Univerzita Hradec Králové

Fakulta informatiky a managementu

Katedra informačních technologií

Návrh zabezpečené sítě IoT zařízení pro SMART implementace

Bakalářská práce

Autor: Jan Hoffmann

Studijní obor: Aplikovaná informatika

Vedoucí práce: Ing. Pavel Blažek, Ph.D.

Hradec Králové 2023

Prohlášení:

Prohlašuji, že jsem bakalářskou práci zpracoval samostatně a s použitím uvedené literatury.

V Hradci Králové dne 27.4.2023 Jan Hoffmann

Poděkování:

Děkuji Mgr. Josefu Horálkovi, Ph.D. a Ing. Pavlu Blažkovi, Ph.D. za metodické vedení práce a za rady, které mi během práce poskytli. Zároveň děkuji za pomoc Ing. Patriku Urbaníkovi, který mi poskytl potřebná zařízení pro praktickou implementaci.

Anotace

Bakalářská práce se věnuje návrhu zabezpečené sítě IoT zařízení, a to především za pomoci jednodeskových počítačů Raspberry Pi, Arduino a vývojových desek ESP. Teoretická část práce se zabývá problematikou IoT (Internet of Things), specifickými vlastnostmi, kterými se IoT vyznačuje, a rozboru jednotlivých komunikačních protokolů, díky kterým, mohou jednotlivá zařízení komunikovat. Součástí teoretické části je také podrobnější popis použitého komunikačního protokolu MQ Telemetry Transport (MQTT). Praktická část bakalářské práce je zaměřena na realizaci dané struktury IoT sítě, obsahující senzory, aktuátory a ústřední komunikační počítač. Zřetel je brán na stabilní připojení jednotlivých zařízení, a hlavně na zabezpečení prvků zapojených v dané síti. Výsledkem práce je zabezpečená síť IoT zařízení.

Annotation

Title: Design of secure IoT device network for SMART implementation

The Bachelor Thesis is devoted to the design of a secure IoT network using single-board computers such as Raspberry Pi, Arduino, and ESP. The theoretical part of the bachelor thesis deals with IoT (Internet of Things) and specific features that characterize IoT. It also covers the analysis of each communication protocol used by devices to communicate. Part of the theoretical section is, as well, a more detailed description of the communication protocol MQ Telemetry Transport (MQTT). The practical segment of the Bachelor Thesis focuses on the realization of the IoT network structure that consists of sensors, actuators, and a central hub computer. Attention is paid to a stable connection of each device, mainly to the security of elements connected to the network. The result of the thesis is a steady, secured IoT network.

Obsah

Seznam obrázků

Seznam tabulek

Seznam rovnic

1 Úvod

Výrazným znakem pro dnešní společnost je zjednodušení či zefektivnění procesů. Spoustu rutinních prací a úkonů dnes zastávají roboti (chytrá zařízení), jenž firmám šetří peníze a čas. Původně byla zařízení tohoto typu umístěna a využívána pouze určitými společnostmi. V dnešní době se však spousta chytrých (Smart) zařízení dostává mezi širší okruh společnosti. Již není vzácností ve spoustě domácností narazit na takováto chytrá zařízení. Jmenovitě chytré žárovky, chytré spotřebiče, nebo dokonce komplexní chytré systémy. Popularita takzvaných chytrých domácností neustále narůstá.

Chytré spotřebiče mají mnoho výhod, které z nich dělají užitečný doplněk do domácnosti. Avšak s výhodami přichází i několik podstatných nevýhod, které mohou odradit nejednoho potencionálního zájemce. Tyto nevýhody se týkají především bezpečnosti. Zařízení, která domácnosti využívají, mají neustálý přístup k síti a neustále komunikují s různými servery. To znamená, že jsou viditelná pro okolní svět. Když se k tomu připočte nedostatečné zabezpečení domácí sítě – slabá hesla či nízká úroveň zabezpečení, stávají se chytrá zařízení snadným cílem hackerských útoků. Mezi hojně používaná hobby zařízení v IoT se řadí například Arduino nebo jednodeskové počítače značky Raspberry Pi.

Téma této práce – Návrh zabezpečené IoT sítě pro SMART implementace – nastiňuje základní problematiku IoT (Internet of Things) konceptu. Popisuje protokoly, které se běžně využívají pro komunikace mezi zařízeními používanými v chytrých domácnostech, a speciálně se zabývá protokolem MQTT. Datový protokol MQTT je zlehčeným a nenáročným protokolem, který slouží k předávání zpráv mezi jednotlivými zařízeními v síti. Díky své jednoduchosti a nenáročnosti ho mohou využívat zařízení, která obsahují „jednoduchý“ procesor. Protokol MQTT však není jediný protokol používaný ve světě IoT. Mezi další rozšířené datové protokoly patří například AMQP (Advanced Message Queuing Protocol) nebo velice známý HTTP protokol, který slouží k přenosu hypertextových dokumentů. Protokol HTTP ale není pro svět IoT plně přizpůsobený, a to například kvůli one-to-one komunikaci či kvůli limitaci na synchronní požadavky.

Práce je rozdělena do dvou částí – teoretická část a praktická část. Teoretická část se zabývá základní definicí IoT (Internet of Things), specifikováním základních vlastností IoT a definováním protokolů používaných v odvětví IoT. Kromě protokolu MQTT práce specifikuje i ostatní protokoly, které lze v IoT využít.

Praktická část se zabývá vytvořením testovací / demo sítě tvořené z řídicí jednotky v podobě Raspberry Pi, aktuátorů, senzorů, vývojových desek ESP a zařízení s Kali Linux, pomocí kterého budou otestována případná bezpečnostní rizika.

2 Cíl práce

Cílem práce je specifikace problematiky Internetu věcí, se zaměřením na komunikační protokol MQTT v porovnání s ostatními protokoly využívanými v IoT. Dalším cílem je návrh chytré implementace pomocí Raspberry Pi, mikrokontrolerů ESP8266 a vývojového prostředí Node-RED. Během tvorby demo sítě jsou představeny možnosti, které takováto demo situace nabízí. Z návrhu vyplynou určitá data, na jejichž bázi se může čtenář této práce rozhodnout, zda je vhodné nasazení chytré domácnosti do jeho infrastruktury.

3 Metodika zpracování

V této práci je použita metoda rešerše literatury, analýzy technologií a implementace praktického příkladu. Rešerše je zaměřena na teoretické základy problematiky IoT. Analýza technologií si klade za cíl specifikaci a porovnání protokolů používaných v IoT. Praktická implementace je zaměřena na tvorbu demo sítě, skládající se z jednodeskového počítače Raspberry Pi, senzorů (vlhkosti, teploty, …), aktuátorů a vývojových desek ESP. Tato jednoduchá demo síť představuje onu chytrou domácnost. Zařízení s Kali Linux bude využito jako nástroj, pomocí kterého bude síť jednoduše oskenována na případná bezpečnostní rizika. Implementace sítě povede k určitým poznatkům o bezpečnosti a stabilitě chytrých domácností / IoT řešení.

4 Internet věcí

4.1 Vymezení pojmu

Internet věcí (anglicky Internet of Things) je velice zjednodušeně soubor fyzických zařízení, zapojených do jedné sítě za účelem vzájemné výměny dat. Mezi zařízení tvořící tuto síť spadají například domácí spotřebiče, senzory, čidla, různorodá elektronická zařízení, či například vozidla. Hlavním znakem těchto zařízení je síťová konektivita, která jim umožňuje komunikovat s ostatními zařízeními v síti.

Internet věcí může být definován jako „propojení věcí" sloužící k vnímání informací reálného světa. Aplikace a použití internetu se rozšiřuje do běžného života a IoT je nový přístup zakomponování internetu do osobního, profesionálního a sociálního života. IoT lze vnímat jako propojený set kohokoliv, čehokoliv, kdykoliv a kdekoliv. (…) Termín „Internet of Things" nebo „Internet of Objects" reprezentuje různorodé zařízení s různými velikostmi a schopnostmi, která jsou připojená na internet (Hussain, 2017).

To, jak rozšířený „Internet věcí" v dnešní době je, odráží také autoři Wortmann, F. a Flüchter, K. ve své práci Internet of Things (2015). Během posledních několika měsíců bylo téměř nemožné nenarazit na termín „Internet věcí" (IoT). Především uplynulý rok zaznamenal obrovský nárůst zájmu o Internet of Things (Wortmann & Flüchter, 2015).

4.2 Vznik a vývoj

Zatímco je pojem Internet věcí v dnešní době čím dál více používán, méně známá je jeho historie. Internet věcí nevznikl z ničeho nic a rozhodně ve svých počátcích nevypadal tak, jak vypadá nyní. Termín Internet věcí je již 16 let starý. Avšak myšlenka propojených zařízení tu byla mnohem déle, alespoň od sedmdesátých let devatenáctého století (Lueth, 2015). Většina publikací se však shoduje v následujícím. Termín „Internet věcí" nezávisle razil Kevin Ashton, zaměstnanec společnosti Procter & Gamble, později MIT, v roce 1999. Dokonce i samotný Kevin Ashton se k tomuto tématu vyjadřuje následovně: Mohu se mýlit, ale jsem si poměrně jistý, že fráze „Internet of Things" započala svoji existenci jako titulek

prezentace, kterou jsem vytvořil v Procter & Gamble (P&G) v roce 1999[1] (překlad autora) (Ashton, 2009).

Stejně tak jako u mnoha nových konceptů lze počátky IoT nalézt zpět v Massachusettskému technologickému institutu (MIT), konkrétně z prací prováděných centrem Auto-ID. Tato skupina pracovala v oboru síťové radiofrekvenční identifikace (RFID) (M. A. Ezechina et al., 2015). Hlavní tématikou Internetu věcí, je zapojit mobilní vysílače krátkého dosahu do každodenního života a umožnit tak vznik novým formám komunikace mezi lidmi a věcmi (Stallings et al., 2016).

S vývojem Internetu věcí souvisí také jedna zásadní otázka. Je pro lidstvo IoT důležité? V práci The Internet of Things (IoT): A Scalable Approach to Connecting Everything. Napsané autory M. A. Ezechina, K. K. Okwara, C. A. U. Ugboaja, je tato otázka zajímavě vysvětlena. IoT je kritické pro vývoj lidstva: Vzhledem k tomu, že se populace planety rozrůstá, je čím dál tím více důležité, aby lidé dokázali co nejlépe spravovat zemi a zdroje, které nabízí. Navíc, lidé touží žít zdravé, naplňující a komfortní životy. A to samé přejí svým blízkým a rodinám. Pokud tedy zkombinujeme schopnosti IoT – vnímat, sbírat informace, analyzovat a distribuovat data v obrovském měřítku, s lidským vnímáním informací, lidstvo bude mít dostatečné množství znalostí, díky kterým nejen přežije, ale bude v následujících měsících, letech, či stoletích rozkvétat (M. A. Ezechina et al., 2015).

Základní princip důležitosti vzájemné komunikace mezi lidmi popisuje také úryvek zmíněný při předávání Nobelovi ceny v roce 2014. Lidstvo se vyvíjí, jelikož komunikuje. Když byl poprvé objeven oheň a sdílen mezi ostatními jedinci, nemuseli ho tito jedinci znovuobjevovat, nýbrž jim byla tato schopnost předána právě pomocí komunikace. Lepší, modernější příklad, je objevení helix struktury DNA molekul, které nesou genetickou informaci z jedné generace na druhou. Po tom, co byl článek

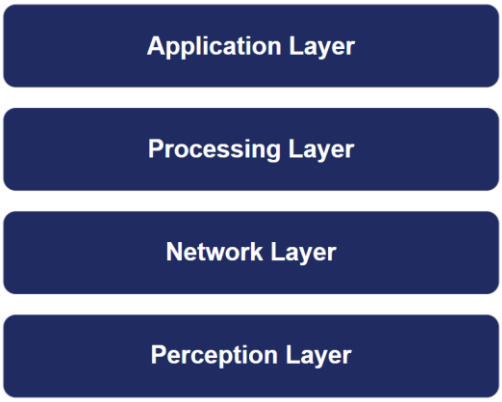
[1] *„I could be wrong, but I'm fairly sure the phrase "Internet of Things" started life as the title of a presentation I made at Procter & Gamble (P&G) in 1999.“* (Ashton, 2009)

o objevení helix struktury DNA publikován ve vědeckých novinách Jamesem Watsonem a Francisem Crickem v dubnu 1953, byla ho genetická medicína schopna využít jako odrazový můstek pro další obrovské (M. A. Ezechina et al., 2015).

4.3 Základní architektura IoT

Svět očekává, že do roku 2020 bude navzájem propojeno více než 25 miliard věcí, což je obrovské číslo. Je tedy třeba, pro tento rapidně se rozrůstající segment propojených objektů, najít vhodné, flexibilní a přizpůsobivé architektury různého typu a různého rámce využití. Není zde žádný „správný" konsenzus IoT architektury, který by byl obecně využíván. Existuje však velké množství různých architektur, navržených různými výzkumnými ústavy. Zatím však žádná, z těchto mnoha architektur nebyla vybrána jako referenční (Gemirter et al., 2021).

Jak je zmíněno v citaci výše, základní architekturu IoT nelze jednoznačně specifikovat, jelikož existuje více různých druhů architektur, které byly v poslední době navrženy. Jsou zde ovšem aspekty, na kterých se tyto architektury shodují. Z těchto společných prvků lze vyvodit základní architekturu IoT. Tato základní architektura se skládá ze čtyř vrstev. První vrstvou je vrstva aplikační (anglicky application layer). Dále se jedná o vrstvu procesní (processing layer), vrstvu síťovou (network layer) a nakonec vrstvu vjemovou (perception layer). Každá z těchto vrstev se vyznačuje specifickými vlastnostmi a specifickými komponenty.



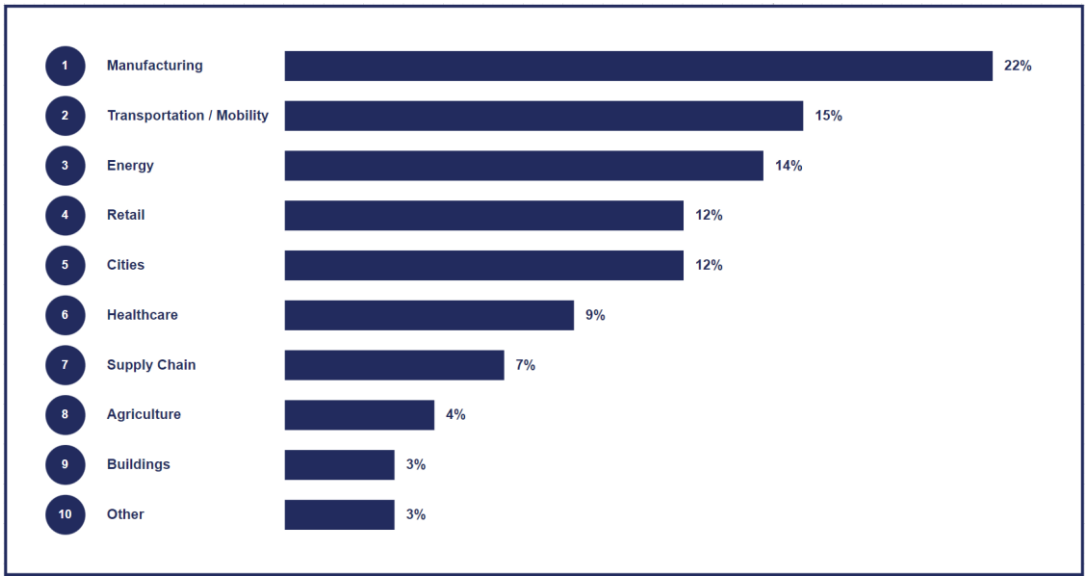
Obrázek 1 - Architektura čtyř vrstev
Zdroj: (autor, dle předlohy Scaler Academy, 2022)

Takzvaná architektura čtyř vrstev není jediná architektura, která IoT definuje. Jsou zde ještě další dvě architektury, které vznikly rozpadem vrstev architektury čtyřvrstvé. První z nich je architektura pěti vrstev. Tato architektura vznikla derivací takzvané business vrstvy z vrstvy aplikační. Do business vrstvy spadají uživatelé, kteří IoT systém využívají a také business procesy.

Druhá, šestivrstvá architektura, vznikla rozpadem procesní vrstvy a převzetím business vrstvy z architektury pětivrstvé. Zmíněná procesní vrstva se rozpadla na dvě datové podvrstvy, které spolu úzce souvisí. Datové proto, že procesní vrstva je také často nazývána jako datová procesní, či jenom datová. První z těchto vrstev je vrstva datová abstraktní, která se stará o agregaci dat a poskytuje přístup k datům. Druhou vrstvou je vrstva datová akumulační. Ta má na starost samotné ukládání dat.

4.4 Hlavní aplikační domény IoT

Internet of Things se rozšířil bleskovým tempem a našel své využití v obrovském množství různých odvětví. Níže jsou některá z těchto odvětví blíže specifikována.



Obrázek 2 - Top 10 oblastí IoT
Zdroj: (autor, dle předlohy Knud Lasse Lueth, 2020)

Z obrázku výše lze usoudit, že největší zastoupení IoT je v industriálním prostředí. Následuje doprava, za kterou je na třetí pozici energie.

4.4.1 Smart Health

Smart Health, či také Smart Healthcare se zrodila z konceptu „Smart Planet“ od společnosti IBM, v NY, USA v roce 2009. Jednoduše řečeno, Smart Planet je inteligentní infrastruktura, která využívá senzory k získávání informací, rozesílá informace skrze IoT a zpracovává informace za pomocí super-počítač a cloud computing technologií. Může tak koordinovat sociální systémy a přispět tak k řízení lidské společnosti. Smart Healthcare využívá technologie, jako například chytré hodinky, IoT, či mobilní zařízení. Díky těmto zařízením může dynamicky přistupovat k informacím, propojovat lidi a instituce, a tak aktivně zpracovávat potřeby zdravotnického ekosystému (Tian et al., 2019).

Konkrétně chytré zdravotnictví se skládá z několika různých účastníků. Patří mezi ně doktoři, pacienti, nemocnice a výzkumné instituce. Jedná se o komplexní celek, který obsahuje mnoho různých odvětví, včetně prevence onemocnění a monitoringu, diagnostiky a léčby, managementu nemocnic, či lékařského výzkumu (Pramanik et al., 2017).

Níže jsou vypsané hlavní aplikační oblasti IoT ve zdravotnictví:

- Monitorování pacientů „remote“ (na dálku) – Tento typ monitorování umožňuje institucím kontrolovat data pacienta v reálném čase. Díky tomu lze zefektivnit léčbu samotného pacienta. Možnost monitorování na dálku také umožňuje pacientům léčbu z domova, čímž se výrazně sníží náklady na hospitalizaci.

- Chytrá zařízení – Díky chytrým zařízením, například náramkům/hodinkám, lze monitorovat status daného jedince. Tato zařízení jsou schopna extrahovat informace, jako například tep, hladinu kyslíku v krvi, či teplotu. Nové Apple Watch 8. generace již nabízí například funkci EKG. Existují také speciální chytrá zařízení, díky kterým lze sledovat mentální stav pacienta. Tato zařízení měří tep a tlak, díky čemuž jsou schopna určit mentální stav pacienta.

- Robotické operace – Využitím malých, na internet připojených robotů, lze provádět složité operace uvnitř lidského těla, které by bylo obtížné realizovat pomocí lidských rukou. Robotické operace prováděné malými IoT zařízeními mohou pomoct snížit velikost řezů nutných pro chirurgický zákrok. Díky tomu se celý proces stává méně invazivním a hojení je tak mnohem rychlejší.

- Poživatelné senzory – Díky těmto senzorům, je možné získávat informace, například z trávicího traktu, méně invazivním způsobem. Tyto senzory poskytují informace o hodnotách pH v žaludku, či pomáhají přesněji určit zdroj vnitřního krvácení. Tato zařízení musí být dostatečně malá, aby bylo pro pacienta snadné je spolknout. Dále také musí být schopna se lehce rozpustit a projít tělem bez zanechání jakýchkoliv zbytků.

4.4.2 Smart Industry

Indrustrial Internet of Things, zkráceně IIoT lze charakterizovat jako využití konceptu IoT v průmyslu. Proč je vlastně využití IoT v průmyslu výhodné? V kontextu výroby je průmyslový Internet věcí důležitý z mnoha důvodů. Jedním z nejdůležitějších je analytika. Výrobci potřebují sbírat data, na základě kterých, jsou schopni určit stav výroby, výkonnost výroby, či zpomalovací elementy výroby. Díky těmto datům mohou výrobci činit rozhodnutí vedoucí ke zlepšení efektivity podniku. Tento proces je výhodné automatizovat kombinací IoT systémů s umělou inteligencí (AI)(Oleksii Tsymbal, 2022).

Seznam níže byl zpracován na základě zdroje: (*The Top 5 Applications For Industrial IoT*, 2023)

- Asset Tracking (sledování majetku) – Jedná se o využití technologie GPS v kombinaci se senzory instalovanými v dopravních prostředcích, či přepravních bednách, za účelem sledování zásilek. Senzory komunikují za pomoci mobilní sítě, což umožňuje organizacím neustále sledovat polohu daných předmětů. Tuto technologii lze využít například pro sledování pronajatých vozidel nebo pro správu flotil. Díky velkému množství dat

z těchto senzorů mohou organizace vylepšit své logistické plánování a optimalizovat tak doručování zboží.

- Preventative Maintenance (prediktivní údržba) – Dříve se pojem „prediktivní údržba“ týkal periodické kontroly daného stroje daným pracovníkem. Zapojení IoT do průmyslu však otevírá nové možnost. Místo toho, aby manažeři, jejichž společnosti implementovali IoT zařízení do výroby, iniciovali návštěvy mechaniků, na základě časových údajů, mohou daná zařízení kontrolovat na dálku v reálném čase a předcházet tak případným poruchám. Senzory, instalované do těchto zařízení, monitorují klíčové výkonnostní metriky a posílají upozornění, pokud jsou detekovány nějaké problémy. To vede ke snížení počtu servisních požadavků, zlepšení efektivity výroby a snížení provozních nákladů.

- Bezpečnost zaměstnanců a pracovního prostředí – Díky propojení určitého množství senzorů a následné implementaci těchto senzorů do pracovního prostředí, lze monitorovat bezpečnost zaměstnanců, pracoviště, či získávat statistiky využívání nářadí/nástrojů. Díky tomu je pro společnosti snazší dodržovat předpisy výroby.

4.4.3 Smart Energy

Dalším z obrovských odvětví, do kterého IoT rychle proniká, je energetika. Díky využití IoT v energetice lze zefektivnit účinnost nebo dodávku energie od výroby ke spotřebiteli (Abir et al., 2021).

Za pomoci chytrých IoT systémů lze sledovat spotřebu energie spolu s dalšími parametry. To umožňuje snížit spotřebu energie až o 40 %. Implementované senzory a inteligentní zařízení mění energetický průmysl tím, že umožňují více do hloubky monitorovat spotřebu energie a lépe analyzovat faktory, jež spotřebu ovlivňují. Existuje velké množství různých návrhů / přístupů, které se touto problematikou zabývají. Avšak se liší cílem zkoumání. Mezi tyto cíle patří například

lepší efektivita, lepší správa zdrojů, snížení emisí, či zlepšení obecného povědomí (Sittón-Candanedo et al., 2019).

4.5 Pohled na komunikace

4.5.1 Wi-Fi

Pojem Wi-Fi označuje lokální bezdrátovou síť, která je klasifikovaná dle standardu 802.11 institutu IEEE (Insitute of Electrical and Electronics Engineers). V průběhu let od svého zavedení, byl standard 802.X dále rozvinut o několik rozšíření. Každé rozšíření se vyznačuje odlišnými charakteristikami, jako například použitým frekvenčním pásem, či dosahem. Každopádně, hlavním cílem každého rozšíření je navýšení toku dat. *Tabulka 1* sumarizuje nejpodstatnější rozšíření spolu s jejich charakteristikami (Retscher, 2020).

	802.11	a	b	g	n	ac	ax
Alt. označení	-	-	-	-	Wi-Fi 4	Wi-Fi 5	Wi-Fi 6
Frekvenční pásmo [GHz]	2.4	5	2.4	2.4	2.4, 5	5	2.4, 5
Šířka pásma [MHz]	22	20	22	20	20, 40	20, 40, 80, 160	20, 40, 80, 160

Tabulka 1 - Rozšíření 802.X
Zdroj: (autor, dle předlohy Retscher, 2020)

Každé frekvenční pásmo má své výhody i nevýhody. Jednoduše, čím vyšší je frekvence, tím kratší je dosah signálu (díky vyššímu útlumu signálu). Tím pádem má teoreticky pásmo 2.4 GHz větší dosah, jelikož dokáže překonat stínící materiály s menší ztrátou. Nevýhodou však je, že toto frekvenční pásmo je používáno u mnoha elektronických zařízení, které využívají rádiovou komunikaci. Díky tomu tak může docházet k vzájemnému rušení. Výhodou pásma 5 GHz je vyšší přenosová rychlost. Tato výhoda je vykoupena nižší prostupností signálu skrze stěny.

