VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

DOMÁCÍ AUTOMATIZACE ZALOŽENÁ NA PLATFORMĚ ARDUINO/WEMOS/RPI

HOME AUTOMATION BASED ON ARDUINO/WEMOS/RPI PLATFORM

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **JAN JEŽEK**
AUTHOR

VEDOUCÍ PRÁCE **Ing. MICHAL KAPINUS**
SUPERVISOR

BRNO 2020

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačové grafiky a multimédií (UPGM) Akademický rok 2019/2020

Zadání bakalářské práce

Student: **Ježek Jan**
Program: Informační technologie
Název: **Domácí automatizace založená na platformě Arduino/WeMos/RPi**
Home Automation Based on Arduino/WeMos/RPi Platform
Kategorie: Vestavěné systémy

Zadání:

1. Seznamte se s technologiemi používanými pro domácí automatizaci a zabezpečení prostor (např. čidla pohybu, světelné/magnetické závory, LED osvětlení apod.).
2. Navrhněte sadu modulů určených pro sběr dat ze sledovaného prostoru a řízení vybraných periferií, se zaměřením na detekci a prevenci nežádoucí přítomnosti lidí. Navrhněte způsob komunikace mezi jednotlivými moduly a centrální řídící jednotkou implementovanou na platformě Raspberry Pi.
3. Navrhněte uživatelské rozhraní pro řízení celého systému a tvorbu uživatelských akcí založených na naměřených (např. pohyb v místnosti) a externích (např. obdržený email) datech.
4. S využitím existujících senzorů realizujte vybrané moduly (HW i SW), implementujte komunikaci mezi nimi a centrální řídící jednotkou.
5. Implementujte navržené uživatelské rozhraní formou webové aplikace.
6. Funkčnost celého systému či jeho vybraných částí stanovených po dohodě s vedoucím prakticky ověřte. Diskutujte dosažené výsledky a zvažte případná rozšíření či vylepšení.
7. Vytvořte stručný plakát nebo video prezentující vaši bakalářskou práci, její cíle a výsledky.

Literatura:

- Dle pokynů vedoucího.

Pro udělení zápočtu za první semestr je požadováno:

- Body 1, 2, 3 a rozpracovaný bod 4 zadání.

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

Vedoucí práce: **Kapinus Michal, Ing.**
Vedoucí ústavu: Černocký Jan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2019
Datum odevzdání: 28. května 2020
Datum schválení: 1. listopadu 2019

Abstrakt

Cílem této práce je navrhnout a implementovat systém domácí automatizace zaměřený na propojování jednotlivých zařízení v domácnosti pomocí podmínek a akcí. Například jakmile magnetická závora zaznamená otevření dveří, odešle se příkaz o rozsvícení světel v chodbě, pokud však zároveň nedojde k přečtení povolené karty RFID čtečkou, bude zapnut alarm. Problém je vyřešen pomocí serveru běžícího na mikropočítači Raspberry Pi, který s určitou periodou prochází a kontroluje veškeré podmínky, které uživatel definoval pomocí webového rozhraní a na základě jejich splnění vykonává dané akce. Vytvořené řešení poskytuje možnost, jak jednoduše propojit chytrá zařízení v domácnosti a jak je zautomatizovat. Uživatel sám musí naprogramovat jednotlivé moduly, komplexní podmínky i akce a tato práce mu přináší infrastrukturu ke komunikaci a příklady jakým způsobem tyto moduly navrhnout.

Abstract

This work aims to propose and implement a home automation system based on individual devices' connection using conditions and actions. For example, when a magnetic barrier determines that a door was opened, a command that turns on lights in the hallway is sent. However, if a card reader does not register allowed card, the alarm will be triggered. This problem is solved by a server running on Raspberry Pi, which is periodically checking all conditions that a user has defined via a web page. Based on their fulfillment, it executes predefined action. Created solution provides a way to connect smart devices in household easily and how to automate them. Users must program each module, complex conditions, and actions by themselves. This work presents an infrastructure for communication and examples on how to create such modules.

Klíčová slova

automatizace, chytrá domácnost, zabezpečení, RPi, Raspberry Pi, NodeMCU, Nette, PHP

Keywords

automatization, smart-home, security, RPi, Raspberry Pi, NodeMCU, Nette, PHP

Citace

JEŽEK, Jan. *Domácí automatizace založená na platformě Arduino/WeMos/RPi.* Brno, 2020. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Michal Kapinus

Domácí automatizace založená na platformě Arduino/WeMos/RPi

Prohlášení

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

.....
Jan Ježek
4. června 2020

Poděkování

Tímto bych rád poděkoval vedoucímu mé práce, Ing. Michalu Kapinusovi, za opakovanou pomoc i ve chvílích, kdy jsem nebyl tím nejlepším studentem. Vždy mi ochotně poradil a poskytl odbornou pomoc. Také bych chtěl poděkovat své přítelkyni za obrovskou psychickou podporu při psaní této práce.

Obsah

Kapitola 1

Úvod

Tato práce se soustředí na stále rostoucí oblast internetu věcí. Na trhu se vyskytují komerční i otevřená řešení umožňující plně automatizovat domácnost, ať už z pohledu zabezpečení (čtečky karet, senzory pohybu apod.), tak i z pohledu uživatelského komfortu (rozsvěcení světel, roztahování žaluzií apod.).

K napsání této práce mě vedl můj zájem o tuto oblast a nechuť pořizovat si drahé výrobcem dodávané produkty. Na trhu existuje spousta možností, ale žádné mnou nalezené řešení nedovoluje vzít libovolné programovatelné zařízení a kupříkladu jej pomocí WiFi ho inteligentně propojit s dalšími.

Komerční sféra dle očekávání disponuje mnoha nabídkami kompletní automatizace celé domácnosti pomocí jednoduše připojitelných zařízení na úkor otevřenosti systému (nutnost používat produkty vytvořené firmou, která systém vyrobila) a za mnohdy vysokou pořizovací cenu. Otevřená řešení poskytují prakticky neomezenou šanci k rozšiřování a přizpůsobení systému dle požadavků uživatele. Vyžaduje ovšem technickou znalost fungování internetu věcí, čas na zapojení a samotné programování. Oproti komerční sféře nabízí mnohem nižší cenu, která se pohybuje v řádech jednotek dolarů za jedno zařízení.

Cílem této práce je navrhnout a implementovat systém domácí automatizace zaměřený na propojování jednotlivých zařízení pomocí podmínek a akcí (např. magnetická závora zaznamená otevření dveří -> rozsviť světla v chodbě -> RFID čtečka do 60 sekund nepřečte povolenou kartu -> zapni alarm), který bude jednoduchý na implementaci, cenově dostupný a snadno rozšiřitelný.

V této práci jsou popsány základní prvky typicky používané pro vytvoření systému domácí automatizace. Je navržena architektura systému obsahující jednotlivá zařízení s důrazem na zabezpečení domácnosti proti nepovolaným osobám. Bylo vytvořeno webové rozhraní, přes které může uživatel podmínky a akce nastavovat.

Kapitola 2

Technologie

Následující kapitola popisuje veškeré použité technologie od mikropočítačů až po programovací jazyky. Dále je zde obecně popsán způsob zapojení potřebných senzorů a spotřebičů.

2.1 Hardware

V této sekci jsou popsány fyzické součástky (mikropočítače, senzory apod.) použité v této práci.

2.1.1 Raspberry Pi

Raspberry Pi [18] je miniaturní jednodeskový počítač (viz. obrázek 2.1). Byl vyvinut v roce 2012 nadací Raspberry Pi Foundation spolu s operačním systémem Raspbian odvozeným z jedné z nejstarších distribucí Linuxu – Debianu. Je optimalizován pro procesory použité na desce, ovšem počítač je schopen spustit prakticky jakýkoliv operační systém včetně Windows. Verze z roku 2019 disponuje čipem Broadcom BCM2711, který v sobě obsahuje čtyř-jádrový procesor Cortex-A72 64-bit taktovaný na 1,5 GHz. Tato verze má HDMI a USB 2.0 a 3.0 konektory, Ethernet, IEEE 802.11ac i Bluetooth. Vyžaduje napájecí zdroj 5V přes USB-C, GPIO, popřípadě Ethernet. Deska je pokryta 17 GPIO piny, které umožňují připojení různých senzorů a dalších zařízení (například teplotního čidla, magnetické závory, RFID čtečky, LED atd.).

Na trhu existuje spousta rozšiřujících desek, které poskytují nejen zvýšení počtu GPIO pinů či USB portů, ale také například ovládání DC motoru. GPIO (univerzální vstupní/-výstupní pin) je elektrický kontakt, který lze libovolně programovat. Je možno jej tedy nastavit jako vstup, kdy počítač čte hodnotu nebo výstup, kde počítač vysílá programátorem nastavenou hodnotu. Pro některé GPIO piny se dá nastavit virtuální pull-up rezistor (viz. bod 2.2.4), není tedy potřeba jeho zapojení mimo desku.

V této práci je použit Raspberry Pi 3+ (na obrázku 2.1) z roku 2018. Běží na něm skript pro zpracování zpráv a webový server.

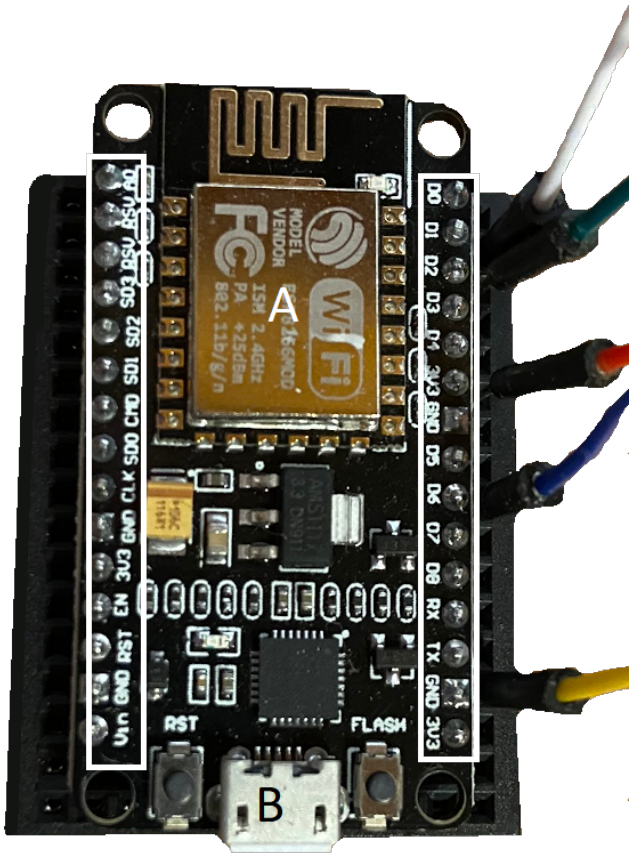
Obrázek 2.1: Srovnání velikosti Raspberry Pi (v ochranném obalu) s kartou. Nahoře se nachází GPIO piny (A). Vpravo jsou USB porty (B) a RJ45 konektor (C) pro připojení síťového kabelu. Na spodní části RPi můžeme vidět výstup na 3.5mm audio jack (D), konektor pro zapojení kamery (E), HDMI konektor (F) a napájení pomocí micro-USB (G). Samotný procesor se nachází pod písmenem H.

2.1.2 NodeMCU

NodeMCU [25] je otevřená platforma internetu věcí. Je založená na 32-bitovém WiFi mikročipu ESP8266, který je taktovaný na 80 MHz. Jeho firmware používá skriptovací jazyk Lua [22]. Pro potřeby této práce je ovšem programován v prostředí Visual Studio Code[1] s použitím jazyka C++. NodeMCU DEVKIT [4] je miniaturní vývojová deska (na obrázku 2.2) obsahující 128 kB paměti, 4 MB úložného prostoru a je programována a napájena pomocí USB (popřípadě externím napájením připojeným na některý z GPIO pinů). Tato deska má k dispozici 13 GPIO pinů. Pomocí micro-USB se připojí do počítače a dá se jednoduše programovat. Její cena se pohybuje při objednání z čínských obchodů kolem 2 amerických dolarů, v českých obchodech pak její zakoupení vyjde na zhruba 230 Kč.

[1] editor zdrojového kódu od společnosti Microsoft (https://code.visualstudio.com/), s PlatformIO (https://platformio.org/) je to jednoduchý a velice efektivní nástroj při vývoji vestavěných zařízení

V této práci je použit jako základ pro všechny moduly.

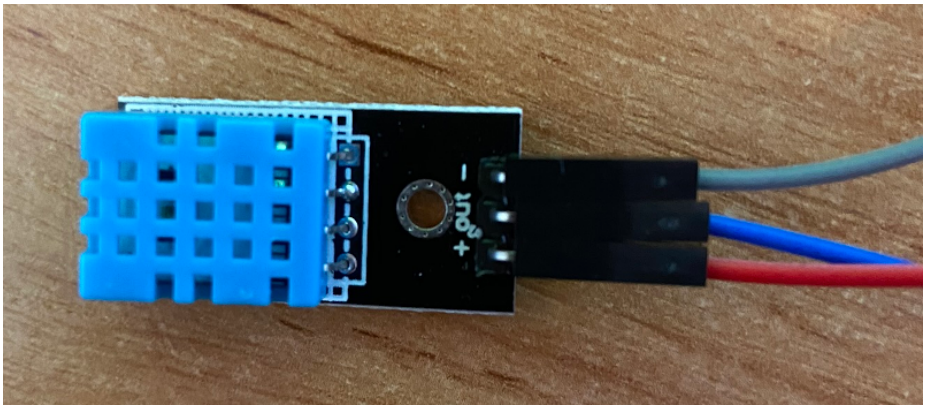


Obrázek 2.2: Zapojení vývojové desky NodeMCU na nepájivém poli. WiFi mikročip (A), micro-USB konektor pro programování a napájení (B) a po stranách GPIO piny.

2.1.3 DHT11 senzor

DHT11 [13] (na obrázku 2.3 pouze modrá část vlevo) patří do rodiny senzorů, které se používají pro měření teploty a vlhkosti. Levnější verze DHT11 se od dražší DHT22 liší přesností a rozsahem měřených hodnot. Samotná součástka má na vývodu 4 piny (zdroj napětí, zemi, analogový výstup na data a digitální výstup na data). Pro komunikaci s ním je použita knihovna *DHT sensor library*[2].

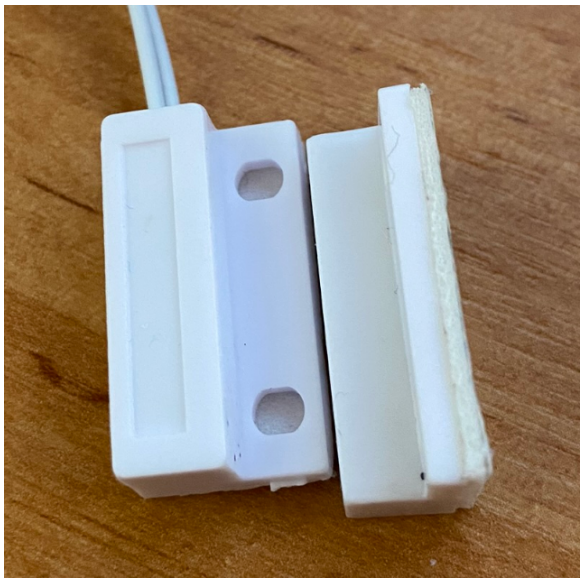
[2]https://github.com/adafruit/DHT-sensor-library



Obrázek 2.3: DHT11 senzor připájený na desku. Výstupní piny v pořadí odspodu jsou zdroj napětí, data a země.

2.1.4 Magnetická závora

Magnetická závora (na obrázku 2.4) je jednoduché zařízení o dvou částech, využívající detekce magnetického pole. Jeden ze dvou kabelů se uzemní (viz. bod 2.2.2), druhý se zapojí přes pull-up rezistor (bod 2.2.4) do mikrokontroleru. Pokud se části nedotýkají, není obvod uzavřen a na výstupu je logická jednička (výsledek pull-up rezistoru). V případě, že se části dotknou, se obvod spojí a na výstupu je logická nula.



Obrázek 2.4: Magnetická závora. Vlevo je část připojená k mikrokontroleru, vpravo je část umístěná na dveře/okno. Logická hodnota se změní, jakmile se části dostatečně přiblíží k sobě.

