VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
DEPARTMENT OF INTELLIGENT SYSTEMS

IOT S REKONFIGUROVATELNÝMI UZLY V PYTHONU

IOT WITH RECONFIGURABLE NODES IN PYTHON

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE ERIK PITKO
AUTHOR

VEDOUCÍ PRÁCE doc. Ing. VLADIMÍR JANOUŠEK, Ph.D.
SUPERVISOR

BRNO 2019

Ústav inteligentních systémů (UITS) Akademický rok 2018/2019

Zadání bakalářské práce

21600

Student: **Pitko Erik**
Program: Informační technologie
Název: **IoT s rekonfigurovatelnými uzly v Pythonu**
IoT with Reconfigurable Nodes in Python
Kategorie: Operační systémy
Zadání:

1. Seznamte se s existujícími hardwarovými platformami na bázi ESP8266/ESP32 a s přístupy k tvorbě software pro tyto systémy, zaměřte se na možnost programovat tyto plaformy v jazyce Python.
2. Seznamte se s existující pokusnou implementací OS pro rekonfigurovatelný IoT uzel v jazyce MicroPython na platformě ESP32, komunikující protokolem MQTT.
3. Navrhněte a realizujte další potřebné komponenty pro realizaci IoT systémů, tj. databázi, uživatelské rozhraní a nástroje pro monitorování a správu systému.
4. Proveďte testování za pomoci vhodné aplikace a vyhodnoťte dosažené výsledky.

Literatura:

- Drahovský, P.: Rekonfigurovatelný IoT uzel na bázi ESP8266/ESP32. Bakalářská práce FIT VUT v Brně. URL: https://wis.fit.vutbr.cz/FIT/db/dir.php/rp/2017/BP/20280.pdf

Pro udělení zápočtu za první semestr je požadováno:

- Body 1, 2 a část návrhu.

Podrobné závazné pokyny pro vypracování práce viz http://www.fit.vutbr.cz/info/szz/

Vedoucí práce: **Janoušek Vladimír, doc. Ing., Ph.D.**
Vedoucí ústavu: Hanáček Petr, doc. Dr. Ing.
Datum zadání: 1. listopadu 2018
Datum odevzdání: 15. května 2019
Datum schválení: 1. listopadu 2018

Abstrakt

Cieľom tejto práce je vytvoriť systém pre správu a monitorovanie rekonfigurovateľných ESP32 uzlov, jeho následná realizácia a implementácia aplikácie pre monitorovanie dát z jednotlivých senzorov. Systém umožňuje nahrávať, spúšťať a monitorovať aplikácie napísané v jazyku MicroPython na uzloch. Tieto uzly môžu komunikovať s ostatnými aplikáciami, spustenými na akomkoľvek uzle v rámci danej siete, alebo v rámci jedného MQTT brokeru.

Abstract

The main goal of this thesis is creation and implementation of monitoring software for reconfigurable ESP32 nodes equipped with the existing operating system MPOS. This system is then used to solve specific task. The monitoring system is capable of loading, activating and maintaining applications written in MicroPython language that is running on these nodes. Nodes can communicate with other applications using the MQTT broker.

Kľúčové slová

Internet vecí, ESP32, MicroPython, Android, PCB, MQTT, Android, Node.js, Pug, mobilná aplikácia, Web, REST, MPOS

Keywords

Internet of Things, ESP32, MicroPython, Android, PCB, MQTT, Android, Node.js, Pug, mobile application, Web, REST, MPOS

Citácia

PITKO, Erik. *IoT s rekonfigurovatelnými uzly v Pythonu.* Brno, 2019. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce doc. Ing. Vladimír Janoušek, Ph.D.

IoT s rekonfigurovatelnými uzly v Pythonu

Prehlásenie

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne pod odborným vedením doc. Ing. Vladimíra Janouška, Ph.D. a s použitím odbornej literatúry a ďalších informačných zdrojov, ktoré sú všetky citované v práci a uvedené v zozname literatúry na konci práce.

. . . . . .
Erik Pitko
9. mája 2019

Poďakovanie

Touto cestou by som chcel poďakovať doc. Ing. Vladimírovi Janouškovi, Ph.D. za odbornú pomoc a cenné rady pri vedení tejto bakalárskej práce.

Obsah

1 Úvod **2**

2 Distribuované riadiace systémy **3**
2.1 Mikrokontroléry a mikroprocesory 3
2.2 Internet vecí 4
2.3 Automatizácia domácnosti 4

3 Možnosti komunikácie IoT zariadení **5**
3.1 Wifi 5
3.2 Bluetooth 6
3.3 LoRa 6
3.4 ZigBee 7
3.5 Analýza senzorov a integrovaných obvodov 7
3.6 Komunikačné protokoly 14

4 Dynamicky rekonfigurovateľný operačný systém **16**
4.1 PNOS 16
4.2 MPOS 16
4.3 Hardvérové platformy pre systém MPOS 17

5 Prípad použitia a návrh systému **19**
5.1 Prípad použitia spravovaného distribuovaného riadiaceho systému 19
5.2 Návrh správcovského systému pre MPOS 22
5.3 Návrh mobilnej aplikácie 22

6 Implementácia a testovanie **26**
6.1 Modifikácia MPOS 26
6.2 Zariadenia pre demoaplikáciu 28
6.3 Správcovský systém 30
6.4 Android klientská aplikácia 35
6.5 Testovanie demoaplikácie 42

7 Záver **45**

Literatúra **46**

A Zdrojové kódy použité v moduloch **50**

B Schéma zapojenia **59**

Kapitola 1

Úvod

V posledných pár rokoch sa venuje malým vstavaným mikrokontrolérom mimoriadna pozornosť a vďaka rapídnemu vývoju týchto platforiem prenikajú do nášho každodenného života čoraz viac. Buď v podobe malej uzatvorenej krabičky, ktorá ovláda rôzne časti domácnosti, automatizovaného „smart“ mesta s parkovacími asistentmi, alebo plne automatizovanej výrobnej linky, pohodlne ovládanej cez aplikáciu z mobilného zariadenia.

Moduly ESP8266/ESP32 sú výsledkom vývoja od prvého vydania čipu v roku 2014 od firmy Ai-Thinker (ESP-01) so schopnosťou Wi-Fi komunikácie a jednoduchých TCP/IP spojení. Vďaka nízkemu počtu externých komponentov sú tieto moduly veľmi lacné na výrobu, čo umožňuje široké nasadenie v rôznych „low-cost“ projektoch a riešeniach.

Preto sa rýchlo stali najpopulárnejšou možnosťou pre hobbistov a vývojárov. Napriek ich relatívne nízkemu výkonu zvládajú spracovanie interpretovaného jazyka MicroPython.

Cieľom tejto práce je vytvoriť komplexný systém umožňujúci správu, monitorovanie rekonfigurovateľných ESP32 uzlov a implementáciu jednoduchej aplikácie na monitorovanie dát v tomto systéme. Na týchto uzloch je nainštalovaný upravený operačný systém napísaný v jazyku MicroPython. Tento systém umožňuje spúšťať tzv. mikro aplikácie, ktoré fungujú ako samostatné celky systému a realizujú jednoduchú komunikáciu medzi implementovanými modulmi.

Kapitola 2

Distribuované riadiace systémy

Distribuovaný riadiaci systém je zložený z niekoľkých podsystémov, ktoré sú medzi sebou prepojené pomocou komunikačnej zbernice a ktoré sa spoločne podieľajú na riadení. Tento prístup sa začal v automatizácií systémov rýchlo rozvíjať a nasadzovať v čase, keď vznikli výkonné a spoľahlivé komunikačné zbernice, ktoré bolo možné k tomuto účelu využiť. Cena jednotlivých zariadení s procesorom postupne klesla na úroveň, kedy nebol výrazný rozdiel medzi použitím jedného výkonného prvku, alebo niekoľkých menších prvkov.[15]

Medzi hlavné výhody distribuovaných systémov oproti centralizovaným patrí:

- Jednoduchšie odlaďovanie novej aplikácie (vďaka tomu, že aplikácia ako celok je rozdelená do podsystémov a môže byť ladená po častiach).
- Postupné rozširovanie (riadiaci systém je možné postupne rozširovať pridávaním ďalších zariadení).
- Vyššia dostupnosť (pri poruche jedného zariadenia môže riadiaci systém pracovať ďalej).

2.1 Mikrokontroléry a mikroprocesory

Mikroprocesor zahŕňa iba centrálnu výpočtovú jednotku(CPU). Všetky funkčné bloky, ako napríklad pamäť a iné periférie, musia byť pripojené k mikroprocesoru externe. Spolu tak formujú mikroprocesorovú dosku. Vo väčšine prípadov sú požiadavky pre vytvorenie zariadenia, ktoré obsahuje všetky funkčné bloky na jednom integrovanom čipe. Takýto systém sa nazýva „Single-Board Microcomputer“, teda v preklade jednočipový mikropočítač. Jednočipový mikropočítač sa inak nazýva aj mikrokontrolér. Príkladom takýchto mikrokontrolérov sú: rôzne Arduino dosky, Intel MCS-51, Atmel 89CXX, STM32, ...

Mikrokontrolér sa skladá z mikroprocesora a aspoň jedného z nasledujúcich komponentov:

- Pamäť
- Digitálno-analógový prevodník (D/A)
- Analógovo-digitálny prevodník (A/D)
- Paralelné vstupno-výstupné rozhranie

- Sériové vstupno-výstupné rozhranie
- Časovače a počítadlá

Takéto mikrokontoléry sú známe ako „general-purpose“, čo znamená, že sú určené pre naprogramovanie na rôzne účely. Využitie takéhoto zariadenia je teda závislé na našom konkrétnom použití. Napríklad ovládanie krokového motorčeka, alebo slúžiaci ako jednoduchý teplomer.[3][39]

2.2 Internet vecí

Prepojenie veľkého množstva zariadení a ich vzájomná komunikácia dostala názov Internet Vecí (Internet of Things - IoT). Jedná sa o výpočtový koncept, ktorý popisuje prepojenie bezdrôtových zariadení, ktoré sa vedia identifikovať a komunikovať medzi ostatnými zariadeniami za účelom monitorovania, úspory energie, alebo zvýšenia bezpečnosti domácností a podnikov. Tento názov sa využíva hlavne v súvislosti s pojmami ako „Inteligentná domácnosť“ či „Inteligentné mesto“[37]

2.3 Automatizácia domácnosti

Automatizácia domácnosti využíva koncept IoT a umožňuje jednotlivým užívateľom monitorovať a ovládať rôzne typy elektrických zariadení. Najčastejší typ automatizácie domácnosti umožňuje ovládanie svetla či kúrenia na základe informácií o množstve prítomného svetla resp. teploty v miestnosti.[22]

Takáto automatizácia prináša mnoho výhod, napríklad:

- Úspora elektrickej energie (vďaka monitorovaniu stavu miestnosti môže systém zabrániť potencionálnemu plytvaniu energie, napríklad v miestnosti, ktorá je menej využívaná, alebo počas doby, kedy užívateľ nie je v domácnosti)
- Bezpečnosť (kamery môžu byť tiež súčasťou domácej automatizácie a môžu varovať užívateľa o nezvyčajnom pohybe v domácnosti)
- Pohodlie pre užívateľa (úprava osvetlenia pri pozeraní filmu, alebo zmena digitálnych obrázkov na stenách, podľa toho, kto sa momentálne nachádza v miestnosti)

Kapitola 3

Možnosti komunikácie IoT zariadení

Každý IoT projekt má svoje jedinečné požiadavky pre komunikáciu. Medzi základne požiadavky patrí bezdrôtovosť, vzdialenosť na ktorú dokážu zariadenia komunikovať, rýchlosť prenášaných dát a spotreba elektrickej energie. V tejto kapitole sú rozobraté rôzne najpoužívanejšie možnosti komunikácie ich rozdiely, výhody, nevýhody, typické príklady použitia a rôzne protokoly využívané pri komunikácii. Tieto sú využívané pri posielaní dát z mikrokontrolérov, alebo medzi senzormi a mikrokontrolérmi).[2][17]

3.1 Wifi

Vďaka širokému pokrytiu WiFi sietí a vysokej dostupnosti patrí tento spôsob komunikácie medzi najrozšírenejšie a ekonomicky najvýhodnejšie spôsoby komunikácie. Kvôli svojím rôznym štandardom ponúka decentnú vzdialenosť pripojenia, zatiaľ čo nestráca na rýchlosti komunikácie.

Najrozšírenejší štandard, používaný pre WiFi siete je 802.11n, ktorý ponúka teoretickú rýchlosť až do 600Mbps so vzdialenosťou približne do 50m za cenu vysokej spotreby.

Veľkou nevýhodou takejto komunikácie je spotreba, ktorú toto pripojenie vyžaduje počas komunikácie. Ďalšou nevýhodou WiFi komunikácie je bezpečnosť. Tieto siete sú väčšinou terčom rôznych útokov, preto musia byť dôkladne zabezpečené a sledované.

Najčastejšie sa tento typ komunikácie používa v prípadoch, kde sa nevyžaduje veľká vzdialenosť a je dostupné stabilné napájanie. Napríklad v domácnostiach, skladoch, alebo v iných budovách, kde je jednoducho dostupná už predom vytvorená infraštruktúra.[31]

HaLow

WiFi HaLow je nový štandard, ktorý vznikol na základe 802.11ah a bol špecificky vyvinutý pre IoT zariadenia. Ponúka vyššiu vzdialenosť pri komunikácii a zároveň nižšiu spotrebu. Tento nový štandard si vyžaduje novú infraštruktúru a špecializované zariadenia pre komunikáciu.

3.2 Bluetooth

Bluetooth je bezdrôtová technológia, ktorá po rokoch stagnácie zaznamenala vysoký rozmach. Táto technológia spája zariadenia väčšinou cez mástra v podobe napríklad chytrých telefónov a má dosah približne 5 až 10m. Zohráva veľmi významnú rolu pri komunikácií na krátke vzdialenosti. Táto technológia sa stala štandardom hlavne pri tzv. „nositeľných“ produktoch, ako sú napríklad chytré hodinky, náramky, prstene, ...

Bluetooth Low-Energy - BLE

Je technológia, ktorá ponúka podobnú vzdialenosť, no vyžaduje oveľa nižšiu spotrebu. Spotrebu má tak nízku, že zariadenia, ktoré túto technológiu využívajú, by mali na baterku vydržať aj niekoľko rokov. BLE ale nebolo navrhnuté pre posielanie veľkého množstva dát, ale skôr pre malú občasnú komunikáciu. Preto sa skôr hodí pre nositeľné produkty, alebo senzory do automobilu.[4]

Bluetooth 5

Bluetooth 5 je nový štandard, ktorý oproti 4 ponúka dvojnásobne vyššiu rýchlosť (približne 2Mbps), až štvornásobne väčšiu vzdialenosť pri komunikácií a nižšiu odozvu. Zatiaľ čo prináša množstvo výhod, tak energetická náročnosť vôbec nestúpla.

Namiesto jednej dátovej rýchlosti sa dá vybrať medzi niekoľkými možnosťami:

- 2Mbps
- 1Mbps
- 500kbps
- 125kbps

Čím nižšiu rýchlosť komunikácie využívame, tým väčšiu vzdialenosť vieme využiť.[14]

3.3 LoRa

Slovo LoRa pochádza z anglického „Long Range“, čo v preklade znamená „Ďaleký dosah“. LoRa patrí medzi jednu z najspoľahlivejších možností obojsmernej bezdrôtovej komunikácie, ktorá vďaka svojej nízkej frekvencii (868MHz v Európe) umožňuje prenos malého množstva dát na veľmi veľké vzdialenosti. V prípade priaznivého terénu môže ľahko presiahnuť vzdialenosť 10km, pričom priechodnosť cez stavby je relatívne priaznivá. Pri využití stožiarových antén je možný prenos dát aj cez 100km pri veľmi nízkej rýchlosti prenosu dát.

Táto technológia sa skladá z dvoch častí. Fyzickej vrstvy nazývanej LoRa a časti, ktorá reprezentuje vyššie vrstvy komunikácie - LoRaWAN.[36].

LoRaWAN

LoRaWAN špecifikuje komunikačný protokol LPWAN a sieťovú architektúru systému pre účely IoT. Názov LPWAN pochádza z anglického „Low Power Wide Area Network“, čo v preklade znamená „Nízko energetická rozsiahla sieť“.

Táto sieť(LoRaWAN) využíva komunikačný systém zariadení: gateway - server. Gateway komunikuje so serverom prostredníctvom štandardného IP pripojenia. Koncové zariadenia používajú bezdrôtovú komunikáciu s gateway. Komunikácia koncových zariadení je štandardne obojsmerná. Kvôli bezpečnosti sa používa niekoľko kódovaní dát. Zabezpečenie na sieťovej a aplikačnej úrovni a zabezpečenie na samotnom zariadení. Koncový bod má pridelený unikátny 64 bitový identifikátor a dva 128 bitové kľúče, s ktorými sú dáta šifrované pomocou pokročilého šifrovacieho štandardu - AES.

Táto technológia sa výborne hodí pre geografickú lokáciu a pre priemyselné využitie. Jej veľkou výhodou je to, že nevyžaduje stálu komunikáciu a teda umožňuje spojenie s veľkým množstvom zariadení cez veľké vzdialenosti s využitím nízkej spotreby - čo je vhodné pre IoT zariadenia. Na rozdiel od iných pripojení neumožňuje prenos Real-Time údajov, čo znamená, že pri prenose treba počítať s vysokým oneskorením. Zároveň prebieha vo voľnom pásme, čo môže spôsobovať problémy pri nedodržiavaní regulácie.

3.4 ZigBee

ZigBee je technológia, ktorá nám ponúka jednoduché a energeticky nenáročné spojenie s dosahom od 10 do 100m a s rýchlosťou približne 250kbps. Pre komunikáciu sa využívajú dva typy zariadení. Plne funkčné zariadenie (FFD) a zariadenie s redukovanou funkcionalitou (RFD). FFD môže komunikovať s RFD a inými FFD zariadeniami, zatiaľ čo RFD vie zaslať iba krátke správy a komunikovať iba s jedným FFD súčasne. Preto RFD zariadenie vieme vytvoriť s minimálnymi požiadavkami na výkon a pamäť. Každé FFD zariadenie si vie vytvoriť svoju vlastnú sieť (PAN). Všetky tieto siete vedia komunikovať nezávisle na sebe vďaka určeniu jedinečného identifikátora pre danú sieť.