4.6 Aplikační vrstva

Aplikační vrstva je nejvrchnější vrstva architektury Internetu věcí. Tato vrstva slouží jako rozhraní pro uživatele, kteří chtějí IoT využívat. Existuje široké spektrum oblastí, ve kterých aplikaci IoT nalezneme. Patří mezi ně například průmyslová výroba, logistika, zdravotnictví, či pouze lidské domácnosti. Oblasti využití IoT jsou více specifikované v části *Využití IoT*.

Kvůli neustále narůstajícímu využití IoT a exponenciálně rostoucímu počtu připojených zařízení v IoT sítích bylo třeba definovat protokoly, dle kterých budou zařízení komunikovat. Tyto protokoly jsou označovány jako tzv. middleware. Middleware je jednoduše software, který slouží jako most mezi operačním systémem a aplikacemi. Toto označení je hojně využíváno v síťovém prostředí. V této práci je jako hlavní uveden protokol MQTT, který je zde porovnán s protokolem CoAP. Existuje však mnohem větší množství protokolů, které se různí svými přístupy a oblastmi použití. Například protokol XMPP (Extensible Messaging and Presence Protocol), který je postaven na jazyce XML. Tento protokol je lehce škálovatelný, právě díky XML a nabízí soubor technologií pro posílání zpráv, multi-party chat, či video a audio hovory. Dalším z takovýchto protokolů je AMQP neboli Advanced Message Queuing Protocol. Tomuto protokolu dali vzniknout pánové John O'Hara a J. P. Morgan v Londýně roku 2003. V jejich článku *Toward a Comodity Enterprise Middleware* popisují, o co se jedná (O'Hara, 2007).

AMQP je binární protokol a velmi dobře definovaný soubor pravidel pro přenos zpráv aplikací mezi různými systémy pomocí kombinace principů store-and-forward a publish-and-subscribe. Protokol je navržen tak, aby ho bylo možné využít v různých programovacích prostředích, operačních systémech a zařízeních. Dále lze protokol využít pro tvorbu vysoce výkonných síťových implementací využívajících transportní síťové protokoly, jako je TCP, či SCTP (Stream Control Transmission Protocol).

4.7 MQTT

MQTT neboli Message Queue Telemetry Transport, je protokol využívaný pro komunikaci v IoT prostředí. MQTT využívá TCP jako transportní protokol. Byl vytvořen společností IBM za účelem jednoduché machine-to-machine komunikace. MQTT protokol by standardizován ISO/IEC 20922 a byl dále přijat, jako součást OASIS[2] (Organization for the Advancement of Structured Information). V jádru je protokol MQTT protokolem zpráv, který využívá komunikační model publish-subscribe (Nastase, 2017).

4.7.1 Princip MQTT

Základní princip fungování protokolu MQTT popisují ve své práci *The Precision Agriculture Based on Wireless Sensor Network with MQTT Protocol*, Y. Syafarinda a spol. Protokol MQTT je navržen tak, aby minimalizoval bandwidth a energetickou náročnost a byl co možná nejvíce jednoduchý a vhodný pro snadnou integraci. Hlavní komponent MQTT tvoří klienti, servery, nebo takzvaní brokers, publishers, subscribers a topics. MQTT klient je jakékoliv zařízení, které se chová jako publisher nebo subscriber. MQTT broker je zodpovědný za příjem všech zpráv, filtrování, rozhodování a posílání zpráv zpět klientovi. Publisher je klient, který posílá zprávy klientům, kteří se přihlásili k jednotlivým brokerům. Subscriber/s jsou klienti, jenž přijímají zprávy skrze brokery dle požadovaného tématu (Syafarinda et al., 2018).

Jednodušeji vysvětleno, v modelu protokolu MQTT dochází k výměně zpráv mezi klientem a brokerem zpráv. Klient může být buď poskytovatel zpráv (Publisher), či odběratel zpráv (subscriber). Poskytovatel zasílá všechna data charakterizovaná dle témat brokerovi. Odběratelé přijímají data od poskytovatelů. Od koho budou odběratelé data přijímat, určuje předplatné daných témat. Důležité je podotknout,

[2] OASIS Open je místo, kde se jednotlivci, organizace a vlády spojují, aby řešili některé z největších technických výzev světa prostřednictvím vývoje otevřeného kódu a otevřených standardů *(Oasis Open, 2023).*

že původní termín broker byl v dnešní době standardizován, jako pojem server. Různé literatury uvádí různá označení, ale shodují se na těchto dvou.

Obrázek 3 - Schéma MQTT
Zdroj: (autor, dle předlohy OpenLabPro, 2019a)

4.7.2 Struktura zpráv

Zprávy rozesílané mezi jednotlivými zařízeními, mají pevně danou strukturu vycházející ze základní definice protokolu MQTT. Zpráva MQTT se skládá z následujících třech částí.

Obrázek 4 - Struktura MQTT zpráv
Zdroj: (autor)

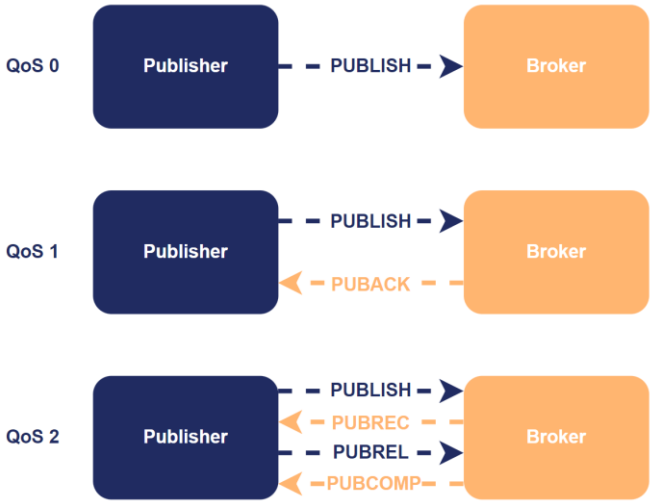
4.7.3 Fixní hlavička

První z nich je **fixní hlavička**. Fixní hlavička je přítomna ve všech zaslaných zprávách, to znamená, že je povinná. Fixní hlavičku tvoří *typ zprávy (message type)*, *příznaky specifické pro každý MQTT paket (flags specific to each MQTT packet)* a *zbývající délka (remaining length)*. Typ zprávy a specifické příznaky tvoří dohromady jeden byte. Zbývající délka poté zabírá od jednoho do čtyř bytů. Typů zprávy existuje v protokolu MQTT celkem patnáct. Pro ilustraci jsou zde uvedeny pouze ty nejznámější. CONNECT (0001) – tímto se označuje žádost klienta o připojení k serveru. Směr přenosu je od klienta k serveru (brokerovi). CONNACK (0010) – potvrzení připojení. Tento typ zprávy využije server směrem ke klientovi, aby mu potvrdil připojení. PUBLISH (0011) – publikace zprávy. Jedná se o obousměrný typ zprávy – klient může poslat PUBLISH směrem k brokerovi a naopak. PUBACK (0100) – potvrzení publikace. Pro PUBACK platí stejná pravidla jako pro PUBLISH. Další částí fixní hlavičky jsou příznaky specifické pro každý MQTT

paket. Jako specifické příznaky se používají první čtyři bity fixního záhlaví. Mezi příznaky patří například DUP – když klient nebo broker MQTT provede opětovné odeslání paketu, nastaví se duplikát. Tento příznak se hojně využívá v typech zpráv PUBLISH, SUBSCRIBE atd. Další příznak je QoS neboli kvalita služeb. Kvalita služeb má tři úrovně – úroveň nula, jedna a dva (ipc2U.cz, 2023).

Obrázek 5 - Fixní hlavička
Zdroj: (autor, dle předlohy ipc2U.cz, 2023)

QoS úroveň je jinými slovy úmluva mezi odesílatelem zprávy a jejím příjemcem, která garantuje doručení dané zprávy. Nejnižší úroveň – úroveň nula (at most once) znamená, že bude vynaloženo maximální úsilí, avšak doručení není garantované. Druhá úroveň – úroveň jedna (at least once) říká, že zpráv bude doručena alespoň jednou. Aby se tak stalo, odesílatel zprávy tuto zprávu ukládá, dokud mu nepřijde od příjemce PUBACK (potvrzení o přijetí zprávy). Poslední úrovní, kterou QoS nabízí, je úroveň dva (exactly once). Druhá úroveň garantuje, že každá zpráva je vybraným příjemcům doručena pouze jednou. Úroveň dva QoS je tak nejspolehlivější, zároveň je však nejpomalejší ze všech úrovní (The HiveMQ Team, 2015a).



Obrázek 6 - Úrovně QoS
Zdroj: (autor, dle předlohy ipc2U.cz, 2023)

Posledním příznakem je RETAIN. Pokud je tento příznak nastaven na hodnotu true, znamená to, že broker ukládá poslední zprávu tohoto (RETAIN) typu pro odpovídající téma. V praxi to funguje následujícím způsobem. Klient A publikuje RETAIN zprávu do *myhome/livingroom/temperature.* Chvíli potom se klient B rozhodne odebírat *myhome/#*. Hned po této akci klient B obdrží RETAIN zprávu *myhome/livingroom/temperature*. Klient B z této zprávy zjistí, že se jedná o zprávu typu RETAIN, jelikož broker tuto zprávu zaslal s příznakem RETAIN nastaveným na hodnotu true. Klient B se v tuto chvíli může rozhodnout, jak s touto zprávou naloží (The HiveMQ Team, 2015b).

4.7.4 Variabilní hlavička

Druhou částí celkové struktury zpráv MQTT je variabilní hlavička. Variabilní hlavička je obsažena v některých záhlavích. To znamená, že není součástí všech zpráv (ipc2U.cz, 2023).

Protocol name je součástí variabilní hlavičky ve zprávě typu CONNECT. Toto pole je šifrováno pomocí UTF a zobrazuje jméno protokolu. Následuje pole *Protocol version*, které je opět přítomné pouze ve zprávě typu CONNECT. Jedná se o pole délky osm bitů, jež representuje revizní úroveň protokolu používaného uživatelem. Dalším polem po verzi protokolu je *Connect flags*. Velikost Connect flags je jeden byte. Takzvaných vlajek/příznaků existuje celkem šest. Patří mezi ně například *User Name flag*, *Password flag*, *či Clean Session flag*. Pro Connect flags byte platí opět to, že se vyskytuje ve zprávě typu CONNECT. Další pole variabilní hlavičky je *Keep alive timer*. Toto pole je přítomné ve zprávě typu CONNECT. Keep alive timer ukládá maximální časový interval mezi zprávami přijatými od klienta. Umožňuje serveru detekovat propady sítě bez toho, aniž by musel čekat na dlouhý TCP/IP timeout. Po Keep alive timer poli existuje pole *Connect return code*. Toto pole je využíváno ve zprávách typu CONNACK. Connect return code jednoduše indikuje návratový kód zprávy. Může to být například 0, což znamená, že připojení bylo přijato, kód 5 znamená například, že připojení bylo odmítnuto z důvodu špatné autorizace. Posledním polem variabilní hlavičky je *Topic name*. Toto pole je přítomné v MQTT PUBLISH typu zprávy. Topic name je šifrované pomocí UTF-8 a identifikuje

informace kanálu, skrze který jsou data sdílena (International Business Machines Corporation (IBM) & Eurotech, 2010).

Obrázek 7 - Variabilní hlavička
Zdroj: (autor, dle předlohy ipc2U.cz, 2023)

4.7.5 Payload

Poslední – třetí částí struktury zprávy je *payload (náklad)*. Payload je volitelný a liší se dle typu paketu. Toto pole častokrát obsahuje data, která se posílají. Pro CONNECT paket je payload ID klienta, uživatelské jméno a heslo. Pro PUBLISH paket je to zpráva, jež má být poslána (OpenLabPro, 2019b).

4.7.6 Struktura zprávy typu CONNECT

Pro názornost je níže celá struktura zprávy se všemi jejími součástmi, fixní hlavičkou, variabilní hlavičkou a nákladem rozepsána pro paket CONNECT.

Fixní hlavička zprávy typu CONNECT má následující podobu. Hodnota CONNECT je jedna, převedeno do dvojkové soustavy 0001. Vlajky DUP, QoS a RETAIN nejsou pro zprávu typu CONNECT využity, tudíž zůstávají nulové. Remaining length (neboli zbývající délka) je délka variabilní hlavičky dohromady s délkou nákladu (payload). Remaining length je tedy 12 bytes. Variabilní hlavička zprávy typu CONNECT obsahuje příznaky, Keep alive timer, název protokolu apod. Pro CONNECT zprávu jsou příznaky User name flag a Password flag nastaveny na hodnotu true, stejně tak příznak Clean session flag. Keep alive timer lze nastavit například na hodnotu deseti sekund – převedeno do hexadecimálního formátu 0x000A. Payload zprávy CONNECT obsahuje jeden nebo více UTF-8 šifrovaných textových elementů, které závisí na příznacích označených ve variabilní hlavičce. Jelikož jsou příznaky User name flag a Password flag nastaveny na true, musí být vyplněny i v části payload. Část payload má pevnou strukturu, dle které jsou jednotlivá pole uspořádána. První je *Client Identifier* (ID klienta). Toto ID může nabývat od jedné do dvaceti tří znaků a slouží k identifikaci daného klienta. Musí platit, že Client ID je unikátní skrze celou skupinu klientů přistupujících k danému

serveru (brokerovi). Pokud Client ID obsahuje více než dvacet tři znaků, je mu přístup zamítnut. Po Client ID následují dvě Will pole – Topic a Message. *Will Message* je publikována do *Will Topic*. Will message definuje obsah zprávy, která je poslána do Will Topic, pokud se klient nečekaně odpojí. Může se jednat o zprávu, jež má délku nula. Po Will polích následuje *User Name*, který, pokud je User Name flag nastavena na hodnotu *true*, musí být vyplněn. User Name neboli uživatelské jméno identifikuje uživatele, který se připojuje k serveru a je užito pro následnou autentizaci. Doporučená délka uživatelského jména je dvanáct znaků. Po uživatelském jménu následuje poslední pole – *Password* (heslo). Pokud je příznak Password flag pravda, musí být heslo vyplněno. Opět, stejně jako uživatelské jméno, je heslo využito při ověřování uživatele (International Business Machines Corporation (IBM) & Eurotech, 2010).

Po odeslání zprávy typu CONNECT následuje určitá odezva (response). Server posílá CONNACK zprávu zpět klientovi. Pro komunikaci klienta se serverem a následné zasílání CONNECT a CONNACK zpráv platí určitá nepsaná pravidla. Pokud je klient se stejným Client ID připojen k danému serveru, „starší“ klient bude odpojen serverem před tím, než se dokončí proces zprávy CONNECT. Pokud klient nedostane od serveru zprávu typu CONNACK v rámci určitého časového úseku, měl by uzavřít TCP/IP připojení a restartovat spojení tím, že otevře nový socket a pokusí se o znovu-zaslání zprávy typu CONNECT. Pokud server neobdrží zprávu typu CONNECT v rámci určitého časového úseku po iniciování TCP/IP propojení, server by měl uzavřít toto připojení. V rámci obou výše zmíněných pravidel „určitý časový úsek“ závisí na typu aplikace a komunikační infrastruktury, a tudíž se může různit (International Business Machines Corporation (IBM) & Eurotech, 2010).

4.7.7 Sémantika témat

„Témata jsou znaky s kódováním UTF-8. Hierarchie témat má stromovou podobu, což usnadňuje jejich organizaci a přístup k datům. Témata se skládají z jedné nebo více úrovní, které jsou odděleny lomítkem“ (ipc2U.cz, 2023).

Při vytváření určité aplikace a stromu témat, který tato aplikace bude využívat, je třeba následovat určité postupy a doporučení. Zde jsou uvedeny ty nejdůležitější z nich. Téma musí být alespoň jeden znak dlouhé. Jména témat jsou citlivá na velká a malá písmena. GROUNDFLOOR a GroundFloor jsou tedy dvě různá témata. Jména témat mohou obsahovat slova oddělená mezerníkem. Například Ground floor je validní téma. Lomítko vytváří samostatné téma. /groundfloor je něco jiného než groundfloor, jelikož /groundfloor lze přirovnat k +/+ a k /+, avšak ne k +. Posledním pravidlem je neuvádění null znaku (unicode \x0000) v žádném ze jmen témat. IBM doporučuje: *„Délka stromu je limitována 64k, ale vně stromu neexistují žádné limity, co se týče počtu úrovní. Může zde existovat libovolný počet kořenových úrovní, což znamená, že může existovat libovolné množství stromů témat“* (International Business Machines Corporation (IBM) & Eurotech, 2010).

Příkladem základního tématu, ve kterém snímač vlhkosti umístěný v ložnici publikuje data serveru, může být: */home/groundfloor/bedroom/humidity*.

Subscriber však nemusí pokaždé přijímat data pouze z jednoho daného tématu. Z tohoto důvodu byly nastaveny takzvané zástupné znaky (wildcards). Zástupných znaků existují dvě úrovně.

Zástupným znakem **jednoúrovňovým** je **znaménko plus (+)**. Jak již z názvu vyplývá, tímto zástupným znakem lze nahradit pouze jednu úroveň. Například při následujícím použití */home/+* nahrazuje znak + */home/groundfloor*, či */home/second-floor*, avšak nenahrazuje */home/groundfloor/bedroom*. Tento zástupný znak může být využit pro jakoukoliv úroveň, vždy vedle oddělovače témat („/“), či samostatně. Je tedy validní */home/+* a také *+.* Validní však není */home+*. Lze ho také využít uprostřed daného stromu témat - */home/groundfloor/+/humidity*.

Zástupným znakem **víceúrovňovým** je **znaménko hash (#)**. Tento zástupný znak dokáže nahradit více úrovní daného tématu. Pokud tedy klient začne odebírat téma */home/groundfloor/#* bude dostávat zprávy na témata: */home/groundfloor*, */home/groundfloor/bedroom*, */home/groundfloor/bedroom/humidity,* či například */home/second-floor/bedroom/humidity*. Stejně tak, jako pro jednoúrovňový znak platí, že ho lze využít samostatně, nebo jen za lomítkem, i víceúrovňový znak hash lze použít samostatně, či za lomítkem. Tudíž */home/#* a # jsou validní, ale */home#* validní není. Zároveň také platí, že víceúrovňový znak # musí být vždy posledním znakem stromu témat. */home/groundfloor/#* je platné, nýbrž */home/#/bedroom* platné není (International Business Machines Corporation (IBM) & Eurotech, 2010).

4.7.8 Výhody a nevýhody MQTT

Stejně tak, jako všechny ostatní protokoly, i protokol MQTT má své výhody a nevýhody. Některé z nich přímo vyplývají z obsahu této práce.

Nenáročnost a efektivita.

MQTT protokol ke své činnosti požaduje minimální náklady na data a energii. Díky tomu může být používán například malými mikrokontrolery, které obsahují slabší procesory a menší operační paměť (RAM). Jak již bylo zmíněno výše, protokol MQTT je kromě HW šetrný i k síti. Díky posílání menšího množství paketů má protokol nízké nároky na rychlost sítě (esperso, 2020).

Bezpečnost

V dnešní době je velmi důležité, aby protokoly využívané pro komunikaci byly bezpečné. I to je vlastnost, kterou protokol MQTT splňuje. Díky kryptografickému protokolu TLS (Transport Layer Security) lze jednoduše šifrovat zprávy. MQTT mimo to nabízí také možnost ověřovat klienty pomocí moderních autentizačních protokolů, jako je například OAuth (esperso, 2020).

Spolehlivost

Další podstatnou vlastností protokolu MQTT je jeho spolehlivost. Jak bylo zmíněno výše, QoS je obsažena jako příznak ve fixní hlavičce a má tři úrovně. Díky tomu lze

fungování protokolu MQTT nastavit na co možná nejspolehlivější – třetí úroveň. Tím se zajistí stabilní přenos zpráv mezi zařízeními.

Výdrž na baterii

Jelikož je protokol MQTT nenáročný, nepotřebuje ke svému fungování konzumovat velké množství energie. Společnost IBM vyvinula MQTT pro monitorování potrubí v pouštích. Jelikož neměli přístup k elektrické síti, museli vyvinout protokol, který má co možná nejmenší spotřebu elektrické energie, vhodný pro komunikaci skrze satelity. V dnešní době je většina IoT zařízení propojena skrze bezztrátovou komunikaci a jsou poháněna Li-ion bateriemi. Vědci porovnávali protokol MQTT s protokolem HTTP na zařízeních poháněných baterií a výsledkem bylo zjištění, že MQTT konzumuje sto sedmdesátkrát méně energie na 3G sítích a čtyřicet sedmkrát méně energie na Wi-Fi v porovnání s HTTP protokolem. MQTT tedy umožňuje vývojářům IoT vytvořit zařízení, která vydrží více než deset let pouze na baterii (esperso, 2020).

Fungování na TCP/IP i Non-TCP/IP sítích

I přes to, že protokol MQTT běžně funguje na sítích TCP/IP, neznamená to, že by nebyl schopný fungovat na sítích jiného typu. Dokud daný protokol umožňuje bezztrátové, obousměrné připojení, podporuje samotné MQTT. Například síť Zigbee není TCP/IP sítí, či existují i protokoly neuspořádané – například UDP. Pro tyto sítě byla vytvořena upravená verze protokolu MQTT – MQTT-SN (MQTT Sensor Networks). Vazby typu **Many-to-Many**, **1-to-1, 1-to-Many,** či **Many-to-1** jsou všechny podporovány. Jelikož v protokolu MQTT funguje, že jakékoliv připojené zařízení může odebírat jakékoliv téma, fungují i tyto typy propojení, či rozesílání zpráv (esperso, 2020).

4.8 SMQTT

SMQTT je prakticky rozšíření protokolu MQTT. Jelikož je protokol SMQTT založen na mechanismu zasílání šifrovaných zpráv, poskytuje zabezpečený standard pro komunikaci. V tomto protokolu posílá odběratel (subscriber) zašifrované zprávy všem uzlům, které tuto zprávu přijmou a po dešifrování s ní dále pracují. Procesy šifrování a dešifrování se provádějí za pomoci hlavního klíče – master key. Protokol SMQTT má čtyři hlavní fáze – nastavení, šifrování, publikování a dešifrování.

Seznam níže byl zpracován na základě (GeeksForGeeks, 2020):

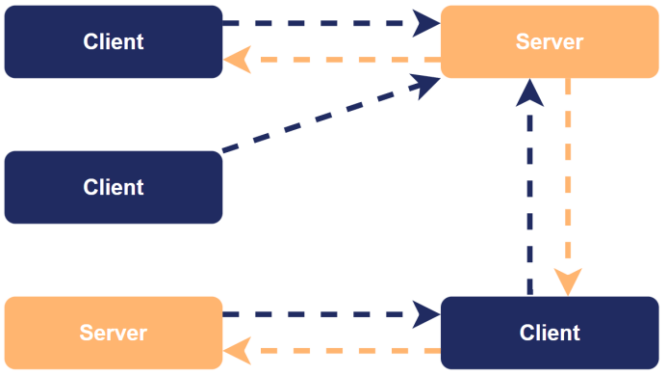
1. Ve fázi nastavení, se poskytovatel zpráv i odběratel zpráv zaregistrují poblíž brokera a obdrží tak hlavní klíč (master key).
2. Ve fázi šifrování, broker zašifruje publikované zprávy.
3. Ve fázi publikování poskytne broker zašifrovaná data odběratelům.
4. Ve fázi dešifrování, která je poslední fází, jsou data/zprávy dešifrovány odběratelem pomocí hlavního klíče.

Základní důvody využití tohoto protokolu popisuje Omnia Ayman ve své práci Analysis Of Application Layer Protocols Used In Internet Of Things neboli Analýza protokolů aplikační vrstvy používaných v oblasti Internetu věcí: Konfigurace MQTT je vhodná pro práci v nezabezpečených sítích, jelikož nemá vynucené žádné bezpečnostní komponenty. Zabezpečení protokolu MQTT závisí na šifrování SSL/TLS. Jedná se o obecnou normu pro ověřování ve prostředí IoT. Použití šifrování SSL/TLS v prostředí IoT je však velmi nákladné. Proto vznikla varianta MQTT – SMQTT. SMQTT je zabezpečený protokol MQTT, ve kterém je zpráva zakódována a přenesena do různých uzlů (nodes), což je pro využití v prostředí IoT velmi výhodné. Zabezpečené varianty SMQTT a SMQTT-SN byly rozšířeny na MQTT a MQTT-SN, individuálně závislé na zásadě Key/Cipher Text Policy, založené na ECC – kryptografii eliptických křivek (Ayman, 2019).