2.1.5 Čtečka karet

RFID (Radio Frequency Identification) [21] je technologie sloužící k přenosu a ukládání dat pomocí elektromagnetických vln. Lze ji najít v různých odvětvích průmyslu, například

při kontrole výrobních procesů, v logistice a expedici, při identifikaci zvířat, u zabezpečení předmětů proti krádeži apod.

Jedná se o bezkontaktní identifikaci, není tedy nutné, aby čtečka přišla do přímého kontaktu s identifikátorem (kartou/čipem). Identifikátor je většinou pasivní (nemá zdroj energie) a je tedy napájen z aktivní čtečky karet.

Systémy RFID pracují na různých vlnových délkách od 125 kHz (používaných při evidenci docházky, čtecí vzdálenost je do cca 20 cm) až po 5,8 GHz (mýtné brány, čtecí vzdálenost až několik desítek metrů).

Každá přístupová karta obsahuje informace. Čtečka karet potřebuje vědět, jak jsou jednotlivé informace zasílané a jak organizované. Těmto údajům říkáme formát karty. V této práci je použita RFID čtečka implementující formát Wiegand 26 (na obrázku 2.5). Wiegand 26 se stal široce využívaným standardem. Skládá se z 8 bitů, které určují nejčastěji výrobce čipu, 16 bitů pro data a 2 paritní bity[3].

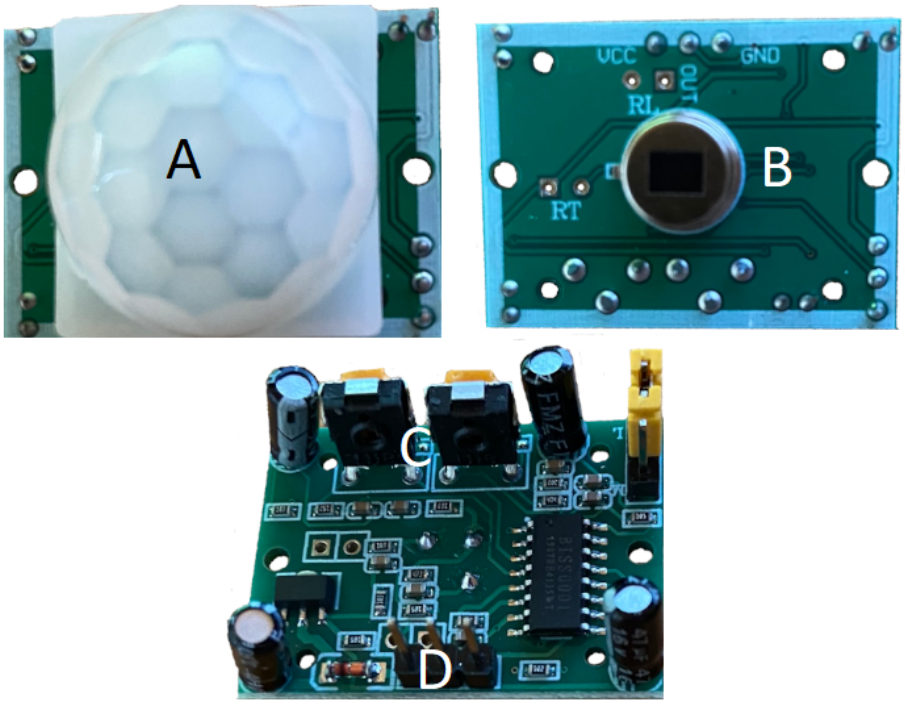
Přenos dat v protokolu Wiegand používá dva datové vodiče (*DATA Low* a *DATA High*). Pokud není na čtečce aktivita, je na oba vodiče přivedena logická jednička (bod 2.2.2). Jakmile má dojít k přenosu bitu s hodnotou jedna, nastaví se na vodiči *DATA High* logická nula, druhý vodič zůstává nezměněn (pro přenos hodnoty nula je princip obrácen).

Obrázek 2.5: RFID čtečka použitá v práci. Červený kabel je zdroj napětí, černý země (viz. bod 2.2.2). Zelené a bílé kabely jsou datové. Pomocí šedého kabelu lze ovládat zabudovaný bzučák. Význam fialového kabelu jsem ve specifikaci nenašel, nicméně na čtečce je napsáno, že se jedná o kabel *weigand 26–bit* (čtečka funguje i bez připojení tohoto kabelu).

[3]Bit přidaný k posloupnosti. Rozlišujeme sudou a lichou paritu. Sudá parita znamená, že sečteme počet jedniček v posloupnosti a paritní bit nastavíme tak, aby součet byl sudý.

2.1.6 Pohybový senzor

PIR (Passive InfraRed) – pasivní infračervený senzor (na obrázku 2.6) detekuje pohyb objektů vyzařujících infračervené záření [6]. Jedná se o účinný a levný způsob detekce pohybu a proto je hojně používán v zabezpečovací technice, při automatickém rozsvěcování světel v místnosti apod.



Obrázek 2.6: Světelný senzor společně s krytkou (A). Pod písmenem (B) je poté stejná součástka bez krytky. U písmene C jsou dva potenciometry (regulovatelné rezistory) umožňující nastavit senzitivitu a zpoždění pro detekci pohybu. Vývody (D) jsou zleva zdroj napětí, datový pin a země (viz. bod 2.2.2).

2.2 Základy zapojování obvodů

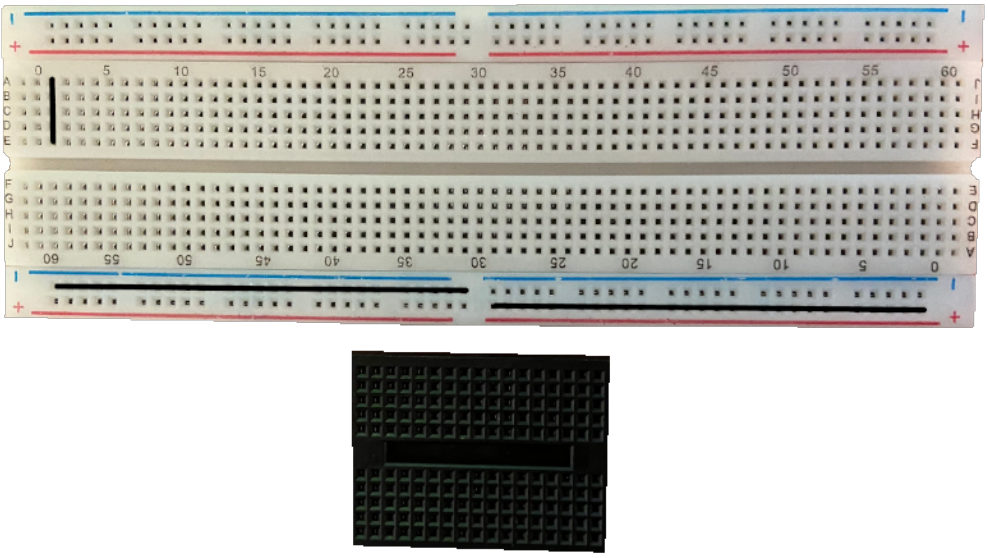
Tato část práce popisuje potřebné minimum pro zapojení využitých senzorů a spotřebičů.

2.2.1 Nepájivá pole

Nepájivé pole je ideální pro experimentování s obvody, součástky se totiž nemusí pájet a stačí je pouze propojit zastrčením kabelů do připravených otvorů.

Na obrázku 2.7 můžeme vidět velké a malé nepájivé pole. Modré a červené zdířky slouží k připojení zdroje napětí a země. Jsou spojeny po celé šíři pole, popřípadě přerušeny uprostřed a tím rozděleny na dvě části (v obrázku znázorněno černou čárou). Horní a dolní polovina jsou na sobě nezávislé. Řádky A-E a F-J jsou navzájem propojeny (znázorněno černou čárou), ale sloupce 0-60 opět vystupují samostatně.

Cena se v českých obchodech pohybuje okolo 100 Kč za větší pole a kolem 30 Kč za menší pole.



Obrázek 2.7: Nahoře velké nepájivé pole se zdířky pro zdroj napětí a zemi v horní a spodní části pole – propojeny jsou zde sloupce. V prostřední části jsou spojeny pouze řádky (stejně tak i v málem poli dole).

2.2.2 Zdroj napětí a země

Hardware použitý v této práci vydává na svých výstupech maximálně 3,3V (voltů). Při zapojování se setkáme s pojmy uzemnit, logická nula a logická jednička. Tyto výrazy se vždy vztahují k užitému hardwaru. Uzemnění je synonymem k logické nule. Znamená to, že na vstupu/výstupu mikrokontroleru je hodnota blízká 0V. Při logické jedničce je hodnota blízká 3.3V. V případě, že potřebujeme některou součástku uzemnit, připojíme ji na pin, který vydává logickou nulu.

2.2.3 Práce diodami

LED musí mít v obvodu připojený rezistor (funguje to i bez něho, ale ničíme jak LED, tak použitý hardware).

Intenzita svitu diody je závislá na proudu, který jí prochází. Z tohoto důvodu vzniká na diodě úbytek napětí. Náš hardware vydává přibližně 3,3V (bod 2.2.2). Úbytek napětí na diodě je možno vyčíst v její specifikaci. Pro náš příklad si řekněme, že to jsou 2V. Tento rozdíl napětí – 1,3V – se dle fyzikálních zákonů nemůže nikam ztratit a proto potřebujeme rezistor.

Z Ohmova zákona víme, že napětí (U) je rovno součinu odporu (R) a proudu (I). Máme tedy následující vztahy:

$$U = R * I$$
$$R = \frac{U}{I}$$
$$I = \frac{U}{R}$$

Ze specifikace diody je možno zjistit také maximální proud (I_{max}), který může diodou procházet, v našem příkladu to bude 10 mA (miliampérů). Dosazením do vzorce nám vyjde

minimální odpor 200Ω (ohmů), který musíme v obvodu použít.

$$R = \frac{2V}{0,01A} = 200\Omega$$

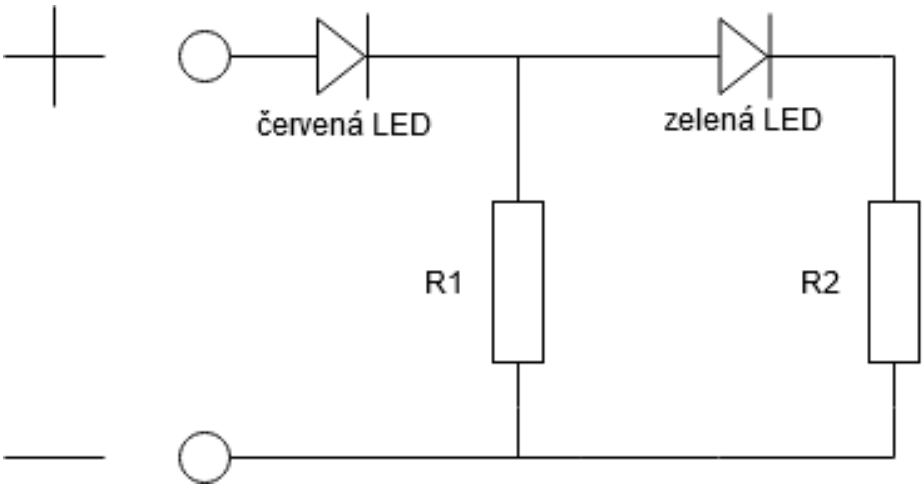
V případě, že bychom použili nedostatečný odpor (nebo nepoužili žádný), proud procházející obvodem by byl příliš velký a mohl by poškodit jak diodu, tak užitý hardware.

Tato práce využívá pro diody rezistory o velikosti odporu 1kΩ, jelikož odpory na diodách jsou pro potřeby této práce zanedbatelné a 1kΩ odpory jsem měl již k dispozici. Použití větší hodnoty než je minimální odpor pouze způsobí, že diody budou svítit méně jasněji.

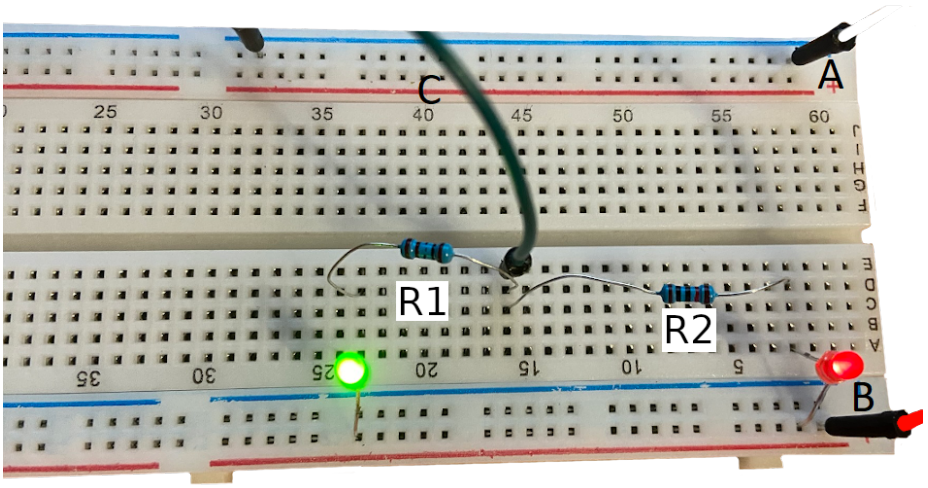
Dioda má dva konce – anodu a katodu. Pro její rozsvícení musíme uzavřít obvod, anodu (delší nožička) připojíme na zdroj napětí a katodu uzemníme (viz. obrázky 2.8 a 2.9).

Pokud chceme diodu ovládat pomocí mikrokontroleru, musíme jeden z konců diody (popřípadě oba) připojit ke GPIO pinu a hodnotu na něm nastavovat pomocí kódu. V této práci jsou diody napevno připojeny na zdroj napětí (logická jednička) a katoda je přes rezistor připojená k mikrokontroleru (pokud chci diodu zapnout, přivedu na příslušný pin logickou nulu, v opačném případě logickou jedničku).

Jas diody můžeme regulovat snížením/zvýšením odporu či snížením/zvýšením napětí. Nejčastějším způsobem regulace svitu je pulzně šířková modulace (převzato z [5]). Signál (logická jednička nebo logická nula) je zakódován v určitém intervalu jako poměr mezi stavy zapnuto/vypnuto. Tomuto poměru se říká střída. Cyklus, kdy dojde k přenosu jedné střídy, se nazývá perioda. LED má dva stavy: zapnuto/vypnuto. V případě, že je LED trvale zapnutá, je střída 100:0, což znamená, že stav zapnuto trvá 100 časových jednotek a stav vypnuto trvá 0 časových jednotek. V opačném případě 0:100 stav zapnuto trvá 0 časových jednotek a stav vypnuto 100 časových jednotek. Změnou střídy na 75:25 je dosaženo stavu, kdy LED svítí pouze na 75%.



Obrázek 2.8: Diagram zapojení dvou diod viz. obrázek 2.9



Obrázek 2.9: Zapojení dvou diod v nepájivém poli. Kabel A (vpravo nahoře) je připojen na pin GND (země) v mikrokontroleru. Kabel B (vpravo dole) je připojen na pin 3V3 (3.3V – zdroj napětí). Kabel C je pouze propojen s kabelem A. Odpor připojený k zelené diodě je menší než odpor červené diody a proto zelená svítí jasněji.

2.2.4 Pull-up a pull-down rezistor

Připojíme-li na GPIO pin mikrokontroleru tlačítko bez použití pull-up rezistoru, bude pin ve stavu vysoké impedance (logická hodnota na pinu není definovaná a byla by ovlivňována elektromagnetickým šumem z okolí). Chceme-li mít na pinu pro nezmáčknuté tlačítko logickou jedničku, musíme tlačítko přes rezistor připojit na zdroj napětí (viz. obrázek 2.10) a vzniká nám pull-up rezistor. Pokud chceme mít v klidovém stavu logickou nulu, tlačítko přes rezistor uzemníme (pull-down rezistor).

