ZÁPADOČESKÁ UNIVERZITA V PLZNI
FAKULTA ELEKTROTECHNICKÁ

KATEDRA ELEKTROENERGETIKY

DIPLOMOVÁ PRÁCE

Monitorování environmentálních veličin s centrálním zpracováním dat

Bc. Petra Kristová **2021**

ZÁPADOČESKÁ UNIVERZITA V PLZNI
Fakulta elektrotechnická
Akademický rok: 2020/2021

ZADÁNÍ DIPLOMOVÉ PRÁCE
(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Bc. Petra KRISTOVÁ**
Osobní číslo:	**E19N0006K**
Studijní program:	**N2644 Aplikovaná elektrotechnika**
Studijní obor:	**Aplikovaná elektrotechnika**
Téma práce:	**Monitorování environmentálních veličin s centrálním zpracováním dat**
Zadávající katedra:	**Katedra elektroenergetiky**

Zásady pro vypracování

Navrhněte systém pro centrální sběr a zpracování dat ze vzdálených autonomních senzorů environmentálních veličin. Systém bude zahrnovat příklad řešení senzorického uzlu pro sběr daných veličin, nashromážděná data se budou dále zpracovávat v rámci uživatelské aplikace. Při návrhu zohledněte přehledné strukturované uspořádání dat, preferujte stávající prověřené standardy informačních datových sítí.

1. Vytipujte vhodné senzory pro měření environmentálních veličin a navrhněte hardwarové uspořádání autonomního zařízení pro lokální měření vybraných veličin.
2. Zvolte a implementujte vhodný způsob komunikace s přihlédnutím ke stávajícím standardům v datových informačních sítích.
3. Zvolte a navrhněte vhodný způsob ukládání dat, jejich zpracování a vizualizaci s využitím moderních webových technologií.
4. Implementujte software jak pro vzdálený autonomní modul, tak pro uživatelskou aplikaci.

Rozsah diplomové práce: **40 – 60 stran**
Rozsah grafických prací: **podle doporučení vedoucího**
Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam doporučené literatury:

1. Yiu, Joseph. The Definitive Guide to ARM? Cortex?-M3 and Cortex?-M4 Processors, Elsevier Science & Technology, 2013.

Vedoucí diplomové práce: **Ing. Kamil Kosturik, Ph.D.**
Katedra elektroniky a informačních technologií

Datum zadání diplomové práce: **9. října 2020**
Termín odevzdání diplomové práce: **27. května 2021**

L.S.

Prof. Ing. Zdeněk Peroutka, Ph.D.
děkan

Doc. Ing. Karel Noháč, Ph.D.
vedoucí katedry

V Plzni dne 9. října 2020

Abstrakt

Práce pojednává o návrhu systému pro sběr dat environmentálních veličin pomocí měřicích uzlů, přičemž získané hodnoty se dále zpracovávají v rámci uživatelské aplikace. Nejprve jsou zmíněny vybrané IoT technologie, pomocí kterých lze realizovat přenos dat do centrální jednotky. V další části se práce věnuje environmentálním veličinám, které je možné měřit, a několika konkrétním zvoleným senzorům. Dále je popsána architektura celého systému a návrh hardwarového uspořádání měřicího uzlu. Nakonec je implementován firmware senzorického uzlu a uživatelská aplikace pro vizualizaci naměřených dat s využitím moderních webových technologií.

Klíčová slova

Internet věcí, environmentální veličina, senzor, teplota, vlhkost, kvalita ovzduší, mikrokontrolér, cloud, webová aplikace

Abstract

The master thesis presents the design of a system for data collection of environmental variables using measuring nodes. The obtained values are further processed in a user application. First, some selected IoT technologies that can be used to transfer data to the central unit are mentioned. The next part is about environmental variables that can be measured and also about several specific sensors. Furthermore, the architecture of the whole system and the design of the hardware arrangement of the measuring node are described. Finally, the firmware of the sensor node and the user application for visualization of measured data using modern web technologies are implemented.

Key words

Internet of things, environmental variable, sensor, temperature, humidity, air quality, microcontroller, cloud, web application

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracovala samostatně, s použitím odborné literatury a pramenů uvedených v seznamu, který je součástí této diplomové práce.

Dále prohlašuji, že veškerý software, použitý při řešení této diplomové práce, je legální.

Kristová

.....

podpis

V Plzni dne 27.5.2021

Bc. Petra Kristová

Poděkování

Tímto bych ráda poděkovala vedoucímu diplomové práce Ing. Kamilu Kosturikovi, Ph.D. za cenné profesionální rady, připomínky a metodické vedení práce.

Obsah

Úvod

Internet věcí je oblastí, která se v poslední době velmi rozmáhá, což je jedním z důvodů rychlého rozvoje dalších domén, které s tímto pojmem souvisejí. Spadají sem například webové technologie, mobilní aplikace či různá cloudová úložiště pro data přijatá od zařízení v rámci konkrétního systému. Cloudové služby často kromě uložení dat do databáze poskytují i další užitečné funkčnosti jako statistiky přenesených dat, hosting webové uživatelské aplikace, cloudové funkce (např. kontrola příchozích dat, pravidelné odmazání starých dat), strojové učení apod. Další podstatnou oblast, úzce spjatou s internetem věcí, představují senzory pro měření nejrůznějších veličin. I zde dochází k neustálému pokroku a zlepšující se dostupnosti senzorů pro běžné uživatele.

Předmětem diplomové práce je návrh systému pro sběr dat environmentálních veličin pomocí měřicích uzlů s další možností zpracování hodnot v rámci uživatelské aplikace. V zařízení pro sběr dat jsou použity senzory pro měření několika veličin, které přímo souvisí s kvalitou životního prostředí v obytných prostorech – teplota, relativní vlhkost a kvalita ovzduší (koncentrace CO_2 a VOC – těkavých organických látek). Získané hodnoty jsou dále odesílány do centrálního úložiště dat (cloudu). Odtud jsou poté vyčítány webovou aplikací, která s využitím moderních technologií představuje příjemné multiplatformní rozhraní pro autorizované uživatele.

V první kapitole práce jsou popsány vybrané IoT technologie, pomocí kterých lze zajistit přenos dat od jednotlivých senzorických uzlů k centrální jednotce. Jsou zde zahrnuty jak komunikační protokoly, tak bezdrátové technologie. V další kapitole 2 jsou zmíněny environmentální veličiny, které lze měřit, a několik konkrétních senzorů, které byly vybrány pro následnou implementaci. V kapitole 3 je zdokumentována architektura navrženého systému, je zde obsaženo blokové schéma a stručné rozebrání jednotlivých bloků včetně komunikace mezi nimi. Podrobnějším popisem hardwarového uspořádání měřicího uzlu se zabývá kapitola 4, kapitola 5 se pak věnuje firmwarové implementaci. Nakonec je v kapitole 6 popsána uživatelská aplikace, a to jak z pohledu uživatelského rozhraní, tak i z pohledu dalších funkčností, které se dějí na pozadí.

Seznam symbolů a zkratek

IoT.....Internet věcí (Internet of Things)

VOC.....Těkavá organická látka (Volatile Organic Compound)

MQTT.....Message Queuing Telemetry Transport

TCP/IP.....Transmission Control Protocol/Internet Protocol

SSL/TLS.....Secure Sockets Layer/Transport Layer Security

QoS.....Quality of Service

JSON.....JavaScript Object Notation

BSON.....Binary JSON

CBOR.....Concise Binary Object Representation

HTTP.....Hypertext Transfer Protocol

HTTPS.....Hypertext Transfer Protocol Secured

URL.....Uniform Resource Locator

WLAN.....Wireless Local Area Network

IEEE.....Institute of Electrical and Electronics Engineers

MIMO.....Multiple Input, Multiple Output

BLE.....Bluetooth Low Energy

CSS.....Chirp Spread Spectrum

WAN.....Wide Area Network

PM.....Particulate Matter (Pevné částice)

I^2C.....Inter-Integrated Circuit

LTE.....Long Term Evolution

VoLTE.....Voice over LTE

NDIR.....Non Dispersive Infra-Red

UART.....Universal Asynchronous Receiver-Transmitter

REST.....Representational State Transfer

API.....Application Programming Interface

USB.....Universal Serial Bus

DPS.....Deska plošných spojů

GPIO.....General-Purpose Input/Output

SD.....Secure Digital

SPI.....Serial Peripheral Interface

I2S.....Inter-Integrated Circuit Sound

LED.....Light-Emitting Diode

PWM.....Pulse Width Modulation

ADC.....Analog-to-Digital Converter

DAC.....Digital-to-Analog Converter

CPU.....Central Processing Unit

RTC.....Real-Time Clock

SRAM.....Static Random-Access Memory

RGB.....Red Green Blue

ESP-IDF.....Espressif IoT Development Framework

SNTP.....Simple Network Time Protocol

NoSQL.....Non Structured Query Language

MAC.....Medium Access Control

SDK.....Software Development Kit

1 IoT technologie

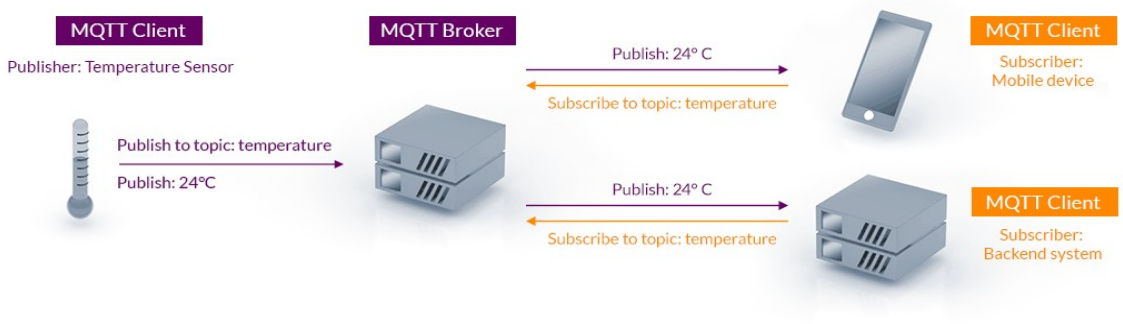
Jak již bylo řečeno v úvodu, internet věcí je v dnešní době velkým tématem. Prakticky vše, co lze připojit k internetu, může být považováno za IoT zařízení (s výjimkou zařízení k tomu přímo určených jako počítač apod.). Patří sem například „chytrá“ lednička, která sama nakoupí, inteligentní termostat, který zohlední aktuální předpověď počasí apod. Nedílnou součástí takových systémů je komunikace mezi jednotlivými uzly, bez které by tento koncept nemohl plnit svoji funkci. Právě komunikaci budou věnovány následující podkapitoly.

1.1 Komunikační protokoly

V této části budou rozebrány základní vlastnosti několika vybraných komunikačních protokolů, které lze využít v konkrétním systému. Reálně jich samozřejmě existuje více, z důvodu rozsahu této práce však bude pozornost věnována pouze některým.

1.1.1 MQTT

MQTT (Message Queuing Telemetry Transport) je jedním z nejvyužívanějších komunikačních protokolů v oblasti IoT. Jeho hlavní výhodou je úspornost zpráv (krátké hlavičky) a jednoduchost. Veškeré zprávy jsou předávány mezi zařízeními v síti pomocí centrálního uzlu (brokeru) a jsou hierarchicky roztříděny do jednotlivých témat (topic) - např. ložnice/světlo, budova-1/podlaží-2/byt-1/kuchyně/teplota apod. Zařízení mohou v daném tématu buď publikovat (publish), tedy odesílat data brokeru, nebo se naopak přihlásit k odběru zpráv daného tématu (subscribe), tedy přijímat určitá data. Jednotlivé uzly mohou publikovat či odebírat zprávy více témat, přičemž navíc platí, že témata publikování konkrétního uzlu se nemusí shodovat s jeho tématy odběru. Tato architektura je znázorněna na obr. 1.1.

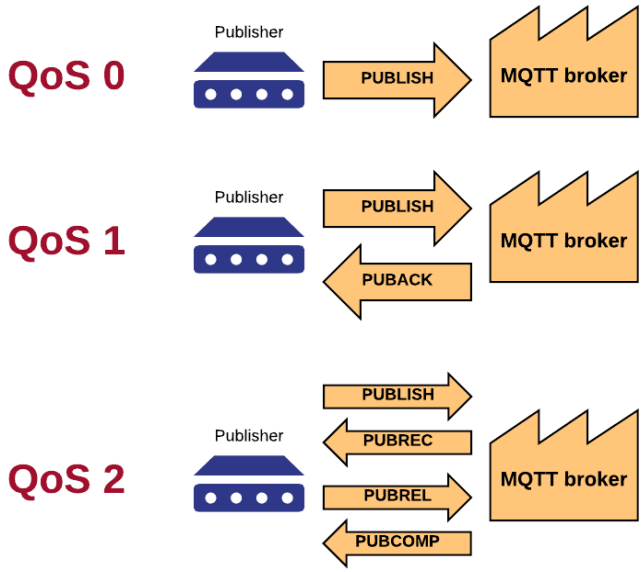


Obr. 1.1. MQTT publish / subscribe architektura (převzato z [2])

Přenos probíhá pomocí protokolu TCP/IP (Transmission Control Protocol/Internet Protocol), případně může MQTT využívat i SSL/TLS (Secure Sockets Layer/Transport Layer Security), což jsou zabezpečené protokoly využívající TCP/IP, které zajišťují zašifrovaný způsob komunikace. Podrobnější informace je možné nalézt např. v [3].

Obsah přenášených zpráv není přesně daný, nejčastěji se používá formát JSON (JavaScript Object Notation), BSON (Binary JSON), nebo CBOR (Concise Binary Object Representation), přičemž BSON a CBOR jsou komprimované formáty. Maximální velikost MQTT zprávy je 268435455 bytů, tedy cca 268 MB.

MQTT protokol řeší i spolehlivost doručení zpráv pomocí 3 úrovní QoS (Quality of Service). Na úrovni 0 (at most once – maximálně jednou) není doručení zprávy nijak garantováno, publisher odešle zprávu brokeru, který ji následně rozešle zařízením odebírajícím dané téma (subscriberům), celý přenos je tedy nespolehlivý. Na úrovni 1 (at least once – alespoň jednou) publisher pošle zprávu brokeru, který ji dále předá subscriberům. Když mu odběratelé potvrdí přijetí zprávou PUBACK, broker taktéž potvrdí publisherovi příjem a ten může odeslanou zprávu smazat. Broker před odesláním PUBACK publisherovi může i nemusí čekat na potvrzení všech příjemců, záleží na konkrétní aplikaci. V případě úrovně spolehlivosti QoS 2 (exactly once – právě jednou) pošle publisher zprávu brokeru, který potvrdí příjem zprávou PUBREC. Publisher mu odpoví zprávou PUBREL, broker následně zprávu smaže a ukončí přenos zprávou PUBCOMP zaslanou publisherovi. Komunikace pro jednotlivé úrovně QoS je naznačena na obr. 1.2.



Obr. 1.2. Úrovně QoS protokolu MQTT (převzato z [1])

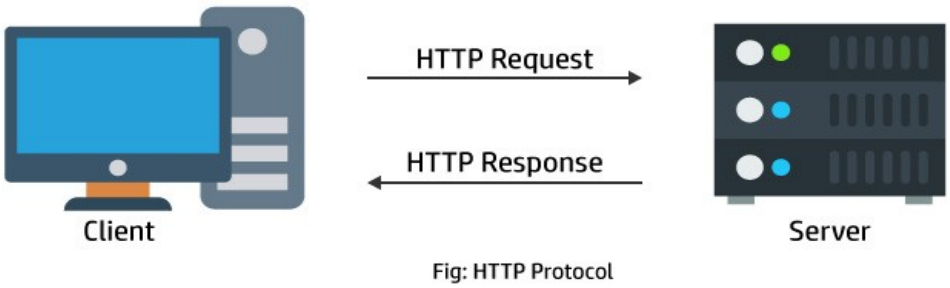
Další funkcí tohoto protokolu je vyhodnocení, zda jsou připojená zařízení aktivní. Pokud konkrétní uzel během určité doby neodešle zprávu, je považován za odpojený. V tomto případě rozešle broker ostatním zařízením zprávu tzv. last will (poslední vůli). Ta může být definována při připojení nového zařízení k brokeru.

Jak vyplynulo z předchozího textu, MQTT protokol je obousměrný, což znamená, že umožňuje jak odeslání zprávy ze zařízení do centrální jednotky, tak odeslání z centrální jednotky do zařízení. To může být užitečné např. při změně systémových parametrů uzlu obdržením zprávy od brokeru. Co se týče brokerů, lze v praxi nalézt mnoho implementací, jednou z nejznámějších je např. Mosquitto. Existuje také řada MQTT knihoven pro různé programovací jazyky a v neposlední řadě i mnoho zařízení, která tento protokol podporují.

1.1.2 HTTP

Dalším protokolem, který je možné při komunikaci využít, je HTTP (Hypertext Transfer Protocol). Ten je základem webu (World Wide Web), ale obecně slouží k přenosu informací mezi zařízeními připojenými k internetu. Typicky je používán webovými prohlížeči, kdy se pomocí hypertextového odkazu načte webová stránka.

Komunikace probíhá tak, že je nejprve navázáno spojení s HTTP serverem, následně klient odešle požadavek (request). Server poté požadavek zpracuje a pošle klientovi odpověď (response). Nakonec je spojení uzavřeno. Tato architektura je znázorněna na obr. 1.3. Spojení vždy navazuje klient, tzn. že komunikace je „jednosměrná“.



Obr. 1.3. Komunikace přes HTTP protokol (převzato z [4])

HTTP protokol přidává do přenášených zpráv mimo samotných dat i další informace. HTTP request obsahuje verzi HTTP, URL (Uniform Resource Locator), HTTP metodu, hlavičku (request header) a případně samotná data (body). HTTP metoda představuje typ akce, která se očekává od serveru, nejčastějšími jsou např. GET (v odpovědi jsou následně očekávány určité informace, např. data z databáze nebo vůbec samotná stránka), či POST (klient typicky odesílá informace serveru, např. při vyplnění webového formuláře). Kompletní přehled těchto metod je možné dohledat v [5].

HTTP response typicky obsahuje HTTP status kód, hlavičku (response header) a případně data (body). HTTP status kód je třímístné číslo, které jednoznačně udává výsledek operace prováděné na serveru (např. 200 – OK, 201 – Založeno, 404 – Nenalezeno, 500 – Interní chyba serveru atd.). Kompletní přehled těchto kódů lze nalézt v [6].

Podobně jako u MQTT je samozřejmě možné využít i zabezpečenou variantu přenosu v podobě HTTPS (HTTP Secured). Opět je použito protokolu SSL/TLS, který se stará o zašifrovanou komunikaci. Dříve byl HTTPS vyžadován hlavně tam, kde hrozila ztráta citlivých údajů (např. internetové bankovnictví, internetové obchody nebo obecně

přihlašování), dnes se již však standardně používá téměř na všech webech a jeho nepřítomnost je známkou nedůvěryhodnosti.

HTTP protokol tedy na jednu stranu přidává k samotným datům řadu údajů, díky čemuž jsou přenášené zprávy delší než u MQTT, na druhou stranu jsou ale snadno čitelné pro člověka. HTTP sám o sobě díky architektuře klient – server neumožňuje kontrolu, zda jsou jednotlivé měřicí uzly v systému aktivní, což ale nemusí být nutně problém, záleží na konkrétní aplikaci.

1.2 Bezdrátové technologie

Další důležitou částí IoT systémů jsou bezdrátové technologie, které zajišťují propojení zařízení v síti. Jsou tedy na nižší vrstvě než protokoly popsané v kapitole 1.1, které se už starají o konkrétní způsob komunikace mezi jednotlivými uzly. V následujících podkapitolách bude stručně popsáno několik vybraných bezdrátových technologií.

