Univerzita Pardubice

Fakulta elektrotechniky a informatiky

Škálovatelný dashboard

Silvia Čmilanská

Bakalářská práce

2022

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: 2021/2022

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Silvia Čmilanská**
Osobní číslo:	**I19074**
Studijní program:	**B2646 Informační technologie**
Studijní obor:	**Informační technologie**
Téma práce:	**Škálovatelný dashboard**
Zadávající katedra:	**Katedra informačních technologií**

Zásady pro vypracování

Cílem práce je vytvoření přehledného dashboardu, který bude zobrazovat data načtená ze senzorů. Informace, zobrazované na dashboardu, bude možné definovat pomocí uživatelského dialogu. Výstupní data bude možné zobrazit i pomocí grafů.

Rozsah pracovní zprávy: **min. 30 stran**
Rozsah grafických prací: **–**
Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam doporučené literatury:

CAMERON, Neil. Electronics Projects with the ESP8266 and ESP32:: Building Web Pages, Applications, and WiFi Enabled Devices. Apress;, 2020. ISBN 978-1484263358.
KONDOOR, Kondoor. *Kick-Start to MicroPython using ESP32 / ESP8266*. Kamataka: Independently published, 2021. ISBN 979-8574626757.

Vedoucí bakalářské práce: **Ing. Soňa Neradová, Ph.D.**
Katedra informačních technologií

Datum zadání bakalářské práce: **17. prosince 2021**
Termín odevzdání bakalářské práce: **13. května 2022**

L.S.

Ing. Zdeněk Němec, Ph.D. v.r.
děkan

Ing. Jan Panuš, Ph.D. v.r.
vedoucí katedry

V Pardubicích dne 28. února 2022

Prohlašuji:

Tuto práci jsem vypracovala samostatně. Veškeré literární prameny a informace, které jsem v práci využila, jsou uvedeny v seznamu použité literatury.

Byla jsem seznámena s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Beru na vědomí, že v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů, a směrnicí Univerzity Pardubice č. 9/2012, bude práce zveřejněna v Univerzitní knihovně a prostřednictvím Digitální knihovny Univerzity Pardubice.

V Pardubicích dne 1. 8. 2022

Silvia Čmilanská

Poděkování

Tímto bych ráda poděkovala své vedoucí bakalářské práce Ing. Soně Neradové, Ph.D. za poskytnutí námětu, ochotu a cenné rady při zpracování této práce. Dále bych chtěla poděkovat také své rodině a přátelům za pochopení a podporu v průběhu celého studia.

ANOTACE

Cílem bakalářské práce je návrh a vytvoření přehledného dashboardu, který zobrazuje data načtená ze senzorů. Informace zobrazované na dashboardu je možné definovat pomocí uživatelského dialogu. Výstupní data je možné zobrazit i pomocí grafů. Tento systém je vytvořen na platformě Raspberry Pi. Pro komunikaci mezi zařízeními využívá protokol Message Queuing Telemetry Transport. Teoretická část práce je věnována představení technologií, které byly využity pro implementaci praktické části - technologie internet věcí, komunikační protokoly využívané v IoT, jazyk Python, konkrétně webový framework Django a nástroj Docker. V praktické části je popsán návrh a implementace webové aplikace a použité hardwarové komponenty.

KLÍČOVÁ SLOVA

Internet věcí, Raspberry Pi, ESP32, MQTT, Django, dashboard, senzory

TITLE

Scalable dashboard

ANNOTATION

The bachelor thesis aims to design and create a clear dashboard that displays data read from sensors. The information displayed on the dashboard can be defined using a user dialog. Output data can also be displayed using graphs. This system is created on the Raspberry Pi platform. It uses the Message Queuing Telemetry Transport protocol to communicate between devices. The theoretical part of the work is devoted to the introduction of technologies that were used for the implementation of the practical part. The Internet of Things technologies, communication protocols used in IoT, the Python language, specifically the Django web framework and the Docker tool are described here. The practical part describes the design and implementation of the web application and the hardware components used.

KEYWORDS

Internet of Things, Raspberry Pi, ESP32, MQTT, Django, dashboard, sensors

OBSAH

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM ZKRATEK

ARPANET	Advanced Research Projects Agency Network
BLE	Bluetooth Low Energy
CoAP	Constrained Application Protocol
CSMA/CA	Carrier Sense Multiple Access/Collision Avoidance
DAS	Data Aquisition System
GPIO	General Purpose Input/Output
HTTP	Hypertext Transfer Protocol
IEEE	Institute of Electrical and Electronics Engineers
IoT	Internet of Things
IP	Internet Protocol
LAN	Local Area Network
M2M	Machine to Machine
MQTT	Message Queuing Telemetry Transport
MVC	Model-View-Controller
MVT	Model-View-Template
OOP	Object-Oriented Programming
ORM	Object–Relational Mapping
PAN	Personal Area Network
QoS	Quality of Service
SQL	Structured Query Language
SSL	Secure Sockets Layer
TCP	Transmission Control Protocol
TLS	Transport Layer Security
UDP	User Datagram Protocol
USB	Universal Serial Bus
VCC	Voltage Common Collector
Wi-Fi	Wireless Fidelity
WLAN	Wireless Local Area Network
WPA	Wi-Fi Protected Access
YAML	YAML Ain't Markup Language

ÚVOD

Internet věcí neboli IoT (Internet of Things) je v dnešní době jednou z nejrychleji se rozvíjejících technologií. S těmito technologiemi se běžně setkáváme v domácnostech, průmyslu nebo i ve zdravotnictví, kde uživatelům značně usnadňují život. Příkladem využití IoT může být chytrá domácnost, kde se v závislosti na denní době a světle automaticky rozsvěcí světla a zatahují rolety, reguluje se vlhkost a teplota vzduchu, automaticky se provádí kontrola vypnutí spotřebičů apod. V průmyslu může být IoT využito pro monitorování výroby, vzdálenou správu zařízení, nebo usnadnění zjištění stavu a údržby zařízení [1]. Pro tyto technologie bylo vytvořeno několik platforem umožňujících jejich jednoduchou správu a přehlednou vizualizaci výstupních dat.