2.3 Použité jazyky

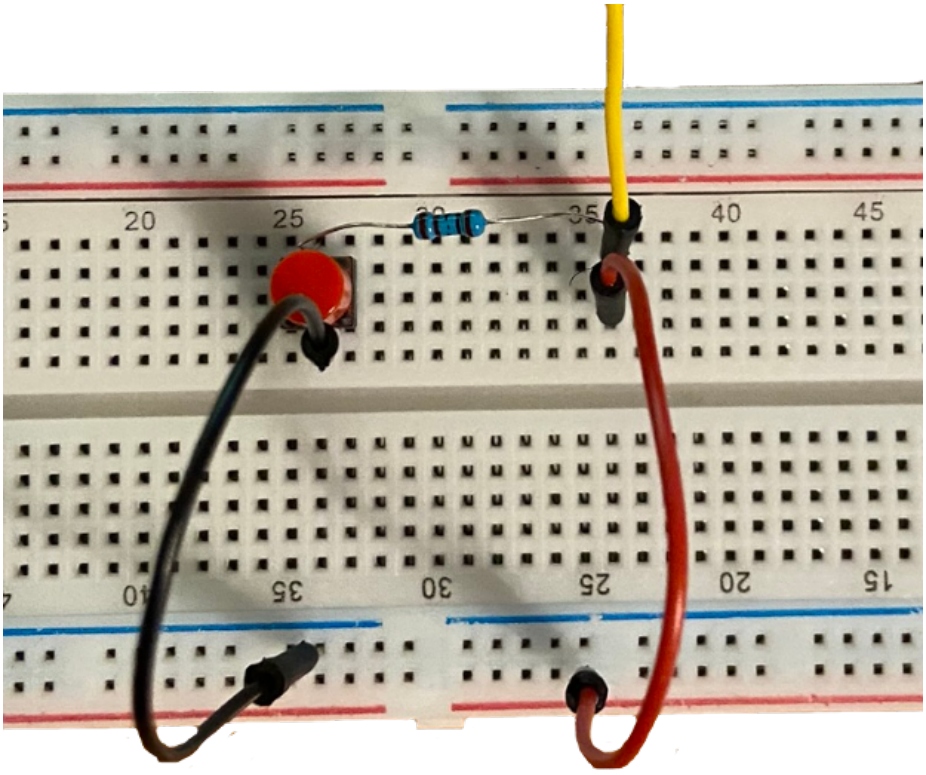
2.3.1 Python

Python [17] je vysokoúrovňový skriptovací programovací jazyk z roku 1991 a od roku 2018 se řadí mezi ty nejoblíbenější. Poslední řada (verze 3) byla vydána v roce 2008 a v současné době se jedná o jedinou aktivně podporovanou řadu. Python umožňuje tvorbu rozsáhlých aplikací (včetně grafického rozhraní). Nabízí využití nejen objektově orientovaného programování, ale také procedurálního a v omezené míře i funkcionálního. Díky jeho velice jednoduché syntaxi je velmi častou první volbou začínajících programátorů.

Základní vlastnosti jazyka Python:

- odsazení pomocí bílého místa (nepoužívá složené závorky)
- řádky nejsou ukončeny středníkem
- dynamické typování (programátor nepíše typy proměnných)
- nepodporuje konstrukci *switch-case*

V této práci je použit pro implementaci RPi serveru.



Obrázek 2.10: Zapojení tlačítka s využitím pull-up rezistoru. Tlačítko je pomocí černého kabelu uzemněno. Přes rezistor je poté červeným kabelem přivedeno ke zdroji napětí. Ve stejném sloupci je připojen žlutý kabel, který při neuzavřeném obvodu čte logickou jedničku a při zmáčknutém tlačítku logickou nulu.

Ve výpisu 2.1 můžeme vidět část skriptu běžícího na RPi. Je zde ukázáno volání funkcí, způsob zápisu cyklů a podmínek, práce s proměnnými a odsazení.

```
while True:
    if initialized:
        print()
        print('Server is up and running')

        resetDatabaseNewThread()
        conditions = database.getAllConditions()

        while len(conditions) > 0:
            condition = conditions[0]

            evaluateResult(0, conditions)

        for dataID in processedData:
            database.setProcessed(dataID)
```

```
        del processedData
        processedData = []

        time.sleep(config.checkPeriod)
```

Výpis 2.1: Část skriptu běžícího na RPi.

2.3.2 PHP

PHP [19] je skriptovací jazyk primárně vytvořený pro automatizaci jednoduchých úkonů, kde počet řádků kódu je v řádu stovek a pro programování webových aplikací a dynamických internetových stránek s použitím značkovacího jazyka HTML. Při použití PHP pro webové stránky jsou skripty prováděny na straně serveru, k uživateli je přenášen pouze výsledek (většinou nová stránka). PHP je platformově nezávislý jazyk. Podporuje mnoho knihoven pro různé účely (práce se soubory, databázovými systémy, internetové protokoly apod.). Patří mezi nejrozšířenější skriptovací jazyky určené pro tvorbu webových stránek.

Produkty jako Wikipedia, Wordpress a Facebook jsou postavené přímo na PHP, popřípadě z něho vychází.

Výpis 2.2 obsahuje ukázku kódu webového serveru. Můžeme zde vidět funkci přijímající jeden parametr, volání *$this* jako referenci na třídu, ve které se nacházíme, způsob zápisu podmínek a práci s výjimkami.

Základní vlastnosti jazyka PHP:

- proměnné jsou značeny znakem dolar *$*
- používá složené závorky pro oddělení kontextu
- řádky jsou ukončeny středníkem
- dynamické typování
- textové literály a názvy proměnných rozeznávají malá a velká písmena

V této práci je použit jako základ pro webový server, přes který uživatel spravuje podmínky a akce.

```
public function actionEdit($ID)
{
    $handler = $this->handlerModel->getByID($ID);

    if (!$handler)
    {
        throw new Nette\Application\BadRequestException();
    }

    $this['handlerForm']->setDefaults($handler->toArray());
}
```

Výpis 2.2: Příklad kódu v jazyce PHP.

2.3.3 JSON

JSON [7] je obecný serializovatelný[4] způsob zápisu dat nezávislý na počítačové platformě určený pro přenos dat. Vstupem může být libovolná datová struktura (objekt, řetězec, číslo). Výstupem je vždy řetězec. JSON se vyznačuje hlavně jednoduchostí a srozumitelností pro člověka.

```
{"klíč":"hodnota"}
```

2.3.4 HTML

HTML [26] je značkovací jazyk používaný pro tvorbu webových stránek. Je charakterizován množinou značek a vlastností. Názvy značek a jejich vlastnosti se uzavírají mezi < a >. Značky jsou až na výjimky párové, ale existují i nepárové. V ideálním případě definuje pouze uspořádání stránky, ale má i konstrukce pro tvorbu vzhledu.

Vznikl v roce 1990 společně s protokolem HTTP[5]. Jeho autor Tim Berners-Lee [9] ho vytvořil z nutnosti existence jednoduššího způsobu pro tvorbu dokumentů, zároveň také představil první webový prohlížeč. Poslední verze jazyka 5.3 byla vydána v roce 2017.

Na začátku dokumentu (viz. výpis 2.3) je vždy značka *<!DOCTYPE html>* značící, že se jedná o HTML dokument. Následuje značka *html* označující celý dokument. V ní jsou dvě hlavní značky *head* obsahující hlavičku dokumentu (ke vložení kaskádových stylů (bod 2.3.5), názvu dokumentu apod.) a samotné tělo dokumentu *body*.

```
<!DOCTYPE html>
<html>
    <head>
        <!-- hlavicka stranky -->
        <title>Ukazka HTML</title>
        <link rel="stylesheet" href="nas_kaskadovy_styl.css">
    </head>
    <body>
        <!-- telo stranky -->
        <p>Text odstavce</p>
    </body>
</html>
```

Výpis 2.3: Příklad dokumentu v jazyce HTML.

2.3.5 CSS

CSS [27] je jazyk pro popis způsobů zobrazení elementů v jazycích HTML, XHTML či XML. Vznikl v roce 1996 a definuje selektory, které určují u které značky budeme ovlivňovat její vzhled. Existují tři způsoby, jak CSS vložit do webových stránek:

- přímo do značky jako vlastnost “style”
- na začátek souboru do párové značky “style”
- vložením CSS do externího souboru a propojením do HTML

[4] převedení datové struktury na posloupnost bitů, která se dá přenést beze ztráty po síti
[5] protokol (předpis, pomocí kterého probíhá komunikace) pro přenos hypertextových dokumentů

CSS poskytuje rozsáhlé možnosti formátování a v případě externího souboru také jednoduchost údržby (např. barvu všech odkazů na stránce změníme pouze na jednom místě).

```
<p style="color:red;">Text odstavce je červený</>
```

2.4 Ostatní

2.4.1 Protokol MQTT

MQTT [24] je standardizovaný výkonnostně nenáročný protokol pro vyměňování zpráv mezi zařízeními, založený na principu publish a subscribe, fungující zpravidla na aplikační vrstvě modelu TCP/IP. Definuje zprostředkovatele zpráv a několik klientů. Zprostředkovatel je server, který přijímá zprávy od klientů (odesílatelů) a rozesílá je klientům, kteří se přihlásili k odebírání zpráv z určitého kanálu (příjemcům). Příjemce i odesílatel o sobě nemusí mít žádné informace. MQTT protokol odesílá data v prostém textovém formátu, ovšem může použít TLS pro vytvoření spojení s uživatelským jménem a heslem. Každé spojení může definovat různou kvalitu připojení:

- doručení pouze jedné zprávy
- odesílání dokud nepřijde potvrzení o přijetí
- odeslání pouze jedné zprávy

MQTT je v této práci páteří komunikace mezi serverem a moduly.

Na obrázku 2.11 můžeme vidět připojení *Client A* (v našem případě RPi – bod 2.1.1 k *Broker* (zprostředkovatel zpráv, který běží na RPi). *Client B* (v našem případě NodeMCU – bod 2.1.2) zasílá zprávu se značkou *retain* (v této práci jí není využito), která způsobí, že zpráva na zprostředkovateli zůstane. Ihned poté, co se *Client A* připojí k odebírání zpráv z kanálu, zprostředkovatel přepošle všechny zprávy se značkou *retain*. *Client A* odesílá zprávu, ale jelikož nikdo není připojen k odběru z kanálu *temperature/floor*, zpráva je zahozena. *Client B* znovu odesílá zprávu, která je ihned doručena všem připojeným klientům (v tomto případě pouze *Client A* a jelikož zpráva nemá značku *retain*, je ze zprostředkovatele odstraněna).

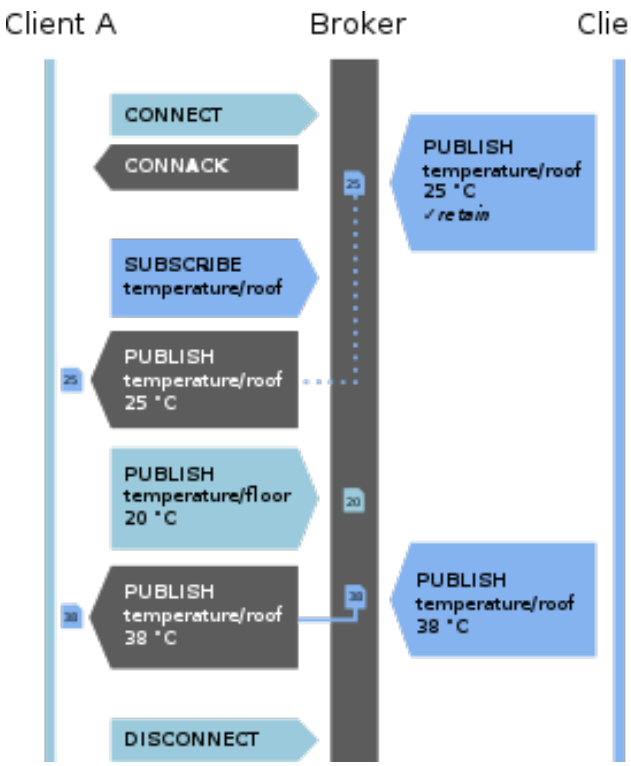
2.4.2 Nette Framework

Nette [15] je open source framework[6] pro tvorbu webových aplikací v PHP. Využívá událostmi řízené programování a z velké části je založen na použití komponent. Jeho součástí je ladící nástroj *Tracy* [16], který zachytává chyby vzniklé během chodu programu a díky tomu se jednodušeji debuguje. Nette nabízí plnou moc nad vzhledem formulářů, využívá jednoduché rozhraní pro přístup k databázi a samo generuje výsledný SQL kód.

Využívá návrhový vzor MVP (Model-View-Presenter) [8], který rozděluje aplikaci do třech hlavní částí (obrázek 2.12):

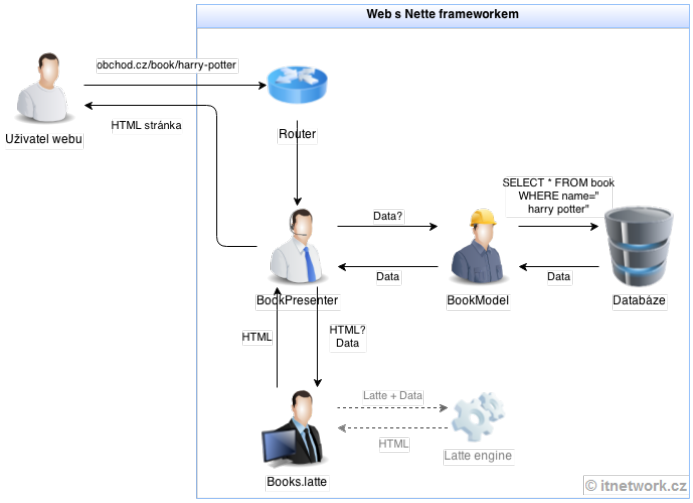
- modelů (Model) implementujících komunikaci z databází
- pohledů (View) obsahujících dokumenty napsané v šablonovacím systému Latte [14] (využívá HTML – bod 2.3.4)

[6] softwarová struktura, která slouží jako podpora při programování a vývoji



Obrázek 2.11: Komunikace pomocí MQTT protokolu (převzato z https://wikipedia.org/wiki/MQTT)

- prezenterů (Presenter), které řídí veškerou logiku aplikace (načítají data z modelů, zpracovávají je a posílají do pohledů)



Obrázek 2.12: Princip fungování architektury MVP (převzato z https://www.itnetwork.cz/php/nette/zaklady/uvod-do-php-frameworku-nette/)

2.4.3 MySQL

MySQL [11] je systém řízení báze dat[7] vlastněný společností Oracle Corporation. První vydání pochází z roku 1995. K dispozici je jako svobodný a volně šiřitelný software.

Používají ho známé společnosti v produktech jako Netflix, GitHub a YouTube.

Je multiplatformní[8], vyvinut v jazycích C a C++ a ovládá se pomocí SQL[9]. Ve výpisu 2.4 lze vidět vytvoření databázové tabulky pomocí SQL a vložení a vyčtení dat z této tabulky. Použití *%s* místo přímého vpisování hodnot do SQL příkazu nám pomůžeme lépe se bránit proti SQL injection (technika útoku pomocí SQL). Pokud bychom měli příkaz viz. poslední řádek výpisu 2.4, mohlo by se stát, že bude dotaz odchycen a do proměnné bude vloženo například *';DROP TABLE handlers; –*. Příkaz by poté vymazal celou databázovou tabulku.

```
"CREATE TABLE handlers (ID INT AUTO_INCREMENT PRIMARY KEY,
    HandlerID VARCHAR(20) NOT NULL, ModuleID VARCHAR(20) NOT NULL,
    ApplianceID VARCHAR(20) NOT NULL,
    Alias VARCHAR(100), Description VARCHAR(250),
    File VARCHAR(100) NOT NULL, Function VARCHAR(100) NOT NULL,
    FOREIGN KEY(ModuleID) REFERENCES
        modules(ModuleID) ON DELETE CASCADE,
    FOREIGN KEY(ApplianceID) REFERENCES
        appliances(ApplianceID) ON DELETE CASCADE);"

    "INSERT INTO handlers (HandlerID, ModuleID,
        ApplianceID, Alias,
        Description, File, Function)
    VALUES (%s, %s, %s, %s, %s, %s, %s);"

    "SELECT * FROM handlers WHERE ModuleID = %s AND applianceID = %s;"

    "SELECT * FROM handlers WHERE ModuleID = '" + moduleID + "';"
```

Výpis 2.4: Příklad použití jazyka SQL.

[7] software zajišťující práci s databází

[8] dá se použít na všech nejznámějších operačních systémech

[9] SQL je od roku 1986 standardizovaný strukturovaný dotazovací jazyk (umožňuje ovládání databáze pomocí příkazů) používaný pro práci s daty v relačních databázích

Kapitola 3

Návrh

Na začátku této kapitoly jsou shrnuta řešení dostupná na trhu, je předložen úvod do problematiky automatizace, znázorněn návrh, jak bude vypadat zapojení prvků v systému a jaké parametry by měl splňovat webový server.