1.2.1 Wi-Fi

Wi-Fi v dnešní době asi není nutno dlouze představovat. Jedná se o označení standardů IEEE 802.11, které popisují bezdrátový přístup k síti (WLAN), základní vlastností je velká přenosová kapacita. Pro tento bezdrátový přenos jsou vymezeny frekvenční pásma 2,4 GHz a 5 GHz (kompletní přehled vymezení radiových kmitočtů v ČR je uveden v [8], [9]). Každé z těchto frekvenčních pásem je dále rozděleno na více subpásem, aby se sousední bezdrátové sítě navzájem nerušily.

Skupina IEEE 802.11 obsahuje řadu standardů, které se stále rozšiřují již od roku 1990. Jako příklad lze uvést standard 802.11b, který pracuje na frekvenci 2,4 GHz a umožňuje rychlost přenosu až 11 Mb/s. Dalším je 802.11a, který funguje na frekvenci 5 GHz a je schopen přenášet data s rychlostí až 54 Mb/s. Standard 802.11g je nástupcem 802.11b, který je již schopen dosáhnout přenosové rychlosti až 54 Mb/s v pásmu 2,4 GHz. Prvním standardem, který specifikuje MIMO (Multiple input, Multiple Output – několik vstupů, několik výstupů), je 802.11n, někdy také označován jako Wi-Fi 4. Ten může operovat jak na frekvenci 2,4 GHz, tak na 5 GHz s rychlostí až 600 Mb/s. V současné době většina zařízení využívá standardu 802.11ac, někdy označovaného jako Wi-Fi 5, který

pracuje na frekvenci 5 GHz a umožňuje rychlost přenosu až 3,46 Gb/s. Také podporuje MIMO, což znamená, že může být odesíláno více datových proudů zároveň, podrobněji je tato technologie popsána např. v [11]. Nejnovějšími standardy jsou 802.11ad a 802.11ah, také označovány jako Wi-Fi 6. 802.11ad pracuje na frekvenci 60 GHz a přenos je velmi rychlý, dosahuje až 6,7 Gb/s, což je bohužel vyváženo malou maximální vzdáleností. Standard 802.11ah má naopak delší dosah, funguje na blízkých frekvencích pod 1 GHz a je schopný dosáhnout až 347 Mb/s. Pro podrobnější info viz [10].

V současné době se na trhu vyskytuje řada dostupných Wi-Fi modulů, jejichž součástí je často i mikrokontrolér. Pro aplikaci pak tedy již není potřeba žádný přídavný hardware. Mezi nejrozšířenější moduly patří např. ESP8266 nebo ESP32 od výrobce Espressif.

1.2.2 Bluetooth

Bluetooth je standardem bezdrátového přenosu dat na krátké vzdálenosti. Pracuje ve frekvenčním pásmu 2,4 GHz a je velmi rozšířen ve světě mobilních zařízení (např. bezdrátová sluchátka, chytré hodinky apod.). V kontextu internetu věcí stojí za zmínku hlavně verze Bluetooth 4, také označována jako BLE (Bluetooth Low Energy), která je – jak již z názvu vyplývá – navržena pro nízkou spotřebu zařízení. Data jsou přenášena pomocí 40 kanálů o šířce pásma 2 MHz, maximální dosažitelná rychlost přenosu je 3 Mb/s. Tato technologie se hodí pro aplikace, které trvale přenášejí menší objemy dat.

1.2.3 ZigBee

ZigBee vychází ze standardu IEEE 802.15.4 a stejně jako Bluetooth operuje ve frekvenčním pásmu 2,4 GHz. Přenosová rychlost může dosáhnout až 250 kb/s, je tedy o něco pomalejší. To je ale vyváženo možností přenášet signál až na vzdálenost 200 m. Jedná se o poměrně jednoduchý protokol, který díky důrazu na nízkou spotřebu nachází využití právě v zařízeních obsahujících senzory a mikrokontroléry. Pro propojení jednotlivých uzlů lze využít různé síťové topologie, což může být výhodou při rozšiřování sítě. Teoretický maximální možný počet uzlů v síti je (dle oficiální specifikace v [14]) až 65000.

1.2.4 LoRa

Další bezdrátovou technologií využívanou v IoT je LoRa (zkratka pro Long Range). Oproti výše zmíněným umožňuje přenášet data na podstatně delší vzdálenosti (v řádu kilometrů). Samozřejmě je také určena pro zařízení s nízkou spotřebou. Lze dosáhnout přenosových rychlostí od 300 b/s až 50 kb/s, v porovnání s ostatními technologiemi je tedy pomalejší. Její velkou výhodou je však možnost použití i v odlehlých lokalitách, kde nelze využít jinou přenosovou technologii. Pracuje v bezlicenčním frekvenčním pásmu (oblast pod 1 GHz), s čímž se z hlediska vysílání váže možnost využití pouze 1% času. Prakticky to tedy znamená omezení počtu zpráv (během hodiny lze vysílat pouze 36 sekund).

LoRa je označení pro techniku modulace rozprostřeného spektra, která je odvozená od technologie CSS (Chirp Spread Spectrum) – pro více informací viz [16]. Tato modulace se při přenosu dat provádí na fyzické vrstvě. LoRaWAN (LoRa Wide Area Network) je komunikační protokol, který operuje až ve vyšší vrstvě nad touto technologií. Typickou topologií sítě LoRaWAN je hvězda, v jejímž středu se nachází gateway (odchozí brána), která umožňuje převod radiového signálu na IP (Internet Protocol). Tato síť má u nás slušné pokrytí a pokud v dané oblasti není žádný operátor provozující gateway, je možné vybudovat si gateway vlastní a stát se tak prvním operátorem na daném území. Podrobnější popis lze najít v [15], [17].

1.2.5 SigFox

SigFox je technologie závislá na operátorovi, který musí mít v dané oblasti základnovou stanici (na rozdíl od LoRaWAN). Tato síť má topologii hvězdy, jejíž střed tvoří právě základnová stanice. Opět lze komunikovat na velmi dlouhé vzdálenosti (až desítky km) a protokol je navržen pro přenos krátkých zpráv, což umožňuje nízkou spotřebu připojených zařízení. Tato technologie pracuje na bezlicenčním pásmu (oblast pod 1 GHz), musí zde tedy být počítáno s určitým časovým omezením přenosu. V České republice je tato síť zaváděna ve spolupráci s mobilním operátorem T-Mobile a v současné době poskytuje na našem území velmi dobré pokrytí. To se zatím nedá říct o všech státech, pro kompletní přehled globálního pokrytí viz [18]. Zajímavostí SigFoxu je poskytování

cloudových služeb, které jsou přímo integrované do sítě. Podrobnější informace lze dohledat v [19], [20].

1.2.6 NB-IoT

Poslední bezdrátovou technologií, které se tato práce bude věnovat, je NB-IoT (Narrowband). Ve světě se používá celá řada různých frekvenčních pásem, z nichž část používaná u nás je frekvencemi blízká bezlicenčnímu pásmu. Přenos má tedy z hlediska dosahu a průchodnosti skrz překážky podobné vlastnosti jako technologie LoRa či SigFox. Přenosová rychlost může dosáhnout až 200 kb/s, díky použití licencovaného pásma lze ale vysílat neomezeně. Jedná se o technologii, která podporuje velmi nízkou spotřebu připojených zařízení. V rámci různých zemí je možné setkat se s různými frekvenčními pásmy, což může být komplikované z hlediska hardwaru, kdy je třeba brát ohled na to, aby zvolený modul podporoval všechna potřebná pásma a měl na příslušné frekvence naladěnou anténu. NB-IoT je provozován operátory mobilních sítí, pro připojení je tedy nutná SIM karta. Další informace jsou k nalezení např. v [23].

Pokrytí touto technologií se ve světě různí, některé státy používají místo NB-IoT technologii LTE Cat M1, také označovanou jako LTE-M, která umožňuje přenos dat rychlostí až 1 Mb/s a na rozdíl od NB-IoT podporuje i přenos hlasu – VoLTE (Voice over LTE). Podrobnější informace a porovnání obou technologií lze najít v [24], přehled pokrytí ve světě je dostupný v [25].

2 Měření environmentálních veličin

Životní prostředí, ve kterém se pohybujeme, má přímou souvislost s naším zdravím i s naším pohodlím. S postupem času se navíc stávají dostupnější další technologie a senzory, které se dříve ke koncovému uživateli běžně nedostaly (ať už z důvodů finančních či jiných). Tyto technologie se navíc s rozvojem internetu věcí stále posouvají dopředu.

Aktuálně je možné měřit řadu environmentálních veličin, patří mezi ně např. teplota, relativní vlhkost vzduchu, koncentrace CO_2, VOC (těkavé organické látky), PM (pevné částice) či koncentrace formaldehydu, který dokáže způsobit vážné zdravotní komplikace. Z důvodu rozsahu se tato práce bude věnovat pouze několika vybraným veličinám.

2.1 Použité senzory

V následujících podkapitolách bude podrobněji probráno měření několika environmentálních veličin pomocí konkrétních senzorů, které byly zvoleny pro implementaci.

2.1.1 HDC2080

Tento integrovaný senzor od výrobce Texas Instruments slouží k měření teploty a zároveň i relativní vlhkosti. Je schopen změřit teplotu v rozsahu -40 až +85 °C a relativní vlhkost od 0 do 100 %. Co se týče přesnosti, výrobce udává ve specifikaci pro teplotu typickou odchylku ±0,2 °C a maximální odchylku až ±0,4 °C. Pro relativní vlhkost platí dle specifikace odchylka typicky ±2 % a maximálně ±3 %. Tento čip operuje při napájecím napětí 1,62 V až 3,6 V a byl navržen pro aplikace s nízkou spotřebou. Komunikuje přes rozhraní I^2C (Inter-Integrated Circuit). Umožňuje aktivovat automatický mód měření, kdy se vždy v cyklu po požadovaném intervalu senzor „probudí“, provede měření a poté se opět přepne do režimu nízké spotřeby (sleep mode). Tento interval může být naprogramován v rozsahu 5 vzorků za sekundu až 1 vzorek za 2 minuty. V režimu sleep mode senzor odebírá proud typicky 50 nA, maximálně 100 nA, což podstatně snižuje průměrnou spotřebu a vlastní ohřev čipu. Při měření s intervalem 1 vzorek za sekundu

a rozlišení 11 bitů udává výrobce průměrnou spotřebu 550 nA (za předpokladu měření obou dvou veličin zároveň). Rozlišení převodu obou veličin lze nastavit v příslušném registru, čímž je také ovlivněna doba konverze a tedy i samotná spotřeba. Je možné aktivovat i přerušení (interrupty), kdy senzor může informovat připojený mikrokontrolér, že měření a převod dat byly dokončeny, nebo některá z měřených veličin přesáhla určité stanovené meze. Čip obsahuje i integrovaný ohřívač, který je možné zapnout pro odstranění kondenzátu, jenž se může vyskytnout při použití zařízení v prostorách s vysokou relativní vlhkostí. Více informací je možné najít v [22].

Relativní vlhkost je v tomto případě měřena kapacitním snímáním. Senzor obsahuje dvě elektrody, mezi které je vložena tenká vrstva polymerního dielektrika. Polymer se zvyšující se vlhkostí absorbuje molekuly vody ze vzduchu v okolí a mění tak svou relativní permitivitu, což se promítne i ve změně kapacity a může být dále zpracováno. Tento jev je podrobněji popsán v [22]. Vše je již vyřešeno v rámci tohoto integrovaného čipu, z jehož registrů lze vyčíst naměřené hodnoty a následně je přepočítat dle příslušných vztahů v datasheetu. Princip měření teploty není ve specifikaci výrobcem uveden.

2.1.2 SGP40

Tento senzor od firmy Sensirion je určen pro měření kvality ovzduší vnitřních prostor, konkrétně VOC (těkavých organických látek), které mohou u člověka způsobit řadu zdravotních komplikací jako podráždění očí, bolesti hlavy, zhoršené soustředění, nevolnosti, závratě, dušnost či kožní alergické reakce. V horším případě může kontakt s těmito látkami vést u člověka i ke krvácení z nosu, poškození některých orgánů, selhání centrálního nervového systému nebo k rakovině. Zdravotní důsledky se samozřejmě mohou velmi lišit v závislosti na konkrétní chemikálii – některé látky jsou vysoce toxické, u některých naopak nejsou známy žádné dopady na zdraví. Dále také záleží na množství dané látky, kterému byl člověk vystaven, a jak dlouho.

Látkami znečišťující ovzduší mohou být škodlivé plyny – např. aceton (obsažen v barvách, lepidlech apod.) nebo toluen (vylučován z nábytku, matrací či stavebního materiálu). Dalším plynem je etanol, jehož zdrojem je alkohol, čisticí prostředky, parfémy apod. Lze měřit také různé zápachy, např. amoniak a aminy, které se uvolňují z moči domácích mazlíčků, nebo sirovodík a těkavé sloučeniny síry, které jsou vylučovány

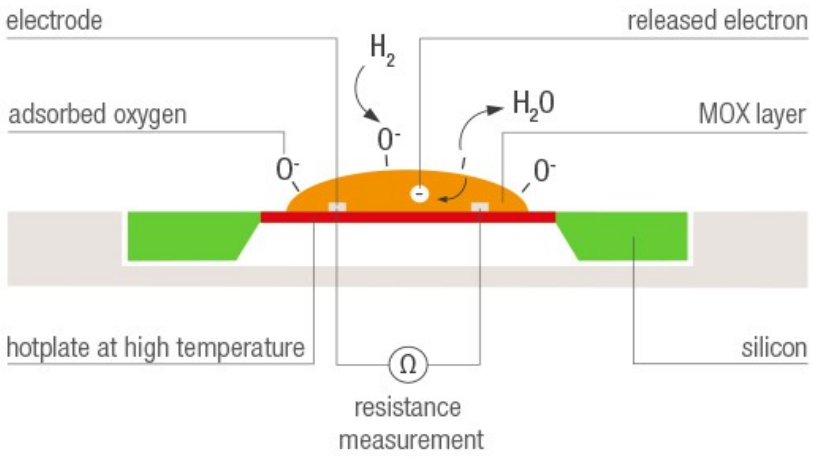
z hnijícího jídla či při metabolické činnosti (plynatosti). Dalšími zdroji může být např. i lidský dech nebo cigaretový kouř. Lze tedy snímat znečištění způsobeno řadou různých podnětů.

SGP40 měří tzv. VOC index, což znamená, že vyhodnocuje pouze relativní změny koncentrace VOC látek vzhledem k historii místnosti, neměří tedy přímo absolutní hodnotu. VOC index nabývá hodnot 0 až 500 a je vztažen k průměrné koncentraci VOC v místnosti za posledních 24 hodin. Senzor není schopen detekovat plyny, které jsou přítomny moc dlouho, po 3 hodinách začne VOC index klesat zpět na průměrnou hodnotu (100), protože se adaptuje na okolní prostředí. Hodnota 100 udává typické složení plynu v místnosti za posledních 24 hodin, rozmezí hodnot 100 až 500 značí zhoršení kvality ovzduší (zvýšení koncentrace VOC) a rozmezí 0 až 100 znamená zlepšení. Podrobnější informace si lze přečíst v [27], [28].

Tento senzor může operovat při napájecím napětí 1,7 V až 3,6 V, teplotě -10 až +50 °C a relativní vlhkosti 0 až 90 %. Je navržen pro snadné použití např. v osvěžovačích vzduchu nebo ve ventilačních systémech s řízením dle potřeby. Komunikace je realizována přes rozhraní I^2C a při implementaci je možné využít knihovnu vytvořenou výrobcem, která obsahuje jednak funkce pro komunikaci s čipem a jednak algoritmus pro převod naměřeného signálu na VOC index. Tento algoritmus je velmi robustní a při pravidelném měření se automaticky adaptuje na okolní prostředí senzoru, proto výrobce doporučuje provádět měření pravidelně po 1 s. V čipu je také obsažen velmi malý ohřívač, který umožňuje automatickou kompenzaci relativní vlhkosti. Tato kompenzace lze zařídit i tak, že se senzoru pošlou při spuštění měření údaje o aktuální teplotě a vlhkosti přes I^2C z mikrokontroléru, v tom případě zůstává ohřívač vypnutý. V režimu nízké spotřeby (idle) SGP40 odebírá typicky 34 µA, maximálně 105 µA. Při pravidelném měření s intervalem 1 s při napájecím napětí 3,3 V je průměrná spotřeba uváděná výrobcem typicky 2,6 mA, maximálně 3 mA. Při tomto intervalu měření je doba odezvy udaná od výrobce pod 10 s. Pro více informací viz [28].

Měření je postaveno na bázi oxidů kovu (metal-oxid), přesněji řečeno na technologii MOXSens® Technology vyvinutou přímo výrobcem. Princip snímání je založen na zahřívání tenké vrstvy nanočástic oxidu kovu, přičemž dochází k adsorbci kyslíku, který následně reaguje s okolním (měřeným) plynem a tím uvolňuje své elektrony. Dochází tedy

ke změně elektrické vodivosti vrstvy metal-oxidu, která je následně měřena. Tento princip je naznačen na obr. 2.1. SGP40 obsahuje hned čtyři snímací prvky.



Obr. 2.1. Princip měření VOC pomocí senzoru SGP40 (převzato z [29])

Běžné metal-oxidové senzory využívající tento způsob měření nejsou dlouhodobě příliš stabilní, což je způsobeno nevratnou kontaminací siloxany. Jejich působením tyto senzory významně ztrácí citlivost a zvyšuje se jejich doba odezvy. Siloxany jsou bohužel všudypřítomné, nelze tedy jejich kontaktu se senzory zabránit. Technologie MOXSens® Technology od Sensirionu je vůči nim odolná a senzory SGP jsou tak poměrně přesné a stabilní i po dlouhou dobu. Porovnání je znázorněno na obr. 2.2. Pro více informací viz [29].

Obr. 2.2. Porovnání stability senzorů SGP a běžných metal-oxidových senzorů

2.1.3 SCD30

Posledním použitým senzorem je SCD30 opět od firmy Sensirion, který slouží k měření koncentrace CO_2. Příliš vysoká hodnota může způsobit bolesti hlavy, zhoršení soustředění a mírné nevolnosti. Ve venkovním prostředí je obsaženo cca do 500 ppm (parts per million – počet částic na jeden milion) CO_2 částic, v kvalitním ovzduší do 1000 ppm. Senzor SCD30 může komunikovat přes rozhraní I^2C, UART a nebo PWM. Je schopný změřit koncentrace v rozsahu 0 až 40000 ppm, pokud se pro komunikaci využívá rozhraní I^2C či UART, nebo v rozsahu 0 až 5000 ppm, pokud se používá PWM výstup. Maximální odchylka ±(30 ppm + 3% z naměřené hodnoty) udávaná výrobcem je však garantována pouze v rozsahu 400 až 10000 ppm. Senzor pracuje při napájecím napětí 3,3 V až 5,5 V, teplotě 0 až 50 °C a při relativní vlhkosti 0 až 95% (bez kondenzace). Čip obsahuje také integrovaný senzor teploty a relativní vlhkosti pro korekce naměřených hodnot, ten ale bohužel není příliš přesný. Během měření odebírá SCD30 dle specifikace 75 mA, při měření s intervalem 1 vzorek za 2 sekundy činí průměrná spotřeba typicky 19 mA. Není tedy primárně určen pro zařízení s nízkou spotřebou, to je ale dáno charakterem měření (viz dále). Doba odezvy udávaná výrobcem je 20 s, životnost až 15 let. Měření je plně kalibrováno a linearizováno. V současnosti se jedná o jeden z nejlepších senzorů na trhu, co se týče poměru cena-výkon. Více informací o senzoru SCD30 lze najít v [30], informace o hodnotách CO_2 pak v [31].

