VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

BEZDRÁTOVÁ SENZOROVÁ SÍŤ

WIRELESS SENSOR NETWORK

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **MATÚŠ NOSKO**
AUTHOR

VEDOUCÍ PRÁCE — **prof. Dr. Ing. PAVEL ZEMČÍK**
SUPERVISOR

BRNO 2023

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Zadání bakalářské práce

148107

Ústav:	Ústav počítačové grafiky a multimédií (UPGM)
Student:	**Nosko Matúš**
Program:	Informační technologie
Specializace:	Informační technologie
Název:	**Bezdrátová senzorová síť**
Kategorie:	Vestavěné systémy
Akademický rok:	2022/23

Zadání:

1. Prostudujte existující řešení a dostupnou literaturu na téma bezdrátové senzorové sítě se zaměřením na "embedded" řešení senzorů a WiFi sítě.
2. Vytipujte vhodné funkce senzorové sítě a navrhněte/vyberte odpovídající architekturu sítě a implementační platofmru.
3. Rozeberte dosažitelné možnosti navržené senzorové sítě a možnosti jejího ovládání, případně i připojení do cloudu.
4. Implementujte senzorovou síť a demonstrujte její vlastnosti a možnosti na vhodném příkladě.
5. Ověřte a diskutujte výsledky práce, navrhněte vhodný postup testování a popište i možnosti dalšího pokračování práce.

Literatura:
Dle pokynů vedoucího práce.

Při obhajobě semestrální části projektu je požadováno:
Body 1-3 zadání.

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

Vedoucí práce:	**Zemčík Pavel, prof. Dr. Ing., dr. h. c.**
Vedoucí ústavu:	Černocký Jan, prof. Dr. Ing.
Datum zadání:	1.11.2022
Termín pro odevzdání:	10.5.2023
Datum schválení:	31.10.2022

Abstrakt

Práca bola zameraná na zabezpečenú senzorovú sieť založenú na mesh topológii, ktorá je postavená na zariadeniach od spoločnosti Espressif. Sústredí sa tiež na zabezpečenie kódu a prepojenie tejto siete s cloudom. Vytvorená mesh sieťje dynamická, regeneratívna a samo-organizujúca. Celá sieť je postavená na Wi–Fi, takže je jednoduché ju pripojiť do internetu, pretože jedinou podmienkou pre jej fungovanie je mať na mieste inštalácie Wi-Fi, takže nie je potrebné mať žiadne špeciálne zariadenie pre pripojenie siete k internetu – gateway-less. Aj keď jeden uzol zlyhá, sieť sa sama zorganizuje podľa novo vytvorených podmienok a funguje ďalej. Na zabezpečenie zariadení sa využívajú najmodernejšie metódy zabezpečenia, ako je bezpečné nahranie firmwaru a šifrovanie flash pamäte. Pre pripojenie mesh siete k serveru sa používa protokol MQTT. MQTT broker, databáza a API rozhranie sú na strane serveru dockerizované. Vďaka týmto docker kontajnerom je možne serverové časti jednoducho rozširovať. Taktiež hodnoty zo zariadení sú uchovávané v databáze a zobrazované na webovej stránke. Celé toto riešenie siete bolo implementované v prostredí ESP-IDF a otestované na zariadeniach ESP32–S3.

Abstract

The work was focused on creating a secure sensor network based on mesh topology, built on devices from Espressif. It also concentrates on code security and connecting this network to the cloud. The mesh network created is dynamic, regenerative, and self-organizing. The entire network is built on Wi-Fi, making it easy to connect to the internet, as the only condition for its operation is to have Wi-Fi at the installation site, eliminating the need for any special device to connect the network to the internet – it's gatewayless. Even if one node fails, the network self-organizes according to the newly created conditions and continues to function. The most modern security methods are used to secure the devices, such as secure boot and flash encryption. The MQTT protocol is used to connect the mesh network to the server. The MQTT broker, database, and API interface on the server side are dockerized. Thanks to these docker containers, it is easy to extend server parts. Also, values from devices are stored in a database and displayed on a website. The entire network solution was implemented in the ESP-IDF environment and tested on ESP32-S3 devices.

Kľúčové slová

mesh, sieť mesh, šifrovanie pamäte flash, bezpečné zavedenie firmwaru, asymetrická kryptografia, symetrická kryptografia, gateway-less, Docker, MQTT, Wi-Fi, mesh Wi-Fi

Keywords

mesh, mesh network, flash encryption, secure boot, asymmetric cryptography, symmetric cryptography, gateway-less, Docker, MQTT, Wi-Fi, mesh Wi-Fi

Citácia

NOSKO, Matúš. *Bezdrátová senzorová síť.* Brno, 2023. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce prof. Dr. Ing. Pavel Zemčík,

Bezdrátová senzorová síť

Prehlásenie

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením pána prof. Dr. Ing. Pavla Zemčíka. Uviedol som všetky literárne pramene, publikácie a ďalšie zdroje, z ktorých som čerpal.

.....
Matúš Nosko
15. mája 2023

Poďakovanie

Rád by som poďakoval svojmu vedúcemu práce, prof. Dr. Ing. Pavlovi Zemčíkovi, za jeho rady a odborné vedenie. Taktiež by som rád poďakoval svojmu bratovi, Ing. Svetozárovi Noskovi, za jeho rady a trpezlivosť. Ďalej by som chcel poďakovať mojej rodine a blízkym za ich podporu. A nakoniec, ďakujem aj svojim kolegom z práce za ich rady a podporu.

Obsah

1 Úvod **2**

2 Súčasný stav poznania v IoT **3**
2.1 História IoT 3
2.2 Aktuálny stav IoT 3
2.3 Technologické výzvy v IoT 4

3 Technológie využívané v IoT **7**
3.1 Komunikačné technológie v IoT 7
3.2 Vývojové dosky v IoT 12
3.3 Obecné technológie v zabezpečení 14
3.4 Zabezpečenie platformy 17
3.5 Zabezpečenie komunikácie v rámci IoT siete 18
3.6 Serverové technológie v kontexte IoT 19
3.7 Spotreba elektrickej energie 24
3.8 Budúcnosť IoT 25

4 Aktuálne dostupné riešenia **26**
4.1 Zhodnotenie aktuálnych riešení 27
4.2 Motivácia a vymedzenie cieľov práce 28

5 Návrh systému **29**
5.1 Návrh architektúry serveru a jeho komponentov 29
5.2 Návrh architektúry koncových zariadení 31

6 Implementácia ekosystému siete a testovanie **36**
6.1 Implementácia serverovej časti 37
6.2 Implementácia senzorovej siete 42
6.3 Impementácia kocových zariadení 46
6.4 Testovanie ekosystému bezdrôtovej siete a navrhované vylepšenia 50

7 Záver **53**

Literatúra **54**

A Obsah priloženého pamäťového média **56**

Kapitola 1

Úvod

Internet vecí (angl. Internet of Things, ďalej len IoT) sa stal trendom a súčasťou každodenného života. V dnešnej dobe sa IoT zariadenia nachádzajú v mnohých domácnostiach a tvoria komplexnejšie celky. V kontexte domácností ide o Smart home, v priemysle o Industry 4.0, v rámci miest o Smart City. S popularitou IoT rastie aj počet novo vznikajúcich projektov v tejto oblasti. Už nejde len o odborníkov z oblastí informačných technológii a veľké spoločnosti, ale aj o nadšencov so zámerom si zjednodušiť a uľahčiť každodenné činnosti. Na jednej strane je to pre IoT ako také veľmi prospešné, keďže vznikajú obrovské komunity, napr. Raspberry, Arduino a pod., kde ľudia zdieľajú získané skúsenosti a pomáhajú si v riešení problémov. Na druhej strane táto popularita so sebou nesie aj nežiadúce efekty. Často diskutovaným problémom v IoT je zabezpečenie. IoT systém môže spravovať veľké množstvo zariadení, od teplomerov cez ovládanie svetiel po ovládanie klimatizácie, vykurovania, kde je zabezpečenie kritické a veľmi dôležité. Prípadný útočník by mohol ovládať a prezerať aktuálne hodnoty (napr. by vedel vyčítať kedy je užívateľ doma), poprípade vypnúť kúrenie (kde hrozí poškodenie rozvodov mrazom) alebo zbytočne plytvať energiami (klimatizácia a kúrenie zapnuté súčasne). Ďalším negatívom je veľké množstvo aplikácií na správu zariadení, neexistujúci štandard, v podstate každá spoločnosť si vytvára celý vlastný ekosystém, a teda integrácia do už existujúcich systémov iných značiek je komplikovaná. Cieľom tejto bakalárskej práce je predstaviť a ukázať návrh architektúry celého ekosystému, od serveru až po koncové zariadenia, súčasne túto sieť implementovať a prepojiť so serverom.

Kapitola 2

Súčasný stav poznania v IoT

Táto časť popisuje aktuálny stav IoT, vrátane používaných technológií, populárnych platforiem a súhrnu očakávanej budúcnosti trhu. Poskytne tiež prehľad histórie a technológii, ktoré viedli k vzniku IoT.

2.1 História IoT

Prvé zariadenie ovládané cez internet bolo vyvinuté v roku 1990 vedcom Johnom Romkeym. Bol to hriankovač, ktorý sa dal cez internet zapnúť a vypnúť. V roku 1994 sa vedcovi Steveovi Mannovi podarilo prenášať obraz takmer v reálnom čase z jeho kamery Wearcam na internetovú stránku. Následne v roku 1997 Paul Saffo po prvýkrát popísal senzory a ich budúcnosť. V roku 1999 Kevin Ashton, výkonný riaditeľ Auto-ID Centra na MIT, zadefinoval pojem "Internet vecí". V tom istom roku bola vynájdené aj identifikovanie predmetov pomocou RFID. Následne LG oznámilo svoje plány na prvú inteligentnú chladničku. V roku 2003 bola identifikácia predmetov pomocou RFID využívaná aj v americkej armáde. Postupom času záujem o technológie IoT rástol, rovnako ako ponuka.[17]

2.2 Aktuálny stav IoT

Aktuálne sa nachádzame v dobe, v ktorej je IoT veľmi populárne a táto popularita rapídne rastie. Nie len z hľadiska technológii, ale aj počtom používateľov. IoT umožňuje ľuďom merať svet okolo seba, automatizovať monotónne činnosti. Ako príklad je možné uviesť inteligentné kvetináče, skleníky, polievanie trávnikov. Tieto zariadenia sa často integrujú do väčších systémov, ktoré dokážu poskytnúť detailnú analýzu okolitého prostredia. Ná úrovni domácností ide o Smart Home, na úrovni miest zase o Smart City, v priemysle sa hovorí o Priemysle 4.0. Existujú aj ďalšie odvetvia napr. zdravotníctvo, poľnohospodárstvo, autonómne vozidlá, pričom v každom sa výrobcovia sústredia na iné hlavné vlastnosti, typicky:

- Cena – je veľmi dôležitým faktorom pri výbere IoT zariadenia koncovým užívateľom. Nižšia cena umožňuje dostupnosť a atraktivitu pre väčší počet užívateľov, prípadne viacero zariadení na jedného, napr. teplomer a vlhkomer v každej miestnosti domácnosti. Pomer cena–kvalita prevedenia, odolnosť, výdrž a funkcionalita, kde je tento pomer často optimalizovaný s ohľadom na cenu, napr. konštrukcia zariadenia, limitovaný interval zberu dát. Drahšie zariadenia sú komplexnejšie systémy, ktoré spĺňajú často priemyselné normy, majú veľmi vysokú spoľahlivosť a dostupnosť.

- Odolnosť – odzrkadľuje podmienky, v ktorých je možné zariadenia prevádzkovať, napr. vnútorné alebo vonkajšie použitie, UV stabilita, vlhkosť, vodeodolnosť. Zväčša sa jedná o priamu úmeru, a teda čím má byť zariadenie odolnejšie, tým je vyššia cena.
- Spoľahlivosť – pravdepodobnosť správneho fungovania zariadenia počas určitej doby, počas ktorej bude poskytovať správny výstup.
- Dostupnosť – časový interval počas ktorého zariadenie funguje správne a reaguje na príkazy.
- Výdrž batérie – ak je zariadenie napájané batériou, je dôležité optimalizovať spotrebu energie, aby vydržalo čo najdlhšie. Na zaistenie tejto výdrže sa využívajú rôzne funkcie mikrokontrolérov, ako napríklad spánok, hlboký spánok alebo prerušenia.
- Zabezpečenie – zameranie sa na riadenie prístupov k dátam a správe samotného zariadenia. Taktiež zabezpečenie komunikácie zariadenia, keďže často je pripojené na internet.
- Rozšíriteľnosť – ekosystém rôznych senzorov a aktuátorov, ktoré sa dokážu pripojiť do jedného systému a následne sa dynamicky rozširovať o ďalšie zariadenia.

Dáta zo zariadení sa často následne zbierajú na server, kde sa spracujú, a už spracované sa využívajú na weboch alebo v aplikáciách. Všetky tieto faktory prispievajú k veľmi pozitívnemu rastu trhu IoT [3], čo nepochybne prináša nové príležitosti v tejto oblasti. Príkladom môžu byť inteligentné hodinky, ktoré patria do sektora zdravotnej starostlivosti a stávajú sa čoraz dostupnejšími a populárnejšími.

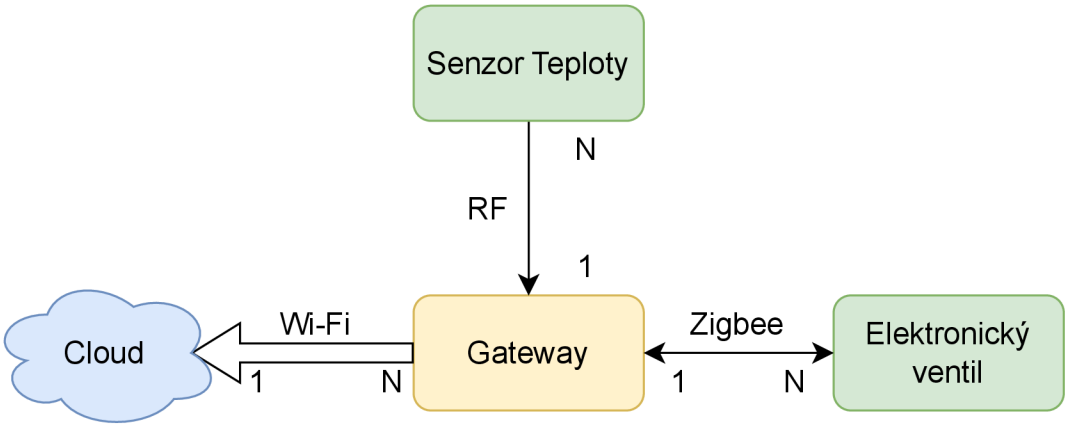
2.3 Technologické výzvy v IoT

Vytvorenie bezdrôtovej IoT siete je náročná úloha, ktorá prináša mnoho výziev. Tejto problematike sa venovali v článku [6], a tieto výzvy sa dajú zhrnúť následovne:

- Rôznorodosť – Zariadenia v sieti môžu patriť do iných skupín ako sú prepínače, snímače, aktuátory atď. Tieto zariadenia sú často postavené na rozdielnej elektronike a podporujú rôzne technológie na komunikáciu.
- Škálovateľnosť – Adresovanie, zber dát, správa a obsluha obrovského množstva zariadení.
- Komunikácia – IoT zariadenia používajú na komunikáciu rôzne technológie rozdelené na drôtové a bezdrôtové.
- Spotreba energie – Cielenie na optimálnu spotrebu, využívanie rôznych možností zariadenia na optimalizovanie spotreby.
- Ochrana súkromia – je dôležitá, pretože únik dát môže spôsobiť problémy. Útočník by mohol získať informácie o našej polohe, či je užívateľ doma alebo nie, a dokonca aj fotografie z bezpečnostných kamier. Zabezpečenie zariadení je teda veľmi dôležitá a náročná výzva.
- Sebaorganizácia – Zariadenia by sa mali samostatne organizovať podľa definovaných pravidiel systému, a to bez veľkého zásahu človeka.

- Interoperabilita – Jasne definované štandardizované spôsoby výmeny dát by mali existovať pre úspešnú spoluprácu rôznorodých zariadení.

Samozrejme, tu nie sú vymenované všetky výzvy, ale len tie najčastejšie, na základe ktorých je možné navrhnúť architektúru senzorovej siete.



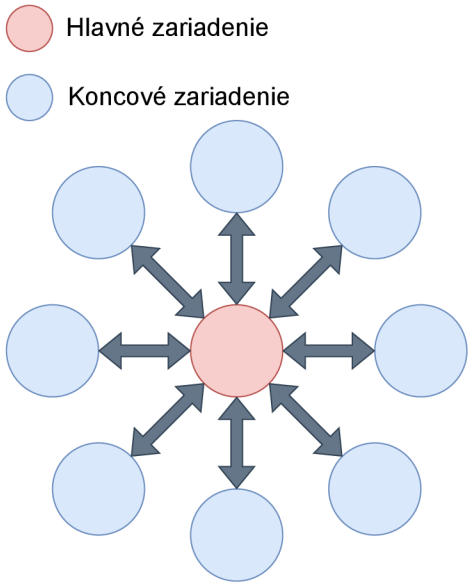
Obr. 2.1: Ukážka senzorovej siete s rôznymi technológiami

Na obrázku 2.1 je zobrazená rôznorodosť technológií, s ktorými je potrebné počítať. Taktiež je nutné počítať s potenciálne veľkým množstvom zariadení v sieti. Zber dát na gateway[1] je v tomto prípade sprostredkovaný energeticky úspornou komunikáciou, ale s nízkou prenosovou rýchlosťou. Zariadenia môžu v určitých intervaloch posielať svoj stav na gateway, ktorý ich následne pošle do cloudu pomocou Wi-Fi.

Popularita IoT je spôsobená aj veľkou variabilitou vo výbere technológií. Základným prvkom IoT je hardvér, ktorý musí spĺňať požiadavky týkajúce sa spotreby energie, výpočtového výkonu, veľkosti, prevádzkových podmienok, počtu a rozmanitosti periférií. Komunikáciu medzi jednotlivými zariadeniami zabezpečuje väčšinou bezdrôtová komunikácia, ako napríklad Wi-Fi, LoRaWAN (Long-Range Wide Area Networks), BLE (Bluetooth Low-Energy), Zigbee alebo Ethernet. Zozbierané dáta sa ukladajú na server, kde sú spracovávané v rámci konceptu nazvaného Cloud computing. Jedným z populárnych spôsobov pripojenia zariadení na server je protokol MQTT. Vďaka rastúcej výpočtovej sile mikrokontrolérov sa časť spracovania zozbieraných dát môže vykonávať priamo na samotnom zariadení, čo sa nazýva Edge Computing. Zhlukovanie jednotlivých zariadení do väčších sietí sa môže realizovať typicky v dvoch kategóriách: centralizované, decentralizované.

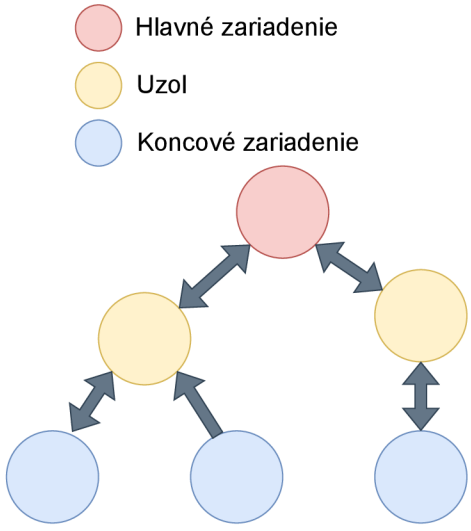
[1] gateway – brána medzi IoT sieťou a internetom

Centralizovaná sieť – v sieti sú zariadenia pripojené len na jedno hlavné riadiace zariadenie, ktoré riadi a rozosiela dáta v celej sieti.



Obr. 2.2: Ukážka cenrealizovanej siete

Decentralizovaná sieť – v sieti nie sú zariadenia pripojené na jedno hlavné, ale pripájajú sa na uzly v sieti.



Obr. 2.3: Ukážka decentralizovanej siete

Existuje aj distribuovaný typ siete. V tomto type sú si uzly vzájomne rovnocenné a neexistuje žiadny centrálny bod. Je kladený dôraz na vysokú dostupnosť a rovnomerné rozloženie záťaže medzi jednotlivými uzlami.

Kapitola 3

Technológie využívané v IoT

3.1 Komunikačné technológie v IoT

Komunikačné technológie sú obecne dôležité pre prenos dát z koncových zariadení smerom do internetu, ale aj medzi zariadeniami v rámci jednej siete. Ide aj o prepojenie zariadení medzi sebou, keďže si vymieňajú dáta a vykonávajú zadané úlohy bez účasti človeka. Tieto zariadenia sú často vybavené rôznou elektronikou, ktorá pomáha pri sprostredkovaní prepojenia zariadení a pripojenia celej IoT siete na internet:

Wi-Fi je postavená na štandardoch IEEE 802.11 a vyvíjaná združením Wi-Fi Alliance. Je jednou z najviac rozšírených technológii na svete. Vďaka spätnej kompatibilite s už existujúcimi sieťami Wi-Fi hrá dôležitú úlohu v IoT. Typicky bezdrôtové IoT siete v nejakom bode používajú Wi-Fi na pripojenie do internetu, napr. na obrázku 2.1. Taktiež je možné pomocou Wi-Fi prepojiť jednotlivé zariadenia medzi sebou a vytvoriť komplexnejší celok.

Bluetooth je taktiež veľmi rozšírená technológia, ktorá podporuje najmä komunikáciu medzi zariadeniami typu bod–bod. Používa sa najmä na bezdrôtové streamovanie dát, napr. prenos hudby do slúchadiel, prepojenie mobilného telefónu s autom. Pre IoT zariadenia a zariadenia s nízkou spotrebou energie bola vyvinutá technológia Bluetooth Low Energy (BLE), ktorá využíva rôzne spôsoby spánku na dosiahnutie čo najnižšej spotreby energie.