3.1 Studium existujících řešení

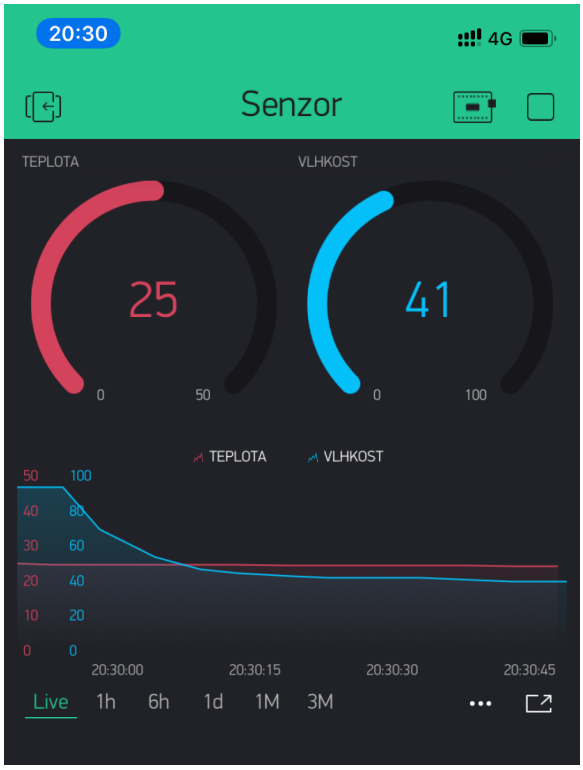
Na trhu existuje nespočet komerčních řešení. Zde je výčet těch nejčastějších, se kterými se může uživatel v oblasti, kterou se v této práci zabýváme, setkat.

3.1.1 IFTTT

IFTTT je asi nejznámější bezplatně dostupné řešení založené na stejném principu, jako tato bakalářská práce. Umožňuje propojit externí služby (Gmail, Facebook apod.) buď s dalšími službami a nebo s chytrými zařízeními v domácnosti. V praxi například odešle email, pokud konkrétní uživatel na Twitteru přidá příspěvek s určitým hashtagem. Jeho největší nevýhodou je uzavřenost systému. Uživatel má k dispozici pouze API, ale nemůže si řešení upravit podle sebe. Na trhu je možno najít spoustu komerčních produktů (kamer, termostatů, světel, atd.), které využívají řešení IFTTT, popřípadě lze s použitím nabízeného API vytvořit vlastní zařízení. Budoucím rozšířením této práce může být napojení IFTTT na vytvořený systém a tím poskytnou uživateli příležitost jednoduše reagovat na externí služby (email, SMS, atd.).

3.1.2 Blynk

Blynk [20] je služba umožňující vytvoření jednoduchého rozhraní pro zobrazení dat ze senzorů a jejich ovládání (na obrázku 3.1). V bezplatné verzi nabízí připojení až dvaceti zařízení a použití 2 000 *Blynk.Energy* (měna pro vkládání dalších uživatelských prvků). Každý další prvek v aplikaci má předem stanovenou cenu (například za tlačítko je to 200 *Blynk.Energy*). Pokud uživatel prvek odstraní, bude mu cena vrácena. Nejlevnější balíček 1000 *Blynk.Energy* stojí 79 Kč, nejdražší poté obsahuje 28 000 jednotek měny za 579 Kč. Aplikaci na obrázku 3.1 jsem zprovoznil zhruba do 30 minut za 1 500 *Blynk.Energy*. Služba je velmi uživatelsky přívětivá, knihovna pro propojení hardwaru dobře zdokumentovaná a nenáročná na používání. Nevýhodou je, že uživatel nemůže zařízení nijak automatizovat (minimálně ne pomocí této služby, případná automatizace by musela být v mikrokontroleru s ovládáním přes tlačítka, posuvníky apod.) a fakticky se za ni musí zaplatit.



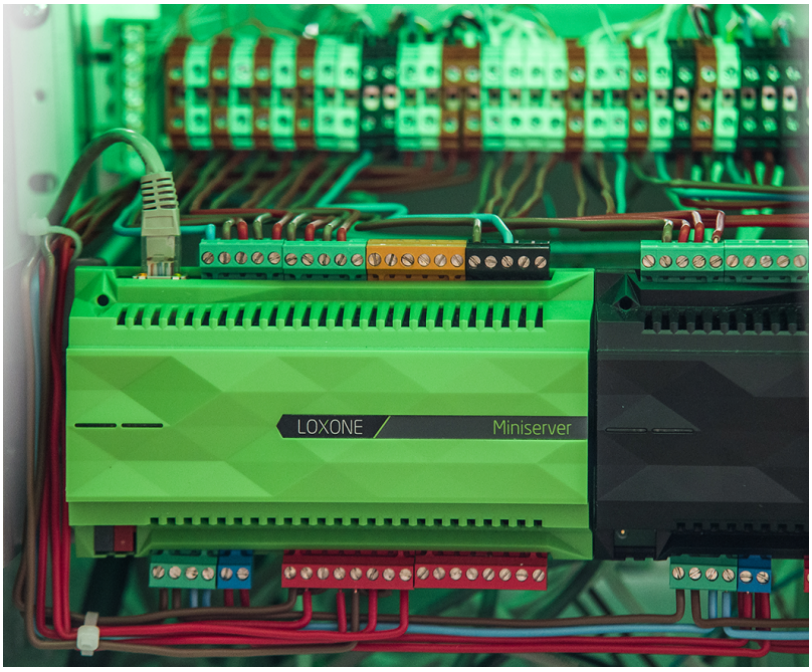
Obrázek 3.1: Příklad mnou vytvořeného rozhraní pro zobrazení dat z teplotního senzoru pomocí služby Blynk.

3.1.3 PushBullet

PushBullet je webové rozšíření, popřípadě aplikace, umožňující připojení libovolného počtu zařízení (počítačů, telefonů, prohlížečů) a následné zasílání zpráv mezi nimi. Lze například klonovat upozornění z telefonu do počítače. Dá se využít pro notifikaci uživatele pomocí SMS zpráv nebo upozornění v telefonu.

3.1.4 Loxone

Rakouská společnost Loxone založená roku 2009 patří mezi průkopníky v ovládání chytrých domů. Poskytuje komplexní řešení automatizace domácnosti pomocí centrální jednotky *Miniserver* (na obrázku 3.2). Na webových stránkách nabízí příklady kalkulací pro rodinný dům o rozloze 120 m^2. Nejprémiovější varianta obsahuje 74 funkcí včetně ovládání světel pomocí detektorů pohybu, regulace vytápění, vypínání definovaných elektrických okruhů při opuštění domu či centrálního audia ve všech místnostech apod. Toto vše vyjde zhruba na 386 000 Kč včetně DPH bez odborné montáže komponent. Řešení se ovládá primárně přes mobilní aplikaci, a samotná společnost propaguje méně ovládacích prvků fyzicky přítomných v místnostech. Díky síti partnerů si uživatel může vybrat řešení přímo na míru (nemusí využít všech 74 funkcí, popřípadě může přikoupit další) a mít chytrou domácnost na klíč. Velikou nevýhodou tohoto řešení je naprostá uzavřenost celého systému. Uživatel dostane mobilní aplikaci, přes kterou ovládá dům. Jestliže se mu některé ovládací prvky nelíbí nebo mu například chybí, nemůže se vzhledem a funkčností nic udělat. Pokud chce rozšířit svůj dům o další chytré prvky, musí se obrátit na společnost *Loxone* a vybrat si některé z jejích řešení.



Obrázek 3.2: Ukázka zapojení centrální jednotky *Miniserver* od společnosti Loxone (převzato z https://www.digitalnidomy.cz/otevreli-jsme-pro-vas-novy-loxone-showroom/).

3.1.5 Fibaro

Yatun, s.r.o. je výhradní dodavatel řešení Fibaro s centrální jednotkou *FIBARO Home Center*. Ovládá se přes mobilní aplikaci (obrázek 3.3) popřípadě přes webové rozhraní. Společnost nenabízí komplexní řešení jako v případě Loxone (3.1.4), ale spíše jednotlivá zařízení, která se připojí k serveru. Všechny produkty používají otevřený protokol Z-Wave určený pro internet věcí (některé prvky se dají ovládat i pomocí Apple HomeKit). Uživatel v tomto případě není vázán pouze na jednoho výrobce a může si vybrat z mnoha chytrých zařízení. Samotná technologie ovládání je ale opět proprietární a proto nelze změnit například uživatelské rozhraní.

Obrázek 3.3: Ukázka ovládání mobilní aplikace Fibaro. Na levé straně je vidět pohled na hlavní menu s oblíbenými položkami pro rychlejší manipulaci. Na pravé straně je ovládání rolet v ložnici (převzato z https://www.fibaro.com/cz/smart-home-app/).

3.1.6 Hlasově ovládané systémy

Technologičtí giganti jako Amazon, Google a Apple patří mezi nejznámější hlasová řešení na trhu. Hlasoví asistenti Alexa[1], Google Assistant[2] a Siri[3] umožňují připojit podporovaná zařízení a ta ovládat hlasem. Řešení spolehlivě funguje v angličtině. S češtinou si v omezené míře produkty také rozumí a lze předpokládat, že za několik let bude možné s těmito asistenty hovořit plynně v jakémkoliv jazyce. Společnosti také nabízí mobilní aplikace pro zobrazení statistik a jednodušší nastavení činností.

3.2 Úvod do problematiky

Automatizací rozumíme zvýšení komfortu obyvatel domu, zaručení jejich bezpečí, popřípadě snížení spotřeby energií pomocí vyhodnocení naměřených hodnot ze senzorů umístěných v prostorách domu. Tím nejjednodušším případem automatizace je otevření dveří při vstupu do prodejny, rozsvícení světel na chodbě při průchodu kolem čidla pohybu apod. Obliba použití takovýchto systémů roste i v běžných domácnostech v rámci modernizace, kde mnohé společnosti nabízejí své možnosti řešení (viz. bod 3.1). V nejvyšších úrovních automatizace hovoříme prakticky o umělé inteligenci, která dokáže sama předpovídat naše

[1] více než 100 000 podporovaných zařízení od více než 9 500 výrobců [1]
[2] přes 10 000 zařízení od více než 1 000 výrobců
[3] HomeKit [3] - v říjnu roku 2019 450 podporovaných zařízení

potřeby a úmysly (například zaznamenáním počtu osob, ba dokonce identifikací příchozích členů domácnosti i cizích návštěvníků, kteří s námi do domu vstoupili) a sama na ně náležitě reagovat. V nižších úrovních nám automatizace předkládá data ze senzorů, na které můžeme buď sami přes rozhraní reagovat, nebo na ně dům pomocí předem naučených zvyklostí reaguje sám.

Tohoto systému se většinou docílí propojením dílčích zařízení, která jsou schopná spolu navzájem komunikovat a vyměňovat si informace, které jsou ve většině případů studovaných systémů automatizace domácnosti dosazovány do podmínek, které jsou následně vyhodnocovány a na jejichž základě jsou prováděny jednotlivé akce. Uživatel do tohoto systému manuálně zasahuje a aktivně s ním komunikuje.

Všední uživatelé bez pozadí technologického vzdělání se spokojí s komerčním uzavřeným systémem, neboť je pro ně důležitý spíše výsledek procesu (na základě pohybu se rozsvítí světlo), nežli proces samotný (například struktura komunikace mezi zařízeními).

Pro technologicky zdatnějšího uživatele může být pohodlnější a lákavější si systém vytvořit sám, protože má povědomí, jak vše funguje a může si jej tedy kompletně přizpůsobit dle svých požadavků.

3.3 Návrh systému

Středobod celého systému je tvořen serverem běžícím na libovolném kusu hardwaru (nejčastěji je jím však kvůli ceně a jednoduchosti výkonnější mikropočítač). Přes tento server probíhá veškerá komunikace (přijímá data a vydává impulzy ostatním prvkům v systému). Jsou na něm uložena veškerá data o systému, běží na něm webové rozhraní a v našem případě i aplikace umožňující komunikaci v systému.

Řídící jednotky (desky, ke kterým jsou připojeny senzory a spotřebiče) jsou vlastně mikrokontrolery, které komunikují pouze se serverem. Nemají tedy povědomí o ostatních řídících jednotkách a fungují jako autonomní celky (jsou modularizované). Pro zajištění integrity systému řídící jednotky periodicky oznamují serveru, že jsou s ním stále ve spojení a fungují.

Komunikace mezi řídící jednotkou a serverem je ve většině případů zprostředkovávána prostřednictvím WiFi. K přenosu dat lze využít mnoha protokolů, například HTTP či MQTT. Dalším možným přístupem je užití vysílače/přijímače rádiového signálu.

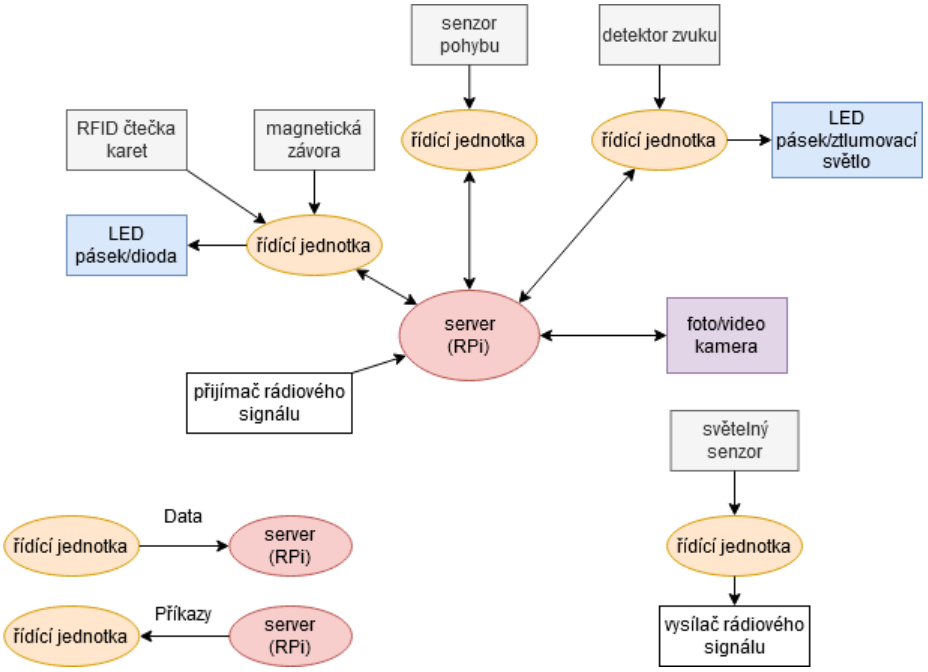
Senzory (například teplotní čidla) zaznamenávají různé hodnoty periodicky či v rámci reakce na provedenou akci uživatele (přečtená přístupová karta) a předávají je řídícím jednotkám, se kterými jsou propojeny. Hodnoty mohou být poté různě zpracovávány (například převedeny do určitého formátu) a odeslány serveru.

Spotřebiče jsou veškeré prvky v systému, které aktivně neprodukují data, ale dají se jakkoliv ovládat (například je vypínat a zapínat). Čekají na přijetí příkazu od řídící jednotky, který následně zpracují a vyplní.

V práci dále není rozlišováno, zda se jedná o senzor či spotřebič, a proto jsou jednotně označovány jako *zařízení*. Tato zařízení jsou připojena k řídící jednotce, v ojedinělých případech poté přímo k serveru (například kamera).

Na obrázku 3.4 je konkrétní návrh systému, který je rozdělen do modulárních částí připojených k serveru. Jednotlivé části (moduly) jsou ovládány řídící jednotkou.

Podrobnější popis jednotlivých navržených modulů v bodě 3.5.



Obrázek 3.4: Návrh systému s použitím centrálního serveru a zařízení připojených k řídícím jednotkám, popřípadě k serveru samotnému. Modrou barvou jsou vyobrazeny spotřebiče, které přijímají pouze příkazy, šedou poté senzory, které data pouze poskytují serveru a fialovou senzor, který provádí oba dva výše jmenované úkony. Vlevo dole je znázorněn význam jednotlivých šipek. Vpravo dole je řídící jednotka připojena k vysílači rádiového signálu a nekomunikuje tedy standardně pomocí WiFi.