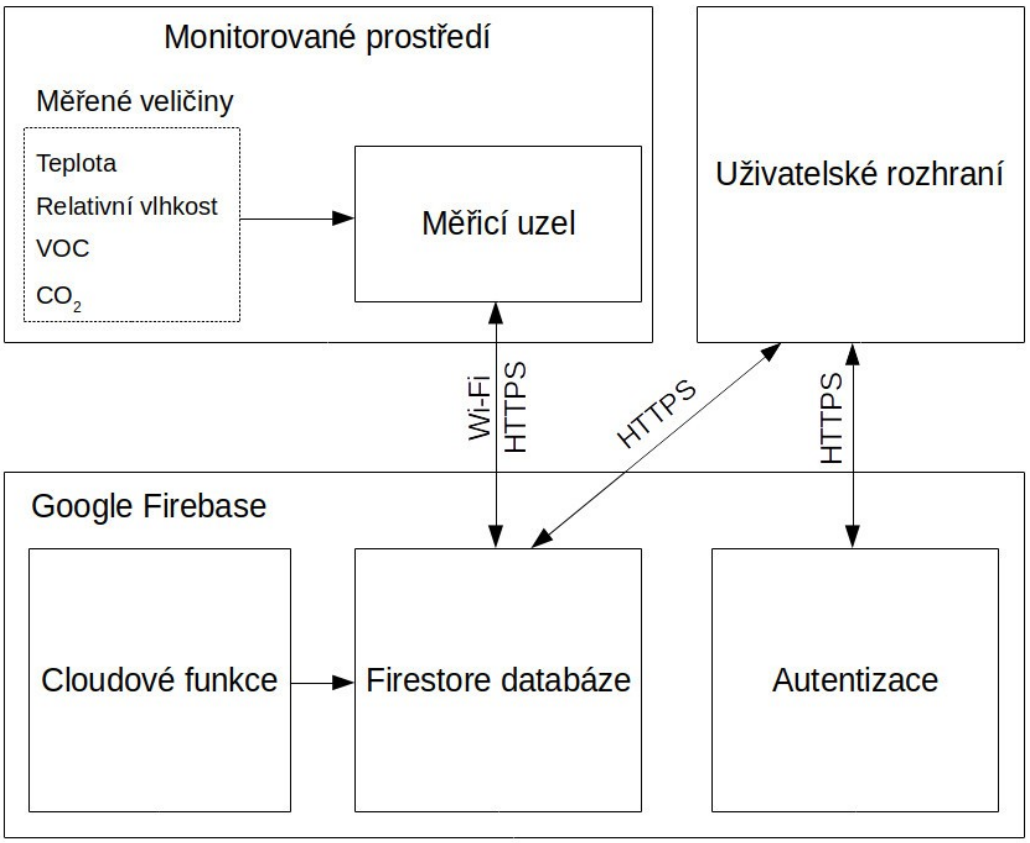
Měření je založeno na principu NDIR (Non Dispersive Infra-Red), který využívá toho, že částice různých plynů absorbují záření o různých vlnových délkách. Maximální absorpce CO_2 a minimální absorpce ostatních plynů je dosaženo při infračerveném záření o vlnové délce 4,3 μm. Zdroj záření se nachází na jedné straně komory naplněné plynem, která je prosvěcována. Neabsorbovaná část záření dále dopadá na druhé straně komory na dva snímače s optickými filtry, kde je měřena její intenzita. Jeden filtr propouští záření v pásmu 4,3 μm, druhý (referenční) záření o takové vlnové délce, která je absorbována plyny pouze minimálně, typicky se jedná 4 μm. Při výpočtu koncentrace CO_2 se pak počítá s poměrem těchto intenzit, je tedy vykompenzováno kolísání zdroje záření, což zajišťuje i dlouhodobou stabilitu.

3 Architektura systému

Dále se již práce bude věnovat samotnému návrhu a implementaci aplikace pro monitorování environmentálních veličin. Následující podkapitoly popisují architekturu systému jako celku, jeho jednotlivé bloky a komunikaci mezi nimi.

3.1 Blokové uspořádání

Systém pro monitorování environmentálních veličin se skládá z několika bloků, které mezi sebou navzájem komunikují a zajišťují tak požadovanou funkčnost. Blokové schéma je znázorněno na obr. 3.1.



Obr. 3.1. Blokové schéma systému

Měřicí uzel představuje zařízení, které obsahuje potřebné senzory, řídicí mikrokontrolér a další komponenty nezbytné pro sběr lokálních dat. Nachází se tedy přímo v kontaktu s měřeným prostředím a s požadovanou frekvencí snímá teplotu, relativní vlhkost a kvalitu ovzduší v podobě koncentrace CO_2 a VOC. Toto zařízení může být jednak napájeno z baterie, díky charakteru měření některých senzorů, které nejsou příliš úsporné, ho však lze napájet i ze sítě za použití adaptéru s výstupním stejnosměrným napětím 5 V. Podrobnější popis je dále uveden v kapitole 4.

Z měřicího uzlu jsou data odesílána do centrálního úložiště, které je součástí cloudové platformy Google Firebase. Zde jsou mimo naměřená data od jednotlivých uzlů uloženy i konfigurace zadané uživatelem pro jednotlivá zařízení (např. interval měření nebo interval odesílání dat do databáze). V rámci tohoto bloku je vyřešena i autentizace při vstupu do uživatelské aplikace a další funkce, které se provádí na pozadí. Této části se podrobněji věnuje kapitola 6.1.

Poslední částí je webová aplikace v podobě uživatelského rozhraní (frontend), které umožňuje přihlášeným uživatelům vizualizaci naměřených dat z různých uzlů. Uživatelé také mohou měnit nastavení některých parametrů, které již byly zmíněny výše (např. interval měření). Frontend je podrobněji popsán v kapitole 6.2.

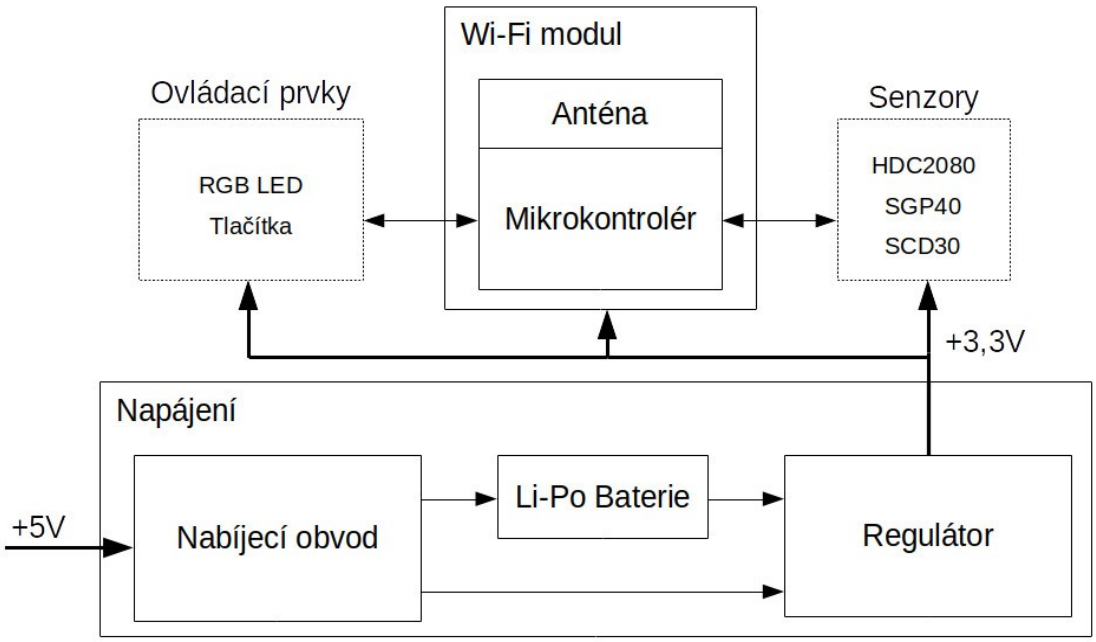
3.2 Způsob komunikace

Cílem této kapitoly je popsat, jakým způsobem mezi sebou komunikují jednotlivé bloky systému. Měřicí uzel posílá naměřená data přes Wi-Fi do centrálního úložiště. Jako komunikační protokol byl zvolen HTTPS (tedy zabezpečený HTTP), který platforma Google Firebase (konkrétně její Firestore databáze) podporuje. Nejjednodušší je použít v aplikaci přímo jednu z nativních knihoven, které existují pro různé programovací jazyky, tato možnost se však hodí spíše pro implementaci mobilních či webových aplikací. Pro přístup k Firestore databázi v případech, kdy není možné takovou knihovnu využít (jako např. IoT aplikace), jsou Googlem specifikovány jednotlivé endpointy v podobě REST API, jejichž zavoláním lze v databázi provádět různé operace – např. vytvořit nový záznam, upravit již existující data, smazat záznam nebo vyčíst požadované záznamy. Data se přenáší pomocí formátu JSON, přičemž je navíc u každé hodnoty jasně dáno, jakého je datového typu.

Komunikace mezi uživatelskou aplikací a Google Firebase vypadá podobně, na rozdíl od měřicího uzlu je zde ale využito zmíněných knihoven. To platí jak pro komunikaci s databází, tak pro funkci autentizace uživatele. Uživatelská aplikace je napsána pomocí platformy Angular, o veškerou komunikaci s Google Firebase se stará knihovna *angular/fire*.

4 Hardwarové uspořádání měřicího uzlu

Měřicí uzel sestává z napájení, řídicího mikrokontroléru (jehož součástí je i Wi-Fi anténa) a několika senzorů. Jednotlivé bloky budou podrobněji probrány v následujících podkapitolách. Blokové schéma je znázorněno na obr. 4.1. Kompletní schéma je připojeno v příloze *A*, vizualizace návrhu desky plošných spojů v přílohách *B* a *C*. Kompletní projekt vývojového nástroje, ve kterém byl návrh vytvořen, lze najít v přiloženém archivu ve složce *kicad_env_monitor*.



Obr. 4.1. Blokové schéma měřicího uzlu

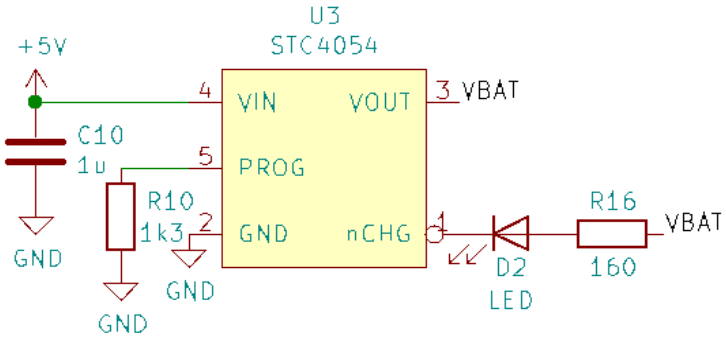
4.1 Napájení

K napájení lze využít jednak baterii, díky vysoké spotřebě některých použitých senzorů (což vychází z charakteru měření a nelze to tedy příliš ovlivnit) je však možné napájet zařízení i ze sítě při využití adaptéru s výstupním stejnosměrným napětím 5 V. Lze tedy použít i klasickou nabíječku k telefonu s konektorem micro USB. Pro implementaci byl zvolen akumulátor typu Li-Po, jeho jmenovité napětí je tedy 3,7 V, kapacita 2000 mAh.

Zařízení obsahuje nabíjecí obvod STC4054 od firmy STMicroelectronics, který umožňuje nabíjet baterii, přičemž uzel je napájen ze sítě a může tedy v průběhu nabíjení dále fungovat. Ať už z baterie nebo z nabíjecího obvodu, napětí je dále přivedeno na vstup spínaného regulátoru TPS63020-Q1 od výrobce Texas Instruments, který ho převede na hodnotu 3,3 V. Toto napětí je pak dále vedeno k jednotlivým součástkám.

4.1.1 Nabíjecí obvod STC4054

Tento obvod pracuje při napájecím napětí 4,25 až 6,5 V, lze tedy k nabíjení využít i USB. Nabíjecí proud může dosáhnout úrovně až 800 mA, maximální hodnotu lze nastavit externím odporem (v návrhu součástka R10). STC4054 operuje při teplotě v rozsahu -40 až +85 °C. Pokud není připojen adaptér, teče z baterie zpět do tohoto obvodu proud 2 µA. Při připojeném adaptéru, pokud se zrovna nenabíjí (standby mode), udává výrobce spotřebu typicky 150 µA, maximálně 300 µA.

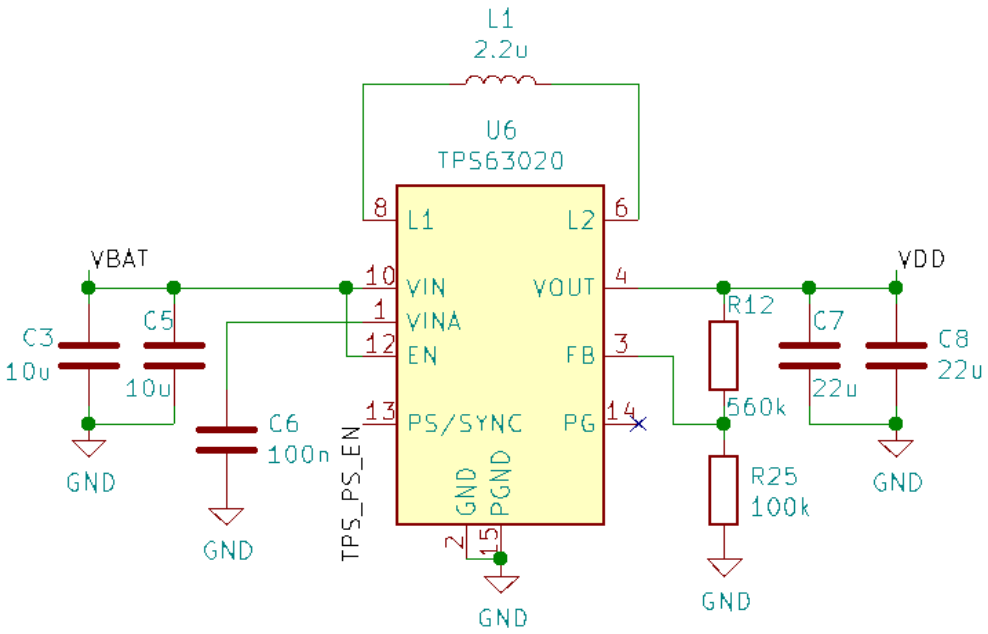


Obr. 4.2. Zapojení součástky STC4054 v návrhu

Nabíjení probíhá tak, že pokud je napětí baterie menší než 2,9 V, nabíjí se proudem o velikosti 1/10 nastavené hodnoty, dokud se baterie nedostane na bezpečnou úroveň. Po překročení hodnoty 2,9 V se přejde do režimu nabíjení konstantním proudem. Když se napětí na baterii blíží konečné hodnotě (4,2 V), začne režim nabíjení konstantním napětím a proud pomalu klesá až na 1/10 zvolené hodnoty, kdy je nabíjecí cyklus ukončen. Obvod umožňuje automatické dobíjení, které začne, když napětí na baterii klesne pod 4,05 V. Pokud se baterie zrovna nabíjí, rozsvítí se indikační LED (v návrhu součástka D2). Pro více informací o tomto čipu viz [34].

4.1.2 Spínaný regulátor TPS63020-Q1

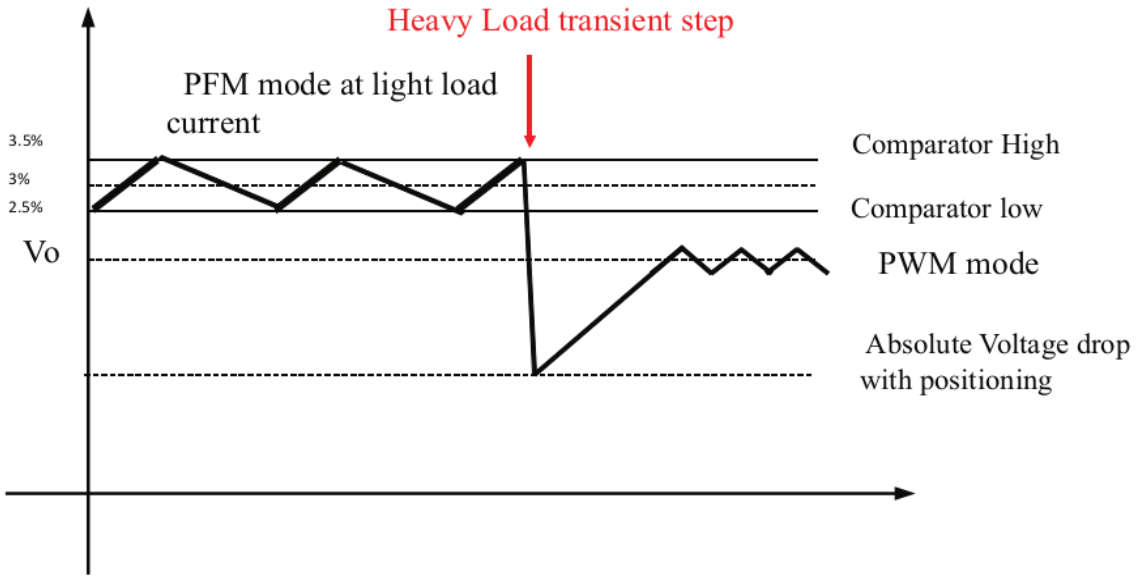
Jak již bylo zmíněno výše, dále je nutné převést napětí na úroveň 3,3 V, k čemuž slouží spínaný regulátor TPS63020-Q1. Jedná se o regulátor typu buck-boost, tzn. umožňuje snížení i zvýšení vstupního napětí na požadovanou výstupní hodnotu, přičemž mezi těmito režimy lze přecházet zcela automaticky. Na vstup je možné přivést napětí o velikosti 1,8 až 5,5 V, výstupní napětí lze nastavit v rozmezí 1,2 až 5,5 V pomocí odporového děliče (v návrhu součástky R12 a R25). Maximální výstupní proud (kontinuální) může být 2 A při režimu zvyšování napětí a 4 A při režimu snižování napětí. Tento čip pracuje při teplotě -40 až +125 °C a dosahuje účinnosti až 96%. Frekvence spínání je typicky 2,4 MHz a klidový proud menší než 50 µA. V rámci čipu jsou již implementovány ochrany proti přehřátí, přepětí i zkratu.



Obr. 4.3. Zapojení součástky TPS63020-Q1 v návrhu

Pro lepší účinnost při nízkém výstupním výkonu lze aktivovat úsporný režim (power save mode). To je možné pomocí pinu PS/SYNC, který v tomto případě musí být nastaven na nízkou úroveň. V návrhu je sem přiveden signál TPS_PS_EN z mikrokontroléru (pin IO26) kvůli měření baterie, které je podrobněji vysvětleno v kapitole 4.1.3. Pokud je pin PS/SYNC nastaven správně, do úsporného režimu se přejde, jestliže průměrný proud cívkou (v návrhu součástka L1) klesne přibližně pod 100 mA. Pak regulátor pracuje

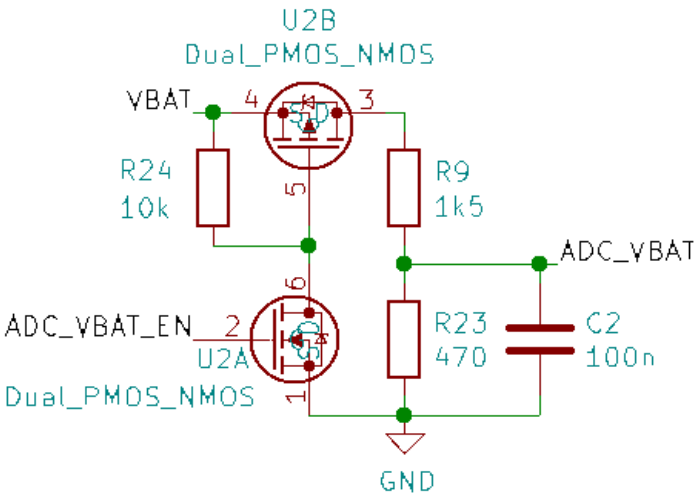
na nižší spínací frekvenci a s nižším klidovým proudem. Po přechodu do úsporného režimu přestane převodník pracovat a výstupní napětí začne padat až k hodnotě cca o 2,5% vyšší, než je navolena. Poté regulátor začne výstupní napětí opět zvyšovat tím, že začne pracovat s nastaveným proudem cívkou o něco vyšším, než je potřeba pro aktuální zátěž. Napětí stoupá až na hodnotu přibližně o 3,5% vyšší, než je požadované výstupní napětí. Střední hodnota výstupního napětí v úsporném režimu je tedy cca o 3% vyšší, než jaká byla nastavena. Při zvýšení průměrného proudu cívkou nad 100 mA regulátor opět automaticky přejde do PWM režimu. Tato funkčnost je názorně ukázána na obr. 4.4. Podrobnější informace o tomto čipu lze najít v [35].