4.9 CoAP

Dalším hojně využívaným protokolem ve světě IoT je protokol CoAP. Zkratka CoAP znamená Constrained Application Protocol – Omezený aplikační protokol.

Jedná se o nenáročný M2M protokol od IETF CoRE (Constrained RESTful Environments) Working Group. Protokol podporuje jak request/response, tak i resource/observe (varianta publish/subscribe) architektury. CoAP byl především vyvinut pro spolupráci s HTTP a RESTful Web prostřednictvím jednoduchých proxy. Na rozdíl od protokolu MQTT, protokol CoAP využívá URI (Universal Resource Identifier) místo témat (topics). Poskytovatel zpráv poskytne data na určitou URI. Odběratel odebírá určitý zdroj, jež je identifikovaný pomocí unikátní URI. Když poskytovatel zpráv publikuje nová data na danou URI, jsou všichni odběratelé informování o nové hodnotě dané URI (Naik, 2017).



Obrázek 8 - Schéma CoAP
Zdroj: (autor, dle předlohy Wallarm, 2023)

Protokol CoAP využívá UDP jako transportní protokol a DTLS pro zabezpečení. To znamená, že klienti a servery komunikují pomocí datagramů bez připojení s menší spolehlivostí. CoAP využívá „potvrzené“ a „nepotvrzené“ zprávy k zajištění dvou různých úrovní QoS. Potvrzené zprávy musí být potvrzeny příjemcem pomocí ACK paketu, nepotvrzené zprávy tímto procesem neprochází. CoAP nabízí více funkcionalit než protokol MQTT, jelikož podporuje vyjednávání o obsahu (content negotiation) za účelem vyjádření preferované reprezentace zdroje. To umožňuje

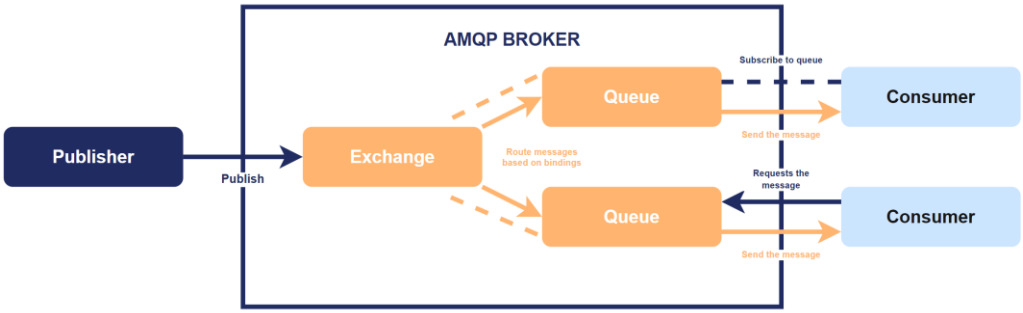
klientovi a serveru vyvíjet se nezávisle a přidávat nové reprezentace, aniž by se navzájem ovlivňovaly (Naik, 2017).

4.10 AMQP

Dalším z protokolů, které jsou v IoT využívány, je protokol AMQP.

V technické specifikaci vytvořené přímo společností OSASIS je protokol popsán následovně: Protokol AMQP je rozdělen do několika různých vrstev. V nejnižší vrstvě se definuje binární peer-to-peer protokol, který slouží k přenosu zpráv mezi dvěma procesy skrze síť. Dále je definován abstraktní formát zpráv s konkrétním standardem šifrování. Každý proces v AMQP musí být schopný přijímat a posílat zprávy v tomto standardu (amqp.org, 2011).

Obrázek níže reprezentuje strukturu fungování AMQP:



Obrázek 9 - Schéma AMQP
Zdroj: (autor, dle předlohy Wallarm, 2023a)

Z obrázku výše vyplývá, že zařízení propojená skrze AMQP protokol mají různé role. První z nich je publisher, někdy také nazývané jako producer. Tento typ zařízení publikuje zprávy. Druhým typem je consumer. Consumer přijímá zprávy od publishera a dále s nimi pracuje. Aby se zpráva dostala od publishera ke správnému consumerovi, je zapotřebí AMQP broker. Jedním z nejpopulárnějších brokerů je RabbitMQ[3]. Broker využívá dvě komponenty – *exchange* a *queue*. Exchange přijímá zprávy od publishera a přesměrovává je do jednotlivých front

[3] Open-source broker, který podporuje jak AMQP, tak například MQTT a STOMP (RabbitMQ, 2023).

(queue). Consumer si poté tyto zprávy z dané fronty vytáhne a dále s nimi nakládá (IoT Boys, 2018).

4.11 Porovnání specifikace protokolů

Tabulka níže slouží k analýze jednotlivých protokolů aplikační vrstvy, zmíněných v této práci.

	MQTT	CoAP	AMQP
Rok vydání	1999	2010	2003
Architektura	Client/Broker	Client/Server Client/Broker	Client/Server Client/Broker
Velikost hlavičky	2 Byte	4 Byte	8 Byte
QoS	0, 1, 2	Conf., Non-conf.	Settle, Unsettle
Transportní protokol	TCP (MQTT-SN UDP)	UDP, SCTP	TCP, SCTP
Zabezpečení	TLS/SSL	DTLS, IPSec	TLS/SSL, IPSec, SASL

Tabulka 2 - Porovnání protokolů aplikační vrstvy
Zdroj: (Naik, 2017)

Z tabulky vyplývá, že nejnovější z protokolů je CoAP. Z hlediska efektivity a úspory paměti lze vidět, že nejmenší hlavičku má protokol MQTT – a to pouze 2 bajty. Všechny protokoly využívají určitý systém QoS a podporují TLS zabezpečení. CoAP využívá DTLS, to je ale protokol postavený na TLS.

4.12 TLS (Transport Layer Security)

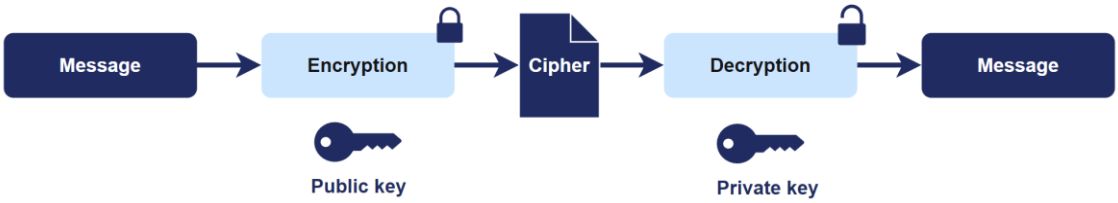
Kapitola níže se věnuje porovnání zabezpečení jednotlivých protokolů. Jelikož se o zabezpečení těchto protokolů stará TLS (v různých formách), je tento odstavec věnován popisu tohoto protokolu.

TLS se kryptografický protokol, který poskytuje end-to-end zabezpečení dat poslaných mezi aplikacemi skrze internet (…) Protokol TLS využívá kombinaci symetrické a asymetrické kryptografie, jelikož tato kombinace poskytuje ideální kompromis mezi výkonem a zabezpečením při přenosu dat (Internet Society, 2023).

4.12.1 Asymetrická kryptografie

Protokol TLS využívá jak asymetrickou, tak symetrickou kryptografii pro zajištění bezpečného spojení. Co to ale asymetrická kryptografie znamená?

Asymetrická kryptografie je známá jako kryptografie veřejného klíče. Odesílatel používá veřejný (public) klíč příjemce pro zašifrování dané zprávy. Příjemce poté použije svůj privátní (private) klíč, pomocí kterého zprávu dešifruje (Thilagavathy & Institute of Electrical and Electronics Engineers, 2014).



Obrázek 10 - Asymetrická kryptografie
Zdroj: (autor, dle předlohy Thilagavathy & Institute of Electrical and Electronics Engineers, 2014)

Výhody asymetrické kryptografie

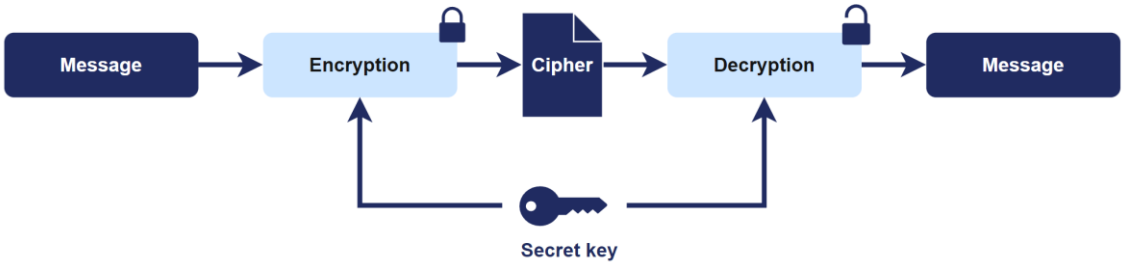
- Data mohou být dešifrována pouze pomocí privátního klíče.
- Pokud je veřejný klíč ztracen / ukraden, data nejsou kompromitována.

Nevýhody asymetrické kryptografie

- Je o poznání pomalejší než symetrická kryptografie.
- Pokud dojde ke ztrátě privátního klíče, nelze ho získat zpět (Arnaud, 2023).

4.12.2 Symetrická kryptografie

Druhým typem kryptografie, kterou protokol TLS využívá, je symetrická kryptografie, někdy také nazývaná jako kryptografie sdíleného klíče (shared-key cryptography). U tohoto typu kryptografie má odesílatel i příjemce stejný klíč pro šifrování i dešifrování (Thilagavathy & Institute of Electrical and Electronics Engineers, 2014).



Obrázek 11 - Symetrická kryptografie
Zdroj: (autor, dle předlohy Thilagavathy & Institute of Electrical and Electronics Engineers, 2014)

Níže uvedené výčty byly vytvořeny na základě (Arnaud, 2023):

Výhody symetrické kryptografie

- Je jednodušší na implementaci.
- Je rychlejší než asymetrická kryptografie.
- Hodí se pro práci s většími objemy dat.

Nevýhody symetrické kryptografie

- Pokud dojde ke ztrátě klíče, mohou být data kompromitována.
- Klíč musí být sdílen pomocí další strany.

4.12.3 Vztah TLS a SSL

Posledním bodem tohoto nadpisu je vztah mezi protokolem TLS a SSL. Oba pojmy jsou hojně využívané, avšak méně lidí tuší, jak spolu tyto dva protokoly souvisí. Protokol TLS je postaven na nejnovější verzi protokolu SSL – 3.0. Protokol SSL byl vyvinut společností Netscape Communications Corporation v roce 1994 a byl určen k zabezpečení webových relací. První verze protokolu – 1.0 nebyla nikdy veřejně vydaná a druhá verze – 2.0 byla rychle nahrazena verzí 3.0 (Internet Society, 2023).

Níže jsou postupně seřazeny verze protokolů, v pořadí, v jakém vycházely. Vytvořeno na základě (Kinsta.com, 2022):

- SSL 1.0 – nikdy nebyl veřejně vydán
- SSL 2.0 – vydán v roce 1995
- SSL 3.0 – vydán v roce 1996
- TLS 1.0 – vydán v roce 1999 jako upgrade SSL 3.0
- TLS 1.1 – vydán v roce 2006
- TLS 1.2 – vydán v roce 2008
- TLS 1.3 – vydán v roce 2018 (nejaktuálnější verze)

4.13 Porovnání zabezpečení protokolů

Bezpečnost je jedním z nejdůležitějších faktorů při vybírání správného aplikačního protokolu. Vybrání protokolu s nízkou mírou zabezpečení může velkému množství komplikací. V následujících odstavcích je popsáno, jakým způsobem řeší jednotlivé protokoly (zmíněné výše) zabezpečení.

4.13.1 Zabezpečení MQTT (SMQTT)

Hlavní komponentou každé IoT aplikace postavené na MQTT protokolu je MQTT broker. MQTT broker, jak již bylo zmíněno výše, nabízí klientům různé služby. Hlavní zranitelností MQTT brokeru je jeho zahlcení, což vede k DoS útoku. Útočník kompromituje brokera a během DoS útoku posílá falešné kontrolní nebo datové pakety.

Existují dvě možná opatření, díky kterým lze DoS útoku předcházet. Prvním z nich je certifikát založený na SSL/TLS autentizaci. Ten však není doporučený pro IoT zařízení, jelikož správa certifikátů znatelně zvyšuje výpočetní a komunikační nároky. Zároveň generování a distribuce klíčů SSL/TLS protokolu snižuje výkon MQTT. Druhé bezpečnostní opatření, které lze aplikovat v oblasti DoS útoku, je regulace šíření (throttling). Throttling slouží k omezení rychlosti přenosu zpráv, čímž brání útočníkovi v identifikaci často odebíraných témat. Regulace šíření však není účinná při velkých DoS útocích, jelikož při ní může dojít k odmítnutí důležitých zpráv (A. P. & K., 2019).

Samozřejmě je zde další řešení zabezpečení MQTT protokolu, a to jeho rozšíření – SMQTT, které je detailněji popsáno výše v práci. Vyvstává tedy otázka, zda je lepší využít pouze MQTT protokol, či se více vyplatí použít protokol SMQTT. Z informacích uvedených v této práci vychází, že toto rozhodnutí záleží na konkrétních požadavcích dané aplikace. Pokud má být IoT aplikace svižná, má mít nízkou latenci a vysokou rychlost přenosu dat, lze použít čistě protokol MQTT. To však znamená, že bezpečnostní opatření musí být aplikována na jiných úrovních sítě. Na druhou stranu, pokud je pro IoT aplikaci klíčové, aby byla vysoce zabezpečená, lze zvolit protokol SMQTT, jelikož disponuje řadou bezpečnostních funkcí, jako například detekcí a prevencí DoS útoků.

4.13.2 Zabezpečení CoAP

Mezi hlavní bezpečnostní rizika protokolu CoAP patří následující – vytvořeno na základě (A. P. & K., 2019):

- Parsing útoky, při kterých útočník způsobí pád vzdáleného uzlu (remote node) tím, že na něm spustí libovolný kód.
- Caching útoky, při kterých útočník získá kontrolu nad proxy serverem s funkcí „cachování“. To představuje hrozbu pro klienty, kteří si vyměňují data s tímto proxy serverem. Útočník může jednoduše získat tato data a odcizit je.

- Amplification útoky – tento útok je typem DDoS útoku, při kterém útočník použije koncová zařízení k převedení malého paketu na velký paket. CoAP server dokáže redukovat tyto útoky díky Blocking/Slicing modům.
- Spoofing útoky.
- Cross-Protocol útoky, ke kterým dochází při translaci z TCP protokolu na UDP.

Pro zabezpečení protokolu CoAP lze využít Datagram Transport Layer Security neboli DTLS. DTLS lze využít v oblastech automatické správy bezpečnostních klíčů, či šifrování dat a autentizaci. CoAP dohromady s DTLS podporou je považován za zabezpečený CoAP protokol (CoAPs, secure-CoAP) (Raza et al., 2012).

Při použití DTLS nastává jeden problém. Hlavičky (headers) protokolu DTLS jsou velmi dlouhé a nevejdou se do jedné IEEE 802.15.4 MTU. MTU (Maximum Transmission Unit) je jednoduše maximální přenosová jednotka. Pro 802.15.4 je to 127 bajtů. Aby bylo možné protokol DTLS používat, musí být využit standard 6LoWPAN. 6LoWPAN (IPv6 over Low-power Wireless Personal Area Network) poskytuje mechanismy které dokážou komprimovat hlavičku. Z toho důvodu je výhodné využít tyto dvě technologie dohromady a poskytnout tak protokolu CoAP dostatečné zabezpečení (Raza et al., 2012).

4.13.3 Zabezpečení AMQP

O zabezpečení protokolu AMQP se starají dvě hlavní technologie – SASL[4] a TLS. Tyto technologie mají za úkol zabezpečit protokol AMQP na největší možné úrovni, avšak jako každý IoT protokol, i pro AMQP existují určité hrozby. Dle (McAteer et al., 2017) jsou to například:

[4] SASL (Simple Authentication and Security Layer) je framework, který poskytuje autentizaci a stará se o zabezpečení dat v tzv. connection-oriented protokolech (Melnikov & K. Zeilenga, 2006).

- Replay – Útočník vymění/přehraje data mezi komunikačními relacemi. Aplikace si myslí, že se jedná o běžného uživatele, který s ní interaguje. Díky tomu může získat určité informace a data.
- Masquarade – Útočník získá přístup do systému, či vykoná podezřelé úkony tím, že se vydává za autorizovanou entitu.
- Modification – Útočník přidá, či odebere data z obsahu komunikovaného přes síť.
- Denial of Service (DoS) – Útočník tímto typem útoku znemožní přístup k serveru běžným uživatelům.

4.13.4 Shrnutí zabezpečení protokolů

Zde je stručné shrnutí možností zabezpečení u výše zmíněných protokolů vytvořené dle předlohy (McAteer et al., 2017):

- MQTT – k zabezpečení protokolu MQTT lze využít jeho rozšíření – SMQTT, které dokáže šifrovat zprávu před publikací brokerem. Takto šifrovaná zpráva je následně zaslána odběratelům, u kterých je dešifrována pomocí takzvaného master key. K šifrované komunikaci lze také využít port, jež podporuje TLS.
- CoAP – pro zabezpečení tohoto protokolu je vhodné využít protokol DTLS, který poskytuje možnost šifrování dat a také například generování bezpečnostních klíčů.
- AMQP – autentizaci doručení zajišťuje SASL (Simple Authentication and Security Layer). Stejně jako u MQTT, šifrování poskytuje TLS (Transport Layer Security).

5 Technologické řešení – simulace IoT

5.1 Prostředí implementace

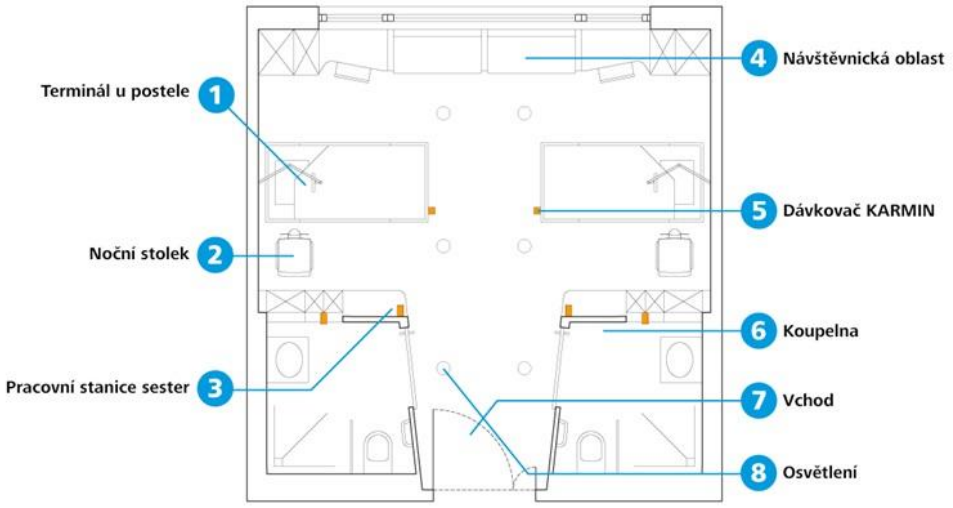
Celá architektura je navržená tak, aby ji bylo možné využít v pacientském pokoji. Navzájem propojená zařízení – senzory, mikrokontrolery a aktuátory se starají o to, aby se pacient cítil komfortně.

Důležitým aspektem pro pacienta je také struktura / rozpoložení daného pokoje. Ta může mít velký vliv na hygienu a následné šíření infekčních onemocnění. Po rešerši jednotlivých projektů zabývajících se architekturou pacientského pokoje, vyvstal jako nejlepší výzkumný projekt KARMIN, jež se zabýval návrhem nového standardu architektury pacientského pokoje.

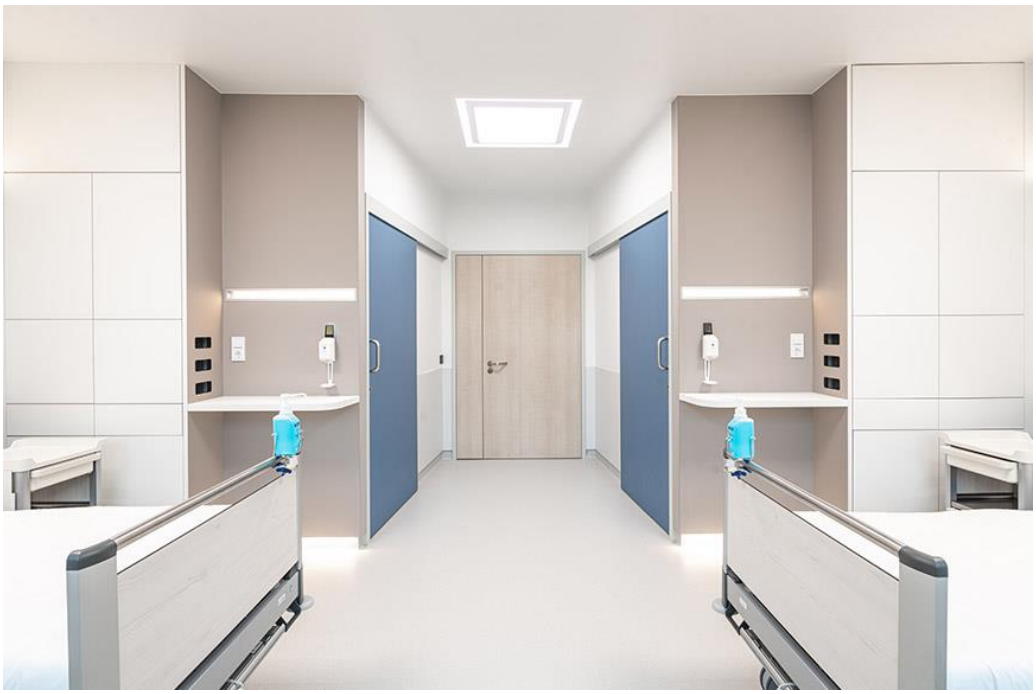
„KARMIN („Krankenhaus, Architektur, Mikrobiom und Infektion“) je projekt, který byl podporován v letech 2016 až 2022 Federálním ministerstvem pro vzdělávání a výzkum (BMBF) v rámci finančního podpůrného programu „Zwanzig20“ a jako součást výzkumné sítě „InfectControl 2020“. Kromě Technické univerzity v Braunschweigu (koordinátora: Institut pro konstruktivní návrh, průmyslovou a zdravotnickou výstavbu) jsou součástí projektu také Charité – Berlínská univerzitní nemocnice (Institut pro hygienu a environmentální medicínu), Univerzitní nemocnice v Jeně se skupinou Septomics Research Group a společnost Röhl GmbH z Waldbüttelbrunnu u Würzburgu.“ (Překlad autora Moellmann et al., 2022)

Jeden z příkladů, jak může architektura pokoje ovlivnit, je přímo uvedený na stránkách Hartmann Group: *„Hygiena závisí zejména na činnosti lidí na nemocničním pokoji: na chování pacientů, na pohybu nemocničního personálu během pracovních činností v místnosti, na tom, kde a jak se v místnosti pohybují návštěvy a na tom, jak je místnost uklízena a dezinfikována. Například z tohoto důvodu jsou všechny předměty v prototypové místnosti KARMIN, které se dotýkají podlahy, pohyblivé, takže je lze snadno během úklidu přesunout.“* (*Uzdravující architektura: Prevence infekcí již od návrhu*, 2023)

Implementace je zasazena do prostředí dvoulůžkového pokoje. Obrázky níže slouží k bližší specifikaci architektury tohoto typu pokoje. Zároveň zde jsou také zahrnuty vizualizace toho, jak takovýto pokoj může vypadat v realitě.



Obrázek 12 - Schéma pokoje
Zdroj: (Harmann.info, 2023a)



Obrázek 13 - Vizualizace pokoje
Zdroj: (Harmann.info, 2023b)

5.2 Schéma implementace

Pro celou implementaci bylo vytvořeno jednoduché schéma, dle kterého je specifikována. Níže je přiložen obrázek, jež toto schéma vykresluje. Ve středu celého schéma se nachází router. Konfigurace tohoto routeru je popsaná v části *Nastavení lokální sítě*. K routeru skrze Wi-Fi přistupuje několik zařízení.