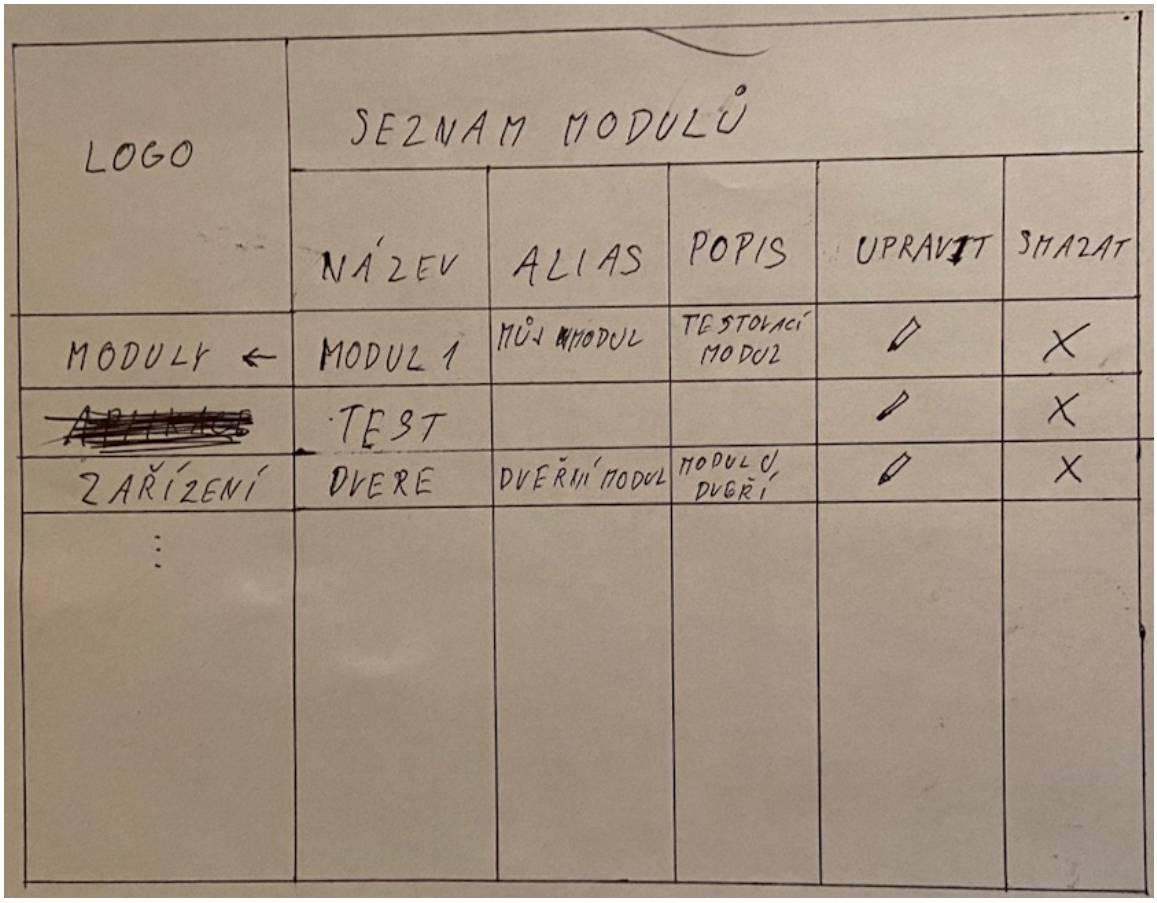
3.4 Webový server

Webové rozhraní jsem vytvořil z důvodu usnadnění práce uživateli. Bez něj by musel vkládat podmínky a akce do databáze ručně, popřípadě používat napsané skripty.

Webová aplikace bude umožňovat uživateli prohlížet veškeré moduly, které budou v přehledné tabulce a dále procházet a studovat k nim příslušící údaje. Pomocí webového rozhraní může uživatel tyto hodnoty pouze mazat (jelikož jejich přidávání je řešeno v konfiguraci řídící jednotky – bod 4.3.3) a upravovat jejich popisky. Jedna z hlavních funkcí bude možnost nastavit podmínky a akce, které se vykonají na základě dat získaných ze senzorů. Aplikace také bude umět zobrazit statistiky k jednotlivým zařízením a uživatel by měl mít možnost si tato data rychle a jednoduše prohlížet.

Na obrázku 3.5 je návrh rozložení webového rozhraní. V levé části stránky se nachází menu, pomocí kterého může uživatel přepínat zobrazení jednotlivých prvků. Hlavní část stránky zabírá vždy tabulka obsahující seznam prvků zvolených pomocí menu (na nákresu jsou to moduly).

V tabulce jsou vypsány všechny relevantní informace (například název a popis prvku). Uživatel má možnost v každé tabulce data mazat a s výjimkou dat ze senzorů je i editovat (pouze informační prvky – popisky apod). Jelikož veškeré údaje jsou převzaty z konfigurace, může uživatel přidávat pouze podmínky a akce.



Obrázek 3.5: Návrh uživatelského rozhraní. Vlevo se nachází menu pro navigaci. Hlavní část zabírá tabulka pro zobrazení vybraného prvku.

3.4.1 Definice modulů a zařízení

Z důvodu zachování konzistence dat jsem se rozhodl, že při konfiguraci modulu proběhne výměna informací o všech zařízeních a datových typech, které poskytuje, jelikož v době připojení k serveru modul ví jakými zařízeními disponuje (je naprogramován uživatelem, aby s těmito zařízeními pracoval). Pokud by uživatel vkládal informace o modulech sám, mohlo by dojít k nekompatibilitě zadaných dat s reálným stavem modulu a tím pádem k nefunkčnosti celého systému. O procesu konfigurace se můžete dočíst v bodě 4.3.3.

3.4.2 Definice podmínek

Jelikož jsem chtěl podmínky a akce jednoduše řetězit za sebe, rozhodl jsem se je rozdělit na dva logické celky.

Při definování jednoduchých podmínek uživatel vybere modul a zařízení, se kterými chce pracovat, a definuje jaká data mají být hlídána (např. přečtená karta v případě RFID čtečky). Pro tato konkrétní data určí s pomocí předem připravených operátorů (rovnost, větší/menší atd.) jakých hodnot mají nabývat (např. mají se rovnat hodnotě F6F929). Při nastavování podmínek uživatel neřeší datové typy, protože v databázi jsou informace uloženy vždy v textovém formátu.

Při vytváření složité podmínky uživatel zasahuje přímo do kódu skriptu běžícího na RPi. V uživatelském rozhraní poté definuje soubor a název funkce, kde se nachází její implementace. O tom, jak jsou podmínky implementovány a jak vytvořit komplexní podmínku, se dočtete v bodě 4.2.3.

Tyto podmínky lze libovolně spojovat pomocí logických funkcí AND a OR. Následně uživatel definuje akci, která bude vykonána po splnění této podmínky.

3.4.3 Definice akcí

Definice akcí využívá podobného principu jako definice podmínek. V případě jednoduchých akcí uživatel vybírá modul, zařízení a data, která mají být změněna a novou hodnotu těchto dat.

Složitou akci lze definovat odkazem na soubor a názvem funkce, pomocí které je akce implementována.

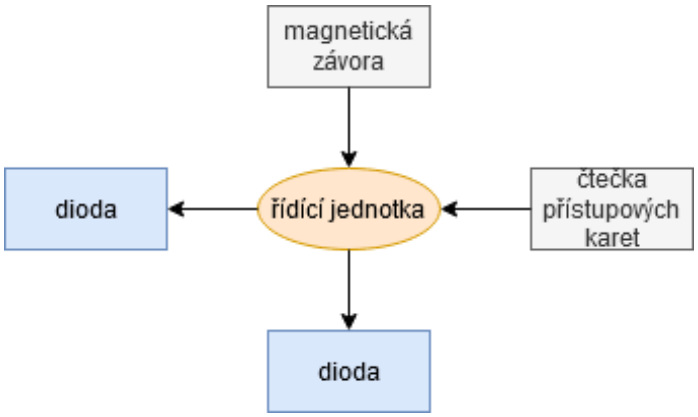
Uživatel může akce řetězit a to takovým způsobem, že definuje následující akci, která bude provedena po vykonání současně probíhající akce. Dále o akcích v bodě 4.2.4.

3.5 Navržené moduly

Modul by měla být samostatná jednotka vykonávající pouze práci, ke které byla určena. Z pohledu návrhu není až tak důležité, zda bude modul ovládat RPi (bod 2.1.1), NodeMCU (bod 2.1.2), popřípadě jiný mikrokontroler. Práce počítá s tím, že zvolený hardware bude mít nějakou možnost přenést data na server (ať už přes WiFi, kabelem, či přes 433MHz vysílače).

3.5.1 Dveřní modul

V mém návrhu má k dispozici dvě diody pro poskytnutí zpětné vazby uživateli. Hlavní součástí jsou ovšem dva bezpečnostní prvky a to čtečka přístupových karet a magnetická závora umístěná na dveřích (schéma na obrázku 3.6).



Obrázek 3.6: Návrh dveřního modulu. Šedou barvou jsou znázorněny senzory, které posílají naměřené údaje řídící jednotce. Modrou barvou poté spotřebiče, které přijímají příkazy.

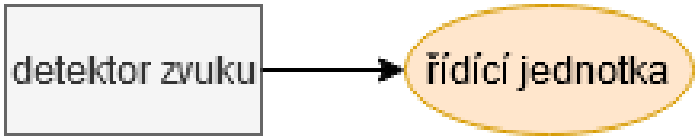
3.5.2 Senzor pohybu

Senzor pohybu (bod 2.1.6) bychom pro správné zabezpečení domu/bytu měli umístit minimálně do každé místnosti, která má vnější přístupový bod (dveře/okna). Z tohoto důvodu by při návrhu měl senzor pohybu figurovat jako jeden z nejčastěji používaných modulů (schéma na obrázku 3.7). Pro možnost indikace, zda senzor zaznamenal pohyb, jsem zde umístil jednu diodu (bod 2.2.3).

Obrázek 3.7: Návrh pohybového senzoru (šedou barvou) s použitím diody (modrá barva) pro indikaci snímání.

3.5.3 Detektor zvuku

Detektor zvuku (schéma na obrázku 3.8) bude přes den neaktivní (může být aktivován například přečtením přístupové karty při odchodu z domu). V určitou hodinu, kterou uživatel nastaví, se zapne a bude sloužit jako detekce pro případ rozbití oken apod.



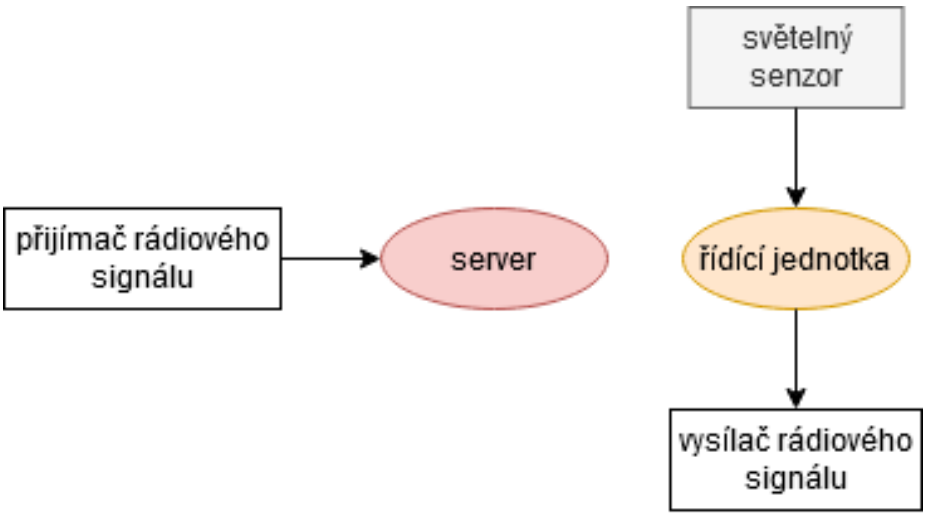
Obrázek 3.8: Znázornění modulu s detektorem zvuku.

3.5.4 Světelný senzor

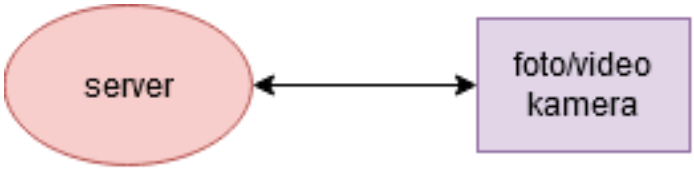
Světelný senzor (schéma na obrázku 3.9) je připojen k řídící jednotce, která nedokáže komunikovat pomocí WiFi. Z tohoto důvodu je k ní připojen vysílač rádiového signálu pracující na frekvenci 433MHz. Přijímač tohoto signálu je připojen přímo k serveru, který zpracovává údaje o hladině svitu v místnosti. Modul se dá využít například pro automatické zatažení žaluzií v případě velkého svitu, nebo pro úsporu elektřiny.

3.5.5 Kamera

Modul s kamerou (schéma na obrázku 3.10) může být připojen přímo k serveru (na RPi je k tomuto účelu speciální konektor). V závislosti na umístění RPi může sloužit k zajištění bezpečnosti (například nahrávání dění u vchodových dveří při zaznamenání jejich otevření), jako dětská chůvička nebo kontrola domácích mazlíčků apod. Šipky ve schématu jsou oboustranné, jelikož kamera zároveň poskytuje data i přijímá příkazy.



Obrázek 3.9: Schéma zapojení světelného senzoru. Ten je připojen k řídící jednotce, která nemůže komunikovat pomocí WiFi a je proto připojena k vysílači rádiového signálu. Přijímač rádiového signálu je umístěn přímo u serveru, který data zpracovává.



Obrázek 3.10: Modul kamery připojený přímo k serveru.

Kapitola 4

Implementace

Tato kapitola popisuje způsob, jakým je v práci implementována komunikace mezi serverem a řídícími jednotkami, jak je vytvořeno webové rozhraní a jak jsou naprogramovány řídící jednotky i samotný server. Je zde popsáno, jak byly výsledky testovány a výčet všech mnou implementovaných modulů.

4.1 Komunikace

Tato část popisuje jakou technologii jsem zvolil pro výměnu zpráv mezi zařízeními a datovou strukturu zpráv.

4.1.1 MQTT broker

Jako MQTT broker je použit Mosquitto od Eclipse Foundation. Je to vlastně aplikace běžící na počítači (v našem případě na RPi), která přijímá veškeré zprávy (ať už od řídících jednotek nebo serveru) a přeposílá je dále. Hlavní výhodou této technologie je, že klienti (jednotlivé řídící jednotky) o sobě vůbec nevědí, jsou pouze připojeni k totožnému kanálu. Uživatel se může k brokerovi připojit a přímo sledovat veškerou probíhající komunikaci, popřípadě sám zasílat do kanálů zprávy. Broker by měl být zabezpečen heslem, ovšem v tomto případě vše probíhá na lokální síti uživatele a komunikace je tedy nezabezpečená.

4.1.2 Princip vyměňování zpráv

Komunikace mezi řídící jednotkou a serverem je založena na vzájemném vyměňování zpráv přes WiFi pomocí protokolu MQTT (bod 2.4.1). Každá vyměněná zpráva má speciální strukturu dat ve formátu JSON (bod 2.3.3).

V případě, že komunikace pomocí WiFi nebude možná, se dá použít přijímač a vysílač 433MHz signálu. Pokud taková situace nastane, musí být přijímač připojen buď přímo k serveru, který bude data zpracovávat a vkládat do databáze a nebo k jiné řídící jednotce, která se k přijímači bude chovat jako k zařízení a přes WiFi standardním způsobem přenese data (stejně se dá pracovat i se zasíláním příkazů).

Ve výpisu 4.1 lze vidět níže popsaná struktura vyměňovaných zpráv.