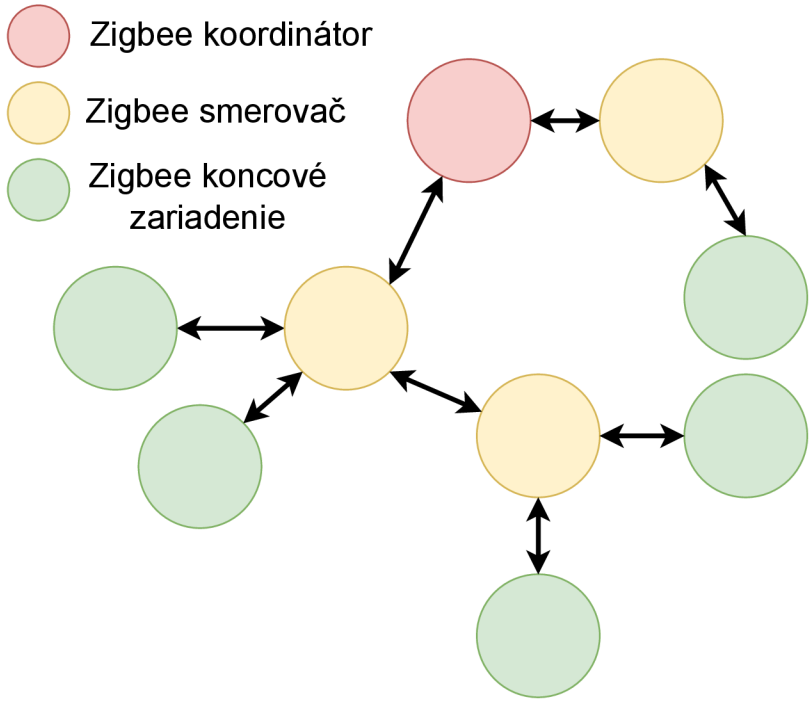
Zigbee je podrobne vysvetlená v [12]. Je to technológia postavená na štandardoch 802.15.4. Je vyvíjaná a udržiavaná skupinou firiem Zigbee Alliance. Sprostredkováva komunikáciu medzi energeticky nenáročnými zariadeniami. Podporuje obrovské mesh siete, kde sa môže nachádzať až 65000 zariadení pracujúcich lokálne v pásmach <1GHz a 2.4GHz. V takejto Zigbee sieti sa vyskytujú tri typy zariadení:

Koordinátor – ide o riadiacu jednotku siete (tiež označovaná ako brána, most). Jeho úlohou je vytvoriť sieť, vytvoriť a udržiavať zabezpečenie, pridávať nové zariadenia a spravovať celú sieť. Je nutné aby koordinátor bol trvale napájaný a vždy bol len jeden.

Smerovač – je zariadenie taktiež trvalo napájané. Typicky ide o žiarovky, vypínače, zástrčky, zásuvky, teda také zariadenia, ktoré sú vždy pripojené do elektrickej siete. Jeho úlohou je vytvárať cestu od jedného zariadenia k druhému, a teda smerovať správy v rámci siete.

Koncové zariadenia – sú základným prvkom siete a môžu len prijímať alebo odosielať dáta. Nemôžu smerovať dáta a teda môžu komunikovať len so smerovačmi alebo priamo s koordinátorom. Koncové zariadenia sú typicky napájané z batérie a ide o rôzne snímače a senzory.

Ukážka takejto siete môže vyzerať ako na obrázku 3.1.



Obr. 3.1: Ukážka Zigbee senzorovej siete

Thread [1] je protokol určený pre bezdrôtové mesh siete s nízkou spotrebou energie a nízkou latenciou. Tak ako Zigbee je tiež postavený na štandarde IEEE 802.15.4. Technologické výzvy rieši použitím otvorených a overených štandardov. Thread využíva na pripojenie zariadení do internetu internetový protokol IPv6. Celý internet funguje na protokole IP cez počítače, smerovače, až po spojenia po celom svete. IP je teda spôsob ako zariadenia komunikujú medzi sebou bez ohľadu na to, aké technológie pripojenia používajú (Ethernet, Wi-Fi, LTE atď.). To má výhodu v možnosti integrovať sieť Thread do väčších sietí IP, bez potreby brán alebo prekladačov. V sieti je viacero typov zariadení s rôznymi funkciami[2]:

Hraničný smerovač – je zariadenie, ktoré môže smerovať správy medzi sieťou Thread a inou nie–Thread sieťou, napr. Wi-Fi alebo Bluetooth. V jednej sieti môže byť týchto smerovačov viac, čo je výhodou pri dostupnosti siete. Taktiež sa starajú o správu siete, pripájanie nových členov siete a zabezpečenie.

[1] **What is Thread** *Thread Group* [online]. [cit. 2023-03-12] Prevzaté z https://www.threadgroup.org/What-is-Thread/Thread-Benefits

[2] **Roles and Types** *OpenThread* [online]. [cit. 2023-03-12] Prevzaté z https://openthread.io/guides/thread-primer/node-roles-and-types

Koncové zariadenie spôsobilé byť smerovač – je zariadenie, ktoré sa v prípade potreby môže stať smerovačom, inak sa chová ako koncové zariadenie. Voľbu či je nutný nový smerovač si sieť obstaráva sama bez zásahu človeka.

Minimálne koncové zariadenia – vysielač majú stále zapnutý a nemusia si pýtať spravý od svojho nadradeného smerovača, tzv. polling.

Sleepy End Device (zariadenie s možnosťou spánku) – väčšinu času strávia v režime spánku, musia si správy pýtať od nadradeného smerovača. Vďaka takejto úspore energie môžu pracovať roky napájané z batérie.

Štruktúra siete Thread je teda veľmi podobná Zigbee, taktiež sú oba protokoly postavené na rovnakom štandarde IEEE 802.15.4 a dokonca existuje možnosť prepojiť spolu tieto dve siete.

Matter [3] – (skôr známy ako Project CHIP) je open–source štandard pripojenia IoT zariadení v rámci Smart Home. Cieľom tohto štandardu je prepojiť rôzne zariadenia rôznych výrobcov v rámci domácnosti. Ide o pomerne nový štandard vytvorený v roku 2019. Súčasne s rôznymi výrobcami zariadenia nemusia podporovať ani rovnakú komunikačnú technológiu, pričom Matter združuje tieto štandardy: Wi-Fi, Z-Wave, Thread, Wi-Fi. Dôveryhodnosť/potenciál tohto štandardu dokladajú aj zakladatelia, keďže sa jedná aj o popredných výrobcov na trhu IoT ako Amazon, Google, Apple, Samsung Smart Things a Zigbee Aliance, následne sa k nim pripojili ďalší výrobcovia napr. Ikea, Huawei. Ide teda o štandard, ktorý sa snaží spojiť viacero komunikačných technológii do jedného celku, v ktorom sa rôzne zariadenia vedia dorozumieť.

Je možné si všimnúť, že okrem využitia Wi-Fi je potrebné vždy danú sieť pripojiť do internetu pomocou hraničného smerovača alebo brány. Typicky tieto hraničné zariadenia majú na sebe dva rôzne vysielače, ako na obrázku 2.1, konkrétne toto zariadenie by na sebe malo vysielač Wi-Fi spolu s vysielačom Zigbee a RF. Samozrejme v praxi sa tieto technológie rôzne kombinujú aby daná sieť čo najviac vyhovovala danej lokalite. Ako príklad môže byť bytovka s mnohými Wi-Fi sieťami, v takom prípad dáva väčší zmysel použiť iné pásmo, keďže pásmo Wi-Fi môže byť zahltené ostatnými Wi-Fi sieťami. Zhrnutie a porovnanie jednotlivých technológii je môžné vidieť v tabuľke 3.1.

[3] **Matter** *Connectivity Standards Alliance* [online]. [cit. 2023-04-22] Dostupné z https://csa-iot.org/all-solutions/matter/

Charakteristika	**Zigbee**	**Wi-Fi**	**Thread**	**Bluetooth LE**
IEEE Štandard	802.15.4	802.11	802.15.4	802.15.1
Frekvenčné pásmo	2.4 GHz	2.4 GHz, 5GHz	2.4 GHz	2.4 GHz
Pracovná vzdialenosť	50 – 100m	150m	30m	10m
Spotreba el. prúdu v špičkách	30 mA	116 mA	12.3 mA	12.5mA
Spotreba energie na bit	185.9 $\mu W/bit$	0.00525 $\mu W/bit$	11.7 $\mu W/bit$	0.153 $\mu W/bit$
Dátový prenos	250 Kbps	1 Gbps	250 kbps	1 Mbps
Topológia	Hviezda, Zhľuk, Mesh	Hviezda, Mesh	Mesh	Star-bus
Počet uzlov v sieti	65000	250/prístupový bod	300	jeden k viacerým (angl. one-to-many)

Tabuľka 3.1: Porovnanie komunikačných protokolov

MQTT je populárny protokol na zasielanie správ v IoT. Protokol definuje ako môžu zariadenia publikovať (angl. publish) a odoberať (angl. subscribe) správy. Používa sa na distribúciu a výmenu správ medzi zariadeniami IoT, PLC a IIoT atď. Je založený na princípe posielania správ medzi klientmi a brokerom (preložené sprostredkovateľ), ktorý sa stará o doručovanie a posielanie správ.

Broker je typicky umiestnený na servery, kde pracuje neustále. Najznámejšie implementácie brokera sú napr. Mosquitto, EMQX, RabbitMQ a ďalšie. Ide o open–source projekty, avšak existujú aj platené projekty, narp. HiveMQ, EMQX Enterprise. Typicky broker poskytuje aj administratívnu stránku s aktuálnym prehľadom, napr. počty prichádzajúcich/odchádzajúcich správ, počet pripojených klientov. Správy sú triedené do tém (angl. topics), ktoré zariadenia môžu buď publikovať alebo odoberať. **Témy** sú usporiadané reťazce slov, oddelené lomkami, kde tieto reťazce typicky vystihujú a popisujú svoju množinu. Žiadne pravidlá okrem syntaktických v podstate neexistujú, takže je na užívateľovi, aby si navrhol ako tieto témy budú vyzerať. Môže to byť napríklad takto:

```
/kuchyňa-0/teplota/okno
```

Táto téma napríklad odosiela teplotu. Príklad s kuchyňou je členený do nasledujúcich celkov:

```
kuchyňa-0 + teplota + okno
```

Publikovanie dát zariadením znamená odosielanie dát na brokera, ktorý ich nasmeruje na adresáta/-ov. Na druhú stranu, ak zariadenie odoberá nejakú tému a broker prijme správu adresovanú na takúto tému, tak broker odošle túto správu všetkým odberateľom danej témy. Zariadenie môže publikovať a aj odoberať súčasne. MQTT poskytuje taktiež možnosť prijímať a odosielať správy v kontexte rôznych úrovní v téme. Existujú na to špeciálne znaky, nasledujúci príklad predpokladá umiestnenie viacerých senzorov teploty, vlhkosti a snímač otvorenia okna v kuchyni:

\+ – nahradzuje jednu úroveň, takže napríklad:

```
/kuchyňa-0/teplota/+
```

predstavuje všetky senzory teploty v kuchyni napr. teplota miestnosti, teplota pri oknách atď. Analogicky to funguje vo všetkých úrovniach témy, napríklad:

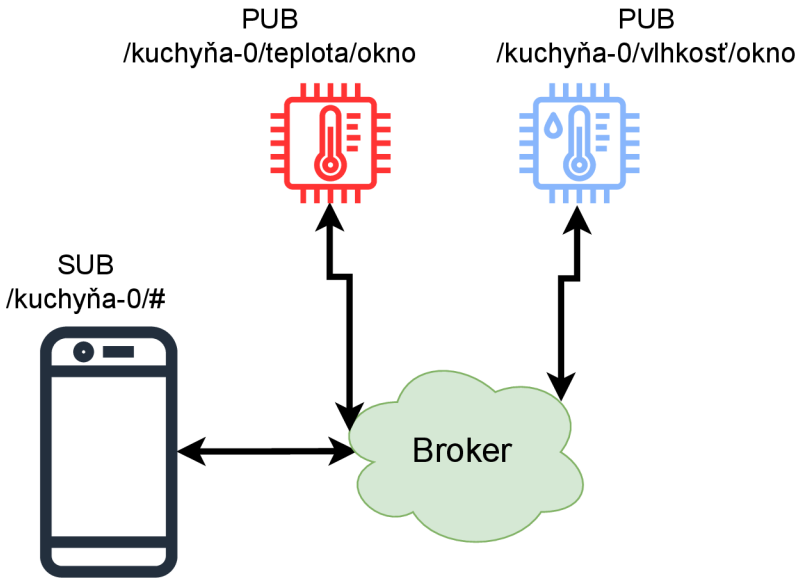
```
/kuchyňa-0/+/okno
```

predstavuje všetky typy senzorov na okne.

\# – je viacúrovňové vnorenie a teda je možné prijať alebo odoslať správu na všetky témy nachádzajúce sa za #, napríklad:

```
/kuchyňa-0/#
```

predstavuje všetky senzory v kuchyni. V prípade potreby odoberať všetky témy, ktoré prijíma broker, by táto téma na odoberanie vyzerala /#, samozrejme funguje aj na odoslanie.
Príklad takejto senzorovej siete s rôznymi témami je zobrazená na nasledujúcom obrázku:



Obr. 3.2: Ukážka odoberania a publikovania správ pomocou MQTT

Pripojenie klienta k brokeru prebieha pomocou 4 protokolov a to TCP, TLS, WebSockets a WebSockets Secure. Tieto protokoly zabezpečujú komunikáciu medzi zariadeniami a brokerom, pričom na zabezpečenie tejto komunikácie sa typicky využíva asymetrická a symetrická kryptografia.

Podstata MQTT protokolu zaručuje rôzne garancie doručenia správy pomocou QoS (Quality of Service) a príznaku retain.

QoS – vytvára tri úrovne a to 0, 1 a 2, pričom každá úroveň garantuje niečo iné:

- **0** tzv. najviac raz (angl. at most once) klient publikuje správu a o nič iné sa nestará, tak isto odošle správu s `QoS = 0` broker odberateľom danej témy.
- **1** tzv. aspoň raz (angl. at least once) klient publikuje správu a čaká na potvrdenie od brokera o prijatí správy, ak príde potvrdenie, je výmena ukončená.
- **2** presne raz (angl. exactly once) klient odošle správu a čaká na potvrdenie, ak príde potvrdenie odosiela správu o uvoľnení a čaká na potvrdenie od brokera, ak príde je výmena uzavretá.

Príznak retain – (preložené zachovať) ak je správa odoslaná s týmto príznakom, uchová sa na strane brokera a aj napriek rozoslaniu všetkým odberateľom sa nezahadzuje, ale zachová sa a rozosiela sa novým odberateľom danej témy. Je nutné poznamenať, že sa uchováva vždy posledná takto prijatá správa, to znamená prepisovanie starej správy s retain príznakom novou, v rámci jednej témy.

MQTT je teda silný nástroj na komunikáciu v IoT, umožňujúci jednoduchú správu zariadení, poskytujúci kontrolu nad sieťou vďaka brokeru, zaručenie doručenia správ vďaka `QoS`. Typicky sa využíva v kombinácii s komunikačnými technológiami vysvetlenými vyššie.

3.2 Vývojové dosky v IoT

Populárni výrobcovia sú napr. Raspberry, Espressif, Arduino, STMicroelectronics a nespočetne veľa ďalších. Každý výrobca má rôzne modely, určené na rôzne prípady použitia. Niektoré ľahko dostupné vytipované modely, vhodné pre senzorovú sieť sú:

Raspberry Pi Pico je malé zariadenie od spoločnosti Raspberry určené pre IoT, s nasledovnými technickými špecifikáciami:

- dvojjadrový procesor Arm Cortex-M0+
- 264kB Statickej operačnej pamäte
- 2MB QSPI pamäte
- GPIO piny, UART, SPI, I^2C, PWM
- Teplotný senzor priamo na doske
- Rôzne režimy spánku

Podpora programovacích jazykov C a MicroPython. Základná verzia nedisponuje žiadnou technológiou na bezdrôtové pripojenie (verzia Pi Pico W a WH disponuje 2.4GHz Wi-Fi). Nevýhodou je neexistencia možnosti zabezpečenia kódu, v podstate každý môže zo zariadenia vyčítať kód.

Raspberry Pi 4 je veľmi populárne zariadenie vďaka obrovskému výkonu, operačnému systému Linux, konektivite a pod. V IoT sa veľmi často využíva ako hlavný bod v centralizovaných sieťach (pojem vysvetlený v sekcii 2.2), prípadne môže mať aj rolu priamo serveru pre zberanie, ukladanie a spracovanie dát lokálne.

Espressif ESP32 (S3) je výkonné zariadenie s technológiou Wi-Fi a Bluetooth priamo na čipe a s konfiguráciou:

- 32bitový jedno alebo dvojjadrový mikroprocesor Xtensa LX7

- 512KB Statickej operačnej pamäte
- 384KB Pamäte len na čítanie
- Najčastejšie 4 – 16 MB Flash pamäte
- GPIO piny, UART, SPI, I^2C, I^2S, PWM, Hallov senzor a ďalšie
- Teplotný senzor priamo na doske
- Rôzne režimy spánku
- Bezpečný boot, šifrovanie pamäte Flash, hardvérová akcelerácia kryptografie napr. AES, Hash (SHA-2) a ďalšie

Výhodou je WiFi a Bluetooth priamo na čipe, takže netreba žiadne ďalšie komponenty pre komunikáciu bezdrôtovo so zariadením. Zariadenie taktiež disponuje funkcionalitou vzdialenej aktualizácie OTA – Over The Air update. Taktiež existuje možnosť zabezpečiť kód, takže zariadenie má lepšie predpoklady použitia v komerčnom sektore, či už pomocou zašifrovania pamäte flash (angl. flash encryption)[4] alebo bezpečného zavedenia (angl. secure boot)[5].

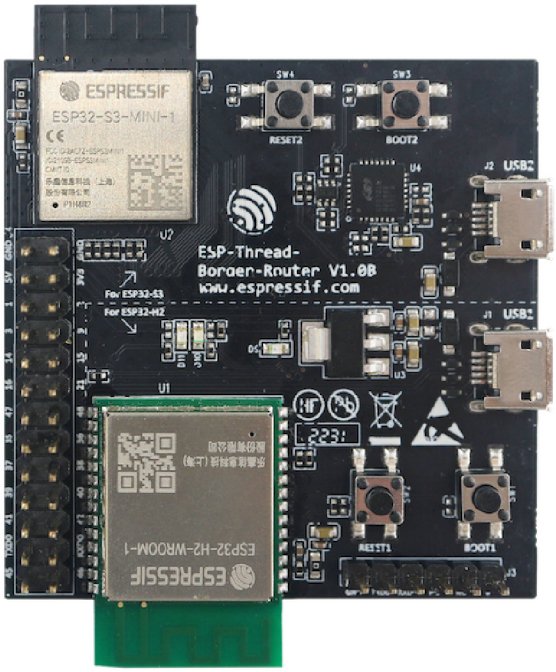
Arduino UNO (REV3) je veľmi populárna vývojová doska. Jej výhodou je rýchly vývoj vďaka Arduino IDE a komunite Arduino. Často sa používa v zariadeniach ako sú RC autá a RC lode. Ďalšou výhodou je napájanie, keďže GPIO piny pracujú na 5V s 20mA, taktiež vstupné napätie je v rozmedzí 7 – 12V. Nevýhodou môže byť veľkosť a absencia Wi-Fi alebo iných bezdrôtových komunikačných technológii, cena, obmedzená pamäť. Základná konfigurácia vyzerá následovne:

- Procesor 8–bitový jedno jadrový ATMega328P
- 32KB ISP Flash
- 1KB Pamäte len na čítanie
- 2KB Statickej operačnej pamäte
- GPIO piny, UART, PWM, I2C, SPI
- Napájanie 7-12V, I/O napätie 5v 20mA

Vývojové kity určené na vývoj technológii ako napr. Zigbee a Thread, sú zaujímavé v tom, že musia typicky zvládať spracovávať jednu z vymenovaných technológii, v kombinácii s Wi-Fi. Napríklad od spoločnosti Espressif vývojový kit určený na vývoj technológie Thread, zariadenia spracováva Thread v kombinácii s Wi-Fi alebo BLE.

[4] **Flash Encryption** *Esprressif* [online]. [cit. 2023-03-14] Prevzaté z https://docs.espressif.com/projects/esp-idf/en/latest/esp32s3/security/flash-encryption.html

[5] **Secure boot** *Esprressif* [online]. [cit. 2023-03-14] Prevzaté z https://docs.espressif.com/projects/esp-idf/en/latest/esp32s3/security/secure-boot-v2.html



Obr. 3.3: Vývojový kit určený na vývoj Thread siete[6]

Na obrázku 3.3 je vidieť vývojový kit od spoločnosti Espressif, ktorý je určený na vývoj Thread siete. Sú to v podstate dva čipy ESP32, ktoré podporujú rôzne komunikačné technológie – Wi-Fi/BLE a Thread. Tieto čipy sú medzi sebou prepojené pomocou UART alebo SPI rozhrania. V podstate na podobnom princípe typicky fungujú IoT brány, ktoré sú sprostredkovateľmi pripojenia do internetu.

3.3 Obecné technológie v zabezpečení

V IoT sieti sa na rôznych miestach vyžaduje zabezpečenie napr. komunikácia, ochrana kódu uloženého v zariadeniach, a ide o veľmi rozsiahlu a komplexnú oblasť. V IoT sa najčastejšie využívajú klasické kryptografické postupy (samozrejme prispôsobené k potrebám IoT zariadení):

Symetrická kryptografia funguje na princípe tajného kľúča, ktorý majú obe strany a je rovnaký. Pri komunikácii si musia účastníci vymeniť tento kľúč. Je vhodnejšia pre šifrovanie objemnejších dát, keďže je typicky rýchlejšia ako asymetrická kryptografia, takéto porovnanie sa vykonalo v článku [15]. Nepoužíva sa len na šifrovanie dát pri komunikácii, ale napríklad aj pri šifrovaní pamäte flash.

Vlastnosti – algoritmy majú najmä menšie nároky na výpočtový výkon. Veľkosť kľúčov je pomerne malá napr. algoritmus AES-256 kt. používa aj ESP32–S3, používa kľúč o veľkosti 512 bitov (64 bytov). Tento kľúč je teda jednoduchšie uložiť na zariadení s obmedzenou veľkosťou pamäte. Typickou vlastnosťou je

[6] **Introduction to ESP Thread Border Router** *Espressif* [online]. [cit. 2023-04-22] Dostupné z https://docs.espressif.com/projects/esp-thread-br/en/latest/esp32/br_introduction.html

nutnosť mať pred–zdielaný tajný kľúč, aby bola na každom konci komunikácie jedna kópia. Taktiež je náročné kľúče spravovať, keďže platí:

$$Keys = n * (n - 1)/2$$

kde n je počet účastníkov komunikácie a teda, napríklad pre 10 ľudí musíme spravovať a distribuovať 45 kľúčov, zjednodušená ukážka je zobrazená na obrázku 3.4.