Hlavním cílem této práce je návrh a tvorba přehledného dashboardu, který zobrazuje data načtená ze senzorů a umožňuje také jejich vizualizaci pomocí interaktivního grafu. Tento systém je implementován pomocí Docker kontejnerů běžících na platformě Raspberry Pi a data ze senzorů jsou čtena a odesílána ze zařízení ESP32. Pro komunikaci mezi zařízeními tento IoT systém využívá technologie Wi-Fi a protokol Message Queuing Telemetry Transport.

Teoretická část práce je věnována představení technologie internet věcí, kde popisuje jeho využití, historii, architekturu, komunikační modely z hlediska typu zařízení a z hlediska způsobu výměny zpráv. Dále jsou zde popsány používané komunikační protokoly technologie IoT na fyzické i aplikační vrstvě, kde je mimo jiné popsán i protokol MQTT. Následně jsou teoreticky představeny technolgie, které byly použity k implementaci praktické části. Jedná se o jazyk Python s jeho webovým frameworkem Django a Python knihovna Plotly pro tvorbu grafů. Poslední kapitola teoretické části je o technologii Docker, kde je představen koncept Dockeru spolu s kontejnerovou virtualizací, jeho komponenty a vysvětlení konfiguračních souborů Dockerfile a docker-compose.yaml.

Praktická část obsahuje popis požadovaných vlastností a návrh řešení spolu s hardwarovým a aplikačním schématem výsledného IoT systému. Další část je věnována popisu implementace systému, kde jsou popsány použité hardwarové komponenty a jejich zapojení. Následně je vysvětleno programování modulu ESP32, implementace Docker kontejnerů, databázový model, nastavení komponent MQTT protokolu a popis výsledné webové aplikace.

1 INTERNET VĚCÍ

Pojem internet věcí, známý pod zkratkou IoT, anglicky jako Internet of Things je označení pro technologii tvořící síť velkého množství propojených fyzických zařízení připojených k internetu. Tato chytrá zařízení vykonávají specifické činnosti nebo provádějí pravidelná měření. Výsledná data provedených měření jsou sdílena v síti ostatním zařízením, která sdílená data přijímají a interpretují je pro vykonání nějaké akce nebo pro prezentaci výsledků uživateli. [1]

Věci z hlediska IoT jsou zařízení tvořena elektronikou, software a senzory, díky nimž jsou prováděna měření. Výsledná data jsou poskytována pro následné zpracování zařízeními, která autonomně provádějí akce na základě obdržených sdílených dat. Jedná se tedy o autonomní systém schopný samostatně fungovat bez lidského přičinění a data v něm sdílená jsou vytvářena pouze neživými objekty. [1], [2]

1.1 Kde se IoT využívá

Internet věcí se v posledních letech rapidně rozšiřuje a dnes tuto technologii můžeme nalézt napříč odvětvími a to nejen v průmyslu, ale i v domácnostech pro automatizaci a zjednodušení běžných činností.

Příklady využití IoT v různých odvětvích jsou zpracovány podle zdroje[3]:

- **Zemědělství** - Využití chytrých zařízení v zemědělství přináší v posledních letech zvýšení celkové produkce i kvality. Konkrétními příklady použití mohou být například měření vlhkosti a množství živin v půdě, stanovení správné doby sklizně, nebo například vytvoření hnojiva na míru potřebám dané zeminy podle výsledků chemického rozboru dat, která byla nasbírána měřením chytrých zařízení.
- **Zdravotnictví** - IoT zařízení jsou ve zdravotnictví využívána nejčastěji pro zaznamenávání zdravotního stavu pacientů jako například měření krevního tlaku, hladiny cukru v krvi, váhy atp. K těmto informacím poté může přistupovat lékař v reálném čase, stejně tak i k historickým záznamům a díky tomu postupovat s léčbou dle zjištěných hodnot. Chytrá zařízení mohou také u pacienta monitorovat zda neupadl nebo jestli u něj nebyly naměřené kritické hodnoty vitálních funkcí. V takovém případně je na základě naměřených dat automaticky zavolána zdravotnická pomoc.

- **Chytrá města** - Zařízení připojená do infrastruktury internetu věcí pomáhají řízení a monitorování mnoha důležitých součástí běžného chodu města. Připojená zařízení jako například senzory, měřiče nebo světla ve městech přispívají ke zlepšení a automatizaci různých částí infrastruktury a veřejných služeb. Velké využití mají ve městech například měřiče spotřeby elektrické energie, díky kterým jsou energetické společnosti schopny efektivněji řídit dodávku energií dle aktuálního stavu spotřeby. Tyto chytré měřiče pomáhají také spotřebitelům s jejich financemi díky možnosti sledovat vlastní spotřebu energie. Městům pomáhá také implementace chytrého řešení nakládání s odpadem. Použití senzorů indikujících zaplněnost popelnic umožňuje velice efektivní plánování trasy sběru odpadu bez nutnosti zastávek ke sběru na místech, kde to ještě není potřeba. Dalšími příklady využití IoT ve městech může být například sledování znečištění ovzduší nebo využití chytrých zařízení v dopravě. [4]
- **Chytrá domácnost** - Technologie internetu věcí si v domácnostech našla své místo například pro různá měření jako je monitorování spotřeby, teploty, vlhkosti, nebo také pro automatizaci různých činností jako vzdálené ovládání oken, žaluzií, klimatizace, světel, propojení zařízení s chytrými reproduktory atp. Příkladem komplexnější implementace IoT v domácnosti může být automatický zavlažovací systém fungující podle výsledků měření vlhkosti, nebo automaticky regulovaná klimatizace dle naměřené teploty po uživatelově vzdáleném požadavku pro automatickou regulaci optimální teploty před příchodem uživatele domů.
- **Nositelná zařízení** - Velice populární jsou i malá chytrá nositelná zařízení jako například hodinky, prstýnky nebo různé nalepovací senzory na tělo. Tato zařízení často ve spojení s mobilními aplikacemi poskytují běžným uživatelům detailní přehled o jejich fyzické aktivitě, kvalitě spánku a mnoha dalších informacích o jejich zdravotním stavu, které následně mohou konzultovat s lékařem.

1.2 Historie

Tato kapitola byla zpracována podle zdroje [5].

Ačkoliv technologie internetu věcí zažívá největší rozvoj až v posledních několika letech, tak samotný koncept připojených zařízení mezi sebou v síti má kořeny již v první polovině 19. století, kdy byl vynalezen první elektromagnetický telegraf. Dalším velkým

krokem, který položil základy vývoje internetu věcí byl vynález prvního rádiového hlasového přenosu na začátku 20. století. [6] Opravdové počátky IoT však přišly až s počátkem internetu.