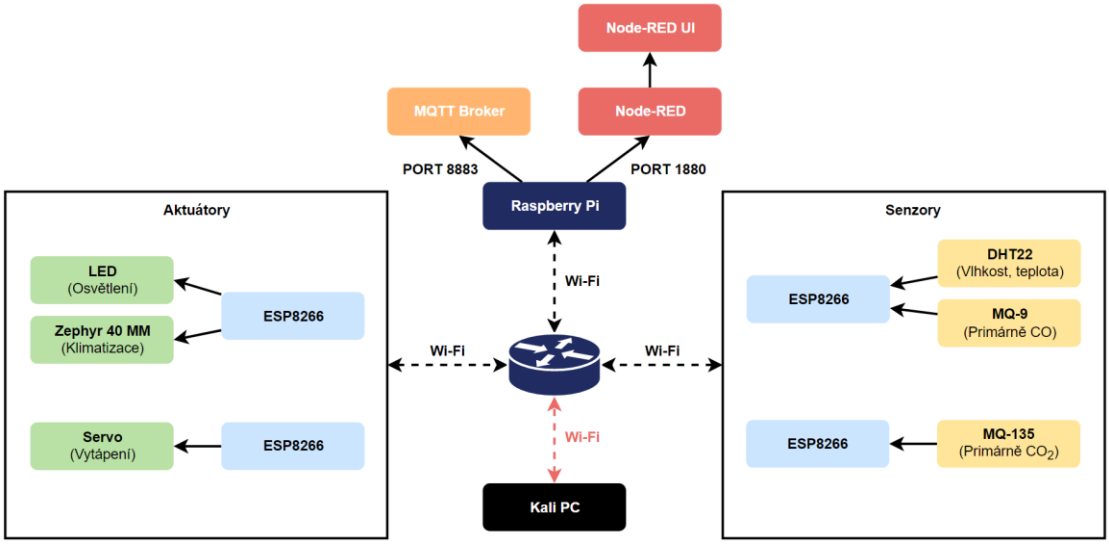
Prvním z těchto zařízení je Raspberry Pi 3 B+. Konfigurace Raspberry Pi je popsána v části *Konfigurace Raspbberry Pi.* Raspberry Pi slouží jako MQTT Broker, který poslouchá na portu číslo **8883** (1883 bez TLS/SSL). Dále je zde nainstalován Node-RED, jenž běží na portu číslo **1880**. Dashboard (Node-RED UI) je nakonfigurován na adresu **https://raspberrypi.local:1880/ui**.

K routeru je dále připojeno pět vývojových desek ESP8266. Tato ESP jsou rozdělena do dvou skupin – první skupinou jsou ESP, která kontrolují aktuátory.

- LED – osvětlení celého pokoje
- Zephyr 40 MM – větrák – klimatizace celého pokoje
- Servo – hlavice topení

Druhou skupinou ESP, jsou ESP, do nichž jsou zapojeny senzory:

- DHT22 – snímá vlhkost a teplotu v pokoji
- MQ-9 – měří koncentraci CO v pokoji
- MQ-135 – měří koncentraci CO_2 v pokoji



Obrázek 14 - Schéma implementace
Zdroj: (autor)

Níže uvedená tabulka specifikuje jednotlivá zařízení, jež byla použita v implementaci.

Název zařízení	Počet	Účel
Raspberry Pi 3 B+	1	MQTT Broker, Node-RED
PC	1	Kali Linux
ESP8266	5	Mikrokontrolery pro senzory a aktuátory
LED	1	Osvětlení pokoje
Zephyr 40 MM	1	Klimatizace
Servo	1	Hlavice topení
DHT22	1	Senzor vlhkosti a teploty
MQ-9	1	Senzor měření plynů
MQ-135	1	Senzor měření kvality ovzduší

Tabulka 3 - Seznam zařízení
Zdroj: (autor)

5.3 Prototyp zapojení

Aby bylo možné celou implementaci otestovat a zapojit, byl vytvořen zjednodušený model této sítě. Model vychází ze schématu uvedeném na obrázku číslo 12. V podbodech nadpisu *Zapojení ESP8266 mikrokontrolerů* je u každého uvedeno schéma zapojení vytvořené v programu Fritzing[5].

5.4 Schéma témat

V kapitole o sémantice témat v sekci MQTT bylo zmíněno, že jednotlivá zařízení buď odebírají, či publikují do těchto témat (topics). Pro implementaci byla vytvořena struktura těchto témat. Pro přehlednost byl dodržován vzorec */podlaží/místnost/…* Pro tuto konkrétní testovací implementaci bylo zvoleno podlaží číslo tři a místnost/pokoj číslo sedm. Všechna témata budou tedy začínat *f3/r7/….* *f3* je jednoduše zkratka pro floor 3 a *r7* je zkratka pro room 7. Na tomto základu se dále stavěla témata. Aby bylo schéma co možná nejpřehlednější, byly vytvořeny skupiny.

5.4.1 States

Do této skupiny jsou posílány hodnoty prostředí v dané místnosti. Mezi tyto hodnoty patří: teplota, vlhkost, kvalita vzduchu a případná koncentrace hořlavých plynů, jako je například propan. Pro každý senzor bylo vyvozeno zvláštní téma.

f3/r7/states/humidity – vlhkost v místnosti

f3/r7/states/temperature – pro teplotu v místnosti

f3/r7/states/mq9/ratio – pro poměr $R_s(gas) / R_0$

f3/r7/states/mq9/alarm – pro zaslání zprávy „Limit exceeded“ v případě překonání hranice

f3/r7/states/mq9/co/ppm – pro zaslání koncentrace oxidu uhelnatého

[5] Fritzing je open-source hardwarová iniciativa. Nabízí sofware, díky kterému mohou uživatelé navrhovat své vlastní prototypy, sdílet je s ostatními, či navrhovat rozvržení desek plošných spojů (Fritzing, 2023).

f3/r7/states/mq9/lpg/ppm – pro zaslání koncentrace LPG

f3/r7/states/mq9/ch4/ppm – pro zaslání koncentrace CH_4

f3/r7/states/mq135/ppm – pro zasílání kvality ovzduší

f3/r7/states/mq135/alarm – pro zaslání zprávy „Limit exceeded“ v případě, překročení určené hranice

f3/r7/states/mq135/co2/ppm – pro zaslání koncentrace oxidu uhličitého

f3/r7/states/mq135/nh3/ppm – pro zaslání koncentrace NH_3

f3/r7/states/mq135/ethanol/ppm – pro zaslání koncentrace ethanolu

f3/r7/states/mq135/toluene/ppm – pro zaslání koncentrace toluenu

f3/r7/states/mq135/acetone/ppm – pro zaslání koncentrace acetonu

5.4.2 Controls

f3/r7/controls/lights – posílá se #*on* nebo #*off* pro provedení akce vypnutí / zapnutí osvětlení v dané místnosti

f3/r7/controls/vent – posílá se opět #*on* nebo #*off* = vypnutí / zapnutí větrání

f3/r7/controls/heating – posílá se hodnota, na kterou se má teplotní hlavice nastavit (0–5)

5.4.3 Scenes

Posledním tématem jsou scény. Scény reprezentují nastavení více zařízení najednou dle přednastavených profilů. Například zde může být scéna *Night*, která jednoduše zatáhne rolety, zhasne všechna světla a topení nastaví na příznivou teplotu ke spánku.

f3/r7/scenes/nazev_sceny

f3/r7/scenes – hodnota *#night* – vypne všechna světla a nastaví udržování teploty v místnosti na 22 °C[6] - to znamená polohu 3 na hlavici topení, větrání se nijak nemění

f3/r7/scenes – hodnota #*morning* – zapne světla v místnosti, zapne ventilaci

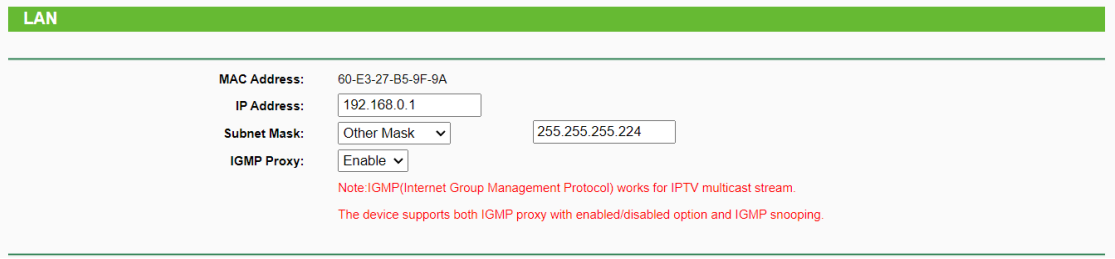
5.5 Nastavení routeru

Aby spolu mohla zařízení komunikovat skrz Wi-Fi, bylo třeba nakonfigurovat router. Byl použit router *tp-link TL-WR841N.* Na začátek je důležité zmínit, že hned po přihlášení do routeru byly změněny jeho přístupové údaje. Tyto údaje bývají v základu nastaveny na přihlašovací jméno: *admin* a heslo *admin*. V případě, že si uživatel tato základní hesla nezmění, dává tak případným útočníkům možnost snáze se dostat do nastavení jeho domácího routeru.

5.5.1 Nastavevní LAN

Jako první se na routeru nakonfigurovala LAN. IP adresa samotného routeru je 192.168.0.1 – to je tedy také takzvaná výchozí brána (default gateway). Maska sítě byla nastavena na 255.255.255.224 (/27). To znamená, že bez adresy rezervované pro broadcast (192.168.0.30) a adresy rezervované samotným routerem je zde prostor pro dalších 30 adres.

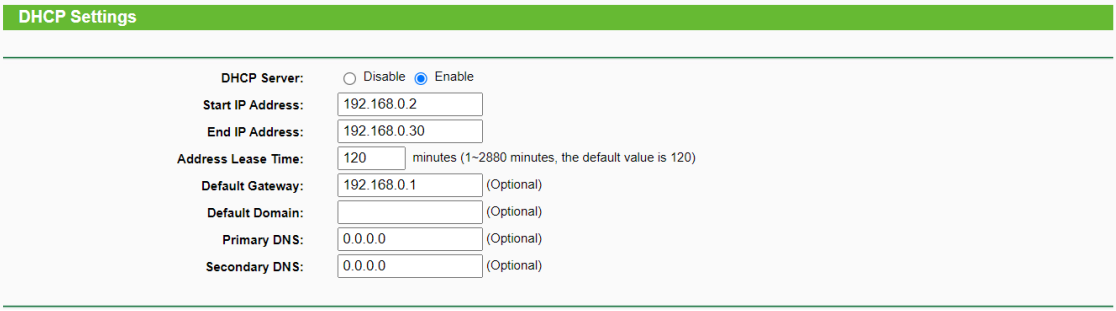
[6] Tato teplota byla nastavena dle normy ČSN EN 12831 (tzbinfo, 2023)



Obrázek 15 - Nastavení LAN routeru
Zdroj: (autor)

5.5.2 Nastavení DHCP serveru

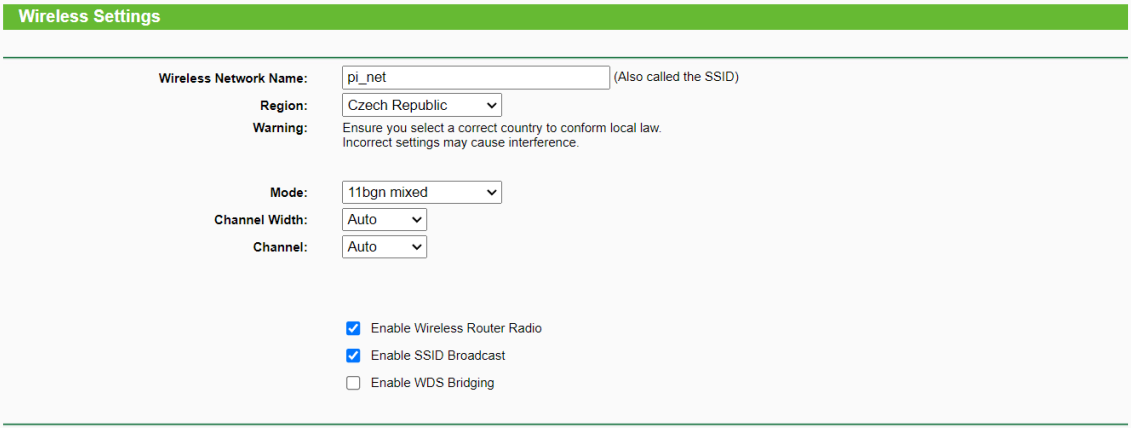
Jako druhý byl nastaven DHCP server, aby fungovalo dynamické přiřazování IP adres v síti. Rozsah, ve kterém DHCP server adresy přiděluje, byl 192.168.0.2 až 192.168.0.29.



Obrázek 16 - Nastavení DHCP routeru
Zdroj: (autor)

5.5.3 Nastavení bezdrátového připojení k routeru

Aby zařízení, jako ESP, či Raspberry Pi byla schopna přistupovat k routeru bezdrátově – skrze Wi-Fi, bylo třeba nakonfigurovat bezdrátové připojení. SSID bylo nastaveno na *pi_net* a heslo bylo vygenerováno pomocí generátoru. Typ zabezpečení byl nastaven na WPA/WPA2 – Personal. WPA2 využívá šifrování typu AES spolu s heslem, pomocí kterého se uživatel do sítě připojí.



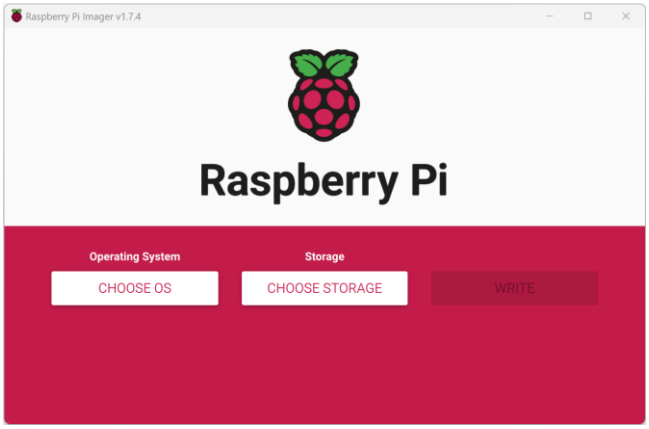
Obrázek 17 - Nastavení bezdrátového připojení
Zdroj: (autor)

5.6 Konfigurace Raspberry Pi 3 B+

5.6.1 Operační systém

Operační systém, který byl vybrán pro Raspberry Pi 3 B+ je Raspberry Pi OS Lite (64-bit). Jedná se o operační systém, který je odvozený od Debianu 11 (Bullseye), který neobsahuje desktop prostředí. Konkrétně tato verze systému je kompatibilní s Raspberry Pi 3/4/4000 (Raspberry Pi, 2023a).

Aby bylo možné operační systém na Raspberry Pi 3 B+ nainstalovat, byl použit nástroj Raspberry Pi Imager verze 1.7.4.



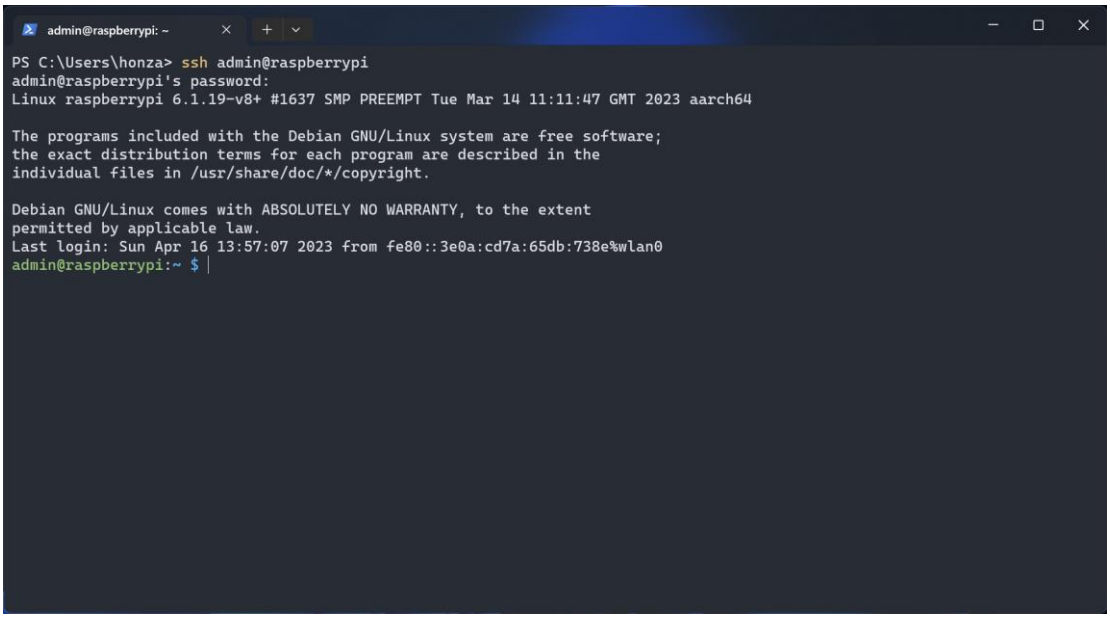
Obrázek 18 - Aplikace Raspberry Pi Imager
Zdroj: (autor)

Pomocí tohoto nástroje lze nainstalovat operační systém na SD kartu, která je následně vložena do samotného Raspberry Pi 3. Před nahráním systému –

stisknutím tlačítka WRITE lze nakonfigurovat užitečné věci, jako například hostname Raspberry Pi přístup skrze SSH[7], či bezdrátové připojení k síti. Pro tuto implementaci bylo hostname nastaveno na *raspberrypi*, byl povolen SSH přístup s uživatelským jménem admin a vygenerovaným, bezpečným heslem a byl nastaven bezdrátový přístup k síti nakonfigurované v předchozím odstavci – *pi_net*.

5.6.2 Základní nastavení

Po úspěšném zavedení operačního systému a „nabootování" bylo třeba Raspberry Pi aktualizovat. Toho bylo docíleno pomocí příkazů *sudo apt-get update && sudo apt-get* upgrade. K Raspberry Pi bylo po celou dobu přistupováno skrze nakonfigurovaný SSH přístup.



Obrázek 19 - SSH přístup pomocí PowerShell
Zdroj: (autor)

[7] SSH, je také známý pod pojmem Secure Shell. Jedná se o síťový protokol, který správcům umožňuje bezpečný přístup k zařízení (Michael Cobb & Peter Loshin, 2021).

5.6.3 Instalace Mosquitto

Eclipse Mosquitto je open source zprostředkovatel zpráv, který implementuje protokol MQTT ve verzích 5.0, 3.1.1. a 3.1. Mosquitto je tzv. lightweight, což znamená, že je vhodné pro použití na všech zařízeních – od úsporných jednodeskových počítačů po plnohodnotné servery (mosquitto.org, 2023a).

Z výše uvedeného textu vyplývá, že je možné nainstalovat Mosquitto na Raspberry Pi 3. Mosquitto bylo nainstalováno pomocí *sudo apt install -y mosquitto mosquitto clients*. Hned po instalaci bylo nastaveno, aby se MQTT broker spustil ihned po startu zařízení (*sudo systemctl enable mosquitto.service*).

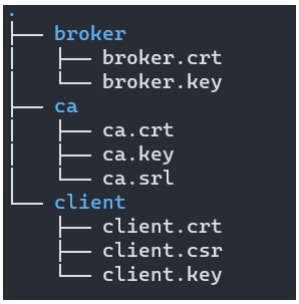
V rámci zabezpečení byly nastaveny přístupové údaje, které musí klienti, připojující se k MQTT brokeru znát. Toho je možné docílit pomocí příkazu: *sudo mosquitto_passwd -c /etc/mosquitto/passwd uzivatelske_jmeno*. V případě této implementace bylo uživatelské jméno nastaveno na *client*. Je samozřejmě možné, povolit přístup k MQTT brokeru bez jakýchkoliv přístupových údajů. To ale následně umožní případným útočníkům posílat nechtěné zprávy a ovládat celou implementaci skrze napadený broker.

5.6.4 Nastavení TLS/SSL pro Mosquitto

Jako výchozí port pro komunikaci je port 1883. Tento port však nepodporuje šifrované připojení skrze TLS/SSL. Existuje však další port – port 8883. Tento port šifrované připojení podporuje, avšak aby bylo možné ho použít, muselo být vygenerováno několik certifikátů a klíčů za pomoci nástroje OpenSSL.

Prvním krokem bylo vytvoření klíče a certifikátu takzvané CA (certificate authority). Certifikační autorita (CA) je entita, která má za úkol ověřovat identitu subjektů (webových stránek, e-mailových adres atd.) a vydávat digitálně podepsané certifikáty, které jsou vázané k veřejnému klíči serveru (překlad autora) (Aas et al., 2019). Dále bylo potřeba vygenerovat certifikáty a klíče pro samotný broker a následně pro klienty, kteří k tomuto brokeru budou přistupovat.

Je důležité podotknout, že se jedná o tzv. *self-signed* certifikáty, jelikož byla vytvořena vlastní CA, která tyto certifikáty „podepsala“. Alternativní možností by bylo využití například certifikační autority *Let's encrypt*.



Obrázek 20 - Stromová struktura složky s certifikáty
Zdroj: (autor)

Jak je z obrázku výše patrné, pro certifikáty a klíče byla vytvořena samostatná struktura, do které byly na Raspberry Pi uloženy. Konkrétní cesta je */home/admin/certs*.

Tato nastavení byla třeba zanést do *mosquitto.conf* souboru. Mosquitto.conf je konfigurační soubor, který Mosquitto využívá. Tento soubor může být umístěn kdekoliv odkud ho Mosquitto je schopné načíst. Ve výchozím nastavení Mosquitto nepotřebuje konfigurační soubor a jsou použity výchozí hodnoty (mosquitto.org, 2023a).

```
  GNU nano 5.4                          etc/mosquitto/mosquitto.conf
# Place your local configuration in /etc/mosquitto/conf.d/
#
# A full description of the configuration file is at
# /usr/share/doc/mosquitto/examples/mosquitto.conf.example

port 8883

cafile /home/admin/certs/ca/ca.crt
#capath /home/openest/certs/ca

# Path to the PEM encoded server certificate.
certfile /home/admin/certs/broker/broker.crt

# Path to the PEM encoded keyfile.
keyfile /home/admin/certs/broker/broker.key
require_certificate true

use_identity_as_username true

allow_anonymous false
password_file /etc/mosquitto/passwd
```

Obrázek 21 - Konfigurační soubor mosquitto.conf
Zdroj: (autor)

Port 8883 říká mosquittu, jaký port má použít, *cafile* odkazuje na soubor ca.crt (lze také využít cestu – *capath*). *Certfile* ukazuje na certifikát brokera – broker.crt a *keyfile* odkazuje na klíč brokera. *Require_certificate true* znamená, že client musí serveru předložit validní certifikát, pokud s ním chce komunikovat. Pokud je *use_identity_as_username* nasteveno na hodnotu *true* znamená to, že pro přístup bude využito Common Name (CN) klienta, namísto CN brokera. Poslední dvě položky – *allow_anonymous* a *password_file* pouze znamenají, že není povolen anonymní přístup, nýbrž musí klienti přistupovat k brokeru pomocí přihlašovacích údajů – ty jsou definované právě v *password_file* (mosquitto.org, 2023b).

5.6.5 Instalace firewall

Pro Linux existuje mnoho různých firewallů. Většina z nich používá balíček *iptables*, který se nachází nad linuxovým *netfiltering* systémem a je v Raspberry Pi OS integrovaný hned po instalaci. Problém je, že tento balíček není plně nakonfigurován. Samotná konfigurace iptables může být pro normální uživatele komplikovaná. Z toho důvodu byl pro tuto implementaci vybrán balíček *ufw (Uncomplicated Fire Wall)*. Jedná se o výchozí firewall v distribuci Ubuntu (Raspberry Pi, 2023b).

Pro zprovoznění firewallu ufw bylo třeba ho na zařízení nainstalovat pomocí příkazu *sudo apt install ufw*. Aby se firewall spouštěl hned po startu zařízení, byl použit příkaz *sudo ufw enable* (lze použít *sudo ufw disable* pro zakázání firewallu po startu). Aby fungoval Node-RED servis, SSH a Secure-MQTT, bylo třeba povolit tyto porty v konfiguraci ufw pomocí příkazů *sudo ufw allow 1880* pro Node-RED, *sudo ufw allow 8883* pro secure-mqtt a *sudo ufw allow ssh* pro povolení portu 22 - SSH. Naopak port 1883, který využívá nezabezpečené MQTT byl zakázán – *sudo ufw deny 1883*.

Jako poslední byl nakonfigurován limit přihlášení skrze SSH. Pokud se daná IP adresa pokusí připojit k brokeru minimálně šestkrát za posledních třicet sekund, bude přidána do listu blokovaných IP adres – *sudo ufw limit ssh/tcp*.