Asymetrická kryptografia je založená na jednosmerných funkciách. Ich základnou vlastnosťou je jednoduchosť výpočtu výstupu zo vstupu, ale je veľmi obtiažne z výstupu vypočítať vstup. Príkladom takejto funkcie môže byť násobenie, kde je jednoduché vynásobiť dve čísla, ale rozklad súčinu na súčinitele je obtiažny. Je vhodná pre veľké prostredia s mnoho účastníkmi, kde si dáta vymieňajú často iní účastníci komunikácie. Každý účastník má dva svoje kľúče, jeden verejný a druhý privátny. Verejný a privátny kľúč sú v matematickom vzťahu a takéto kľúče sa nazývajú kľúčový pár. Privátny kľúč je udržiavaný v absolútnom utajení, narozdiel od verejného, ktorý je k dispozícii všetkým účastníkom komunikácie. Pre zašifrovania sú potrebné oba kľúče a prebieha to nasledovne:

Šifrovanie pomocou verejného kľúča – zašifrované dáta sú dešifrované len pomocou privátneho kľúča, s ktorým je verejný kľúč v páre.

Šifrovanie pomocou privátneho kľúča – šifrovanie pomocou privátneho kľúča vytvára digitálny podpis, čo zaručuje pôvod správy. Dešifrovanie tejto správy potom prebieha pomocou verejného kľúča, ktorý je súčasťou kľúčového páru spolu s privátnym kľúčom. Týmto spôsobom je zabezpečená aj autentifikácia správy, čo znamená, že je možné overiť jej pravosť.

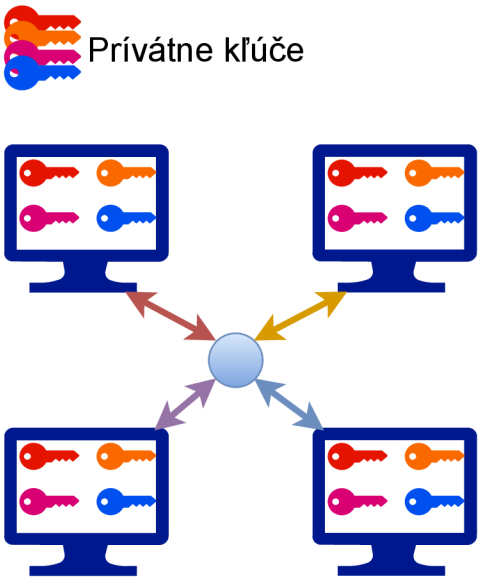
Ak dôjde k úniku privátneho kľúča, je nutné vygenerovať nový pár kľúčov.

Vlastnosti – vďaka nemožnosti odvodenia privátneho kľúča z verejného, môže byť verejný kľúč dostupný všetkým, takže rieši problém distribúcie kľúčov a správa kľúčov je jednoduchšia. Počet kľúčov potrebných pre zabezpečenú komunikáciu je definovaný ako:

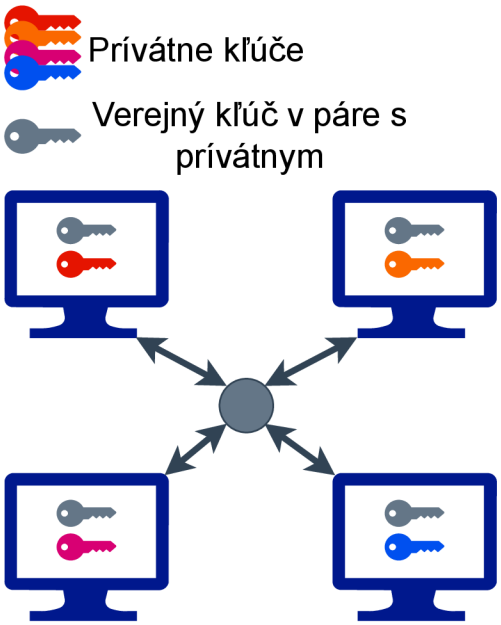
$$Keys = 2n$$

Napríklad pre 10 ľudí potrebujeme spracovať 20 kľúčov, resp. 10 kľúčových párov. Zjednodušená ukážka takejto komunikácie je zobrazená na obrázku 3.5. Digitálny podpis garantuje autenticitu odosielateľa, nepopierateľnosť odoslanej správy, zaručuje aj integritu správy, čo znamená, že správa nebola zmenená počas prenosu. V porovnaní so symetrickou kryptografiou má pomalší proces šifrovania, takéto porovnanie sa vykonalo v článku [15]. Autenticita verejného kľúča sa nedá overiť, takže účastníci si musia overiť, či verejné kľúče skutočne patria druhej strane. Veľkosť kľúčov môže byť napr. 2048–bitov, 4096–bitov atď., čo môže byť v oblasti IoT, kde majú zariadenia obmedzenú pamäť limitujúce.

V praxi sa často asymetrická kryptografia používa na výmenu kľúčov symetrickej kryptografie, ktorá následne vykonáva šifrovanie dát, keďže je rýchlejšia a výkonovo úspornejšia.

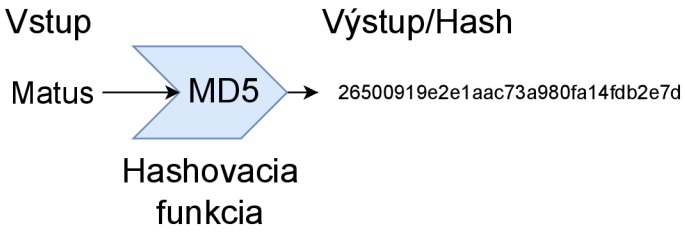


Obr. 3.4: Ukážka šifrovanej komunikácie pomocou symetrickej kryptografie



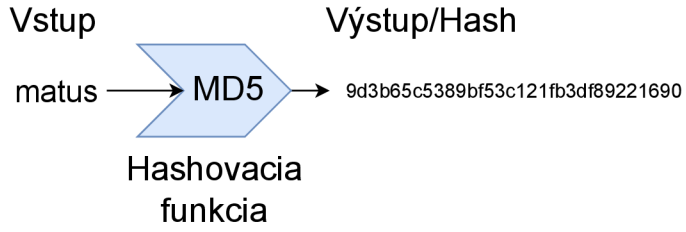
Obr. 3.5: Ukážka šifrovanej komunikácie pomocou asymetrickej kryptografie

Hashovanie je proces transformovania vstupných dát na novú hodnotu alebo kľúč so statickou dĺžkou. Dáta sú vstupom hashovacej funkcie a výstupom je hash. Tento proces je zvyčajne jednosmerný, z hashu nevieme získať vstup. Príkladom hashovacích funkcii sú napríklad MD5, SHA-2, SHA-1.



Obr. 3.6: Ukážka hashovania slova

Veľmi často sa hashovanie používa pri ukladaní hesiel do databázy, takže ak by unikli dáta, tak by sa útočník dostal len k hashom hesiel a nie čitateľným heslám. Príklad hashovania slova je zobrazený na obrázku 3.6, kde je ukázaný hash z hesla Matus. Aby hashovacia funkcia bola považovaná za bezpečnú, musí spĺňať tzv. lavínový efekt, to znamená, že ak zmeníme čo i len jeden bit vstupnej správy, tak výsledný hash bude drasticky odlišný, ako na obrázku 3.7.



Obr. 3.7: Ukážka lavínového efektu hashovania slova

Taktiež sa používa pri overení integrity prenesených dát, napr. súboru. Vypočíta sa hash pred odoslaním a po prijatí a následne sa porovnajú, ak sú hashe rovnaké tak bol súbor prenesený úspešne, ak sa nerovnajú, došlo ku chybe. Dobrý hashovací algoritmus by nemal vytvárať pre dva odlišné vstupy rovnaké výstupy. V prípade ak sa to stane, ide o kolíziu hashu. Dobrý algoritmus by mal mať túto šancu na kolíziu veľmi malú.

V praxi sa používajú všetky tri vymenované možnosti, či už na zabezpečené spojenie, šifrovanie dát alebo kontrolu integrity dát.

3.4 Zabezpečenie platformy

Zabezpečenie zariadenia je veľmi dôležitá téma v rámci IoT a aj jedna z najväčších výziev IoT. Pri zabezpečení môže vývojár spraviť všetko správne, ale dané riešenie je len tak bezpečné, ako najslabší článok [1]. Ide najmä o zabránenie útočníkovi v neoprávnenom manipulovaní so zariadením. Taktiež zabrániť možnosť vyčítania kódu zo zariadenia, keďže môže obsahovať citlivé užívateľské dáta, know–how. Známymi spôsobmi ako docieliť takéto zabezpečenie zariadenia sú:

Secure boot (preložené bezpečné zavedenie) ide o záruku toho, že len dôveryhodný software bude spustený na zariadení. Všeobecne ide o využitie asymetrickej kryptografie nasledujúcim spôsobom:

1. Do výrobkov, zariadení je vložený alebo distribuovaný verejný kľúč.
2. Pomocou privátneho kľúča sa podpíše nový software.
3. Takto podpísaný software sa distribuuje do zariadení.
4. Zariadenie vďaka verejnému kľúču, dokáže overiť, či nový software pochádza od dôveryhodného zdroja.

Taktiež sa vďaka tomuto procesu overuje aj to, či je daný software nepoškodený, takže sa zabraňuje spusteniu neovereného alebo poškodeného softwaru.

Flash encryption (preložené šifrovanie pamäte flash) je metóda na zamedzenie získania nešifrovaných/čitateľných dát z pamäte flash. Ide o využitie najmä symetrickej kryptografie. Do zariadenia sa vkladá privátny kľúč, napr. vypálený do raz zapisovateľných registrov. Pri výmene softwaru sa takýto software zašifruje privátnym kľúčom, nahrá sa do zariadenia, ktoré si ho dešifruje, skontroluje či je spustiteľný, a použije ho. Je nutné si teda tento privátny kľúč uschovať v bezpečí. Aby bol tento systém čo najbezpečnejší, tak by sa mal pre každé zariadenie vytvárať unikátny privátny kľúč. To vo výsledku znamená, že aj keby unikne jeden privátny kľúč, ostatné zariadenia nie sú ohrozené.

Obe spomenuté technológie sa nepoužívajú len v IoT, ale nachádzajú sa aj bežných počítačoch alebo mobilných telefónoch[7].

3.5 Zabezpečenie komunikácie v rámci IoT siete

Ide taktiež o veľmi dôležitú časť IoT a rôzne komunikačné technológie sa dajú rôzne zabezpečiť. Tieto technológie využívajú teda rôzne praktiky spomenuté aj v článku [9], a dajú sa zhrnúť nasledovne:

Wi-Fi ponúka možnosť komunikácie zabezpečenej, napr. WPA1–3, aj nezabezpečenej tzv. plain text[8], pričom je aktuálne odporúčané používať WPA3. WPA3-Personal poskytuje väčšiu ochranu v prípade využívania jednoduchých hesiel. V prámci WPA3-Personal sa používa SAE[9], vďaka čomu sa zvýšila ochrana voči náhodnému hádaniu hesla. SAE je v jednoduchosti bezpečný protokol na autentifikáciu vyžadujúci len heslo [13].

Bluetooth Low Energy (BLE) – poskytuje rôzne vrstvy zabezpečenia. Sú postavené na dvoch módoch a to `režim zabezpečenia 1` a `režim zabezpečenia 2`. Oba využívajú symetrickú kryptografiu, algoritmus AES-128. `Režim zabezpečenia 2` je zložitejší a obsahuje 4 levely zabezpečenia:

- Level 1 – žiadne zabezpečenie
- Level 2 – šifrovanie a podpisovanie dát, ale nevykonáva sa autentifikácia počas výmeny kľúčov
- Level 3 – komunikácia je zabezpečená, ale párovací proces, ktorý vytvára zdieľaný kľúč využíva zraniteľný algoritmus

[7] **Verified Boot** *Android* [online]. [cit. 2023-04-20] Prevzaté z https://source.android.com/docs/security/features/verifiedboot

[8] plain text – čitateľné resp. nešifrované dáta

[9] SAE – Simultaneous Authentication of Equals – simultánne overovanie rovnocennosti

- Level 4 – párovanie už nie je zraniteľné kvôli používaniu Diffie-Hellmanovej eliptickej krivky s krivkou P-256 na odvodenie zdieľaného kľúča

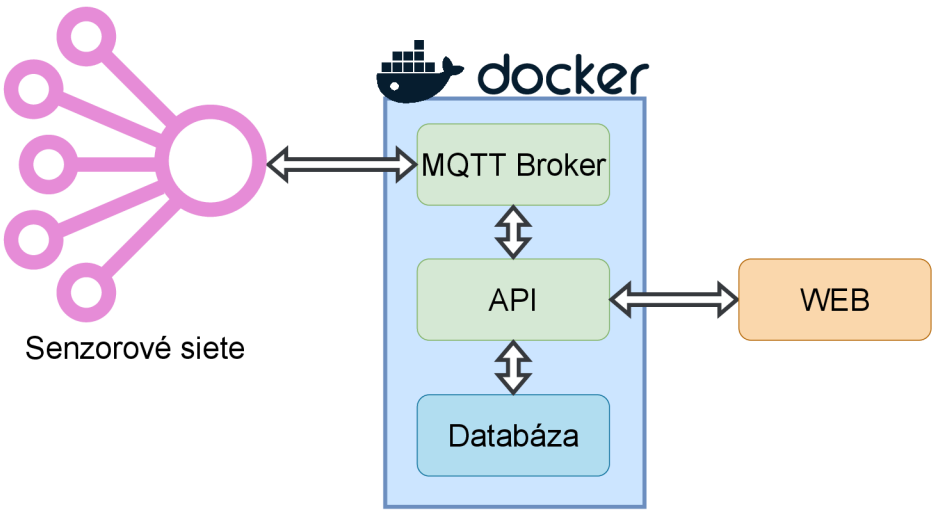
Zigbee – využíva AES-128 a pre zabezpečenie definuje dva typy kľúčov a to `network key`, ktorý je rozšírený medzi všetkými zariadeniami siete a poskytuje ochranu voči protivníkmi mimo siete. Taktiež viacero `link keys`, ktoré sú zdielané medzi dvojicami zariadení aby poskytovali end–to–end šifrovanie.

Thread – sieť je zabezpečená a šifrovaná, nové zariadenia musia preukázať svoju autenticitu, aby sa mohli pripojiť do siete. Tento mechanizmus je popísaný v [18].

MQTT – vyžaduje aby klient inicioval spojenie, keďže toto spojenie je postavené na TCP tak nie je šifrované, avšak existuje šifrovaná komunikácia, a to MQTT cez TLS so štandardným portom 8883. Existujú aj riešenia cez WebSockets[10] a tieto sa dajú taktiež zabezpečiť pomocou TLS.

3.6 Serverové technológie v kontexte IoT

Server je v IoT miesto, kde sa odohráva ukladanie a spracovanie dát, spravovanie zariadení, resp. celých sietí. Typicky umožňujú pripojenie z celého sveta, takže užívateľ nie je viazaný polohou aby mohol ovládať svoje zariadenia. Existujú aj riešenia IoT sietí, ktoré sú väčšinu času odpojené od internetu, a vyžadajú aby užívateľ bol pripojený v lokálnej sieti, v ktorej je aj celá sieť, aby ich bolo možné spravovať. Typicky sa v týchto offline sieťach nachádza lokálny server, ktorý sa stará o správu siete a ukladanie dát, napr. chaty. Návrh bežnej architektúry serveru je zobrazený na obrázku 3.8.



Obr. 3.8: Ukážka architektúry serveru a komunikačných kanálov

Všeobecne existuje veľké množstvo rôznych projektov, ktoré implementujú jednotlivé serverové časti, zobrazené na obrázku 3.8, teda MQTT Borker, API, Databáza, ale aj iné, napr. rôzne vyrovnávače záťaže atď. Na jednom servery typicky beží mnoho služieb súčasne, ktoré spolu komunikujú, a každá má inú funkciu. Tak ako je na obrázku 3.8, niektoré služby

[10] WebSockets – technológia umožňujúca obojsmernú komunikáciu

nemusia byť dostupné z internetu, napr. databáza, ktorá pracuje v pozadí, a z bezpečnostného hľadiska sa k nej dá pripojiť len v rámci serveru. Na vytváranie takéhoto prostredia v rámci serveru, ktoré môže obsahovať mnoho rôznych technológií sa typicky používajú kontajnery. Tieto kontajnery v podstate vytvárajú potrebné prostredie pre určitú aplikáciu, pričom tento kontajner je izolovaný od systému, aj od ostatných kontajnerov. Ďalšou výhodou použitia kontajnerov je škálovateľnosť serverovej časti, jej bezpečnosť a správa jednotlivých služieb. Z hľadiska škálovateľnosti sa kontajnery môžu rôzne pridávať a odoberať, podľa potrieb daného systému či vyťaženia. Populárnymi implementáciami týchto serverových technológii sú:

Docker využíva tzv. kontajnery na vytvorenie prostredia pre aplikácie. Využíva sa na prekladanie, testovanie a nasadenie aplikácii, pričom na jednom servery môže týchto kontajnerov bežať viacero. Každý kontajner je izolovaný od systému, a aj od ostatných kontajnerov. Kontajner sa vytvára typicky už z buď existujúcich obrazov, tzv. `images` alebo pomocou súboru `Dockerfile`. Vývojár má možnosť prepojiť dva rôzne kontajnery, a na budovanie systému zloženého z viacerých kontajnerov slúži `Docker compose`.

Dockerfile je súbor, ktorý má na každom riadku jeden príkaz, čo má daný kontajner vykonať. Slúži na pripravenie prostredia, inštaláciu a testovanie kontajneru. Jednoduchý príklad takéhoto súboru môže vyzerať nasledovne:

```
#FastAPI Dockerfile
FROM python:3.10

WORKDIR /code
ENV PYTHONPATH "${PYTHONPATH}:/code/app"
COPY ./requirements.txt /code/requirements.txt

RUN pip install --no-cache-dir --upgrade -r /code/requirements.txt

COPY ./src /code/app

CMD ["python", "app/main.py"]
```

Výpis 3.1: Ukážka súboru dockerfile

Dockerfile obsahuje inštrukčnú sadu, vďaka ktorej dokáže vytvoriť prostredie podľa požiadaviek aplikácie. Tieto príkazy sa vykonávajú v už bežiacom kontajnery, takže po jeho zmazaní nič nezostane na hostiteľskom systéme.

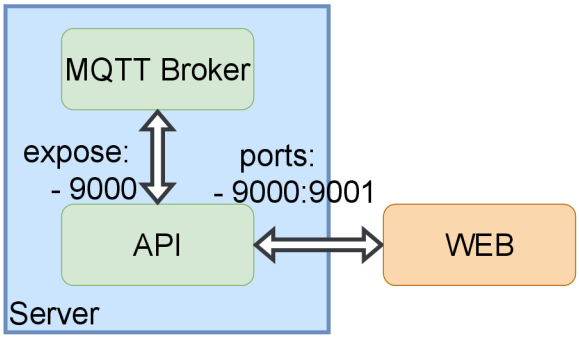
Docker compose [11] je nástroj na spúšťanie a definovanie viac–kontajnerových systémov. Konfiguruje sa pomocou yaml konfiguračného súboru, kde sú zadefinované inštrukcie na vytvorenie kontajneru, vzťahy a závislosti medzi kontajnermi. Jednoduchý konfiguračný súbor je zobrazený v 1 a predstavuje jednu službu. Tento konkrétny, definuje kontajner pre API, ktoré je znázornené na obrázku 3.9.

[11] **Docker Compose overview** *docker* [online]. [cit. 2023-04-20] Dostupné z `https://docs.docker.com/compose/`

```
api:
    image: bp_api:latest
    build:
        dockerfile: Dockerfile
    ports:
        - 9000:9001
    expose:
        - 9000
    depends_on:
        - MQTT_BROKER
```

Konfiguračný súbor 1: Ukážka konfiguračného súboru pre nástroj docker compose

Na vnútornú komunikáciu medzi kontajnermi sa používa príkaz `expose`, ktorý sprístupní port pre iný kontajner. Aplikácie spustené mimo docker compose nemajú k nemu prístup. Na namapovanie portu na port hostiteľského systému, sa používa príkaz `ports`. Príkaz má tvar `HOST:KONTAJNER`. Po vytvorení tejto väzby je daný port k dispozícii aj aplikáciam mimo prostredie docker compose. `Depends on` predstavuje vytvorenie závislosti na `MQTT BROKER`, to znamená spustenie `api` až po spustení brokera.



Obr. 3.9: Ukážka mapovania portov a komunikácie medzi kontajnermi

Na obrázku 3.9 sú zobrazené prepojenia medzi kontajnermi, s odpovedajúcimi príkazmi na toto prepojenie. V takomto prípade je možné sa pripojiť na API cez port 9000. Kontajnery súčasne poskytujú akúsi ochrannú vrstvu v rámci bezpečnosti. Ak sa útočník rozhodne získať kontrolu nad kontajnerom, tak zostane len v danom kontajnery, a má obmedzený prístup k hostiteľskému systému. Z hľadiska výkonu sú kontajnery veľmi dobre optimalizované. Majú veľmi malé nároky na výkon, a zanedbateľnú réžiu na výkon procesora, až na I/O[12] intenzívne systémy, kde by sa mali kontajnery využívať opatrne [10].

Neodmysliteľnou časťou na uchovávanie dát je databáza. Databázy sa rozdeľujú na dva typy a to relačné a nerelačné, pričom relačné sú vhodné na uchovávanie, vyhľadávanie a manipuláciu presne definovaných, štrukturovaných dát s presným typom. Tieto dáta ukladajú do tabuliek, príkladom implementácie takéhoto typu databázy môže byť PostgreSQL, MySQL a ďalšie. Na druhú stranu nerelačné databázy nie sú

[12] I/O input–output – vstupno–výstupné – v tomto kontexte napr. databáza

závislé na presne definovanej štruktúre záznamu, neukladajú dáta do tabuliek, ale ukladajú ich spôsobom vhodným pre daný typ, napr. JSON, Key-value pár, grafové databázy. Vďaka tomu sú flexibilnejšie, a dovoľujú kombinovať odlišné štruktúry. Príkladom implementácie **nerelačnej databáze** môže byť MongoDB, Apache Cassandra a ďalšie. IoT systémy vyžadujú spracovať veľké množstvo komplexných požiadavkov a transakcií. V takomto prípade môže byť **relačná databáza** vhodnejšou voľbou, keďže tieto databázy typicky zaručujú spoľahlivosť transakcií vďaka ACID.

Atomicita (**A**tomicity) – transakcia je nedeliteľná jednotka a vykoná sa ako celok, alebo sa nevykoná vôbec.

Konzistencia (**C**onsistency) – transakcia prevádza databázu z jedného konzistentného stavu do druhého konzistentného stavu.

Izolovanosť (**I**solation) – zaručuje, že pri súbežnom vykonávaní transakcií bude databáza v rovnakom stave, ako keby boli dané transakcie vykonané sekvenčne.

Trvalosť (**D**urability) – garantuje, že ak bola transakcia odovzdaná/potvrdená[13] zostane potvrdená, aj pri chybe systému, napr. výpadok napájania, zjednodušene povedané, po odovzdaní transakcie zmeny, ktoré vykonala sú uložené v pamäti nezávislej na napätí.