O první připojené zařízení se přičinili programátoři z univerzity Carnegie Mellon University v 80. letech 20. století. Jednalo se o automat na nápoj Coca Cola umístěný v areálu univerzity. Těmto studentům se podařilo automat připojit do sítě ARPANET, aby mohli vzdáleně kontrolovat teplotu a množství nápojů v automatu. Tento vynález poté inspiroval výzkum a vývoj dalších vzájemně propojených přístrojů po celém světě.

Dalším krokem vpřed ve vývoji podoby internetu věcí jak ho známe dnes byl rok 1990, kdy John Romkey a Simon Hackett připojili toastovač k internetu pomocí TCP/IP protokolu. Zařízení bylo doplněno automatickou rukojetí, která uměla dálkově do toastovače vložit a vyjmout chléb místo uživatele. Pro doplnění chleba do automatické rukojeti bylo však zapotřebí zásahu uživatele, nejednalo se tedy ještě o plnohodnotné IoT zařízení. [6] O rok později přišli vědci z univerzity Cambridge s myšlenkou využití jejich prototypu první webkamery k monitorování aktuálního množství kávy v kávovaru, který byl přítomen v jejich laboratoři. Tento návrh realizovali naprogramováním webkamery k pořizování snímků v pravidelných časových intervalech několikrát za minutu a následně k odesílání pořízených snímků na počítače v lokální univerzitní síti tak, aby každý uživatel mohl pohodlně na dálku zkontrolovat zda je v kávovaru ještě káva.

Největším historickým milníkem vzniku internetu věcí byl však rok 1999, kdy technologický průkopník Kevin Ashton během prezentace pro firmu Procter & Gamble poprvé zmínil pojem „The Internet of Things“ jako pojmenování pro technologii několika připojených zařízení pomocí identifikace na rádiové frekvenci. Tato zařízení měla pomoci při řízení dodavatelského řetězce. Proto je Kevin Ashton také někdy nazýván otcem technologie IoT.

V následujícím desetiletí zájem veřejnosti o technologii IoT stoupal velice rychle, jelikož na trh byly postupně uváděny nové technologie a různá nová připojená zařízení. V roce 2000 byla firmou LG uvedena na trh chytrá lednička, umožňující provádět videohovory a online nakupování. Dalším velkým krokem bylo vydání prvního iPhonu v roce 2007. Rok 2008 je oficiálně považován za rok vzniku pravého internetu věcí, jelikož v té době počet připojených zařízení překročil počet lidí na zemi. V následujících letech vznikaly výrobky jako například chytrý termostat pro vzdálené ovládání teploty v domě, různé druhy IoT

senzorů, atp. Postupně se IoT také integrovalo do téměř každého odvětví průmyslu, začala vznikat chytrá města a to až doposud, kdy má rozvoj technologie internetu věcí stále velice rychlé tempo.

1.3 Architektura

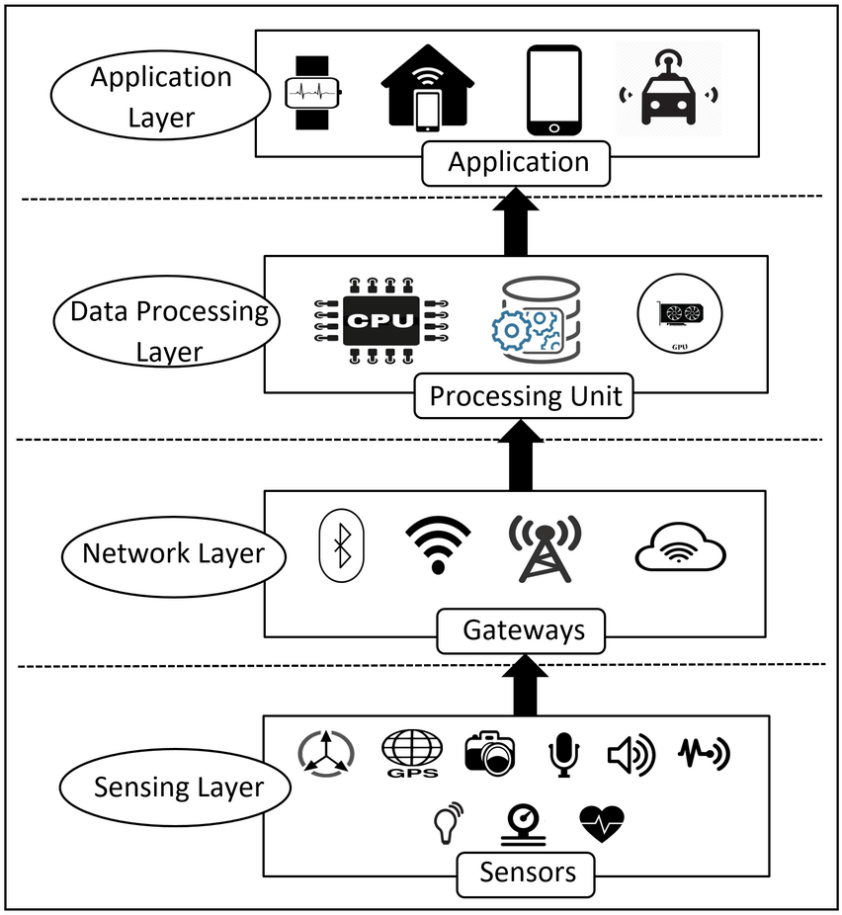
Množství způsobů využití internetu věcí neustále roste, tedy implementace této technologie musí být vždy přizpůsobena konkrétním potřebám. Neexistuje proto žádná striktně předepsaná architektura, která by při implementaci měla být dodržována. IoT zařízení a aplikace mohou být popsány jako síť věcí, které jsou složeny z hardwaru, softwaru, síťového připojení a senzorů. Na první vrstvě jsou prvky, které přijímají data ze senzorů nebo je samy generují, dále jsou data převedena do digitální podoby a odeslána k dalšímu zpracování, analýze a následné archivaci. Tento ekosystém lze obecně popsat pomocí čtyř hlavních architektonických pilířů odvozených podle toku dat v IoT systému. Tuto všeobecnou architekturu internetu věcí lze tedy rozdělit na vrstvu snímání, síťovou vrstvu, vrstvu zpracování dat a aplikační vrstvu. [8], [9]