- při zasílání dat z řídících jednotek se použije struktura *moduleData*
- pro odesílání příkazů je použita struktura *moduleCommand*

- *moduleID* je v celém systému unikátní identifikátor modulu[1]
- *applianceID* je identifikátor zařízení (dveřní modul, červená LED atd.) unikátní vůči *moduleID*[2]
- *dataCount/commandCount* obsahuje počet odesílaných dat/příkazů[3]
- *data/command* je pole struktur (dat či příkazů) obsahující vždy *id*[4] a *value*[5]

```
{
  "moduleData":
  {
    "moduleID":"ESP123456",
    "applianceID":"temp_bedroom",
    "dataCount":"2",
    "data": [
      {
        "id":"temperature",
        "value":"21.8"
      },
      {
        "id":"humidity",
        "value":"42"
      }
    ]
  },
  "moduleCommand":
  {
    "moduleID":"ESP654321",
    "applianceID":"fan_bedroom",
    "commandCount":"2",
    "command": [
      {
        "id":"on",
        "value":"true"
      },
      {
        "id":"speed",
        "value":"80"
      }
    ]
  }
}
```

Výpis 4.1: Struktura zasílaných dat

[1] v této práci je jako tento identifikátor použito číslo čipu NodeMCU
[2] dva různé moduly mohou mít připojená dvě různá zařízení se stejným názvem
[3] používá se pro validaci a pro jednodušší rozlišování mezi zprávami při konfiguraci
[4] identifikátor hodnoty unikátní vůči modulu a zařízení
[5] naměřená hodnota v případě dat a nová hodnota u příkazů

4.2 Server

Jako server jsem pro tuto práci zvolil Raspberry Pi, jelikož je snadno dostupný a umožňuje spuštění operačního systému. Na něm běží zprostředkovatel zpráv MQTT (bod 4.1.1) a zároveň skript pro příjem dat ze senzorů, vyhodnocování podmínek a odesílání příkazů. Dále je zde zprovozněn webový server, který slouží k ovládání celého systému.

4.2.1 Struktura souborů

V hlavním souboru jsou funkce, které se volají při přijetí dat od řídících jednotek (více v bodech 4.1 a 4.3.3) a nekonečná smyčka, která každé 2 sekundy prochází aktuální podmínky z databáze a kontroluje, zda jsou splněny (viz. body 4.2.3 a 4.2.4).

MQTT klient je implementován s pomocí knihovny Paho MQTT [2] ve dvou samostatných souborech – jeden slouží k přijímání dat od řídících jednotek a druhý k odesílání příkazů.

Soubor *database_setup.py* musí být uživatelem spuštěn dříve než server, po zapnutí totiž vytvoří všechny potřebné databázové tabulky.

Pomocí funkcí uložených v souboru *database_controller.py* se pracuje s databází. Aby byla použita vždy nejaktuálnější data z databáze, je v nekonečné smyčce v hlavním souboru vždy před kontrolou podmínek obnoveno (ukončeno a opět navázáno) spojení z databází.

Uživatel může zásahem do souboru *config.py* například změnit periodu kontroly podmínek, názvy použitých kanálů apod.

4.2.2 Přijetí dat serverem

Uživatel do hlavičky souboru *handlers_init.py* vloží všechny své soubory a funkce, které se starají o příjem a uloží data do databáze. Tyto soubory jsou uloženy ve složce *handlers*. Každá funkce má vstupní parametr *moduleData*, který obsahuje JSON strukturu definovanou v bodě 4.1. Implementace je plně v rukách uživatele, který může zde přijatá data libovolně upravovat. V případě úspěchu by měla implementační funkce vkládat data do tabulky *data* a při neúspěchu vyvolávat výjimku *ParsingDataError*.

Příklad použití je uveden v souboru *example.py*.

4.2.3 Implementace podmínek

V předem stanovené periodě (2 sekundy) server načítá všechny podmínky z databáze a zahajuje jejich vyhodnocování.

V cyklu prochází podmínky a vybírá tu první z pole podmínek a následně volá funkci *evaluateResult*. Tato funkce poté zavolá funkci *processCondition* (rozděleno do dvou funkcí z důvodu použití rekurze) a v ní, pokud se jedná o jednoduchou podmínku, načítá z databáze pro aktivní zařízení[6] poslední nezpracovaný údaj, popřípadě poslední údaj pro neaktivní zařízení. Z tohoto důvodu se může stát, že pokud dojde k zaznamenání stejného údaje v časech 0 a 1, bude v čase 2 zpracován údaj zaznamenaný v čase 1 a až v čase 4 bude teprve zpracován údaj z času 0. V ojedinělých případech (například při ukládání hodnoty každou sekundu) by toto chování mohlo dělat problém. Při běžném použití ovšem tato situace nastane velmi zřídka. Pokud existuje nějaká nová hodnota načteného údaje, je pomocí operátoru a hodnoty v podmínce vyhodnocena.

[6] jedná se o senzor – zařízení zasílá data na server

V případě kladného vyhodnocení a neexistence další podmínky, je zavolána požadovaná akce. Pokud existuje navazující podmínka a je použit logický operátor *AND*, řízení se vrátí do funkce *evaluateResult*, která vyhledá navazující podmínku a rekurzivním voláním sebe sama ji vyhodnotí. Použijeme-li logický operátor *OR*, je vráceno řízení a probíhá mazání všech následujících podmínek z pole a vyhodnocení akce na poslední zřetězené podmínce.

Jestliže podmínka není splněna, nevykonává se žádná akce a pokud je definován logický operátor *OR*, vyhodnocuje se další podmínka. V případě operátoru *AND* jsou všechny navazující podmínky ignorovány. Jinými slovy je zde využito principu odloženého vyhodnocování[7].

Pro vyhodnocení komplexních podmínek se volá uživatelem vytvořená implementace uložená ve složce *conditions* (název souboru musí být definován v *conditions_init.py*). Uživatel zde může libovolně načítat data z databáze pomocí funkcí v souboru *database_controller.py* a provádět složité operace. Po vyhodnocení uživatel vrací řízení (návratová hodnota *success* a nebo *fail*) a v případě existence další podmínky se provádí její vyhodnocení (popřípadě volání akce).

4.2.4 Implementace akcí

Po zavolání funkce *processAction* (ať už automaticky z jednoduchých či uživatelem z komplexních podmínek) se v případě jednoduché akce vytvoří příkaz a ten se odešle na zařízení definované v akci (ta musí být předána jako parametr funkce). Nová hodnota měněného údaje se uloží do databáze.

Komplexní akce se vyhodnocují stejným způsobem jako komplexní podmínky. Uživatel ve složce *actions* vytvoří soubor s funkcí (definované v akci) a tento soubor vloží na začátek *actions_init.py*. V implementaci akce má přístup k databázi a může zde tedy kontrolovat další hodnoty a následně zasílat příkazy zařízením. Na konci implementace vrací uživatel řízení hlavnímu programu a v případě existence zřetězené akce je funkce *processAction* rekurzivně zavolána s novým parametrem.

4.3 Řídící jednotka obecně

4.3.1 WiFi

Každá řídící jednotka je naprogramovaná tak, aby při prvním spuštění vystavila svoji WiFi, na kterou se uživatel připojí pomocí hesla uloženého v konfiguračním souboru řídící jednotky. Při připojení otevře v prohlížeči stránku 192.168.4.1 a zadá přihlašovací údaje (SSID a heslo) k domácí WiFi. Na stejné stránce také může upravit adresu a port MQTT serveru. Při každém dalším spuštění se řídící jednotka pokusí připojit k síti pomocí zapamatovaných údajů. Při neúspěšném použití zapamatovaných údajů je opět vystavena vlastní WiFi a proces se opakuje.

Pro větší uživatelský komfort je možné při změně hesla WiFi v domácí síti změnit hesla na všech aktuálně připojených zařízeních. Toho je docíleno pomocí zprávy přes konfigurační kanál, kde server rozešle informaci o novém SSID a heslu sítě všem připojeným řídícím jednotkám. Tyto se po přijetí zprávy o nové konfiguraci sítě a dvouminutovém zpoždění od sítě odpojí a následně se zkouší připojit s novým nastavením. Uživatel zavolá skript *change_wifi_pass.py* s parametry SSID a heslo. Následně změní samotné nastavení WiFi

[7] v případě logického operátoru *AND* se řetězí, dokud podmínka platí, u *OR* dokud neplatí

pomocí administrace routeru. Doporučuji toto nastavení udělat co nejdříve po rozeslání informace o změně, jelikož se všechna zařízení od sítě odpojí.

V případě, že v průběhu činnosti dojde k odpojení od WiFi (například výpadek internetu), se řídící jednotka po doběhnutí jednoho cyklu v hlavní smyčce programu (maximálně po 30 sekundách) restartuje. Tím je znovu spuštěn proces připojení k WiFi a k serveru. Pokud dojde k odpojení od serveru (popřípadě od MQTT brokera), přejde řídící jednotka nejpozději po 60 sekundách do režimu konfigurace. V tomto módu začne blikat konfigurační dioda a odesílají se zprávy *initialize*, dokud nepřijde zpráva *registerDone* (v této chvíli řídící jednotka stále přijímá data od svých senzorů, ale nikam je neposílá a ani je neukládá). Po přijetí této zprávy se nepozději do 60 sekund zruší konfigurační režim. Zda je řídící jednotka plně funkční lze zjistit voláním funkce *MQTT::isInitialized*.

4.3.2 Přizpůsobení

Soubor *Config.h* obsahuje veškerá nastavení, která se mohou měnit pro každý modul (například název modulu, počet připojených zařízení atd.).

Kód souboru *mqtt.cpp* ve funkci *handleCommands* upraví tak, aby zavolala funkci (nejlépe na základě identifikátoru zařízení) definovanou v souboru *Handler.h*. Tato funkce se zavolá po příjetí příkazu a obstará jeho vykonání. Implementace je (spolu s veškerými údaji použitými při konfiguraci) uložená v souboru *handler.cpp*.

V souboru *main.cpp* potom uživatel volá funkce pro své senzory. Doporučuji pro zaznamenávání dat ze senzorů využívat přerušení, jelikož hlavní smyčka programu je použita pro zaručení, že je řídící jednotka připojena k serveru (v některých implementovaných modulech není pro čtení ze senzorů použito přerušení – bod 4.4.2).

4.3.3 Konfigurace

Na obrázku 4.1 lze vidět jakým způsobem se řídící jednotka po spuštění připojí k serveru.

Po nastavení odebírání zpráv z kanálů *commands* a *config_server* vyšle do konfiguračního kanálu (*config_modules*) svůj unikátní identifikátor ve zprávě *initialize* (pokud je parametr *reset* nastavený na *true*, server z databáze modul nejdříve odstraní a až poté pokračuje v konfiguraci). Server žádost zaregistruje a v případě, že identifikátor má již v databázi, pouze odešle potvrzovací zprávu (*registerDone*). V opačném případě vyšle do konfiguračního kanálu příkaz *registerModule*, kterým říká, aby řídící jednotka poslala svůj název a popis. Identifikátor zařízení může být v těchto zprávách prázdný, jelikož nás momentálně nezajímá a není zpracováván. Uvedená komunikace je zobrazena ve výpisu 4.2.

```
{
  "moduleData":
  {
    "moduleID":"ESP123456",
    "applianceID":"",
    "dataCount":"2",
    "data": [
      {
        "id":"initialize",
        "value":"true"
      },
      {
```

```
        "id":"reset",
        "value":"false"
      }
     ]
  }

  "moduleCommand":
  {
    "moduleID":"ESP123456",
    "applianceID":"",
    "commandCount":"1",
    "command": [
      {
        "id":"registerModule",
        "value":"true"
      }
     ]
  }
}
```

Výpis 4.2: Zprávy zasílané při initializaci. První zpráva oznamuje serveru žádost o zahájení inicializace bez smazání předchozích dat. Druhá obsahuje odpověď serveru pokud řídící jednotku nemá v databázi.

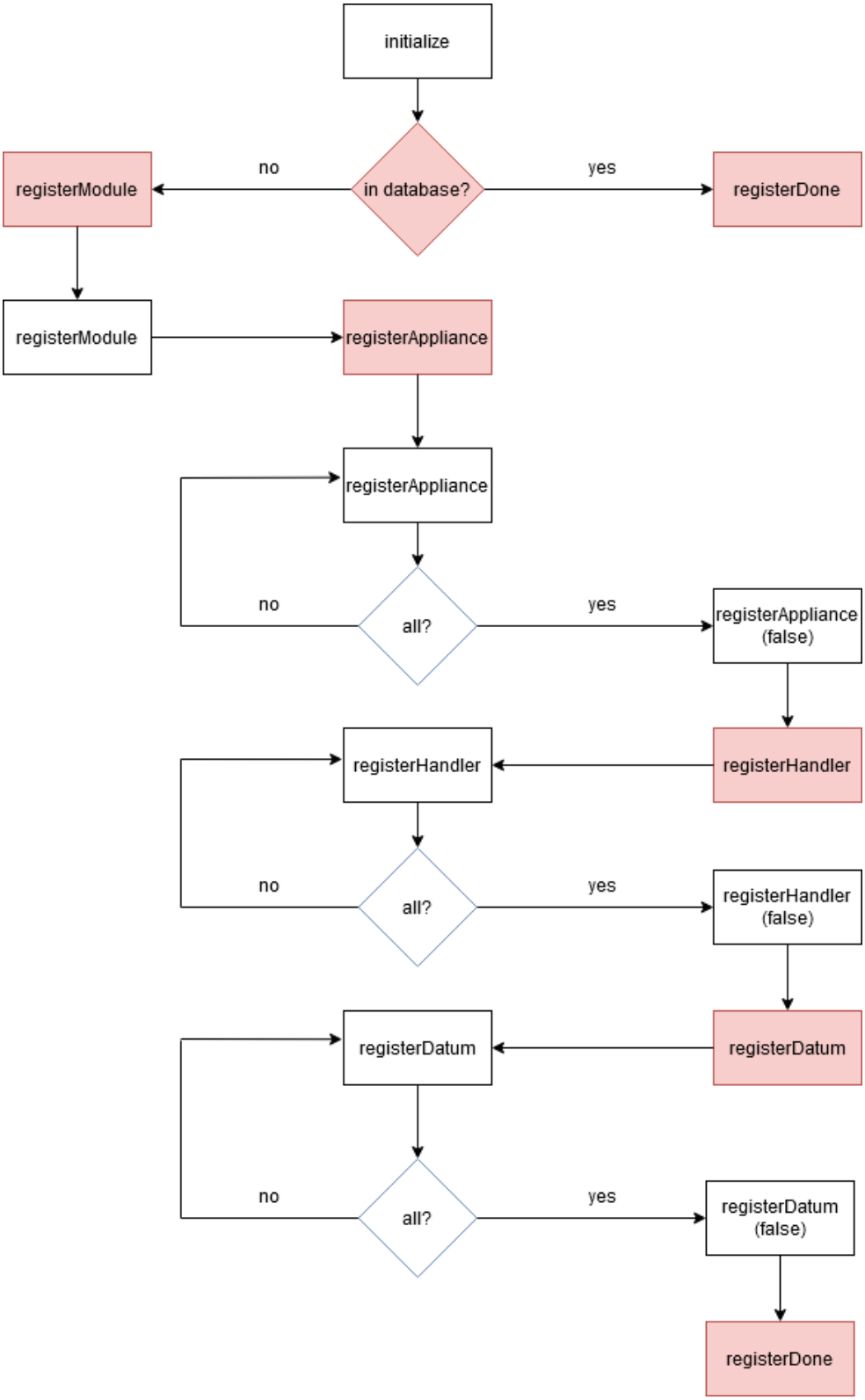
Po přijetí této zprávy serverem je řídící jednotka přidána do databáze a následně je vyslán příkaz *registerAppliance*. Na základě této zprávy řídící jednotka začne odesílat informace (název a popis) o svých zařízeních. Před odesláním další zprávy čeká na odpověď od MQTT brokera (konfigurace je tak uměle prodloužena, jelikož řídící jednotce dochází při odesílání velkého množství zpráv v krátkém časovém horizontu paměť). Po odeslání informace o posledním zařízení řídící jednotka zašle zprávu *registerAppliance* s hodnotou *false*.

Server odpoví zprávou *registerHandler*. Řídící jednotka po přijetí tohoto příkazu začne odesílat serveru názvy souborů (stejný princip jako u registrování zařízení), ve kterém se nachází funkce, které budou zpracovávat přijatá data (více v bodě 4.2.2). Jakmile je odeslána poslední informace, odešle se zpráva *registerHandler* s hodnotou *false*.

Sever po přijetí této zprávy vyšle příkaz *registerDatum*. Řídící jednotka začne posílat údaje (název, popis, datový typ) o jednotlivých hodnotách, které spotřebiče ovlivňují (např. teplota, zapnutí LED, karta přečtená ze čtečky atd.). Princip je opět stejný jako v případě registrace zařízení a zpracovávačů dat (čeká se na potvrzení od MQTT brokera a po odeslání posledního údaje řídící jednotka odpovídá stejnou zprávou s hodnotou *false*.

Po tomto procesu server odesílá zprávu *registerDone* čímž dává najevo, že připojení řídící jednotky k serveru bylo dokončeno.

Jakákoliv data ze zařízení řídící jednotka dále vysílá pomocí kanálu *data* a veškeré příkazy přijímá na kanálu *commands* (příkazy jsou tedy odesílány naráz všem řídícím jednotkám, ale pokud se *moduleID* ve zprávě nerovná *moduleID* řídící jednotky, je zpráva ignorována). Ke kanálu *config_server* zůstává modul stále připojený, jelikož v intervalu 30 sekund je zpráva *initialize* odesílána znovu (řídící jednotka se tak ujišťuje, že je stále připojená k serveru – respektive k Python skriptu běžícímu na RPi) a navíc na tomto kanálu může přijít příkaz ke změně parametrů WiFi připojení.



Obrázek 4.1: Konfigurace řídící jednotky po spuštění. Červenou barvou označeny akce serveru, bílou akce řídící jednotky.

4.4 Implementované moduly

4.4.1 Dveřní modul

Dle návrhu (viz. bod 3.5.1) jsem implementoval modul umístěný u dveří. Zvolil jsem červenou a žlutou diodu. Jejich chování je definováno v souboru *handler.cpp* a jedná se o jejich o rozsvícení v případě, že je přijat údaj *led_red_on* popřípadě *led_yellow_on* s hodnotou *true* nebo o jejich zhasnutí pro hodnotu *false*.

RFID čtečka (bod 2.1.5) komunikující pomocí protokolu Wiegand je popsána v souborech *CardReader.h* a *cardReader.cpp*. Pro její implementaci je využita knihovna Yet Another Arduino Wiegand Library [12]. Po zaznamenání karty je identifikátor převeden do hexadecimálního tvaru a pod údajem *card_id* odeslán na server (kód pro zachycení byl převzat z tutoriálů použité knihovny).

Jeden kabel magnetické závory je s nastavením interního pull-up rezistoru (bod 2.2.4) připojen do řídící jednotky, druhý je uzemněn. V případě, že je na pinu logická hodnota nula (o logických hodnotách v bodu 2.2.2), jsou dveře zavřené.

Výsledné zapojení je vidět na obrázku 4.4.

4.4.2 Senzor teploty a vlhkosti

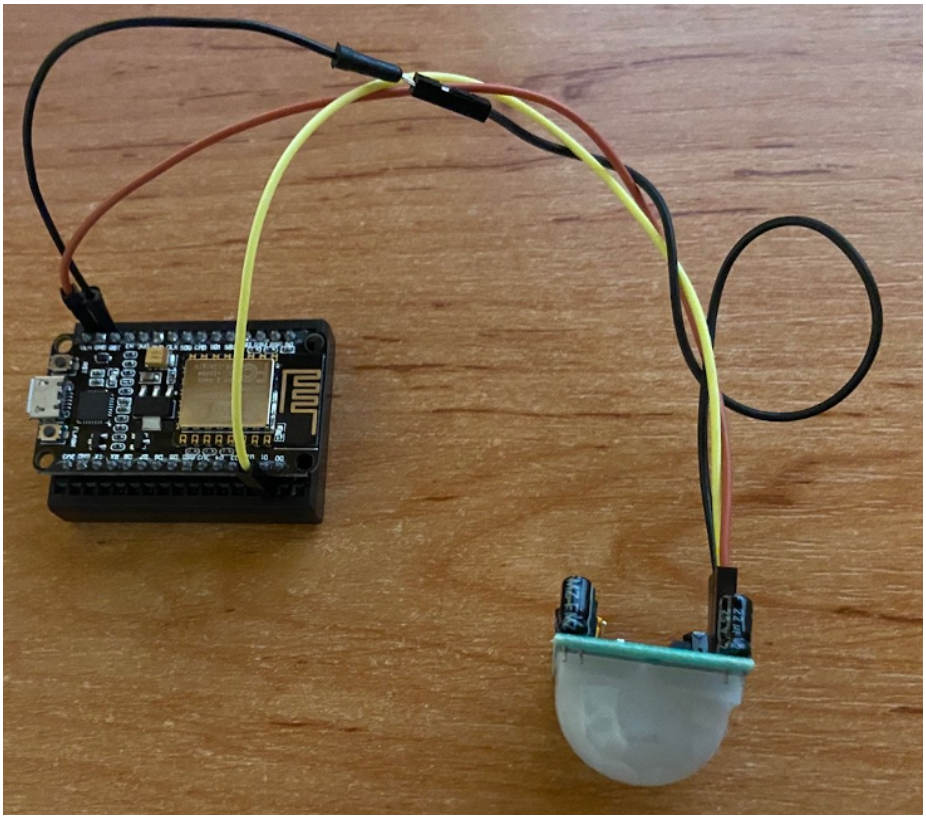
Snímání teploty a vlhkosti je prováděno senzorem DHT11 (bod 2.1.3) pomocí knihovny DHT sensor library [10]. Pomocí dvou funkcí se načtou obě dvě hodnoty a odešlou se na server pod údaji *temperature* a *humidity*. Modul je napojen na službu Blynk (bod 3.1.2) a jeho data lze zobrazit v mobilní aplikaci (podobně jednoduše se dají propojit i další moduly). Dle doporučení v dokumentaci DHT11 jsem k datovému kabelu zapojil rezistor. Jelikož jsem neměl k dispozici rezistor s nižším odporem a z důvodu omezeného místa v nepájivém poli, které neposkytovalo dostatečný prostor pro paralelní zapojení, jsem zvolil 10kΩ odpor.

Výsledné zapojení je vidět na obrázku 4.3.

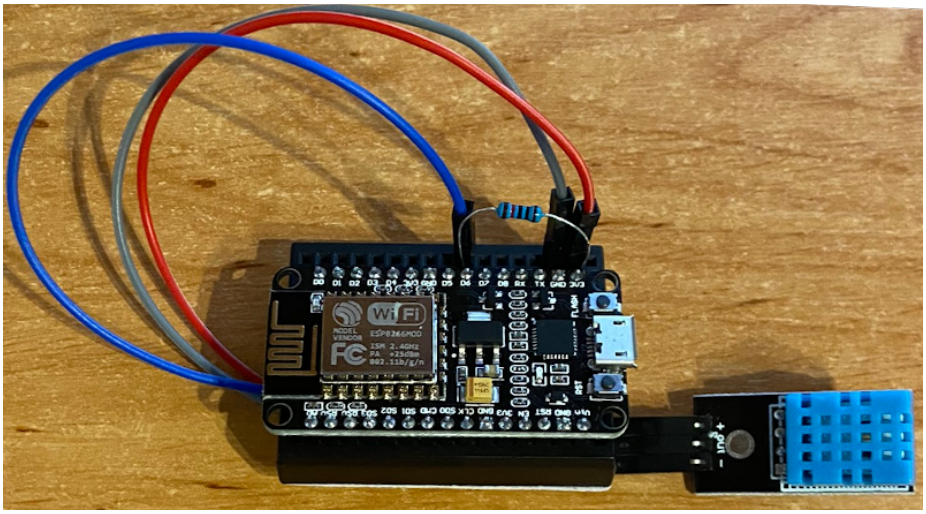
4.4.3 Senzor pohybu

Pasivní infračervené čidlo reaguje na pohyb v zaznamenávané oblasti. Ke čtení dat není potřeba použít knihovnu, stačí v určité periodě (zvolil jsem jednu sekundu) číst logickou hodnotu na připojeném pinu.

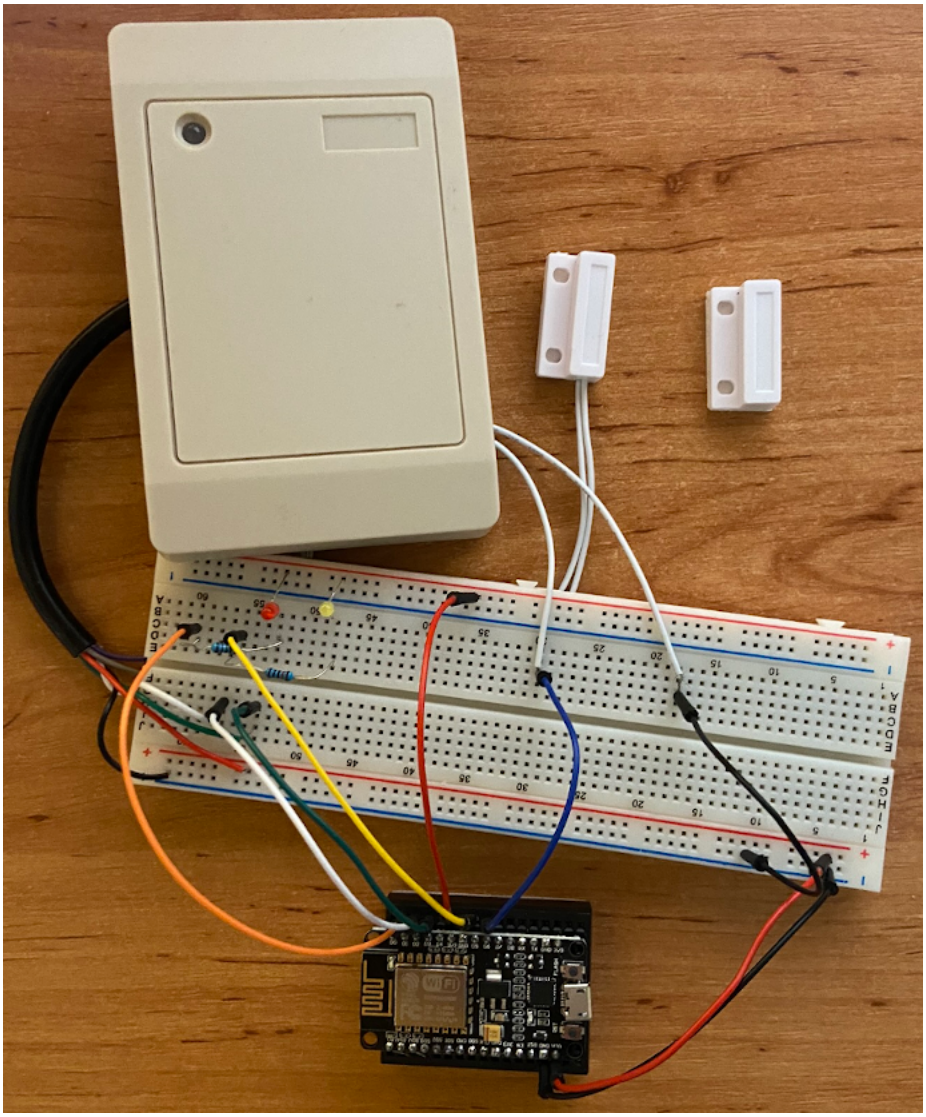
Výsledné zapojení je vidět na obrázku 4.2.



Obrázek 4.2: Zapojení pasivního infračerveného čidla (senzoru pohybu). Hnědý kabel je zdroj napětí, černý země a žlutý je datový.



Obrázek 4.3: Zapojení teplotního a vlhkostního senzoru DHT11. Červený kabel je zdroj napětí, šedý země a modrý kabel je datový.



Obrázek 4.4: Zapojení dveřního modulu. Červené kabely jsou zdroje napětí a černé země. Bílý a zelený kabel slouží ke komunikaci s RFID čtečkou. Oranžový a žlutý kabel jsou přes 1kΩ rezistory připojeny k diodám. Modrý kabel je připojen k magnetické závoře.

4.5 Webový server

Web jsem se rozhodl implementovat v jazyce PHP (bod 2.3.2) s použitím frameworku Nette (bod 2.4.2). Uživatelské rozhraní je plně angličtině, protože veškeré ostatní technické části práce jsou také v angličtině.

4.5.1 Struktura

V této části je popsána základní struktura souborů vytvořená pomocí frameworku Nette.

V kořenovém adresáři se nachází soubor *.htaccess* (je obsažen i v mnoha dalších adresářích), který například zakazuje vstup do některých adresářů, popřípadě umožňuje přepiso-

vání adres (odstranění přípon souborů apod.). Soubory *composer.json* a *composer.lock* jsou použity programem Composer [23] a slouží ke správě všech knihoven.

Adresář app

Tato složka obsahuje veškerou implementaci webové aplikace. Jsou v ní složky obsahující modely a prezentery (obsahuje složku pohledů), definici formulářů a konfigurační údaje webu. Názvy všech modelů, prezenterů, pohledů a s nimi souvisejících formulářů začínají názvem části, o kterou se starají (například ActionModel, ActionPresenter apod.). Kromě základních prezenterů pro zpracování chyb jsem vytvořil tyto:

- ActionPresenter - stará se o vytváření, editace a mazání akcí (bod 3.4.3)
- ConditionPresenter - implementuje podmínky viz. bod 3.4.2
- AppliancePresenter - editace zařízení
- DataPresenter - prohlížení dat ze zařízení
- DatumPresenter - editace údajů (z uživatelského pohledu nazváno Property)
- HandlerPresenter - správa funkcí implementujících příjem dat
- ModulePresenter - editace informací o modulech
- HomepagePresenter - připravený pro implementování grafů na domovské stránce

Adresář log

Obsahuje hlášení o chybách z *Tracy* a textové soubory s veškerými problémy, které nastaly při chodu serveru.

Adresář temp

Do tohoto adresáře se ukládají pohledy a předem načtená databázová struktura. Slouží ke zrychlení načítání webových stránek. Programátor může tuto složku použít jako dočasný úložný prostor apod. Mnohdy se mi stalo, že se mnou provedené změny graficky či funkčně neprojevily. Ve většině případů stačilo tuto složku promazat a nechat vše znovu vygenerovat.

Adresář vendor

Zde jsou uloženy všechny použité knihovny načtené pomocí programu Composer.

Adresář www

Tento adresář obsahuje obrázky (ať už statické, či v galerii), skripty (složka *js*), kaskádové styly (složka *css*) apod. Měl by být použit pro trvalé uložení souborů (například když chce uživatel nahrávat dokumenty na webové stránky, do galerie atd.).

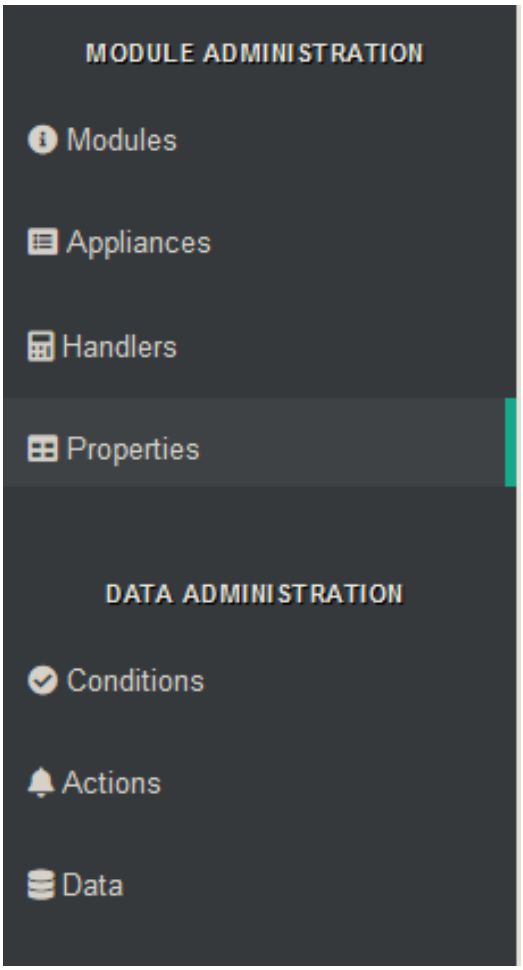
4.5.2 Vzhled

Pro vzhled webového rozhraní jsem záměrně vybral jednoduchý motiv, aby byla pozornost uživatele udržována na nejdůležitějších prvcích aplikace.

Na obrázku 4.5 je vidět výsledné menu aplikace, které je neměnné. Je rozděleno na dvě hlavní sekce – správu modulů (Module Administration) a práci s daty (Data Administration).

Část správy modulů obsahuje veškeré hodnoty, které jsou do databáze vkládány automaticky při konfiguraci.