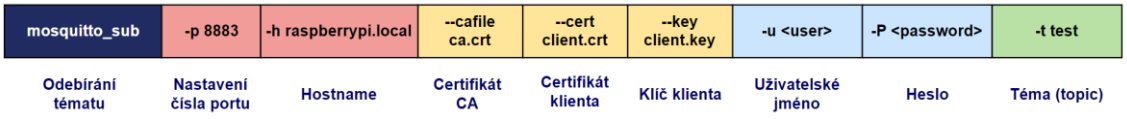
5.6.6 Testování Mosquitto

Po úspěšném nastavení TLS/SSL šifrování a mosquitto.conf konfiguračního souboru bylo třeba vše vyzkoušet. Pomocí příkazu *sudo systemctl restart mosquitto* bylo Mosquitto restartováno. Pro zjištění stavu brokera byl použit příkaz *sudo systemctl status mosquitto*. Toto vrátilo následující obrazovku:

```
● mosquitto.service - Mosquitto MQTT Broker
     Loaded: loaded (/lib/systemd/system/mosquitto.service; enabled; vendor preset: enabled)
     Active: active (running) since Mon 2023-04-17 12:38:47 BST; 20h ago
       Docs: man:mosquitto.conf(5)
             man:mosquitto(8)
    Process: 813 ExecStartPre=/bin/mkdir -m 740 -p /var/log/mosquitto (code=exited, status=0/SUCCESS)
    Process: 814 ExecStartPre=/bin/chown mosquitto /var/log/mosquitto (code=exited, status=0/SUCCESS)
    Process: 815 ExecStartPre=/bin/mkdir -m 740 -p /run/mosquitto (code=exited, status=0/SUCCESS)
    Process: 816 ExecStartPre=/bin/chown mosquitto /run/mosquitto (code=exited, status=0/SUCCESS)
   Main PID: 817 (mosquitto)
      Tasks: 1 (limit: 779)
        CPU: 1min 1.286s
     CGroup: /system.slice/mosquitto.service
             └─817 /usr/sbin/mosquitto -c /etc/mosquitto/mosquitto.conf
```

Obrázek 22 - Status Mosquitto MQTT Brokera
Zdroj: (autor)

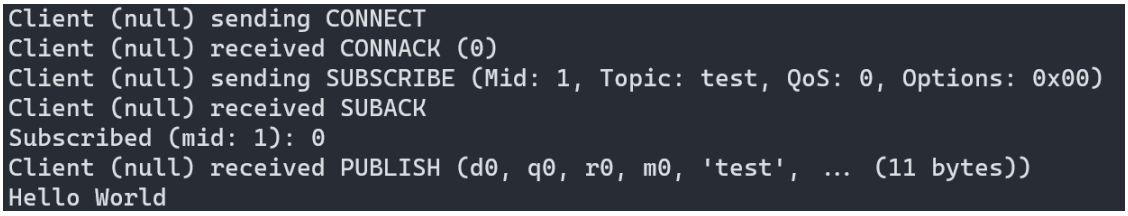
Pro otestování odebírání určitého tématu a následné publikování zprávy do tohoto tématu byly spuštěny dvě instance SSH přístupu – dvě okna v Microsoft Power Shell. V prvním okně byl zadán příkaz na odebrání tématu *test*:



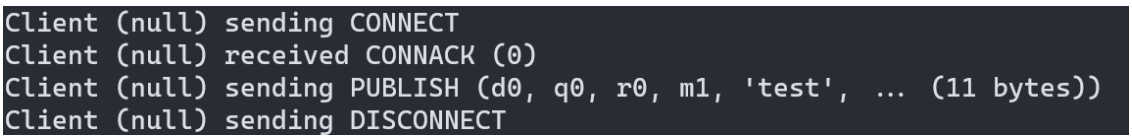
Obrázek 23 - Struktura zprávy mosquitto_sub
Zdroj: (autor)

Ve druhém okně byl použit příkaz *mosquitto_pub* pro publikování zprávy *„Hello World“* do tématu *test*:

Obrázek 24 - Struktura zprávy mosquitto_pub
Zdroj: (autor)



Obrázek 25 - Výsledek příkazu mosquitto_sub
Zdroj: (autor)



Obrázek 26 - Výsledek příkazu mosquitto_pub
Zdroj: (autor)

Obrázek 25 ukazuje, co vše se posílá – jsou zde zprávy CONNECT, CONNACK, SUBSCRIBE, SUBACK a při publikování do tématu PUBLISH. Na posledním řádku je hodnota, která byla poslána – *Hello World*.

To znamená, že vše bylo dobře nastaveno a MQTT Broker fungoval přes šifrovaný port 8333 bez jakýchkoliv problémů.

5.6.7 Instalace Node-RED

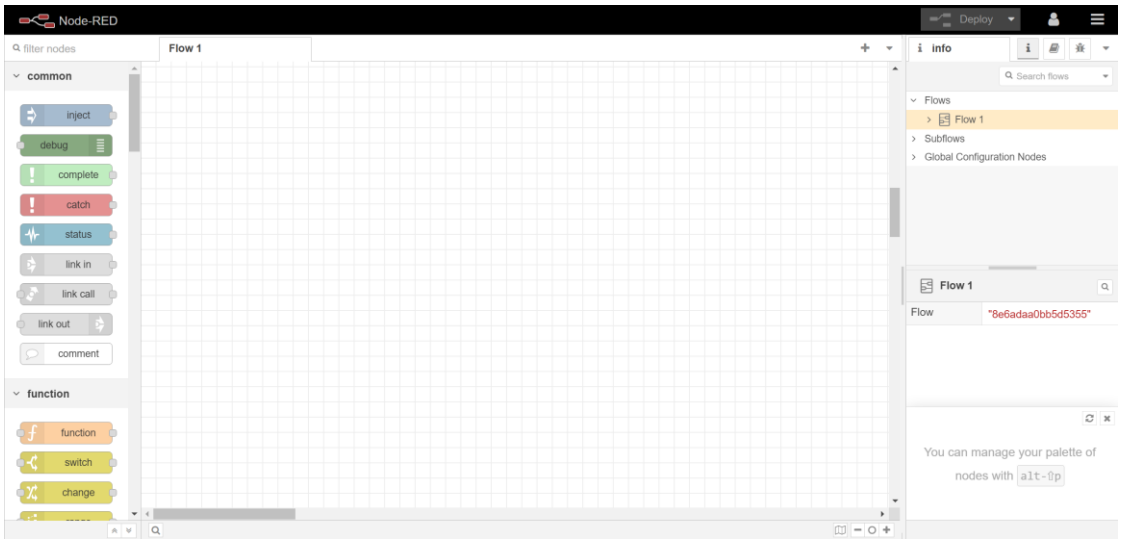
Node-RED je vizuální nástroj využívající paradigmat programování založeném na toku. Je postavený na Node.js, který se využívá pro vývoj systémů v IoT. V prostředí Node-RED může vývojář propojovat zařízení a online služby dle svého vlastního uvážení, přičemž vždy existuje více způsobů, jak stejný systém zapojit (Clerissi et al., 2018).

Nástroj Node-RED byl do Raspberry Pi nainstalován pomocí příkazu *bash <(curl -sL https://raw.githubusercontent.com/node-red/linux-installers/master/deb/update-nodejs-and-nodered)*. Při instalaci Node-RED bylo třeba nakonfigurovat několik věcí. Jednou z nich bylo například vytvoření konfiguračního souboru *settings.js*, či nastavení hesla a jména pro uživatele typu admin. Tyto přístupové údaje byly uloženy do souboru, který byl zašifrován pomocí heslové fráze.

Nakonec byl Node-RED spuštěn jako service – *node-red-start*. Aby se Node-RED spouštěl hned po startu zařízení, bylo toto třeba povolit pomocí příkazu *sudo systemctl enable nodered service* (Nodered.org, 2023).

```
● nodered.service - Node-RED graphical event wiring tool
     Loaded: loaded (/lib/systemd/system/nodered.service; enabled; vendor preset: enabled)
     Active: active (running) since Mon 2023-04-17 14:36:07 BST; 19h ago
       Docs: http://nodered.org/docs/hardware/raspberrypi.html
   Main PID: 1240 (node-red)
      Tasks: 11 (limit: 779)
        CPU: 1min 7.228s
     CGroup: /system.slice/nodered.service
             └─1240 node-red
```

Obrázek 27 - Status Node-RED
Zdroj: (autor)



Obrázek 28 - Prostředí Node-RED
Zdroj: (autor)

5.6.8 Zabezpečení Node-RED

Stejně jako Mosquitto, i Node-RED jde nastavit tak, aby využíval TLS/SSL šifrování. Lze u něj povolit HTTPS přístup. Toho je možné docílit v settings.js souboru služby Node-RED „odkomentováním“ řádků zobrazených na obrázku níže:

```
/** The following property can be used to enable HTTPS
 * This property can be either an object, containing both a (private) key
 * and a (public) certificate, or a function that returns such an object.
 * See http://nodejs.org/api/https.html#https_https_createserver_options_requestlistener
 * for details of its contents.
 */

/** Option 1: static object */
https: {
  key: require("fs").readFileSync('/home/admin/certs/client/client.key'),
  cert: require("fs").readFileSync('/home/admin/certs/client/client.crt')
},
```

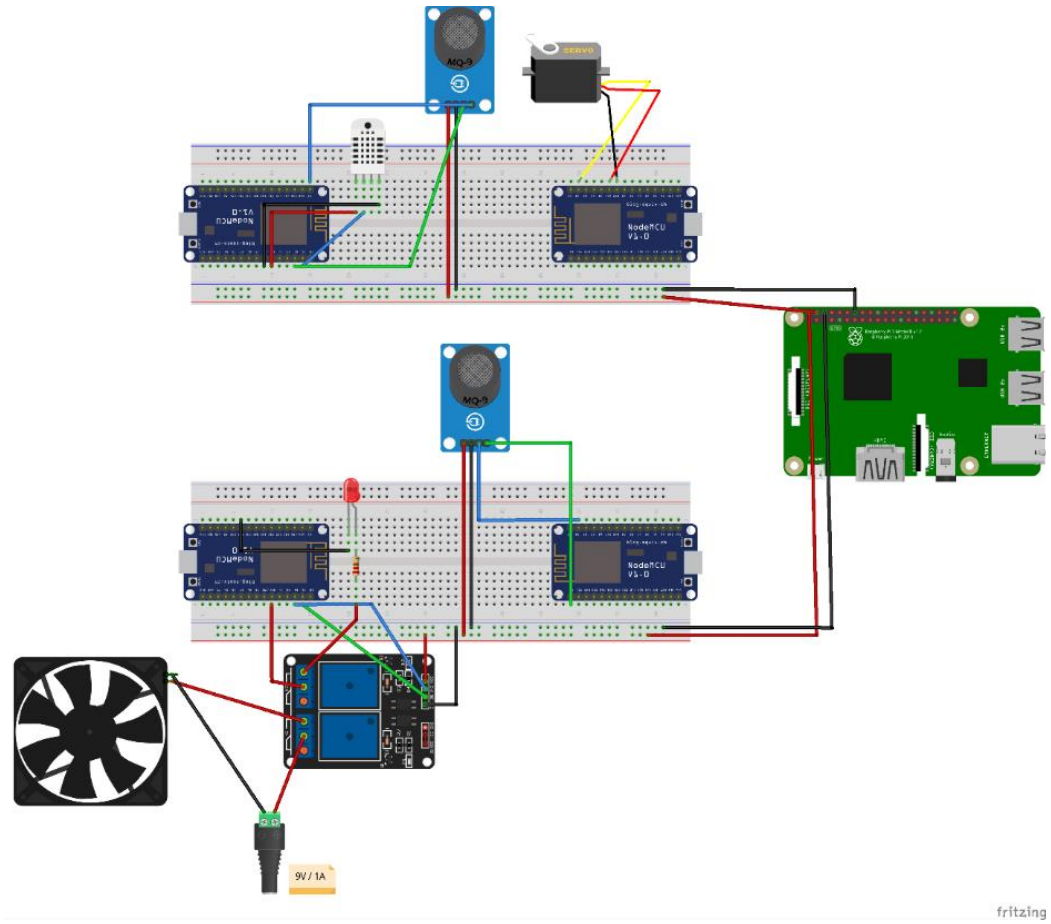
Obrázek 29 - HTTPS přístup k Node-RED
Zdroj: (autor)

Jako *key* a *cert* byly opět nastaveny soubory *client.key* a *client.crt*, jelikož Node-RED je vlastně klient přistupující k MQTT Brokeru. Zároveň přímo v prostředí Node-RED bylo nutné nahrát tyto dva soubory společně se souborem *ca.crt* do TLS konfigurace.

Posledním nutným krokem bylo nainstalování Dashboard palety do Node-RED. Paletu si lze představit jako soubor nástrojů, které obohatí Node-RED o další funkcionality – v tomto případě o prvky dashboard (tlačítka, grafy, …).

5.7 Konfigurace ESP8266 mikrokontrolerů

Celé schéma zapojení je patrné z obrázku níže. Stejně jako ostatní schémata, také toto bylo vytvořeno pomocí nástroje Fritzing. Zařízení jsou propojena skrze nepájivá pole, známá také jako *breadboards*. Raspberry Pi 3 zde slouží nejen jako broker, ale také jako 5V napájení takzvaných *power rails* každého nepájivého pole. Je to z toho důvodu, že ESP8266 je schopné poskytnout napětí 3V, avšak senzory MQ-9 a MQ-135 mají pracovní napětí 5V. Větrák je napájen z externího zdroje 9 V/1 A adaptérem. LED i větrák je ovládaný skrze relé. LED by se dala zapojit i bez pomoci relé, ale jelikož si tato práce klade za cíl vytvořit prototyp reálné implementace, je třeba využívat postupy z reálné praxe a dle předpisů.



Obrázek 30 - Schéma zapojení
Zdroj: (autor)

5.7.1 Základní konfigurace ESP8266 v Arduino IDE

ESP8266 je levný Wi-Fi mikročip se zabudovanou podporou TCP/IP vyvinutý společností Espressif Systems (Espressif.com, 2023). Tento mikročip je jedním z hlavních zařízení celé implementace. Samotná implementace obsahuje čtyři tato zařízení.

Obrázek 31 - EPS8266
Zdroj: (LaskaKit, 2023)

Mikrokontrolery ESP byly programovány pomocí nástroje Arduino IDE:

1. Jedná se o open-source software, to znamená, že uživatelé, kteří jej používají, mají přístup ke zdrojovému kódu a mohou ho dle svých potřeb upravovat.
2. Arduino „jazyk“ je vlastně pouze rozšířený jazyk C++.
3. Obsahuje rozsáhlou knihovnu již předepsaného / předpřipraveného kódu, která uživateli usnadní například práci s čidly.
4. Je kompatibilní mezi platformami – kód lze vyvíjet na Windows, Linux, či MacOS.
5. Má vestavěnou sériovou komunikaci – to umožňuje vývojové desce komunikovat s počítačem skrze USB.
6. Podporuje PWM (pulzně šířková modulace). Jedná se o vestavěnou funkci vývojových desek, jež umožňuje přesné řízení množství energie dodávané do zařízení, jako například LED (Siddhesh Shinde, 2023).

Určitou část kódu zařízení ESP sdílejí, jelikož se jedná o konfiguraci TLS, Wi-Fi a přístupu k MQTT brokeru. Pro TLS jsou zde čtyři konstanty – *ca_cert[]*, *client_key[]*, *client_cert[]* a *fingerprint*. Jedná se o certifikáty, klíč a SHA-1 fingerprint zařízení.

Díky těmto konstantám je ESP schopné navázat bezpečnou komunikaci skrze TLS. Dále je zde *SSID* a heslo pro přístup k Wi-Fi. Aby se ESP bylo schopné připojit k MQTT brokeru, nesmí zde chybět *mqtt_broker* – hostname Raspberry Pi 3, *mqtt_username* – uživatelské jméno, *mqtt_password* – heslo a *mqtt_port* – port, přes který bude komunikace navázána – 8883. Pro připojení k Wi-Fi je vytvořena instance *WiFiClientSecure espClient* a pro ovládání posílání a přijímání zpráv pomocí MQTT je zde instance *PubSubClient*. Základní funkce – *void setup()* tedy vypadá následovně:

```
122  void setup() {
123    // Inicializace sériové komunikace
124    Serial.begin(115200);
125
126    // Načtení certifikátu CA
127    BearSSL::X509List cert(ca_cert);
128    espClient.setTrustAnchors(&cert);
129
130    // Načtení certifikátů klienta a klíče klienta
131    BearSSL::X509List client_crt(client_cert);
132    BearSSL::PrivateKey key(client_key);
133    espClient.setClientRSACert(&client_crt, &key);
134
135    // Nastavení momentálního času - nutné pro TLS připojení
136    time_t t = now();
137    espClient.setX509Time(t);
138
139    // Nastavení fingerprintu
140    espClient.setFingerprint(fingerprint);
141
142    // Inicializace Wi-Fi
143    WiFi.begin(ssid, password);
144
145    while (WiFi.status() != WL_CONNECTED) {
146        delay(500);
147        Serial.println("Connecting to WiFi..");
148    }
149
150    Serial.println("Connected to the WiFi network");
151
152    // Nastavení serveru MQTT
153    client.setServer(mqtt_broker, mqtt_port);
154
155    // Nastavení callbacku - funkce, která se stará o zpracování zpráv
156    client.setCallback(callback);
157
158
159    // Smyčka pro připojení k MQTT brokeru
160    while (!client.connected()) {
161        String client_id = "ESP1-";
162        client_id += String(WiFi.macAddress());
163        Serial.printf("The client %s connects to the public mqtt broker\n", client_id.c_str());
164        if (client.connect(client_id.c_str(), mqtt_username, mqtt_password)) {
165            Serial.println("Connected to MQTT broker!");
166        } else {
167            Serial.print("Failed with state ");
168            Serial.print(client.state());
169            delay(2000);
170        }
171    }
```

Obrázek 32 - Smyčka funkce setup
Zdroj: (autor)

Stejná smyčka je použita u všech zařízení. Každé zařízení ESP samozřejmě odebírá jiná témata a obsahuje odlišné konstanty označující například GPIO na desce.

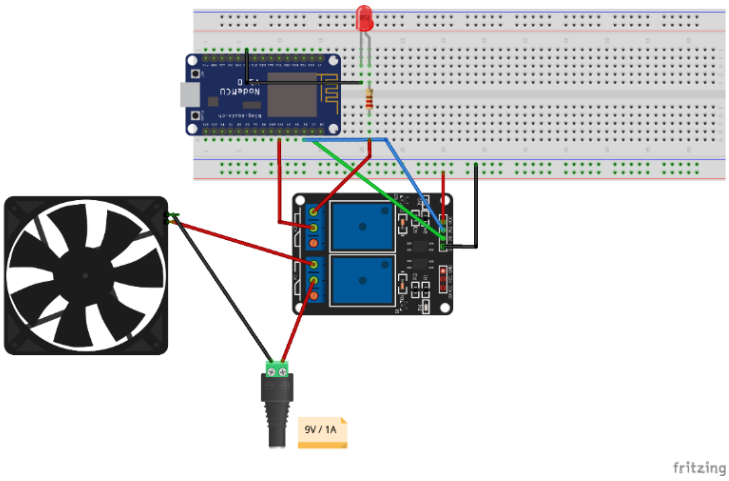
Podstatné je také zmínit funkci *callback*. Ta bere jako vstupní parametry *topic* – téma, *payload* – náklad a length – délku. Tato funkce má na starost vyhodnocování přijatých zpráv skrze MQTT.

```
void callback(char *topic, byte *payload, unsigned int length) {}
```

Obrázek 33 - Funkce callback
Zdroj: (autor)

5.7.2 Konfigurace LED a Zephyr 40 MM

První ESP mikrokontroler, který byl konfigurován, byl ten, k němuž je připojené osvětlení (LED) a ventilace (Zephyr 40MM větráček).



Obrázek 34 - Schéma zapojení LED a Zephyr 40 MM
Zdroj: (autor)

Jak je z obrázku výše patrné, zařízení (LED, větrák) jsou připojena přes relé. K tomu, aby se rozsvítila LED dioda, či roztočil větrák, je třeba přepnout relé. Obě zařízení jsou připojena do pozice NO (normally open). Při testování spínání relé bylo zjištěno, že je obtížné relé přepnout napětím 3V, proto je napájeno přímo z Raspberry Pi 3, které umí poskytnout napětí 5V.

Kompletní kód je přiložen v příloze bakalářské práce a skládá se z několika podmínek umístěných právě ve funkci *callback*. Tyto podmínky reagují na přijaté zprávy z různých témat, které dané ESP odebírá. V tomto případě EPS odebírá témata *f3/r7/controls/lights*, *f3/r7/controls/vent* a *f3/r7/scenes*.

```
// Pro téma f3/r7/controls/lights
if (strcmp(topic, lights_topic) == 0) {

  if (message == "#on"){
    digitalWrite(REL_LED, LOW);
  }
  if (message == "#off"){
    digitalWrite(REL_LED, HIGH);
  }

}
```

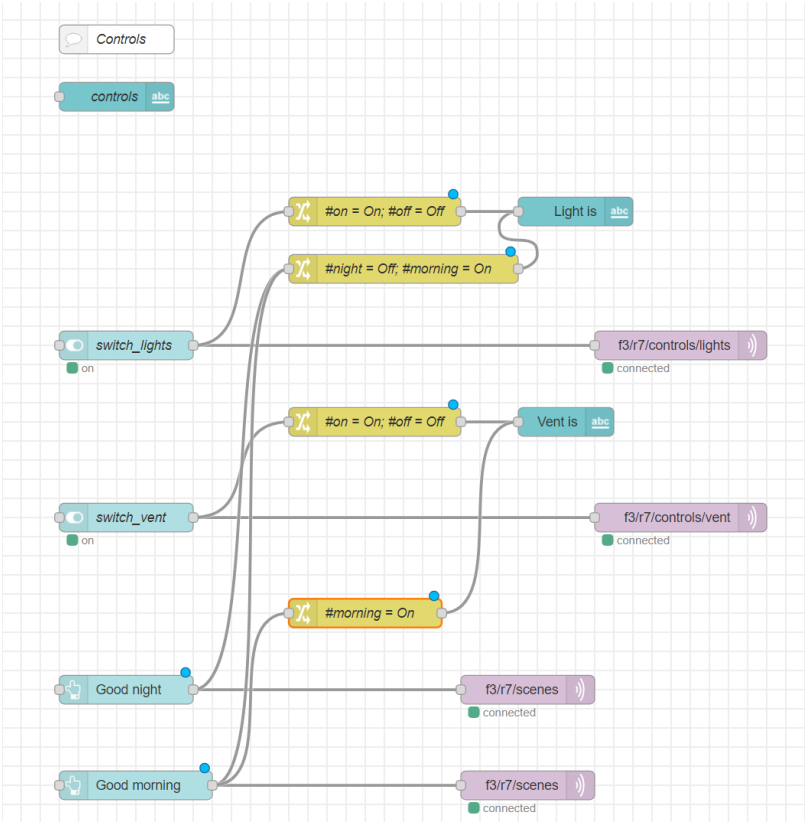
```
// Pro téma f3/r7/controls/vent
if (strcmp(topic, vent_topic) == 0) {

  if (message == "#on"){
    digitalWrite(REL_VENT, LOW);
  }
  if (message == "#off"){
    digitalWrite(REL_VENT, HIGH);
  }

}
```

Obrázek 35 - Podmínky pro LED a Zephyr
Zdroj: (autor)

V prostředí Node-RED jsou LED a větrák kontrolovány pomocí přepínacích tlačítek (switch) – světle modré obdélníky. Po přepnutí se posílá zpráva do patřičného tématu – fialové obdélníky. Schéma Node-RED obsahuje žluté obdélníky. Jedná se o takzvané *change nodes*. Tyto uzly mají na starost měnění nákladu zprávy MQTT a následné nastavování textu, který vidí uživatel ve své dashboard.

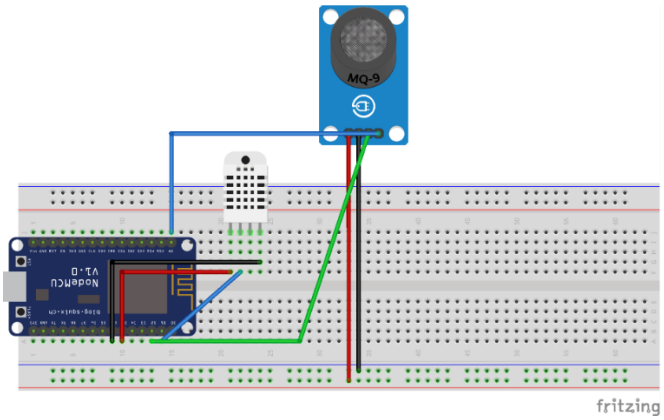


Obrázek 36 - Schéma LED a Zephyr – Node-RED
Zdroj: (autor)

Skrze dashboard může uživatel vypínat a zapínat osvětlení v místnosti. Zároveň může ovládat větrání v daném pokoji. Dashboard mu zobrazuje stav těchto dvou zařízení, a to v podobě textového pole. Spolu s přepínacími tlačítky byly nakonfigurovány také dvě testovací scény. Scéna *Good night*, která po spuštění vypne osvětlení místnosti, nýbrž větrání nechá zapnuté. Scéna *Good morning*, která po spuštění uvede do provozu jako osvětlení, tak větrání v místnosti (pokud bylo z nějakého důvodu vypnuto).