Na obrázku 1 lze vidět jednotlivé vrstvy IoT architektury, které jsou popsány níže:

- **Vrstva snímání** - Mezi prvky vrstvy snímání patří především senzory, akční prvky a zařízení. Pokud se jedná o senzory, tak je hlavním cílem této vrstvy identifikovat veličinu a automaticky získat data o prostředí z fyzického zařízení pomocí měření. Senzory zaznamenávají data ohledně aktuálního stavu daného procesu nebo provádějí měření podmínek okolního prostředí jako jsou vlhkost, teplota, rychlost, svítivost, množství kapaliny v nádrži, rychlost průtoku kapaliny a mnoho dalších. Akční prvky jsou koncová zařízení, která neustále naslouchají a očekávají pokyn od vzdáleného zařízení k nějaké akci. Velmi častá je také spolupráce senzorů a akčních prvků k vykonání nějaké akce, která je podmíněna určitými podmínkami vnějšího prostředí. To znamená, že pokud je na senzoru naměřena požadovaná hodnota podmínek vnějšího prostředí, která vyžaduje okamžitou akci, tak je odeslána zpráva akčnímu prvku, který poté akci provede v reálném čase. [8], [10]
- **Síťová vrstva** - Tato vrstva funguje jako komunikační kanál propojující všechna zařízení k přenosu dat, která byla získána z předchozí vrstvy snímání. Hlavními prvky této vrstvy jsou Data Aquisition System (DAS) neboli systém sběru dat a síťové brány. Systém sběru dat slouží ke sběru dat ze senzorů a také k převodu analogového

formátu dat do digitální podoby. Poté jsou data odeslána do sítě. Síťová vrstva může pro přenos načtených dat k ostatním zařízením implementovat různé komunikační technologie jako například Wi-Fi, Bluetooth, Zigbee, Z-Wave, atp. V této fázi je kumulováno největší množství dat, jelikož jsou data přijímána paralelně z mnoha senzorů zároveň. Proto je zde potřeba zavést také určité filtrování potřebných dat a jejich případnou komprimaci pro optimalizaci přenosu. [8], [9], [10]

- **Vrstva zpracování dat** - V této fázi je prováděno hlavní zpracovávání dat a jejich analýza pro potřeby daného IoT systému. Je zde implementováno filtrování nepotřebných informací z přijatých digitalizovaných dat podle požadavků pro následnou analýzu. Na základě výsledků analýzy mohou být provedeny kroky k vykonání dalších akcí v rámci IoT systému. To znamená, že výsledky lze odeslat pomocí síťové vrstvy ostatním připojeným zařízením, které vykonají podmíněnou akci. Ke zlepšení funkce celého IoT systému je možné použít strojové učení, díky kterému se rychleji zpracuje zpětná vazba o efektivitě a tím lze snadněji zlepšit funkci daného ekosystému. Zpracovaná data a výsledky analýzy jsou následně odeslány do datového úložiště, odkud jsou poté dostupná pro vizualizaci v aplikaci. [8], [10]
- **Aplikační vrstva** - Posledním prvkem obecné architektury technologie internetu věcí je aplikační vrstva. Jsou zde prezentovány výsledky dostupné z předchozí vrstvy zpracování dat pomocí přehledných vizualizací například ve formě dashboardu. Tato vrstva je orientována na koncové uživatele a mimo prezentaci výsledků měření a analýzy poskytuje uživateli ovládací prvky k provedení požadovaných akci v rámci IoT systému. Jak lze vidět na obrázku 1, tak IoT aplikace lze využívat na různých zařízeních a tyto aplikace mohou spravovat systémy od chytrých domácností až po například chytrou dopravu. [8]



Obrázek 1: Obecná architektura IoT. [7]

1.4 Komunikační modely

V IoT systému je propojeno mnoho různých typů zařízení, které zastupují odlišné role při síťové komunikaci. Způsob připojení a komunikace zařízení mezi sebou z hlediska provozu lze popsat pomocí komunikačních modelů. Proto v roce 2015 vydal výbor Internet Architecture Board (IAB) oficiální dokument pojednávající o síťovém provozu mezi chytrými zařízeními. Dokument popisuje hlavní čtyři komunikační modely využívané IoT zařízeními. Tyto modely jsou základním popisem způsobu, jak v systému dochází ke sběru, sdílení a zpracování dat. [11], [12]

Popis jednotlivých komunikačních modelů z hlediska typu zařízení dle IAB:

- **Device-to-Device** - Model zařízení k zařízení popisuje dvě nebo více zařízení ve stejné síti, která jsou k sobě připojena na přímo bez prostředníka. K připojení dochází nejčastěji pomocí PAN protokolů jako jsou Bluetooth, Zigbee nebo Z-Wave. Je však možné použít komunikaci i přes Internet. Komunikační model Device-to-

Device je využíván nejčastěji v chytrých domácnostech, a to u chytrých žárovek, termostatů, zámků atp. Zpravidla jsou mezi těmito zařízeními posílána pouze malá množství dat zajišťující jednoduché akce. Nevýhodou protokolů implementujících tento model je častá vzájemná nekompatibilita. Například zařízení využívající protokol Zigbee nejsou nativně kompatibilní se zařízeními využívající Z-Wave protokol. Proto je nutné při implementaci chytré domácnosti zvážit použití zařízení využívajících ideálně jednotného komunikačního protokolu. [11], [12]

- **Device-to-Cloud** - Model popisující připojení zařízení do cloudu využívá ke komunikaci Ethernet, Wi-Fi nebo mobilní síť. Jedná se o přímé připojení zařízení k aplikační službě běžící v cloudu. Tento model se využívá k výměně dat, řízení provozu zpráv a pro komunikaci mezi různými systémy. Příkladem použití může být zařízení zaznamenávající údaje o osobním automobilu jako je rychlost, poloha, zrychlení atd. Naměřená data jsou následně odeslána do cloudu, kde jsou analyzována a následně mohou být využita pojišťovnami k vytvoření pojistek odpovídajících výsledným hodnotám. [11], [12]
- **Device-to-Gateway** - Tento model popisuje připojení zařízení do cloudu s využitím brány k řízení komunikace mezi různými sítěmi a komunikačními technologiemi. To znamená, že komunikaci mezi zařízením a cloudem zprostředkovává aplikační software běžící na lokální bráně. To IoT systému nabízí i další možnosti jako implementaci pokročilejšího zabezpečení a překlad komunikace napříč různými protokoly. [11]
- **Cloud-to-Cloud** - Cloud to cloud architektura je nazývána také jako Back-end Data-Sharing model. Tento systém je navržen tak, aby umožňoval přístup k naměřeným datům i třetím stranám. To znamená, že s využitím tohoto modelu lze získat data z cloudu k provedení analýzy v kombinaci s daty z dalších zdrojů. Tento systém je rozšířením modelu Device-to-Cloud. [11]