Vďaka ACID sú vykonané transakcie, a teda aj spracované dáta, validné aj pri rôznych nečakaných situáciach. Príkladom relačnej databáze môže byť napríklad PostgreSQL.

PostgreSQL – ide o open–source relačný databázový systém, prvá stabilná verzia bola vydaná ešte koncom minulého storočia. Využíva a rozširuje SQL jazyk a pridáva rôzne funkcie. Vďaka otvorenému zdrojovému kódu vznikli rôzne rozšírenia, napr. populárny doplnok PostGIS. Svoju popularitu získal vďaka osvedčenej architektúre, spolahlivosti, integrite dát, balíčkom pridaných funkcí a v neposlednom rade, vďaka odhodlanej open–source komunite[14].

Na vytváranie databáz sa môžu využívať rôzne prístupy, typicky sa jedná o využite jazyka SQL alebo techniky ORM:

SQL – je jazyk, ktorý umožňuje vývojárom pracovať s databázou, typicky sa jedná o relačné databázy, ale aj niektoré nerelačné často podporujú tento jazyk, resp. veľmi podobný jazyk, napr. Amazon DynamoDB, Apache Cassandra. Pomocou jednotlivých príkazov sa vykonávajú následne rôzne operácie nad databázou, napr. vytváranie tabuliek, pridávanie dát, mazanie dát, upravovanie dát atď.

ORM – (Object–Relational Mapping) je metóda, ktorá umožňuje vývojárom pracovať s databázou pomocou objektovo-orientovaného pohľadu. ORM vytvára tabuľky na základe zadefinovaných objektov v programovacom jazyku, napr. Python. Tento prístup následne umožňuje jednoduché vytváranie, upravovanie, mazanie a pridávanie dát, keďže je to veľmi podobné klasickému objektovo orientovanému programovaniu. Zadefinovanie tabuľky v jazyky Python s balíčkom SQLAlchemy vyzerá nasledovne:

[13] odovzdaná – príkaz commit

[14] **About PostgreSQL** *PostgreSQL* [online]. [cit. 2023-04-21] Dostupné z https://www.postgresql.org/about/

```python
class TemperatureDevice(Base):
    __tablename__ = "temperature_device"
    id = Column(BigInteger, primary_key=True, autoincrement=True)
    value = Column(Float)
    timestamp = Column(TIMESTAMP(timezone=True), primary_key=True)
    owner_id = Column(UUID, ForeignKey("devices.id_dev"))
    owner = relationship("Device", back_populates="temperature_device")
```

Kód 2: Ukážka vytvorenia návrhu tabuľky pomocou techniky ORM v jazyku Python

Úryvok kódu 2 zobrazuje príklad vytvorenia návrhu tabuľky priamo v programovacom jazyky Python, s využitím knižnice SQLAlchemy a techniky ORM. Súčasne sú vytvorené aj vzťahy, v tomto prípade `1:N`, jedno zariadenie vlastní `N` nameraných hodnôt.

Typické rozhranie, ktoré spája rôzne technológie a dovoľuje odlišným technológiám spolu komunikovať, sa nazýva API[15]. Aplikácie, ktoré spolu komunikujú, napr. MQTT Broker a Databáza z obrázku 3.8, majú jeden spoločný bod a to práve API. Existujú rôzne typy API napr. REST, SOAP.

REST – populárny spôsob vytvárania API. Ide o princíp klient/server, postavený na komunikácii pomocou HTTP dotazov. Umožňuje vykonávanie funkcí ako vytváranie, čítanie, mazanie a aktualizovanie dát, tiež známe ako CRUD[16]. Napr. na získanie záznamu využije HTTP GET dotaz, na vytvorenie POST, zmazanie DELETE, aktualizovanie PUT. Na posielanie správ využíva rôzne formáty ako JSON, XML, text, avšak typicky sa jedná o JSON. Vytvára koncové body (angl. endpoint), ktoré následne klient volá pričom sa následne vykoná nejaká CRUD operácia.

SOAP – zavisí na XML formáte správ a definuje veľmi silne typovaný rámec pre zasielanie správ. Využíva vzor RPC[17] pri ktorom sa odosielajú parametre pre volané funkcie, ktoré následne vrátia výsledok. Správy sa odosielajú len vo formáte XML. SOAP poskytuje vyššie zabezpečenie oproti REST.

API je rozhranie medzi dvoma (alebo viacerými) aplikáciami, ktoré zaručuje, aby všetky strany mali definovaný správny typ parametrov, návratových hodnôt a volaných funkcií. API sa tiež stará o správu prístupových práv, zabezpečenie a ďalšie aspekty, ktoré umožňujú bezpečnú a efektívnu komunikáciu medzi aplikáciami/procesmi a pod. Teda je to akýsi prostredník medzi jednotlivými aplikáciami, ktorý napomáha tomu aby si rozumeli, a splnili to, čo daná aplikácia požaduje. Typicky sa v IoT vyskytuje REST API, keďže umožňuje rôzne formáty správ a taktiež sú správy menšie z pohľadu veľkosti.

Na zobrazovanie uložených dát sa využíva napr. webová stránka, ktorá sa následne pripája na API, aby sa získali potrebné dáta. Ďalej to môže byť napr. mobilná aplikácia, ktorá sa tiež pripojí na to isté API. V prípade webovej stránky beží na servery webový server, ktorý po zadaní URL do vyhľadávača poskytne stránku užívateľovi, typicky dnes pomocou HTTPS. V kontexte IoT často táto stránka poskytuje všeobecné údaje o zariadeniach, vykresľuje grafy, umožňuje správu a konfiguráciu zariadení.

[15] API – Aplication Programing Interface
[16] CRUD – Create, Read, Update, Delete
[17] Remote Procedure Call – vzdialené volanie procedúr/funkcí

Jednou z možností ako pripojiť IoT zariadenia na server je pomocou MQTT. Bod, na ktorý sú pripojené všetky zariadenia v rámci MQTT sa nazýva broker. Tento broker beží neustále na servery a spravuje pripojenia, prijíma a doručuje správy podľa QoS požiadavkov, vytvára nové spojenia, vďaka asymetrickej kryptografii môže overovať autenticitu zariadenia. Implementácii MQTT brokerov je veľa, napr. EMQX, Mosquitto, HiveMQ, VerneMQ atď. Základnú funkcionalitu majú často rovnakú, líšia sa v rýchlosti implementácie podpory najnovších verzií MQTT, v programovacom jazyku v akom sú napísané, v škálovatelnosti atď. Bližšie porovnanie brokerov je popísané v článku [5], z ktorého vyplýva, že napr. EMQX je dobre pripravené pre IoT siete, vďaka konštantne dobrým výsledkom. Taktiež vďaka podpore zľukovania je pripravené na prípadnú škálovatelnoť, v prípade spravovania veľkého počtu zariadení.

3.7 Spotreba elektrickej energie

IoT zariadenia sa v podstate delia na dva druhy. Na zariadenia napájané alebo možné napájať z batérie. Typicky ide o koncové zariadenia, ktoré v určitých definovaných intervaloch vykonajú nejakú činnosť, a prejdú na určitú dobu do energeticky úsporného režimu, napr. spánku. Druhým druhom sú zariadenia s vyššou spotrebou, ktoré nie je efektívne možné napájať z batérie. Typicky sú to zariadenia, ktoré neustále vykonávajú činnosť napr. brány, smerovače, alebo zariadenia s väčším výpočtovým výkonom. Technológie ako Zigbee, Thread sú dobre uspôsobené na energetickú úsporu. Na druhú stranu pri Wi-Fi, napr. zariadenie ESP32, to nie je úplne jednoduchá úloha. Tak ako je popísané v článku [11], je najlepšie skombinovať zníženie frekvencie procesora a využiť režimy spánku, keďže spotreba ESP32 sa priemerne pohybuje pri bežnej prevádzke až na 200mAh. Spoločnosť Espresiff do svojich zariadení implementuje rôzne úrovne spánku. V režime spánku Light–sleep[18] je napájanie väčšiny komponentov, napr. väčšina RAM, CPU, periférii, redukované. V režime spánku Deep-sleep[19] je väčšina komponentov úplne vypnutá a zostávajú spustené len RTC hodiny, a ULP koprocesor.

Vďaka týmto rôznym optimalizáciám sa stalo populárné aj takzvané wireless energy harvesting – WEH[20]. Ako je popísané v článku [14], ide o sľubnú technológiu vďaka jednoduchosti použitia a dostupnosti. V článku prezentovali fungujúci senzor WEH. Skladá sa z jednotky WEH, ktorá ťaží RF energiu a jednotka PMU, ktorá kontroluje celý senzor a spravuje spotrebu energie. Zdroje RF rozdelili do dvoch kategórii. Špecializovaný zdroj, ktorý je vytvorený za cieľom dodania predom presne určeného množstva energie. Druhým je zdroj z okolitého prostredia, ktorým sú napríklad Wi-Fi prístupové body, rádiové vlny atď.

Zariadenia napájanie pomocou WEH sa bežne predávajú, napr. Zigbee Green Power[21]. Tieto zariadenia sú dokonca bez batérie, a sú napájané napr. z kinetickej energie stláčanie vypínača.

Zaujímavým spôsobom zdroja energie v rámci WEH môže byť taktiež kinetická energia davu ľudí. Ako je popísané v [16], ide o miesta, kde sa vyskytujú davy ľudí, ako nákupné centrá. V tomto prípade sa elektrická energia získava pomocou obojsmerných dverí, a takto získanej energii, môžu byť napájané rôzne billboardy, svetlá atď.

[18] Light–sleep – ľahký spánok

[19] Deep–sleep – hlboký spánok

[20] wireless energy harvesting – bezdrôtové ťaženie energie

[21] **Zigbee Green Power** *Zigbee* [online]. [cit. 2023-04-22] Dostupné z https://csa-iot.org/all-solutions/green-power/

3.8 Budúcnosť IoT

Tak ako bolo vyššie spomenuté, predpokladá sa, že trh IoT bude aj naďalej rásť. Väčšina analýz sa zhoduje na tom, že bude rásť, avšak líšia sa v tom koľko. V aktuálnej situácii je veľmi skloňovanou témou spotreba elektrickej energie, či už mestá, priemysel alebo obydlia ľudí. IoT zariadenia v takejto situácii môžu automatizovať a lepšie plánovať spotrebu elektrickej energie. Napríklad v článku [2] sa zamerali na optimalizáciu vykurovania, ventilácie a klimatizácie, s využitím IoT. Veľmi sa rozšírili domáce solárne elektrárne, kde IoT zariadenia môžu pomôcť a pomáhajú, vďaka plánovaniu. IoT zariadenia určité činnosti spúšťajú v dobe svietenia slnka, napr. pranie, polievanie záhrady atď. S týmto rastom a postupným pripájaním domácností a iných celkov do IoT sa ľudia vzdávajú svojho súkromia za cenu pohodlia. Preto musia firmy dbať na zabezpečenie a spoľahlivosť IoT zariadení, aby nedochádzalo k únikom citlivých údajov. V neposlednom rade aj užívateľ hrá veľkú rolu v rámci zabezpečenia, a mal by si vždy voliť silné heslá. Tak ako je popísané v [19], útokov na IoT sýstem je veľa, napr. útoky na zariadenie v sieti:

Node Capturing [22] – ide o pokus útočníka získať kontrolu nad uzlom, ktorý má pod sebou ďalšie senzory a aktuátory. Môže ísť aj pokus nahradenia uzla za podvrhnutý uzol, čo môže mať za následok prelomenie celého IoT systému.

Malicious Code Injection Attack [23] – zariadenia sú typicky aktualizované bezdrôtovo, čo poskytuje útočníkovi priestor pre spustenie prípadného škodlivého kódu.

Side-Channel Attacks [24] – HW resp. platforma môže umožniť cez spotrebu energie, architektúru procesoru, elektromagnetické vyžarovanie uník informácii o hardwari, a umožniť útočníkovi zaútočiť na slabiny v návrhu platformy. Tieto útoky sú teda založené na rôznych nedostatkoch danej platformy z pohľadu HW.

Tieto útoky nemusia byť len na koncové zariadenia a brány, ale aj útok na server, pričom ide zväčša o:

SQL Injection [25] – ide o jeden z najrozšírenejších útokov nie len v IoT. Pokúša sa vykonať škodlivý SQL príkaz napr. získanie celej databázy alebo jej vymazanie.

Man–in-the–Middle Attack [26] – v kontexte IoT ide o snahu útočníka získať kontrolu nad MQTT brokerom, pričom potom môže ovládať/odpočúvať celú komunikáciu.

Existuje mnoho ďalších ako odpočúvanie komunikácie, zahltenie serveru dotazmi atď. Je teda jasné, že zabezpečenie bude hrať taktiež kľúčovú rolu v budúcnosti IoT.

[22] Node Capturing – ovládnutie uzla
[23] Malicious Code Injection Attack – útok vložením škodlivého kódu
[24] Side-Channel Attacks – útoky cez bočný kanál
[25] SQL Injection – vloženie SQL
[26] Man–in-the–Middle Attack – útok muža v strede

Kapitola 4

Aktuálne dostupné riešenia

Trh s IoT je veľmi široký a saturovaný veľkými firmami, napr. Netatmo, Xiaomi, iGET, tieto spoločnosti sú významnými hráčmi na trhu nie len s IoT, ale všeobecne aj s elektronikou. Takéto spoločnosti zaberajú široké spektrum zariadení s rôznymi účelmi, napr. teplomery, spínače, snímače, a špecializujú sa vo veľkej miere nie len na Smart Home, ale aj na priemysel a podobne. Táto kapitola sa bližšie pozrie na to, ako veľké firmy budujú siete z hľadiska architektúry, vymenuje typické vlastnosti a porovná jednotlivé spôsoby architektúry resp. návrhu.

Typicky veľké spoločnosti ako tie vyššie vymenované, využívajú obe vysvetlené topológie. Napríklad Xiaomi má produkty využívajúce Zigbee a Wi-Fi. V prípade Zigbee ide o decentralizovanú topológiu, a v prípade Wi-Fi o centralizovanú topológiu. Typicky sa spoliehajú na jedno špecializované zariadenie, ktoré funguje ako brána. Tento fakt, v prípade Zigbee a Thread, vyplýva aj z podstaty toho, ako fungujú tieto technológie, a to je nutnosť mať minimálne jedno špecifické zariadenie, ktoré zvládne požiadavky danej siete spracovať a následne smerovať v internete.

Aj tento fakt spôsobil fragmentáciu trhu, keďže každá spoločnosť si v podstate vytvorila vlastný ekosystém a užívateľa uzavrela v ňom, takže si pre každú značku musel na ovládanie sťahovať ďalšiu aplikáciu, alebo kupovať špecifické zariadenia od danej značky. Avšak toto negatívum si uvedomili aj výrobcovia, a sú viditeľné pokroky aj v tejto oblasti. Napríklad dlhodobo existuje open–source projekt Home Assistant[1], ktorý práve rieši túto fragmentáciu, a umožňuje výrobcom a užívateľom zaintegrovať rôzne výrobky do tejto platformy, takže v konečnom dôsledku má užívateľ jednu aplikáciu na ovládanie zariadení od rôznych výrobcov (samozrejme špecifické nastavenia sú realizované cez aplikácie od výrobcov daného zariadenia). Fakt fragmentácie trhu potvrdzuje aj to, že niektorí výrobcovia sa rozhodli začať spolupracovať aj na zjednotení rôznych komunikačných štandardov do jedného s názvom Matter. V podstate podporuje viaceré štandardy, a teda zariadenia od rôznych výrobcov používajúce rôzne štandardy môžu spolu komunikovať. Tieto spájania sú samozrejme aj v rámci iných aplikácii napr. Google Home, Apple HomeKit a ďalšie, pričom zariadenia často podporujú viaceré tieto aplikácie.

Tomuto rozšíreniu IoT zariadení a ich vývoju taktiež napomáha aj rozširujúca sa komunita nadšencov, dostupnosť a jednoduchosť vývoja. Väčšie komunity ľudí, prevažne vývojárov, sa vytvorilo okolo spoločností Arduino, Raspberry, Espressif, a ďalších. Spoločnosť Arduino neposkytuje len HW, ale taktiež vytvorilo mnoho open-source knižníc so zamera-

[1] **Home Assistant** *Home Assistant* [online]. [cit. 2023-04-22] Dostupné z https://www.home-assistant.io/

ním na uľahčenie programovania HW. Tieto knižnice nepodporuje len spoločnosť Arduino, ale aj ďalšie, napr. Espressif, STM. Taktiež programovací jazyk MicroPython určite dovoľuje širšiemu spektru ľudí niečo vytvoriť. Vytvára akúsi nadstavbu, za cieľom uľahčenia programovania, avšak za cenu straty určitých možností rôznych platforiem, resp. odtieneniu sa od HW.

4.1 Zhodnotenie aktuálnych riešení

Aktuálne riešenia bezdrôtových senzorových sietí obsahujú rôzne senzory a aktuátory, ktoré sú v podstate základným stavebným kameňom IoT sietí. Typicky sa tieto siete spoliehajú na špecifické brány, ktoré tieto siete pripájajú do internetu. Samozrejme existujú riešenia, kde hraničný smerovač nie je nutný, typicky ide o riešenia, kde je len jedno zariadenie a to priamo podporuje Wi-Fi. Tu ale nastáva problém s dosahom, keďže lokálna Wi-Fi musí mať dosah až k tomu zariadeniu. Zariadenia v sieti sú typicky jednoúčelové, to znamená, že merajú len jednu fyzikálnu veličinu, a teda užívateľ ich má nainštalovaných viacero. Tieto siete sa používajú skoro všade, napr. ovládanie vykurovania, ovládanie svetiel v domácnostiach, v priemysle na automatizáciu, v zdravotníctve atď. Typicky sa tieto siete následne pripájajú na cloud od daného výrobcu, ktorý následne ponúka spracovanie dát, a ich vizualizáciu. Typicky tieto dáta ukladá a poskytuje po určitú dobu, pričom výrobca určuje aj akú periodicitu dát ukladá, napr. meraná teplota každých 10 min. V rámci prepojiteľnosti zariadení od rôznych výrobcov sa situácia zlepšuje, ale často za cenu orezania funkcií, napr. v Google Home lampu ide zapnúť a vypnúť, ale nejde presne určiť teplotu farby, pričom v aplikácii od výrobcu to možné je.

V rámci spájania IoT zariadení do väčších celkov je dobrým príkladom už vyššie spomenutý Home Assistant. Ide teda o open–source projekt a aj komunitu, ktorá sa zameriava na automatizáciu v rámci domu, a kladie dôraz na bezpečnosť a súkromie. Tento projekt typicky využíva lokálny server spustený na Raspberry PI. Vďaka komunite podporujú rôzne zariadenia, pričom tieto integrácie rôznych značiek naprogramovali typicky nadšenci alebo priamo výrobcovia, napr. Philips Hue, Bosch, Tesla Powerwall. Túto rôznorodosť má za dôledok práve aj fakt, že ide o open–source projekt a každý kto chce môže prispieť. Samozrejme toto nie je jediný projekt, ktorý takto združuje IoT komunitu, avšak vyniká v množstve a rôznorodosti, unikátnosti podporovaných zariadení[2].

Ako ďalším potencionálnym projektom, ktorý sa snaží zjednotiť zariadenia od rôznych výrobcov a rôzne komunikačné technológie je Matter, avšak zatiaľ sa jedná o mladý projekt, ktorý svoje možnosti ešte len ukáže, ale jeho budúcnosť je sľubná. Poskytuje rôzne výhody či už pre užívateľa, napr. podpora zariadení od rôznych výrobcov, ale aj pre výrobcov.

Populárnym spôsobom ukladania dát je aktuálne taktiež pomocou cloudových služieb od veľkých spoločností ako Microsoft Azure, Amazon AWS, Google Cloud atď., ktoré poskytujú all–in–one riešenia priamo pripravené na IoT. Celú serverovú časť dodajú a spravujú oni, čo uľahčuje a zrýchľuje vývoj, avšak často za vyššiu prevádzkovú cenu.

[2] **Integrations** *Home Assistant* [online]. [cit. 2023-04-22] Dostupné z https://www.home-assistant.io/integrations/

4.2 Motivácia a vymedzenie cieľov práce

Z popísaných typických vlastností bezdrôtových senzorových sietí je jasné, že sa v prípade architektúr jedná už o zabehnuté a overené spôsoby budovania sietí. Avšak aj v takto zaplnenom trhu sa nájde priestor na vylepšenia a prípadné skúmanie. V prípade tejto práce sa jedná o zameranie sa na bezbránový spôsob budovania siete, takže odpadá smerovač, ktorý typicky musí podporovať komunikačný štandard siete a Wi-Fi. V rámci serveru zameranie sa na škálovateľnosť a jednoduchú prenositeľnosť. Taktiež popísanie a navrhnutie zabezpečenia zariadení a celej komunikácie. V podstate vytvorenie softwarového ekosystému bezdrôtovej senzorovej siete, vytvorenie architekturý od koncového zariadenia až po server za cieľom splnenia nasledujúcich vlastností:

Rozšíriteľnosť – do siete sa musia dať dynamicky pripojiť nové zariadenia. Serverové komponenty sa musia dať dynamicky rozširovať, keďže počet pripojených zariadení v jeden moment môže dynamicky rásť, a teda aj samotná záťaž resp. odozva.

Zabezpečenie – zariadenia musia byť zabezpečené voči nechcenému vyčítaniu kódu. Taktiež aj komunikácia musí byť medzi zariadeniami a serverom zabezpečená.

Bezbránový prístup – užívateľ nepotrebuje žiadnu bránu aby mohla sieť fungovať, a teda musí fungovať aj jedno zariadenie samostatne.

Uchovávanie dát – dáta musí byť možné uchovať, aby sa následne zozbierané dáta mohli zobrazovať na webe.

Zobrazovanie dát na webe – zobraziť aktuálne prípadne aj historické dáta na webovej stránke.

Jednoduchosť používania – je nutné aby zariadenia bolo jednoduché používať. V prípadoch ako inicializácia siete, prevádzka, pridávanie zariadení, cieliť na jednoduchosť používania.

V podstate ide o využitie overenej typickej architektúry IoT siete, avšak novou vlastnosťou a to je bezbránové riešenie, čo so sebou nesie určité výhody ale aj nevýhody.