Obr. 4.4. Funkční režimy regulátoru TPS63020-Q1 (převzato z [35])

4.1.3 Měření napětí baterie

Baterie je měřena pomocí A/D převodníku řídicího mikrokontroléru. Tento převodník není příliš přesný, vyhodnocuje se tedy spíše případný pokles napětí než přímo hodnota. Zapojení použité v návrhu je znázorněno na obr. 4.5.



Obr. 4.5. Zapojení měření baterie

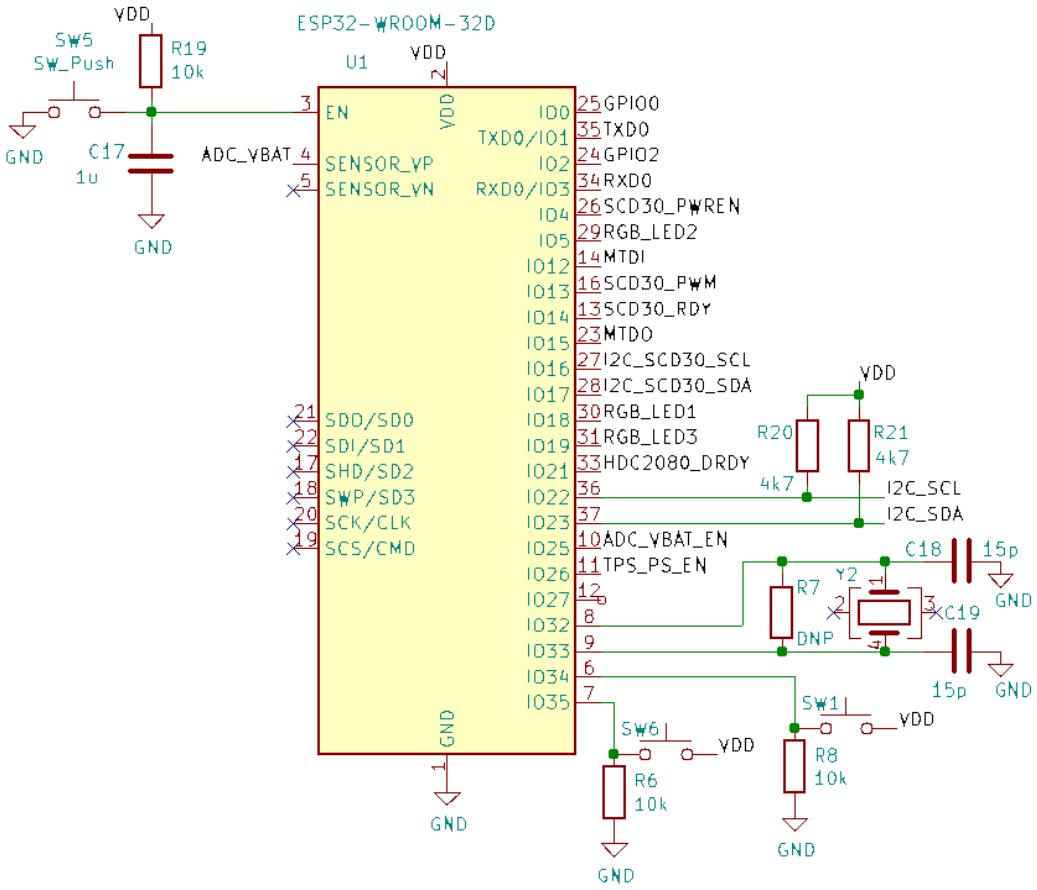
Signál VBAT představuje napětí baterie, ADC_VBAT vede na vstup A/D převodníku (pin SENSOR_VP) a signál ADC_VBAT_EN je připojen na pin mikrokontroléru IO25. Napětí baterie je měřeno přes odporový dělič R9, R23. Tranzistor U2B je za běžného provozu uzavřen, aby nemohl přes dělič protékat proud a baterie se tak nevybíjela. Před měřením se vždy na hradlo tranzistoru U2A přivede z mikrokontroléru signálem ADC_VBAT_EN vysoká úroveň, která tranzistor otevře, což následně vede i k otevření tranzistoru U2B. Poté začne přes odporový dělič protékat proud a po ustálení hodnoty je již možné změřit napětí.

Měření lze provádět pouze za předpokladu, že regulátor TPS63020-Q1 není zrovna v úsporném režimu, jinak by docházelo ke značnému zkreslení získaných hodnot. Reference A/D převodníku totiž závisí na napájecím napětí mikrokontroléru, které je při úsporném režimu regulátoru lehce vyšší než požadovaná hodnota a více kolísá kolem své střední hodnoty (viz obr. 4.4). Proto je vždy navíc nutné tento režim před měřením zakázat nastavením signálu TPS_PS_EN na vysokou úroveň (viz obr. 4.3).

4.2 Řídicí mikrokontrolér

Pro implementaci byl zvolen mikrokontrolér ESP32-WROOM-32D od výrobce Espressif. Disponuje řadou periferií a rozhraní – SD karta, kapacitní dotykový senzor, Hallův senzor, Ethernet, SPI, UART, I^2S, I^2C, LED PWM, PWM pro řízení motorů, GPIO,

ADC a DAC. Součástí je integrovaný oscilátor, který běží na frekvenci 40 MHz, je však možné připojit i externí krystal (v návrhu součástka Y2). Tento mikrokontrolér pracuje při napájecím napětí 3 až 3,6 V, doporučená teplota se pohybuje v rozmezí -40 až +85 °C. Pro komunikaci lze využít Bluetooth, BLE nebo Wi-Fi, přičemž Wi-Fi anténa na plošném spoji je již přímo součástí tohoto modulu. Wi-Fi operuje ve frekvenčním pásmu 2,4 GHz a podporuje standardy IEEE 802.11b/g/n. Z hlediska spotřeby je na tom při vysílání nejhůře 802.11b, kdy je odebíráno typicky 240 mA, nejlépe naopak 802.11n, kdy se odebírá typicky 180 mA. Pro příjem je výrobcem udávána u všech tří standardů stejná spotřeba – typicky 95 až 100 mA. Přenosová rychlost může dosáhnout až 150 Mb/s, maximální výkon antény je 20 dBm.



Obr. 4.6. Zapojení mikrokontroléru ESP32-WROOM-32D v návrhu

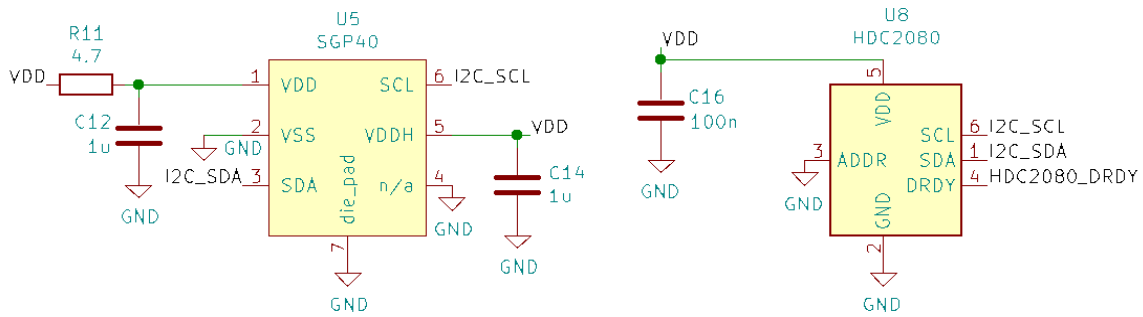
Co se týče spotřeby, umožňuje tato řada mikrokontrolérů různé úsporné režimy. Nevhodnější z nich je pro tuto aplikaci režim deep-sleep, kdy jsou napájeny jen RTC periferie, RTC paměť a ULP (Ultra-Low-Power) koprocesor a je tedy možné probudit hlavní procesor (CPU) časovačem nebo externím přerušením (např. stisk tlačítka). Data,

která je nutno uchovat i při deep-sleep režimu, lze uložit do části RTC paměti – RTC Fast Memory (8 KB SRAM). V provozu (tzn. pokud není aktivován úsporný režim), uvádí výrobce průměrnou spotřebu 80 mA. Pokud je aktivován deep-sleep, mikrokontrolér odebírá proud maximálně 150 µA. Více informací lze nalézt v [36], [37].

4.3 Senzory

Pro implementaci bylo vybráno několik senzorů – HDC2080 pro snímání teploty a relativní vlhkosti, SGP40 pro sledování změn obsahu VOC v ovzduší a SCD30 pro měření koncentrace CO_2. Použité senzory již byly podrobně popsány v rámci kapitoly 2.1, tato část se bude věnovat spíše záležitostem souvisejícím se zapojením daných součástek.

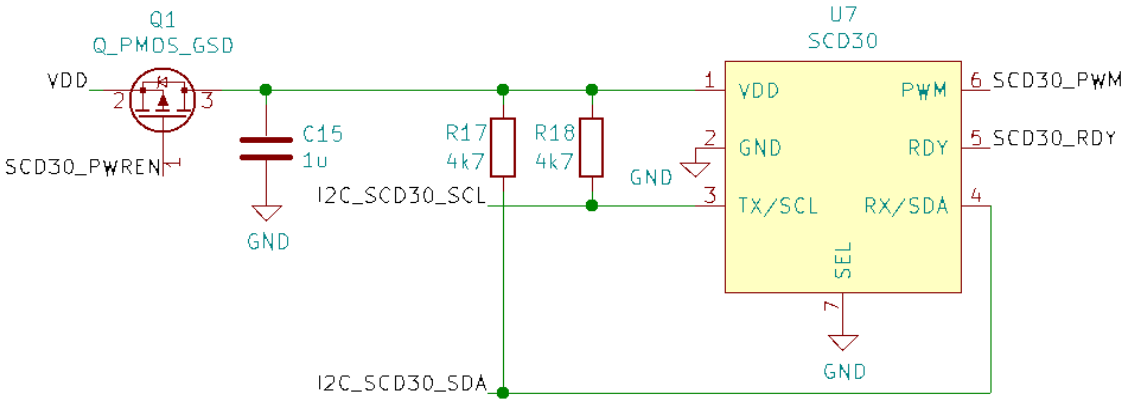
Komunikace se všemi senzory je realizována pomocí I^2C sběrnice, přičemž jsou využita dvě samostatná rozhraní. Jedno slouží pro komunikaci se senzory HDC2080 a SGP40 a v rámci mikrokontroléru jsou pro tuto funkci použity piny IO22 (signál I2C_SCL) a IO23 (signál I2C_SDA) – viz obr. 4.6. Druhé rozhraní slouží pouze pro komunikaci se senzorem SCD30, u mikrokontroléru jsou pro tento účel vyhrazeny piny IO16 (signál I2C_SCD30_SCL) a IO17 (signál I2C_SCD30_SDA). U obou rozhraní jsou implementovány externí pull-up rezistory (v návrhu součástky R17, R18, R20, R21).



Obr. 4.7. Zapojení součástek SGP40 a HDC2080 v návrhu

Komunikační rozhraní jsou takto oddělena z důvodu úspory energie. Senzor SCD30 není z hlediska spotřeby příliš úsporný, což vychází z charakteru měření a nedá se s tím tedy moc dělat. V návrhu byla tato skutečnost zohledněna a do zapojení byl přidán tranzistor Q1, který umožňuje odpojení tohoto senzoru od napájení signálem SCD30_PWREN vyvedeným z mikrokontroléru (pin IO4). Po odpojení napájení nejsou

napájeny ani pull-up rezistory, což zamezuje tomu, aby tudy mohl protékat proud přes ochranné diody uvnitř senzoru (backpowering).

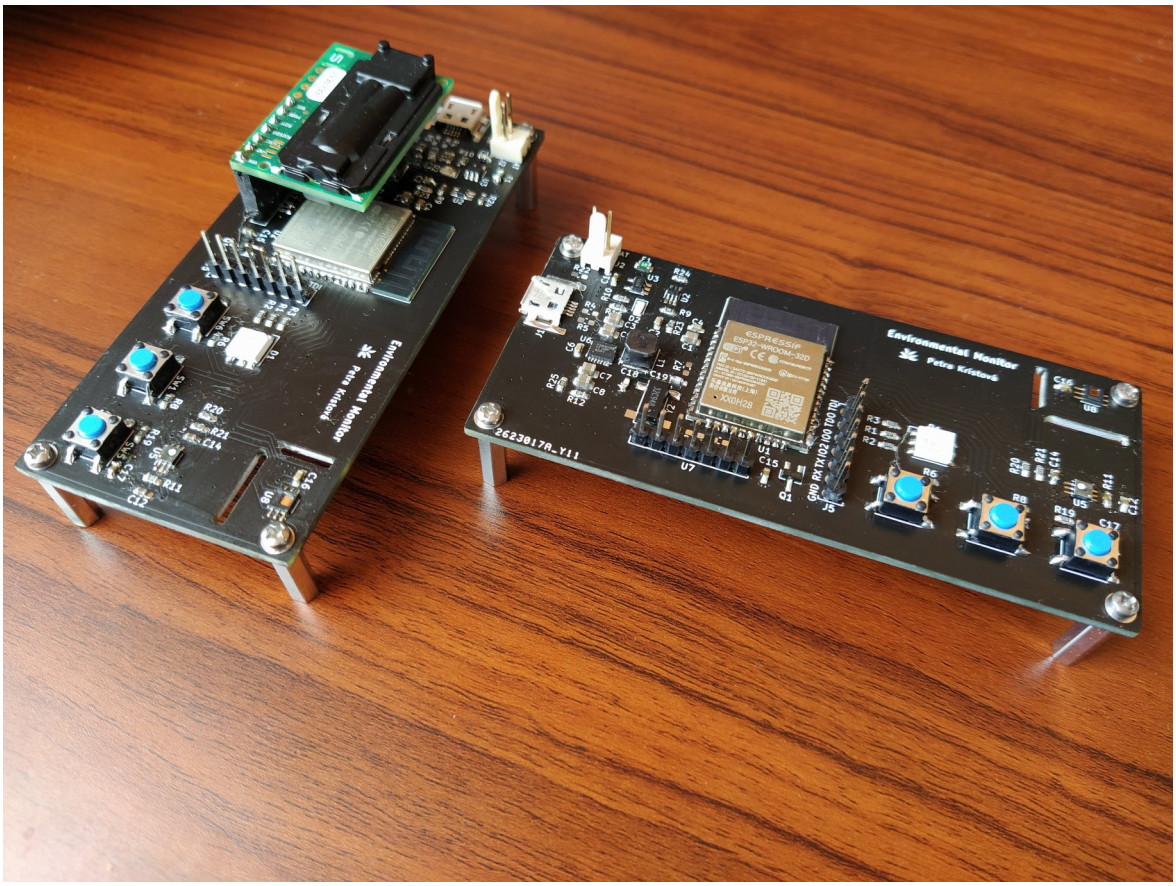


Obr. 4.8. Zapojení součástky SCD30 v návrhu

4.4 Realizace DPS

Deska plošných spojů má dvě vrstvy (top a bottom). Nalevo je umístěna část, která řeší napájení – micro USB konektor, konektor pro připojení baterie společně s pojistkou, nabíjecí obvod a spínaný regulátor. Uprostřed desky se nachází řídicí mikrokontrolér, pod jehož anténou a ještě dalších 5 mm kolem ní není rozlitá měď, aby nedocházelo k ovlivnění antény a nebyla tak ohrožena její správná funkčnost. To platí pro obě dvě vrstvy desky. Dole vedle mikrokontroléru je vyveden konektor pro připojení senzoru SCD30, napravo je umístěn konektor pro signály potřebné k nahrávání firmwaru a případnému ladění (debugging). V pravé dolní části desky se nachází ovládací prvky – RGB LED a tři tlačítka. RGB LED slouží k indikaci probíhajících operací nebo jejich výsledků – např. připojování k Wi-Fi, úspěšné/neúspěšné odeslání dat do databáze apod. Tlačítko SW5 slouží k restartu zařízení, stiskem SW1 se provede rekonfigurace zařízení dle aktuálních parametrů v databázi. Tlačítko SW6 je zde zatím pouze připraveno pro případné budoucí rozšíření funkčnosti. V pravé části desky jsou umístěny i senzory SGP40 a HDC2080. Kvůli tepelnému odstínění jsou kolem HDC2080 přidány v desce výřezy, aby bylo eliminováno zkreslení naměřených výsledků v důsledku oteplení od okolních součástek.

Co se týče konstrukčního uspořádání, v rozích DPS jsou přidány distanční sloupky, na kterých pak může být zařízení položeno. Baterie je umístěna na spodní části desky, kam je přichycena plastovým páskem, který funguje na principu suchého zipu – baterii lze tedy v případě potřeby i odepnout a vyměnit. Osazené DPS ukazuje obr. 4.9, kde jsou zatím bez připojené baterie.



Obr. 4.9. Osazené DPS s modulem SCD30 (vlevo) a bez něj (vpravo)

5 Implementace firmware

Pro programování mikrokontrolérů ESP32 je možné využít framework ESP-IDF (Espressif IoT Development Framework) přímo od výrobce, který obsahuje řadu knihoven. Dalšími možnostmi je použití knihoven pro Arduino nebo zkombinování obou těchto způsobů zakomponováním knihoven pro Arduino do projektu postaveného na ESP-IDF jako jednu z komponent. Arduino bylo vyloučeno z důvodu případné potřeby funkcí na nízké úrovni. Po chvíli zkoumání využití knihoven pro Arduino jako komponenty ESP-IDF byl učiněn závěr, že je tato kombinace poměrně křehká ohledně kompatibility verzí obou technologií.

5.1 Použité technologie

Pro implementaci firmwaru bylo zvoleno řešení postavené čistě na ESP-IDF, konkrétně byla použita verze 4.0.2. Jako vývojový nástroj byl využit Visual Studio Code verze 1.54.2 společně s dalšími doinstalovanými doplňky (nejdůležitějším byl Espressif IDF verze 1.0.3). Kromě výchozích knihoven v rámci ESP-IDF byly navíc použity knihovny od výrobce Sensirion a knihovna pro komunikaci s Firestore databází. Veškeré zdrojové kódy mimo knihoven ESP-IDF, které nebyly přibaleny kvůli velikosti, lze najít v přiloženém archivu ve složce *esp_env_monitor*.

5.2 Použité enumerační a datové typy

Před samotným začátkem popisu zdrojových kódů budou v této kapitole pro úplnost uvedeny enumerační a datové typy používané v aplikaci. Díky využití rozsáhlých knihoven se v programu samozřejmě využívá daleko větší množství enumerací a datových typů, to ale není předmětem této práce a vše může být nalezeno v oficiální dokumentaci ([39]) a příslušných knihovnách.

5.2.1 Výsledek operace (*esp_err_t*)

Tento datový typ je definován přímo v knihovnách ESP-IDF, konkrétně v souboru *esp_err.h*. Interně se jedná o typ *int32_t*, který pro různé výsledky operací nabývá různých hodnot. V rámci této aplikace je většinou testováno pouze to, zda operace proběhla

úspěšně, tzn. jestli je návratový kód roven *ESP_OK* (hodnota 0). Všechny možné návratové kódy lze najít v příslušné knihovně.