Rozdělení komunikačních modelů dle způsobu výměny zpráv je zpracováno dle zdroje [13]:

- **Request-Response model** - Tento model funguje podle klient-server architektury. To znamená, že klient jakožto IoT zařízení vytváří požadavky a server na ně následně odpovídá. Server tedy po přijetí požadavku klienta získá požadovaná data, vytvoří odpověď a pošle ji zpět klientovi. Jedná se o bezstavový model, jelikož požadavky

jsou zpracovávány nezávisle na sobě a data nejsou mezi jednotlivými požadavky nikde ukládána.

- **Publisher-Subscriber model** - Tento model funguje na základě spolupráce tří součástí, kterými jsou publisher, broker a konzumer. Publisher je zdrojem dat v systému, tedy vytváří data a odesílá je na vybrané téma nezávisle na existenci konzumera. Broker je zařízení v postavení serveru, které spravuje všechna témata a řídí provoz zpráv v systému. Jeho úkolem je přijímat data od publisherů a odesílat je dle tématu správnému konzumerovi. Konzumer odebírá zprávy stanoveného tématu, které jsou mu doručeny prostřednictvím brokeru. Konzumeři ani publisheři o sobě vzájemně v systému nevědí a veškerá komunikace je zprostředkována výhradně prostřednictvím brokeru.
- **Push-Pull model** - Model push-pull je složen ze součástí publisher, konzumer a datové fronty. Publisher publikuje data a vloží je do fronty. Data jsou následně z fronty postupně odebírána konzumerem. Publisher a konzumer o sobě v systému vzájemně nevědí. Fronta v tomto komunikačním modelu funguje jako buffer, který řeší problém rozdílné rychlosti sdílení dat publisherem a přijímání dat konzumerem.
- **Exclusive pair** - Tento model je obousměrný a implementuje plně duplexní komunikaci mezi klientem a serverem. Připojení je neustále otevřené dokud není odeslán požadavek klienta na ukončení spojení. Server si drží záznamy o všech otevřených spojeních, jedná se tedy o stavový model. Jakmile je spojení vytvořeno, tak jsou v systému vyměňovány zprávy mezi klientem a serverem. [13], [14]

1.5 Komunikační protokoly fyzické vrstvy

Pro implementaci systému internetu věcí jsou převážně využívána bezdrátově připojená zařízení, proto jsou zde následně popsány pouze bezdrátové komunikační protokoly fyzické síťové vrstvy. Nejčastěji používanými standardy pro komunikaci v systému IoT jsou Wi-Fi, Bluetooth, Zigbee a Z-Wave. Výběr mezi nimi záleží v požadavcích na systém a liší se zejména v přenosové vzdálenosti, spotřebě energie a přenosové rychlosti dat.

1.5.1 Wi-Fi

Standard Wi-Fi je v dnešní době nejpoužívanější technologií pro bezdrátovou komunikaci. Vychází ze standardu 802.11, který byl vydán institucí IEEE v roce 1997 jako úplně první WLAN standard. Od té doby se neustále vyvíjí pro zvýšení přenosových rychlostí a efektivity. Tato technologie je využívána zejména v domácnostech a ve firemních sítích. [15]

Různé verze standardu 802.11 se liší hlavně v rychlosti přenosu dat a frekvenci, na které komunikace probíhá. Rychlost přenosu dat v síti je udávána v jednotkách Mbit/s, tj. megabity za sekundu. Frekvence popisuje na jaké rádiové frekvenci jsou přenášena data v dané síti. Standardními frekvenčními pásmy pro Wi-Fi jsou 2,4 GHz a 5 GHz. [16]

První verze standardu 802.11 z roku 1977 poprvé definovala protokol, který umožňoval kompatibilitu bezdrátové komunikace zařízení v místní síti LAN. Implementuje protokol CSMA/CA, který umožňuje zkontrolovat, zda je kanál volný ještě před zahájením přenosu dat. Tím se během komunikace předchází kolizím. Tato verze standardu fungovala na frekvenci 2,4 GHz s maximální propustností 2 Mbit/s. Tento protokol měl velké nedostatky kvůli problémům s nedostatečnou propustností a vysokou cenou zařízení. [15]

Od té doby vyšlo mnoho dalších verzí protokolu 802.11 s označeními *b, a, g, n, ac* a *ax*. Každá nová verze přinesla zlepšení v rychlosti přenosu dat a v šířce pásma. V roce 1999 vyšly verze standardu 802.11b a 802.11a. Verze *b* fungovala na frekvenci 2,4 GHz s maximální přenosovou rychlostí 11 Mbit/s a verze *a* byla provozována na frekvenčním pásmu 5 GHz s přenosovou rychlostí až 54 Mbit/s. Standard 802.11g byl první verzí, která byla využívána pro komerční účely a implementovala kombinaci dobrých vlastností z předchozích verzí *b* a *a*. V roce 2009 vyšla verze 802.11n fungující na pásmech 2,4 GHz a 5 GHz, která přinesla velký posun v rychlosti přenosu dat a to až na 600 Mbit/s. Následující verze 802.11ac, která vyšla v roce 2013 uvedla poprvé přenosovou rychlost v řádech gigabitů. Nazývá se také jako Wi-Fi 5, funguje na stejných pásmech jako verze 802.11n, rychlostně však může dosahovat až 6,8 Gbit/s. Tato verze je dnes komerčně nejpoužívanější, avšak postupně se do popředí začíná dostávat nová verze standardu 802.11ax pod názvem Wi-Fi 6 s maximální přenosovou rychlostí 9,6 Gbit/s díky efektivnější modulaci a poskytuje také vylepšené zabezpečení pomocí WPA3. [15], [17]