Porovnanie

	WiFi	Bluetooth 5	Bluetooth LE	ZigBee	LoRa
Frekvencia	2.4GHz, 5GHz	2.4GHz	2.4GHz	868MHz, 915MHz	<1GHz
Rýchlosť prenosu	100Mbps	1-2Mbps	305kbps	250kbps	0.25-50kbps
Dosah	∼100m	∼40m	∼20m	10-300m	5-21km
Spotreba energie	∼116mA(@ 1.8V)	∼20mA	∼2.5mA	∼40mA	0.3uA - 160mA

Tabuľka 3.1: Porovnanie najbežnejších bezdrôtových technológií[17][32][36]

3.5 Analýza senzorov a integrovaných obvodov

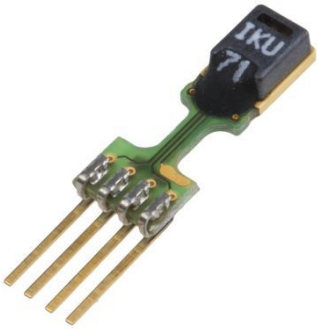
Pri výbere senzorov, ktoré budú použité pri navrhnutom systéme treba brať ohľad na podmienky, v akých budú vystavené, ich cenu a presnosť merania. Pri nasadení senzorov do vonkajšieho prostredia sa musí počítať s veľmi nepriaznivými podmienkami, na ktoré väčšina senzorov nie je uspôsobená.

Výber taktiež záleží na dostupnosti, cene a možnosti integrovania do systému. Pre lepšie porovnanie dostupných senzorov slúži nasledujúci popis a ich zhrnutie.

SHT7X

SHT71 a SHT75 sú senzory schopné merať teplotu a vlhkosť ovzdušia. Prenášajú dáta pomocou sériovej zbernice s využitím integrovaných 14 bitových A/D prevodníkov. Tým pádom sú schopné presného a rýchleho merania, odolného voči externým vplyvom a odstraňuje to nutnosť kalibrácie senzorov. To umožňuje ich jednoduchú výmenu v prípade poruchy.

Tieto senzory sú schopné presného merania do 3% relatívnej vlhkosti(RH) (SHT75 do 1.8% RH) a zvládnutia nepriaznivých podmienok ($-40°C$ - $123.8°C$) - sú teda vhodné pre použitie v extrémnych podmienkach.



Obr. 3.1: Senzor SHT71

AM2315

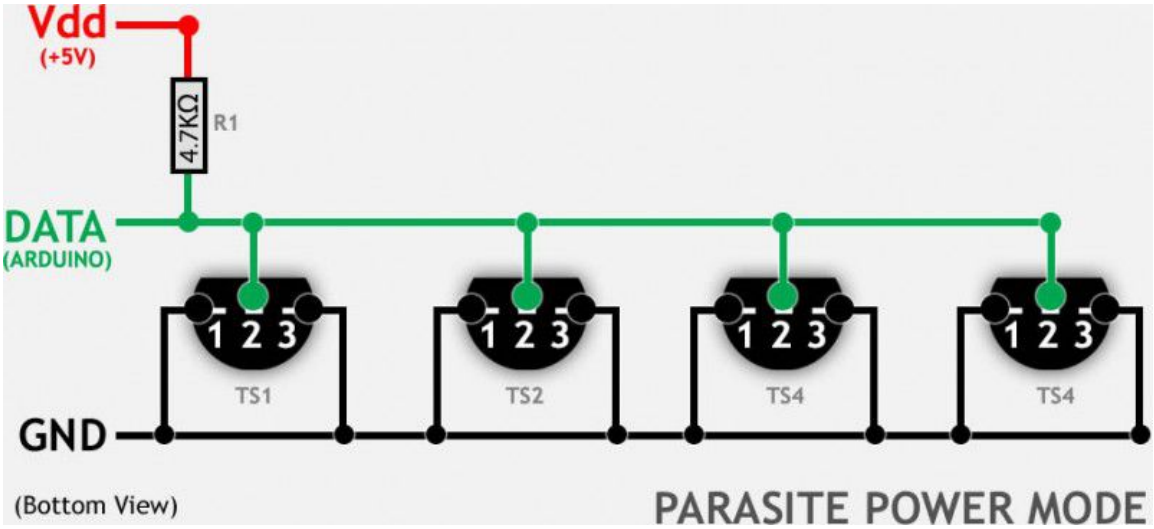
AM2315 je senzor veľmi podobný SHT75. Na rozdiel od SHT75 pracuje s rozhraním I2C a jeho adresu nie je možné zmeniť. To znamená, že na jednu I2C zbernicu nie je možné napojiť viacero týchto senzorov. Tento senzor sa dodáva uschovaný v ochrannom kryte, čo umožňuje vystavenie vonkajšiemu prostrediu.

Obr. 3.2: Senzor AM2315 v ochrannom púzdre

DS18B20

Vďaka jeho vysokej dostupnosti a nízkej cene je tento teplotný senzor jeden z najpoužívanejších teplotných čidiel. Aj napriek nízkej cene umožňuje merať teplotu v rozmedzí od $-55°C$ až po $125°C$, pričom v rozsahu $-10°C$ až $85°C$ má garantovanú presnosť $\pm 0.5°C$. Tento senzor sa dá zakúpiť v TO-92 púzdre, v ktorom pripomína obyčajný tranzistor, alebo jeho vodotesnú verziu zatavenú v nerezovej tyčinke. Pri komunikácii s týmto senzorom je využitá OneWire zbernica, ktorá pre komunikáciu vyžaduje iba jeden komunikačný pin. Taktiež sa dá využiť tzv. parazitný režim, pri ktorom stačí využiť iba dva káble. Keďže má každý tento senzor pridelené svoje vlastné 64 bitové identifikačné číslo, môžeme na jednu OneWire zbernicu napojiť veľký počet týchto senzorov.[20]

Obr. 3.3: Senzor DS18B20 v TO-92 púzdre(vľavo) a jeho vodotesná verzia(vpravo)



Obr. 3.4: Štyri senzory DS18B20 zapojené v režime parazitného módu [38]

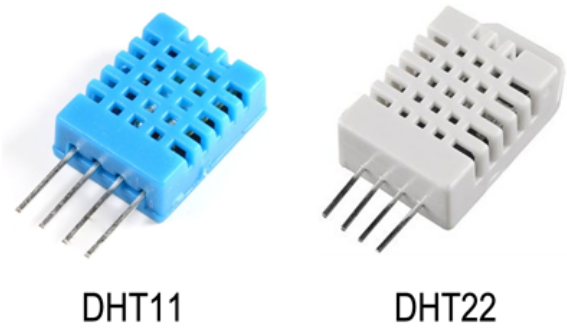
DHT11 / DHT22

Senzor DHT11 a DHT22 využívajú pre komunikáciu iba jeden dátový vodič. Napájaný môže byť v rozmedzí 3.3 až 6V. Jedná sa o jedny z najlacnejších senzorov, ktoré umožňujú meranie vlhkosti. Meranie sa nedá ovládať zo strany mástra. Senzor obnoví hodnotu približne každé dve sekundy a tú vieme následne vyčítať. Malou nevýhodou je, že na jeden vodič sa dá pripojiť iba jeden takýto senzor.

Lacnejšia varianta DHT11 meria teplotu iba v rozsahu 0 až 50°C s presnosťou až $\pm 1°C$ a vlhkosť v rozmedzí 20 až 90% s presnosťou $\pm 4\%$.

Pokročilejší modul DHT22 prináša lepší rozsah a presnosť merania, no stále dosahuje presnosť merania relatívnej vlhkosti $\pm 2\%$, v prípade, že vlhkosť stúpne nad 90% dosahuje presnosť iba $\pm 5\%$.

Napriek všetkým svojím nevýhodám je tento senzor veľmi obľúbený vďaka svojej nízkej cene a vysokej dostupnosti. V prípade, že nepotrebujeme vysokú presnosť ide o vhodnú alternatívu.



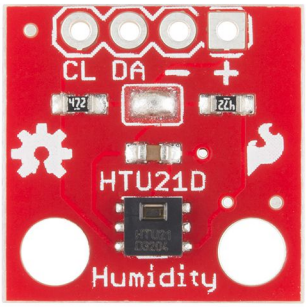
Obr. 3.5: Senzory DHT11 a DHT22[21]

HTU21D

HTU21D je senzor pre meranie teploty a vlhkosti. Jeho veľkou výhodou je dobrá presnosť nameraných údajov a malá veľkosť samotného modulu. Najpresnejšie výsledky dosahuje v teplote od 5 až do 60°C a vlhkosti 10 až 90%. Pre napájanie a komunikáciu je možné využiť napájacie napätie v rozsahu od 3.3 až do 5V, čo nám umožňuje bezproblémovú komunikáciu s mikrokontrolérmi, ktoré využívajú pre komunikáciu iba 3.3, alebo 5V.[19]

Keďže je tento senzor veľmi populárny, sú vytvorené knižnice pre jednoduchú komunikáciu a získavanie dát. Vďaka jeho malej spotrebe (max. 0.5mA pri meraní) je tiež vhodný pre napájanie z batérií.

Veľká nevýhodou tohto senzora je jeho nízka odolnosť voči okolitému prostrediu. Keďže tento senzor nemá žiadnu ochranu, nevydrží dlhodobú vysokú vlhkosť, teda nie je vhodný pre využitie vo vlhkých/vonkajších priestoroch.



Obr. 3.6: Senzor HTU21D s vyvedenými pinmi pre napájanie a komunikáciu pomocou I2C zbernice

	SHT7X	**AM2315**	**DS18B20**	**DHT22**	**HTU21D**
Senzor	Teplota + Vlhkosť	Teplota + Vlhkosť	Teplota	Teplota + Vlhkosť	Teplota + Vlhkosť
Rozhranie	TwoWire	I2C	OneWire		I2C
Teplotný rozsah ($°C$)	od -40 do 123.8	od -40 do 125	od -55 do 125	od -40 do 80	od -40 do 125
Presnosť ($°C$)	$\pm 0.3 - 0.4°C$	$\pm 0.1°C$	$\pm 0.5°C$	$\pm 0.5°C$	$\pm 0.3°C$
Presnosť (% RH)	1.8 - 3	2	-	2 5 (RH<10, RH>90)	3
Rozlíšenie teploty (bit)	12-14	16	9-12	16	11-14
Rozlíšenie vlhkosti (bit)	8-12	16	-	16	8-12
Cena ($)	22.35 - 26.01	29.95	3	9.95	14.95

Tabuľka 3.2: Porovnanie senzorov[35]
[25]

Relé

Princíp fungovania relé je pomerne jednoduchý. Prepínací obvod sa skladá z cievky navinutej na jadre feromagnetického materiálu. Obvod obsahuje kotvu a pružinu. Kotva sa vďaka pružine v kľudovom stave nedotýka vodivej plochy. Ku kontaktu s vodivou plochou dochádza v momente, keď je cievka napojená na elektrický zdroj. Prúd potom v cievke vyvolá magnetický tok. V tomto momente vzniká príťažlivá magnetická sila, ktorá premôže pružinu a pritlačí kotvu ku kontaktu.

Existuje viacero druhov modulov relé. Väčšinou ide minimálne o galvanické oddelenie relé od mikrokontroléra optočlenom. Niektoré moduly dávajú možnosť viacerých kanálov. V princípe nám iba pribudne ďalší ovládací pin a ďalšie relé osadené na doske modulu. V takom prípade sme schopní spínať dve záťaže nezávisle na sebe.[11]

Elektromagnetický ventil

Elektromagnetický ventil je elektricky aktivovaný ventil, ktorý sa obyčajne používa k riadeniu toku, alebo smeru toku kvapaliny.

Cievka, alebo sedlo ventilu je spojené s piestom zo železného kovu. Piest sa posúva v jadrovej trúbke z neželezného kovu, ktorá je obklopená cievkou elektrického vinutia. Pracovný rozsah napätia môže byť ľubovolný. Keď cievkou prechádza elektrický prúd, generuje sa magnetické pole, ktoré otvára/zatvára ventil.

Najbežnejšími elektromagnetickými ventilmi sú dvojcestné, dvojpolohové ventily. Tieto ventily sa môžu jednoducho otvárať a zatvárať. A tým umožňujú prietok, keď je cievka pod napätím. K dispozícií sú dve verzie „normálne uzavreté“ a „normálne otvorené“.

Tento typ ventilu je za normálnych pracovných podmienok uzavretý, resp. otvorený. Normálne otvorené v hydraulickom obvode je opak normálne otvoreného v elektronickom obvode. V elektrickom obvode to znamená, že spínač alebo kontakt je rozpojený a nepreteká elektrický prúd.

Trojcestné sedlové ventily sú taktiež bežné a umožňujú odklonenie toku kvapaliny z jedného kanálu do druhého.

Elektromagnetické ventily pre hydraulické, alebo pneumatické aplikácie sú k dispozícií ako potrubné modulárne jednotky, ako sú pneumatické, alebo hydraulické ISO ventily.[7][28][13]

HC-SR04

Modul HC-SR04 funguje ako ultrazvukový merač vzdialenosti. Tento modul umožňuje spoľahlivú detekciu v rozmedzí 2 centimetrov až 4 metrov. Jeho základná varianta obsahuje všetky komponenty na jednej doske a nemá žiadnu ochranu voči vysokej vlhkosti. Takýto modul teda nie je možné použiť na meranie vodnej hladiny. Jeho drahšia verzia prichádza s oddeleným vode odolným ultrazvukovým transceiverom[1]. Keďže môžme oddeliť elektroniku od samotného senzora, vieme takýto senzor využiť pre meranie hladiny vodnej nádrže.[18]

Pri meraní vzdialenosti pomocou zvukových vĺn treba brať do úvahy teplotu a vlhkosť vzduchu. Tieto parametre môžu spôsobiť male odchýlky pri meraní.

Obr. 3.7: Senzor HC-SR04(vľavo) a jeho vodeodolná verzia(vpravo)

Posuvný register 74HC595

Posuvné registre sú sekvenčné logické obvody, tvorené väčšinou kaskádnym radením bistabilných klopných obvodov typu D alebo JK. Úlohou posuvného registru je zapísať informáciu do pamäti a následne ju posúvať.

Posuvný register posúva vstupnú informáciu od jedného bi-stabilného obvodu k ďalšiemu na základe hodinového impulzu. Každý klopný obvod posunie informáciu zo svojho vstupu na výstup nasledujúceho klopného obvodu s príchodom aktívnej hrany taktovacieho impulzu.

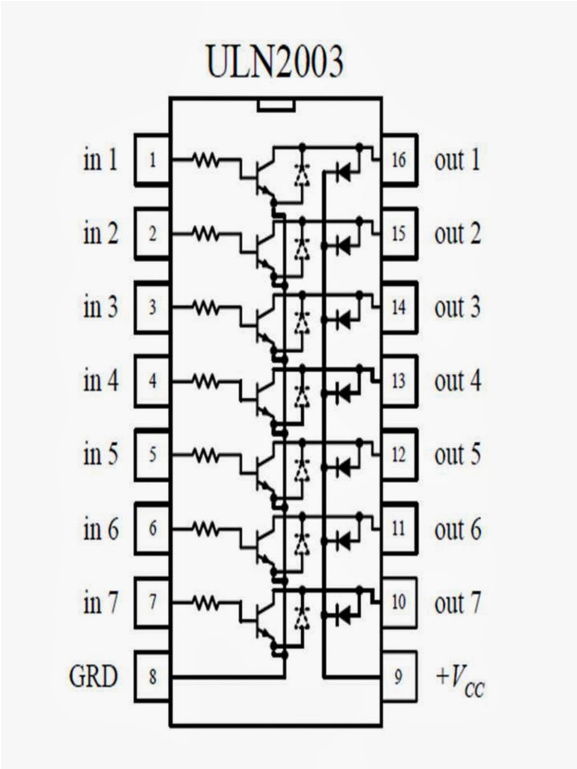
Posuvný register 74HC595 je osem bitový, zostavený z ôsmich klopných obvodov. Funguje na protokole Serial IN Paralel OUT. To znamená, že dáta do neho zapisujeme sériovo

[1] transceiver - vysielač a prijímač v jednom

a následne aktivujeme výstup všetkých paralelných pinov. Taktiež vieme pripojiť viac ako jeden takýto register za seba, čo nám umožňuje zvýšiť počet paralelných výstupov pri zanechaní rovnakého počtu vstupov. Najbežnejšie využitie takéhoto logického obvodu sú veľké LED panely, ktoré sú ovládane pomocou jedného mikrokontroléra. [30][27]

Tranzistorové pole

Tranzistorové pole sa skladá z viacerých tranzistorov v jednom integrovanom čipe. Takýto integrovaný čip nám umožňuje ušetriť miesto na výslednej doske.



Obr. 3.8: Schéma tranzistorového poľa ULN2003 uloženého v DIP16 obale.
Obrázok prevzatý z [1]

.

3.6 Komunikačné protokoly

Popísané často využívané komunikačné protokoly v implementačnej časti.

MQTT

MQTT (predtým Mesage Queuing Telemetry Transport, dnes MQ Telemetry Transport) je jednoduchý a nenáročný protokol pre prenos správ medzi klientmi prostredníctvom centrálneho bodu - brokera. Vďaka jeho nenáročnosti a jednoduchosti je ho možné ľahko implementovať aj do zariadení s „malými“ procesormi a pomerne rýchlo sa rozšíril. Pôvodné bol tento protokol navrhnutý v IBM, dnes za ním stojí Eclipse foundation.

Pri tomto protokole prebieha komunikácia pomocou TCP a používa dnes populárny vzor medzi IoT zariadeniami - publisher-subscriber. Všetky správy sú posielané na jeden centrálny bod (MQTT broker), ktorý sa stará o výmenu správ. Správy sú triedené do tzv. tém (topicov). Zariadenie buď publikuje v danej téme - to znamená, že dané zariadenie posiela dáta na broker, ktorý ich ukladá a distribuuje ďalším zariadeniam, alebo sa zariadenie prihlási na odber z jednej, alebo viacerých tém a broker následne preposiela všetky správy s danou témou na dané zariadenia. Jedno zariadenie môže byť prihlásené na odber z viacerých tém a zároveň na tie témy aj publikovať dáta.

Formát správ nie je nijak obmedzený, celý obsah je preposlaný ako binárne dáta. Najčastejšie sa ale využíva formát JSON, BSON, alebo obyčajná textová správa. Veľkosť správy je obmedzená na necelých 256MB.

MQTT zavádza tri úrovne potvrdenia správ - QoS(Quality of Service). Nie každý klient musí podporovať všetky tri úrovne.