5.2.2 Výsledek operace s Firestore databází (*firestore_err_t*)

Tato enumerace se používá výhradně při operacích týkajících se komunikace s databází. Je definována v převzaté knihovně pro Firestore, konkrétně v souboru *firestore.h*. V rámci aplikace je opět většinou testováno, jestli operace proběhla v pořádku, tzn. zda je návratový kód roven *FIRESTORE_OK* (hodnota 0). Kompletní enumeraci je opět možné najít v dané knihovně.

5.2.3 Důvod probuzení mikrokontroléru (*env_monitor_wakeup_t*)

Jedná se o enumeraci hodnot možných příčin probuzení mikrokontroléru, tzn. jeho přechodu z úsporného režimu do aktivního. V kódu (konkrétně v souboru *env_monitor.h*) je definována takto:

```
typedef enum {
        ENV_MONITOR_WAKEUP_UNDEFINED,
        ENV_MONITOR_WAKEUP_TIMER,
        ENV_MONITOR_WAKEUP_GPIO34,
        ENV_MONITOR_WAKEUP_GPIO35
} env_monitor_wakeup_t;
```

Zde hodnota *ENV_MONITOR_WAKEUP_TIMER* značí probuzení mikrokontroléru od časovače, *ENV_MONITOR_WAKEUP_GPIO34* udává probuzení od tlačítka SW1, *ENV_MONITOR_WAKEUP_GPIO35* probuzení od tlačítka SW6 (zatím pouze připraveno pro budoucí rozšíření) a hodnota *ENV_MONITOR_WAKEUP_UNDEFINED* představuje všechny ostatní možnosti.

5.2.4 I^2C port (*i2c_port_t*)

Tato enumerace je definována v knihovnách ESP-IDF (konkrétně v souboru *i2c.h*) a vypadá následovně:

```
typedef enum {
        I2C_NUM_0 = 0,
        I2C_NUM_1 ,
        I2C_NUM_MAX
} i2c_port_t;
```

Hodnota *I2C_NUM_0* představuje I^2C port 0, *I2C_NUM_1* značí I^2C port 1. Hodnota *I2C_NUM_MAX* značí celkový počet I^2C portů.

5.3 Popis zdrojových kódů

5.3.1 Soubor battery_adc.c

Funkce *void batt_adc_init()* slouží k inicializaci A/D převodníku pro následné měření napětí baterie. Konfiguruje se zde rozlišení převodníku, útlum na použitém kanálu ADC0, dále jsou nastaveny kalibrační konstanty a provedena inicializace GPIO pinu IO25, který slouží pro připojení napájení k odporovému děliči, na kterém se napětí baterie měří (viz kapitola 4.1.3).

Funkce *bool batt_adc_is_initialized()* vrací hodnotu příznaku *is_adc_initialized*, který jinak není mimo tento soubor přístupný. Příznak udává, zda již byla provedena inicializace A/D převodníku.

Funkce *esp_err_t batt_adc_measure()* nejprve připojí napájení k odporovému děliči R9, R23 pomocí nastavení pinu IO25 do vysoké úrovně a vypne úsporný režim regulátoru TPS63020 nastavením pinu IO26 taktéž do vysoké úrovně. Pak se chvíli počká, než se hodnota ustálí, a následně se provede samotné měření zavoláním funkce *batt_adc_process_measurement()*. Naměřené napětí baterie se poté uloží se do proměnné *v_batt*. Napájení děliče se opět odpojí, povolí se úsporný režim regulátoru a funkce vrátí návratový kód.

Funkce *float batt_adc_process_measurement()* provede měření napětí baterie. Nejprve se změří hodnota napětí na odporu R23, přepočte se na skutečné napětí baterie a tato hodnota je následně vrácena.

Funkce *esp_err_t batt_adc_start_diagnostics()* slouží k provedení diagnostiky A/D převodníku při startu zařízení. Stejně jako u funkce *batt_adc_measure()* se nejprve připojí napájení k odporovému děliči, zakáže se úsporný režim regulátoru a počká se na ustálení hodnoty. Následně se v cyklu měří napětí baterie. Pokud některá z hodnot není v rozmezí 3 až 4,5 V nebo pokud se naměří stokrát stejná hodnota, je vyhodnocena chyba. Po měření se odpojí napájení od odporového děliče, povolí se úsporný režim regulátoru a nakonec je vrácen návratový kód.

Funkce *float batt_adc_get_vbatt()* vrací hodnotu proměnné *v_bat*, která je jinak přístupná pouze v rámci tohoto souboru.

Funkce *bool batt_adc_check_batt_state()* zkontroluje, zda došlo ke změně stavu baterie na základě naměřeného napětí. Pokud hodnota klesla pod 3,3 V, vyhodnotí se, že by baterie měla být dobita. Pokud naopak napětí stouplo nad 3,6 V, stav baterie už není vyhodnocen jako rizikový. Následně se ověří, zda se stav baterie od minulé kontroly změnil, a funkce vrátí tento příznak.

Funkce *bool batt_adc_is_batt_empty()* vrací hodnotu příznaku *is_batt_empty*, který jinak není mimo tento soubor přístupný. Udává, zda je napětí baterie pod rizikovou hodnotou 3,3 V.

5.3.2 Soubor communication.c

Funkce *void com_success()* slouží k indikaci úspěšné operace (např. úspěšné odeslání naměřených dat do databáze). Volá dále funkci pro bliknutí zelené LED.

Funkce *void com_error()* naopak indikuje neúspěšný výsledek operace (např. neúspěšné stažení parametrů z databáze). V tomto případě se rozsvítí červená LED.

Funkce *esp_err_t com_wifi_init_sta()* zinicializuje TCP/IP stack, provede počáteční konfiguraci Wi-Fi v režimu klient a pokusí se připojit k síti. Pokud se to poprvé nepodaří, provede se několik opětovných pokusů. Připojování indikuje žlutě rozsvícená RGB LED, pokud připojování skončí i po několikátém pokusu chybou, zavolá se *com_error()*. Nakonec je vrácen návratový kód operace.

Funkce *void com_event_handler()* je obsluhou událostí nastalých při připojování k síti. Podle toho, která událost nastala, nastavuje příslušné bity do úrovně 1, na základě čehož je program dále větven.

Funkce *bool com_is_wifi_initialized()* vrací příznak *is_wifi_initialized*, který jinak není mimo tento soubor přístupný. Jak vyplývá z názvu, příznak udává, zda je již Wi-Fi zinicializovaná.

Funkce *void com_get_firestore_timestamp_value* slouží k převodu formátu timestampu, který využívá ESP32, na formát, který je používán v databázi Google Firebase. Vstupními parametry jsou *timestamp_value* typu *long*, který reprezentuje formát timestampu mikrokontroléru, a *timestamp_str*, který představuje ukazatel na *char*. Uvnitř funkce se z parametru *timestamp_value* získají jednotlivé složky času (den, měsíc, rok, hodina atd.), ze kterých se následně poskládá textový řetězec ve formátu *yyyy-MM-ddTHH:mm:ssZ* (např. 2021-04-18T22:00:00Z). Tento řetězec je uložen na adresu ukazatele *timestamp_str*.

Funkce *firestore_err_t com_get_document()* vyčte z Firestore databáze požadovaný dokument. Vstupními parametry jsou *collection_id* (ukazatel typu *char*), *document_id* (také ukazatel typu *char*) a *document* (ukazatel na ukazatel typu *char*). Z databáze se vyčte konkrétní dokument z dané kolekce a pokud operace proběhne v pořádku, přijatá data jsou uložena na adresu, kam ukazuje *document*. K indikaci výsledku operace se zde používají funkce *com_success()* a *com_error()*. Nakonec je vrácen návratový kód.

Funkce *firestore_err_t com_add_document()* vytvoří ve Firestore databázi nový dokument s daným identifikátorem v požadované kolekci. Tato funkce je podobná vyčítání dokumentu, jediný rozdíl spočívá v operaci prováděné v databázi.

Funkce *firestore_err_t com_update_document()* aktualizuje ve Firestore databázi dokument s konkrétním identifikátorem v požadované kolekci. Funkce je opět velmi podobná předchozím (vyčítání a vytváření dokumentu).

5.3.3 Soubor hdc2080.c

Funkce *esp_err_t hdc2080_measure()* slouží ke změření teploty a relativní vlhkosti pomocí senzoru HDC2080. Získané hodnoty jsou přiřazeny do proměnných *hdc2080_temp* a *hdc2080_humid*. Následně je vrácen návratový kód.

Funkce *float hdc2080_get_temp()* vrátí hodnotu *hdc2080_temp*, která je jinak přístupná pouze v rámci tohoto souboru.

Funkce *float hdc2080_get_humid()* vrací hodnotu proměnné *hdc2080_humid*, ke které mimo tento soubor jinak nelze přistupovat.

5.3.4 Soubor rgb.c

Funkce *void rgb_pwm_init()* provede počáteční inicializaci časovačů, které jsou při PWM použity. Následně se zinicializují i jednotlivé PWM kanály, ke kterým jsou připojeny diody RGB LED.

Funkce *void rgb_set_duty()* nastaví jednotlivým kanálům RGB LED požadovanou střídu PWM. Vstupními parametry jsou *red_duty*, *green_duty* a *blue_duty*, všechny typu *uint8_t*. Každý z těchto parametrů vyjadřuje střídu PWM v procentech, která se má nastavit danému kanálu.

Funkce *void rgb_light_ms()* také nastaví jednotlivým kanálům RGB LED požadovanou střídu, na rozdíl od předchozí funkce lze ale zvolit i čas, jak dlouho mají LED svítit. Kromě stříd pro jednotlivé kanály je tedy dalším vstupním parametrem *time_ms* typu *uint32_t*.

Funkce *void rgb_stop()* zhasne všechny složky RGB LED. Toho se docílí tak, že u všech kanálů se stopne PWM a daný pin se nastaví na úroveň 1.

5.3.5 Soubor sensirion_i2c.c

Tento soubor sdružuje funkce potřebné pro komunikaci se senzory od firmy Sensirion (SGP40, SCD30). V knihovnách od výrobce se tyto funkce využívají, je i připraven hlavičkový soubor, který definuje jejich vstupní parametry, je ale nutné vytvořit si vlastní soubor *sensirion_i2c.c* a zde funkce implementovat. Takto je zajištěno, že knihovny od výrobce jsou univerzální a lze je použít na jakémkoliv mikrokontroléru.

Funkce *int8_t sensirion_i2c_write()* volá další funkci pro zápis dat přes rozhraní I^2C. Podle adresy senzoru se rozhodne, zda se má pro komunikaci využít I2C0 nebo I2C1 (SCD30 používá samostatné rozhraní I2C1). Vstupními parametry jsou *address* typu *uint8_t*, představující adresu, *data* (ukazatel typu *uint8_t*), reprezentující ukazatel na adresu dat k zápisu, a *count* typu *uint16_t*, udávající délku dat k zápisu. Po provedení operace funkce vrátí návratový kód.

Funkce *int8_t sensirion_i2c_read()* je velmi podobná předchozí funkci. Jediným rozdílem je, že data jsou z přes rozhraní I^2C vyčítána ze senzoru a ukládají se na adresu ukazatele *data*.

Funkce *void sensirion_sleep_usec()* slouží ke zpoždění na požadovanou dobu. Vstupním parametrem je zde *useconds* typu *uint32_t*, který udává počet mikrosekund, po které se má čekat.

5.3.6 Soubor sensors.c

Funkce *esp_err_t sens_i2c0_init()* provede inicializaci komunikačního rozhraní I^2C (port I2C0 pro senzory HDC2080 a SGP40). Po provedení operace je vrácen návratový kód. Velmi podobná je i funkce *esp_err_t sens_i2c1()*, která zinicializuje port I2C1.

Funkce *bool sens_get_i2c0_initialized()* vrací hodnotu příznaku *i2c0_initialized*, který značí, zda již byl port I2C0 zinicializován. Obdobná je i funkce *bool sens_get_i2c1_initialized()*, která vrátí hodnotu příznaku *i2c1_initialized*. Obě proměnné jinak nejsou mimo tento soubor přístupné.

Funkce *esp_err_t sens_i2c_write()* slouží pro zápis dat po I^2C. Vstupními parametry jsou *i2c_num* typu *i2c_port_t*, udávající I^2C port (I2C0 nebo I2C1), *addr* typu *uint8_t*, představující adresu, *data* (ukazatel na *uint8_t*), reprezentující ukazatel na data k zápisu, a *data_len* typu *uint16_t*, udávající délku dat. Po provedení zápisu je vrácen návratový kód operace.

Funkce *esp_err_t sens_i2c_write_byte_to_reg()* umožňuje zápis jednoho bytu dat do určitého registru senzoru, což je užitečné pro komunikaci s HDC2080. Vstupními parametry jsou opět *i2c_port_t i2c_num*, *sens_addr* typu *uint8_t*, představující adresu senzoru, *sens_reg_addr* také typu *uint8_t*, udávající adresu daného registru, a opět následují *data*, tentokrát typu *uint8_t*.

Funkce *esp_err_t sens_i2c_read()* je určena k vyčítání přes I^2C. Je velmi podobná funkci *sens_i2c_write()*, jediným rozdílem je vyčítání dat ze senzoru a jejich následné ukládání na adresu ukazatele *data*.

Funkce *esp_err_t sens_i2c_read_from_reg()* slouží ke čtení dat z určitého registru senzoru. Vstupní parametry jsou *i2c_num* (typ *i2c_port_t)* udávající I^2C port (I2C0/I2C1), *sens_addr* (typ *uint8_t)* představující adresu senzoru, *sens_reg_addr* (typ *uint8_t)* udávající adresu registru senzoru, *data* (ukazatel na *uint8_t*) ukazující na adresu, kam se mají uložit data vyčtená ze senzoru, a nakonec *data_len* (typ *uint16_t*), reprezentující délku dat. Opět je vrácen návratový kód operace.

Funkce *int16_t sens_sdc30_init()* provede počáteční inicializaci senzoru SCD30. Pomocí pinu IO4 se připne k senzoru napájení, nastaví se měřicí interval a odstartuje se periodické měření.

Funkce *void sens_scd30_stop()* zastaví periodické měření pomocí senzoru SCD30 a odpojí mu napájení uvedením pinu IO4 do vysoké úrovně.

5.3.7 Soubor sntp_time.c

Funkce *void initialize_sntp()* slouží k inicializaci SNTP protokolu, pomocí kterého může zařízení následně získat aktuální čas z internetu.

Funkce *esp_err_t sntp_time_obtain_time()* provede synchronizaci systémového času se vzdáleným SNTP serverem. Pokud první pokus o synchronizaci není úspěšný, provede se dalších několik opětovných pokusů. Následně funkce vrátí návratový kód.

Funkce *esp_err_t sntp_time_get_current_time()* interně volá *sntp_time_obtain_time()*. Provede se inicializace SNTP, synchronizace systémového času a poté je aktuální čas vypsán (zvolena časová zóna ČR). Nakonec je vrácen návratový kód operace.

5.3.8 Soubor env_monitor.c

Funkce *void env_monitor_get_mac_id()* slouží k získání jednoznačného identifikátoru zařízení (v podobě MAC adresy), který je dále použit jako identifikátor kolekce naměřených dat ve Firestore databázi. Tento identifikátor je uložen do textového řetězce *mac_id*.

Funkce *void env_monitor_set_default_settings()* přednastaví výchozí parametry (interval pro měření dat, interval pro odesílání dat do databáze a příznaky, zda se má měřit VOC index a koncentrace CO_2).

Funkce *void env_monitor_voc_init()* provede počáteční inicializaci VOC algoritmu vytvořeného firmou Sensirion.

Funkce *void env_monitor_set_default_tasks_time()* nastaví výchozí čas pro provedení jednotlivých úkolů (měření dat, odeslání dat do databáze, změření baterie, změření VOC indexu a zpracování dat VOC algoritmem). Dále se také uloží poslední hodina, kdy byla provedena synchronizace času se SNTP serverem (zaznamená se aktuální hodina, typicky se toto děje hned po startu zařízení a následné synchronizaci).

Funkce *void env_monitor_check_tasks()* zkontroluje čas, kdy mají být znovu vykonány jednotlivé úkoly. Pokud už tento čas u některého úkolu uplynul, aktivuje se příslušný příznak (*measure_task_ticked*, *send_task_ticked*, *battery_measure_task_ticked*, *voc_algorithm_task_ticked*).

Funkce *bool env_monitor_measure_task_ticked()* vrací hodnotu příznaku *measure_task_ticked*, který značí, zda by mělo být provedeno měření. Příznak jinak není mimo tento soubor přístupný.

Funkce *bool env_monitor_send_task_ticked()* vrátí hodnotu proměnné *send_task_ticked*, která udává, zda by měla být naměřená data odeslána do databáze.

Funkce *bool env_monitor_battery_measure_task_ticked()* vrací hodnotu příznaku *battery_measure_task_ticked*, který udává, zda by mělo být změřeno napětí baterie.

Funkce *bool env_monitor_voc_algorithm_task_ticked()* vrátí hodnotu proměnné *voc_algorithm_task_ticked*, která značí, zda by mělo být provedeno měření VOC indexu a následné zpracování dat VOC algoritmem.

Funkce *bool env_monitor_is_initialized()* vrací hodnotu příznaku *is_initialized*, který udává, zda proběhla počáteční inicializace po startu zařízení a jsou přednastavené nějaké výchozí parametry (tzn. zda již byla volána funkce *env_monitor_set_default_settings()*).

Funkce *bool env_monitor_get_sntp_retry_next()* vrátí hodnotu příznaku *sntp_retry_next*, který značí, zda se při posledním pokusu o synchronizaci času se SNTP serverem vyskytl problém a synchronizace by měla být provedena znovu.

Funkce *void env_monitor_set_sntp_retry_next()* slouží k nastavení hodnoty příznaku *sntp_retry_next*, který jinak není mimo tento soubor přístupný.

Funkce *bool env_monitor_get_load_settings_retry_next()* vrací hodnotu proměnné *load_settings_retry_next*, která značí, zda se při posledním pokusu o načtení parametrů z databáze vyskytl problém a akce by měla být provedena znovu.

Funkce *void env_monitor_set_default_settings_retry_next()* je naopak určena k nastavení hodnoty příznaku *load_settings_retry_next*.

Funkce *env_monitor_get_sntp_sync_time_flag()* vyhodnotí, zda by měla být provedena synchronizace času se SNTP serverem na základě porovnání aktuálního času

zařízení vyčteného z RTC a proměnné *prev_sntp_sync_hour*. Synchronizace by se měla provádět vždy po půlnoci a poté každých 6 hodin.

Funkce *int env_monitor_count_sleep_time_sec()* spočítá, na jak dlouho se má mikrokontrolér uvést do úsporného režimu deep-sleep. Tato doba je počítána tak, aby se zařízení opět aktivovalo v čas, kdy se má vykonat nejbližší úkol. Návratovou hodnotou je výsledek výpočtu v sekundách.

Funkce *void env_monitor_sleep_start()* slouží k uvedení mikrokontroléru do úsporného režimu deep-sleep. Nejprve se povolí probuzení pomocí časovače po době, která je určena návratovou hodnotou funkce *env_monitor_count_sleep_time_sec()*. Poté je povoleno i probuzení pomocí pinů IO34 a IO35 (tlačítka) a nakonec se aktivuje úsporný režim.

Funkce *env_monitor_wakeup_t env_monitor_get_wakeup_cause()* rozpozná, jakým způsobem bylo zařízení probuzeno z úsporného režimu deep-sleep. Poté je vrácena příslušná hodnota enumerace *env_monitor_wakeup_t*.

Funkce *esp_err_t env_monitor_load_settings()* vyčte z Firestore databáze parametry zařízení s příslušným identifikátorem. Přijatá data se zpracují a provede se rekonfigurace systémových parametrů – interval měření (*measure_time*), interval odesílání dat do databáze (*send_time*) a příznaky, zda se má měřit VOC index (*measure_voc*) a koncentrace CO_2 (*measure_co2*). Aby byly jednotlivé úkoly synchronizovány a zařízení nebylo zbytečně aktivováno z úsporného režimu, přenastaví se následně i čas příštího provedení úkolů. Nakonec je vrácen návratový kód operace.