Motivácia pre vypracovanie tejto práce

Z hľadiska prečo som sa ja osobne, autor, rozhodol pre túto tému je, že už dlhšiu dobu sa zaujímam o odvetvie IoT. Nadchla ma myšlienka ovládať a pripájať rôzne veci na internet. Začal som s meraním teploty v systéme vykurovania a následne som sa presunul k inteligentnému polievaniu. V rámci tohto sa mi podarilo nadviazať spoluprácu s komerčným sektorom. Išlo o vytvorenie platformy, ktorá umožňuje pripojiť zariadenia bez internetu, práve do internetu. Z tejto spolupráce je výsledkom komerčne predávaný produkt. Postupne pri vývoji zariadenia som sa zúčastňoval rôznych súťaží, pričom som prezentoval nie len jedno zariadenie, ale už celú IoT mesh sieť. Pri prieskume trhu práve podobných produktov ako táto mesh sieť, som sa nakoniec rozhodol pokračovať a vytvoriť celý ekosystém. Tým, že je v pláne podpora rôznorodých zariadení, je výhodné mať v úplnej správe server. Preto som následne navrhol celú architektúru serveru, pričom sú to v podstate jednoduché kontajnery. Tieto kontajnery sa dajú rôzne rozširovať, a taktiež celé serverové riešenie je jednoduché presunúť k inému poskytovateľovi. Nakoniec už to bolo dostatok technológii a vedomostí na zváženie bakalárskej práce z tohto projektu, pričom ma zaujímalo či je možné využiť Wi-Fi na vybudovanie bezdrôtovej senzorovej siete.

Kapitola 5

Návrh systému

Cieľom tejto bakalárskej práce je navrhnúť ekosystém, resp. architektúru celého systému bezdrôtovej senzorovej siete, zvoliť vhodnú platformu, a túto sieť pripojiť na server, vytipovať vhodné vlastnosti siete. Práca sa zamerala aj na odlíšenie sa od existujúcich riešení, keďže trh s IoT je veľmi saturovaný veľkými spoločnosťami a tým poskytnúť riešeniu konkurencieschopnosť. Nezamerala sa len na koncové zariadenia a sieť, ale aj na ekosystém ako celok, čiže od koncového zariadenia až po architektúru serveru. V nasledujúcich odsekoch sa načrtne návrh celého ekosystému, pričom sa pôjde od serveru po koncové zariadenia.

5.1 Návrh architektúry serveru a jeho komponentov

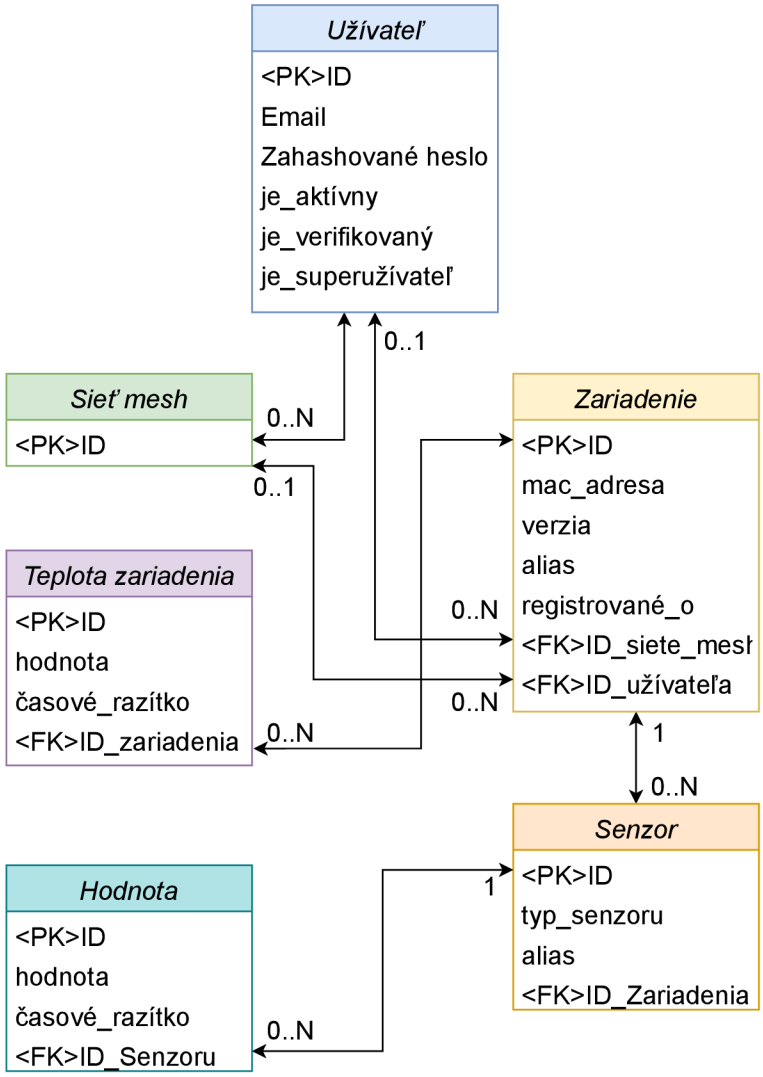
V rámci serveru sa návrh zameral na prenositeľnosť a rozšíriteľnosť celého systému. Na splnenie požiadaviek boli zvolené Docker kontajnery. Kontajnery umožnili horizontálne a vertikálne škálovanie. Horizontálne spočíva v rozširovaní počtu kontajnerov jednotlivých služieb, čo sa nemusí diať iba v rámci jedného serveru, ale aj na viacerých rozmiestnených rôzne po svete. Služby, ktoré zabezpečujú takéto škálovanie pre Docker kontajnery, sú napr. Docker Swarm a Kubernetes. Vertikálne škálovanie znamená pridávanie výkonu v rámci serveru, napr. zvýšenie veľkosti RAM, pridanie vlákien/jadier CPU. Tieto kontajnery sú definované pomocou Docker Compose, takže je veľmi jednoduché prípadné pridanie ďalšieho komponentu. Týmto komponentom môže byť napr. vyrovnávač zaťaže, ktorý rozdeľuje požiadavky medzi rovnaké kontajnery, ktoré práve využívajú horizontálne škálovanie. Príkladom služby, ktorá umožňuje rozdeľovať zaťaž je Nginx, popis a príklad využitia je v článku [8]. Z pohľadu programovacích jazykov sa na serverovej časti využíva len jazyk Python, pričom pri práci s databázou sa využíva v kombinácii s technikou ORM. Ďalej sa popíšu dôvody, prečo sa využili jednotlivé komponenty servera:

MQTT Broker – využíva sa open–source broker EMQX, princíp fungovania brokera je popísaný v 3.1. Splňuje všetky požiadavky na vlastnosti serverových komponentov. Výhodou je taktiež, že EMQX komunita vytvára a udržiava priamo obraz pre docker kontajner. Poskytuje aj správcovský prehľadový web, takže administrátor vie, čo sa s brokerom, ale aj MQTT správami deje.

API – využíva sa REST API vysvetlené v 3.6, a konkrétne na implementáciu je použitý open–source projekt FastAPI, ktorý má veľmi silnú komunitu, vďaka ktorej existuje mnoho doplnkov, napr. MQTT klient pre FastAPI. Využíva sa najmä asynchrónne programovanie, vďaka ktorému je táto knižnica veľmi rýchla a vhodná aj na veľké

projekty. Poskytuje automaticky generovanú dokumentáciu ku koncovým bodom na základe parametrov a CRUD operácii. Taktiež je obrovskou výhodou priama podpora relačných databáz s balíčkom SQLAlchemy a to má za výsledok veľmi rýchle prepojenie API s databázou.

Databáza – využíva sa relačná databáza, princíp fungovania vysvetlený v 3.6, keďže má presne definované hodnoty v jednotlivých stĺpcoch, a teda presne definuje, aké dáta budú z danej tabuľky získané. Využíva sa konkrétne open–source relačná databáza PostgreSQL s rozšírením TimescaleDB, ktoré umožňuje vytvárať tabuľky lepšie pripravené na dáta zbierané v čase, resp. uľahčuje proces návrhu a následnú prácu s takýmito dátami, napr. teplota. Návrh databázy reprezentovaný ER diagramom je zobrazený na obrázku 5.1.



Obr. 5.1: ER Diagram zobrazujúci navrhnutú databázu

Web – web je implementovaný v jazyku TypeScipt pomocou frameworku Vue3 [1]. Vue je moderný nástroj na tvorbu webových aplikácii. Ako predloha je použitá open–sourcová stránka Vuestic Admin[2]. Je vytvorený len jednoduchý príklad stránky na zobrazovanie základných hodnôt, avšak pripravenej na ďalšie rozšírenia.

Tak ako je zobrazené na obrázku 3.8, jednotlivé komponenty sú v docker kontajneroch. Tieto komunikačné kanály sú:

- Broker komunikuje obojsmerne len s API
- Prístup k databázy má len API, takže s databázou komunikuje len API
- Webová stránka komunikuje len s API

Žiadna iná komunikácia v rámci kontajnerov nie je možná, keďže nemajú otvorené iné porty, ako tie cez ktoré komunikujú. Porty na brokerovy a API sú otvorené do internetu, aby sa mohli pripojiť zariadenia resp. užívatelia.

Z pohľadu zabezpečenia sú na servery SSL certifikáty, privátne a verejné kľúče. Broker vlastní všetky tri vymenované, aby na základe asymetrickej kryptografie fungovalo TLS (šifrovanie komunikácie), a aby bolo možné overiť autenticitu jednotlivých zariadení. Web server v základnej implementácii funguje iba cez HTTP, avšak s prípravou pre riešenie pomocou Nginx, ktoré následne poskytne aj HTTPS pripojenie medzi API a webovou stránkou.

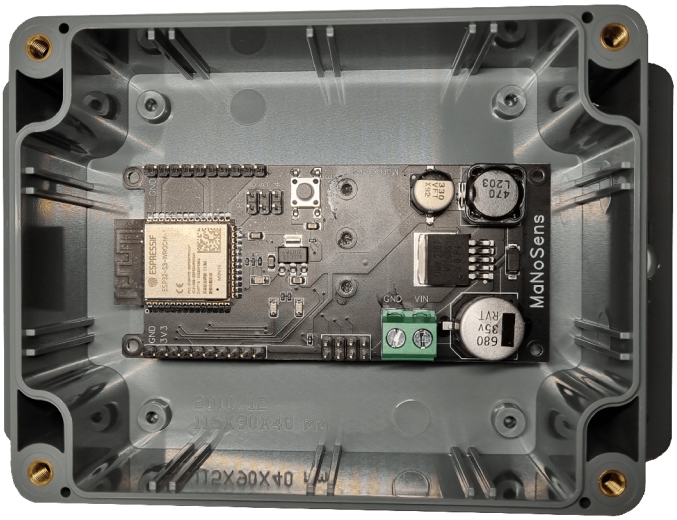
Komunikácia medzi serverom a zariadeniami prebieha primárne cez zabezpečené MQTT správy, avšak napr. aktualizácia zariadení prebieha cez HTTPS. Typicky server beží na statickej IP adrese a vývojár/výrobca si zaobstará doménu, ktorú spáruje s ip adresou serveru. Takto funguje aj návrh v tejto práci. Vďaka faktu, že každé zariadenie má unikátne kľúče asymetrickej a aj symetrickej kryptografie uložené v sebe, dokážu jednoznačne preukázať svoju autenticitu. Má to aj tú výhodu, že ak by unikli kľúče z jedného zariadenia, napr. by boli vyčítané, tak ostatné zariadenia nie sú ohrozené, či už sa jedná o pripojenie na brokera alebo o šifrovanie pamäte flash.

5.2 Návrh architektúry koncových zariadení

S ohľadom na konkrétne vlastnosti, najmä, že riešenie má byť bezbránové, sa v práci použilo zariadenie ESP32–S3. Vďaka tomu, že disponuje Wi-Fi a rôznymi možnosťami zabezpečenia je vhodné na takúto prácu. Konkrétne sa využilo zariadenie navrhnuté na mieru. Toto zariadenie disponuje skoro rovnakými charakteristikami ako bežný vývojový kit ESP32–S3. Rozdiel je v menšom počte GPIO pinov, a má väčšie rozsahy z pohľadu napájania od 5V do 30V. Motiváciou vytvorenia vlastného návrhu dosky bol aj fakt, že v dobe implementácie práce bola rada ESP32–S3 vývojových kitov relatívne nedostupná. Táto vývojová platforma je zobrazená na obrázku 5.2.

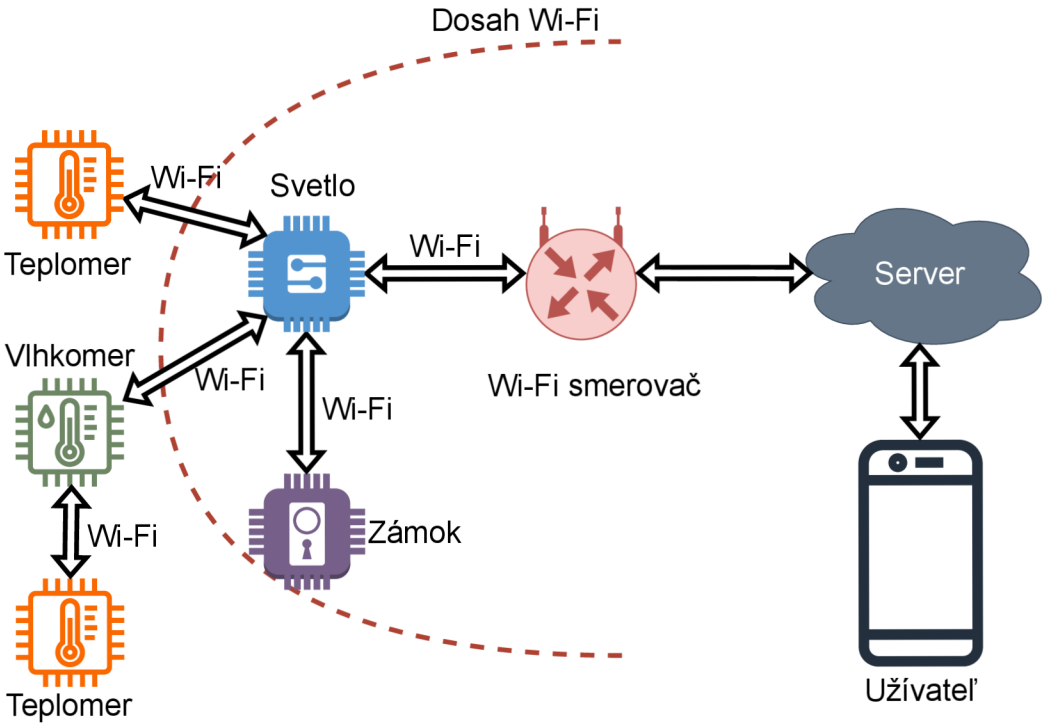
[1] **Vue** *Evan You* [online]. [cit. 2023-04-23] Dostupné z https://vuejs.org/

[2] **Vuestic Admin** *Epicmax* [online]. [cit. 2023-04-25] Prevzaté z https://github.com/epicmaxco/vuestic-admin



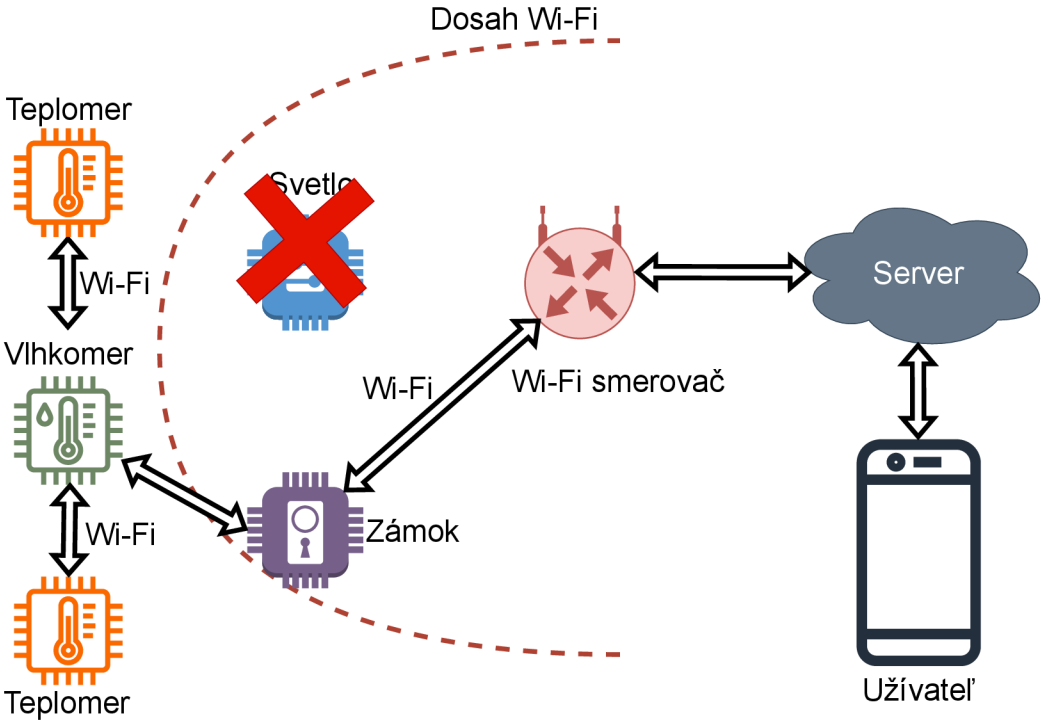
Obr. 5.2: Vývojová platforma na využitá na implementáciu

Vďaka využitiu Wi-Fi je možné vytvoriť bezbránovú IoT sieť, keďže Wi-Fi je veľmi rozšíreným štandardom, a IoT sieť nebude potrebovať žiadnu špecializovanú bránu na komunikáciu medzi sieťou a internetom, ale sa priamo pripojí na lokálny smerovač. Spoločnosť Espressif vytvorila framework pre vývoj na ich zariadeniach s názvom ESP-IDF, využíva programovací jazyk C. Tieto zariadenia taktiež podporujú knižnice od komunity Arduino, popísané v 4. Táto práca využíva oba frameworky, avšak primárne ESP-IDF. Vybranou topológiou je decentralizovaná mesh sieť, vďaka typickým vlastnostiam ako samousporiadanie, obnovenie po výpadku atď. Takto vytvorená Wi-Fi mesh sieť môže vyzerať ako na obrázku 5.3. Dôležité je si všimnúť využívaný štandard na komunikáciu medzi zariadeniami v rámci bezdrôtovej siete. Všetky zariadenia v rámci celej siete komunikujú pomocou Wi-Fi, pričom dodržiavajú všetky typické vlastnosti tejto topológie. Veľmi dôležitou vlastnosťou je aj fakt, že stačí aby bolo jedno zariadenie v dosahu lokálnej siete Wi-Fi, toto zariadenie následne rozširuje dosah siete ďalej v priestore.



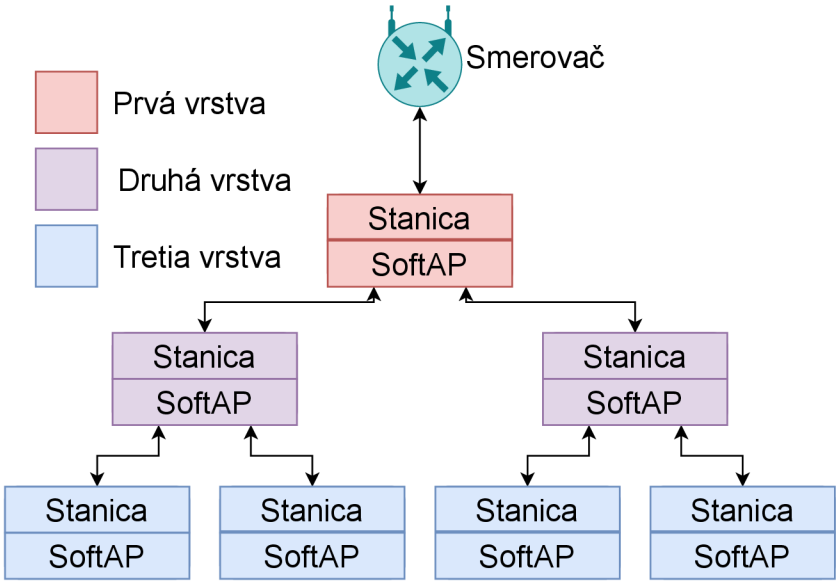
Obr. 5.3: Ukážka Wi-Fi mesh siete

Na obrázku je to simulované pomocou červenej čiary, pričom aj zariadenia za ňou sa dokážu pripojiť. Ďalšou výhodu je taktiež, že Wi-Fi mesh sieť nemusí spoliehať na jedno konkrétne zariadenie, jeden bod zlyhania. Použitím Wi-Fi vzniká `n` bodov pripojenia, presne je to počet zariadení, ktoré sú v dosahu lokálnej siete Wi-Fi a taktiež mesh siete. Tento princíp môže vyzerať ako na obrázku 5.4.



Obr. 5.4: Ukážka reorganizácie siete v prípade výpadku hlavného zariadenia

Pri výpadku zariadenia, ktoré bolo hlavné resp. poskytovalo pripojenie do internetu, sa toto zariadenie, ktoré zlyhalo, nahradí zariadením, ktoré disponuje pripojením na lokálny smerovač. Ostatné zariadenia sa reorganizujú napr. tak ako je zobrazené na obrázku 5.4. Tu je aj vidieť výhoda takéhoto prístupu a poukázanie na `n` bodov pripojenia. V porovnaní so Zigbee, ktoré má jedného koordinátora, takže jeden bod pripojenia a teda ak zlyhá, tak sieť bude odpojená, je Wi-Fi mesh sieť typicky odolnejšia voči výpadku. Taktiež v porovnaní s Threadom, ktorý má taktiež `n` bodov pripojenia, avšak spolieha sa na špecializované zariadenia, ktoré zvládajú v tomto prípade dva štandardy, Wi-Fi a Thread. Zariadenia postavené na ESP32 dokážu byť súčasne pripojené na sieť a aj sieť ďalej rozširovať, vďaka možnosti koexistencie dvoch módov Wi-Fi, jedno slúži ako stanica a druhé ako prístupový bod. Následne prepojovanie vyzerá ako na obrázku 5.5.



Obr. 5.5: Ukážka koexistencie dvoch módov Wi-Fi na ESP32S3 pri ESP-WIFI-MESH

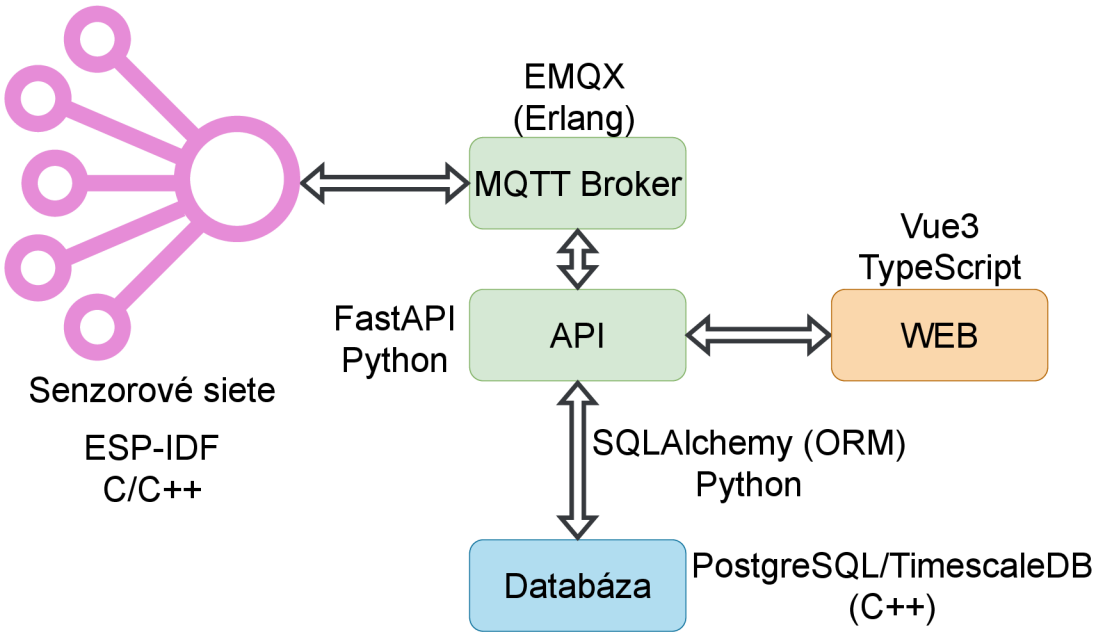
Koncové zariadenia s ohľadom na jednoduchosť inicializácie resp. prvého spustenia, disponujú jednoduchým web serverom, ktorý ponúka základnú inicializáciu zariadenia, čiže napr. prihlasovacie údaje k lokálnej sieti Wi-Fi. Pre jednoduchosť získania webovej stránky zo zariadenia sa využíva mDNS (multicast DNS), typická technológia pre vyhľadávanie napr. tlačiarní v rámci lokálnej siete. To vo výsledku znamená, že užívateľ nemusí zadávať žiadnu ip adresu, ale napr. doménu `bp.local` priamo v prehliadači. Je avšak nutné, aby bol pripojený na sieť, v ktorej je pripojené aj zariadenie, alebo na sieť vytvorenú priamo zariadením.

Z pohľadu zabezpečenia zariadenia disponujú, už ako bolo vyššie spomenuté, jedinečnými asymetrickými a symetrickými kľúčmi. Na ochranu kódu je využitá technika bezpečného zavedenia kódu, súčasne s šifrovaním pamäte flash. Komunikácia s MQTT Brokerom sa šifruje pomocou asymetrickej kryptografie, pričom sa dá overiť aj autenticita zariadenia. Šifrovanie v rámci siete ako takej zabezpečuje samotný štandard Wi-Fi, pričom sa využíva WPA2/WPA3 popísané v 3.5.

Kapitola 6

Implementácia ekosystému siete a testovanie

V rámci tejto kapitoly sa bude postupovať veľmi podobne ako v predošlej kapitole. Implementácia sa rozdelí na serverovú časť a časť s bezdrôtovou senzorovou sieťou. Implementácia a jej výsledok je najmä zameraný na vytvorenie šablóny pre jednoduché rozšírenie a pre prípadné znovupoužitie celého systému. Zameranie sa na konkurencieschopnosť implementácie a to použitím state–of–the–art technológii a splnením špecifikovaných vlastností navrhnutých v 4.2. V predošlej kapitole v rámci návrhu boli vymenované navrhnuté technológie, pre zhrnutie využitých technológii môže slúžiť obrázok 6.1.



Obr. 6.1: Ukážka využitých knižníc/projektov a použitých programovacích jazykov

Na obrázku 6.1 je možné vidieť rôznorodosť použitých technológii so silným zameraním na to, aby boli ich zdrojové súbory open–source a aby komunita okolo nich bola aktívna. Výhodou tohto prístupu je využitie aktualizácií, prípadne rôznych rozšírení v budúcnosti. V prípade nutnosti je taktiež možné jednotlivé komponenty v rámci serveru vymeniť jednoducho za iné, vďaka docker kontajnerom, ktoré sú vysvetlené v 3.6. Na obrázku 6.1 je

vždy popísaný framework/balíček, ktorý bol primárne využitý a pod ním je jazyk, v ktorom je napísaná implementácia. Broker a databáza majú programovacie jazyky podľa toho, v akom jazyku sú ich zdrojové súbory primárne implementované. Nasledujúce kapitoly zhrnú a vysvetlia jednotlivé časti implementácie, pričom sa postupuje od serverovej časti až po samotné zariadenia.

6.1 Implementácia serverovej časti

Jednotlivé komponenty sú v docker kontajneroch a celý tento systém je spojený pomocou docker compose, táto technológia je vysvetlená v 3.6. Toto má obrovskú výhodu v jednoduchosti spustenia celej serverovej časti, v podstate stačí zavolať `docker compose build` a následne `docker compose up`. Následne sa stiahnu a vytvoria jednotlivé kontajnery, pričom pri API, Nginx, Web je pripravený aj dockerfile, ktorý nachystá celé prostredie pre daný kontajner. Jednotlivé kontajnery využívajú mimo otvorených portov medzi sebou aj ďalšiu funkcionalitu, a to vytváranie siete. V podstate ide o rozdelenie kontajnerov podľa toho, kto s kým bude komunikovať. Takáto sieť sa vytvára pomocou konfiguračného príkazu `networks`, ktorý je následne zadaný v rámci konfiguračného yaml súboru, ktorého princíp je vysvetlený v 1. Taktiež sa pri vytvorení kontajnerov vytvoria perzistentné zväzky (angl. volumes), ktoré uchovávajú dáta, napr. databázu, nastavenia brokera atď. Tieto zväzky sa vytvárajú pomocou príkazu `volumes`, a vytvárajú sa len tam, kde je nutné uchovanie dát aj po vypnutí resp. vymazaní kontajneru. Nižšie sa vysvetlia jednotlivé komponenty a ich implementácia:

MQTT Broker

Je využitý broker EMQX, pričom s ohľadom na uchovanie konfigurácie sa pri inicializácii vytvorí zväzok. Inak sa v podstate jedná o veľmi jednoduché spustenie, keďže vďaka kontajneru ho stačí len zapnúť a vybrať príslušné porty, v tomto prípade porty: `18083` prehľadový web, `1883` nezabezpečená tcp komunikácia, `8883` zabezpečená tcp komunikácia. Následne broker už beží a zariadenia sa môžu pripojiť. Broker umožňuje vytvoriť rôzne listenery (preložene komponenta, ktorá odchytáva udalosti vyvolané vo volanej metóde/funkcií), pričom každý môže mať inú konfiguráciu, napr. TCP, TLS, Web Sockety, zabezpečené Web Sockety. V prípade zabezpečenej komunikácie je nutné vytvoriť certifikát, privátny a verejný kľúč, a zadať ich tomuto listenerovi na použitie. Zadať ich je možné pomocou konfiguračných súborov, ale aj priamo pomocou webu. V rámci implementácie bola využitá možnosť nahratia cez web. Web poskytuje rôzne štatistiky, či už o celej sieti alebo o jednotlivých klientoch, dáta sú zobrazené pomocou grafov, rôznych tabuliek a výpisov.

API

Využíva sa jazyk Python a FastAPI, pričom sa vytvára konkrétne REST API, vysvetlené v 3.6. Implementujú sa jednotlivé koncové body, pričom tieto body reprezentujú rôzne CRUD operácie. Prepojenie medzi brokerom a API zabezpečuje MQTT klient implementovaný balíčkom FastApi-MQTT[1]. Tento klient sa hneď po spustení API pripojí na brokera a odoberá všetky správy, ktoré následne spracováva. Nejedná sa o jediný prístup ako sa dá prepojiť s API. Ďalšou možnosťou je vytvoriť v rámci EMQX brokera dátový most. Tento

[1] **FastApi-MQTT** *sabuhish* [online]. [cit. 2023-04-26] Prevzaté z https://github.com/sabuhish/fastapi-mqtt

prístup je špecifický pre EMQX brokera, kde po prijatí správy na brokerovi a na základe predom definovaných pravidiel je zavolaný koncový bod na API. Prístup na API pomocou MQTT klienta je jednoduchší a rýchlejší na vývoj, keďže dátové mosty je nutné pracne vytvárať na brokerovi. Takto vytvorený koncový bod môže vyzerať následovne:

```
@router_users.get("/get-devices", response_model=list[schemas.DeviceBase])
async def get_users_devices(PARAMETRE):
# Telo funkcie
```

Kód 3: Ukážka koncového bodu volaného operáciou GET

Úryvok kódu predstavuje koncový bod volaný s operáciou GET, pričom odošle späť pole zariadení. Pomocou tohto prístupu sú implementované všetky koncové body, pričom niektoré nemusia mať ani návratovú hodnotu. S ohľadom na prípadné rozšírenia sú využité FastAPI smerovače. Ide o logické členenie koncových bodov do skupín. V príklade úryvku kódu 3 sa takáto logická skupina nazýva `router users`. Ide o praktiku, ktorá sa využíva prevažne pri väčších projektoch a vytvára lepší prehľad v koncových bodoch, ale aj v samotnom kóde. Celé API je implementované pomocou asynchrónneho programovania. Taktiež aj následné volania operácii nad databázou sú asynchrónne. FastAPI taktiež ponúka rozšírenie na vytváranie, ukladanie a spravovanie užívateľských účtov, s názvom FastAPI Users[2]. Tieto účty sa následne využívajú na prihlasovanie na prehľadovom webe ekosystému.

Databáza

Využíva sa PostgreSQL s rozšírením TimescaleDB. Na zistovanie prehľadu, správu, ovládanie databázy PostgreSQL sa typicky využíva aplikácia pgAdmin[3]. Ide o aplikáciu, ktorá sa priamo pripojí na databázu a je možné si prehliadať jednotlivé tabuľky, alebo volať požadované SQL príkazy (jazyk SQL je vysvetlený v 3.6). TimescaleDB je plne kompatibilná s PotgreSQL a len pridáva rôzne možnosti na spracovanie a uchovávanie dát zbieraných v čase. V prípade teploty zariadenia je práve vhodné využiť nový typ tabuľky, tzv. `hyper table`, ktorá je určená na uchovanie dát, ktorých je mnoho a majú závislosť na čase. Táto tabuľka sa pri bežnom používaní chová rovnako ako klasická tabuľka, avšak jej sila tkvie v získavaní určitých úsekov dát, napr. dáta určené na grafy. Funkcia `time bucket` je zaujímavá v tom, že súčasne zvládne vrátiť určitý úsek dát a aj spriemerovať dané hodnoty, v podstate ich pripraviť už priamo na použitie. Funkcia `time bucket` pre získanie dát za posledných `n` dní je využitá v kóde 4.

[2] **FastAPI Users** *FastAPI Users* [online]. [cit. 2023-04-27] Prevzaté z https://fastapi-users.github.io/fastapi-users/11.0/

[3] **pgAdmin** *pgAdmin org* [online]. [cit. 2023-04-27] Prevzaté z https://www.pgadmin.org/

```python
async def get_mb_last_n_temperature(db: AsyncSession, dev_id: uuid.UUID, n: int):
    stmt = (
        select(
            func.time_bucket(text("'10 minutes'"),
                             models.TemperatureDevice.timestamp).label("bucket"),
            func.avg(models.TemperatureDevice.value).label("avg_value"),
        )
        .where(models.TemperatureDevice.timestamp >
               text(f"NOW() - INTERVAL '{n} days'"))
        .where(models.TemperatureDevice.owner_id == dev_id)
        .group_by("bucket")
        .order_by("bucket")
    )
    result = await db.execute(stmt)
    data = result.fetchall()
    return data
```

Kód 4: Ukážka využitia funkcií rozšírenia TimescaleDB

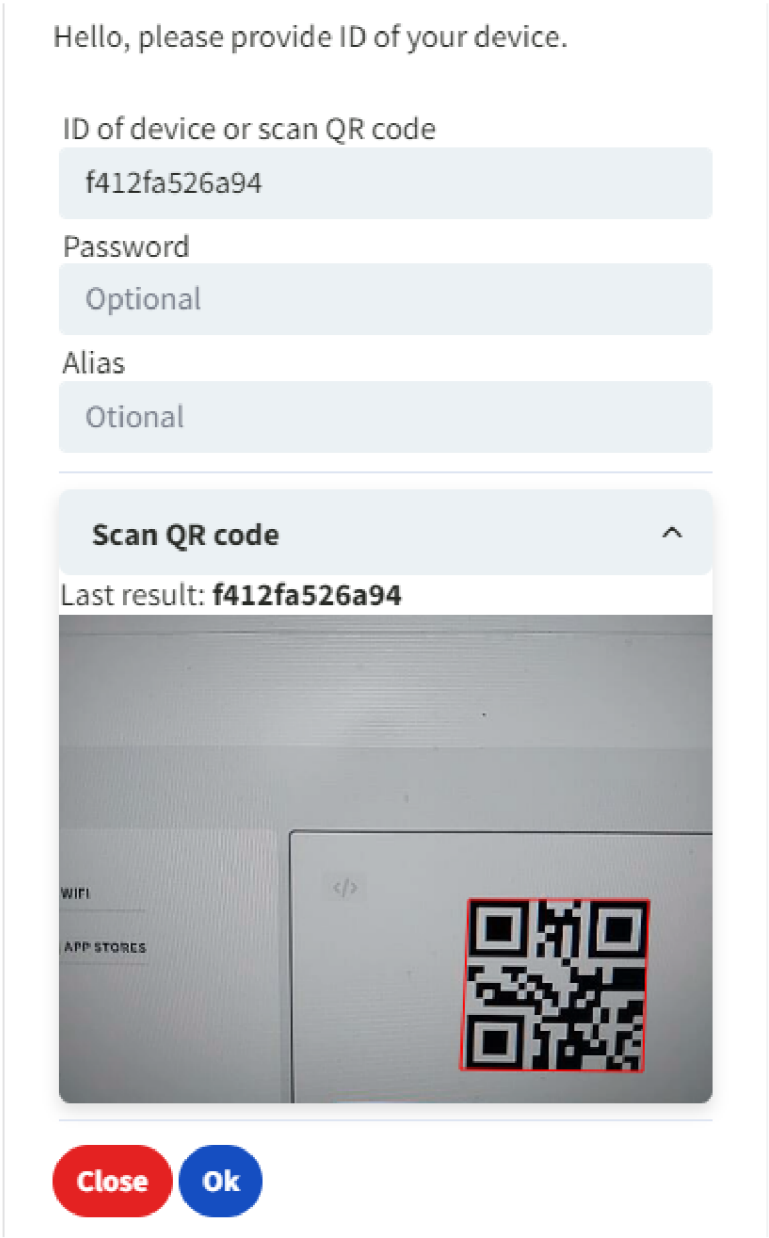
Tak, ako je vidieť, je využitá technika ORM vysvetlená v 3.6. Ďalším krokom bude vysvetlenie implementácie prepojenia API a databázy. Aj keď nejde o komponentu v samostatnom kontajnery, ale implementovanú v rámci API, ide o veľmi dôležitú časť celého ekosystému.

Prepojenie API s databázou

Na implementáciu sa využíva programovací jazyk Python a balíček SQLAlchemy, tento nástroj je popísaný v [4]. Ide o sadu nástrojov, určených na prácu s SQL a ORM. Ako takýto prístup k databáze vyzerá, je zobrazený v 4. Tento balíček priamo podporuje FastAPI, takže v rámci fungovania a vývoja nenastal žiaden problém. Databáza je s API prepojená asynchrónne, aby sa dosiahlo čo najrýchlejšie spracovanie požiadaviek. To sa vyznačuje v kóde práve kľúčovým slovom `await`, ktoré stojí pred funkciami vykonávajúcimi operácie nad databázou, príkladom môže byť opäť 4.

Web

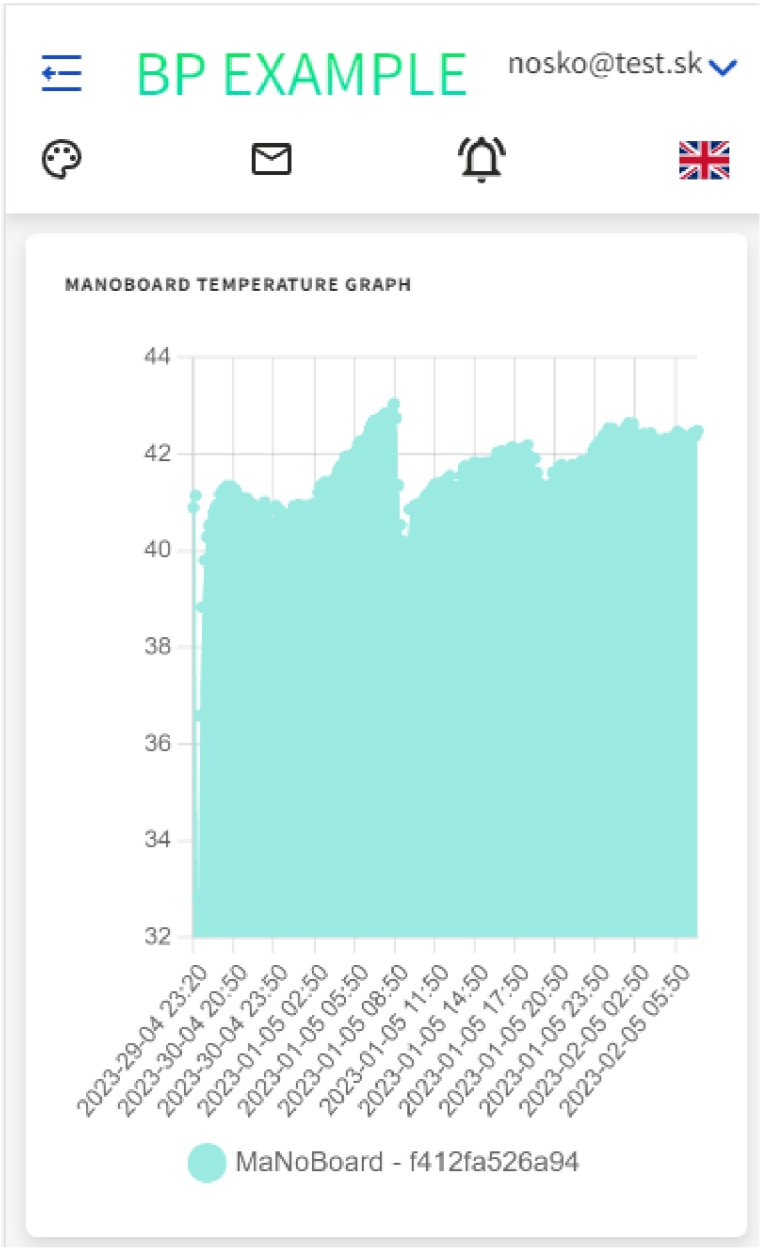
Je implementovaný len na zobrazenie základného prehľadu o zariadeniach. Je naprogramovaný v jazyku TypeScript s využitím frameworku Vue3. Taktiež je implementované registrovanie a prihlasovanie užívateľov. Na pridávanie zariadení užívateľom je vytvorený formulár, ktorý číta pomocou webkamery alebo fotoaparátu QR kód. Náhľad stránky na pridávanie zariadení užívateľom je zobrazený na obrázku 6.3.



Obr. 6.2: Ukážka webovej stránky zobrazujúcej pridanie nového zariadenia pomocou QR kódu

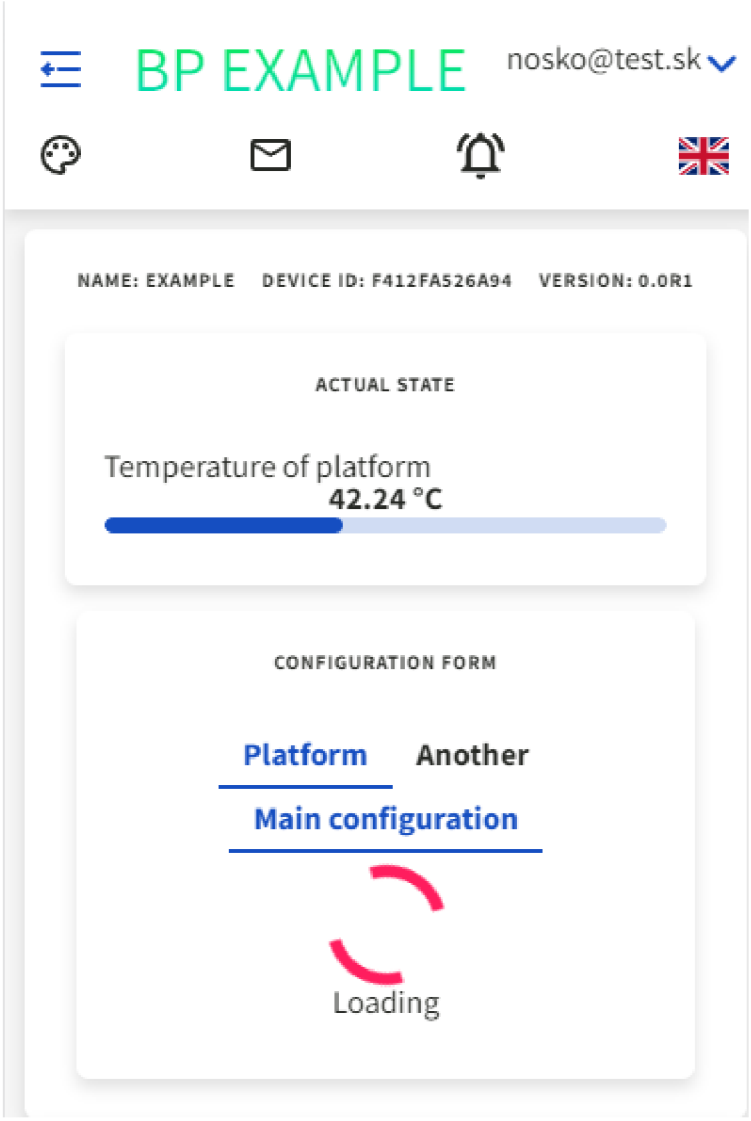
Takto implementované pridávanie zariadení cieli na jednoduchosť používania. Vďaka použitiu QR kódu užívateľ nemusí opisovať dlhý reťazec náhodných čísel a písmen, avšak v prípade potreby je možné identifikátor zadať aj ručne.

Vďaka tomu, že sa v rámci celého ekosystému využíva databáza na uchovávanie dát, tak je možné vykresľovať rôzne grafy. Príklad je možne vidieť na obrázku 6.3.



Obr. 6.3: Ukážka webovej stánky zobrazujúca graf teploty zariadenia

Obrázok 6.3 zobrazuje príklad grafu, kde hodnoty grafu boli získané vďaka `time bucket`. Získavanie normalizovaných hodnôt pre takéto typy grafov, veľmi jednoduché. Dáta do tohoto grafu vracia funkcia zobrazená v kóde 4. Web volá priamo koncové body definované v API, z ktorých následne získava potrebné hodnoty, prípadne vykoná prihlásenie/registráciu nového užívateľa. Samozrejmosťou je aj zobrazovanie aktuálnych hodnôt, stránka je pripravená na rozšírenia podľa potrieb a v základe zobrazuje aktuálnu teplou zariadenia. Ak by bolo vyžadované, je možné aby web zobrazoval rôzne konfiguračné hodnoty. Táto stránka je zobrazená na obrázku 6.4.



Obr. 6.4: Ukážka webovej stánky zobrazujúca aktuálny stav zariadenia

Práve rozšírenia konfigurácie zariadenia by boli umiestnené v kartičke s názvom `CONFIGURATION FORM`. Bolo cielené na to, aby bol web nachystaný na rozšírenia o rôzne ďalšie druhy zariadení a senzorov, pričom sa môže ďalej stavať na tomto implementovanom základe. V podstate implementovaná časť je pripravená na prevádzku a ponúka základný prehľad o jednotlivých zariadeniach v sieti.

6.2 Implementácia senzorovej siete

Táto sekcia je zameraná na implementáciu senzorovej siete. V rámci implementácie sa využíva programovací jazyk C/C++, a na implementáciu Wi-Fi mesh siete je využitá knižnica od spoločnosti Espressif. Na preklad a nahrávanie preloženého firmwaru je použitý prekladový systém PlatformIO[4], ktorý podstatne uľahčuje správu knižníc a aj následný preklad

[4] **What is PlatformIO?** *PlatformIO* [online]. [cit. 2023-04-12] Prevzaté z https://docs.platformio.org/en/latest//what-is-platformio.html

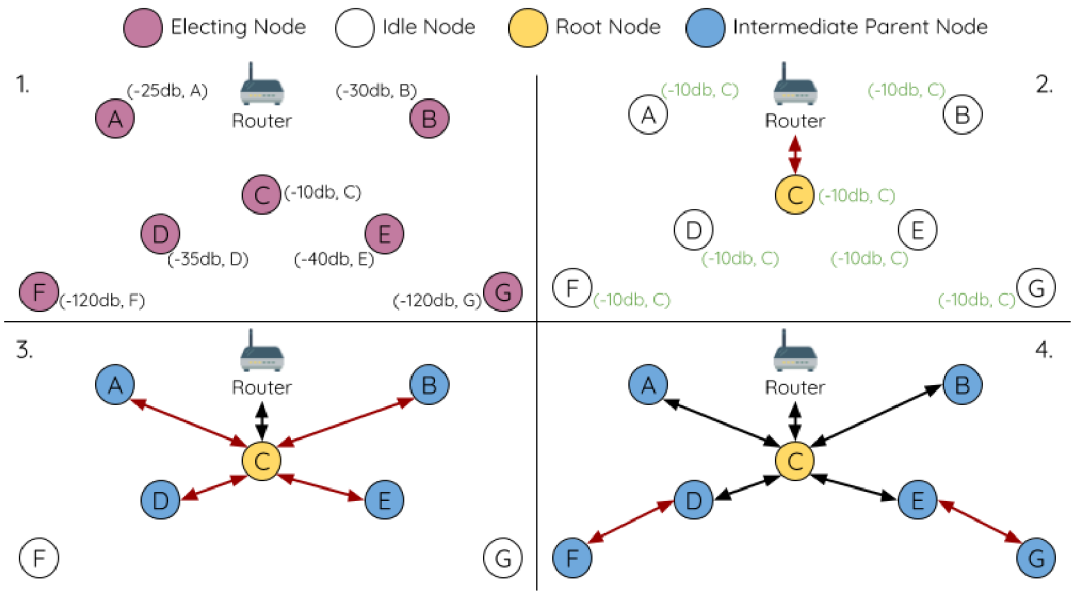
a nahrávanie firmwaru do zariadenia. Implementácia je vytvorená z príkladov implementácii priamo od výrobcu, čo vo výsledku znamená, že samotné vytvorenie mesh siete je jednoduché. Mesh sieť je otestovaná výrobcom a taktiež sa urýchlil a zjednodušil vývoj vďaka priloženým príkladom.

Inicializácia siete vyžaduje vlastne len prihlasovacie údaje k lokálnej sieti Wi-Fi a heslo, ktoré bude použité na zaheslovanie Wi-Fi mesh siete. Táto inicializácia vyzerá nasledovne:

```
// Časť inicializácie k lokálnej Wi-Fi
memcpy((uint8_t *) &cfg.router.ssid, LOCAL_SSID, strlen(LOCAL_SSID));
memcpy((uint8_t *) &cfg.router.password, LOCAL_PWD, strlen(LOCAL_PWD));
// Časť inicializácie k mesh Wi-Fi sieti
memcpy((uint8_t *) &cfg.mesh_ap.password, "MESH_PASSWD", strlen("MESH_PASSWD"));
```

Kód 5: Ukážka koncového bodu volaného operáciou GET

Inicializuje sa štruktúra, ktorá obsahuje ďalšie nastavenia, napr. maximálny počet pripojených klientov, automatické prepínanie kanálov, atď. Po nastavení sa mesh sieť spustí a spustené zariadenie sa pokúsi nájsť mesh siete v okolí, ak nenájde žiadnu sieť, ku ktorej by sa pripojilo, tak sa pripojí k lokálnej Wi-Fi. Veľmi dôležitou časťou procesu vytvárania mesh siete je voľba, ktoré zariadenie bude koreňové/hlavné. Doslova sa jedná o voľby, keďže každé zariadenie v rámci siete odošle všetkým zariadeniam aký silný má signál na lokálny smerovač. Zariadenie s najlepším signálom vyhráva a stáva sa hlavným zariadením. Zistenie najvhodnejšieho hlavného zariadenia sa deje tak, že každý uzol vysiela svoju silu signálu so smerovačom, ak uzol zistí silnejšiu silu signálu od iného uzlu, tak následne vysiela ďalej túto zistenú silu signálu. To vo výsledku znamená, že sa do celej siete spropaguje ako kandidát zariadenie s najlepším signálom so smerovačom. Vysielaný signál predstavuje hlas pre kandidáta s danou hodnotou signálu na smerovač. Pre lepšiu predstavu je vytvorený diagram:



Obr. 6.5: Ukážka budovania mesh siete, implementovanej knižnicou ESP-WIFI-MESH[5]

Na obrázku 6.5 je vidieť práve proces voľby koreňového uzlu, a následného vytvorenia celej siete. Pričom popis k jednotlivých štádií siete je nasledovný:

1. Na začiatku všetky uzly začnú vysielať ich MAC adresu a silu signálu k smerovaču.
2. Po niekoľkých iteráciách sa stane to, že všetky uzly budú vysielať MAC adresu a silu signálu najvhodnejšieho kandidáta. Najvhodnejší kandidát sa pripojí na smerovač lokálnej siete Wi-Fi.
3. Následne sa uzly A,D,E,B pripoja na koreňový uzol.
4. Zvyšné uzly sa pripoja na najbližšie uzly v sieti, a tým je výstavba siete dokončená

Tento proces sa vykonáva vždy pri spustení zariadení, v prípade, že sieť už je aktívna, tak sa nové zariadenie pripojí do siete a automaticky usporiada. Počas celého behu programu si jednotlivé zariadenia v sieti udržiavajú smerovaciu tabuľku, takže je možné správy adresovať priamo na jednotlivé uzly v sieti.

Štruktúra správ, resp. paketov, odosielaných medzi zariadeniami v sieti je kompatibilná so štandardom dátových rámcov definovaných štandardom IEEE 802.11, a dokonca je možné tieto správy adresovať na adresy mimo mesh sieť. Obsah správ je možné posielať vo formátoch binárny, HTTP, JSON, MQTT. V rámci implementácie je použitý práve formát správ MQTT. Ak nejaké zaradenie/uzol potrebuje odoslať nové dáta, tak dané dáta sa zabalia do príslušného formátu a odošlú na koreňový uzol. Tento uzol dané dáta len prepošle na MQTT brokera. Za celú sieť je pripojený na brokera práve len koreňový uzol. Odosielanie vnútorných správ vyzerá v kóde nasledovne:

```
// Vytvorenie JSONu pre MQTT správy
std::ostringstream ss;
ss << "{internal_temp: " << temp << ",device_id:" << chipId << ','
<< "timestamp:" << esp_mesh_get_tsf_time() << "}";
mesh_data_t data;
// Pripravenie dát
data.data =  (uint8_t*)strdup(ss.str().c_str());
data.size = ss.str().length();
// Protokol v akom sú odosielané správy
data.proto = MESH_PROTO_MQTT;
// Typ adresovanie či ide o adresovanie v rámci siete alebo mimo
data.tos = MESH_TOS_P2P;
// NULL predstavuje adresu koreňového/hlavného uzlu
esp_mesh_send(NULL, &data, MESH_PROTO_MQTT, NULL, 0);
```

Kód 6: Ukážka ako sa odosielajú správy na koreňové zariadenie v rámci siete

V úryvku kódu 6 ide o komunikáciu v rámci siete, ak by správa nebola adresovaná na koreňový uzol, ale na iné zariadenie v sieti, tak by vo funkcii `esp_mesh_send(...)` nebol prvý parameter `NULL` ale `MAC adresa` cieľového zariadenia. Celé adresovanie funguje teda na jednotlivých MAC adresách, tieto MAC adresy sú vypálené v zariadení priamo pri výrobe

[5]Obrázok prevzatý z https://docs.espressif.com/projects/esp-idf/en/v5.0/esp32/api-guides/esp-wifi-mesh.html **Automatic Root Node Selection** *Espressif*

samotného čipu. Môže sa veľmi ojedinele stať, že dôjde ku kolízii MAC adries, v takomto prípade je možné danú MAC adresu zmeniť na inú.

Implementácia z tejto práce poskytuje zariadeniam časové razítko, pričom synchronizácia času v rámci siete bude popísaná v nasledujúcom odseku.

Synchronizácia času v rámci siete

Pri implementácii sa využila na synchronizáciu času funkcia na synchronizáciu času TSF (Timing Synchronization Function), vysvetlená v [7], zavedenej v rámci samotného štandardu IEEE 802.11. Zjednodušene ide o to, že všetky stanice v rámci nejakej WLAN (Wireless Local Area Network – prelož. bezdrôtová lokálna sieť) si musia udržiavať synchronizovaný časovač medzi sebou. To vo výsledku znamená, že hlavné/koreňové zariadenie má dva synchronizované časovače, jeden má synchronizovaný s reálnym časom vďaka protokolu NTP, a druhý má synchronizovaný v rámci mesh siete. Ak zariadenie chce odoslať dáta na brokera tak, ak sa jedná o koreňové zariadenie, správu pošle priamo s časovým razítkom, avšak ak iné zariadenie chce poslať správu, pošle ju s časovým razítkom TSF. Následne koreňové zariadenie vypočíta správny čas. Tento výpočet je relatívne jednoduchý, keďže sa vlastne len vypočíta doba cesty správy a tá sa odpočíta od aktuálneho času. Doba cesty sa počíta ako rozdiel medzi dvomi časovými známkami TSF. Zariadenia ESP32 na získanie TSF časovej známky disponujú funkciou `esp_mesh_get_tsf_time()`, ktorá je použitá aj v príklade kódu 6.

Ochrana proti výpadku serveru

Veľkou výzvou, ktorá nastane ak sa vytvára sieť, ktorá nemá ani jedno špecifické zariadenie na spravovanie siete, je závislosť na servere. Ak sa server alebo jeho komponenty dostanú do chybového stavu, alebo sa rovno vypnú, tak bezbránová sieť môže prestať fungovať. Príkladom môže byť situácia, kedy užívateľ má vytvorené pravidlo na prepojenie vnútornej teploty s vykurovaním, ak teplota klesne pod `x`, spusti kotol. Ak je bezbránová sieť čisto závislá na servere, tak ak tento server spadne, celá sieť prestane fungovať. Tento problém je popísaný v dalšej sekcii.

Na túto ochranu boli vymyslené dve techniky reprezentované tabuľkami pravidiel. Jedna tabuľka obsahuje smerovanie hodnôt, a teda ktoré hodnoty sa majú kam posielať. Druhá tabuľka obsahuje pravidlá správania, čo sa má vykonať, ak daná požadovaná hodnota doputuje na konkrétne zariadenie. Obe tabuľky sú vo formáte JSON a je možné vytvárať viacero pravidiel. Príkladom jedného pravidla smerovacej tabuľky hodnôt môže byť:

```
"0":{
      "from":"f4:12:fa:52:48:14",
      "to":"f4:12:fa:52:48:14",
      "key_name":"internal_temp"
}
```

Konfigurácia 7: Ukážka pravidla pre smerovanie hodnôt v rámci ochrany proti výpadku serveru

Koreňové zariadenie bude preposielať hodnotu označenú ako `internal_temp` medzi zariadeniami s vyššie uvedenými MAC adresami. Na tomto preposielaní je ďalej postavená

aj druhá tabuľka, ktorá určuje, čo dané hodnoty na cieľovom zariadení ovplyvnia. Takéto pravidlo je zobrazené v 8.

```
"f4:12:fa:52:48:14":{
      "value":40,
      "key_name":"internal_temp",
      "from":"f4:12:fa:52:48:14",
      "true_condition":"1",
      "false_condition":"0",
      "what":"GPIO4",
      "operation":"<"
}
```

Konfigurácia 8: Ukážka definovania pravidla a aké činnosti sa majú vykonať

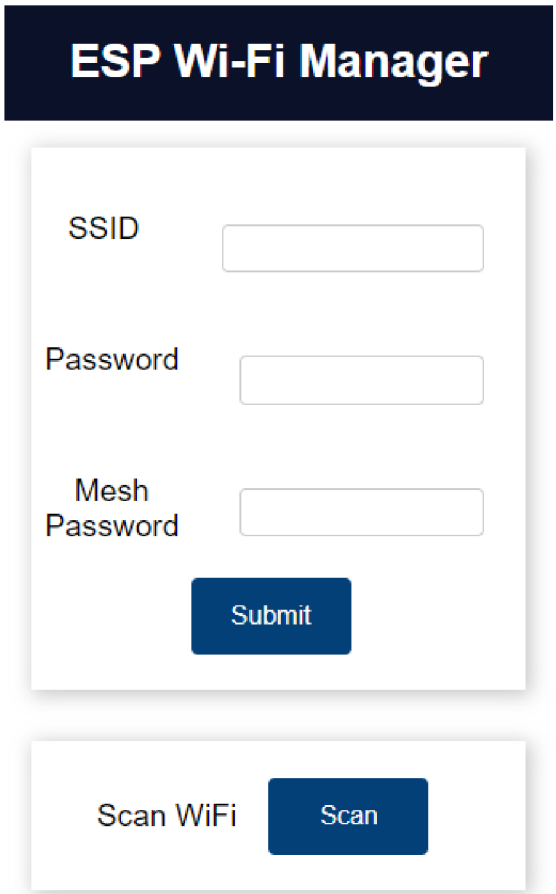
Návrh tejto tabuľky cielil aj na rôznorodosť možností využitia, preto sa v nej aktuálne nachádza viacero kľúčov, ktoré sú zamerané na inú logiku. Následné spracovanie a vykonanie úlohy sa vykonáva priamo v kóde. V tomto prípade ide o vypínanie resp. zapínanie LEDky na základe teploty zariadenia. Tieto tabuľky si udržiava koreňové zariadenie a pri spustení ich získa od serveru[6]. Ak sa pripojí nové zariadenie do mesh siete, tak koreňové zariadenie zistí, či existujú nejaké pravidlá, ktoré musí dané zariadenie plniť, a ak áno, tak mu ich pošle a dané zariadenie si ich nastaví. Prípadné implementovanie editoru týchto pravidiel na webe by malo byť jednoduché vďaka využitiu formátu JSON.

6.3 Impementácia kocových zariadení

Táto sekcia je zameraná na jednotlivé koncové zariadenia, a ich implementáciu a životný cyklus, zabezpečenie, ale aj prípravu a nasadenie zariadení do prevádzky. Na implementáciu firmwaru sa využívajú knižnice z ESP-IDF a Arduino, a rôzne možnosti operačného systému RTOS, najmä plánovač úloh.

Jednotlivé zariadenia majú implementovanú možnosť vytvárať webové servery, ktoré užívateľovi distribuujú webovú stránku, ak sa pripojí na určitú adresu. Aby nedochádzalo ku kolíziám, že viacero webových serverov je spustených naraz na rovnakej doméne, tak je spustený vždy len jeden server v rámci siete, a to ten na koreňovom zariadení. Táto stránka poskytuje základný prehľad, a je pripravená na prípadne rozšírenia v budúcnosti. Súčasne s webom je spustené aj REST API, aby web mohol zobrazovať rôzne údaje, či už o samotnom zariadení alebo o celej sieti. Zariadenie disponuje dvomi rozličnými webovými servermi, ktoré sú určené na niečo iné, a nikdy nie sú spustené súčasne. Jeden je teda prehľadový a druhý je určený na prvotnú inicializáciu. Aby bolo možné tieto stránky poskytnúť pripojenému užívateľovi, musia byť nahrané v pamäti flash. Stránka na inicializáciu je zobrazená na obrázku 6.6.

[6]Ide o proof–of-concept vyriešenia problému s výpadkom serveru, a teda nie sú implementované všetky možné situácie, ktoré môžu nastať



Obr. 6.6: Ukážka inicializačnej webovej stánky

Zariadenie teda poskytuje plnohodnotnú stránku, pričom užívateľ sa pri prvom spustení zariadenia len pripojí na Wi-Fi, ktorú vytvára nové zariadenie, a vyplní formulár zobrazený na obrázku 6.6. S cieľom na jednoduchú inicializáciu má užívateľ možnosť si nechať vypísať všetky Wi-Fi siete v dosahu s ich silou signálu. Vďaka využitiu technológie mDNS, užívateľ nemusí zadávať žiadnu ip adresu do vyhľadávacieho poľa prehliadača, ale zjednodušenú doménu, ktorú zvolí vývojár, napr. `bp.local`. Po inicializácii sa zariadenie pokúsi pripojiť na Wi-Fi sieť zadanú vo formulári, pričom súčasne sa spustí druhý prehľadový web. Ak užívateľ zadal nesprávne údaje, zariadenie sa snaží v určitých intervaloch pripojiť. Ak uplynie čas `300 sekúnd`, zariadenie vymaže uložené údaje a znova sa dostane do stavu prvej inicializácie. Počas tohto intervalu beží web, na ktorom je taktiež možné vynútiť vymazanie uložených údajov, je to za cieľom aby užívateľ nemusel čakať daný interval. Následne sa celý proces opakuje do momentu, kedy sa zariadenie pripojí na Wi-Fi. Samozrejmosťou je perzistencia uložených údajov.

Rozdelenie pamäte flash na oddiely

Keďže na zariadení je nutné uchovávať rôzne druhy dát, firmware, bootloader, webové stránky, perzistentné dáta, je v rámci implementácie využitá špecifická partition table (prelož. tabuľka oddielov). Použité zariadenie v rámci implementácie disponuje `16 MB` pamäte flash. Táto pamäť je rozdelená na rôzne oddiely, pričom sa práve využíva tabuľka oddielov na špecifikovanie veľkosti a druhu jednotlivých segmentov. Táto tabuľka sa udáva sa

vo formáte `CSV`, pričom sa v nej udávajú typy jednotlivých segmentov a ich veľkosť, prípadne či daný oddiel má byť zašifrovaný. Na pomery dnešných veľkostí pamäte flash sa môže zdať táto pamäť relatívne malá, avšak napríklad pre firmware sú vyhradené `4 MB`, a reálne preložený celý implementovaný firmware zaberá približne `30%` z toho. Pre firmware sú vyhradené dva oddiely, pričom jeden je vždy aktuálne spustený firmware, a druhý je využívaný v prípade OTA aktualizácie. Tento prístup umožňuje návrat k predchádzajúcej verzií.

Aktualizácia zariadenia

Koreňové zariadenie je možné aktualizovať, tento proces sa vyvolá pomocou MQTT správy zo serveru. Zariadenie sa pomocou TLS pripojí na aktualizačný server a následne prebehne poslanie nového firmwaru. Tento aktualizačný server nie je znázornený na obrázku 3.8, pretože nie je povinnou súčasťou serveru. Tento server v najjednoduchšej implementácii má jeden koncový bod REST API, pričom tento koncový bod vracia zašifrovaný súbor. V rámci zvýšenej ochrany je tento firmware zašifrovaný na strane serveru, a posielajú sa zašifrované dáta, čiže ide o dvojitú ochranu. Zariadenie si následne prijatý firmware rozšifruje. V základe knižnice od spoločnosti Espresiff umožňujú aktualizáciu len firmwaru, v rámci tejto práce bolo implementované rozšírenie, ktoré umožňuje aktualizovať akýkoľvek oddiel v rámci pamäte flash. Je to implementované z toho dôvodu, že ak by bolo nutné aktualizovať webové stránky na zariadení, ktoré sú uložené mimo firmwaru, tak by to bez tohto rozšírenia nebolo možné. Pričom všetko ostatné zostáva rovnaké, len na strane zariadenia je naviac implementovaná časť s týmto rozšírením. Pri dešifrovaní firmwaru sa využíva knižnica `mbedTLS`.

Zabezpečenie koncového zariadenia

Ide o náročný proces, keďže je nutné vytvoriť a spravovať viacero kľúčov asymetrickej a symetrickej kryptografie, ktoré sú vysvetlené v 3.3. S cieľom ochránenia zariadenia pred rôznymi útokmi (príklady útokov sú popísané v 3.8), je kladený veľký dôraz na ich zabezpečenie. Zariadenia majú vytvorený špeciálny šifrovaný oddiel v rámci pamäte flash, kde sú uložene potrebné kľúče a certifikáty.

Aktualizácie zariadenia – každé zariadenie obsahuje rovnaký certifikát aktualizačného serveru, ktorý slúži na vytvorenie zabezpečeného spojenia. Súčasne každé zariadenie môže mať jedinečný alebo spoločný privátny kľúč na odšifrovanie prenášaného nového oddielu. Ak by všetky zariadenia zdieľali jeden privátny kľúč, ide o riziko, že ak by čo i len z jedného zariadenia unikol, útočník môže následne ovplyvniť aj ostatné zariadenia. Avšak keďže tieto súbory sú prenášané cez už zabezpečený kanál, sa toto riziko zmenšuje, taktiež sa zmenšuje vďaka využitiu bezpečného zavedenia kódu. Výhodou je fakt, že je nutné uchovať len jeden verejný kľúč na strane serveru, inak je nutné spravovať pre každé zariadenie jedinečný. Obe možnosti sú k dispozícii, je už na vývojárovi, ktorú z možností si zvolí.

Bezpečné zavedenie kódu – každému zariadeniu pred tým, ako idú do ostrej prevádzky, je vypálený do jednorázových registrov verejný kľúč. Tento verejný kľúč má v správe vývojár/výrobca, a k nemu odpovedajúci privátny kľúč. Ide o asymetrickú kryptografiu, a tento kľúčový pár zabezpečuje nepopierateľnú identifikáciu pôvodu firmwaru. Po vypálení týchto registrov a po prvom úspešnom spustení firmwaru na zariadení,

zariadenie spúšťa ochranu zavedenia kódu. Túto ochranu vykonáva priamo bootloader, pričom práve vďaka verejnému kľúču overuje digitálny podpis firmwaru. Preto vývojár musí pred aktualizáciou zariadenia daný firmware najskôr podpísať pomocou svojho privátneho kľúču a až následne môže zariadenie aktualizovať.

Šifrovanie pamäte flash – pamäť flash sa šifruje pomocou symetrickej kryptografie, takže je potrebný len jeden tajný kľúč. Sú dva spôsoby ako tieto kľúče uchovať.
Jednou z možností je túto funkcionalitu na zariadení zapnúť a generovanie a uloženie kľúču nechať v správe samotného zariadenia. Takto zašifrovaná pamäť flash je navždy šifrovaná a nedá sa nijako fyzicky aktualizovať. Jedinou možnosťou aktualizácie je cez vzdialenú aktualizáciu. Ak nastane situácia, že zariadenie nemá implementovanú funkcionalitu vzdialenej aktualizácie, tak je už navždy uzamknuté, a bude na ňom bežať len ten firmware, ktorý tam bol vypálený pred zapnutím šifrovania pamäte flash.
Druhou možnosťou je mať správu kľúčov a ich generovanie vo vlastnej réžii. V podstate to znamená vygenerovať tajný kľúč, využíva sa algoritmus AES-128 alebo AES-256, a tento kľúč vypáliť do jednorázových registrov. Ak by bolo v budúcnosti potrebné zariadenie fyzicky aktualizovať, je nutné pred vypálením firmwaru do zariadenia tento firmware zašifrovať pomocou príslušného algoritmu a tajného kľúču, a až potom vypáliť firmware. Umožňuje to aktualizovať zariadenie aj bez nutnosti implementácie vzdialenej aktualizácie.

Mód sťahovania cez UART – do zariadení sa všetky potrebné dáta ako firmware, tabuľka oddielov, webové stránky atď., nahrávajú cez UART. Na zariadeniach ESP32–S3 je možnosť zvoliť rôzne módy sťahovania cez UART. Pred odoslaním zariadenia do prevádzky je odporúčané tento mód zvoliť na zabezpečený alebo dokonca úplne vypnúť. Ak je vyžadované zariadenie ešte v budúcnosti fyzicky aktualizovať, je vhodné zvoliť len zabezpečený mód, avšak pri aktualizácii je nutné dodržať všetky ostatné bezpečnostné postupy, ktoré boli zvolené v rámci zabezpečenia zariadenia. Tento mód nijakým spôsobom neovplyvňuje vzdialenú aktualizáciu. V rámci tejto práce sa využíva zabezpečený mód.

Proces zabezpečenia zariadenia je relatívne komplikovaný a nebezpečný, keďže v prípade nedodržania presných pokynov sa môže zariadenie dostať do nepoužiteľného stavu. Pri implementácii tejto práce bolo viacero zariadení zničených, preto bude v jednoduchosti popísaný postup zabezpečenia, a teda ako pripraviť zariadenia na nasadenie do prevádzky.

Na generovanie kľúčov a následne nahrávanie do zariadenia boli využité dodávané nástroje od výrobcu, teda od spoločnosti Espressif. Predpokladá sa s nikdy nepoužitým zariadením:

1. Vygenerovanie kľúča symetrickej kryptografie na šifrovanie pamäte flash.
2. Vypálenie kľúča z predošlého bodu do registrov zariadenia
3. Vygenerovanie kľúča symetrickej kryptografie na šifrovanie oddielu pamäte v ktorom sa nachádzajú citlivé dáta, napr. kľúče.
4. Vypálenie kľúča z predošlého bodu do oddielu na to určeného, v rámci tabuľky oddielov sa nazýva `nvs_keys`.
5. Podpísať bootloader privátnym kľúčom určeným na bezpečné zavedenie kódu.

6. Tento bootloader nahrať do zariadenia. Od tohto bodu musí byť firmware taktiež podpísaný.

7. Nahrať všetky ostatné dáta.

Tento postup bol vymyslený a navrhnutý v rámci implementácie práce, a využíva takmer všetky možnosti zabezpečenia zariadenia, ktoré výrobca ponúka. Oddiel, ktorý obsahuje citlivé dáta z bodu 4., slúži na uchovanie kľúčov použitých pri šifrovaní oddielu s názvom `nvs`. Ide o pamäť nevyžadujúcu napájanie a dáta sa v nej ukladajú štýlom kľúč hodnota. Práve v tejto pamäti sú uložené všetky ostatné kľúče, využité následne vo firmwari, napr. pripojenie na aktualizačný server, odšifrovanie nového firmwaru, pripojenie na brokera. Tieto kľúče najmä využíva práve koreňové zariadenie, ktoré sa vďaka nim pripojí na brokera alebo aktualizačný server.

6.4 Testovanie ekosystému bezdrôtovej siete a navrhované vylepšenia

V rámci tejto práce boli navrhnuté testy ekosystému, ktoré by mali byť na tomto ekosystéme ale aj na podobných ekosystémoch vykonané. Keďže sa jedná o relatívne veľký záber technológii, a v podstate sa práca zamerala od hardwaru na fyzických zariadeniach, až po služby bežiace na serveri, je nutné pokryť ako hardware, tak aj software. Testy vhodné pre tento ekosystém môžu vyzerať nasledovne:

- **Test stability bezdrôtovej siete** – celá sieť by mala byť stabilná a vďaka využitým technológiám by mala stabilný stav udržiavať nepretržite.
- **Test stability zariadenia** – distribuovaný firmware do zariadení musí byť stabilný, aby nedochádzalo k zbytočným výpadkom v rámci siete, keďže to môže spomaliť fungovanie celej siete, kvôli zbytočnej réžii pri opakovanom organizovaní siete.
- **Test stability celého ekosystému** – celý ekosystém by mal zvládať väčšie počty užívateľov bez problémov a udržiavať sa v stabilnom stave.
- **Test poveternostnej odolnosti** – zariadenia sú konštruované na umiestnenie do boxu, ktorý môže isť do náročných podmienok. Zariadenia sa nesmú prehrievať, musia zvládať pracovať aj v nižších teplotách a taktiež vo vyšších vlhkostiach. Taktiež je nutné otestovať, kde sa nachádza rosný bod takejto krabičky so zariadením.
- **Test elektromagnetickej kompatibility** – všetky elektronické zariadenia musia spĺňať test elektromagnetickej kompatibility. Je zameraný najmä na vysielanie nežiadúceho rušenia v okolí zariadenia.
- **Test kontinuálneho fungovania siete pri výpadku serveru** – pri výpadku serveru je nutné, aby celá sieť pokračovala vo fungovaní, keďže pri výpadku v nevhodný moment môžu zariadenia spôsobiť škodu na majetku.
- **Test užívateľského prostredia webových stránok na zariadeniach** – stránky na zariadeniach by mali byť užívateľsky jednoduché a prívetivé.
- **Test užívateľského prostredia webovej stránky na servere** – priehľadový web by mal byť jednoduchý a užívateľsky prívetivý.

- **Test funkčnosti aktualizácie zariadenia** – zariadenia by malo byť možné aktualizovať, keďže po nasadení do prevádzky je to jediný spôsob nápravy prípadných chýb.
- **Test duplicity identifikátoru zariadenia** – zariadenia musia prejsť týmto testom, aby nenastala situácia, že dve rozdielne zariadenia budú vystupovať pod rovnakým identifikátorom.
- **Test vizualizácie aktuálneho stavu zariadenia** – zariadenie by malo disponovať možnosťou svoj stav vizuálne zobrazovať, aby užívateľ pomocou pohľadu na zariadenie vedel, v akom je aktuálne stave.
- **Test zabezpečenia zariadenia** – zariadenie, ktoré ide do prevádzky musí byť zabezpečené.
- **Test spotreby elektrickej energie** – zmerať spotrebu elektrickej energie zariadení.

Niektoré z testov nie je možné v rámci tejto práce vykonať, alebo boli vykonané len čiastočne. To kvôli časovej náročnosti (mesiace), finančnej náročnosti, veľkého počtu potrebných zariadení (vyššie stovky až nižšie tisíce) a nároky na počet užívateľov (nižšie stovky). Vykonané testy s ich popisom sú:

- **Vykonanie testu stability bezdrôtovej siete** – test bol vykonaný na sieti o veľkosti troch zariadení. Zariadenia boli rôzne vypínané. V každom z prípadov sa sieť obnovila do funkčného stavu. Jediným postrehom je, že v prípade ak sú zariadenia vedľa seba položené, občas sa medzi zariadeniami vymení, kto je koreňové zariadenie. Tento výsledok je v poriadku a úspešný, keďže sa nepredpokladá, že zariadenia budú hneď vedľa seba.
- **Vykonanie testu stability zariadenia** – test bol vykonaný na zariadení, ktoré bolo spustené 2 týždne v kuse. Žiaden výpadok nebol zaznamenaný, stav či zariadenie bolo pripojené sa overovalo posielaním správ na server. Tento výsledok potvrdila aj štatistika z MQTT brokera. Zariadenie bolo pripojené na brokera s nastaveným parametrom `keep alive` (prelož. udržať na živе) na 5 sekúnd, to predstavuje interval medzi paketmi prijatými na brokerovi počas ktorého broker považuje zariadenie za online.
- **Vykonanie testu kontinuálneho fungovania siete pri výpadku serveru** – zariadeniu bolo vytvorené pravidlo na rozsvietenie LED na základe teploty iného zariadenia v sieti. Následne bol vypnutý server. Sieť pokračovala aj naďalej v plnej funkčnosti.
- **Vykonanie testu funkčnosti aktualizácie zariadenia** – zariadenie pripravené do prevádzky bolo aktualizované pomocou vzdialenej aktualizácie. Spustenie aktualizácie prebehlo pomocou MQTT správy. Taktiež bolo simulované prerušenie spojenia počas aktualizácie. Zariadenie sa zachovalo podľa očakávania a neuskutočnilo výmenu nového firmwaru, ale spustilo starý funkčný firmware.
- **Vykonanie testu duplicity identifikátoru zariadenia** – test zistenia duplicity bol vykonaný pomocou jednoduchého dotazu na REST API, ktoré sa pokúsilo získať záznam s týmto identifikátorom z databázy. Tento test je dôležitý aj ak by sa MAC adresy používali ako jedinečné identifikátory keďže môže dôjsť ku kolízii.

- **Vykonanie testov užívateľských rozhraní** – tieto testy vyžadujú väčšiu vzorku ľudí, avšak test bol vykonaný na 10 ľudoch vo veku 16–57 rokov. Na základe pripomienok bola najmä inicializačná stránka zariadenia upravená. Pridala sa možnosť vypísania všetkých sietí Wi-Fi v okolí. V rámci prehľadového webu na servery bola pridaná možnosť pridania zariadenia pomocou QR kódu. Najmä starší užívatelia z tejto vzorky by ocenili lokalizáciu do češtiny alebo slovenčiny.

- **Vykonanie testu vizualizácie aktuálneho stavu zariadenia** – zariadenie, na ktorom bola implementovaná táto práca, disponuje dvomi LED a teda môžu sa rôzne využiť na zobrazovanie stavu. Aktuálne jedna slúži na zistenie, či sa zariadenie úspešne pripojilo na Wi-Fi a druhá, či sa zariadenie úspešne pripojilo na server. Obe LED fungovali a nevykazovali žiaden problém. Z užívateľského pohľadu by bolo vhodné aby LED nemali rovnakú farbu, ale aby boli farebne rozdielne.

- **Vykonanie testu zabezpečenia zariadenia** – zo zariadenia bola vyčítaná časť flash pamäte, pričom následne bola táto časť porovnaná s pôvodným firmwarom pred zašifrovaním a neboli zhodné. Súčasne do zariadenia bol nahratý nezašifrovaný firmware, tento firmware zariadenie neprijalo. Taktiež bol do zariadenia nahratý nepodpísaný zašifrovaný software, tento taktiež odmietlo.

- **Vykonanie testu spotreby elektrickej energie** – zariadenie bolo pripojené na laboratórny zdroj. Koncové zariadenie malo priemernú spotrebu 400mAh a koreňové približne 450mAh. Neboli využité žiadne spôsoby na šetrenie elektrickej energie a teda zariadenia boli neustále spustené. Z tohto testu vyplýva, že zariadenia nie sú v takomto stave vhodné na napájanie z batérie.

Tieto testy boli zamerané aj na validáciu pripravenosti zariadenia na prevádzku v reálnych podmienkach. V rámci vykonávania testov boli odhalené nedostatky najmä na užívateľskom prostredí.

Navrhované vylepšenia ekosystému

Ide o relatívne veľký systém, ktorý ešte obsahuje určité nedostatky, ktoré sú zväčša svojou podstatou netriviálne na implementáciu. Ako prvý nedostatok je možnosť aktualizovať len koreňové zariadenie. Tu je určite možnosť na vylepšenie o aktualizáciu akéhokoľvek zariadenia v sieti. Ďalším možným vylepšením je celý proof–of–concept tabuliek správania, keďže aktuálne sa dajú získať len zo serveru, a sú uložené len na koreňovom zariadení. Taktiež editor pre vytváranie pravidiel pre jednotlivé zariadenia by mohol byť vhodné vylepšenie, avšak tu je otázne či skôr nenapojiť tento systém na existujúci projekt, napr. Home Assistant. Veľkou výzvou na vylepšenie je spotreba elektrickej energie, aby zariadenia boli vhodné aj na napájanie z batérie. Taktiež zaujímavým vylepšením môže byť integrácia do projektu Matter.

Kapitola 7

Záver

Cieľom tejto práce bolo zhodnotenie aktuálnych riešení v oblasti IoT a vytvorenie architektúry, návrhu a implementácie bezdrôtovej senzorovej Wi-Fi siete, so zameraním na modifikovateľné zariadenia. Taktiež túto sieť pripojiť na server. Tento cieľ bol splnený.

Na splnenie cieľov tejto práce bolo najskôr nutné si naštudovať rôzne technológie a zoznámiť sa s ich možnosťami. Tieto technológie boli zamerané najmä na IoT, ale aj na serverové služby a hardware IoT zariadení. Taktiež bolo nutné naštudovať state–of–the–art trhu IoT. Následne som vytvoril prehľad o aktuálnom stave riešení v rámci IoT a z tohto som vytipoval vhodné vlastnosti pre bezdrôtovú senzorovú sieť. V rámci rozšírenia možností tejto senzorovej siete som túto sieť pripojil na server, pričom architektúru serveru som navrhol so zameraním na IoT. Túto architektúru som vytvoril na základe aktuálnych dostupných riešení v tejto oblasti. Následne som celý tento ekosystém implementoval s dôrazom na zabezpečenie jednotlivých častí. Výslednú implementáciu som následne otestoval a zhodnotil možné vhodné vylepšenia.

Výsledkom práce je bezdrôtová senzorová sieť postavená na Wi-Fi. Taktiež boli implementované nové funkcie umožňujúce nezávislosť celej siete na servery. Vďaka využitiu vhodných technológii je navrhnutá sieť aj gatewayless (prelož. bezbránová). Súčasne sa uchovávajú historické dáta na servery a môžu sa zobrazovať na webovej stránke. Z tejto stránky je možné zariadenia taktiež ovládať. Práca bola prezentovaná na konferencii Excel@FIT. Základ práce je už použitý a nasadený v komerčnom sektore, pričom sa predpokladá, že sa výsledok tejto práce taktiež využije.

Najväčšou a najzložitejšou časťou práce bolo zabezpečenie a návrh architektúry siete. Na druhú stranu implementácia prebiehala v podstate bez problémov.

V budúcnosti sa plánujem aj naďalej venovať tomuto odvetviu, najmä vylepšeniam tejto práce. Taktiež preskúmaniu projektov, ktoré spájajú fragmentovaný IoT trh, napr. Matter, Home Assistant, a prípadnú integráciu do takýchto systémov. V neposlednom rade optimalizovať spotrebu elektrickej energie zariadení, súčasne sa pozrieť na využitie princípu energy harvesting. Dlhodobo sa zamerať skôr na špecializovanú časť trhu, o ktorú veľké firmy nemajú z hľadiska veľkosti až taký záujem. Takýto trh zahrňuje zariadenia ako fotovoltaické elektrárne, elektrické brány, technológie pre vodohospodárstvo a zabezpečené zariadenia určené pre priemysel a kritickú infraštruktúru.

Literatúra

[1] Abdellatif, K. M., Hériveaux, O. a Thillard, A. *Unlimited Results: Breaking Firmware Encryption of ESP32-V3* [Cryptology ePrint Archive, Paper 2023/090]. 2023. https://eprint.iacr.org/2023/090. Dostupné z: https://eprint.iacr.org/2023/090.

[2] Al Ali, A., Zualkernan, I. A., Rashid, M., Gupta, R. a Alikarar, M. A smart home energy management system using IoT and big data analytics approach. *IEEE Transactions on Consumer Electronics*. 2017, zv. 63, č. 4, s. 426–434. DOI: 10.1109/TCE.2017.015014.

[3] Al Sarawi, S., Anbar, M., Abdullah, R. a Al Hawari, A. B. Internet of Things Market Analysis Forecasts, 2020–2030. In: *2020 Fourth World Conference on Smart Trends in Systems, Security and Sustainability (WorldS4)*. 2020, s. 449–453. DOI: 10.1109/WorldS450073.2020.9210375.

[4] Bayer, M. SQLAlchemy. In: Brown, A. a Wilson, G., ed. *The Architecture of Open Source Applications Volume II: Structure, Scale, and a Few More Fearless Hacks*. Aosabook.org, 2012. Dostupné z: http://aosabook.org/en/sqlalchemy.html.

[5] Bender, M., Kirdan, E., Pahl, M.-O. a Carle, G. Open-Source MQTT Evaluation. In: *2021 IEEE 18th Annual Consumer Communications & Networking Conference (CCNC)*. 2021, s. 1–4. DOI: 10.1109/CCNC49032.2021.9369499.

[6] Butun, I., Österberg, P. a Song, H. Security of the Internet of Things: Vulnerabilities, Attacks, and Countermeasures. *IEEE Communications Surveys & Tutorials*. Firstquarter 2020, zv. 22, č. 1, s. 616–644, [cit. 12.4.2023]. DOI: 10.1109/COMST.2019.2953364. ISSN 1553-877X.

[7] Chen, P. a Yang, Z. Understanding Precision Time Protocol in Today's Wi-Fi Networks: A Measurement Study. In: *2021 USENIX Annual Technical Conference (USENIX ATC 21)*. USENIX Association, Júl 2021, s. 597–610. ISBN 978-1-939133-23-6. Dostupné z: https://www.usenix.org/conference/atc21/presentation/chen.

[8] Chi, X., Liu, B., Niu, Q. a Wu, Q. Web Load Balance and Cache Optimization Design Based Nginx under High-Concurrency Environment. In: *2012 Third International Conference on Digital Manufacturing & Automation*. July 2012, s. 1029–1032. DOI: 10.1109/ICDMA.2012.241.

[9] Dragomir, D., Gheorghe, L., Costea, S. a Radovici, A. A Survey on Secure Communication Protocols for IoT Systems. In: *2016 International Workshop on Secure Internet of Things (SIoT)*. Sep. 2016, s. 47–62. DOI: 10.1109/SIoT.2016.012.

[10] Felter, W., Ferreira, A., Rajamony, R. a Rubio, J. An updated performance comparison of virtual machines and Linux containers. In: *2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. 2015, s. 171–172. DOI: 10.1109/ISPASS.2015.7095802.

[11] Gatial, E., Balogh, Z. a Hluchý, L. Concept of Energy Efficient ESP32 Chip for Industrial Wireless Sensor Network. In: *2020 IEEE 24th International Conference on Intelligent Engineering Systems (INES)*. 2020, s. 179–184. DOI: 10.1109/INES49302.2020.9147189.

[12] Han, D.-m. a Lim, J.-h. Smart home energy management system using IEEE 802.15.4 and zigbee. *IEEE Transactions on Consumer Electronics*. 2010, zv. 56, č. 3, s. 1403–1410. DOI: 10.1109/TCE.2010.5606276.

[13] Harkins, D. Simultaneous Authentication of Equals: A Secure, Password-Based Key Exchange for Mesh Networks. In: *2008 Second International Conference on Sensor Technologies and Applications (sensorcomm 2008)*. 2008, s. 839–844. DOI: 10.1109/SENSORCOMM.2008.131.

[14] Kamalinejad, P., Mahapatra, C., Sheng, Z., Mirabbasi, S., M. Leung, V. C. et al. Wireless energy harvesting for the Internet of Things. *IEEE Communications Magazine*. 2015, zv. 53, č. 6, s. 102–108. DOI: 10.1109/MCOM.2015.7120024.

[15] Panda, M. Performance analysis of encryption algorithms for security. In: *2016 International Conference on Signal Processing, Communication, Power and Embedded System (SCOPES)*. Oct 2016, s. 278–284. DOI: 10.1109/SCOPES.2016.7955835.

[16] Ren, L., Luo, Y., Lu, G., Cong, M., Wang, X. et al. A bidirectional and low-frequency energy harvester for collecting human crowd energy in shopping malls. *Energy Conversion and Management*. 2022, zv. 252, s. 115046. DOI: https://doi.org/10.1016/j.enconman.2021.115046. ISSN 0196-8904. Dostupné z: https://www.sciencedirect.com/science/article/pii/S019689042101222X.

[17] Suresh, P., Daniel, J. V., Parthasarathy, V. a Aswathy, R. H. A state of the art review on the Internet of Things (IoT) history, technology and fields of deployment. In: *2014 International Conference on Science Engineering and Management Research (ICSEMR)*. 2014, s. 1–8. DOI: 10.1109/ICSEMR.2014.7043637.

[18] Thread Group. *Thread commissioning white paper* [online; Navštívené: 20.4.2023]. July 2015. Dostupné z: https://www.threadgroup.org/Portals/0/documents/support/CommissioningWhitePaper_658_2.pdf.

[19] Yang, Y., Wu, L., Yin, G., Li, L. a Zhao, H. A Survey on Security and Privacy Issues in Internet-of-Things. *IEEE Internet of Things Journal*. 2017, zv. 4, č. 5, s. 1250–1258. DOI: 10.1109/JIOT.2017.2694844.

Príloha A

Obsah priloženého pamäťového média

Priložené pamäťové médium sa skladá z adresárovej štruktúry:

- /Device – obsahuje všetky súbory spojené s implementáciou zariadenia
- /Server – obsahuje všetky súbory spojené s implementáciou serverovej časti
- /Docs – obsahuje zdrojové súbory Latexu pre text tejto práce