- QoS 0 (At most once) - správa je odoslaná bez akéhokoľvek potvrdenia a teda doručenie takej správy nie je nijak zaručené
- QoS 1 (At least once) - správa je doručená aspoň raz každému zariadeniu prihlásenému na odber v danej téme
- QoS 2 (exactly once) - správa je doručená práve jeden krát

Všetky správy patria do určitej témy. Tieto témy sú hierarchické, oddelené lomkami. Hierarchia nie je pevne daná, záleží na aplikácií, ktorú si navrhneme. Správne navrhnúť hierarchiu nemusí byť triviálna úloha, je dôležité rozmyslieť si v rámci návrhu dátový model a rozhranie aké bude využité pri skutočnom behu aplikácie.

Okrem QoS sa pri správe nastavuje príznak (flag) uchovania správy, ktorý hovorí, že broker nemá správu po preposlaní zahodiť, ale uložiť a poslať novému odberateľovi danej témy. Posiela sa vždy posledná uložená správa v danej téme. Toto vieme využiť napríklad v aplikácii, ak chceme posledné dáta zo senzorov prečítať - vždy po spustení aplikácie bez nutnosti posielať novú požiadavku pre všetky zariadenia. [23][12]

REST

REST (Representational State Transfer) je architektúra rozhraní, navrhnutá pre distribuované prostredie. Rozhranie REST je použiteľný pre jednoduchý a jednotný prístup k zdrojom. Zdroje môžu byť dáta, alebo ak sa dajú popísať konkrétnymi dátami tak stavy aplikácie. Všetky zdroje majú vlastný identifikátor URI. REST definuje štyri základné metódy prístupu k nim.

Tieto štyri základné metódy pre prístup k zdrojom sú známe pod označením CRUD a teda vytvorením dát (Create), získaniu požadovaných dát (Retrieve), zmena dát (Update) a zmazanie (Delete). Tieto metódy sú implementované pomocou odpovedajúcich metód HTTP protokolu.

REST vyniká hlavne v spojení s JSON formátom, čo je v dnešnej dobe štandard pre API webových služieb. Vďaka svojej bezstavovosti umožňuje jednoduchú interakciu s modernými webovými aplikáciami, ktoré sú často založené na paralelnom spracovaní distribuovaného obsahu. [24]

Kapitola 4

Dynamicky rekonfigurovateľný operačný systém

Dynamicky rekonfigurovateľný operačný systém je systém, ktorý umožňuje propagáciu systémových zmien do systémovej implementácie počas behu systému. Pre zmenu konfigurácie nie je nutné systém pozastaviť, ani reštartovať. Aby sme mohli dokázať túto rekonfigurovateľnosť, je nutné systém rozdeliť na menšie aplikácie, ktoré dokopy umožňujú funkciu celého systému.

4.1 PNOS

PNOS je dynamicky rekonfigurovatelný operačný systém pre vstavané zariadenia, špecifikovaný s využitím Petriho sietí. Špecifikácia systému sa skladá z popisov modulov systému, ich vnútorných procesov. Komunikácia medzi modulmi je popísaná pomocou referenčných Petriho sietí. Využitie Petriho sietí pre špecifikáciu systému umožňuje špecifikovanie, ladenie, testovanie, simuláciu a overovanie spustiteľného modelu. Dynamická rekonfigurovateľnosť bola dosiahnutá dekompozíciou celkovej funkcionality na relatívne malé interpretovateľné výpočetné časti. Tie časti, ktoré nie sú aktuálne rekonfigurované ostanú bežať bez akejkoľvek zmeny.

Samotný beh takto špecifikovaného systému stojí na platforme PNOS, ktorá na interpretáciu hosťovaných sietí využíva virtuálny stroj Petri Nets Virtual Machine (PNVM).

Systém PNOS využíva na komunikáciu medzi oddelenými interpretovanými časťami protokol, z ktorého vychádza aj komunikačný protokol MPOS. [33]

4.2 MPOS

Systém MPOS(Micro Python Operating System) vychádza zo systému PNOS. Prináša podobné možnosti rekonfigurácie aké poskytuje PNOS pre moduly ESP8266 a ESP32, s využitím interpretovaného jazyka Micropython namiesto Petriho sietí.

Systém MPOS dodržiava rovnaký komunikačný protokol, ktorý využíva PNOS, vďaka čomu je možné dosiahnuť spoločného behu a spolupráce oboch systémov na jednej sieti.

Komunikačný protokol MPOS spolu so servisnými správami je popísaný v [5]

4.3 Hardvérové platformy pre systém MPOS

ESP32

Wifi modul ESP32 je SoC (System on Chip) mikrokontrolér s integrovanou WiFi a Bluetooth technológiou. Modul bol vytvorený firmou Espressif Systems a na trh prišiel v roku 2016 ako nástupca o dva roky staršieho WiFi čipu ESP8266 s veľkým počtom vylepšení a nových funkcií. [10][34][8]

Základné parametre:

- 240 MHz dvojjadrový mikrokontrolér Tensilica LX6.
- Integrovaná 520 KB SRAM.
- 16 MB flash pamäť namapovaná do CPU kódového priestoru.
- Integrovaný WiFi vysielač a prijímač.
- Integrovaný dvojmódový Bluetooth (podporujúci technológiu BLE).
- Pracovné napätie 2.3V až 3.6V.
- 34 GPIO pinov (podporujúce rozhrania ako UART, SPI, I2C a niektoré piny vybavené A/D a D/A prevodníkmi).

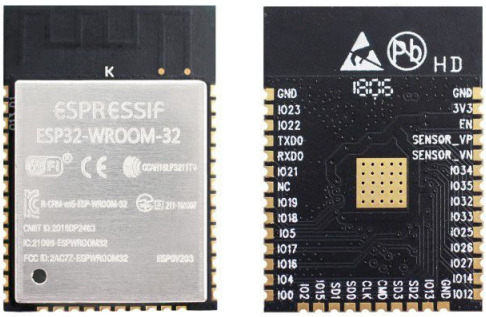
Ovládanie a programovanie modulu

Bežný spôsob ovládania modulu umožňuje ovládať iba jeho WiFi časti pôvodným firmwarom modulu ESP32 pomocou tzv. AT príkazov cez sériovú linku. Ďalšia možnosť ovládania modulu je s využitím iného firmwaru, ako je napríklad NodeMCU, ktorý umožňuje ovládanie WiFi časti a vytvorenie vlastného riadiaceho programu a ovládania GPIO pinov.

Typy modulov

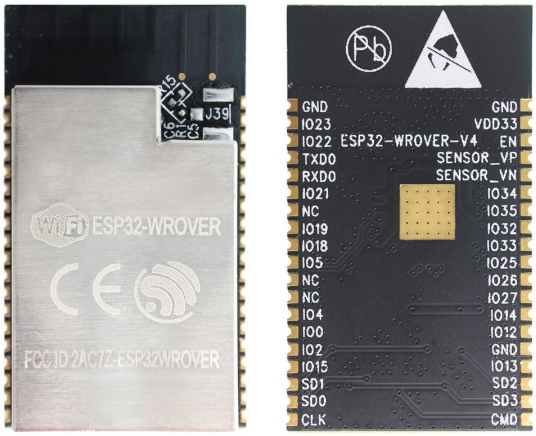
Pre vysoký dopyt po týchto zariadeniach sú dostupné rozličné moduly, napríklad:

- ESP-WROOM-32 je najmenší modul určený priamo pre konečné produkty.



Obr. 4.1: ESP-WROOM-32 [9]

- ESP-WROVER je vylepšený ESP-WROOM32, vybavený naviac o 4 MB pseudo statickou RAM(PSRAM). Modul je k dispozícií v dvoch variantách. Prvá obsahuje vstavanú PCB anténu a druhá s IPEX anténou.



Obr. 4.2: ESP-WROVER [9]

- Rozličné typy vývojových modulov so zabudovaným USB rozhraním pre programovanie cez sériovú linku, poskytujúci napájanie celého modulu, zabudovaným OLED displejom alebo zabudovaním napájaním z batérie.

ESP-WROOM-32 je vývojový modul s už zabudovaným USB rozhraním pro programovanie cez sériovou linku a zároveň poskytujíci napájenie celého modulu.

NodeMCU

[29, NodeMCU je open-source firmware (operační systém) umožňující vytvořit vlastní řídící program a ovládat GPIO piny čipu ESP32. Pro použití NodeMCU je nutné přehrát původní firmware, který je v čipu nahrán od výrobce. Vytvořený program se nahrává přímo do čipu, díky čemuž je Wifi modul schopen samostatné funkce bez pomocného mikrokontroleru. Vývojové prostředí PlatformIO ulehčuje práci programátora, neboť nahrávání firmwaru NodeMCU provádí automaticky]

Kapitola 5

Prípad použitia a návrh systému

Dynamicky rekonfigurovateľný operačný systém ako MPOS má široké využitie - či už v domácnosti, alebo v profesionálnom nasadení. So vzrastajúcim počtom zariadení vzniká problém udržiavať prehľad o tom, aké aplikácie bežia na akom zariadení a stav, v akom sa práve nachádzajú zariadenia a aplikácie.

Táto kapitola podrobne popisuje návrh správcovského systému pre MPOS, konkrétny problém a návrh jeho riešenia s využitím ESP mikrokontrolérov a operačného systému v interpretovanom jazyku MicroPython (MPOS) a návrh mobilnej aplikácie pre zobrazenie výsledkov.

5.1 Prípad použitia spravovaného distribuovaného riadiaceho systému

Pre ukážku, ako daný systém môže fungovať v nasadení som sa rozhodol systém implementovať do konkrétneho rozsiahlejšieho prípadu. Tento systém bude mať na starosti automatizáciu chaty, kde bude rozdelený na niekoľko samostatných modulov, ktoré budú spolu tvoriť jeden systém.

Popis problému

Chata sa skladá z niekoľkých izieb, pivnice a podkrovia. Táto chata sa nachádza na území s veľmi vysokou vlhkosťou vzduchu. V pivnici teda často dochádza k zrážaniu vody na stenách, kde sa následne tvoria súvislé prúdy vody, čo ma za následok vytápanie pivnice, pri čom dochádza k znehodnoteniu majetku.

V pivnici sa nachádzajú dve vodné čerpadlá, bojler pre ohrev vody a potrubie pre napustenie vody do vodnej nádrže v prípade nadmerného sucha. Jedno vodné čerpadlo čerpá vodu z nádrže do tlakovej nádoby a následne tlačí vodu do vodovodného systému, druhé slúži na výťah vody z nádrže.

Návrh riešenia

Vlhkosť v pivnici som sa rozhodol riešiť dvoma spôsobmi. Prvý spôsob predstavuje montáž ventilačného systému, ktorý bude mať za úlohu vetrať vzduch iba v prípade ak sú splnené určité podmienky - vonku sa nachádza suchší vzduch ako vo vnútorných priestoroch. Takýto systém by mal byť čo najtichší s čo najnižšou spotrebou elektrickej energie, no zároveň dosť výkonný pre rýchle prevetranie celého priestoru.

Druhý spôsob zahrňuje inštaláciu automatického odvlhčovača vzduchu pre prípad, ak sa dlhodobo nedá vyvetrať vzduch. Problém s odvlhčovačom vzduchu je jeho vysoká cena prevádzky a vysoká hlučnosť. Preto bude musieť byť obmedzený iba pre prípad nutnosti. Pre lepšiu predstavu som vytvoril schému návrhu viď. 5.1.

Hygrometer[1] zvyčajne poskytne iba relatívnu vlhkosť k danej teplote. Obsah vody sa ale veľmi výrazne mení na základe teploty. Preto je potrebné zistiť absolútnu vlhkosť ovzdušia. Absolútna vlhkosť ovzdušia sa dá reprezentovať aj ako rosný bod[2]. V porovnaní s relatívnou vlhkosťou sa považuje za presnejšiu reprezentáciu merania vlhkosti v priestore. Rosný bod sa dá jednoducho odvodiť na základe relatívnej vlhkosti a teploty ovzdušia. [26][16]

Pre vetranie miestnosti teda treba zistiť vonkajšiu aj vnútornú absolútnu vlhkosť vzduchu. Systém bude na základe rozdielu medzi týmito dvoma hodnotami prevetrávať vzduch v miestnosti. Pre meranie relatívnej vlhkosti a teploty v miestnosti bude využitý senzor SHT75. Z dôvodu vysokej vonkajšej vlhkosti bude vonku použitý senzor AM2315, ktorý je schopný odolávať náročnejším podmienkam.

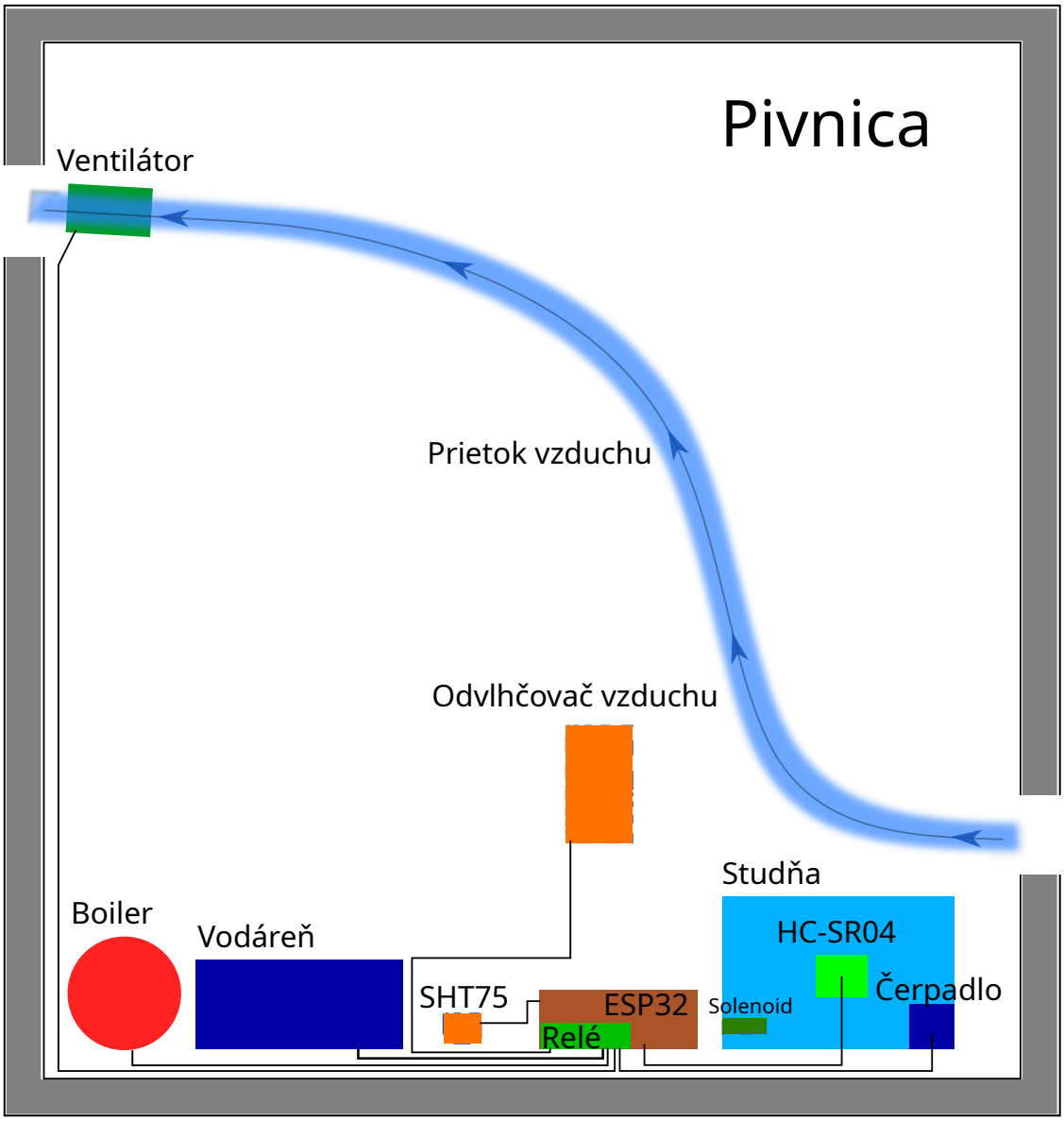
Ultrazvukový senzor HC-SR04 bude zaznamenávať výšky hladiny vody vo vodnej nádrži. Je nutné, aby takýto senzor bol vodotesný, keďže sa bude nachádzať priamo nad vodnou hladinou. Tento senzor sa bude starať o automatické ovládanie množstva vody v nádrži. Na vypúšťanie a využívanie vody budú využité obidve čerpadlá a na napúšťanie bude využitý dvojitý solenoidový ventil[3].

Systém bude teda obsahovať dva moduly ESP32 a monitorovací systém. Jeden modul bude napojený v podkroví a ten bude monitorovať vonkajšiu teplotu. Druhý modul bude v pivnici. Tento modul bude merať vnútornú teplotu pivnice, vlhkosť, riadiť napúšťanie, stav, odčerpávanie vody, ventiláciu, bojler a na displeji bude zobrazovať informácie zo senzorov. Všetky dáta zo senzorov budú odosielané do databázy pre analýzu.

[1] Hygrometer - zariadenie pre meranie vlhkosti ovzdušia

[2] Rosný bod - teplota pri ktorej je vzduch maximálne nasýtený vodnou parou

[3] Solenoidový ventil - ventil riadený kovovou cievkou, ktorá vytvára elektromagnetické pole pri privedení energie.



Obr. 5.1: Schéma pivnice doplnená o rozloženie senzorov a zariadení potrebných pre navrhnuté riešenie.

Interpretácia výsledkov

Okrem zobrazovania dát na displeji v pivnici by bolo vhodné, keby boli dáta dostupné aj v inej forme. Preto som sa rozhodol vytvoriť mobilnú aplikáciu, ktorá bude sprístupňovať všetky informácie o stave systému spolu s historickými údajmi. Systém však nie je obmedzený iba na konkrétnu aplikáciu. Podľa toho, ako sú napísane konkrétne aplikácie, je možné systém prepojiť aj s inými systémami - ako napríklad populárny Domoticz.

5.2 Návrh správcovského systému pre MPOS

Tento systém bude administrátorským prístupom pre všetky zariadenia so systémom MPOS, napojené na jednom MQTT brokeri pod jednou spoločnou témou.

Systém bude zobrazovať stav zariadení a jednotlivých inštancií aplikácií. Zároveň umožňuje tieto aplikácie a jednotlivé inštancie upraviť.