Funkce *int16_t env_monitor_measure_voc_index()* změří VOC index a předá data VOC algoritmu k dalšímu zpracování. Tato operace by se dle výrobce senzoru měla provádět pravidelně zhruba jednou za sekundu kvůli adaptaci algoritmu na měřené prostředí. Nakonec je vrácen návratový kód operace měření.

Funkce *int16_t env_monitor_process_voc_algorithm()* měří VOC index voláním *env_monitor_measure_voc_index()* a stará se o přídavné měření dalších potřebných hodnot. Kvůli kalibracím je totiž třeba mít k dispozici i hodnoty teploty a relativní vlhkosti,

které se mimo hlavní měření aktualizují nejdéle vždy po 20 sekundách. Na konci funkce vrátí návratový kód operace měření VOC indexu.

Funkce *esp_err_t env_monitor_measure_data()* změří sledované veličiny (teplotu, relativní vlhkost, případně VOC index a koncentraci CO_2) a uloží je do příslušných proměnných. Koncentrace CO_2 je měřena čtyřikrát a ukládají se až poslední data z důvodu, aby se naměřená hodnota mohla ustálit při připojení napájení danému senzoru. Následně se aktualizuje čas příštího měření v proměnné *next_measure_time* a konec celé operace je indikován bliknutím modré LED. Nakonec funkce vrátí návratový kód.

Funkce *firestore_err_t env_monitor_send_data()* nejprve zjistí, zda jsou nějaká data uložená ve frontě dosud neodeslaných dat. Pokud ano, funkce dále pokračuje voláním *env_monitor_send_data_from_buff()*. Pokud je tato fronta prázdná, vytvoří se JSON, do kterého se následně přidají naměřená data včetně příslušného timestampu (vytvořeného funkcí *com_get_firestore_timestamp_value()*). Poté jsou data odeslána do Firestore databáze, kde se vytvoří nový dokument v příslušné kolekci. Pokud odesílání čerstvě naměřených dat selže, zařadí se tento záznam do fronty, kde je uložen až do dalšího odesílání. Následně se aktualizuje čas příštího odeslání a nakonec je vrácen návratový kód.

Funkce *firestore_err_t env_monitor_send_data_from_buff()* slouží k odeslání dat uložených ve frontě do databáze. Vytvoří se JSON, do kterého se přidají jednotlivé záznamy ve frontě. Pokud se právě měřilo, přidají se i čerstvě naměřené hodnoty. Poté se data odešlou v rámci jednoho dotazu do Firestore databáze k dalšímu zpracování, v případě úspěšné operace se vyprázdní fronta a následně funkce vrátí návratový kód.

Funkce *void env_monitor_add_data_to_buff()* přidá právě naměřená data do fronty dosud neodeslaných dat. Nejprve se zkontroluje, zda se záznam do fronty ještě vejde, a pokud ano, jsou tyto data zapsány na konec fronty.

Funkce *firestore_err_t env_monitor_send_batt_state_change()* odešle stav baterie do Firestore databáze. Pokud se operace nepodaří, aktivuje se příznak *batt_state_sync*, pomocí kterého lze při dalším průběhu rozpoznat, zda by se mělo odeslání stavu baterie zopakovat. Nakonec je vrácen návratový kód operace.

Funkce *void env_monitor_check_battery()* slouží ke kontrole stavu baterie. Provede se inicializace A/D převodníku a změří se napětí baterie. Pokud došlo ke změně stavu oproti předchozí kontrole, nebo pokud se minule nepodařilo aktualizovat stav v databázi (je aktivní příznak *batt_state_sync*), odešle se nový stav baterie do databáze pomocí *env_monitor_send_batt_state_change()*. Nakonec se aktualizuje čas příští kontroly a funkce vrátí návratový kód.

Funkce *esp_err_t env_monitor_start_diagnostics()* provede počáteční diagnostiku zařízení. Otestuje se RGB LED, UART, A/D převodník a také zda je funkční komunikace se senzory HDC2080 a SGP40. Komunikace se senzorem SCD30 se testuje samostatně při každém měření, protože je z desky odjímatelný.

Funkce *esp_err_t env_monitor_enable_tps_power_save_mode()* povolí úsporný režim regulátoru TPS63020 stažením pinu IO26 do nízké úrovně. Následně je vrácen návratový kód.

Funkce *esp_err_t env_monitor_disable_tps_power_save_mode()* naopak zakáže úsporný režim regulátoru nastavením pinu IO26 do vysoké úrovně.

5.3.9 Soubor main.c

Chod celého programu vypadá následovně. Na úplném začátku je zinicializována paměť příkazem *nvs_flash_init()*, případně uvolněna příkazem *nvs_flash_erase()*, a provede se počáteční inicializace PWM. Poté se zjistí příčina probuzení mikrokontroléru (přechodu do aktivního režimu z úsporného), na základě čehož se program dále větví.

Pokud příčinou probuzení není ani časovač, ani tlačítka (součástky SW1, SW6), znamená to, že bylo zařízení pravděpodobně restartováno nebo nově připojeno k napájení. Pokud program běží prvně po připojení napájení, zinicializuje se VOC algoritmus. Provede se počáteční diagnostika, získá se jednoznačný identifikátor zařízení v podobě jeho MAC adresy, nastaví se výchozí parametry a povolí se úsporný režim regulátoru. Zinicializuje se Wi-Fi a provede se synchronizace času podle SNTP serveru. Pokud se to nepodaří, aktivuje se příznak *sntp_retry_next* a zařízení se přepne do úsporného režimu, protože bez aktuálního času nemá další funkce smysl. Pokud je tedy při dalším průchodu příznak

sntp_retry_next aktivní, provede se pokus o synchronizaci času znovu. Tato akce je prováděna při jakékoliv příčině probuzení mikrokontroléru vždy po půlnoci a následně každých 6 hodin. Jakmile je nastaven aktuální čas, nastaví se i výchozí časy provedení jednotlivých úkolů. Pokud došlo k restartu zařízení, zinicializuje se komunikace s Firestore databází, odkud se stáhne konfigurace daného zařízení. Pokud se vyčtení parametrů uložených v databázi z nějakého důvodu nepodaří, přednastaví se výchozí hodnoty a aktivuje se příznak *load_settings_retry_next*, díky kterému se pokus o načtení parametrů provede znovu při příštím průchodu programu. Dále program pokračuje akcemi popsanými v následujícím odstavci, které jsou společné i pro probuzení časovačem.

V případě probuzení mikrokontroléru časovačem se nejprve zkontrolují úkoly, které mají být provedeny. V případě aktivního příznaku *battery_measure_task_ticked* se zkontroluje stav baterie a pokud došlo k nějaké změně, odešle se do databáze. Pokud se akce nezdaří, stav baterie se v databázi aktualizuje až při příští plánované kontrole baterie (tj. za 1 hodinu). Pokud je aktivní příznak *measure_task_ticked*, provede se inicializace I^2C a změří se monitorované veličiny (teplota a relativní vlhkost se snímají vždy, VOC index a koncentrace CO_2 jsou volitelné). Pokud je aktivní příznak *send_task_ticked*, zinicializuje se Wi-Fi a komunikace s Firestore a naměřená data se rovnou odešlou do databáze. V opačném případě se data přidají do fronty, kde jsou uchována až do doby, než uplyne čas do jejich odeslání. Data se ukládají do fronty i pokud se jejich odeslání z nějakého důvodu nepodaří. Pokud je uživatelem povoleno měření VOC indexu, zároveň je aktivní příznak *voc_algorithm_ticked* a v daném průchodu programu se neprovádí kompletní měření monitorovaných veličin, naměří se pouze data potřebná pro VOC algoritmus, kterému jsou následně předána k dalšímu zpracování. Nakonec je mikrokontrolér opět uveden do úsporného režimu (deep-sleep).

Pokud je příčinou probuzení tlačítko SW1, provede se inicializace Wi-Fi a komunikace s Firestore databází. Následně se stáhne aktuální konfigurace příslušného zařízení a provede se přenastavení systémových parametrů. Pokud se tato operace nezdaří, aktivuje se příznak *load_settings_retry_next*, který zajistí opětovný pokus o načtení parametrů z databáze při příštím průchodu programu. Poté se mikrokontrolér uvede zpět do úsporného režimu.

6 Uživatelské rozhraní (frontend)

Uživatelská aplikace je postavena na moderních webových technologiích. Skládá se z uživatelského rozhraní (frontendu) a cloudové platformy Google Firebase. Uživatelské rozhraní bude podrobněji popsáno v následujících podkapitolách, platformě Google Firebase je věnována samostatná kapitola 7. Zdrojové kódy pro obě tyto části lze najít v přiloženém archivu ve složce *env-monitor-frontend*.

6.1 Použité technologie

Frontend uživatelské aplikace byl vytvořen pomocí platformy Angular verze 10.1.7. Jako vývojový nástroj byl použit stejně jako v předchozím případě Visual Studio Code verze 1.54.2. Dále byl pro tuto aplikaci využit Node.js verze 14.16.0, balíčkovací systém npm verze 6.14.11 a programovací jazyk TypeScript verze 4.0.7. Pro uživatelské rozhraní byly použity komponenty knihovny Angular Material verze 10.2.7 a pro vizualizaci naměřených dat knihovna ApexCharts verze 3.26.0.

6.2 Datové typy

V této podkapitole budou probrány datové typy vytvořené v rámci aplikace. Datové typy převzaté z použitých knihoven nejsou předmětem této práce a jejich popis může být ve většině případů nalezen v příslušné dokumentaci.

6.2.1 Parametry měřicího uzlu (*DeviceSettings*)

Tato třída představuje systémové parametry měřicího uzlu. Jednotlivé atributy vypadají následovně:

```
class DeviceSettings {
        id: string;
        sendTime: number;
        measureTime: number;
        measureVoc: boolean;
        measureCo2: boolean;
}
```

Atribut *id* představuje identifikátor zařízení, atribut *sendTime* interval odesílání dat z měřicího uzlu do databáze a *measureTime* interval měření. Oba intervaly jsou udávány v sekundách. Atribut *measureVoc* značí, zda má dané zařízení měřit VOC index, a *measureCo2* udává, zda se má měřit koncentrace CO_2.

6.2.2 Stav měřicího uzlu (*DeviceState*)

Tato třída je rozšířením třídy *DeviceSettings*. Obsahuje navíc atribut *empty*, který značí, zda by měla být baterie daného zařízení dobita. V kódu je třída definována následovně:

```
class DeviceState extends DeviceSettings {
        empty: boolean;
}
```

6.2.3 Timestamp používaný ve Firestore (*FirestoreTimestamp*)

Třída *FirestoreTimestamp* představuje formát timestampu používaný Firestore databází. Je definován takto:

```
class FirestoreTimestamp {
        nanoseconds: number;
        seconds: number;
}
```

Zde atribut *seconds* udává sekundy, které uběhly od 1.1.1970 00:00:00 UTC (koordinovaného světového času). Atribut *nanoseconds* představuje další upřesnění, pokud by to bylo potřeba.

6.2.4 Naměřená data (*DeviceData*)

Tato třída slouží pro uložení dat naměřených v určitý časový okamžik. Atributy jsou definovány následovně:

```
class DeviceData {
        timestamp: FirestoreTimestamp | Date;
        temp: number;
        humid: number;
        vocIndex?: number;
        co2ppm?: number;
}
```

Atribut *timestamp* představuje čas, ve který byla data naměřena, atribut *temp* udává teplotu ve stupních Celsia, *humid* reprezentuje relativní vlhkost v procentech, *vocIndex* udává VOC index (bezrozměrná veličina) a atribut *co2ppm* představuje koncentraci CO_2 v jednotkách ppm (parts per million). Atributy s otazníkem jsou nepovinné, u některých měření mohou tyto hodnoty chybět.

6.3 Popis společné části zdrojových kódů

V této části budou rozebrány obecné části kódu využívané napříč aplikací. Konkrétní funkčnosti jednotlivých obrazovek pak budou popsány samostatně v další části.

6.3.1 Třída AuthService

Tato třída sdružuje metody pro ověření uživatele, který k aplikaci přistupuje. Před samotným přístupem k jednotlivým funkčnostem je nutné se nejprve přihlásit pomocí kombinace e-mailu a hesla. Využívá se zde možnosti autentizace pomocí platformy Google Firebase. Pro vzájemnou komunikaci mezi touto platformou a aplikací je použita knihovna *angular/fire*.

Metoda *get authenticated()* vrací příznak, zda je uživatel přihlášen. Podobně funguje i metoda *get currentUser()*, která vrátí informace o přihlášeném uživateli.

Metoda *signIn()* slouží k přihlášení uživatele. Vstupními parametry jsou *email* typu *string*, *password* typu *string*, *onSuccess* a *onError*. Poslední dva parametry jsou odkazy na další metody (callbacky). Při úspěšném přihlášení se pak provede *onSuccess()*, při neúspěšném naopak *onError()*.

Metoda *signOut()* odhlásí aktuálně přihlášeného uživatele. Vstupními parametry jsou odkazy na další metody – *onSuccess* a nepovinný *onError*. Při úspěšném odhlášení se zavolá *onSuccess()*, pokud naopak nastane během této operace chyba, provede se *onError()*, pokud je tato metoda definována.

Metoda *resetPasswordEmail()* slouží pro resetování hesla v případě, že ho uživatel zapomněl. Vstupními parametry jsou *email* typu *string* a opět odkazy na metody *onSuccess* a *onError*, které se zavolají na základě výsledku operace.

Metoda *checkLocalStorage()* zpracuje data uživatele uchovávané v lokálním úložišti prohlížeče a výsledek uloží do proměnné *authUser*, ze které pak lze dále zjistit, zda je uživatel přihlášen, o jakého uživatele se jedná atd. Tato metoda je volána v konstruktoru třídy.

6.3.2 Třída EnvMonitorService

Zde jsou sdruženy metody pro komunikaci s Firestore databází a případnou transformaci dat při předávání. Interně je zde opět použita knihovna *angular/fire*.

Metoda *getSettings()* vyčte z databáze všechny dokumenty v kolekci *settings*, tedy systémové parametry všech zařízení.

Metoda *updateDeviceSettings()* zaktualizuje konkrétní dokument v kolekci *settings*, tzn. slouží k úpravě systémových parametrů určitého zařízení. Vstupními parametry jsou *deviceSettings* typu *DeviceSettings* a odkazy na metody *onSuccess* a *onError*, které jsou provedeny na základě výsledku operace.

Metoda *getMeasuredData()* vyčte z databáze naměřená data určitého zařízení. Vstupními parametry jsou *deviceId* typu *string*, představující identifikátor daného uzlu, a nepovinný *limit* typu *number*, který udává, kolik položek se má maximálně vrátit. Naměřené hodnoty jsou vyčítány od nejnovějších směrem dál ke starším. Pokud není do metody předán žádný limit, vyčtou se všechna data obsažená v dané kolekci.

Metoda *getBatteryState()* slouží k vyčtení dat z kolekce *batteryState*, kde je uložen stav baterie jednotlivých měřicích zařízení.

Metoda *convertTimestampToDate()* slouží k transformaci času z formátu, který využívá Firestore databáze, na formát typu *Date*, který je používán v aplikaci. Vstupním

parametrem je *timestamp* typu *FirestoreTimestamp*, metoda vrací převedená data typu *Date*.

Metoda *isNumber()* ověří, zda je daná hodnota číslo. Vstupním parametrem je *value* typu *any*, návratovou hodnotou je výsledek porovnání.

Metoda *convertToNumber()* zkonvertuje požadovanou hodnotu na datový typ *number*. Vstupním parametrem je *value* typu *any*. Nejprve se otestuje, zda lze daná hodnota převést na číslo, a pokud ano, vrátí se převedené číslo.

6.3.3 Třída AuthGuard

Tato třída je využívána napříč celou aplikací a kontroluje, zda má uživatel přístup na konkrétní stránku (obrazovku). Tyto přístupy jsou definovány v souboru *app-routing.module.ts*. Nepřihlášený uživatel je oprávněn vstoupit pouze na přihlašovací stránku a na obrazovku s formulářem pro resetování hesla.

Metoda *canActivate()* vyhodnotí, zda je uživatel přihlášen. Pokud ano, je vše v pořádku, v případě pokusu o neoprávněný přístup je přesměrován na přihlašovací stránku.

6.3.4 Direktiva NumberValidatorDirective

Pomocí této direktivy lze správně validovat číselné hodnoty na formulářích. Vznikla jako řešení ke komponentě Angular Material Input typu *number*, která neprováděla validace dle očekávání.

Metoda *validate()* provádí samotnou validaci nad formulářovým prvkem, ke kterému je direktiva připojena. Zkontroluje se, jestli je vstupní hodnota číslo a následně i zda je toto číslo v povolených mezích. Na základě toho je vyhodnoceno, zda je hodnota daného formulářového prvku validní, případně je vygenerován seznam příslušných chyb.

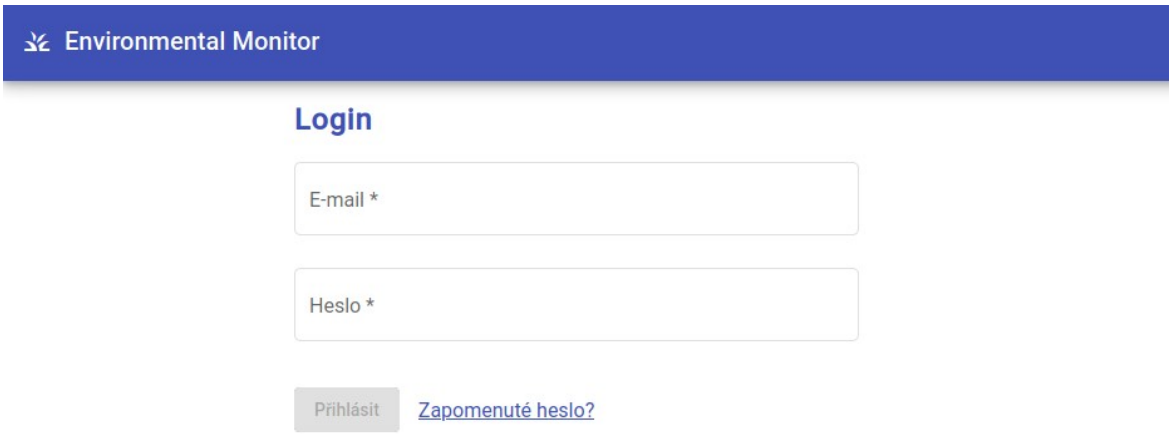
Metoda *isNumber()* vyhodnotí, zda je daná hodnota číslo. Zkontroluje se vstupní parametr *value* typu *string* nebo *number* a následně je vrácen výsledek této operace.

Metoda *convertInputsToNum()* převede povolené meze na datový typ *number*, pokud je to možné. Tyto mezní hodnoty se při běhu programu vyhodnocují dynamicky a pokud by některá z nich byla typu *string*, tento validátor by nefungoval správně.

6.4 Uživatelské obrazovky

6.4.1 Přihlašovací obrazovka

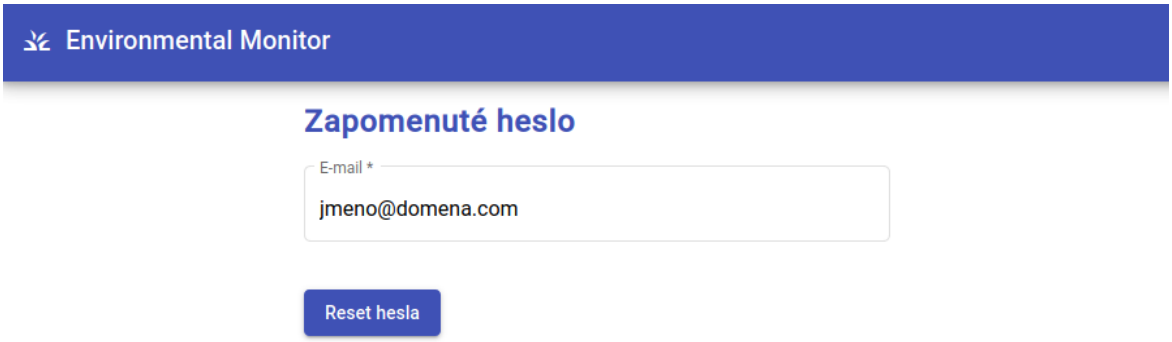
Tato obrazovka obsahuje přihlašovací formulář pro zadání e-mailu a hesla. Pokud je již uživatel přihlášen, je v aplikaci automaticky přesměrován na stránku s přehledy. V opačném případě vyplní přihlašovací údaje a klikne na tlačítko *Přihlásit*. Po úspěšném přihlášení je provedeno přesměrování na přehledovou stránku. Pokud uživatel zapomněl heslo, klikne na odkaz *Zapomenuté heslo?* pod formulářem, který vede na stránku pro resetování hesla.