1.5.2 Bluetooth

Bluetooth je jedna z nejpoužívanějších bezdrátových technologií k přenosu dat na krátkou vzdálenost. Jedná se o standard 802.15.1 vydaný institutem IEEE pro nízkoenergetickou rádiovou komunikaci. Pracuje na rozsahu frekvenčního pásma 2,4-2,485 GHz s dosahem signálu dle prostředí od desítek metrů až po 100 m. Dané frekvenční pásmo je rozděleno do 79 kanálů, kde každý kanál má šířku 1 MHz. Signál je následně vysílán kontinuálně s využitím sekvence kanálů, která je známá vysílajícímu i přijímajícímu zařízení. Klasické Bluetooth má přenosovou rychlost 1-3 Mbit/s. [18], [19]

Internet věcí má dedikovanou verzi Bluetooth, která byla vydána v roce 2010 jako verze Bluetooth 4 taktéž pod názvem Bluetooth LE (Low Energy) nebo Bluetooth Smart. Tato nízkoenergetická verze využívá pouze setinu potřebné energie pro provoz klasické verze Bluetooth, tedy přibližně kolem 0,01-0,50 W. Jedná se o přenos malého množství dat v přerušovaných krátkých časových intervalech rychlostí do 2 Mbit/s. Bluetooth LE nachází využití ve fitness trackerech, chytrých hodinkách, zdravotnických zařízeních jako například v inzulínových pumpách, také pro sledování polohy uvnitř budov, atp. [20]

1.5.3 Zigbee

Zigbee je komunikační technologie pro bezdrátový přenos dat u nízkoenergetických IoT sítí s nízkými náklady na provoz. Tato technologie je vyvinuta dle standardu IEEE 802.15.4 z roku 2003 a pracuje na frekvenčních pásmech 2,4 GHz, 902-928 MHz a 868 MHz. Signál má dosah v rozsahu kolem 10-100 m a dosahuje přenosové rychlosti kolem 20-250 Kbit/s. Své využití má Zigbee hlavně v průmyslu, kde se implementuje u zařízení napájených baterií, která vydrží ve funkci bez zásahu člověka i několik let. Tento protokol umí sjednocovat připojení zařízení různých výrobců a při propojení Zigbee sítě k doméně IP je možné dálkově ovládat a monitorovat daná propojená zařízení v sítích LAN, WAN nebo přes Internet. Tato funkcionalita vytváří infrastrukturu pravého internetu věcí. [21], [22]

Technologie Zigbee obsahuje tři hlavní komponenty, kterými jsou koordinátor, router a koncové zařízení. Koordinátor musí být přítomen v každé Zigbee síti a funguje jako most mezi sítí Zigbee a externí sítí. Během přenosu dat se chová jako hub a přijímá a ukládá důležité informace. Router zde má funkci prostředníka mezi koordinátorem a koncovým zařízením. To znamená, že umožňuje koordinátorovi doručit komunikaci ke správnému

koncovému zařízení. Koncová zařízení jsou zdroji dat a jejich přístup k nadřazeným prvkům sítě je velice omezený pro ušetření spotřeby energie. Struktura Zigbee sítě je obvykle implementována pomocí topologie hvězda, mesh nebo strom. [22]

1.5.4 Z-Wave

Z-Wave je bezdrátová komunikační technologie, která byla navržena přímo pro aplikace spravující řízení, monitorování a čtení dat ze senzorů nebo chytrých zařízení, převážně v chytrých domácnostech a menších komerčních prostředích. Tento protokol byl vynalezen Dánskou firmou Zensys v roce 1999. Následně se technologie rozšířila do Spojených Států v roce 2002 a stala se uznávaným standardem pro domácí automatizaci. [23], [24]

Z-Wave umožňuje komunikaci zařízením s velice malou spotřebou energie a implementuje mesh topologii bez potřeby prvku koordinátora v síti. Funguje na frekvenčním písmu pod 1 GHz, to znamená, že se vyhýbá rušení na hustě obsazeném pásmu 2,4 GHz, na kterém funguje většina z výše zmíněných bezdrátových technologií. Rychlost přenosu dat se pohybuje do 100 kbit/s a přenášená data jsou šifrována pomocí AES128 šifrování. Dosah signálu je kolem 100 m. [23]

1.6 Komunikační protokoly aplikační vrstvy

1.6.1 HTTP

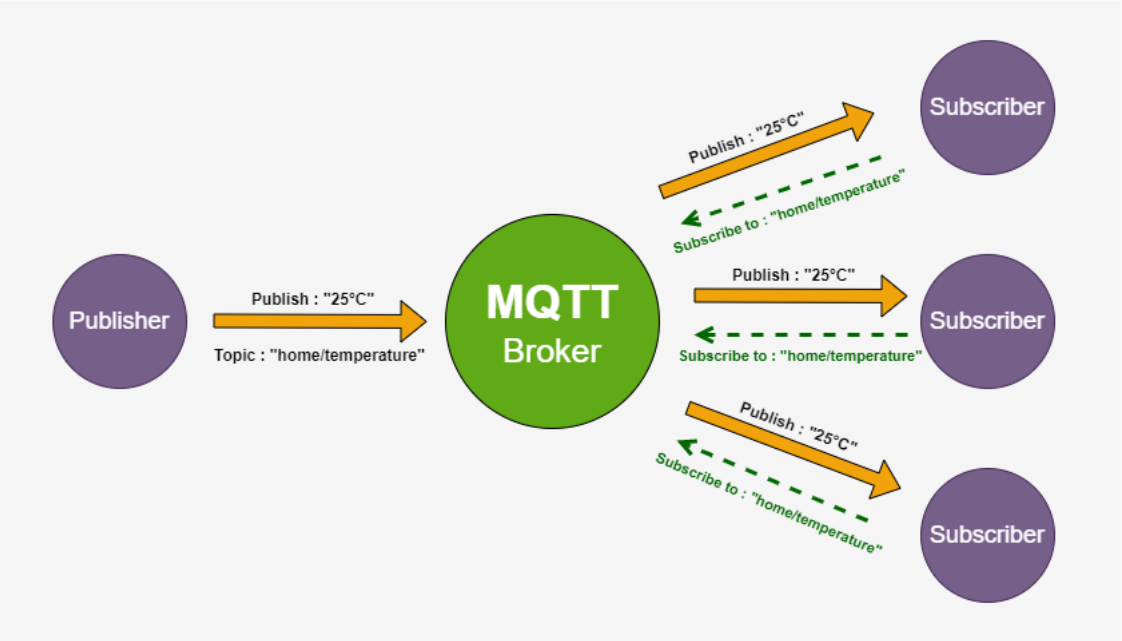
HTTP, neboli Hyper Text Transfer Protocol, je jedním z nejznámějších komunikačních protokolů aplikační vrstvy. Vytvořil jej Tim Berners-Lee ve středisku CERN v roce 1989. Protokol HTTP je využívám službou World Wide Web k přenosu dat již od roku 1990. [25]