Okrem interakcie s databázou bude systém musieť implementovať klienta pre MQTT komunikáciu. Tento klient bude spracovávať a vytvárať servisné správy pre jednotlivé zariadenia na MQTT brokeri. V daných intervaloch (špecifikovaných od užívateľa) bude kontrolovať stav zariadení a pri prípadnej poruche skúsi zariadenie/aplikáciu reštartovať.

Tento systém bude musieť obsahovať algoritmus pre kontrolu zariadení, sledovanie stavu spustených a nainštalovaných aplikácii a logovanie správy - ako z jednotlivých zariadení, tak aj z jednotlivých inštancií.

Systém sa bude skladať z niekoľkých náhľadov:

1. Náhľad na všetky zariadenia, ktoré boli pripojené k systému s ich stavom a pridelenými parametrami.
2. Náhľad na všetky aplikácie s možnosťou upraviť zdrojový kód danej aplikácie.
3. Náhľad na všetky inštancie aplikácií s prideleným zariadením (na ktorom má byť daná inštancia spustená), pridelenou aplikáciou (ktorú ma spustiť) a jej pridelené parametre. Tento náhľad bude obsahovať aj výpis všetkých správ tzv. log inštancie.
4. Prehľad administrátorských účtov.

Návrh databázy v podobe ER diagramu 5.3.

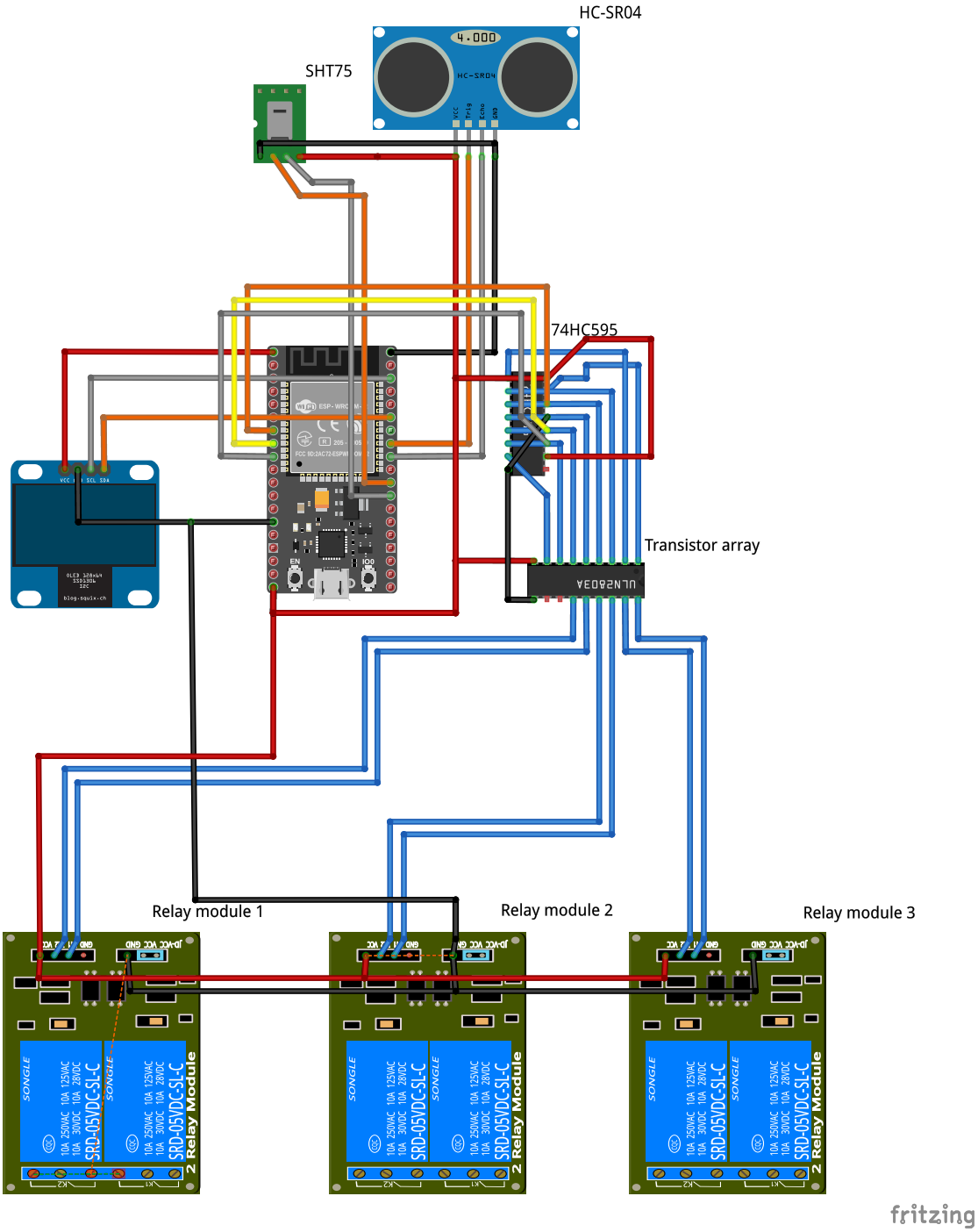
5.3 Návrh mobilnej aplikácie

Mobilná aplikácia bude slúžiť pre demonštračný účel. Umožní užívateľovi náhľad na zaznamenané dáta a zároveň ponúkne užívateľovi možnosť ovládať niektoré prvky.

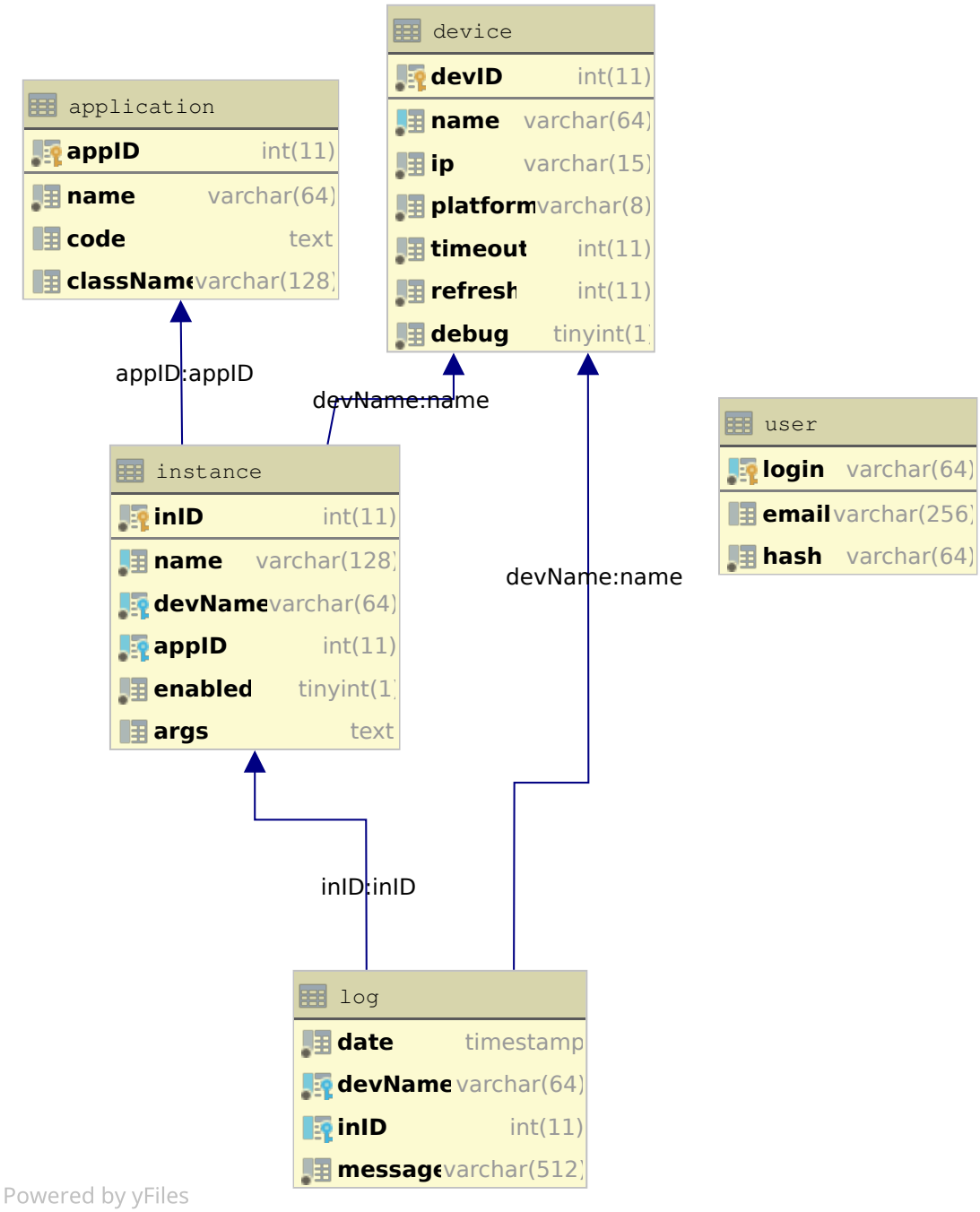
Aplikácia bude mať niekoľko náhľadov:

1. Náhľad na všetky vopred definované premenné, vyčítané z určitej témy na MQTT brokeri.
2. Náhľad na všetky ovládateľné prvky - relé moduly
3. Náhľad, ktorý umožní meniť automatizované prvky v systéme (napr. ventilácia).
4. Náhľad, na ktorom budú zobrazené historické dáta v podobe grafu z daných premenných.

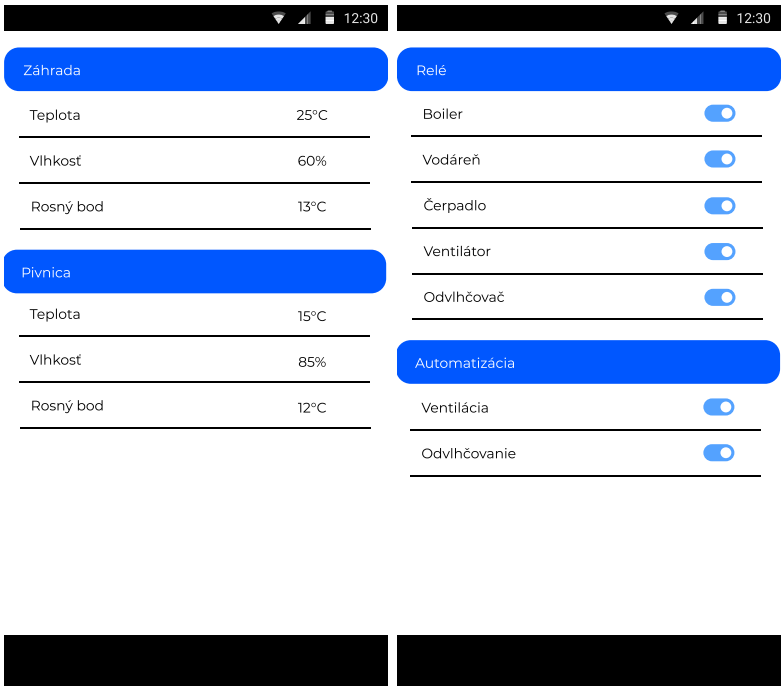
Približný návrh výzoru aplikácie 5.4



Obr. 5.2: Návrh schémy zapojenia ovládacej dosky



Obr. 5.3: Návrh databázy pre správcovský systém.



Obr. 5.4: Návrh mobilnej aplikácie a príklad niekoľkých meraných/ovládateľných prvkov

Kapitola 6

Implementácia a testovanie

V tejto kapitole sú popísané jednotlivé časti systému a jeho najdôležitejšie časti zdrojového kódu, ktoré tvoria jadro systému.

Systém sa skladá z troch najdôležitejších častí:

1. Webová správa, ktorá umožňuje administráciu jednotlivých zariadení a ich funkčného kódu.
2. Samotné zariadenia s pripojenými senzormi.
3. Klientská aplikácia umožňujúca zobraziť namerané hodnoty s ich históriou a možnosťou ovládania aktívnych prvkov

6.1 Modifikácia MPOS

Pre zjednodušenie ovládania a zároveň opravu chýb som realizoval pár zmien oproti originálnemu MPOS systému. Medzi ne patrí aj automatické stiahnutie potrebných knižníc pre správne spustenie systému na zariadeniach pri prvom pripojení zariadenia na internet.

Úprava inštalácie aplikačného kódu

Pôvodný systém prijímal a spracovával všetky správy rovnako. Pre každú prichádzajúcu správu zamenil znaky „{, }“ za „[,]“. Ak ide o správu obsahujúcu aplikačný kód, nesmieme takúto zámenu vykonať. Pre vynechanie aplikačného kódu zo správy bola zavedená zarážka na konci aplikačného kódu „@END“ pridaná na samotný koniec aplikačného kódu. Upravené prijímanie správ je uvedené v 6.1

```
def decodeMessage(topic,msg):
  global runningTasks
  global tasks
  global routingTable
  gc.collect()
  try:
    data = msg.decode('utf-8')
    if "class" in data and "@END" in data:
      part = data[:data.index('class')].replace('{', '[').replace('}', ']')
      part += data[data.index('class'):data.index('@END')]
      part += data[data.index('@END')+4:].replace('{', '[').replace('}', ']')
```

```
12      data=eval(part)
13    else:
14        data=eval(data.replace('{', '[').replace('}', ']'))
15    data[0:2]    # test
16  except Exception as ex:
17    broadcastMqttSend("message_format", ["Error","\"\"+str(ex)+"\"\""])
18    return
```

Výpis 6.1: Upravená funkcia pre spracovanie prichádzajúcej správy v MPOS

Funkcia sendtoTopic()

Pre zjednodušenie odosielania správ na zvolenú tému na MQTT brokeri bola pridaná funkcia „sendtoTopic()“ 6.5. Táto funkcia využíva už zabudovanú funkciu „send()“ a je schopná odoslať na MQTT broker správu obsahujúcu konštantu, premennú, pole, Python slovník alebo aj pole slovníkov. To umožňuje efektívne pracovať s dátami. Voliteľný argument „retain“ umožňuje uloženie správy na MQTT tému pomocou MQTT retain príznaku 3.6. Odosielanie takýchto správ vyžaduje, aby bol smerovač správ aktívny. Inak sa správy neodošlú na zadanú tému.

```
1 #hodnota: 88.1
2 {"*", {"172.24.1.95", "Outside_Temp_Hum", "mqtt"}, {"sensor/raspberry/humidity", 88.1}, "retain"}
```

Výpis 6.2: Príklad správy pre smerovač pre odoslanie hodnoty na špecifickú MQTT tému s MQTT príznakom retain

```
1 #hodnota: [10.12, 4.03]
2 {"*", {"172.24.1.112", "Cellar_Temp_Hum", "mqtt"}, {"pnos/message/ESP_30aea48c28f0/Ventilation/
      cellar", {10.12, 4.03}}}
```

Výpis 6.3: Príklad správy pre smerovač pre odoslanie pola na špecifickú MQTT tému

```
1 #hodnota: {"Relay": "Fan", "State":False}
2 {"*", {"172.24.1.95", "Ventilation", "mqtt"}, {"pnos/message/ESP_Cellar/Relays/relay", {{"Relay", "State
      "}}, {{"Fan", False}}}}
```

Výpis 6.4: Príklad správy pre smerovač pre odoslanie Python slovníku na špecifickú MQTT tému

```
1  def sendtoTopic(self, topic, payload, retain = False):
2    if type(payload) is list and type(payload[0]) is dict:
3      keys = []
4      values = []
5      for dictionary in payload:
6        keys.append(list(dictionary.keys()))
7        values.append(list(dictionary.values()))
8      self.send("mqtt", [topic, keys, values], retain)
9    elif type(payload) is dict:
10     self.send("mqtt", [topic, list(payload.keys()), list(payload.values())], retain)
11   else:
12     self.send("mqtt", [topic, payload], retain)
```

Výpis 6.5: Funkcia pre odosielanie správy na smerovač, ktorý následne správu prepošle na danú tému

6.2 Zariadenia pre demoaplikáciu

Pre riešenie problému popísaného v sekcii 5.1 boli využité dva moduly ESP32 nakonfigurované na rovnakú bezdrôtovú sieť. Tieto zariadenia boli pripojené k rovnakému MQTT brokeru pod spoločnou témou. Zjednodušený nákres so zobrazením modulov, aplikácií a ich vzájomného prepojenia je možné vidieť na obrázku 6.1. Aplikácie na moduloch boli písane tak, aby pre rôzne procesy mohli zdieľať veľkú časť kódu. Procesy sa líšia v názve a zadaných parametroch, no zdieľajú jeden aplikačný kód. Týmto spôsobom je možné ušetriť miesto na konkrétnych moduloch.

Využité knižnice

V moduloch boli pre spracovanie dát zo senzorov využité volne dostupné knižnice. Upravené pre konkrétny prípad.

- **BME280** - Čítanie dát o teplote a vlhkosti zo senzora.
- **HCSR04** - Čítanie nameranej vzdialenosti zo senzora.
- **AM2315** - Čítanie dát o teplote a vlhkosti zo senzora.
- **DS18x20** - Oficiálna knižnica pre čítanie teploty z DS18B20 senzora.

ESP_Cellar

Tento modul sa nachádza v pivnici. Namiesto senzora SHT75 bol nakoniec využitý senzor BMP/BME280 (kvôli problému s knižnicou v interpretovanom jazyku MicroPython). Tento senzor nám umožňuje okrem teploty a vlhkosti merať aj atmosférický tlak. Zapojenie senzorov je navrhnuté na obrázku 5.2. Následné prevedenie vytvorenej dosky je navrhnuté na obrázku 6.11 a popísané v schéme v prílohe B.1.