Obr. 6.1. Přihlašovací obrazovka webové aplikace

6.4.2 Zapomenuté heslo

Pomocí této obrazovky může uživatel resetovat své heslo. Na stránce je obsažen pouze formulář pro zadání e-mailu uživatele, kterému má být heslo resetováno. Uživatel zadá svůj e-mail, stiskne tlačítko *Reset hesla* a na danou adresu je mu následně zaslán e-mail s odkazem pro změnu hesla. Pokud se na tuto stránku dostane přihlášený uživatel, je automaticky přesměrován na obrazovku s přehledy.



Obr. 6.2. Obrazovka pro resetování hesla

6.4.3 Přehledová obrazovka

Přehledová stránka slouží pro vizualizaci aktuálního stavu jednotlivých měřicích uzlů a naměřených hodnot. V horní části obrazovky se nachází rozbalovací panel s přehledem zařízení připojených do systému a jejich aktuálním stavem. Pro každý uzel je zde zobrazen identifikátor, systémové parametry a stav baterie.

Obr. 6.3. Přehledová obrazovka – aktuální stav měřicích uzlů

Nad grafy je umístěno výběrové pole, kde si uživatel zvolí, na jaká data se chce podívat (jakým uzlem byly naměřeny). Po kliknutí do tohoto pole se otevře rozbalovací seznam s identifikátory jednotlivých uzlů, ve kterém lze filtrovat zadáním části textu. Uživatel může vybrat více zařízení najednou. Při vstupu na stránku je ve výchozím stavu již jeden identifikátor předvybrán.

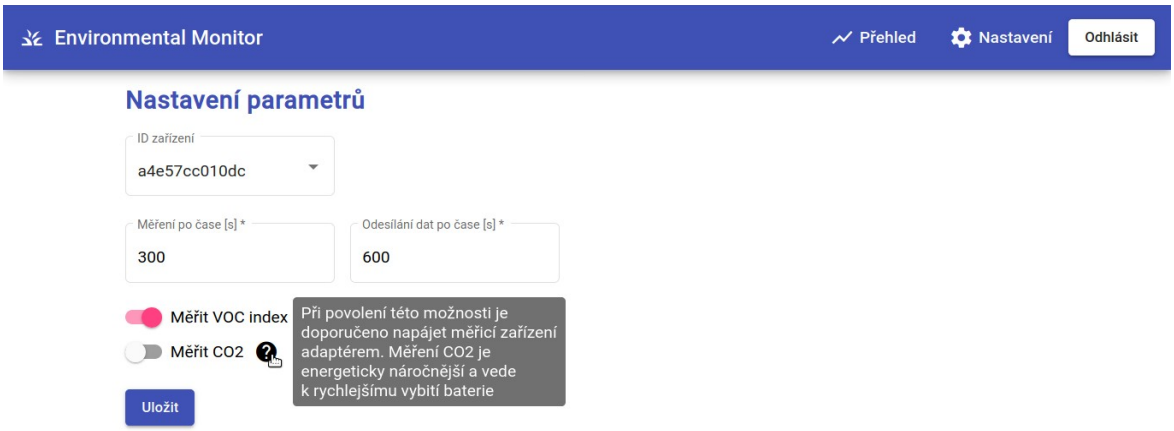
Obr.6.4. Přehledová obrazovka - výběrové pole

Obrazovka dále obsahuje čtyři grafy pro jednotlivé veličiny – teplota, relativní vlhkost, VOC index a koncentrace CO_2. Lze tedy porovnat údaje naměřené jednotlivými uzly, jednotlivé křivky lze navíc v grafech individuálně vypínat. Grafy se chovají interaktivně, uživatel si tedy může např. detailněji přiblížit určité období. Hodnoty lze také vyexportovat pro případné další zpracování.

Obr.6.5. Přehledová obrazovka - grafy

6.4.4 Nastavení parametrů

Tato obrazovka obsahuje formulář pro nastavení systémových parametrů jednotlivých měřicích uzlů. Uživatel si v rozbalovacím seznamu vybere identifikátor uzlu, provede požadované změny parametrů a následně stiskne tlačítko *Uložit*. Tím se nové parametry uloží do databáze, odkud mohou být následně vyčteny příslušným měřicím uzlem.



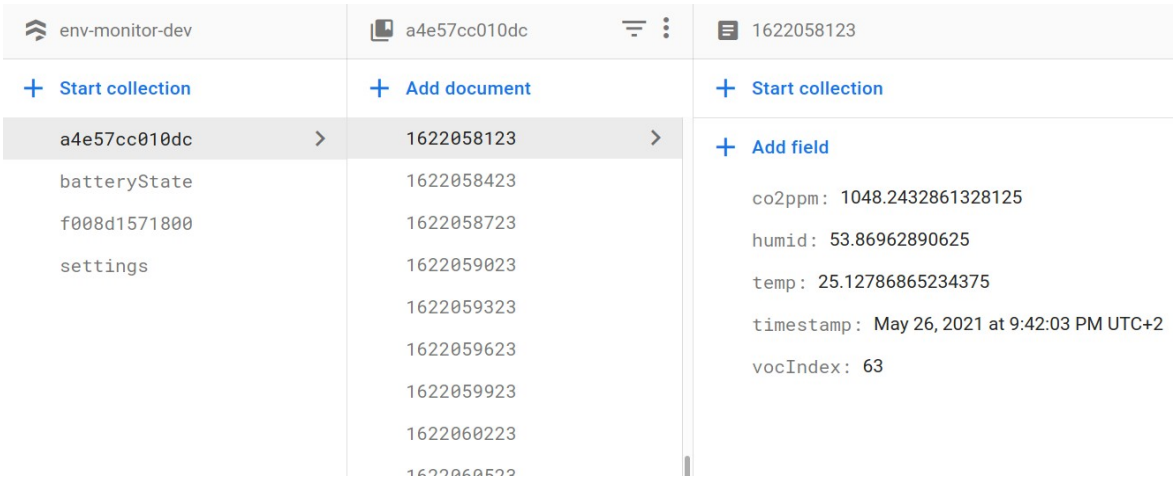
Obr. 6.6. Obrazovka nastavení parametrů

7 Cloudová platforma Google Firebase

Poslední částí, kterou zbývá popsat, je cloudová platforma Google Firebase a její použití v této aplikaci. Tato platforma nabízí řadu služeb, které lze využít jak pro vývoj, tak pro různé analýzy. Je určena hlavně pro mobilní a webové aplikace a pro práci s ní je dostupná řada knihoven. V rámci vývoje poskytuje např. databázi, strojové učení, cloudové funkce, nástroje pro autentizaci, hosting webových aplikací a úložiště souborů. Co se týče nástrojů pro analýzu, nabízí např. možnost reportování chyb v reálném čase, sestavení časové osy příčin, které k dané chybě vedly, okamžité upozornění na kritické chyby, sledování výkonnosti aplikace, nástroje pro distribuci aplikace (správa verzí) nebo zasílání notifikací připojeným zařízením (podpora systémů Android a iOS). Google Firebase také poskytuje nástroje pro analýzu různých údajů – např. z jaké země přistupuje k aplikaci nejvíce uživatelů, jak efektivní je zasílání notifikací (případně jakých) a podobně, což může být následně využito např. pro zlepšení marketingu. Tuto platformu zajišťuje společnost Google, lze ji tedy bez problému integrovat i do dalších systémů od Google, např. Jira, Slack, Play Store, Google Marketing Platform apod. Kompletní přehled služeb lze nalézt na oficiálním webu Google Firebase. V rámci této aplikace je využito pouze několika nástrojů pro vývoj, které budou podrobněji probrány v následujících podkapitolách.

7.1 Firestore Databáze

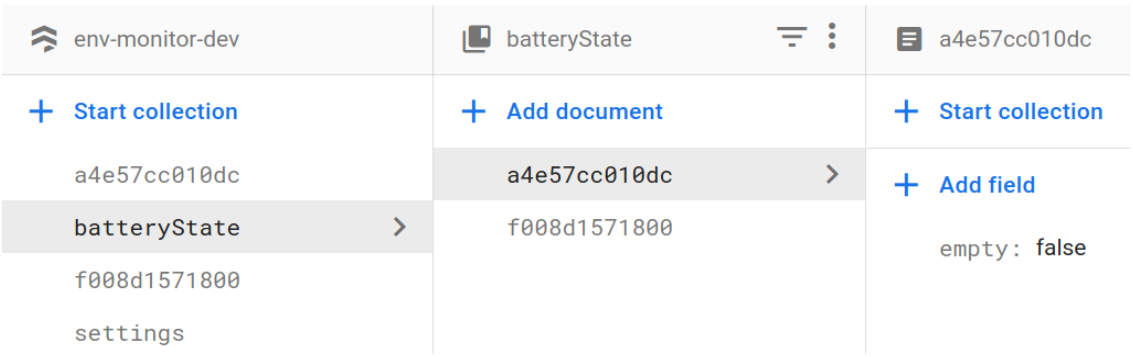
Používaná databáze je typu NoSQL (nerelační), konkrétně dokumentově orientovaná. Tyto databáze dokáží zpracovat velké objemy nestrukturovaných dat, které se rychle mění. Oproti relačním databázím neobsahují jasně danou strukturu se sloupci a řádky, ale skládají se z kolekcí, do kterých jsou zařazeny jednotlivé dokumenty. Ty pak obsahují samotná data. Výborně se hodí pro ukládání dat přijatých ve formátu JSON, pro použití v této aplikaci se tedy tento typ databáze přímo nabízí. Struktura uložených dat je patrná z obr. 7.1.



Obr. 7.1. Struktura naměřených dat v databázi

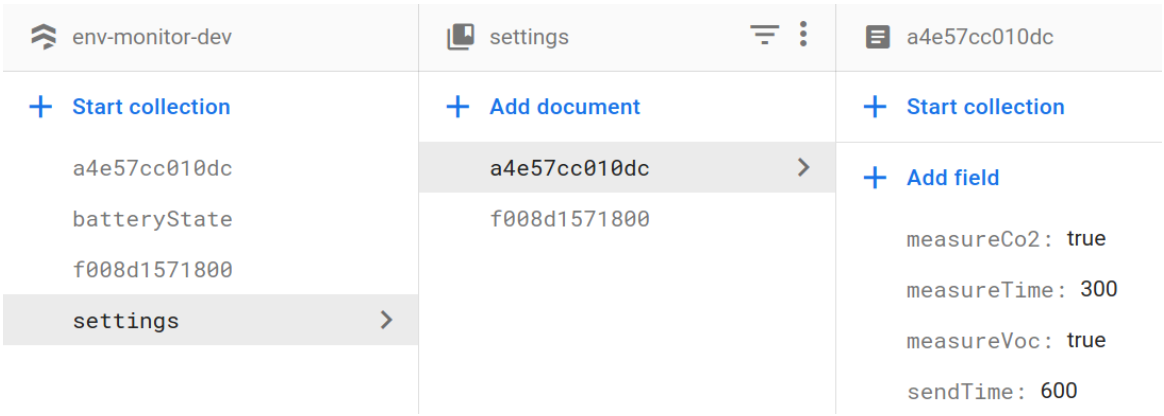
Kolekce *a4e57cc010dc* a *f008d1571800* představují data nashromážděná měřicími uzly připojenými do systému. Jednoznačnými identifikátory těchto kolekcí jsou MAC adresy měřicích zařízení. Tyto kolekce obsahují jednotlivé dokumenty s naměřenými daty. Jako identifikátory dokumentů byly zvoleny timestampy s časem příslušného měření, čímž je zajištěna jejich jednoznačnost.

Další kolekcí je *batteryState*, obsahující dokumenty s aktuálním stavem baterie jednotlivých zařízení v systému. Zde jsou jako identifikátory dokumentů použity identifikátory uzlů. Uspořádání je znázorněno na obr. 7.2.



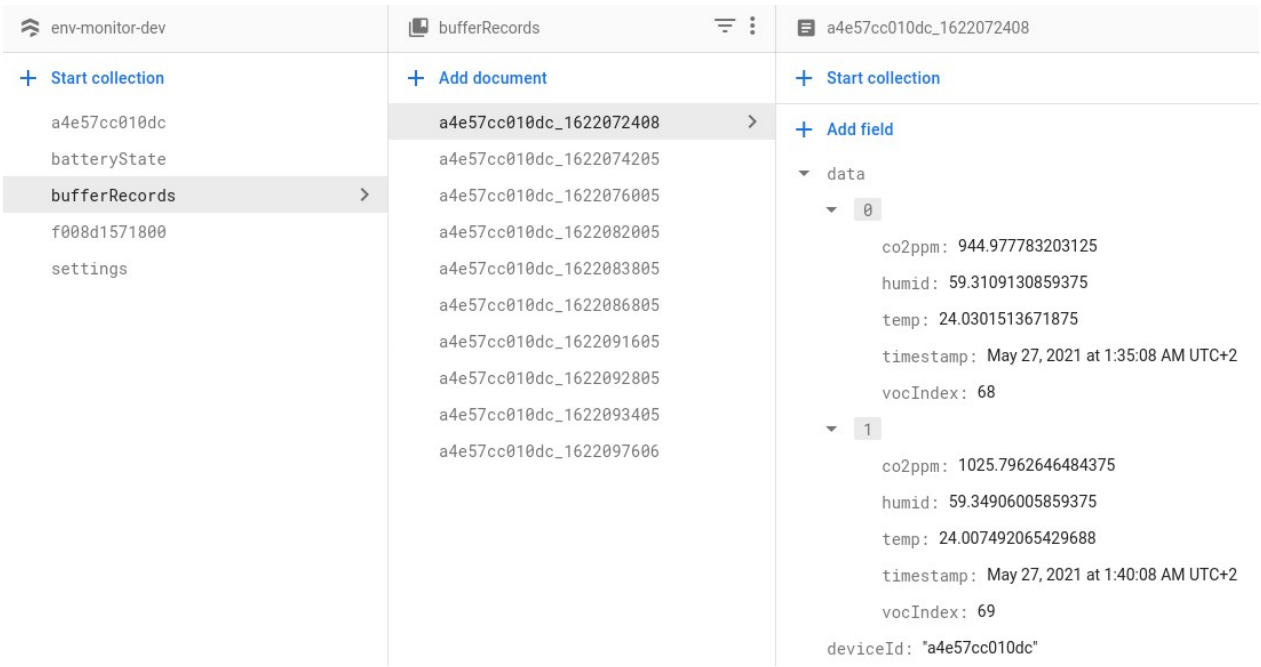
Obr. 7.2. Kolekce s aktuálním stavem baterie měřicích uzlů

Kolekce *settings* obsahuje systémové parametry jednotlivých měřicích zařízení. Opět jsou zde jako identifikátory dokumentů použity identifikátory příslušných uzlů. Struktura je patrná z obr. 7.3.



Obr. 7.3. Kolekce se systémovými parametry měřicích uzlů

V databázi je obsažena ještě další pomocná kolekce *bufferRecords*, která slouží k zachycení příchozích dat, když se z měřicích zařízení posílá místo jednoho záznamu celá fronta nashromážděných dat. Identifikátory dokumentů jsou zde ve formátu *idZarizeni_timestampOdeslani*. Tato struktura je znázorněna na obr. 7.4. V běžném provozu je tato kolekce prázdná, její účel bude podrobněji popsán v kapitole 7.3.1.



Obr. 7.4. Kolekce k příjmu fronty nashromážděných dat

7.2 Autentizace

Jak již bylo zmíněno, přístup k aplikaci je řízen pomocí autentizace platformy Google Firebase. Ta podporuje ověřování uživatelů pomocí různých metod – kombinace e-mailu a hesla, telefonní číslo, účet Google, Facebook, Twitter a další.

Pro tuto aplikaci bylo zvoleno přihlašování pomocí kombinace e-mailu a hesla. Uživatel zadá příslušné údaje do webového formuláře, který je dál předá nástroji Firebase Authentication SDK. Backendové služby ověří správnost údajů a vrátí klientovi výsledek operace.

Autentizace pomocí platformy Google Firebase se navíc stará i o resetování zapomenutého hesla a případnou registraci nových uživatelů (v této aplikaci není registrace implementována). V nastavení je možné upravit texty e-mailů, které se automaticky rozesílají při těchto akcích. Kompletní informace o autentizaci lze najít v [40].

7.3 Cloudové funkce

Pomocí cloudových funkcí lze provádět různé operace, které se vykonávají na serverech. Pro tuto aplikaci byly použity funkce nad Firestore databází, ale obecně je lze provádět i nad dalšími nástroji platformy Google Firebase. Typickým příkladem může být nahrávání obrázku do webové aplikace, kdy se po přijetí vyvolá funkce pro jeho zmenšení a pokud se operace podaří, původní soubor je v úložišti nahrazen.

V této aplikaci bylo využito spouštění funkcí určitou akcí ve Firestore databázi, konkrétně vytvořením nového souboru v požadované kolekci. Zdrojové kódy jsou k dispozici v přiloženém archivu ve složce *env-monitor-frontend/functions*.

7.3.1 Dávkové zpracování dat při přijetí

Tato funkce slouží k dávkovému zpracování dat, která se z měřicích uzlů odešlou najednou. Jedná se o případ, kdy měřicí uzel nasbírá určité množství dat, která si ukládá do fronty, a když přijde čas na odeslání do databáze, pošle všechny záznamy najednou

v rámci jednoho dotazu. Do databáze je tedy zapsán dokument, jehož podoba je znázorněna na obr. 7.4. Poté je třeba tento dokument upravit pro následné snazší vyčítání, což je úkolem této funkce.

Při zápisu nového dokumentu do kolekce *bufferRecords* se vyvolá tato funkce. Pomocí atributu *deviceId* se zjistí, které zařízení data odeslalo. Následně se projdou jednotlivé záznamy obsažené v atributu *data* a uloží se jako samostatné dokumenty v kolekci naměřených dat příslušného zařízení (kolekce s identifikátorem *deviceId*), kde identifikátor dokumentu je vytvořen z času měření (atribut *timestamp*). Když se takto zpracují všechny záznamy daného dokumentu, původní dokument je z kolekce *bufferRecords* smazán, aby zbytečně nezabíral místo.

Závěr

Předmětem této práce byl návrh systému pro centrální sběr a zpracování dat ze vzdálených senzorických uzlů pro měření environmentálních veličin. Byl navržen a implementován hardware měřicího uzlu, o kterém pojednává kapitola 4. Dále byl implementován firmware tohoto zařízení, který je podrobně popsán v kapitole 5. Poté byl proveden návrh a implementace uživatelského rozhraní webové aplikace pro parametrizaci měření a vizualizaci naměřených hodnot včetně stavu uzlů. Nakonec byla implementována i cloudová funkce pro dávkové zpracování většího množství příchozích dat.

Co se týče měření veličin, z grafů na obr. 6.5 je patrné, že hodnoty koncentrace CO_2 lehce kolísají, což je nejspíš způsobeno odpojováním napájení mimo samotné měření. Byl zde zvolen kompromis mezi přesností naměřených hodnot a úsporou energie pro případ, že je zařízení napájené z baterie. Zjistit, kdy jsou získané hodnoty nejpřesnější a zároveň má měření co nejmenší dopad na spotřebu, by mohlo být předmětem dalšího zkoumání.

Dalším potenciálním vylepšením by mohla být úprava měření VOC indexu, který sleduje pouze změny oproti běžnému stavu ovzduší v místnosti. VOC index se sice kvůli zpracování dat VOC algoritmem měří pravidelně jednou za sekundu, ale data jsou ukládána pouze s intervalem hlavního měření. Pokud je tedy nastavený interval tohoto měření příliš vysoký, rychlé změny nebudou v uložených datech patrné. Do budoucna by tedy bylo lepší porovnávat u naměřených dat této veličiny spíše změny a pokud dojde k nějakému výkyvu, danou hodnotu uložit do databáze.

Z hlediska uživatelského rozhraní by mohlo být do budoucna přívětivější přidat jednotlivým uzlům navíc v nastavení nějaké aliasy, pomocí kterých by bylo možné poznat, o který uzel se jedná, aniž by si musel uživatel pamatovat daný identifikátor. Dále by šla přidat možnost registrace nových uživatelů a vytvořit obrazovku pro přidávání nových uzlů do systému.

Co se týče operací nad přijatými daty, bylo by ještě možné přidat několik funkcí. Například při aktualizaci stavu baterie některého ze zařízení by mohl být uživatel informován e-mailem, že baterii bude potřeba dobít, nebo že dobíjená baterie už naopak

není v kritickém stavu. Uživatel by pravděpodobně mohl být informován i jinak, např. notifikací z mobilní aplikace, avšak to by v tuto chvíli vyžadovalo více zkoumání.

Dále by mohla být vytvořena cloudová funkce, která by se automaticky spouštěla v určitou hodinu a kontrolovala, zda jednotlivá zařízení např. za posledních 12 hodin odeslala do databáze nějaká data. Pokud ne, byl by uživatel informován, že dané zařízení je nejspíše neaktivní a mělo by být zkontrolováno.

Podobných vylepšení a zdokonalení by se dala vymyslet celá řada. Důležité v danou chvíli však je, že systém byl navržen včetně jednotlivých bloků, které byly následně implementovány a momentálně jsou plně funkční.

Seznam literatury a informačních zdrojů

[1] MALÝ, Martin. *Protokol MQTT: komunikační standard pro IoT*. [online]. Poslední změna 29.6.2016. [Cit. 7.5.2021]. Dostupné z: https://www.root.cz/clanky/protokol-mqtt-komunikacni-standard-pro-iot/

[2] *MQTT – The Standard for IoT Messaging*. MQTT.org, 2020. [online]. [Cit. 8.5.2021] Dostupné z: https://mqtt.org/

[3] FRUHLINGER, Josh. *What is SSL, TLS? And how this encryption protocol works*. [online]. Poslední změna 4.12.2018. [Cit. 8.5.2021]. Dostupné z: https://www.csoonline.com/article/3246212/what-is-ssl-tls-and-how-this-encryption-protocol-works.html

[4] W3Schools. *Introduction to HTTP*. [online]. [Cit. 8.5.2021]. Dostupné z: https://www.w3schools.in/http-tutorial/intro/

[5] MDN Web Docs. *HTTP Request Methods*. [online]. Poslední změna 4.12.2020. [Cit. 9.5.2021]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods

[6] IANA. *Hypertext Transfer Protocol (HTTP) Status Code Registry*. [online]. Poslední změna 21.9.2018. [Cit. 9.5.2021]. Dostupné z: https://www.iana.org/assignments/http-status-codes/http-status-codes.xhtml

[7] Cloudflare. *What is HTTP?* [online]. [Cit. 9.5.2021]. Dostupné z: https://www.cloudflare.com/learning/ddos/glossary/hypertext-transfer-protocol-http/

[8] Český telekomunikační úřad. *Využívání vymezených rádiových kmitočtů*. [online]. [Cit. 10.5.2021]. Dostupné z: https://www.ctu.cz/vyuzivani-vymezenych-radiovych-kmitoctu

[9] Český telekomunikační úřad. *Využití rádiového spektra*. [online]. [Cit. 10.5.2021]. Dostupné z: http://spektrum.ctu.cz/

[10] SHAW, Keith. *802.11x: Wi-Fi standards and speeds explained*. [online]. Poslední změna 23.4.2020. [Cit. 10.5.2021]. Dostupné z: https://www.networkworld.com/article/3238664/80211x-wi-fi-standards-and-speeds-explained.html

[11] GEIER, Eric. *Co přináší tzv. Druhá vlna Wi-Fi?* [online]. Poslední změna 4.2.2017. [Cit. 10.5.2021]. Dostupné z: https://computerworld.cz/internet-a-komunikace/co-prinasi-tzv-druha-vlna-wi-fi-53621

[12] MEHL, Bernhard. *6 Communication Protocols Used by IoT*. [online]. Poslední změna 8.3.2021. [Cit. 10.5.2021]. Dostupné z: https://www.getkisi.com/blog/internet-of-things-communication-protocols

[13] GREGERSEN, Carsten. *A Complete Guide to IoT Protocols & Standards in 2021*. [online]. Poslední změna 18.12.2020. [Cit. 10.5.2021]. Dostupné z: https://www.nabto.com/guide-iot-protocols-standards/

[14] Zigbee Alliance. *Zigbee*. [online]. Poslední změna 2020. [Cit. 10.5.2021]. Dostupné z: https://zigbeealliance.org/solution/zigbee/

[15] Semtech. *What is LoRa?* [online]. Poslední změna 2021. [Cit. 10.5.2021]. Dostupné z: https://www.semtech.com/lora/what-is-lora

[16] Nanotron. *Chirp Spread Spectrum (CSS)*. [online]. [Cit. 10.5.2021]. Dostupné z: https://nanotron.com/EN/co_techn-css-php/

[17] PECH, Jiří. *IoT technologie: LoRa a LoRaWAN (3/5)*. [online]. Poslední změna 19.2.2019. [Cit. 11.5.2021]. Dostupné z: https://www.eman.cz/blog/iot-technologie-lora-a-lorawan-3-5/

[18] SigFox. *Sigfox Coverage*. [online]. [Cit. 11.5.2021]. Dostupné z: https://www.sigfox.com/en/coverage

[19] PECH, Jiří. *IoT technologie: SigFox (4/5)*. [online]. Poslední změna 9.5.2019. [Cit. 11.5.2021]. Dostupné z: https://www.eman.cz/blog/iot-technologie-sigfox-4-5/

[20] Sigfox. *Sigfox Technology*. [online]. [Cit. 11.5.2021]. Dostupné z: https://www.sigfox.com/en/what-sigfox/technology

[21] Datasheet HDC2080: *HDC2080 Low-Power Humidity and Temperature Digital Sensor*. [online]. Texas Instruments, 2019. [Cit. 11.5.2021]. Dostupné z: https://www.ti.com/lit/gpn/hdc2080

[22] Texas Instruments. *Humidity sensors*. [online]. Texas Instruments, 2018. [Cit. 11.5.2021]. Dostupné z: https://www.ti.com/lit/pdf/snaa318

[23] PECH, Jiří. *IoT technologie: Do budoucnosti s operátory (5/5)*. [online]. Poslední změna 4.7.2019. [Cit. 11.5.2021]. Dostupné z: https://www.eman.cz/blog/iot-technologie-budoucnost-5-5/

[24] MICHALEC, Libor. *Technologie NB-IoT a LTE Cat M1 jsou krok napřed*. [online]. Poslední změna 23.3.2018. [Cit. 11.5.2021]. Dostupné z: https://vyvoj.hw.cz/technologie-nb-iot-a-lte-cat-m1-jsou-krok-napred.html

[25] GSM Association. *Mobile IoT Deployment Map*. [online]. Poslední změna listopad 2020. [Cit. 11.5.2021]. Dostupné z: https://www.gsma.com/iot/deployment-map/

[26] EPA. *Volatile Organic Compounds' Impact on Indoor Air Quality*. [online]. Poslední změna 10.2.2021. [Cit. 12.5.2021]. Dostupné z: https://www.epa.gov/indoor-air-quality-iaq/volatile-organic-compounds-impact-indoor-air-quality

[27] Sensirion. *SGP40 – VOC Index for Experts*. [online]. Poslední změna srpen 2020. [Cit. 12.5.2021]. Dostupné z: https://www.sensirion.com/fileadmin/user_upload/customers/sensirion/Dokumente/9_Gas_Sensors/Datasheets/Sensirion_Gas_Sensors_Datasheet_GAS_AN_SGP40_VOC_Index_for_Experts_D1.pdf

[28] Sensirion. *Preliminary datasheet SGP40: Indoor Air Quality Sensor for VOC Measurements*. [online]. Poslední změna červenec 2020. [Cit. 12.5.2021]. Dostupné z: https://www.sensirion.com/fileadmin/user_upload/customers/sensirion/Dokumente/9_Gas_Sensors/Datasheets/Sensirion_Gas_Sensors_Datasheet_SGP40.pdf

[29] Sensirion. *Gas Sensors*. [online]. Poslední změna 2021. [Cit. 13.5.2021]. Dostupné z: https://www.sensirion.com/en/environmental-sensors/gas-sensors/

[30] Sensirion. *Datasheet Sensirion SCD30 Sensor Module: CO_2, humidity and temperature sensor*. [online]. Poslední změna květen 2020. [Cit. 12.5.2021]. Dostupné z: https://www.sensirion.com/fileadmin/user_upload/customers/sensirion/Dokumente/9.5_CO2/Sensirion_CO2_Sensors_SCD30_Datasheet.pdf

[31] Protronix s.r.o. *Kdy měřit VOC a kdy CO_2?* [online]. Poslední změna 23.6.2020. [Cit. 13.5.2021]. Dostupné z: https://vetrani.tzb-info.cz/vnitrni-prostredi/20843-kdy-merit-voc-a-kdy-co2

[32] SOS electronic s.r.o. *SCD30 je více než NDIR CO2 senzor*. [online]. Poslední změna 29.3.2018. [Cit. 13.5.2021]. Dostupné z: https://www.soselectronic.cz/articles/sensirion/scd30-je-vice-nez-ndir-co2-senzor-2152

[33] ĎAĎO, Stanislav a KREIDL, Marcel. *Senzory a měřící obvody*. 1. vyd. Praha: ČVUT, 1996. 315 s. ISBN 80-01-01500-9.

[34] Datasheet STC4054: *800mA Standalone linear Li-Ion battery charger with thermal*

regulation. STMicroelectronics, 2006. [online]. Poslední změna 4.9.2006. [Cit. 15.5.2021]. Dostupné z: https://www.st.com/resource/en/datasheet/stc4054.pdf

[35] Datasheet TPS63020-Q1: *TPS63020-Q1 High Efficiency Single Inductor Buck-Boost Converter With 4-A Switches*. Texas Instruments, 2015. [online]. Poslední změna leden 2016. [Cit. 15.5.2021]. Dostupné z: https://www.ti.com/lit/gpn/tps63020-q1

[36] Datasheet ESP32: *Datasheet ESP32-WROOM-32D & ESP21-WROOM-32U*. Espressif Systems, 2021. [online]. Poslední změna únor 2021. [Cit. 16.5.2021]. Dostupné z: https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32d_esp32-wroom-32u_datasheet_en.pdf

[37] Datasheet ESP32: *ESP32 Series Datasheet*. Espressif Systems, 2021. [online]. Poslední změna 19.3.2021. [Cit. 16.5.2021]. Dostupné z: https://espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf

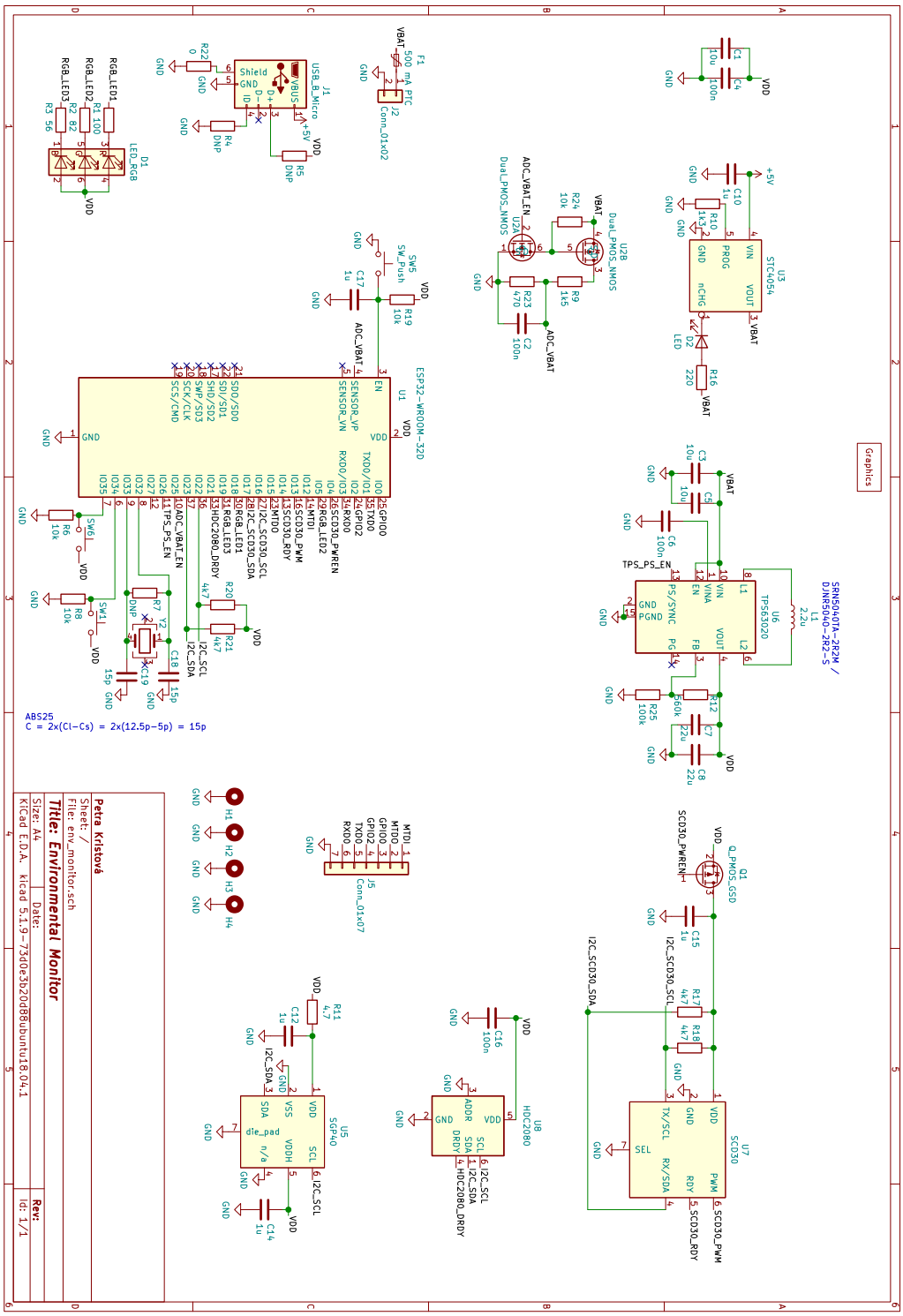
[38] Espressif Systems. *ESP32 Hardware Design Guidelines*. Espressif Systems, 2021. [online]. Poslední změna 30.4.2021. [Cit. 17.5.2021]. Dostupné z: https://www.espressif.com/sites/default/files/documentation/esp32_hardware_design_guidelines_en.pdf

[39] ESP-IDF Documentation: *API Reference – ESP-IDF Programming Guide v4.0.2 documentation*. Espressif Systems, 2016 – 2020. [online]. [Cit. 18.5.2021]. Dostupné z: https://docs.espressif.com/projects/esp-idf/en/v4.0.2/api-reference/index.html

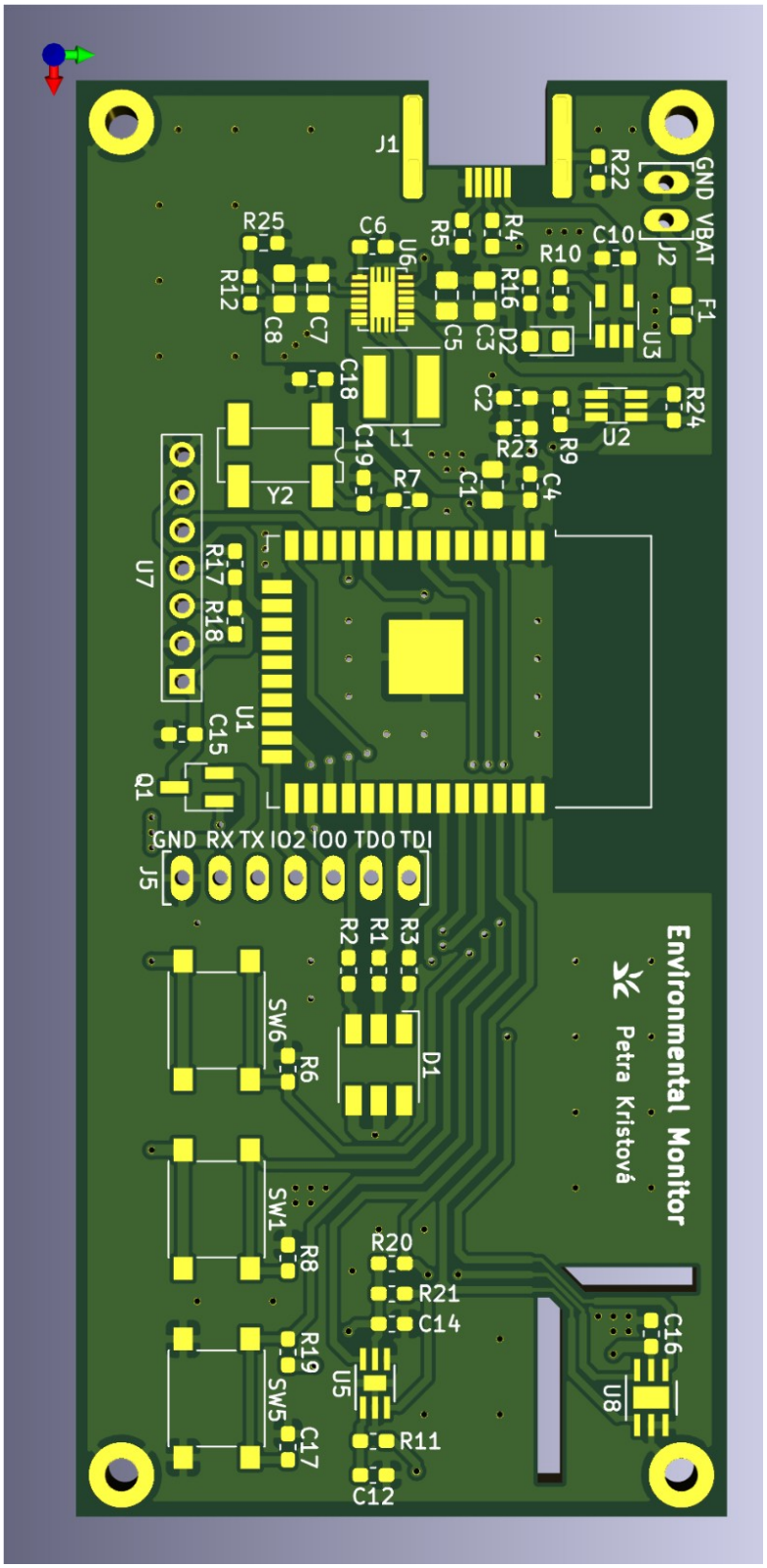
[40] Firebase Documentation: *Firebase Authentication*. Google, 2021. [online]. Poslední změna 11.5.2021. [Cit. 20.5.2021]. Dostupné z: https://firebase.google.com/docs/auth

Přílohy

Příloha A – Elektrické schéma měřicího uzlu



Příloha B – Vizualizace návrhu desky plošných spojů (vrstva top)



Příloha C – Vizualizace návrhu desky plošných spojů (vrstva bottom)