5.7.3 Konfigurace DHT22 a MQ-9

Jako druhé bylo konfigurované ESP se senzory DHT22 a MQ-9. Senzor DHT22 slouží k měření vlhkosti a teploty, zatímco senzor MQ-9 slouží k detekci úniku jedovatých plynů v místnosti – oxidu uhelnatého, metanu a LPG. Senzor DHT22 je napájen přímo z ESP, jelikož mu stačí pouze 3V napětí. Na druhou stranu, senzor MQ-9 vyžaduje provozní napětí 5V. Obrázek níže ilustruje, jakým způsobem byly tyto dva senzory k ESP mikrokontroleru připojeny.



Obrázek 37 - Schéma zapojení MQ-9 a DHT22
Zdroj: (autor)

Senzor DHT22 není těžké zprovoznit, již existuje předepsaná knihovna, díky které lze jednoduše načítat naměřené hodnoty vlhkosti a teploty. Složitější však bylo číst hodnoty ze senzoru MQ9.

Proces nastavení tohoto senzoru se skládal ze dvou kroků. Nejdříve bylo třeba senzor zkalibrovat pomocí prvního programu – nažhavit ho po dobu alespoň 15

minut v prostředí čistého vzduchu. Díky tomu byla získána konstanta R_0, jež se vložila do druhého programu, určeného již pro konkrétní měření (Amjad Ali, 2023).

Pro získání konstanty R_0 bylo nejdříve třeba vytvořit průměr ze 100 měření. Z tohoto průměru bylo vypočteno napětí senzoru dle rovnice:

$$sensor\_voltage = \left(\frac{sensor\_value}{1024}\right) \cdot 5.0$$

Rovnice 1 - Výpočet napětí senzoru MQ-9
Zdroj: (Amjad Ali, 2023)

Dále byl vypočten odpor v čistém vzduchu $R_s(air)$:

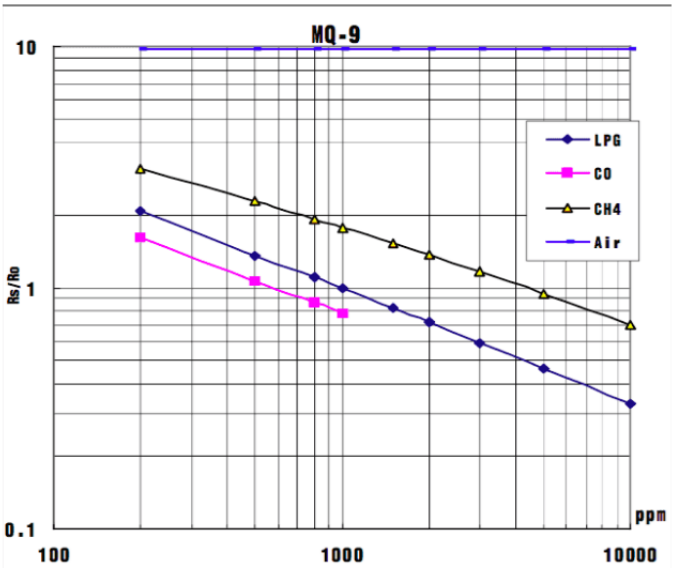
$$R_s(air) = \frac{(5.0 - sensor\_voltage)}{sensor\_voltage}$$

Rovnice 2 - Výpočet odporu v čistém vzduchu
Zdroj: (Amjad Ali, 2023)

Nakonec byla získána samotná konstanta R_0. 9.9 je hodnota vyčtená z grafu a odpovídá křivce vzduchu.

$$R_0 = \frac{R_s(air)}{9.9}$$

Rovnice 3 - Výpočet konstanty R_0
Zdroj: (Henan Hanwei Electronics Co., Ltd, b.r.)



Obrázek 38 - Graf hodnot MQ-9
Zdroj: (Henan Hanwei Electronics Co., Ltd, b.r.)

Konstanta R_0 vyšla po kalibraci na hodnotu **1.02**.

Druhý program již pouze využije tuto konstantu, opět vypočítá odpor v daném plynu $R_s(gas)$ a na závěr určí a pošle hodnotu poměru těchto dvou hodnot.

$$ratio = \frac{R_s(gas)}{R_0}$$

Rovnice 4 - Poměr odporu v daném plynu a konstanty R0
Zdroj: (Amjad Ali, 2023)

Obrázek níže ukazuje výstřižek kódu pro kalibraci senzoru MQ-9 a následné čtení jeho hodnot. Je patrné, že všechny výpočty vychází z výše uvedených rovnic.

```
// Cyklus pro načtení 100 hodnot
for (int x = 0; x < 100; x++) {
  sensorValue = sensorValue + analogRead(A0);
}
sensorValue = sensorValue / 100.0;

sensor_volt = (sensorValue / 1024) * 5.0;    // Výpočet napětí senzoru
RS_air = (5.0 - sensor_volt) / sensor_volt;  // Výpočet oporu v čistém vzduchu
R0 = RS_air / 9.9;                           // Výpočet konstanty R0
```

Obrázek 39 - Kalibrace MQ-9
Zdroj: (autor)

```
// Načtení hodnot senzoru MQ-9
int sensorValue = analogRead(A0);
alarm = digitalRead(MQ9_D);

// Výpočet napětí senzoru
sensor_volt = ((float)sensorValue / 1024) * 5.0;

// Výpočet odporu v daném plynu
RS_gas = (5.0 - sensor_volt) / sensor_volt;

// Výpočet poměru Rs_gas a R0
ratio = RS_gas / R0;
```

Obrázek 40 - Načtení hodnot MQ-9
Zdroj: (autor)

Do logiky senzoru MQ-9 byla také vložena podmínka, která v případě poklesu poměru $R_s(gas)/R_0$ na hodnotu 1.6 a méně vypočítá přibližnou koncentraci oxidu uhelnatého a tuto hodnotu zašle skrze MQTT. Stejné podmínky byly vytvořeny i pro ostatní plyny, které je tento senzor schopný změřit – LPG a CH_4.

```
// Pro výpočet přibližné koncentrace CO
if (ratio < 1.6) {
  ratio = ratio * 100;
  int co_ppm = map(ratio, 160, 79, 200, 1000);

  client.publish(states_mq9_co_ppm, String(co_ppm).c_str());
}
```

Obrázek 41 - Výpočet koncentrace CO (ppm)
Zdroj: (autor)

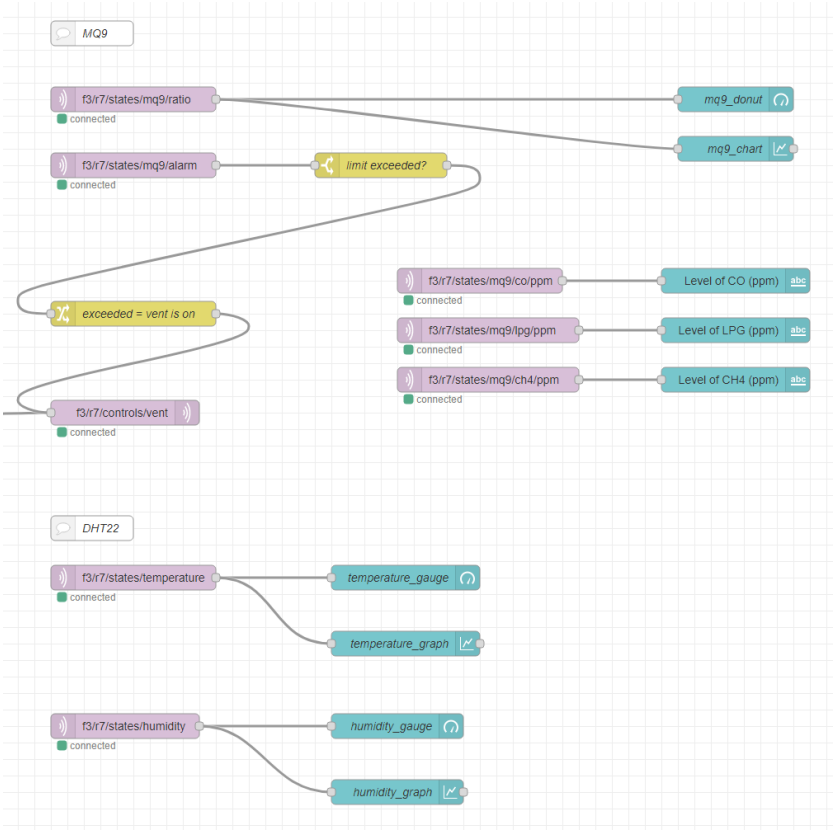
Co se týče digitálního pinu senzoru MQ-9, po překročení nastavené hranice potenciometrem, je zaslána zpráva – *Limit exceeded* (limit překročen).

Čtení hodnot ze senzoru DHT22 je velmi prosté:

```
// Načtení hodnot senzoru DHT22
float humidity = dht.readHumidity();
float temperatureC = dht.readTemperature();
```

Obrázek 42 - Načtení hodnot DHT22
Zdroj: (autor)

V prostředí Node-RED vypadá flow diagram pro tyto dva senzory následovně:



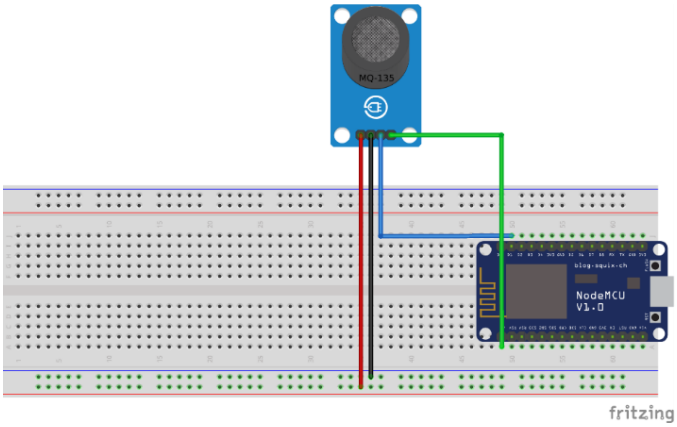
Obrázek 43 - Node-RED diagram pro DHT22 a MQ-9
Zdroj: (autor)

V dashboard vidí uživatel poměr $R_s(gas)/R_0$, teplotu místnosti, vlhkost v místnosti a příslušné grafy – vlhkost během 24 hodin, teplotu během 24 hodin a poměr $R_s(gas)/R_0$ během 24 hodin. Jsou zde také textové prvky, které ukazují, v případě zjištění senzorem, koncentraci jednotlivých plynů v ppm (parts per million).

5.7.4 Konfigurace MQ-135

Jako předposlední bylo konfigurováno ESP se senzorem MQ-135.

„Senzor plynů MQ135 je vstupní modul pro Arduino. Tento modul obsahuje senzor MQ-135, který dokáže detekovat více skupin plynů ovlivňujících kvalitu ovzduší. Senzor reaguje nejvíce na amoniak (NH_3), oxidy dusíku (NO_x), benzen, kouř a oxid uhličitý (CO_2). Aktivní prvek tohoto senzoru je tenká vrstva oxidu cíničitého (SnO_2), jejíž odpor se mění s koncentrací zmíněných plynů.“ (Luboš M., b.r.)



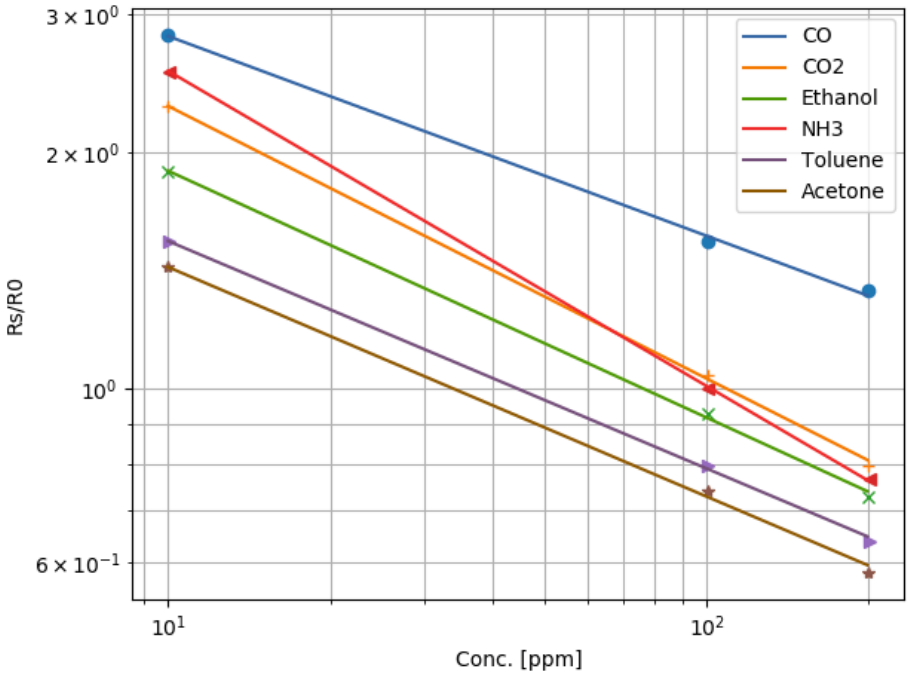
Obrázek 44 - Schéma zapojení MQ-135
Zdroj: (autor)

Senzor MQ-135 funguje na dost podobném principu, jako senzor MQ-9. Obsahuje jeden analogový pin, který posílá signál, jenž představuje koncentraci měřitelných plynů v ovzduší. Zároveň obsahuje digitální pin, který reaguje v případě překročení hranice nastavené potenciometrem na samotném senzoru.

Koncentrace v ppm	Stav ovzduší
0–50	Skvělý
51-100	Dobrý
100-150	Nezdravý pro citlivé jedince
151-200	Nezdravý
201-300	Velmi nezdravý
301-500	Nebezpečný

Tabulka 4 - Stav ovzduší dle koncentrace
Zdroj: (SHOLA USHA RANI et al., 2020)

Hodnoty jednotlivých plynů byly počítány stejným způsobem jako u senzoru MQ-9 – tzn. aplikovala se rovnice $ratio = \frac{R_s(gas)}{R_0}$. Výsledné hodnoty a koncentrace lze vyčíst z grafu:



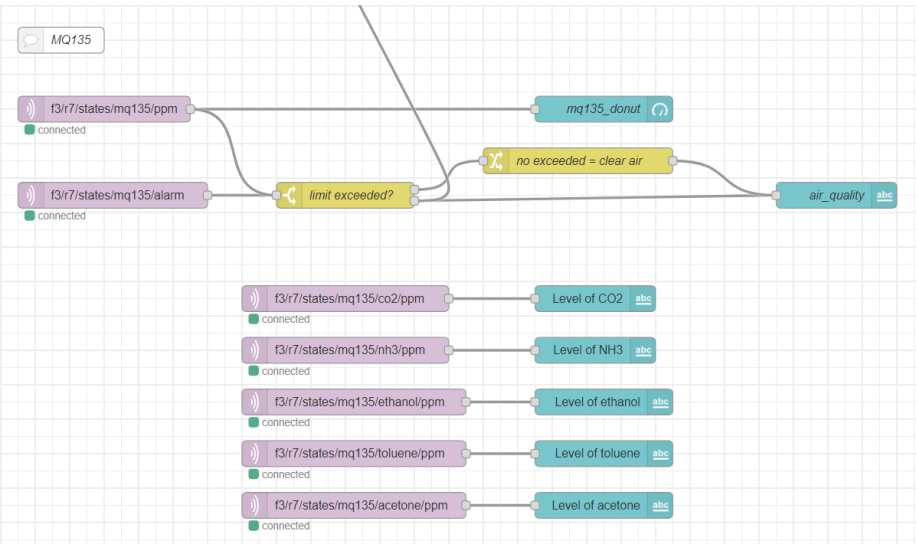
Obrázek 45 - Graf hodnot MQ-135
Zdroj: (Alexander Bessman, 2021)

Co se týče kódu, je téměř totožný s kódem pro senzor MQ-9. Jelikož pro tento senzor byla využita knihovna, nějaké části si lze usnadnit – například vypočítání R_0, či samotné měření. Například pro zjištění ppm existuje metoda *getPPM()*. Níže je uvedená podmínka, která vypočítává koncentraci CO_2 v místnosti.

```
// Pro výpočet přibližné koncentrace CO2
if (ratio < 2.2) {
  ratio = ratio * 100;
  int co2_ppm = map(ratio, 220, 80, 10, 110);

  client.publish(states_mq135_co2_ppm, String(co2_ppm).c_str());
}
```

Obrázek 46 - Výpočet koncentrace CO_2
Zdroj: (autor)

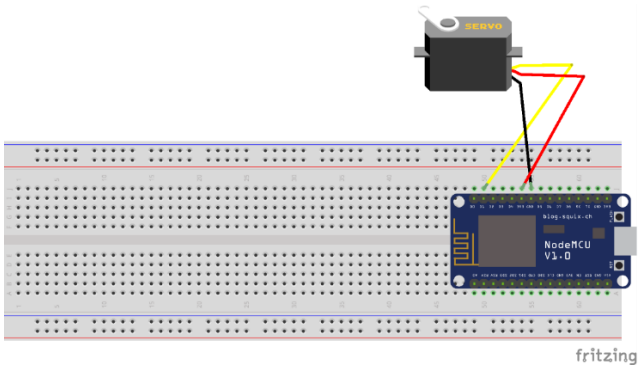


Obrázek 47 - Node-RED schéma MQ-135
Zdroj: (autor)

Na dashboard uživatel uvidí ukazatel kvality ovzduší, celkové ppm naměřené senzorem v podobě „donutu“ – typ ukazatel v dashboard Node-RED. Dále také uvidí koncentraci jednotlivých plynů, které je senzor schopný změřit.

5.7.5 Konfigurace serva

Jako poslední bylo konfigurované ESP, které má na starost ovládání serva. Díky tomu lze ovládat hlavici topení a nastavovat tak teplotu v místnosti.



Obrázek 48 - Schéma zapojení serva
Zdroj: (autor)

Servo bylo naprogramováno tak, že odebírá témata *f3/r7/controls/heating* a *f3/r7/scenes*. Do tématu *heating* je posílána hodnota od nuly do pěti. Tato hodnota je čtena z prvku *slider*, který je umístěný v uživatelské dashboard. Hodnota, kterou si uživatel nastaví, je převedena z rozsahu 0-5 na rozsah 0-180 a následně na to je nastavena poloha serva právě na takto „přemapovanou“ hodnotu uživatele.

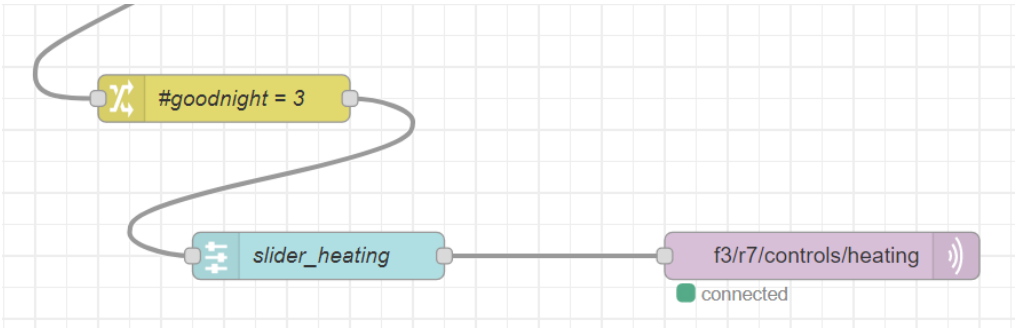
```
// Pro téma f3/r7/controls/heating
if (strcmp(topic, heating_topic) == 0) {
  int new_pos = message.toInt();
  // Přemapování hodnoty z rozsahu 0-5 na rozsah 0-180
  pos = map(new_pos, 0, 5, 0, 180);
  myservo.write(pos);
  delay(15);
}

// Pro téma f3/r7/scenes
if (strcmp(topic, scenes_topic) == 0) {

  if (message == "#night") {
    pos = 3;
    // Přemapování hodnoty z rozsahu 0-5 na rozsah 0-180
    pos = map(pos, 0, 5, 0, 180);
    myservo.write(pos);
  }
  // V případě morning se nic neděje
  if (message == "#morning") {
    return;
  }
}
```

Obrázek 49 - Ovládání serva
Zdroj: (autor)

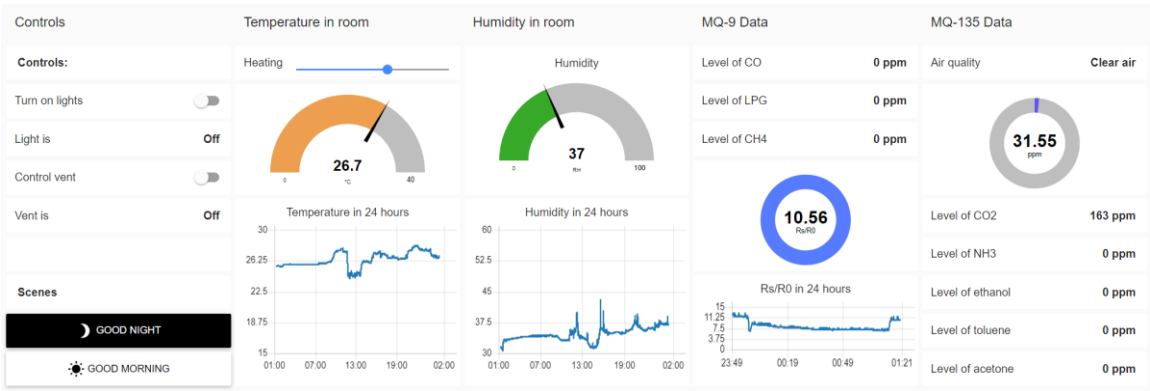
Co se týče diagramu v Node-RED. Slider, který ovládá servo, je přímo napojený na MQTT-out uzel, jež publikuje hodnoty slideru do daného tématu. Na slider je napojena change node, která se stará o nastavení hodnoty slideru na 3, v případě zavolání scény *Good Night*.



Obrázek 50 - Node-RED schéma serva
Zdroj: (autor)

5.7.6 Node-RED Dashboard

Samotná dashboard pro uživatele tedy vypadá následovně.



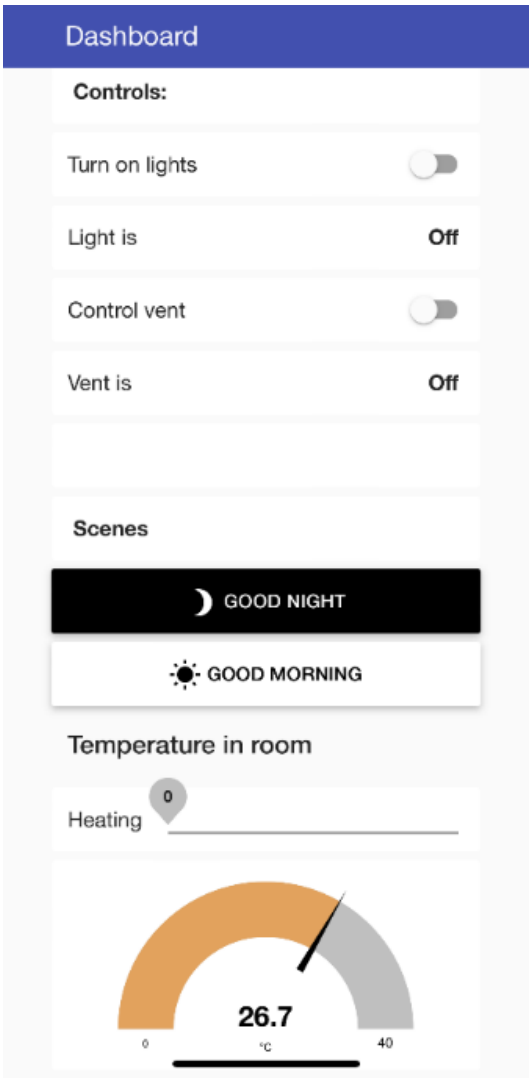
Obrázek 51 - Dashboard uživatele
Zdroj: (autor)

Dashboard je rozdělená do několika skupin. Skupina *Controls* obsahuje ovládání ventilace a osvětlení, spolu se scénami. Skupina *Temperature in room* obsahuje ovládání topení, ukazatel teploty a graf, na kterém lze vidět vývoj teploty za posledních 24 hodin. Skupina *Humidity in room* obsahuje ukazatel vlhkosti a opět graf. Předposlední skupina *MQ-9 Data* obsahuje koncentraci jednotlivých plynů – CO, LPG a CH_4 v ovzduší, ukazatel $R_s(gas)/R_0$ a graf vývoje $R_s(gas)/R_0$ za

posledních 24 hodin. Poslední skupina – *MQ-135 Data*, ukazuje kvalitu ovzduší, ukazatel ppm a koncentrace jednotlivých plynů v ovzduší.