- Modules – moduly (řídící jednotky)
- Appliances – zařízení (senzory a spotřebiče)
- Handlers – funkce, které zajišťují zpracování přijatých dat
- Properties – seznam údajů, které jsou zařízeními poskytovány



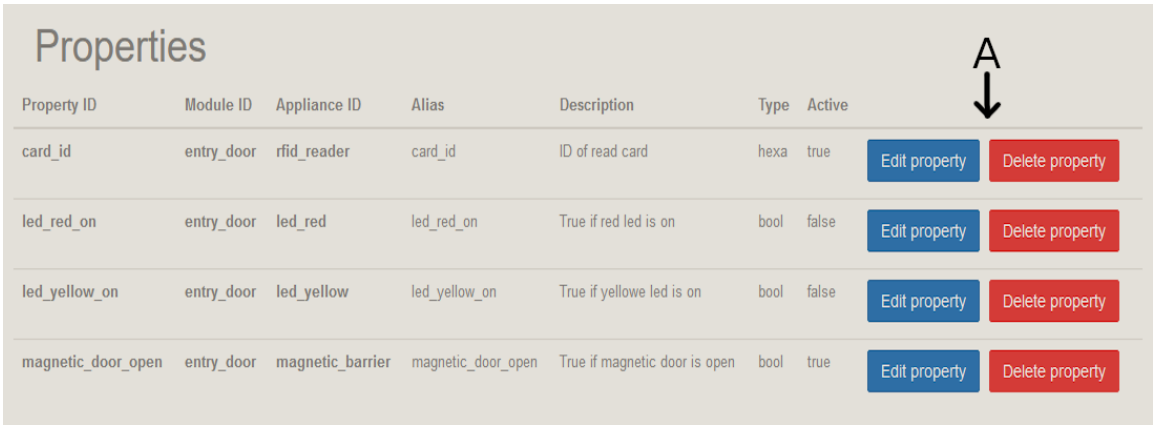
Obrázek 4.5: Menu ve webovém rozhraní. V horní části správa modulů, ve spodní části práce s daty.

Po rozkliknutí konkrétního prvku menu se uživateli v hlavní části webového rozhraní zobrazí tabulka, v níž budou zobrazeny všechny uživatelem požadované hodnoty dle selekce z menu (na obrázku 4.6 zobrazení všech údajů).

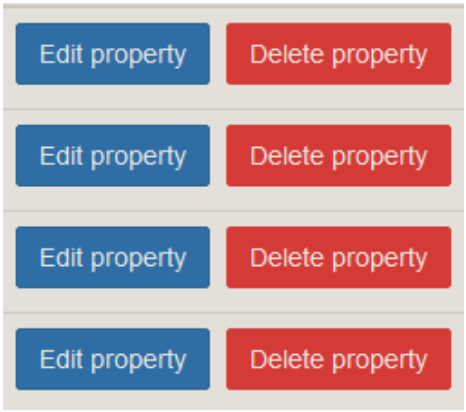
V případě, že v databázi nejsou žádné hodnoty, bude tabulka prázdná.

Většinu záznamů může uživatel mazat a editovat (kromě dat ze zařízení). Pokud se nejedná o akci nebo podmínku, lze upravovat pouze popisky a poznámky.

Nové záznamy může pomocí webové aplikace uživatel přidávat pouze k akcím a podmínkám. U názvu je vždy tlačítko pro přidání (viz. obrázek 4.9). Po rozkliknutí odkazu se uživateli otevře přidávací formulář (formuláře pro editace jsou v případě akcí a podmínek úplně stejné) – viz. obrázek 4.10.

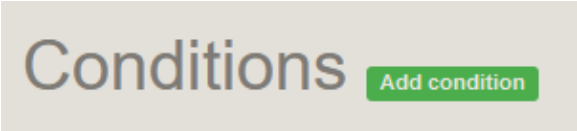


Obrázek 4.6: Tabulka ve webovém rozhraní. Nahoře se vždy nachází název zobrazené hodnoty. Po pravé straně (A) vidíme tlačítka pro správu (detail na obrázku 4.7).



Obrázek 4.7: Detail tlačítek pro správu.

Obrázek 4.8: Zobrazení podmínek. Vedle názvu hodnoty (B) tlačítko pro přidání nové (detail na obrázku 4.9).



Obrázek 4.9: Detail tlačítka pro přidávání nových hodnot.

4.6 Testování

4.6.1 Grafické rozhraní

Způsob ovládání grafického rozhraní jsem testoval na dvou osobách. Jedna pochází z informačního odvětví a dobře rozumí představenému problému. Druhá nemá vzdělání v této oblasti a ovládá počítač na úrovni pokročilých administrativních úkonů (textové a tabulkové editory apod.).

První uživatel neměl kromě nesrozumitelnosti anglického názvu údajů žádné větší výtky k prezentované aplikaci. Na základě této připomínky jsem se rozhodl anglický název změnit z *Datum* na *Property*.

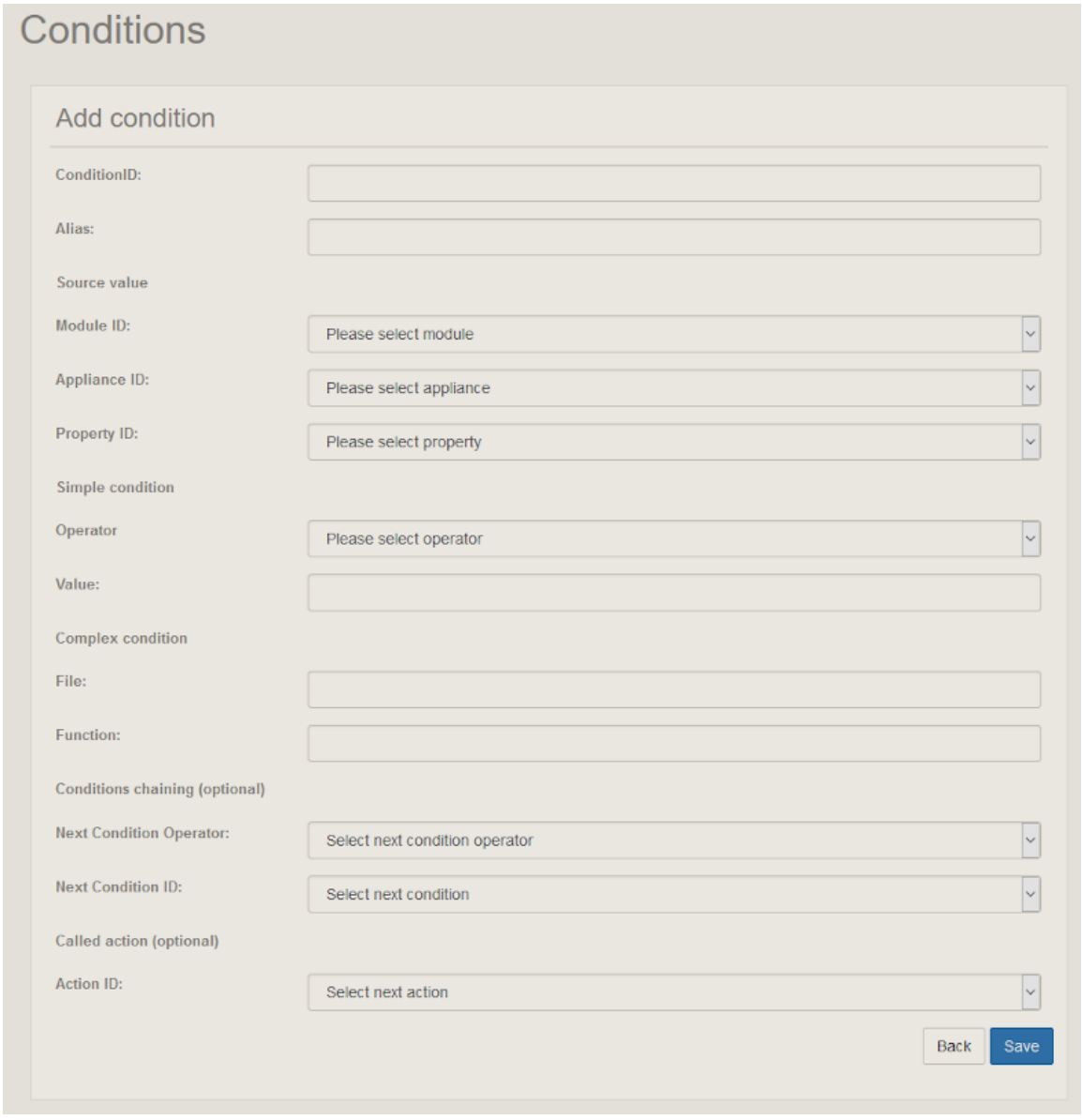
Na základě poznatků druhého uživatele jsem upravil velikosti jednotlivých tlačítek a přeskládal položky v menu.

4.6.2 Funkčnost systému

Vzhledem ke komplexnosti celého systému by bylo testování funkčnosti s pomocí uživatelů neefektivní. Rozhodl jsem se tedy napsat automatické testy, které se dají spustit na serveru.

V souboru *test_data.py* jsou funkce simulující konfiguraci řídící jednotky (musí být spuštěn hlavní skript).

Jakmile doběhne konfigurace systému, je nejprve vyzkoušena jednoduchá podmínka (kontrolována jsou výsledná data – příkaz, který se měl stát po splnění podmínky) s jednoduchou akcí a poté komplexní podmínka s komplexní akcí.



Obrázek 4.10: Formulář pro přidání a úpravu podmínek.

Kapitola 5

Závěr

Cílem této práce bylo vymyslet a implementovat systém pro domácí automatizaci založený na definování podmínek a k nim odpovídajících akcí pomocí Raspberry Pi a NodeMCU. Zpracování bylo doplněno o webové rozhraní k výčtu a editaci požadovaných dat. Obzvláště jsem se zaměřil na prvky, jejichž prostřednictvím bych byl schopen zvyšovat zabezpečení obytného objektu s užitím specializovaných modulů.

Práci jsem začal studiem existujících řešení a použitých technologií, dále zkoumáním jejich funkcionality a poznáváním jejich možností. Byl vytvořen návrh otevřeného systému jenž využívá centrální jednotku Raspberry Pi s jednotlivými řídícími jednotkami na platformě NodeMCU. Tento systém je plně modularizovatelný a dovoluje uživateli jeho přizpůsobení dle specializovaných požadavků. Následně jsem fyzicky zapojil a naprogramoval část navržených modulů, které jsem poté i prakticky testoval.

Dílčí součástí bylo vytvoření webového rozhraní, které umožňuje procházet nejen zaznamenaná data ze senzorů, ale také přidané moduly, vytvořené podmínky a akce.

Práci by nebylo špatné rozšířit napojením na službu PushBullet či Blynk, poskytnout uživateli API a propojit tyto služby s prací bez narušení chodu systému. Následně bych výrazně zjednodušil a zpřístupnil celý systém i technologicky méně zdatnému uživateli pomocí webového rozhraní, přes které by se přímo definovaly moduly i zařízení. Kód komplexních podmínek a akcí by uživatel nemusel ručně vkládat na server, ale použil by přehledný editor přímo na webu.

Výsledkem práce je rozšiřitelný server schopný zpracovávat jakékoliv podmínky a provádět libovolné uživatelem naprogramované akce, komunikační protokol založený na vyměňování zpráv pomocí technologie MQTT a sada základních modulů sloužících ke zvýšení bezpečnosti domácnosti. Prohloubil jsem své znalosti v oblasti internetu věcí, zapojování logických obvodů a získal jsem povědomí o zařízeních sloužících k automatizaci.

Literatura

[1] *Alexa* [online]. [cit. 2020-05-14]. Dostupné z: https://developer.amazon.com/en-US/alexa/connected-devices/compatible.

[2] *Eclipse Paho^{TM} MQTT Python Client* [online]. [cit. 2020-05-07]. Dostupné z: https://github.com/eclipse/paho.mqtt.python.

[3] *HomeKit* [online]. [cit. 2020-05-14]. Dostupné z: https://en.wikipedia.org/wiki/HomeKit.

[4] *NodeMCU DEVKIT V1.0* [online]. 2015 [cit. 2020-05-25]. Dostupné z: https://github.com/nodemcu/nodemcu-devkit-v1.0.

[5] *Pulzně šířková modulace* [online]. 2018 [cit. 2020-05-24]. Dostupné z: https://cs.wikipedia.org/wiki/Pulzn%C4%9B_%C5%A1%C3%AD%C5%99kov%C3%A1_modulace.

[6] *Infračervené záření* [online]. 2019 [cit. 2020-05-25]. Dostupné z: https://cs.wikipedia.org/wiki/Infra%C4%8Derven%C3%A9_z%C3%A1%C5%99en%C3%AD.

[7] *JavaScript Object Notation* [online]. 2020 [cit. 2020-05-24]. Dostupné z: https://cs.wikipedia.org/wiki/JavaScript_Object_Notation.

[8] *Model–view–presenter* [online]. 2020 [cit. 2020-05-25]. Dostupné z: https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93presenter.

[9] *Tim Berners-Lee* [online]. 2020 [cit. 2020-05-24]. Dostupné z: https://cs.wikipedia.org/wiki/Tim_Berners-Lee.

[10] Adafruit. *DHT sensor library* [online]. 2020 [cit. 2020-05-25]. Dostupné z: https://github.com/adafruit/DHT-sensor-library.

[11] Corporation, O. *MySQL* [online]. 2020 [cit. 2020-05-25]. Dostupné z: https://www.mysql.com/.

[12] Costa, P. *Yet Another Arduino Wiegand Library* [online]. [cit. 2020-05-17]. Dostupné z: https://github.com/paulo-raca/YetAnotherArduinoWiegandLibrary.

[13] Electronics, O. *DHT11 Humidity & Temperature Sensor* [online]. [cit. 2020-05-24]. Dostupné z: https://www.mouser.com/datasheet/2/758/DHT11-Technical-Data-Sheet-Translated-Version-1143054.pdf.

[14] Foundation, N. *Latte* [online]. 2008 [cit. 2020-05-25]. Dostupné z: https://latte.nette.org/cs/.

[15] Foundation, N. *Nette* [online]. 2008 [cit. 2020-05-25]. Dostupné z: https://nette.org/cs/.

[16] Foundation, N. *Tracy* [online]. 2008 [cit. 2020-05-25]. Dostupné z: https://tracy.nette.org/cs/.

[17] Foundation, P. S. *Python* [online]. 2001 [cit. 2020-05-24]. Dostupné z: https://www.python.org/.

[18] Foundation, R. P. *Raspberry Pi* [online]. [cit. 2020-05-24]. Dostupné z: https://www.raspberrypi.org/.

[19] Group, T. P. *PHP* [online]. 2001 [cit. 2020-05-24]. Dostupné z: https://www.php.net/.

[20] Inc., B. *Blynk* [online]. 2019 [cit. 2020-05-26]. Dostupné z: http://mqtt.org/.

[21] Kubát, Z. *Čipový přístupový systém*. Praha 2 - Nové Město, CZ, 2013. Maturitní práce. Střední průmyslová škola elektrotechnická. Dostupné z: http://stretech.fs.cvut.cz/2013/sbornik_2013/84.pdf.

[22] Lua.org. *Lua* [online]. 2020 [cit. 2020-05-25]. Dostupné z: https://www.lua.org/.

[23] Nils Adermann, J. B. a contributions many community. *Composer - A Dependency Manager for PHP* [online]. 2020 [cit. 2020-05-28]. Dostupné z: https://getcomposer.org/.

[24] OASIS. *MQTT* [online]. 2009 [cit. 2020-05-25]. Dostupné z: https://blynk.io/.

[25] Team, N. *NodeMcu* [online]. [cit. 2020-05-24]. Dostupné z: https://www.nodemcu.com/index_en.html.

[26] W3C. *HTML* [online]. 2019 [cit. 2020-05-24]. Dostupné z: https://www.w3.org/html/.

[27] W3C. *Cascading Style Sheets* [online]. 2020 [cit. 2020-05-25]. Dostupné z: https://www.w3.org/Style/CSS/.

Příloha A

Plakát

https://blog.kevineikenberry.com/leadership-supervisory-skills/the-door-is-just-a-metaphor/
https://pixabay.com/photos/stopwatch-timer-clock-symbol-icon-2624277/
https://www.clipart.email/download/10249244.html

Příloha B

Obsah přiloženého paměťového média

- **modules** – implementované moduly
  - **dht11** – teplotní senzor
  - **doorModule** – modul u dveří
  - **motionSensor** – pohybový senzor
- **RPi** – skript pro zpracování podmínek a akcí
- **web** – webové rozhraní aplikace
- **plakát**
- **README** – stučný návod pro spuštění