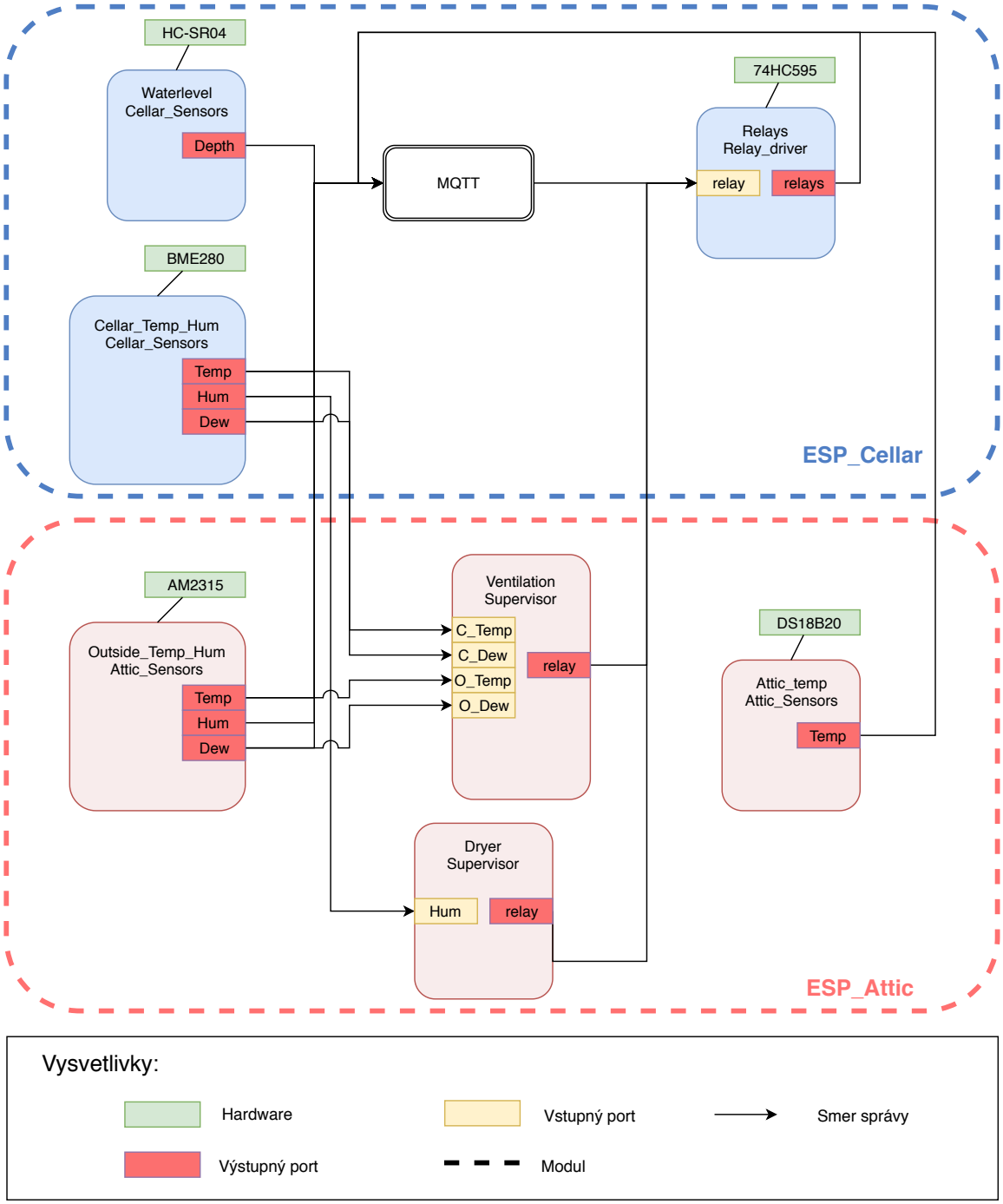
Na tomto module sú spustené tri procesy, z ktorých dva „Waterlevel“ a „Cellar_Temp_Hum“ zdieľajú rovnaký aplikačný kód uvedený v prílohe A.1. Tretí „Relays“ proces pre svoju odlišnosť obsahuje iný zdrojový kód, popísaný v prílohe A.2.

- Proces „Waterlevel“ zaznamenáva dáta zo senzora HC-SR04 popísaného vyššie 3.7. Tieto dáta udávajú výšku vodnej hladiny. Dáta vypisuje iba mobilná aplikácia a nie sú inak spracovávané.
- Proces „Cellar_Temp_Hum“ odosiela teplotu, vlhkosť a rosný bod zo senzora BMP/BME280.
- Posledný proces „Relays“ spracováva požiadavky pre ovládanie relé modulov a následne ovláda konkrétne relé cez SPI zbernicu pomocou registra „74HC595“.

ESP_Attic

Tento modul sa nachádza v podkroví, kde ma prístup k vonkajšiemu senzoru AM2315 pipísaného v sekcií 3.2 a vnútornému teplotnému senzoru DS18B20 viď. 3.3. Na tomto module sú spustené štyri procesy z ktorých „Outside_Temp_Hum“, „Attic_temp“ zdieľajú aplikačný kód popísaný v prílohe A.4 a „Ventilation“ s „Dryer“ zdieľajú aplikačný kód popísaný prílohe A.3.

- Proces „Outside_Temp_Hum“ zaisťuje meranie vonkajšej teploty, vlhkosti a rosného bodu zo senzora AM2315, popísaného v sekcií 3.2. Môj konkrétny modul nevie zmerať hodnotu vlhkosti vyššiu ako 90.1%, čo sa prejaví na priebehoch dát aj na funkcionalite demopríkladu.
- Proces „Attic_temp“ zaisťuje meranie vnútornej teploty zo senzora DS18B20 popísaného v sekcií 3.3. Hodnota slúži iba pre informatívny účel.
- Proces „Ventilation“ spracováva dáta z vonkajšieho senzora AM2315 a senzora v pivnici BMP/BME280 a na základe získaných dát spúšťa resp. vypína vetranie priestoru.
- Posledný proces „Dryer“ spracováva vlhkosť z pivnice. Ak vlhkosť presiahne hodnotu 80%, zapne sa odvlhčovač na dobu, postačujúcu pre odvlhčenie priestoru.



Obr. 6.1: Schéma zapojenia a komunikácie aplikácií a ich procesov na jednotlivých moduloch.

6.3 Správcovský systém

Tento systém som sa rozhodol realizovať v jazyku Javascript s použitím softvérového systému Node.js. REST-ové volania prehliadačov spracováva knižnica express. Všetky náhľady sú dynamicky renderované pomocou Pug knižnice. Jadro aplikácie tvorí súbor „index.js“. V

tomto súbore sú definované všetky REST-ové volania. Konfiguračné parametre sa nachádzajú v súbore „configs.js“. Web server sa spustí súborom „www“ a po spustení načúva na porte 22000.

Knižnice

- **Express** - Vytvorenie webového servera, spracovávanie REST požiadavkov.
- **Pug** - Renderovanie HTML stránok.
- **Cookie-session** - Vytvorenie cookies na strane klienta pre overovanie prihláseného užívateľa.
- **Cookie-parser** - Pomocná knižnica pre spracovanie informácií z cookies.
- **Mqtt** - Spojenie s MQTT brokerom, prihlasovanie na odber jednotlivých tém a odosielanie dát.
- **Mysql** - Spojenie s MySQL databázou.
- **Ace** - Pokročilý editor zdrojového kódu pre web.

Docker

Pre nainštalovanie všetkých potrebných knižníc bol k systému vytvorený DockerFile. Systém je možné spustiť kdekoľvek definovaním iba niekoľkých konfiguračných parametrov.

Vygenerovanie docker obrazu je možné príkazom: docker **build** −t nazov−obrazu .

Spustenie samotnej inštancie je možné príkazom docker run −−name nazov−instancie −d nazov−obrazu:latest

Povinné ENV premenné pri spustení inštancie

- **MQTT_URL** - URL adresa na MQTT broker.
- **MYSQL_HOST** - Adresa MySQL servera.
- **MYSQL_USER** - MySQL užívateľ.
- **MYSQL_PASS** - Heslo pre MySQL užívateľa.
- **MYSQL_DB** - MySQL databáza.

Voliteľné ENV premenné pri spustení inštancie

- **MQTT_USER** - Meno potrebné k prihláseniu na MQTT broker.
- **MQTT_PASS** - Heslo potrebné k prihláseniu na MQTT broker.
- **MQTT_TOPIC** - Hlavná téma pre odoberanie a odosielanie MQTT správ.
- **MQTT_SUBTOPIC** - Vedľajšia téma pre smerovanie MQTT správ medzi zariadeniami a odlišnými témami.
- **MPOS_DEBUG** - Výpis všetkých prijatých správ pre MQTT tému.

```
FROM node:8

WORKDIR /usr/src/app

COPY package*.json ./

RUN npm install

COPY . .

EXPOSE 22000
CMD [ "npm", "start" ]
```

Výpis 6.6: DockerFile pre generovanie Docker obrazu

MQTT klient

Táto časť programu zahŕňa obsluhu jednotlivých zariadení a ich komunikáciu. Sleduje stav zariadení a pokiaľ je aplikácia nastavená na automatickú inštaláciu, automatický ju nainštaluje a spustí.

Zavolaním funkcie „listen()“ sa na pozadí systému spustí proces, ktorý spracováva servisné správy z hlavnej a vedľajšej MQTT témy. Pri prichádzajúcej správe sa zavolá funkcia z MQTT knižnice pre spracovanie správy, ktorá na základe témy rozhodne, o akú správu sa jedná. Pokiaľ ide o správu z hlavnej témy, zavolá funkciu „handleDeviceMessages()“, ktorá spracuje správy zo zariadení. Pri servisnej správe „new_node“ alebo „discover“ vyžaduje správu o stav zariadenia.

Správu o stave zariadenia spracováva funkcia „updateDeviceStatus()“, ktorá vytvorí položku nového zariadenia a následne ju zapíše do databázy. Pokiaľ už takáto položka existuje namiesto uloženia nového zariadenia iba obnoví informácie o existujúcom zariadení, jeho názve, IP adresy a platforme. Do pamäte je uložená informácia o spustených, nainštalovaných aplikáciach a stave pamäti zariadenia. Následne funkcia spustí časovanú funkciu „checkDevice()“. Táto funkcia pravidelne žiada informáciu o stave zariadenia.

Po každej správe zo zariadenia (pokiaľ sa nejedná o správu o stave, alebo servisnú správu „discover“) sa spustí funkcia „saveDeviceLog()“, ktorá uloží celú správu do databázy.

Pokiaľ sa jedná o správu typu „mqtt“, spustí sa funkcia „handleMessageForwarding()“, ktorá spracuje a prepošle správu na uvedenú tému na MQTT broker.

Pre úspešné spustenie inštancie je potrebné aplikačný kód najprv nahrať na zariadenie. Na to slúži funkcia „installApp()“. Pokiaľ aplikačný kód už existuje, je potrebné ho najprv vymazať pomocou funkcie „uninstallApp()“. Po úspešnom nahratí kódu je možné inštanciu spustiť pomocou funkcie „activateApp()“.

Smerovanie správ medzi zariadeniami a MQTT brokerom

Pre úspešnú komunikáciu medzi zariadeniami je nutné dodržiavať určený formát. Pre odoslanie správy na zariadenie slúži zabudované smerovanie v MQTT klientovi.

hlavná-téma/vedľajšia-téma/názov-zariadenia/názov-aplikácie/názov-inštancie/port

Obr. 6.2: Formát MQTT témy pre odoslanie správy na zariadenie

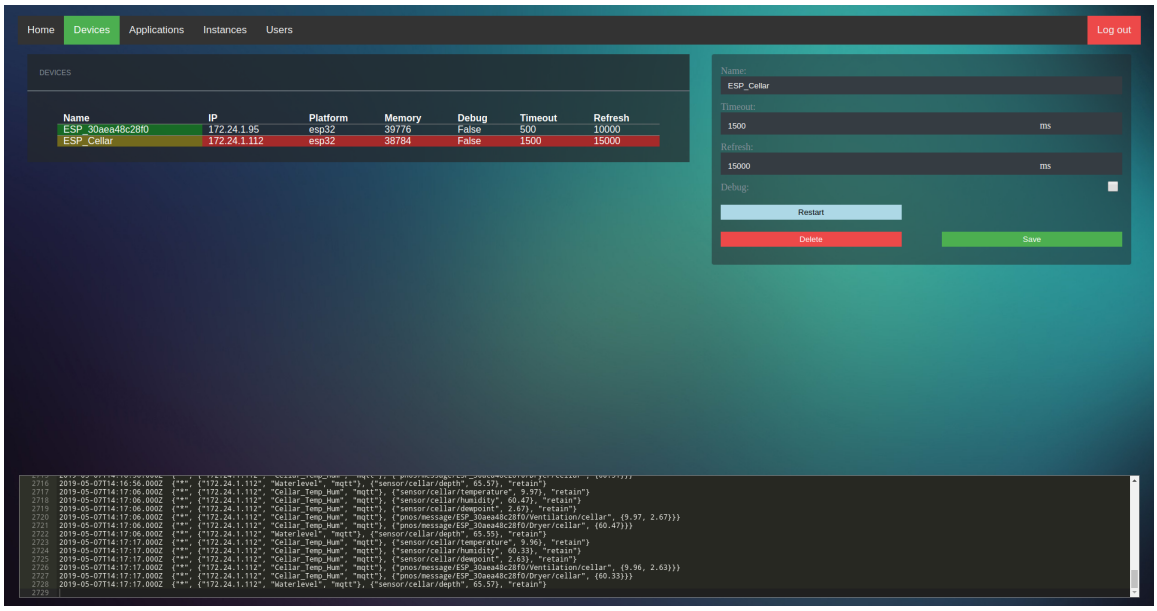
pnos/message/ESP_30aea48c28f0/Ventilation/cellar

Obr. 6.3: Príklad MQTT témy pre odoslanie správy na zariadenie „ESP_30aea48c28f0“, aplikáciu „Ventilation“ a port „cellar“

Užívateľské náhľady

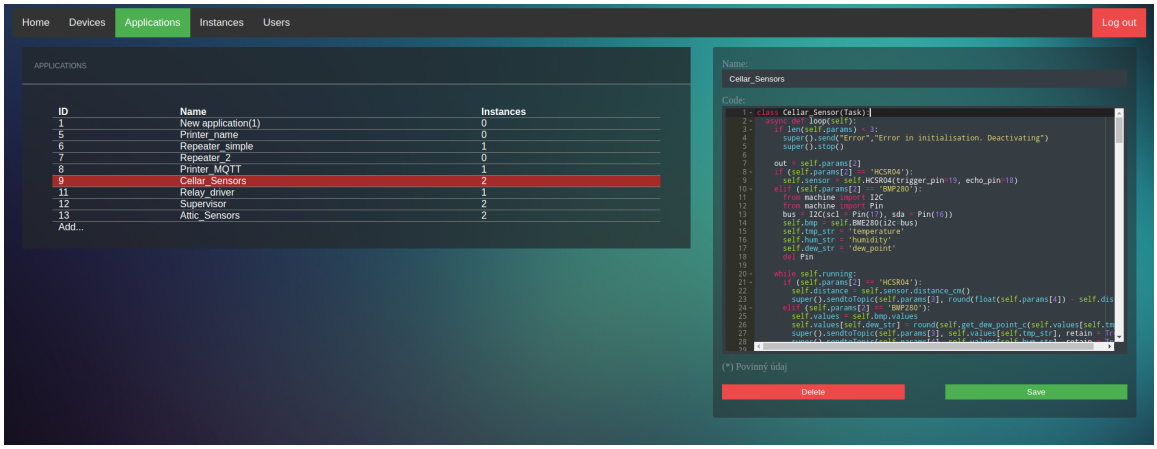
Podľa návrhu v sekcii 5.2 sa užívateľský prístup skladá z niekoľkých náhľadov. Jednotlivé náhľady obsahujú tabuľku podľa definovaného obsahu a možnosť úpravy jednotlivých položiek.

- **Home** - zobrazí sa hneď po načítaní rozhrania. Zobrazuje základne informácie. Je to jediný náhľad dostupný bez prihlásenia užívateľa.
- **Devices** - zobrazí všetky automaticky detekované zariadenia komunikujúce na definovanej hlavnej MQTT téme. Po kliknutí na konkrétne zariadenie v tabuľke sa zobrazí výpis všetkých logov pre konkrétne zariadenie. Umožňuje meniť iba základne parametre. „Timeout“ - čas, za ktorý má považovať zariadenie za neaktívne. „Refresh“ - ako často je potrebné zisťovať stav zariadenia. „Debug“ - zapnutie logovania správ pre zariadenie.



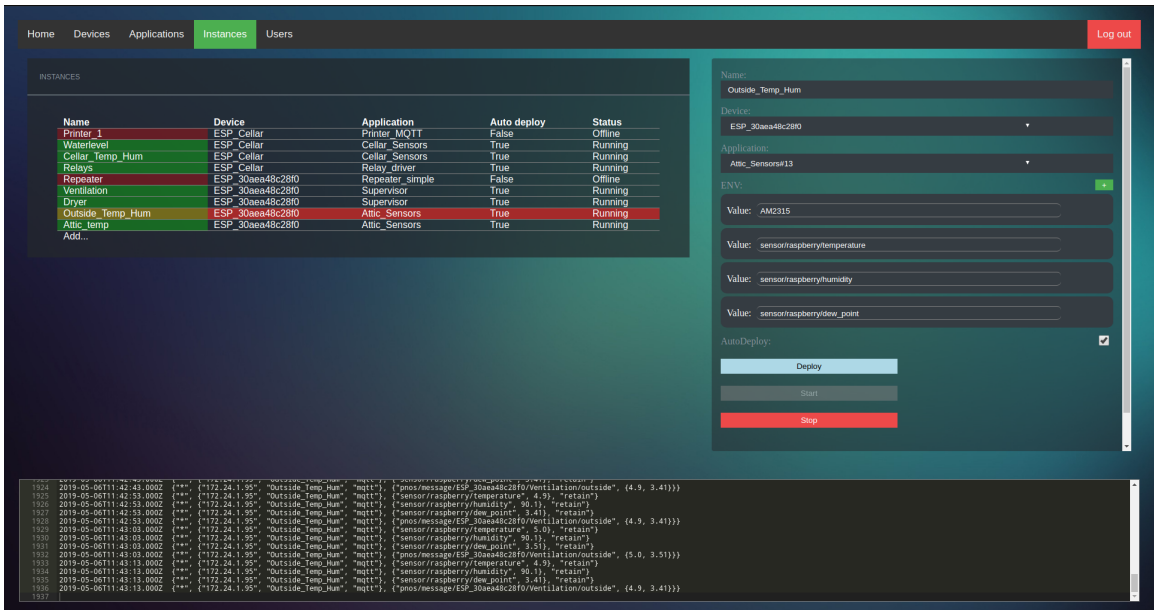
Obr. 6.4: Pohľad na zariadenia pripojené na hlavnú MQTT tému.

- **Applications** - zobrazí všetky aplikácie uložené v databáze. Umožňuje upravovať kód jednotlivých aplikácii pomocou knižnice „Ace“. Zadaný kód aplikácie musí zodpovedať špecifikácii systému popísanej v [6]. Meno aplikácie slúži len pre lepšiu orientáciu pre užívateľa. Skutočný názov nainštalovanej aplikácie je závislé od názvu triedy.