Za dashboard se ještě skrývají automatizace, které uživatel neovlivní. Bylo zde například nastaveno, že pokud se objeví koncentrace kteréhokoliv z nebezpečných plynů v ovzduší, ihned se zapne ventilace. Dalším příkladem automatizace je sepnutí větrání při překonání teploty 27°C.

Dashboard lze zobrazit také na mobilním zařízení. Viz obrázek níže.

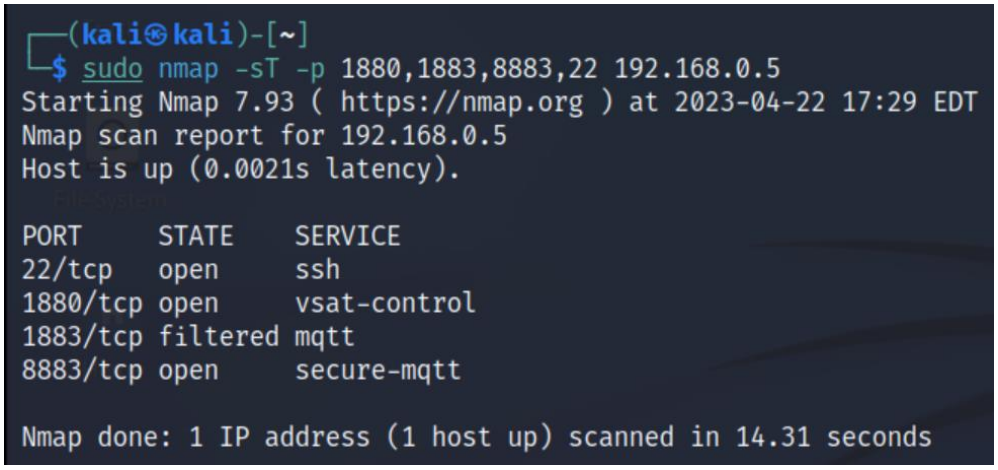


Obrázek 52 - Mobilní verze dashboard
Zdroj: (autor)

5.8 Jednoduchý sken sítě pomocí Kali Linux

Operační systém Kali Linux je linuxová distribuce založená na Debianu a je určená pro práci v oblasti informační bezpečnosti. Pomocí Kali Linux lze provádět úkony, jako jsou například penetrační testování, forenzní analýza, či reverzní inženýrství (Kali.org, 2023).

Sken byl proveden pomocí nástroje *Nmap*. Byly skenovány porty 1880 (Node-RED), 1883 (MQTT), 8883 (SMQTT) a 22 (SSH). Jak je na obrázku níže vidět, všechny tyto porty, až na port 1883, jsou otevřené. *Filtered* u portu 1883 znamená, že port blokuje firewall, tudíž nástroj Nmap není schopný zjistit, jestli se jedná o otevřený, či uzavřený port.



Obrázek 53 - Výsledek skenu sítě
Zdroj: (autor)

6 Shrnutí výsledků

V rámci této práce byla podrobně popsána problematika oblasti Internetu věcí a jeho architektury. Dále byly popsány hlavní aplikační domény IoT, a to Smart Health, Smart Industry a Smart Energy. Součástí práce bylo také představení momentálně nejpoužívanějších komunikačních protokolů v IoT, jako je MQTT, CoAP a AMQP. Tyto protokoly byly analyzovány z hlediska zabezpečení, načež bylo provedeno jejich porovnání. Důraz byl kladen na protokol MQTT. Byla detailně popsána struktura zpráv, témat a zabezpečení tohoto protokolu. Zároveň byly popsány jeho výhody oproti ostatním protokolům. Tento protokol byl následně využit v samotné implementaci, jako hlavní komunikační protokol.

Součástí práce byla také bližší analýza zabezpečení protokolů a popis protokolu TLS/SSL. Bylo zjištěno, že využitím šifrované komunikace, kterou protokol TLS/SSL nabízí, lze účinně zvýšit bezpečnost přenosu dat.

V části implementace byla vytvořena simulace IoT sítě pomocí Raspberry Pi 3, mikrokontrolerů ESP8266 spolu se senzory a aktuátory. Tyto mikrokontrolery byly naprogramovány pomocí jazyka C++. Zároveň byla vytvořena jednoduchá dashboard v prostředí Node-RED, která umožňuje uživateli ovládat osvětlení v místnosti, měřit a měnit teplotu a sledovat důležité údaje o stavu místnosti, jako je vlhkost, kvalita ovzduší, či koncentrace nebezpečných plynů.

Na závěr práce byl spuštěn jednoduchý sken sítě pomocí nástroje *Nmap* implementovaného v operačním systému Kali Linux.

7 Závěry a doporučení

Během psaní práce a implementace sítě se vyskytlo několik nepředvídatelných situací, které do určité míry ovlivnily plynulost procesu.

K první z těchto situací došlo při konfigurování TLS/SSL zabezpečení. Byla vybrána možnost *„self-signed"* certifikátů – tedy vytvořen vlastní certifikační autority a následné podepsání certifikátů a klíčů klienta a brokera. Bylo obtížné vyhledat na internetu návody zaměřené přímo na tuto problematiku. Existuje mnoho způsobů, které lze využít – jedním z nich je například využití certifikační autority *Let's encrypt*. Pro popsání celého procesu byl však nakonec vybrán způsob zmíněný výše – vytvoření vlastní CA. Jakmile byla CA vytvořena a byly úspěšně podepsány certifikáty a klíče, objevil se další problém. Jednalo se o problém s oprávněním přístupu k těmto klíčům, a to i přes to, že byly uloženy do uživatelské složky. Po poskytnutí těchto oprávnění pomocí příkazů *Chmod* byl tento problém vyřešen a vše fungovalo tak, jak mělo.

Další komplikací je řízení stavu UI prvků na Node-RED dashboard. Pokud je přidán do této dashboard prvek typu *switch* (přepínací tlačítko), který má na starost například ovládání světel, je velmi obtížné řídit jeho stav. Pokud je dané světlo rozsvíceno jinak – automatizací, či spuštěním určité scény, není již aktualizován stav tohoto switch tlačítka. Je samozřejmě možné poslat zprávu *false*, či *true*, přímo do tlačítka. To ale nejen přepne jeho stav, ale také přepošle tuto zprávu do dalšího uzlu – v tomto případě MQTT out uzlu. Zpráva je následně poslána skrze MQTT, což vede k provedení úkonu zhasnutí/rozsvícení diody. Díky tomu se program zacyklí. Řízení stavu lze do určité míry implementovat, ale je to velice náročné a nepřehledné.

Posledním problémem, který se vyskytl, je nepřesnost měření senzorů. Jelikož se jedná převážně o „hobby" senzory, dalo se toto chování předpokládat. Senzor DHT22 byl relativně přesný, ale senzory MQ-9 a MQ-135 už na tom byly trochu jinak. Jejich přesnost měření ovlivňuje nahřátí a správné sejmutí konstanty R_0 v procesu kalibrace. Tento proces byl proveden vícekrát a pokaždé vyšla tato konstanta jinak. V závěru toto znamenalo nereálné hodnoty, jako například koncentraci LPG

v místnosti s čistým vzduchem. Vyvstává tedy otázka, zdali se tyto senzory hodí pro použití v profesionálních implementacích.

Je také dobré zmínit, že veškerá hesla, která byla v rámci implementace použita, byla generována. Znamená to, že žádné heslo se neopakuje. Zároveň se nejedná o klasická hesla, nýbrž o takzvané *pass-phrase*s (heslové fráze). Rozdíl mezi heslem a heslovou frází je v tom, že heslová fráze je delší sekvence znaků, jež může vytvářet frázi nebo větu. Je tedy bezpečnější. Příkladem takovéto fráze je *I-love7-ice3-cream*. Z toho vyplývá velmi důležité doporučení: Aby byl systém dobře zabezpečený, je potřeba, aby byla používána silná, neopakující se hesla.

Možné rozšíření této práce: Implementovat propojení s databází MongoDB (nebo jakoukoliv jinou), pro sběr a ukládání dat. Node-RED obsahuje spoustu funkcionalit, pomocí kterých tohoto propojení lze docílit. Druhým rozšířením by bylo přidání více senzorů a aktuátorů do sítě. Jednalo by se například o servo na ovládání žaluzií, fotorezistor na snímání světelnosti, či čidlo pohybu. Zatřetí by bylo možné více do detailu otestovat síť na případná rizika pomocí Kali Linux. Provést tak kompletní penetrační testování a analýzu bezpečnostních rizik, jež se v implementaci mohou skrývat.

8 Seznam použité literatury

[1] A. P., H., & K., K. (2019). Secure-MQTT: An efficient fuzzy logic-based approach to detect DoS attack in MQTT protocol for internet of things. *EURASIP Journal on Wireless Communications and Networking*, *2019*(1), 90. https://doi.org/10.1186/s13638-019-1402-8

[2] Aas, J., Barnes, R., Case, B., Durumeric, Z., Eckersley, P., Flores-López, A., Halderman, J. A., Hoffman-Andrews, J., Kasten, J., Rescorla, E., Schoen, S., & Warren, B. (2019). Let's Encrypt: An Automated Certificate Authority to Encrypt the Entire Web. *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2473–2487. https://doi.org/10.1145/3319535.3363192

[3] Abir, S. M. A. A., Anwar, A., Choi, J., & Kayes, A. S. M. (2021). IoT-Enabled Smart Energy Grid: Applications and Challenges. *IEEE Access*, *9*, 50961–50981. https://doi.org/10.1109/ACCESS.2021.3067331

[4] Alexander Bessman. (2021). *MQ135 gas response curves*. https://i0.wp.com/pslab.io/wp-content/uploads/response.png?w=640&ssl=1

[5] Amjad Ali. (2023). *How to Interface MQ-9 Gas Sensor with Arduino*. https://www.circuits-diy.com/how-to-interface-mq-9-gas-sensor-with-arduino/

[6] amqp.org. (2011). *AMQP v1.0*. https://www.amqp.org/sites/amqp.org/files/amqp.pdf

[7] Arnaud. (2023). Symmetric vs Asymmetric Encryption: What's the difference? *Mailfence*. https://blog.mailfence.com/symmetric-vs-asymmetric-encryption/

[8] Ashton, K. (2009). *That "Internet of Things" Thing: In the Real World Things Matter More than Ideas. RFID Journal*. http://www.itrco.jp/libraries/RFIDjournal-That%20Internet%20of%20Things%20Thing.pdf

[9] Ayman, O. (2019). Analysis of Application Layer Protocols Used in Internet of Things. *SSRN Electronic Journal*. https://doi.org/10.2139/ssrn.3810913

[10] Clerissi, D., Leotta, M., Reggio, G., & Ricca, F. (2018). Towards an approach for developing and testing Node-RED IoT systems. *Proceedings of the 1st ACM SIGSOFT International Workshop on Ensemble-Based Software Engineering*, 1–8. https://doi.org/10.1145/3281022.3281023

[11] esperso. (2020). 7 Advantages of MQTT protocol for IoT Devices. *Medium*. https://medium.com/@esperso/7-benefits-of-mqtt-protocol-for-iot-e463f6a97100

[12] Espressif.com. (2023). *ESP8266*. Espressif.com. https://www.espressif.com/en/products/socs/esp8266

[13] Fritzing. (2023). *Fritzing*. Frizting. https://fritzing.org/

[14] GeeksForGeeks. (2020). Session Layer Messaging Protocols in IoT. *GeeksForGeeks*. https://www.geeksforgeeks.org/session-layer-messaging-protocols-in-iot/

[15] Gemirter, C. B., Senturca, C., & Baydere, S. (2021). A Comparative Evaluation of AMQP, MQTT and HTTP Protocols Using Real-Time Public Smart City Data. *2021 6th International Conference on Computer Science and Engineering (UBMK)*, 542–547. https://doi.org/10.1109/UBMK52708.2021.9559032

[16] Harmann.info. (2023a). *KARMIN dvoulůžkový pokoj*. https://www.hartmann.info/-/media/disinfection/missionprevention/mip-karmin-zweibettzimmer-cz.jpg?mh=600&mw=840&rev=9f06f48762784273b2491f44fa96b8a3&sc_lang=cs-cz&hash=9F928D0195ADE0DE886DD58B53C25099

[17] Harmann.info. (2023b). *KARMIN dvoulůžkový pokoj pacientů*. https://www.hartmann.info/-/media/disinfection/missionprevention/mip-karmin-zweibettzimmer-868x578.jpg?mw=1260&rev=0eae2ce5132d44869e64ae2cdd40c478&sc_lang=cs-cz&hash=CF72D80BAB3D191DCDB6B3EFBB19FC3B

[18] Henan Hanwei Electronics Co., Ltd. (b.r.). *Semiconductor Sensor for CO/Combustible Gas*. Henan Hanwei Electronics Co., Ltd. https://www.laskakit.cz/user/related_files/mq9.pdf

[19] Hussain, F. (2017). Internet of Everything. In F. Hussain, *Internet of Things* (s. 1–11). Springer International Publishing. https://doi.org/10.1007/978-3-319-55405-1_1

[20] International Business Machines Corporation (IBM) & Eurotech. (2010). *MQTT V3.1 Protocol Specification*. https://public.dhe.ibm.com/software/dw/webservices/ws-mqtt/mqtt-v3r1.html#msg-format

[21] Internet Society. (2023). *TLS Basics*. https://www.internetsociety.org/deploy360/tls/basics/

[22] IoT Boys. (2018). What is AMQP Protocol? How AMQP Protocol Works. *IoT Boys*. https://iotboys.com/what-is-amqp-how-amqp-works-for-internet-of-things/

[23] ipc2U.cz. (2023). Co je MQTT a k čemu slouží ve IIoT? *ipc2U.cz*. https://ipc2u.cz/blogs/news/mqtt-protokol

[24] Kali.org. (2023). *Kali.org*. Kali.org. https://www.kali.org/

[25] Kinsta.com. (2022). TLS vs SSL: What's the Difference? Which One Should You Use? *Kinsta.com*. https://kinsta.com/knowledgebase/tls-vs-ssl/

[26] Knud Lasse Lueth. (2020). *Top 10 IoT application areas*. https://iot-analytics.com/top-10-iot-applications-in-2020/

[27] LaskaKit. (2023). *IoT ESP8266 Lua NodeMcu V3 WIFI modul*. Laskakit.cz. https://www.laskakit.cz/iot-esp8266-lua-nodemcu-v3-wifi-modul--tcp-ip/

[28] Luboš M. (b.r.). Senzor plynů MQ-135. *dratek.cz*. https://navody.dratek.cz/navody-k-produktum/senzor-plynu-mq-135.html

[29] Lueth, K. L. (2015). *IoT basics: Getting started with the Internet of Things*. https://iot-analytics.com/wp/wp-content/uploads/2015/03/2015-March-Whitepaper-IoT-basics-Getting-started-with-the-Internet-of-Things.pdf

[30] M. A. Ezechina, K. K. Okwara, & C. A. U. Ugboaja. (2015). *The Internet of Things (Iot): A Scalable Approach to Connecting Everything*. https://www.theijes.com/papers/v4-i1/Version-1/C041109012.pdf

[31] McAteer, I. N., Malik, M. I., Zubair Baig, & Hannay, P. (2017). Security vulnerabilities and cyber threat analysis of the AMQP protocol for the internet of things [PDF]. *Australian Information Security Management Conference*. https://doi.org/10.4225/75/5A84F4A695B4C

[32] Melnikov, A. & K. Zeilenga. (2006). *Simple Authentication and Security Layer (SASL)* (Č. RFC4422; s. RFC4422). RFC Editor. https://doi.org/10.17487/rfc4422

[33] Michael Cobb & Peter Loshin. (2021). Secure Shell (SSH). *TechTarget*. https://www.techtarget.com/searchsecurity/definition/Secure-Shell

[34] Moellmann, J., Jurk, L. A., Zeise, O., & Sunder, W. (2022). *KARMIN: Evaluation des infektionspräventiven Patientenzimmers*. Fraunhofer IRB Verlag.

[35] mosquitto.org. (2023a). *Eclipse Mosquitto™ An open source MQTT broker*. mosquitto. https://mosquitto.org/

[36] mosquitto.org. (2023b). *mosquitto.conf—The configuration file for mosquitto*. https://mosquitto.org/man/mosquitto-conf-5.html

[37] Naik, N. (2017). Choice of effective messaging protocols for IoT systems: MQTT, CoAP, AMQP and HTTP. *2017 IEEE International Systems Engineering Symposium (ISSE)*, 1–7. https://doi.org/10.1109/SysEng.2017.8088251

[38] Nastase, L. (2017). Security in the Internet of Things: A Survey on Application Layer Protocols. *2017 21st International Conference on Control Systems and Computer Science (CSCS)*, 659–666. https://doi.org/10.1109/CSCS.2017.101

[39] Nodered.org. (2023). *Running on Raspberry Pi*. https://nodered.org/docs/getting-started/raspberrypi

[40] *Oasis Open*. (2023). Oasis Open. https://www.oasis-open.org/

[41] O'Hara, J. (2007). Toward a Commodity Enterprise Middleware: Can AMQP enable a new era in messaging middleware? A look inside standards-based messaging with AMQP. *Queue*, *5*(4), 48–55. https://doi.org/10.1145/1255421.1255424

[42] Oleksii Tsymbal. (2022). *IoT in Manufacturing: 6 Industrial IoT Trends in 2023*. https://mobidev.biz/blog/industrial-iot-internet-of-things-trends

[43] OpenLabPro. (2019a). *Introduction to MQTT Protocol*. https://openlabpro.com/wp-content/uploads/2019/10/mqtt-pub-sub-model.jpg

[44] OpenLabPro. (2019b). *MQTT Packet Format*. OpenLabPro. https://openlabpro.com/guide/mqtt-packet-format/

[45] Pramanik, M. I., Lau, R. Y. K., Demirkan, H., & Azad, Md. A. K. (2017). Smart health: Big data enabled health paradigm within smart cities. *Expert Systems with Applications*, *87*, 370–383. https://doi.org/10.1016/j.eswa.2017.06.027

[46] RabbitMQ. (2023). *Which protocols does RabbitMQ support?* RabbitMQ. https://www.rabbitmq.com/protocols.html

[47] Raspberry Pi. (2023a). *Raspberry Pi*. Raspberry Pi. https://www.raspberrypi.com/software/operating-systems/

[48] Raspberry Pi. (2023b). *Raspberry Pi Documentation*. Raspberry Pi. https://www.raspberrypi.com/documentation/computers/configuration.html

[49] Raza, S., Trabalza, D., & Voigt, T. (2012). 6LoWPAN Compressed DTLS for CoAP. *2012 IEEE 8th International Conference on Distributed Computing in Sensor Systems*, 287–289. https://doi.org/10.1109/DCOSS.2012.55

[50] Retscher, G. (2020). Fundamental Concepts and Evolution of Wi-Fi User Localization: An Overview Based on Different Case Studies. *Sensors*, *20*(18), 5121. https://doi.org/10.3390/s20185121

[51] Scaler Academy. (2022). IoT Architecture – Detailed Explanation. *Scaler Academy*. https://www.interviewbit.com/blog/iot-architecture/

[52] SHOLA USHA RANI, S RAJARAJESWARI, JERRY GEORGE JAIMON, & ROSHAN RAVICHANDRAN. (2020). *REAL-TIME AIR QUALITY MONITORING SYSTEM USING MQ135 AND THINGSBOARD*. Vellore Institute of Technology, Chennai,Tamilnadu,India. https://www.researchgate.net/profile/Shola-Usha-Rani/publication/347946855_REAL-TIME_AIR_QUALITY_MONITORING_SYSTEM_USING_MQ135_AND_THINGSBOARD/links/60e2d82e92851ca944aa1013/REAL-TIME-AIR-QUALITY-MONITORING-SYSTEM-USING-MQ135-AND-THINGSBOARD.pdf

[53] Siddhesh Shinde. (2023). What are the Key Pros and Cons of the Arduino Programming Language? *emeritus*. https://emeritus.org/blog/coding-arduino-programming-language/

[54] Sittón-Candanedo, I., Alonso, R. S., García, Ó., Muñoz, L., & Rodríguez-González, S. (2019). Edge Computing, IoT and Social Computing in Smart Energy Scenarios. *Sensors*, *19*(15), 3353. https://doi.org/10.3390/s19153353

[55] Stallings, W., Agboma, F., & Jelassi, S. (2016). *Foundations of modern networking: SDN, NFV, QoE, IoT, and Cloud*. Pearson.

[56] Syafarinda, Y., Akhadin, F., Fitri, Z. E., Yogiswara, Widiawan, B., & Rosdiana, E. (2018). The Precision Agriculture Based on Wireless Sensor Network with MQTT Protocol. *IOP Conference Series: Earth and Environmental Science*, *207*, 012059. https://doi.org/10.1088/1755-1315/207/1/012059

[57] The HiveMQ Team. (2015a). MQTT Quality of Service (QoS) 0,1, & 2 – MQTT Essentials: Part 6. *HiveMQ*. https://www.hivemq.com/blog/mqtt-essentials-part-6-mqtt-quality-of-service-levels/

[58] The HiveMQ Team. (2015b). *MQTT Retained Messages—MQTT Essentials: Part 8*. https://www.hivemq.com/blog/mqtt-essentials-part-8-retained-messages/

[59] *The Top 5 Applications For Industrial IoT*. (2023). Particle.io. https://www.particle.io/iot-guides-and-resources/industrial-iot-applications/

[60] Thilagavathy, D., & Institute of Electrical and Electronics Engineers (Ed.). (2014). *2014 International Conference on Electronics, Communication and Computational Engineering (ICECCE 2014): Hosur, Tamilnadu, India, 17-18 November 2014*. IEEE.