Protokol funguje na architektuře klient-server a využívá komunikační model request-response. To znamená, že při zadání url adresy do prohlížeče je odeslán http požadavek na server, kde je zpracován a následně je serverem odeslána odpověď zpět klientovi. Protokol HTTP je bezstavový a díky jeho komplexitě ho lze použít i pro jiné účely než pouze na přenos dokumentů, například pro přenos binárních dat. Je založený na TCP/IP a využívá TCP port 80, ovšem lze použít i jiné porty. Protokol HTTP neposkytuje zabezpečení, proto vznikla verze protokolu HTTPS pracující na TCP port 443. Ten kombinuje HTTP s protokolem SSL nebo TLS pro poskytnutí šifrované komunikace. [25], [26]

1.6.2 MQTT

Protokol MQTT, celým názvem Message Queuing Telemetry Transport, je protokolem pro komunikaci typu machine to machine (M2M), tzn. pro komunikaci mezi dvěma přístroji v rámci technologie internetu věcí. Je velice nenáročný na provoz, dobře škálovatelný, byl vytvořen pro zařízení s malou šířkou pásma a nízkou spotřebou energie a podporuje šifrování zpráv pomocí SSL/TLS. To z něj činí ideální protokol pro využití v IoT aplikacích. [28]

Tento protokol byl vytvořen v roce 1999 Andym Stanford-Clarkem a Arlenem Nipperem. Protokol byl tehdy vytvořen pro připojení k ropovodu přes satelit s minimální spotřebou baterie a minimální šířkou písma. Hlavními požadavky na protokol byly: jednoduchá implementace, podpora QoS při doručování zpráv, nenáročnost na provoz a šířku pásma, dále aby nezáleželo na formátu posílaných dat a typu zařízení které se bude účastnit komunikace a také aby si udržoval informace o dané relaci. Od svého vydání byl protokol proprietární pouze pro využití firmou IBM. Až v roce 2010 byla vydána verze MQTT 3.1, která již byla bez poplatku a volně k použití. První open-source MQTT broker byl však vydán již v roce 2008 společností Eclipse pod názvem Mosquitto. [29], [30]



Obrázek 2: MQTT protokol. [27]

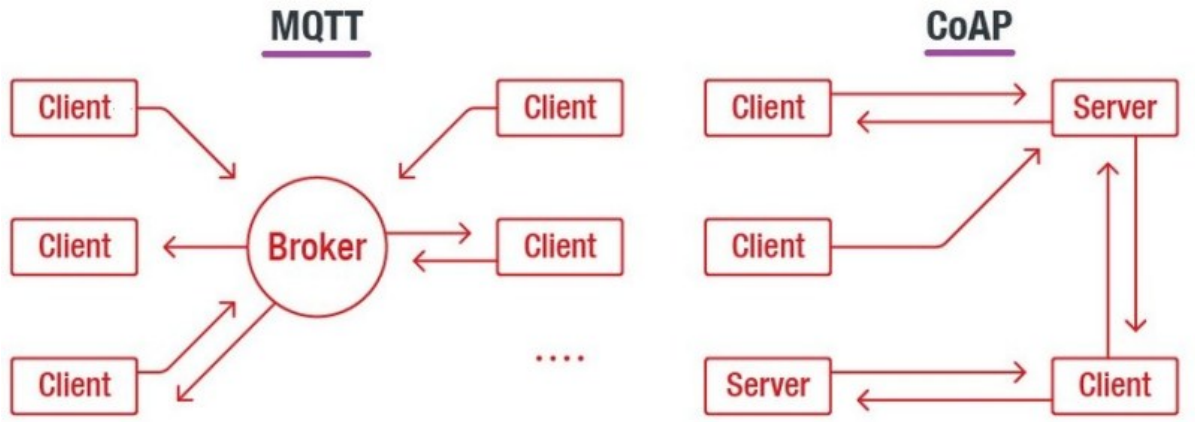
MQTT protokol funguje dle komunikačního modelu publisher-subscriber a je postaven na TCP/IP. Běžný provoz využívá port 1883 a šifrovaný provoz běží na portu 8883. Hlavními součástmi protokolu jsou zpráva, téma, klient a server. Následující přehled je zpracován dle zdroje [28]:

- **Zpráva** - MQTT zpráva obsahuje posílaná data a přídavné informace uspořádané do specifického formátu. Na začátku zprávy jsou odesílaná data, dále je uvedena informace o QoS, poté seznam vlastností obsahující metadata a nakonec název tématu, ve kterém je zpráva posílána. Zprávy jsou posílány v binární podobě.
- **Téma** - Téma, neboli MQTT topic je označení sloužící pro kontrolu, kterému klientovi typu subscriber má broker přijatou zprávu přesměrovat.
- **Klient** - V architektuře protokolu MQTT mají roli klienta dva typy zařízení, a to subscriber a publisher. Publisher je zařízení, které čte nebo generuje data a následně je odesílá brokerovi na konkrétní téma pomocí operace publish. Zařízení s rolí subscriber je také klient, ale na rozdíl od publishera data přijímá a to pomocí operace subscribe na konkrétní témata odeslaná na broker.
- **Server** - Zařízení v roli serveru se nazývá broker. Jeho úkolem je umožnit klientům sdílet a odebírat zprávy na požadovaných tématech. To znamená, že broker spravuje připojení všech klientů, přijímá zprávy odeslané operací publish, zpracovává požadavky na subscribe a unsubscribe operace, přesměrovává zprávy správným klientům a uzavírá spojení s klienty.