Obr. 6.5: Pohľad na aplikácie uložené v databáze.

- **Instances** - zobrazí všetky inštancie uložené v databáze a ich momentálny stav. Umožňuje pridať/meniť parametre inštancie. Tlačidlo „Deploy“ slúži k načítaniu kódu aplikácie na zariadenie. Ak je kód načítaný, umožní systém prístup na ďalšie tlačidlá „Start“ a „Stop“. Tie umožňujú spustenie a ukončenie inštancie. Pre automatické načítavanie, spustenie a reštartovanie pri páde inštancie slúži box „AutoDeploy“.



Obr. 6.6: Pohľad na inštancie uložené v databáze a ich aktuálny stav.

- **Users** - zobrazí prehľad užívateľských účtov z ktorých je možné prihlásenie do systému a zároveň umožňuje užívateľom meniť heslo

6.4 Android klientská aplikácia

Aplikácia sa skladá z hlavnej aktivity, ktorá obsahuje fragmenty pre zobrazenie podľa jednotlivých náhľadov popísaných v sekcií 5.3. Každý náhľad obsahuje ďalšie fragmenty reprezentujúce jednotlivé modely.

Knižnice

V mobilnej aplikácii boli využité mimo zabudovaných android knižníc aj nasledujúce knižnice:

- **RxJava** - Rozširuje vývojový vzor pozorovateľa (Observer), pridáva podporu nízko úrovňového vláknenia aplikácie.
- **RxMQTT** - MQTT klientská knižnica umožňujúca reaktívny prístup k programovaniu.
- **Gson** - Knižnica, ktorá umožňuje preklad JSON serializácie na Java objekty a opačne.
- **Joda** - Knižnica pre štandardizovanú prácu s dátumom a časom.
- **GraphView** - Umožňuje jednoduché vytvorenie grafu z dátovej štruktúry.
- **HTTPCore** - Jednoduché odosielanie/prijímanie REST požiadavkov.

Modely

Jadro aplikácie využíva niekoľko hlavných modelov. Každý model obsahuje konkrétny fragment, ktorý využíva pre následne vykreslenie. Každý takýto fragment má v sebe definovanú dátovú štruktúru, v ktorej má uložené dáta a príslušné miesto, na ktoré sa má vykresliť.

Model senzora

Senzor obsahuje fragment nazvaný „SensorStatusFragment“, ktorý obsahuje dve položky. Položka pre text a položka pre hodnotu zobrazenú vedľa príslušného názvu.

Model ďalej obsahuje informácie o MQTT téme (na ktorej má „načúvať“ na prichádzajúce dáta), text (ktorý sa má nastaviť pri vykreslení fragmentu), pozíciu (pod ktorú sa má zaradiť) a funkciu pre jeho vykreslenie viď. 6.7.

Pre overenie, či sú dané hodnoty aktuálne existuje špeciálny typ modelu senzora „TimeSensor“. Tento model namiesto priameho vypísania dát z príslušnej témy preloží hodnotu podľa štandardného ISO zápisu pre dátum a čas a porovná ho s aktuálnym časom. Následne vypíše rozdiel medzi dvoma príslušnými dátumami.

```
/***
 * Create new relative layout under the set placeholder at the last position
 * @param instance parent fragment instance
 * @param placeHolder RelativeLayout where sensor should be placed in.
 * @param act UI thread activity
 */
public void drawSensor(Fragment instance, RelativeLayout placeHolder, Activity act) {
    Context context = instance.getContext();

    RelativeLayout fl = new RelativeLayout(context);
```

```
    RelativeLayout.LayoutParams flp = new RelativeLayout.LayoutParams(
            RelativeLayout.LayoutParams.MATCH_PARENT,
            RelativeLayout.LayoutParams.WRAP_CONTENT
    );
    View lastView = placeHolder.getChildAt(placeHolder.getChildCount() - 1);
    if (lastView != null)
        flp.addRule(BELOW, lastView.getId());
    fl.setLayoutParams(flp);
    fl.setId(sensorCount++);
//        Adding the RelativeLayout to the placeholder as a last child
    placeHolder.addView(fl);

    sensorFragment = new SensorStatusFragment();
//        Must be defined in strings for language translation
    FragmentTransaction transaction =
        instance.getChildFragmentManager().beginTransaction();
    transaction.replace(fl.getId(), sensorFragment);
    transaction.commit();
    sensorFragment.setText(MainActivity.getResourcebyId(this.sensorText));
    sensorFragment.setPostfix(postfix);
    act.runOnUiThread(() -> sensorFragment.setStatus(sensorStatus));
}
```

Výpis 6.7: Funkcia pre vykreslenie senzora do fragmentu pod špecifickú kategóriu

Model relé

Model relé obsahuje v sebe dve funkcie znázornené v 6.8. Prvá slúži pre publikáciu stavu relé. Druhá funkcia slúži ako tzv. „event listener“ pre zmenu stavu prepínača, ktorá zaistí odoslanie nového stavu ihneď po prepnutí tlačidla.

```
/***
 * Publish relay state to mqtt topic
 */
private void publish() {
    String msg;
    try {
        ObservableMqttClient mqttClient = IMqtt.getInstance().getClient();
        if (!mqttClient.isConnected()) {
            pushToast("Client not connected");
            return;
        }
        List<Relay> list = new ArrayList<>();
        list.add(this);
        msg = new Gson().toJson(list);
        PublishMessage message = PublishMessage.create(msg.getBytes(), 1, false);
        mqttClient.publish(this.topic, message).subscribe();
        notify_subs = true;
        Log.d("Trigger", msg);
    } catch (MqttException e) {
        e.printStackTrace();
    }
}

/***
 * Switch listener, sets new state to the switch and publishes it
 * @param compoundButton button event
```

```
27  * @param b new state
28  */
29 @Override
30 public void onCheckedChanged(CompoundButton compoundButton, boolean b) {
31     if (compoundButton.isPressed()) {
32         if (b) {
33             this.setState(true);
34         } else {
35             this.setState(false);
36         }
37         this.publish();
38     }
39 }
```

Výpis 6.8: Funkcie využívane pri zmene stavu prepínača, zaisťujú odoslanie na MQTT broker

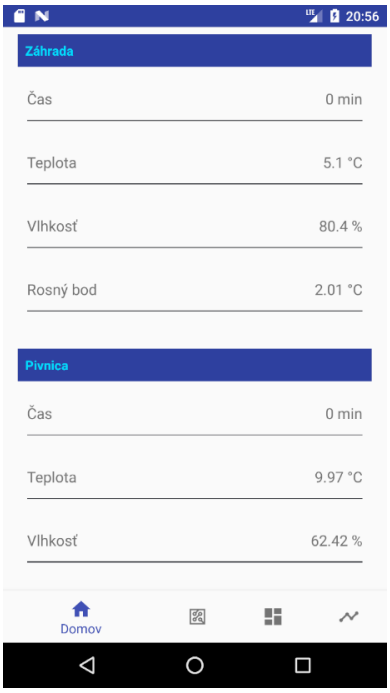
Model grafu

Model grafu sa od ostatných modelov líši tým, že po vykreslení vytvorí proces, ktorý načíta na pozadí dáta. Príslušné dáta načíta z REST-ového volania.

Hlavné fragmenty

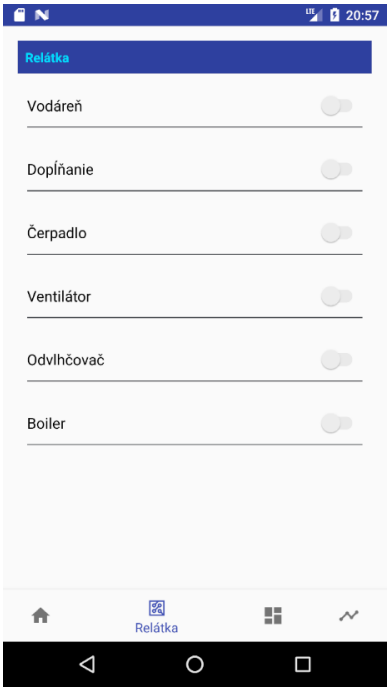
Aplikácia sa skladá zo štyroch hlavných fragmentov:

- **HomeFragment** - zobrazí sa hneď po spustení aplikácie. Obsahuje modely jednotlivých senzorov.



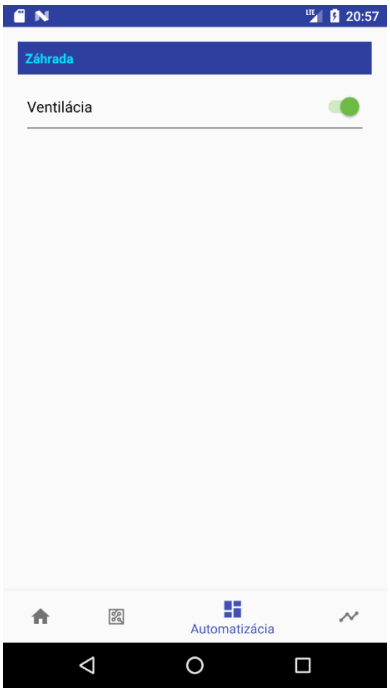
Obr. 6.7: Prehľad informácií z definovaných senzorov.

- **RelayFragment** - zobrazí modely všetkých relé so spoločnou MQTT témou.



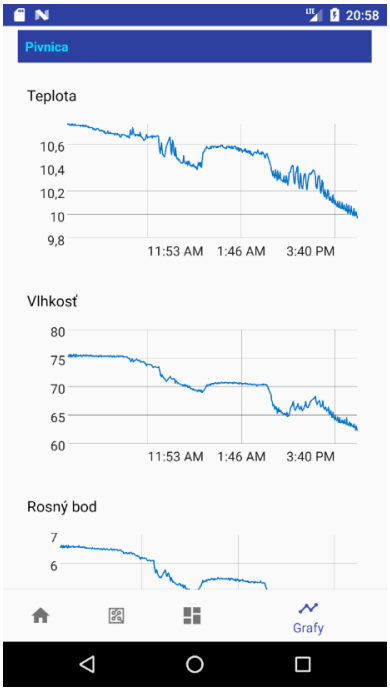
Obr. 6.8: Zobrazenie modifikovateľných položiek systému.

- **AutomationFragment** - nastavenia pre špeciálne podmienky systému.



Obr. 6.9: Zobrazenie špeciálnych nastavení systému.

- **GraphFragment** - vykreslenie jednotlivých grafov.



Obr. 6.10: Zobrazenie historických dát zo senzorov.

Jednotlivé fragmenty obsahujú zoznam modulov, funkcie pre prihlásenie/odhlásenie jednotlivých modulov k ich pridelenej téme na MQTT broker a funkciu pre vykreslenie všetkých modulov zo zoznamu.

HomeFragment

Pri vytvorení tohto fragmentu dochádza k inicializácií všetkých senzorov viď. 6.9. Pre inicializáciu senzoru je potrebné definovať inicializačné parametre:

1. **topic** - téma pre odoberanie dát z MQTT brokeru.
2. **sensorText** - nastaví premennú, ktorej hodnota bude vypísaná pre užívateľa ako názov daného senzora (musí byť preddefinovaná v súbore „strings.xml“ pre preloženie do iných jazykov).
3. **layout** - identifikácia sekcie do ktorej bude senzor priradený.
4. **postfix** - jednotky zapísané za hodnotou.

```
private List<Sensor> sensors;
public HomeFragment() {
    sensors = new ArrayList<>();

//          sensorText must be defined in strings.xml for language translation
    sensors.add(new Sensor("sensor/podtatranskeho/temp", "temp", "home", "°C"));
    sensors.add(new TimeSensor("sensor/garden/time", "time", "garden", "min"));
    sensors.add(new Sensor("sensor/attic/temp", "temp", "attic", "°C"));
//          ...
```

```
}
```

Výpis 6.9: Inicializácia Home fragmentu

Následne sa spustí funkcia „subscribeSensors(),“ ktorá sa prihlási na hromadný odber dát pre všetky senzory z dôvodu zjednodušenia odberu dát.

```
public void subscribeSensors() {
    try {
        ObservableMqttClient mqttClient = IMqtt.getInstance().getClient();
        MainActivity.COMPOSITE_DISPOSABLE.add(
                mqttClient.subscribe("sensor/#", 0).subscribe(msg -> {
                    for (Sensor sensor : sensors) {
                        if (sensor.getTopic().equals(msg.getTopic())) {
                            sensor.setSensorStatus(new String(msg.getPayload()));
                        }
                    }
                })
        );
    } catch (MqttException e) {
        e.printStackTrace();
    }
}
```

Výpis 6.10: Prihlásenie na hromadný odber dát z MQTT brokeru pre senzory

RelayFragment

Inicializácia tohto fragmentu prebieha obdobne ako u senzorov. Rozdiel však tvorí funkcia pre prihlásenie k odberu dát viď. 6.11. Namiesto hromadného odberu z viacerých tém sa pre prenos dát o jednotlivých prepínačoch posiela informácia v rámci jednej témy. V jednej správe sú odoslané informácie o stave všetkých prepínačoch a na strane MQTT brokeru je nastavený „retain,“ čím zaistíme uloženie posledného stavu všetkých prepínačov, ako pre android aplikáciu, tak pre zariadenie, ktoré bude ovládať príslušné relé.

Pri zmene stavu prepínača je taktiež nastavená potvrdzujúca správa pre užívateľa, ktorá sa vypíše v prípade odpovede zo strany zariadenia.

```
public void subscribeRelays() {
    try {
        ObservableMqttClient mqttClient = IMqtt.getInstance().getClient();
        MainActivity.COMPOSITE_DISPOSABLE.add(
                mqttClient.subscribe("relay", 1).subscribe(msg -> {
                    List<Relay> relays;
                    JSONArray json = new JSONArray(new String(msg.getPayload()));
                    if (json.length() > 0) {
                        Type collectionType = new TypeToken<List<Relay>>() {
                        }.getType();
                        relays = new Gson().fromJson(json.toString(), collectionType);
                    } else {
                        return;
                    }
                    for (Relay inRelay : relays) {
                        for (Relay relay : this.relays) {
                            if (inRelay.getRelayName().compareTo(relay.getRelayName()) == 0) {
                                if (getActivity() != null && relay.isNotify_subs()) {
```

```
19                                  relay.pushToast((inRelay.isState() ? getString(R.string.relay_enabled) :
      getString(R.string.relay_disabled)));
20                                  relay.unsetNotify_subs();
21                              }
22                              relay.setState(inRelay.isState());
23                          }
24                      }
25                  }
26              })
27      );
28  } catch (MqttException e) {
29      e.printStackTrace();
30  }
31 }
```

Výpis 6.11: Prihlásenie na odber dát pre fragment, ktorý ovláda relé

Jadro aplikácie

Pri spustení aplikácie sa vytvorí hlavná aktivita aplikácie a vykonajú sa dve hlavné funkcie aplikácie viď. 6.12. Funkcia „onCreate()“, ktorá spôsobí vytvorenie fragmentov podľa jednotlivých náhľadov a vykreslí menu pre jednoduchú navigáciu medzi fragmentami. Následne sa spustí funkcia „onResume()“, ktorá pripojí aplikáciu k MQTT brokeru viď. 6.13 a prihlási jednotlivé fragmenty k odberu dát podľa jednotlivých MQTT tém, definovaných pri vytvorení modelov. Obnovovanie a zasielanie hodnôt riadia jednotlivo vytvorené fragmenty.

```
1  @Override
2  protected void onCreate(Bundle savedInstanceState) {
3      super.onCreate(savedInstanceState);
4      act = this;
5      JodaTimeAndroid.init(this);
6      setContentView(R.layout.activity_main);
7
8  //      Object storing RX functions to prevent garbage collection
9      COMPOSITE_DISPOSABLE = new CompositeDisposable();
10
11 //      Create fragments
12     homeFragment = new HomeFragment();
13     relayFragment = new RelayFragment();
14     automationFragment = new AutomationFragment();
15     graphFragment = new GraphFragment();
16 //      Set home fragment
17     setFragment(homeFragment);
18
19 //      Create navigation menu
20     navigation = (BottomNavigationView) findViewById(R.id.navigation);
21     navigation.setOnNavigationItemSelectedListener(mOnNavigationItemSelectedListener);
22
23 }
24
25 @Override
26 protected void onResume() {
27     super.onResume();
28     mqttConnect();
29 }
```

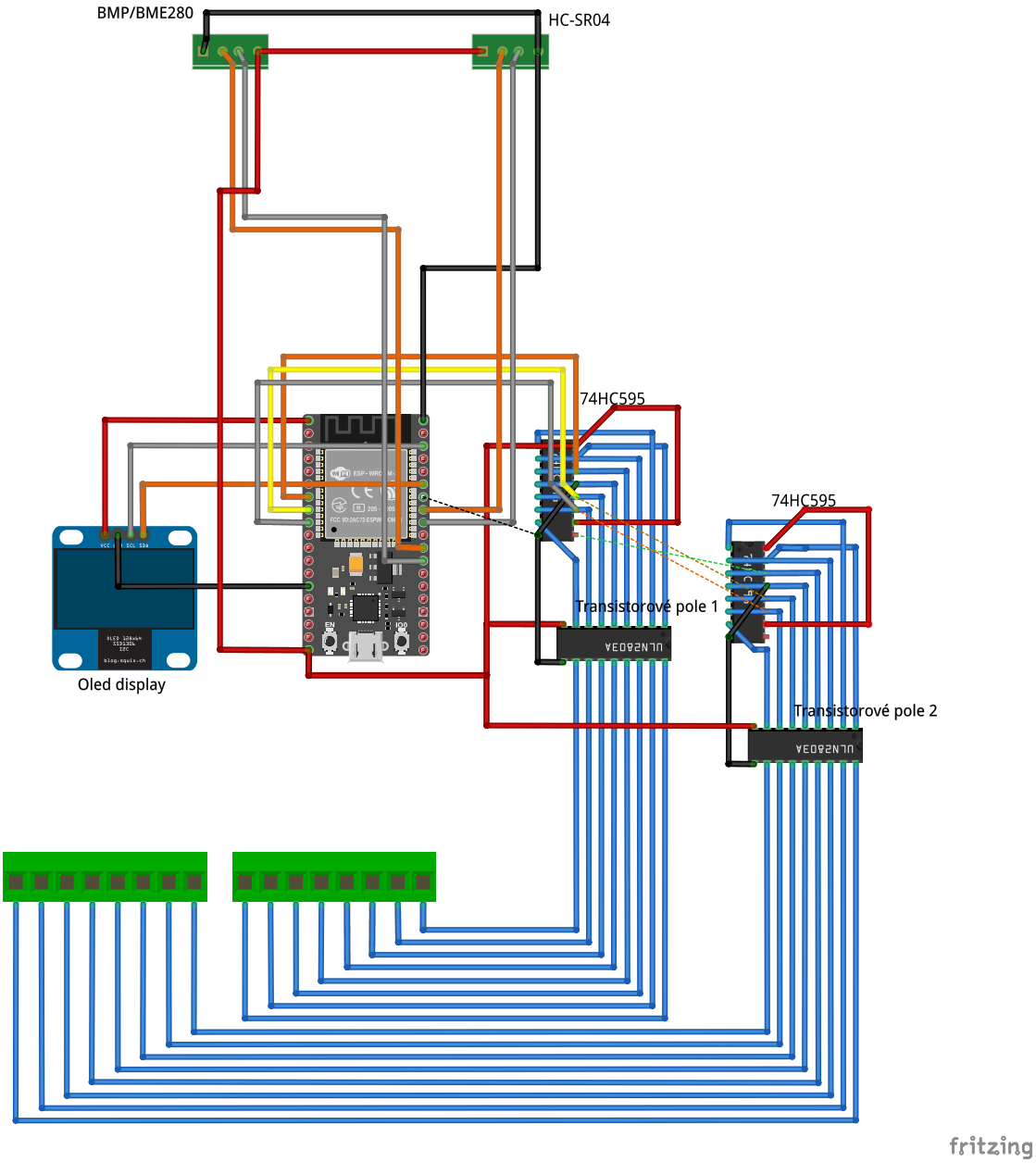
Výpis 6.12: Funkcie využívané po spustení aplikácie.

```java
/***
 * Connect to the MQTT broker, function uses global variable COMPOSITE_DISPOSABLE
 * in order to store RX callback function.
 */
private void mqttConnect() {
    try {
        mqttClient = IMqtt.getInstance().getClient();
        if (mqttClient.isConnected()) {
            Log.i("MQTT", getString(R.string.stat_already_connected));
            return;
        }
        if (!connectionLock.tryAcquire(3, TimeUnit.SECONDS)) {
            pushToast(getString(R.string.stat_err));
            return;
        }
        COMPOSITE_DISPOSABLE.add(
                mqttClient.connect().subscribe(() -> {
                    connectionLock.release();
                    pushToast(getString(R.string.stat_conn));
                    homeFragment.subscribeSensors();
                    relayFragment.subscribeRelays();
                    automationFragment.subscribeRelays();
                }, e -> {
                    connectionLock.release();
                    if (e.getCause() != null) {
                        pushToast(e.getCause().getLocalizedMessage());
                    } else {
                        pushToast(getString(R.string.stat_err));
                    }
                })
        );
    } catch (MqttException e) {
        pushToast(e.getMessage());
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
}
```

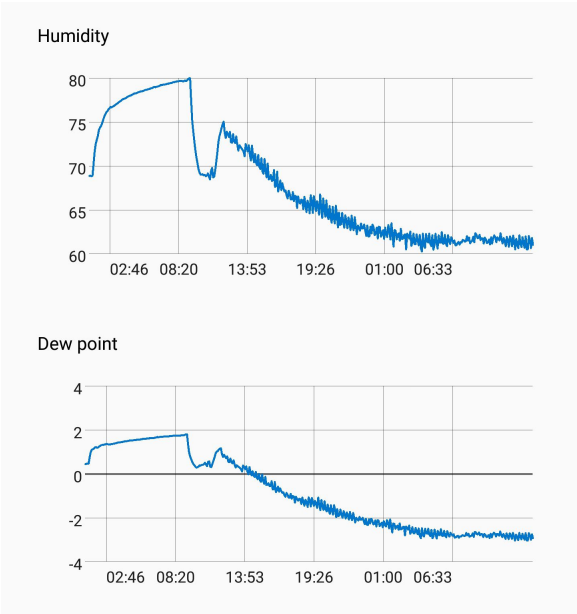
Výpis 6.13: Funkcia pre pripojenie aplikácie k MQTT brokeru.

6.5 Testovanie demoaplikácie

Pred zavedením tohto systému bola miestnosť takmer nevyužiteľná. Dlhotrvajúca vysoká vlhkosť a časté vytápanie vytvárali nebezpečné prostredie pre uskladnený materiál. Po zavedení systému automatického odvetrávania došlo k výraznej zmene podmienok v danej miestnosti. Systém dokáže udržať dlhotrvajúce stabilné podmienky ako je možné vidieť na grafe 6.12.



Obr. 6.11: Návrh schémy zapojenia ovládacej dosky



Obr. 6.12: Stav po zavedení systému

Kapitola 7

Záver

Cieľom tejto práce bolo vytvorenie systému pre správu zariadení s existujúcim operačným systémom na báze interpretovaného jazyka MicroPython a jeho následná praktická aplikácia za účelom ochrany majetku, zlepšenia podmienok, alebo zníženia nákladov na údržbu.

Hlavný prínos tejto práce spočíva v jednoduchom ovládaní jednotlivých uzlov cez navrhnutý a implementovaný administrátorský webový systém.

Funkčnosť tohto systému je prezentovaná na demoaplikáci, ktorej úlohou je automatické odvetrávanie pivničných priestorov a jednoduché ovládanie aktívnych prvkov na diaľku. Všetky namerané dáta zo senzorov sú dostupné vo vytvorenej mobilnej aplikácií, vrátane historických údajov vo forme grafov. Pre prezentovanie údajov zo systému nie je nutné využiť navrhnutú aplikáciu. Systém je možné jednoducho doplniť o iné, už hotové aplikácie pre zobrazovanie a ukladanie dát, ako napríklad Home Assistant alebo Domoticz.

Tento systém by bolo v budúcnosti vhodné doplniť o automatickú analýzu nazbieraných údajov a prípadné upozornenia užívateľa(napr. formou notifikácie) o vzniknutých problémoch, alebo výraznejších odchýlkach(anomáliách) pri zaznamenávaní údajov.

Pri návrhu mikro aplikácií pre MPOS som sa často stretol s problémom fragmentácie pamäte, kvôli čomu museli byť aplikácie výrazne upravené. Pre rozsiahlejšie použitie by som odporúčal aplikácie rozdeliť na čo najväčší počet uzlov.

Literatúra

[1] Alltopnotch: *ULN2003AN DIP16 DARLINGTON TRANSISTOR ARRAY*. [Online; navštíveno 22.04.2019].
URL https://alltopnotch.co.uk/product/5-x-uln2003an-darlington-transistor-array/

[2] Components, R.: *11 Internet of Things (IoT) Protocols You Need to Know About*. [Online; navštíveno 09.03.2019].
URL https://www.rs-online.com/designspark/eleven-internet-of-things-iot-protocols-you-need-to-know-about

[3] Deshmukh, A. V.: *Microcontrollers [Theory and applications]*. Tata McGraw-Hill Publishing Company Limited, 2005, ISBN 0-07-058595-4, 5-7 s.

[4] Dočekal, D. B.: *Co je to Bluetooth LE, BLE, Bluetooth Low Energy, Bluetooth Smart, Bluetooth 5?* [Online; navštíveno 09.03.2019].
URL https://365tipu.cz/2017/09/06/tip883-co-je-to-bluetooth-le-ble-bluetooth-low-energy-bluetooth-smart-bluetooth-5/

[5] Drahovský, P.: *Rekonfigurovatelný IoT uzel na bázi ESP8266/ESP32*. 2018, 16-19 s.
URL https://www.vutbr.cz/www_base/zav_prace_soubor_verejne.php?file_id=180670

[6] Drahovský, P.: *Rekonfigurovatelný IoT uzel na bázi ESP8266/ESP32*. 2018, 25-27 s.
URL https://www.vutbr.cz/www_base/zav_prace_soubor_verejne.php?file_id=180670

[7] E-konstruktér: *Co jsou elektromagnetické ventily?* [Online; navštíveno 09.04.2019].
URL https://e-konstrukter.cz/novinka/co-jsou-elektromagneticke-ventily

[8] Espressif: *ESP32 datasheet*. [Online; navštíveno 22.04.2019].
URL http://espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf

[9] Espressif: *ESP32 Modules and Boards*. [Online; navštíveno 10.04.2019].
URL https://docs.espressif.com/projects/esp-idf/en/latest/hw-reference/modules-and-boards.html

[10] Espressif: *ESP32 overview*. [Online; navštíveno 22.04.2019].
URL https://www.espressif.com/en/products/hardware/esp32/overview

[11] Faltus, R.: *Spinací prvky, rele, tranzistory a tranzistorová pole*. [Online; navštíveno 09.04.2019].

URL https://navody.arduino-shop.cz/zaciname-s-arduinem/spinaci-prvky-rele-tranzistory-a-tranzistorova-pole.html

[12] Hivemq: *MQTT Essentials*. [Online; navštíveno 10.04.2019].
URL https://www.hivemq.com/tags/mqtt-essentials/

[13] Kerns, M.: *AC Vs. DC Solenoids and How They Work*. [Online; navštíveno 08.03.2019].
URL https://sciencing.com/dc-solenoids-how-they-work-5008123.html

[14] Kilián, K.: *Bluetooth 5: jaké jsou největší výhody proti starší verzi 4.2?* [Online; navštíveno 09.03.2019].
URL https://www.svetandroida.cz/bluetooth-5/

[15] Koziorek, J.; Kocián, J.; Chromčák, L.; aj.: *Distribuované systémy řízení*. VŠB-TU Ostrava, 2011, ISBN 978-80-248-2599-1, str. 10.

[16] Lallanilla, M.: *What Is Dew Point?* [Online; navštíveno 23.02.2019].
URL https://www.livescience.com/43269-what-is-dew-point.html

[17] Lee, J.-S.; Su, Y.-W.; Shen, C.-C.: *A Comparative Study of Wireless Protocols: Bluetooth, UWB, ZigBee, and Wi-Fi*. [Online; navštíveno 09.03.2019].
URL
http://eee.guc.edu.eg/Announcements/Comparaitive_Wireless_Standards.pdf

[18] Luboš, M.: *Měřič vzdálenosti ultrazvukový*. [Online; navštíveno 09.04.2019].
URL https://navody.arduino-shop.cz/navody-k-produktum/meric-vzdalenosti-ultrazvukovy.html

[19] Luboš, M.: *Senzor teploty a vlhkosti HTU21D I2C*. [Online; navštíveno 07.04.2019].
URL https://navody.arduino-shop.cz/navody-k-produktum/senzor-teploty-a-vlhkosti-htu21d-i2c.html

[20] Luboš, M.: *Teplotní senzor DS18B20*. [Online; navštíveno 17.03.2019].
URL https://navody.arduino-shop.cz/navody-k-produktum/teplotni-senzor-ds18b20.html

[21] Luisllamas: *MEDIR TEMPERATURA Y HUMEDAD CON ARDUINO Y SENSOR DHT11-DHT22*. [Online; navštíveno 07.04.2019].
URL https://www.luisllamas.es/arduino-dht11-dht22/

[22] Madsen, T. M.: *Home Automation Systems Integration*. [Online; navštíveno 13.04.2019].
URL https://projekter.aau.dk/projekter/files/61078036/1275905809.pdf

[23] Malý, M.: *Protokol MQTT: komunikační standard pro IoT*. [Online; navštíveno 10.04.2019].
URL
https://www.root.cz/clanky/protokol-mqtt-komunikacni-standard-pro-iot/

[24] Malý, M.: *REST: architektura pro webové API*. [Online; navštíveno 10.04.2019].
URL https://www.zdrojak.cz/clanky/rest-architektura-pro-webove-api/

[25] Mavridis, N.; Pierris, G.; BenAbdelkader, C.; aj.: *Smart buildings and the human-machine cloud*. IEEE, 2015, ISBN 978-1-4799-8422-0.

[26] Müllner, D.: *Dew point ventilation controller*. [Online; navštíveno 23.02.2019]. URL http://danifold.net/fancontrol.html

[27] Nasir, S. Z.: *Basics of 74HC595*. [Online; navštíveno 09.04.2019]. URL https://www.rs-online.com/designspark/basics-of-74hc595

[28] Nicholson, J.: *How Does a Solenoid Work?* [Online; navštíveno 08.03.2019]. URL https://sciencing.com/a-solenoid-work-4567178.html

[29] Ondráček, P.: *Úsporné zabezpečovací zařízení s modulem (Espressif) ESP8266/ESP32*. 2018, 22 s. URL https://www.vutbr.cz/www_base/zav_prace_soubor_verejne.php?file_id=174026

[30] Otýpka, I. B. M.: *Sekvenční logické obvody II*. [Online; navštíveno 09.04.2019]. URL https://coptkm.cz/portal/reposit.php?action=0&id=8672

[31] Parekh, J.: *WiFi's evolving role in IoT*. [Online; navštíveno 09.03.2019]. URL https://www.networkworld.com/article/3196191/wifi-s-evolving-role-in-iot.html

[32] Rahman, A. B. A.: *Comparison of Internet of Things (IoT) Data Link Protocols*. [Online; navštíveno 08.03.2019]. URL https://www.cse.wustl.edu/~jain/cse570-15/ftp/iot_dlc.pdf

[33] Richta, T.; Janoušek, V.: *Operating System for Petri Nets-Specified Reconfigurable Embedded Systems*. Tata McGraw-Hill Publishing Company Limited, 2013, ISBN 978-3-642-53855-1, 444-451 s.

[34] root.cz: *ESP32 je tu. Co přinese nástupce ESP8266*. [Online; navštíveno 22.04.2019]. URL https://www.root.cz/clanky/esp32-je-tu-co-prinese-nastupce-esp8266/

[35] Sensirion: *Sensirion Humidity & Temperature Sensmitter SHT1x/SHT7x*. [Online; navštíveno 16.03.2019]. URL http://coecsl.ece.illinois.edu/ge423/sensorprojects/sensirion111.pdf

[36] Soselectronic: *Internet of Things (3.část) – Technologie pro bezdrátový přenos dat*. [Online; navštíveno 08.03.2019]. URL https://www.soselectronic.cz/articles/no-name/internet-of-things-3-cast-technologie-pro-bezdratovy-prenos-dat-2044

[37] Techopedia: *Internet of Things (IoT)*. [Online; navštíveno 10.03.2019]. URL https://www.techopedia.com/definition/28247/internet-of-things-iot

[38] Tweaking4all: *Arduino and DS18B20 Temperature Sensor*. [Online; navštíveno 22.04.2019]. URL https://www.tweaking4all.com/hardware/arduino/arduino-ds18b20-temperature-sensor/

[39] Udayashankara, V.; Mallikarjunaswamy, M. S.: *8051 Microcontroller: Hardware, Software & Applications*. Tata McGraw-Hill Publishing Company Limited, 2009, ISBN 0-07-008681-8, 6-7 s.

Príloha A

Zdrojové kódy použité v moduloch

```
class Cellar_Sensor(Task):
 async def loop(self):
   if len(self.params) < 3:
     super().send("Error","Error in initialisation. Deactivating")
     super().stop()

   out = self.params[2]
   if (self.params[2] == 'HCSR04'):
     self.sensor = self.HCSR04(trigger_pin=19, echo_pin=18)
   elif (self.params[2] == 'BMP280'):
     from machine import I2C
     from machine import Pin
     bus = I2C(scl = Pin(17), sda = Pin(16))
     self.bmp = self.BME280(i2c=bus)
     self.tmp_str = 'temperature'
     self.hum_str = 'humidity'
     self.dew_str = 'dew_point'
     del Pin

   while self.running:
     if (self.params[2] == 'HCSR04'):
       self.distance = self.sensor.distance_cm()
       super().sendtoTopic(self.params[3], round(float(self.params[4]) - self.distance, 2), retain = True)
     elif (self.params[2] == 'BMP280'):
       self.values = self.bmp.values
       self.values[self.dew_str] = round(self.get_dew_point_c(self.values[self.tmp_str], self.values[self.
      hum_str]), 2)
       super().sendtoTopic(self.params[3], self.values[self.tmp_str], retain = True)
       super().sendtoTopic(self.params[4], self.values[self.hum_str], retain = True)
       super().sendtoTopic(self.params[5], self.values[self.dew_str], retain = True)

       super().sendtoTopic(self.params[6], [self.values[self.tmp_str],self.values[self.dew_str]])
       super().sendtoTopic(self.params[7], [self.values[self.hum_str]])
     gc.collect()
     await asyncio.sleep(10)

 def get_dew_point_c(self, t_air_c, rel_humidity):
   import math
   A = 17.27
   B = 237.7
   alpha = ((A * t_air_c) / (B + t_air_c)) + math.log(rel_humidity/100.0)
   del math
```

```
  return (B * alpha) / (A - alpha)

class HCSR04:
  __version__ = "0.2.0"
  __author__ = "Roberto Sanchez"
  __license__ = "Apache License 2.0. https://www.apache.org/licenses/LICENSE-2.0"
  # echo_timeout_us is based in chip range limit (400cm)
  def __init__(self, trigger_pin, echo_pin, echo_timeout_us=500*2*30):
    from machine import Pin
    self.echo_timeout_us = echo_timeout_us
    # Init trigger pin (out)
    self.trigger = Pin(trigger_pin, mode=Pin.OUT, pull=None)
    self.trigger.value(0)

    # Init echo pin (in)
    self.echo = Pin(echo_pin, mode=Pin.IN, pull=None)

  def _send_pulse_and_wait(self):
    from time import sleep_us
    self.trigger.value(0) # Stabilize the sensor
    sleep_us(5)
    self.trigger.value(1)
    # Send a 10us pulse.
    sleep_us(10)
    self.trigger.value(0)
    del sleep_us
    try:
      pulse_time = machine.time_pulse_us(self.echo, 1, self.echo_timeout_us)
      return pulse_time
    except OSError as ex:
      if ex.args[0] == 110: # 110 = ETIMEDOUT
          raise OSError('Out of range')
      raise ex

  def distance_mm(self):
    pulse_time = self._send_pulse_and_wait()
    mm = pulse_time * 100 // 582
    return mm

  def distance_cm(self):
    pulse_time = self._send_pulse_and_wait()
    cms = (pulse_time / 2) / 29.1
    return cms

# prevzate z https://github.com/catdog2/mpy_bme280_esp8266
class BME280:
  # BME280 default address.
  BME280_I2CADDR = 0x76

  # Operating Modes
  BME280_OSAMPLE_1 = 1
  BME280_OSAMPLE_2 = 2
  BME280_OSAMPLE_4 = 3
  BME280_OSAMPLE_8 = 4
  BME280_OSAMPLE_16 = 5

  BME280_REGISTER_CONTROL_HUM = 0xF2
  BME280_REGISTER_CONTROL = 0xF4