[61] Tian, S., Yang, W., Grange, J. M. L., Wang, P., Huang, W., & Ye, Z. (2019). Smart healthcare: Making medical care more intelligent. *Global Health Journal*, *3*(3), 62–65. https://doi.org/10.1016/j.glohj.2019.07.001

[62] tzbinfo. (2023). *Vnitřní výpočtové teploty dle ČSN EN 12831 a doporučené relativní vlhkosti vzduchu dle ČSN 06 0210*. https://vetrani.tzb-info.cz/tabulky-a-vypocty/28-vnitrni-vypoctove-teploty-dle-csn-en-12831-a-doporucene-relativni-vlhkosti-vzduchu-dle-csn-06-0210

[63] *Uzdravující architektura: Prevence infekcí již od návrhu*. (2023). Hartmann. https://www.hartmann.info/cs-cz/mise-prevence-infekci/l/cz/vyzkumny-projekt-karmin

[64] Wallarm. (2023a). *What is Advanced Message Queuing Protocol (AMQP)?* https://assets.website-files.com/5ff66329429d880392f6cba2/619f53ce469a19d18a61ef94_AMQP%20Broker.png

[65] Wallarm. (2023b). *What is CoAP Protocol? Meaning & Architecture*. https://www.wallarm.com/what/coap-protocol-definition

[66] Wortmann, F., & Flüchter, K. (2015). Internet of Things: Technology and Value Added. *Business & Information Systems Engineering*, *57*(3), 221–224. https://doi.org/10.1007/s12599-015-0383-3

9 Přílohy

1) Kód LED a Zephyr 40 MM
2) Kód DHT22 a MQ-9
3) Kód kalibrace MQ-9
4) Kód MQ-135
5) Kód serva
6) Odkaz na GitHub repositář

```
#include <ArduinoWiFiServer.h>
#include <BearSSLHelpers.h>
#include <CertStoreBearSSL.h>
#include <ESP8266WiFi.h>
#include <ESP8266WiFiAP.h>
#include <ESP8266WiFiGeneric.h>
#include <ESP8266WiFiGratuitous.h>
#include <ESP8266WiFiMulti.h>
#include <ESP8266WiFiSTA.h>
#include <ESP8266WiFiScan.h>
#include <ESP8266WiFiType.h>
#include <WiFiClient.h>
#include <WiFiClientSecure.h>
#include <WiFiClientSecureBearSSL.h>
#include <WiFiServer.h>
#include <WiFiServerSecure.h>
#include <WiFiServerSecureBearSSL.h>
#include <WiFiUdp.h>
#include <ESP8266WiFi.h>
#include <PubSubClient.h>
#include <SPI.h>
#include <TimeLib.h>

// SSL
// CA certifikát
const char ca_cert[] PROGMEM = R"EOF()EOF";

// Klíč klienta
const char client_key[] PROGMEM = R"EOF()EOF";

// Certifikát klienta
const char client_cert[] PROGMEM = R"EOF()EOF";

// Fingerptint zařízení
const char *fingerprint PROGMEM = "";

// Wi-Fi
const char *ssid = "pi_net";
const char *password = "";

// MQTT Broker
const char *mqtt_broker = "raspberrypi.local";
const char *mqtt_username = "client";
const char *mqtt_password = "";
const int mqtt_port = 8883;
```

```
// Témata
const char *lights_topic = "f3/r7/controls/lights";
const char *vent_topic = "f3/r7/controls/vent";
const char *scenes_topic = "f3/r7/scenes";

// Nastavení pinů
const int REL_LED = 5;
const int REL_VENT = 4;

BearSSL::WiFiClientSecure espClient;
PubSubClient client(espClient);

void setup() {
  // Inicializace sériové komunikace
  Serial.begin(115200);

  // Načtení certifikátu CA
  BearSSL::X509List cert(ca_cert);
  espClient.setTrustAnchors(&cert);

  // Načtení certifikátů klienta a klíče klienta
  BearSSL::X509List client_crt(client_cert);
  BearSSL::PrivateKey key(client_key);
  espClient.setClientRSACert(&client_crt, &key);

  // Nastavení momentálního času – nutné pro TLS připojení
  time_t t = now();
  espClient.setX509Time(t);

  // Nastavení fingerprintu
  espClient.setFingerprint(fingerprint);

  // Inicializace Wi-Fi
  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.println("Connecting to WiFi..");
  }

  Serial.println("Connected to the WiFi network");

  // Nastavení serveru MQTT
  client.setServer(mqtt_broker, mqtt_port);

  // Nastavení callbacku – funkce, která se stará o zpracování zpráv
  client.setCallback(callback);
```

```
  // Smyčka pro připojení k MQTT brokeru
  while (!client.connected()) {
    String client_id = "ESP1-";
    client_id += String(WiFi.macAddress());
    Serial.printf("The client %s connects to the public mqtt broker\n",
client_id.c_str());
    if (client.connect(client_id.c_str(), mqtt_username, mqtt_password))
{
      Serial.println("Connected to MQTT broker!");
    } else {
      Serial.print("Failed with state ");
      Serial.print(client.state());
      delay(2000);
    }
  }

  // Nastavení módu pinů
  pinMode(REL_LED, OUTPUT);
  pinMode(REL_VENT, OUTPUT);

  // Odebírání témat
  client.subscribe(lights_topic);
  client.subscribe(vent_topic);
  client.subscribe(scenes_topic);
}

// Funkce callback
void callback(char *topic, byte *payload, unsigned int length) {

  // Vypsání nových zpráv na konzoli
  Serial.print("Message arrived in topic: ");
  Serial.println(topic);
  Serial.print("Message: ");

  String message;
  for (int i = 0; i < length; i++) {
    message = message + (char)payload[i];
  }

  Serial.println(message);

  Serial.println("-----------------------");
```

```
  // Pro téma f3/r7/controls/lights
  if (strcmp(topic, lights_topic) == 0) {

    if (message == "#on") {
      digitalWrite(REL_LED, LOW);
    }
    if (message == "#off") {
      digitalWrite(REL_LED, HIGH);
    }
  }

  // Pro téma f3/r7/controls/vent
  if (strcmp(topic, vent_topic) == 0) {

    if (message == "#on") {
      digitalWrite(REL_VENT, LOW);
    }
    if (message == "#off") {
      digitalWrite(REL_VENT, HIGH);
    }
  }

  // Pro téma f3/r7/scenes
  if (strcmp(topic, scenes_topic) == 0) {

    if (message == "#night") {
      digitalWrite(REL_LED, HIGH);
    }
    if (message == "#morning") {
      digitalWrite(REL_VENT, LOW);
      digitalWrite(REL_LED, LOW);
    }
  }
}

void loop() {
  client.loop();
}
```

```
#include <ArduinoWiFiServer.h>
#include <BearSSLHelpers.h>
#include <CertStoreBearSSL.h>
#include <ESP8266WiFi.h>
#include <ESP8266WiFiAP.h>
#include <ESP8266WiFiGeneric.h>
#include <ESP8266WiFiGratuitous.h>
#include <ESP8266WiFiMulti.h>
#include <ESP8266WiFiSTA.h>
#include <ESP8266WiFiScan.h>
#include <ESP8266WiFiType.h>
#include <WiFiClient.h>
#include <WiFiClientSecure.h>
#include <WiFiClientSecureBearSSL.h>
#include <WiFiServer.h>
#include <WiFiServerSecure.h>
#include <WiFiServerSecureBearSSL.h>
#include <WiFiUdp.h>
#include <ESP8266WiFi.h>
#include <PubSubClient.h>
#include <SPI.h>
#include <TimeLib.h>
#include <DHT.h>

// SSL
// CA certifikát
const char ca_cert[] PROGMEM = R"EOF()EOF";

// Klíč klienta
const char client_key[] PROGMEM = R"EOF()EOF";

// Certifikát klienta
const char client_cert[] PROGMEM = R"EOF()EOF";

// Fingerptint zařízení
const char *fingerprint PROGMEM = "";

// Wi-Fi
const char *ssid = "pi_net";
const char *password = "";
```

```
// MQTT Broker
const char *mqtt_broker = "raspberrypi.local";
const char *mqtt_username = "client";
const char *mqtt_password = "";
const int mqtt_port = 8883;

// Témata
const char *states_temperature = "f3/r7/states/temperature";
const char *states_humidity = "f3/r7/states/humidity";
const char *states_mq9_ratio = "f3/r7/states/mq9/ratio";
const char *states_mq9_alarm = "f3/r7/states/mq9/alarm";
const char *states_mq9_co_ppm = "f3/r7/states/mq9/co/ppm";
const char *states_mq9_lpg_ppm = "f3/r7/states/mq9/lpg/ppm";
const char *states_mq9_ch4_ppm = "f3/r7/states/mq9/ch4/ppm";

BearSSL::WiFiClientSecure espClient;
PubSubClient client(espClient);

// Konfigurace DHT22
#define DHTTYPE DHT22
#define DHTPIN 5

DHT dht(DHTPIN, DHTTYPE);

long time_now;
long lastMeasure;

// Digitální pin MQ-9
const int MQ9_D = 4;

// Ostatní proměnné pro práci se senzorem MQ-9
int alarm = 0;
float sensor_volt;
float RS_gas;
float ratio;
const float R0 = 1.02;

void setup() {

  // Inicializace sériové komunikace
  Serial.begin(74880);

  // Načtení certifikátu CA
  BearSSL::X509List cert(ca_cert);
  espClient.setTrustAnchors(&cert);
```

```
  // Načtení certifikátů klienta a klíče klienta
  BearSSL::X509List client_crt(client_cert);
  BearSSL::PrivateKey key(client_key);
  espClient.setClientRSACert(&client_crt, &key);

  // Nastavení momentálního času – nutné pro TLS připojení
  time_t t = now();
  espClient.setX509Time(t);

  // Nastavení fingerprintu
  espClient.setFingerprint(fingerprint);

  // Inicializace Wi-Fi
  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.println("Connecting to WiFi..");
  }

  Serial.println("Connected to the WiFi network");

  // Nastavení serveru MQTT
  client.setServer(mqtt_broker, mqtt_port);

  // Nastavení callbacku – funkce, která se stará o zpracování zpráv
  client.setCallback(callback);


  // Smyčka pro připojení k MQTT brokeru
  while (!client.connected()) {
    String client_id = "ESP1-";
    client_id += String(WiFi.macAddress());
    Serial.printf("The client %s connects to the public mqtt broker\n",
client_id.c_str());
    if (client.connect(client_id.c_str(), mqtt_username, mqtt_password))
{
      Serial.println("Connected to MQTT broker!");
    } else {
      Serial.print("Failed with state ");
      Serial.print(client.state());
      delay(2000);
    }
  }

  dht.begin();
```

```
  // Odebírání témat
  client.subscribe(states_temperature);
  client.subscribe(states_humidity);
  client.subscribe(states_mq9_ratio);
  client.subscribe(states_mq9_alarm);
}

// Funkce callback
void callback(char *topic, byte *payload, unsigned int length) {

  Serial.print("Message arrived in topic: ");
  Serial.println(topic);
  Serial.print("Message: ");

  String message;
  for (int i = 0; i < length; i++) {
    message = message + (char)payload[i];
  }

  Serial.println(message);

  Serial.println("-----------------------");
}

void loop() {
  client.loop();

  // Nastavení času na "teď"
  time_now = millis();

  // Každých 30 sekund je posláno nové měření
  if (time_now - lastMeasure > 10000) {
    lastMeasure = time_now;

    // Načtení hodnot senzoru DHT22
    float humidity = dht.readHumidity();
    float temperatureC = dht.readTemperature();

    // Načtení hodnot senzoru MQ-9
    int sensorValue = analogRead(A0);
    alarm = digitalRead(MQ9_D);

    // Výpočet napětí senzoru
    sensor_volt = ((float)sensorValue / 1024) * 5.0;

    // Výpočet odporu v daném plynu
    RS_gas = (5.0 - sensor_volt) / sensor_volt;
```

```
    // Výpočet poměru Rs_gas a R0
    ratio = RS_gas / R0;

    // Pro výpočet přibližné koncentrace CO
    if (ratio < 1.6) {
      ratio = ratio * 100;
      int co_ppm = map(ratio, 160, 79, 200, 1000);

      client.publish(states_mq9_co_ppm, String(co_ppm).c_str());
    }

    // Pro výpočet přibližné koncentrace LPG
    if (ratio < 2.6) {
      ratio = ratio * 100;
      int lpg_ppm = map(ratio, 260, 70, 200, 10000);

      client.publish(states_mq9_lpg_ppm, String(lpg_ppm).c_str());
    }

      // Pro výpočet přibližné koncentrace CH4
    if (ratio < 2) {
      ratio = ratio * 100;
      int ch4_ppm = map(ratio, 200, 35, 200, 10000);

      client.publish(states_mq9_ch4_ppm, String(ch4_ppm).c_str());
    }


    // Podmínka kontroluje, zdali bylo čtení ze senzorů v pořádku
    if (isnan(humidity) || isnan(temperatureC) || isnan(ratio) ||
isnan(alarm)) {
      Serial.println("Failed to read from sensors!");
      return;
    }

    // Publikování daných hodnot
    client.publish(states_temperature, String(temperatureC).c_str());
    client.publish(states_humidity, String(humidity).c_str());
    client.publish(states_mq9_ratio, String(ratio).c_str());

    if (alarm == 1) client.publish(states_mq9_alarm, "Limit exceeded!");
  }
}
```

```
void setup() {
  // Otevření sériové komunikace
  Serial.begin(9600);
}

void loop() {
  float sensor_volt;  // Napětí senzoru
  float RS_air;       // Odpor v čistém vzduchu
  float R0;           // Konstanta R0
  float sensorValue;  // Hodnota načtená ze senzoru

  // Cyklus pro načtení 100 hodnot
  for (int x = 0; x < 100; x++) {
    sensorValue = sensorValue + analogRead(A0);
  }
  sensorValue = sensorValue / 100.0;

  sensor_volt = (sensorValue / 1024) * 5.0;    // Výpočet napětí senzoru
  RS_air = (5.0 - sensor_volt) / sensor_volt;  // Výpočet oporu v čistém
vzduchu
  R0 = RS_air / 9.9;                           // Výpočet konstanty R0

  // Vypsání na konzoli
  Serial.print("sensor_volt = ");
  Serial.print(sensor_volt);
  Serial.println("V");

  Serial.print("R0 = ");
  Serial.println(R0);
  delay(1000);
}
```

```
#include <ArduinoWiFiServer.h>
#include <BearSSLHelpers.h>
#include <CertStoreBearSSL.h>
#include <ESP8266WiFi.h>
#include <ESP8266WiFiAP.h>
#include <ESP8266WiFiGeneric.h>
#include <ESP8266WiFiGratuitous.h>
#include <ESP8266WiFiMulti.h>
#include <ESP8266WiFiSTA.h>
#include <ESP8266WiFiScan.h>
#include <ESP8266WiFiType.h>
#include <WiFiClient.h>
#include <WiFiClientSecure.h>
#include <WiFiClientSecureBearSSL.h>
#include <WiFiServer.h>
#include <WiFiServerSecure.h>
#include <WiFiServerSecureBearSSL.h>
#include <WiFiUdp.h>
#include <ESP8266WiFi.h>
#include <PubSubClient.h>
#include <SPI.h>
#include <TimeLib.h>
#include <MQ135.h>

// SSL
// CA certifikát
const char ca_cert[] PROGMEM = R"EOF()EOF";

// Klíč klienta
const char client_key[] PROGMEM = R"EOF()EOF";

// Certifikát klienta
const char client_cert[] PROGMEM = R"EOF()EOF";

// Fingerptint zařízení
const char *fingerprint PROGMEM = "";

// Wi-Fi
const char *ssid = "pi_net";
const char *password = "";

// MQTT Broker
const char *mqtt_broker = "raspberrypi.local";
const char *mqtt_username = "client";
const char *mqtt_password = "";
const int mqtt_port = 8883;
```

```
// Témata
const char *states_mq135_ppm = "f3/r7/states/mq135/ppm";
const char *states_mq135_co2_ppm = "f3/r7/states/mq135/co2/ppm";
const char *states_mq135_ethanol_ppm = "f3/r7/states/mq135/ethanol/ppm";
const char *states_mq135_nh3_ppm = "f3/r7/states/mq135/nh3/ppm";
const char *states_mq135_toluene_ppm = "f3/r7/states/mq135/toulene/ppm";
const char *states_mq135_acetone_ppm = "f3/r7/states/mq135/acetone/ppm";
const char *states_mq135_alarm = "f3/r7/states/mq135/alarm";

BearSSL::WiFiClientSecure espClient;
PubSubClient client(espClient);

// Analogový pin A0
#define pinA A0

// Digitální pin D1
const int MQ135_D = 5;

int alarm = 0;
float sensor_volt;
float RS_gas;
float ratio;

MQ135 senzorMQ = MQ135(pinA);

long time_now;
long lastMeasure;

void setup() {

  // Inicializace sériové komunikace
  Serial.begin(57600);

  // Načtení certifikátu CA
  BearSSL::X509List cert(ca_cert);
  espClient.setTrustAnchors(&cert);

  // Načtení certifikátů klienta a klíče klienta
  BearSSL::X509List client_crt(client_cert);
  BearSSL::PrivateKey key(client_key);
  espClient.setClientRSACert(&client_crt, &key);

  // Nastavení momentálního času – nutné pro TLS připojení
  time_t t = now();
  espClient.setX509Time(t);
```

```
  // Nastavení fingerprintu
  espClient.setFingerprint(fingerprint);

  // Inicializace Wi-Fi
  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.println("Connecting to WiFi..");
  }

  Serial.println("Connected to the WiFi network");

  // Nastavení serveru MQTT
  client.setServer(mqtt_broker, mqtt_port);

  // Nastavení callbacku – funkce, která se stará o zpracování zpráv
  client.setCallback(callback);


  // Smyčka pro připojení k MQTT brokeru
  while (!client.connected()) {
    String client_id = "ESP1-";
    client_id += String(WiFi.macAddress());
    Serial.printf("The client %s connects to the public mqtt broker\n",
client_id.c_str());
    if (client.connect(client_id.c_str(), mqtt_username, mqtt_password))
{
      Serial.println("Connected to MQTT broker!");
    } else {
      Serial.print("Failed with state ");
      Serial.print(client.state());
      delay(2000);
    }

    client.subscribe(states_mq135_ppm);
  }
}
```

```
// Funkce callback
void callback(char *topic, byte *payload, unsigned int length) {

  Serial.print("Message arrived in topic: ");
  Serial.println(topic);
  Serial.print("Message: ");

  String message;
  for (int i = 0; i < length; i++) {
    message = message + (char)payload[i];
  }

  Serial.println(message);

  Serial.println("-----------------------");
}


void loop() {
  client.loop();

  // Nastavení času na "teď"
  time_now = millis();

  // Každých 30 sekund je posláno nové měření
  if (time_now - lastMeasure > 10000) {
    lastMeasure = time_now;

    // Načtení hodnoty ppm ze senzoru
    float ppm = senzorMQ.getPPM();

    // Načtení hodnot senzoru MQ-135
    int sensorValue = analogRead(A0);
    alarm = digitalRead(MQ135_D);

    // Výpočet napětí senzoru
    sensor_volt = ((float)sensorValue / 1024) * 5.0;

    // Výpočet odporu v daném plynu
    RS_gas = (5.0 - sensor_volt) / sensor_volt;

    // Výpočet poměru Rs_gas a R0
    ratio = RS_gas / senzorMQ.getRZero();
```

```
    // Pro výpočet přibližné koncentrace CO2
    if (ratio < 2.2) {
      ratio = ratio * 100;
      int co2_ppm = map(ratio, 220, 80, 10, 110);

      client.publish(states_mq135_co2_ppm, String(co2_ppm).c_str());
    }

    // Pro výpočet přibližné koncentrace etanolu (alkoholu)
    if (ratio < 2.9) {
      ratio = ratio * 100;
      int ethanol_ppm = map(ratio, 290, 150, 10, 110);

      client.publish(states_mq135_ethanol_ppm,
String(ethanol_ppm).c_str());
    }

    // Pro výpočet přibližné koncentrace NH3
    if (ratio < 2.5) {
      ratio = ratio * 100;
      int nh3_ppm = map(ratio, 250, 100, 10, 100);

      client.publish(states_mq135_nh3_ppm, String(nh3_ppm).c_str());
    }

    // Pro výpočet přibližné koncentrace toluenu
    if (ratio < 1.6) {
      ratio = ratio * 100;
      int toluene_ppm = map(ratio, 160, 65, 10, 100);

      client.publish(states_mq135_toluene_ppm,
String(toluene_ppm).c_str());
    }

    // Pro výpočet přibližné koncentrace acetonu
    if (ratio < 1.5) {
      ratio = ratio * 100;
      int acetone_ppm = map(ratio, 150, 60, 10, 100);

      client.publish(states_mq135_acetone_ppm,
String(acetone_ppm).c_str());
    }
```

```
    // Podmínka kontroluje, zdali bylo čtení ze senzorů v pořádku
    if (isnan(ppm) || isnan(ratio) || isnan(alarm)) {
      Serial.println("Failed to read from sensor!");
      return;
    }

    // Publikování daných hodnot
    client.publish(states_mq135_ppm, String(ppm).c_str());

    if (alarm == 1) client.publish(states_mq135_alarm, "Limit
exceeded!");
  }
}
```

```
#include <ArduinoWiFiServer.h>
#include <BearSSLHelpers.h>
#include <CertStoreBearSSL.h>
#include <ESP8266WiFi.h>
#include <ESP8266WiFiAP.h>
#include <ESP8266WiFiGeneric.h>
#include <ESP8266WiFiGratuitous.h>
#include <ESP8266WiFiMulti.h>
#include <ESP8266WiFiSTA.h>
#include <ESP8266WiFiScan.h>
#include <ESP8266WiFiType.h>
#include <WiFiClient.h>
#include <WiFiClientSecure.h>
#include <WiFiClientSecureBearSSL.h>
#include <WiFiServer.h>
#include <WiFiServerSecure.h>
#include <WiFiServerSecureBearSSL.h>
#include <WiFiUdp.h>
#include <ESP8266WiFi.h>
#include <PubSubClient.h>
#include <SPI.h>
#include <TimeLib.h>
#include <Servo.h>

// SSL
// CA certifikát
const char ca_cert[] PROGMEM = R"EOF()EOF";

// Klíč klienta
const char client_key[] PROGMEM = R"EOF()EOF";

// Certifikát klienta
const char client_cert[] PROGMEM = R"EOF()EOF";

// Fingerptint zařízení
const char *fingerprint PROGMEM = "";

// Wi-Fi
const char *ssid = "pi_net";
const char *password = "";

// MQTT Broker
const char *mqtt_broker = "raspberrypi.local";
const char *mqtt_username = "client";
const char *mqtt_password = "";
const int mqtt_port = 8883;
```

```
// Témata
const char *heating_topic = "f3/r7/controls/heating";
const char *scenes_topic = "f3/r7/scenes";


BearSSL::WiFiClientSecure espClient;
PubSubClient client(espClient);
Servo myservo;

int pos = 0;

void setup() {
  // Inicializace sériové komunikace
  Serial.begin(57600);

  // Načtení certifikátu CA
  BearSSL::X509List cert(ca_cert);
  espClient.setTrustAnchors(&cert);

  // Načtení certifikátů klienta a klíče klienta
  BearSSL::X509List client_crt(client_cert);
  BearSSL::PrivateKey key(client_key);
  espClient.setClientRSACert(&client_crt, &key);

  // Nastavení momentálního času – nutné pro TLS připojení
  time_t t = now();
  espClient.setX509Time(t);

  // Nastavení fingerprintu
  espClient.setFingerprint(fingerprint);

  // Inicializace Wi-Fi
  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.println("Connecting to WiFi..");
  }

  Serial.println("Connected to the WiFi network");

  // Nastavení serveru MQTT
  client.setServer(mqtt_broker, mqtt_port);

  // Nastavení callbacku – funkce, která se stará o zpracování zpráv
  client.setCallback(callback);
```

```
  // Smyčka pro připojení k MQTT brokeru
  while (!client.connected()) {
    String client_id = "ESP1-";
    client_id += String(WiFi.macAddress());
    Serial.printf("The client %s connects to the public mqtt broker\n",
client_id.c_str());
    if (client.connect(client_id.c_str(), mqtt_username, mqtt_password))
{
      Serial.println("Connected to MQTT broker!");
    } else {
      Serial.print("Failed with state ");
      Serial.print(client.state());
      delay(2000);
    }
  }

  myservo.attach(2);

  client.subscribe(heating_topic);
  client.subscribe(scenes_topic);

}
// Funkce callback
void callback(char *topic, byte *payload, unsigned int length) {

  // Vypsání nových zpráv na konzoli
  Serial.print("Message arrived in topic: ");
  Serial.println(topic);
  Serial.print("Message: ");

  String message;
  for (int i = 0; i < length; i++) {
    message = message + (char)payload[i];
  }

  Serial.println(message);

  Serial.println("-----------------------");
```

```
  // Pro téma f3/r7/controls/heating
  if (strcmp(topic, heating_topic) == 0) {
    int new_pos = message.toInt();
    // Přemapování hodnoty z rozsahu 0-5 na rozsah 0-180
    pos = map(new_pos, 0, 5, 0, 180);
    myservo.write(pos);
    delay(15);
  }

  // Pro téma f3/r7/scenes
  if (strcmp(topic, scenes_topic) == 0) {

    if (message == "#night") {
      pos = 3;
      // Přemapování hodnoty z rozsahu 0-5 na rozsah 0-180
      pos = map(pos, 0, 5, 0, 180);
      myservo.write(pos);
    }
    // V případě morning se nic neděje
    if (message == "#morning") {
      return;
    }
  }
}

void loop() {
  client.loop();
}
```

Příloha č. 6 – Odkaz na GitHub repositář

GitHub repositář: https://github.com/honzahoff/smart_implementace_iot.git

Oskenované zadání práce

UNIVERZITA HRADEC KRÁLOVÉ
Fakulta informatiky a managementu
Akademický rok: 2021/2022

Studijní program: Aplikovaná informatika
Forma studia: Prezenční
Obor/kombinace: Aplikovaná informatika (ai3-p)

Podklad pro zadání BAKALÁŘSKÉ práce studenta

Jméno a příjmení:	**Jan Hoffmann**
Osobní číslo:	**I1900186**
Adresa:	Černilov 509, Černilov, 50343 Černilov, Česká republika
Téma práce:	Návrh zabezpečené sítě IoT zařízení pro SMART implementace
Téma práce anglicky:	Design of secure IoT device network for SMART implementation
Jazyk práce:	Čeština
Vedoucí práce:	Ing. Pavel Blažek, Ph.D. Katedra informačních technologií

Zásady pro vypracování:

Při dodržení šablony závěrečných prací, zpracovat téma datových sítí a jejich implementaci na sítě sdružující IoT zařízení s přihlédnutím ke specifické konstrukci IoT prvků a nutnosti jejich zabezpečení.
Datové sítě
IoT zařízení – konstrukce, komunikace a šifrování
Komunikační protokoly IoT
Návrh zabezpečené sítě pro IoT zařízení

Seznam doporučené literatury:

1, Patel Ch., Internet of Things Security, CRC Press, 2019, ISBN 978-1-138-31863-2
2, Ravulavaru Arvind, Enterprise internet of things handbook, Birmingham : Packt, 2018, ISBN 978-1-78883-839-9
3, David Hanes at all, IoT fundamentals : networking technologies, protocols, and use cases for the internet of things, Cisco Press, 2017, ISBN 978-1-58714-456-1

Podpis studenta: Datum:

Podpis vedoucího práce: Datum: