building Web Applications with Flask*. Packt Publishing Ltd, 2015, [cit. 01.05.2019].

[14] Masse, M.: *REST API Design Rulebook: Designing Consistent RESTful Web Service Interfaces*. "O'Reilly Media, Inc.", 2011, [cit. 08.05.2019].

[15] Oliveira, G. M.; Costa, D. C.; Cavalcanti, R. J.; aj.: Comparison Between MQTT and WebSocket Protocols for IoT Applications Using ESP8266. In *2018 Workshop on Metrology for Industry 4.0 and IoT*, IEEE, 2018, s. 236–241, [cit. 02.05.2019].

[16] Pasternak, E.; Fenichel, R.; Marshall, A. N.: Tips for creating a block language with blockly. In *2017 IEEE Blocks and Beyond Workshop (B&B)*, IEEE, 2017, s. 21–24, [cit. 05.05.2019].

[17] Perras, J.: *Flask Blueprints*. Packt Publishing Ltd, 2015, [cit. 03.05.2019].

[18] Pilgrim, M.: *Ponořme se do Python (u) 3*. Cz. Nic, 2010, [cit. 05.05.2019].

[19] Singh, M.; Rajan, M.; Shivraj, V.; aj.: Secure mqtt for internet of things (iot). In *2015 Fifth International Conference on Communication Systems and Network Technologies*, IEEE, 2015, s. 746–751, [cit. 01.05.2019].

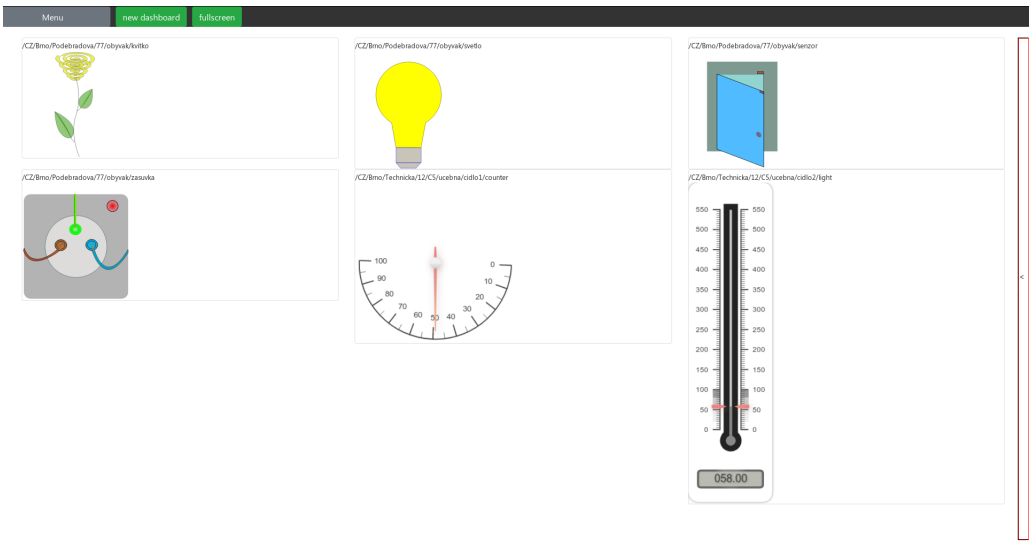
[20] Ulloa, R.: *Kivy–Interactive Applications and Games in Python*. Packt Publishing Ltd, 2015, [cit. 10.05.2019].

Seznam symbolů, veličin a zkratek

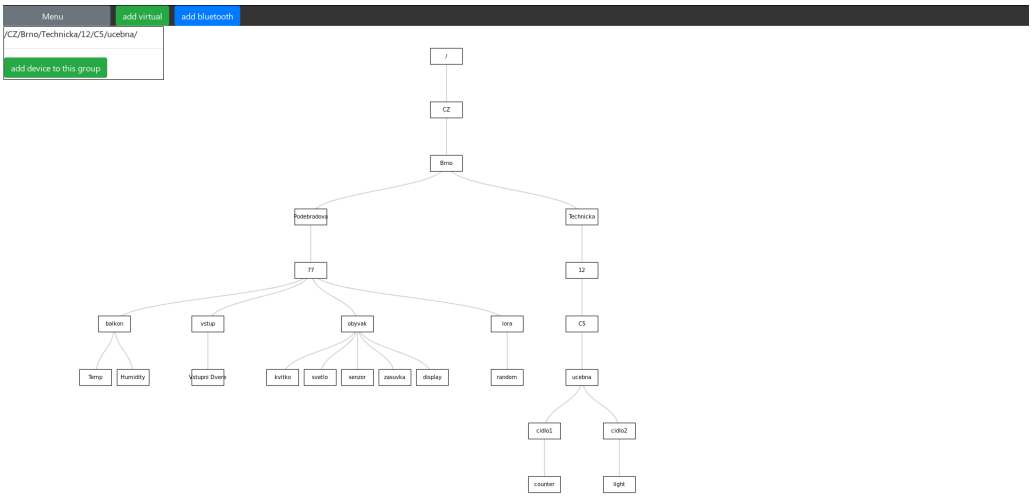
AJAX	Asynchronní JavaScript a XML – Asynchronous JavaScript and XML
API	Application programming interface
BLE	Bluetooth nízké spotřeby – Bluetooth Low Energy
CSS	Kaskádové styly – Cascading Style Sheet
FTDI	Future Technology Devices International
GTK	GIMP Toolkit
HID	Human Interface Device
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IoT	Internet věcí – Internet of Things
IPSP	Internet Protocol Support Profile
JSON	JavaScript Object Notation
JWT	JSON Web Token
MAC	Media Access Control
MHz	Mega Hertz
MQTT	Message Queuing Telemetry Transport
MVC	Model View Controller
ORM	Object-relational mapping
QoS	Kvalita služeb – Quality of Service
QT	Qt toolkit
REST	Representational State Transfer
TLS	Transport Layer Security
UDP	User Datagram Protocol
URL	Uniform Resource Locator
USB	Univerzální sériová sběrnice – Universal Serial Bus
WebGL	WEB Graphics Library
WPAN	Wireless Personal Area Network
XML	Extensible Markup Language
XSS	Cross-Site Scripting
YAML	Ain't Markup Language

Seznam příloh

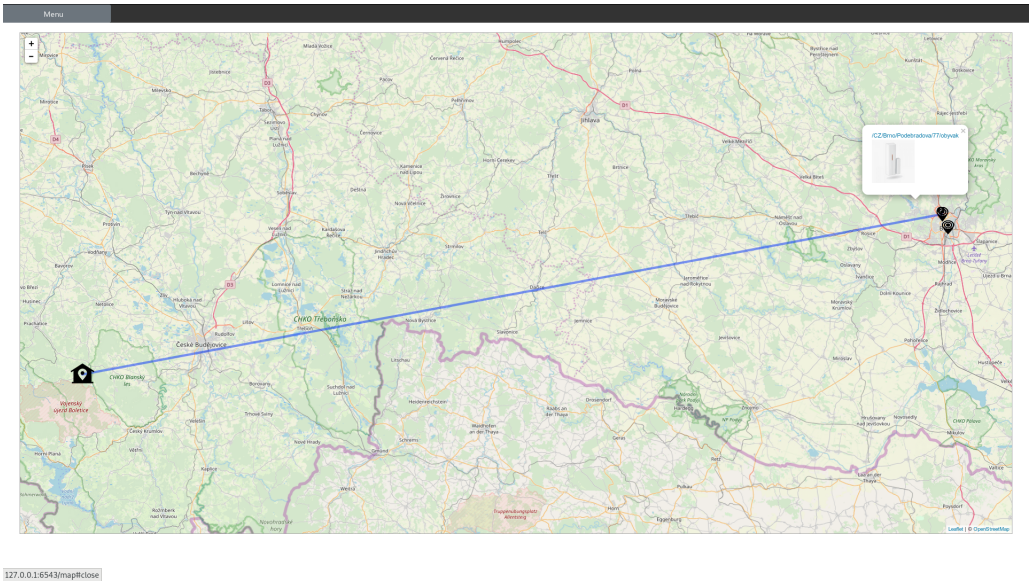
A Náhled grafického prostředí aplikace



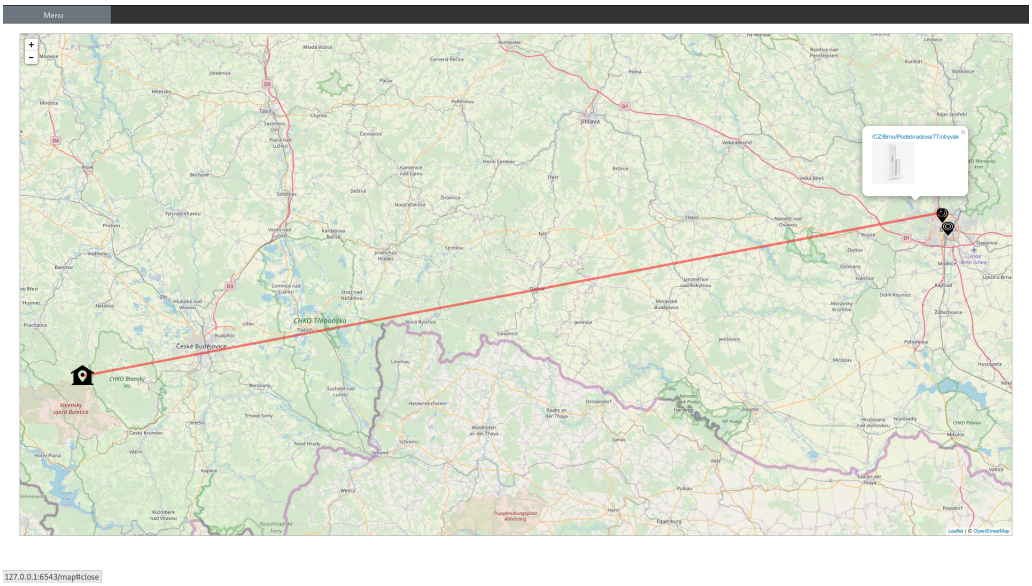
Obr. A.1: Náhled nástěnky.



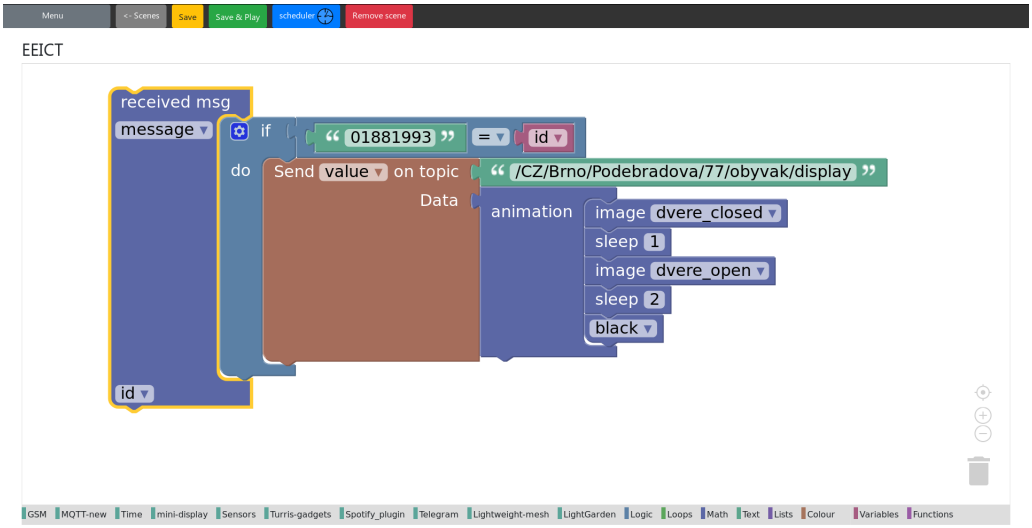
Obr. A.2: Náhled grafu zařízení.



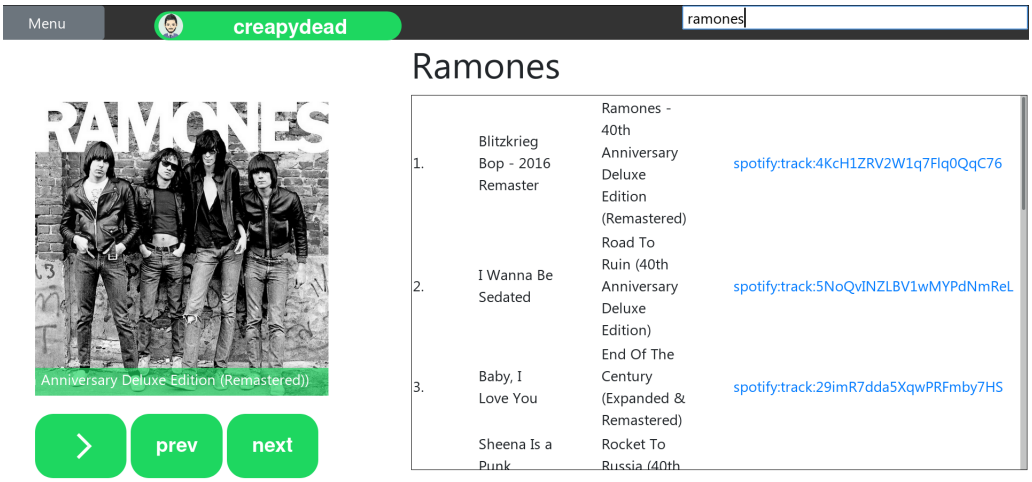
Obr. A.3: Náhled mapy komunikující směrem k serveru.



Obr. A.4: Náhled mapy komunikující směrem ke klientovi.

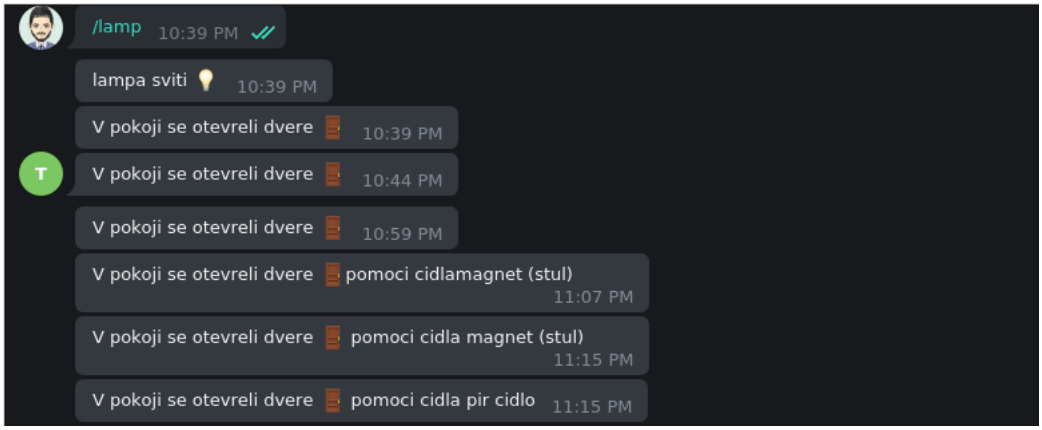


Obr. A.5: Náhled definice scény



Obr. A.6: Náhled nástroje doplňku spotify

Obr. A.7: Náhled nástroje kreslení obrázků pro displej Oled(128x64)



Obr. A.8: Náhled komunikace v aplikaci telegram

B Obsah přiloženého CD

- `/` ..... Kořenový adresář přiloženého CD
  - `Diplomova prace.pdf` ..... Diplomová práce v elektronické podobě
  - `SensorManager-Server` ..... Serverová část aplikace
    - `dockerfile` ..... Soubor umožňující sestavit docker container
    - `main.py` ..... Spouštěcí soubor serverové části systému
    - `README.md` ..... Základní informace pro spuštění serveru
    - `requirements.txt` ... Soubor závislostí pro sestavení virtuálního enviromentu
    - `server` ..... Soubory určené pro Raspbberry PI zobrazovač
      - `api` ..... Složka obsahujcí definice aplikačních rozhraní a handlerů
      - `dashboard` ..... Blueprint dahsboard
      - `devices` ..... Blueprint devices
      - `floorplan` ..... Blueprint floorplan (floorplan + buildings)
      - `instalation` ..... Blueprint obsahující instalaci
      - `map` ..... Blueprint dahsboard
      - `plugins` ..... Složka obsahující jednotlivé pluginy
      - `scenes` ..... Blueprint scén
      - `settings` ..... Blueprint nastavení
      - `static` ... Složka obsahující statické soubory (možné dodávat mimo Flask)
      - `templates` ..... Hlavní + autentizační šablony aplikace
      - `users` ..... Blueprint uživatelů
      - `__init__.py` ..... Hlavní inicializační soubor modulu
      - `acount_manager.py` ..... Správce účtů třetích stran
      - `block.py` ..... Definice puzzle blocku
      - `broadcaster.py` ..... Ohlašovač do sítě
      - `config.py` ..... Hlavní konfigurační soubor
      - `contacts.py` ..... Pole kontaktů
      - `databaze.db.` ..... Soubor databáze
      - `gauge.py` ..... Definice zobrazovače dat
      - `menu.py` ..... Struktura hlavního menu
      - `models.py` ..... Linkování databáze s třídami apliakce
      - `plugin.py` ..... Definice doplňků
      - `scheduler.py` ..... Plánovač scén
  - `SensorManager-RPi` ..... Soubory určené pro Raspbberry PI zobrazovač/Server
    - `Raspberry.zip` ..... Archiv obrazu Raspberry Pi
      - `Raspberry.img` ..... Obraz Raspberry Pi
  - `SensorManager-Client` ..... Základní soubory klienta + ukázky klientů
    - `sm.py` ..... Knihovna umožňující registraci zařízení na server
    - `boot.py` ..... Tento soubor je zaveden jako první po spouštění senzoru
    - `connecter.py` ..... Tento soubor obashuje seznam připojitelných ssid
    - `light_sensor` ..... Příklad odesílající na server hodnotu z AD převodníku
      - `main.py`
    - `oled128x64` ..... Příklad umožňující zobrazovat obrázky a texty na displeji
      - `main.py`