```

```
  def __init__(self, mode=1, address=0x76, i2c=None, **kwargs):
    from array import array
    from ustruct import unpack, unpack_from
      # Check that mode is valid.
    if mode not in [self.BME280_OSAMPLE_1, self.BME280_OSAMPLE_2, self.
     BME280_OSAMPLE_4, self.BME280_OSAMPLE_8, self.BME280_OSAMPLE_16]:
      raise ValueError(
          'Unexpected mode value {0}. Set mode to one of '
          'BME280_ULTRALOWPOWER, BME280_STANDARD, BME280_HIGHRES, or '
          'BME280_ULTRAHIGHRES'.format(mode))
    self._mode = mode
    self.address = address
    if i2c is None:
      raise ValueError('An I2C object is required.')
    self.i2c = i2c

    # load calibration data
    dig_88_a1 = self.i2c.readfrom_mem(self.address, 0x88, 26)
    dig_e1_e7 = self.i2c.readfrom_mem(self.address, 0xE1, 7)
    self.dig_T1, self.dig_T2, self.dig_T3, self.dig_P1, \
        self.dig_P2, self.dig_P3, self.dig_P4, self.dig_P5, \
        self.dig_P6, self.dig_P7, self.dig_P8, self.dig_P9, \
        _, self.dig_H1 = unpack("<HhhHhhhhhhhhBB", dig_88_a1)

    self.dig_H2, self.dig_H3 = unpack("<hB", dig_e1_e7)
    e4_sign = unpack_from("<b", dig_e1_e7, 3)[0]
    self.dig_H4 = (e4_sign << 4) | (dig_e1_e7[4] & 0xF)

    e6_sign = unpack_from("<b", dig_e1_e7, 5)[0]
    self.dig_H5 = (e6_sign << 4) | (dig_e1_e7[4] >> 4)

    self.dig_H6 = unpack_from("<b", dig_e1_e7, 6)[0]

    self.i2c.writeto_mem(self.address, self.BME280_REGISTER_CONTROL,
                         bytearray([0x3F]))
    self.t_fine = 0

    # temporary data holders which stay allocated
    self._l1_barray = bytearray(1)
    self._l8_barray = bytearray(8)
    self._l3_resultarray = array("i", [0, 0, 0])

  def read_raw_data(self, result):
    from time import sleep_us
    self._l1_barray[0] = self._mode
    self.i2c.writeto_mem(self.address, self.BME280_REGISTER_CONTROL_HUM,
                         self._l1_barray)
    self._l1_barray[0] = self._mode << 5 | self._mode << 2 | 1
    self.i2c.writeto_mem(self.address, self.BME280_REGISTER_CONTROL,
                         self._l1_barray)

    sleep_time = 1250 + 2300 * (1 << self._mode)
    sleep_time = sleep_time + 2300 * (1 << self._mode) + 575
    sleep_time = sleep_time + 2300 * (1 << self._mode) + 575
    sleep_us(sleep_time)  # Wait the required time
    del sleep_us

    # burst readout from 0xF7 to 0xFE, recommended by datasheet
    self.i2c.readfrom_mem_into(self.address, 0xF7, self._l8_barray)
```

```
    readout = self._l8_barray
    # pressure(0xF7): ((msb << 16) | (lsb << 8) | xlsb) >> 4
    raw_press = ((readout[0] << 16) | (readout[1] << 8) | readout[2]) >> 4
    # temperature(0xFA): ((msb << 16) | (lsb << 8) | xlsb) >> 4
    raw_temp = ((readout[3] << 16) | (readout[4] << 8) | readout[5]) >> 4
    # humidity(0xFD): (msb << 8) | lsb
    raw_hum = (readout[6] << 8) | readout[7]

    result[0] = raw_temp
    result[1] = raw_press
    result[2] = raw_hum

  def read_compensated_data(self, result=None):
    from array import array
    self.read_raw_data(self._l3_resultarray)
    raw_temp, raw_press, raw_hum = self._l3_resultarray
    # temperature
    var1 = ((raw_temp >> 3) - (self.dig_T1 << 1)) * (self.dig_T2 >> 11)
    var2 = (((((raw_temp >> 4) - self.dig_T1) *
             ((raw_temp >> 4) - self.dig_T1)) >> 12) * self.dig_T3) >> 14
    self.t_fine = var1 + var2
    temp = (self.t_fine * 5 + 128) >> 8
    # pressure
    var1 = self.t_fine - 128000
    var2 = var1 * var1 * self.dig_P6
    var2 = var2 + ((var1 * self.dig_P5) << 17)
    var2 = var2 + (self.dig_P4 << 35)
    var1 = (((var1 * var1 * self.dig_P3) >> 8) +
            ((var1 * self.dig_P2) << 12))
    var1 = (((1 << 47) + var1) * self.dig_P1) >> 33
    if var1 == 0:
      pressure = 0
    else:
      p = 1048576 - raw_press
      p = (((p << 31) - var2) * 3125) // var1
      var1 = (self.dig_P9 * (p >> 13) * (p >> 13)) >> 25
      var2 = (self.dig_P8 * p) >> 19
      pressure = ((p + var1 + var2) >> 8) + (self.dig_P7 << 4)
    # humidity
    h = self.t_fine - 76800
    h = (((((raw_hum << 14) - (self.dig_H4 << 20) -
            (self.dig_H5 * h)) + 16384)
          >> 15) * (((((((h * self.dig_H6) >> 10) *
                        (((h * self.dig_H3) >> 11) + 32768)) >> 10) +
                      2097152) * self.dig_H2 + 8192) >> 14))
    h = h - (((((h >> 15) * (h >> 15)) >> 7) * self.dig_H1) >> 4)
    h = 0 if h < 0 else h
    h = 419430400 if h > 419430400 else h
    humidity = h >> 12

    if result:
      result[0] = temp
      result[1] = pressure
      result[2] = humidity
      return result

    return array("i", (temp, pressure, humidity))

  @property
```

```
218     def values(self):
219       t, p, h = self.read_compensated_data()
220 
221       p = p // 256
222       pi = p // 100
223       pd = p − pi ∗ 100
224 
225       hi = h // 1024
226       hd = h ∗ 100 // 1024 − hi ∗ 100
227       ret = {}
228       ret['temperature'] = round((t/100),2)
229       ret['humidity'] = round(float(str(hi) + '.' + str(hd)),2)
230       ret['pressure'] = round(float(str(pi) + '.' + str(pd)),2)
231       return ret
```

Výpis A.1: Zdrojový kód aplikácie Cellar_Sensors

```
1  class Relay(Task):
2    def setRelays(self, value):
3      self.latch.value(0)
4      self.reg.write(bytes([value]))
5      self.latch.value(1)
6 
7      if len(self.params) > 2:
8        super().sendtoTopic(self.params[2], self.relays, retain=True)
9 
10   def recieve(self, port, payload):
11     import ujson
12     msg = payload[0][1:len(payload[0])−1]
13     msg = msg.replace('[', '{').replace(']', '}')
14     msg = ujson.loads('[' + msg + ']')
15     for newRelay in msg:
16       for relay in self.relays:
17         if relay['Relay'] == newRelay['Relay']:
18           print('oldState = '+ str(relay['State']))
19           print('newState = '+ str(newRelay['State']))
20           print('POSITION = '+ str(relay['Position']))
21           if newRelay['State']:
22             self.relayState = self.relayState | (1 << relay['Position'])
23           else:
24             self.relayState = self.relayState & (~0 ^ (1 << relay['Position']))
25           relay['State'] = newRelay['State']
26     self.setRelays(self.relayState)
27     del ujson
28 
29   async def loop(self):
30     from machine import Pin
31     from machine import SPI
32     self.relayState = 0
33     self.relays = [
34       {
35         'Relay':'Darling',
36         'State': False,
37         'Position': 2
38       },
39       {
40         'Relay':'EVd',
41         'State': False,
42         'Position': 4
```

```
43      },
44      {
45        'Relay':'Pump',
46        'State': False,
47        'Position': 0
48      },
49      {
50        'Relay':'Dryer',
51        'State': False,
52        'Position': 1
53      },
54      {
55        'Relay':'Fan',
56        'State': False,
57        'Position': 3
58      },
59      {
60        'Relay':'Boiler',
61        'State': False,
62        'Position': 5
63      }
64    ]
65    self.latch = Pin(33, Pin.OUT)
66    self.reg = SPI(−1, sck=Pin(25), mosi=Pin(32), miso=Pin(5))
67    self.setRelays(0)
68    while self.running:
69      await asyncio.sleep(100)
```

Výpis A.2: Zdrojový kód aplikácie Relay_driver

```
1 class Supervisor(Task):
2   def recieve(self, port, payload):
3     payload = eval(payload[0])
4     if self.params[2] == 'ventilation':
5       if port == 'cellar':
6         self.cellar['temp'] = payload[0]
7         self.cellar['dew_point'] = payload[1]
8         self.cellar['updated'] = True
9       elif port == 'outside':
10        self.outside['temp'] = payload[0]
11        self.outside['dew_point'] = payload[1]
12        self.outside['updated'] = True
13    elif self.params[2] == 'dryer':
14      if port == 'cellar':
15        self.cellar['hum'] = payload[0]
16        self.cellar['updated'] = True
17
18  def setRelay(self, relay, value):
19    send = [{
20        "Relay":relay,
21        "State":value
22    }]
23    super().sendtoTopic(self.params[3], send)
24
25  async def loop(self):
26    if len(self.params) < 3:
27      super().end()
28    if self.params[2] == 'ventilation' and len(self.params) < 4:
29      super().end()
```

```
30    self.outside = {
31      'temp': None,
32      'hum': None,
33      'dew_point': None,
34      'updated': False
35    }
36    self.cellar = {
37      'temp': None,
38      'hum': None,
39      'dew_point': None,
40      'updated': False
41    }
42    if self.params[2] == 'ventilation':
43      self.setRelay("Fan", False)
44    if self.params[2] == 'dryer':
45      self.setRelay("Dryer", False)
46    while self.running:
47      if self.params[2] == 'ventilation':
48        if self.cellar['updated'] and self.outside['updated']:
49          if self.cellar['dew_point'] >= (self.outside['dew_point'] + 1.2):
50            if self.outside['temp'] >= 5 or self.cellar['temp'] >= 5:
51              self.setRelay("Fan", True)
52              await asyncio.sleep(15*60)
53              self.setRelay("Fan", False)
54              await asyncio.sleep(5*60)
55      if self.params[2] == 'dryer':
56        if self.cellar['updated']:
57          if self.cellar['hum'] > 80:
58            self.setRelay("Dryer", True)
59            await asyncio.sleep(120*60)
60            self.setRelay("Dryer", False)
61      self.cellar['updated'] = False
62      self.outside['updated'] = False
63      await asyncio.sleep(10)
```

Výpis A.3: Zdrojový kód aplikácie Supervisor

```
1  class Attic_Sensor(Task):
2    async def loop(self):
3      if len(self.params) < 3:
4        super().send("Error","Error in initialisation. Deactivating")
5        super().stop()
6
7      out = self.params[2]
8      if (self.params[2] == 'DS18B20'):
9        from ds18x20 import DS18X20
10       from machine import Pin
11       from onewire import OneWire
12       self.ds = DS18X20(OneWire(Pin(14)))
13       addrs = self.ds.scan()
14       if not addrs:
15         raise Exception('no DS18B20 found at bus on pin 14')
16       self.addr = addrs.pop()
17     elif (self.params[2] == 'AM2315'):
18       from machine import I2C
19       from machine import Pin
20       bus = I2C(scl = Pin(5), sda = Pin(4), freq = 20000)
21       self.am = self.AM2315(i2c=bus)
22       del Pin
```

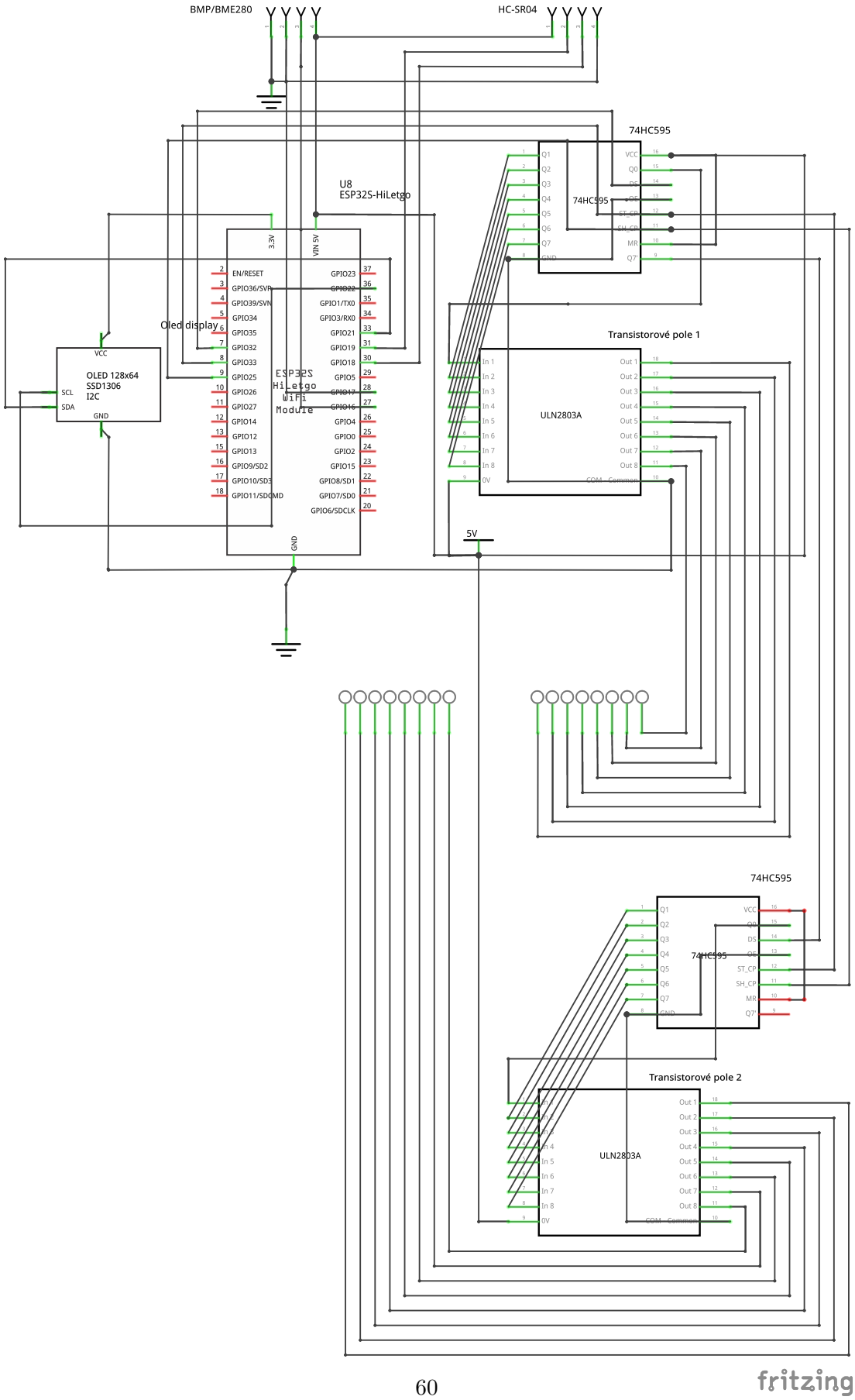
```
  while self.running:
    if (self.params[2] == 'DS18B20'):
      try:
        self.ds.convert_temp()
        await asyncio.sleep(1)
        temp = round(self.ds.read_temp(self.addr),2)
        super().sendtoTopic(self.params[3], temp, retain = True)
      except Exception:
        super().send("Error","Error reading from DS18b20.")
    elif (self.params[2] == 'AM2315'):
      self.am.measure()
      temp = round(self.am.temperature(),2)
      hum = round(self.am.humidity(),2)
      dew = round(self.get_dew_point_c(temp, hum), 2)
      super().sendtoTopic(self.params[3], temp, retain = True)
      super().sendtoTopic(self.params[4], hum, retain = True)
      super().sendtoTopic(self.params[5], dew, retain = True)

      super().sendtoTopic('pnos/message/ESP_30aea48c28f0/Ventilation/outside', [temp,dew])
    gc.collect()
    await asyncio.sleep(10)

def get_dew_point_c(self, t_air_c, rel_humidity):
  import math
  A = 17.27
  B = 237.7
  alpha = ((A * t_air_c) / (B + t_air_c)) + math.log(rel_humidity/100.0)
  del math
  return (B * alpha) / (A - alpha)

# prevzate z https://github.com/mchobby/esp8266-upy/tree/master/am2315
class AM2315:
  AM2315_READREG = 0x03
  def __init__ ( self , i2c, addr = 0x5c ):
    self.addr = addr
    self.rbuf = bytearray (8) # sensor response
    self.i2c = i2c

  def wakeup(self):
      # Wake up the sensor
    try :
      self.i2c.writeto(self.addr, b' ')
      # sensor will never send ACK then OSError will not be raised
    except OSError :
      from time import sleep_ms
      sleep_ms(10)
      del sleep_ms

  def measure(self):
    import time
    self.wakeup()
    # request measure
    wbuf = bytearray(3)
    wbuf[0] = self.AM2315_READREG
    wbuf[1] = 0x00  # Start at adress 0x00
    wbuf[2] = 0x04  # Request 4 byte data
    self.i2c.writeto(self.addr, wbuf) # b '\ x03 \ x00 \ x04')
    # wait 1.5+ ms before reading
```

```
82        time.sleep_ms(2)
83        # Request 8 bytes from sensor
84        # RBUF = ByteArray (8)
85        self.i2c.readfrom_mem_into(self.addr, 0, self.rbuf) # Read from Reg 0
86        del time
87        return self.check_response()
88
89      def check_response(self):
90        if self.rbuf[0] != self.AM2315_READREG:
91          return False
92        if self.rbuf[1] != 4: # Number of bytes requested
93          return False
94        return  True
95
96      def humidity(self):
97        humidity = (self.rbuf[2] * 256) + self.rbuf [3]
98        return humidity / 10
99
100     def temperature(self):
101       temp = self.rbuf[4] & 0x7F
102       temp = (temp * 256) + self.rbuf[5]
103       return temp / 10
```

Výpis A.4: Zdrojový kód aplikácie Attic_Sensors

Príloha B

Schéma zapojenia



Obr. B.1: Finálna schéma zapojenia ovládacej dosky