1.6.3 CoAP

CoAP, celým názvem Constrained Application Protocol, je podobně jako HTTP komunikační protokol pro přenos dokumentů. Avšak na rozdíl od protokolu HTTP je CoAP vytvořen přímo pro potřeby zařízení s omezenými zdroji, která jsou používána pro IoT. Proto je vhodný na použití pro zařízení, která neumí komunikovat přes HTTP nebo TLS, jako jsou 8-bitové mikrokontrolery nebo jednoduché senzory. [32], [33]

Protokol CoAP běží na UDP, na rozdíl od ostatních protokolů stejné kategorie jako HTTP nebo MQTT, které fungují na TCP. Funguje dle komunikačního modelu klient-server, tedy klient posílá požadavky a server na ně reaguje a odesílá zpět odpovědi. Dále podobně jako protokol HTTP používá operace GET, PUT, POST a DELETE. Je vytvořen k dobré spolupráci s protokolem HTTP a REST API. [32]



Obrázek 3: Srovnání architektury protokolů CoAP a MQTT. [31]

2 PYTHON

Tato kapitola byla zpracována podle zdrojů [34], [35].

Python je velmi používaný, interpretovaný, interaktivní, objektově orientovaný skriptovací vyšší programovací jazyk s dynamickou sémantikou. To, že je jazyk interpretovaný znamená, že jej není třeba předem kompilovat, ale je zpracováván až za běhu programu. Interaktivita Pythonu umožňuje psát příkazy přímo v příkazovém řádku a implementace OOP dovoluje zapouzdřovat kód, vytvářet objekty a využívat dědičnost. Mimo jiné podporuje taktéž více programovacích paradigmat jako je procedurální nebo funkční programování a má garbage collector pro správu paměti. Tento jazyk podporuje dynamickou kontrolu typu, takže se sám stará o datové typy jednotlivých proměnných a není je tedy potřeba implicitně deklarovat. Je odvozen z mnoha jazyků jako ABC, Modula-3, C, C++, Algol-68, SmallTalk a Unixový shell spolu s dalšími skriptovacími jazyky.

Syntaxe jazyka Python je čistá a má jednoduchou strukturu, tudíž je to jazyk jednoduchý na naučení a používání. Toto jsou také důvody, proč je doporučován jako první jazyk pro začátečníky ve světě programování. Python lze použít téměř na každém operačním systému například Windows, macOS, Linux a mnoho dalších unixových distribucích.

2.1 Historie

Vytvořil jej Guido van Rossum v Nizozemsku v Národním výzkumném ústavu pro matematiku a informatiku s první oficiální verzí z 20. února 1991. Python je pod licencí GNU General Public License, je to tedy open-source projekt, na kterém se může podílet kdokoliv. Na vývoji a vydávání programovacího jazyka se většinou podílí celá řada profesionálů z jedné firmy, takže je těžké určit, kdo se na něm vlastně nejvíce podílel. Python nebyl vyvíjen celý Guidem van Rossum, ale na vývoji se podíleli tisíce programátorů a testerů, kteří byli z jazyka nadšeni. Pojmenování jazyka vychází z názvu starého televizního pořadu "Monty Python's Flying Circus", nikoliv z pojmenování hada, avšak ve svém logu si Python nese právě obrázek hada. Sám Guido van Rossum vybízí komunitu ke zmínkám Monty Python v dokumentaci, jako takový komunitní vtípek pro zasvěcené. [34], [35]

2.2 Django

Django je velice populární framework Pythonu pro tvorbu webových aplikací. Tento webový framework byl vydán v roce 2005 jako open-source projekt zdarma k použití. Dodnes je tento framework vyvíjen a vylepšován, má velice dobře zpracovanou dokumentaci a velkou podporu komunity. [36], [37]

Django se pyšní filozofií „batteries-included“, což znamená, že framework by měl obsahovat všechny běžné funkcionality potřebné pro tvorbu webové aplikace na rozdíl od nutnosti přidávání samostatných knihoven. Může jít například o vestavěné nástroje pro autentizaci, směrování URL, engine pro tvorbu šablon, objektově-relační mapování (ORM), nástroje pro migraci schémat databáze apod. Tyto všechny funkčnosti má framework Django zabudované, na rozdíl od dalšího populárního webového frameworku pro Python pod názvem Flask, který neobsahuje například nástroje pro implementaci autentizace uživatelů a je pro to potřeba přidávat externí knihovny. [37]

Tento framework oficiálně podporuje připojení databází PostgreSQL, MariaDB, MySQL, Oracle a SQLite. Nejpoužívanější databází ve spojení s Djangem je právě PostgreSQL. Django také poskytuje admin uživatelské rozhraní pro jednoduchou správu dat v databázi přímo z rozhraní aplikace. [38]

2.2.1 Architektura MVT

Framework Django využívá pro vývoj webových aplikací typ architektury Model-View-Template, neboli MVT. Jedná se o návrhový vzor složený ze tří hlavních komponent, a to z modelu, view a template. Následující výčet je zpracován dle zdroje [39]:

- **Model** - Model je stejně jako v MVC rozhraním pro manipulaci s databází. Tato část aplikace zodpovídá za data a datové struktury potřebné pro chod celé aplikace. Django implementuje objektově-relační mapování tak, že každá tabulka v databázi je třídou v Pythonu a jednotlivé sloupce databázové tabulky jsou atributy dané třídy. Modely jsou v Django projektu uloženy v souboru s názvem *models.py*.
- **View** - View se v Djangu chová jako prostředník mezi modelem a template a je umístěno v souboru pod názven *views.py*. V této části aplikace je řešena veškerá aplikační logika. View dle uživatelova požadavku získá potřebná data z databáze

pomocí z modulu model, následně data zpracuje a zobrazí template spolu s navrácenými zpracovanými daty.

- **Template** - Modul template je zodpovědný za část webové aplikace, která je uživateli zobrazována. Tedy obsahuje HTML kód zobrazující statické části aplikace a dále Django Template Language, který se stará o zobrazení dynamického obsahu.

Běžně frameworky pro tvorbu webových aplikací implementují architekturu MVC, neboli Model-View-Controller. Jedná se o velice populární architekturu, která dokáže skvěle izolovat aplikační logiku od uživatelského rozhraní a umožňuje přehledně od sebe oddělovat různé části aplikace. Controller v tomto typu architektury je část aplikace, která se stará o interakci mezi modelem a view. MVT architektura frameworku Django je velice podobná architektuře MVC, avšak Django MVT na rozdíl od MVC řeší část controlleru automaticky a vývojář se tedy nemusí starat o způsob, jak probíhá komunikace mezi modelem a view. [40]

2.3 Plotly

Plotly je oblíbená open-source vizualizační knihovna podporující až 40 různých druhů grafů z různých oblastí. Například pro statistiku, finančnictví, mapy, vědecké grafy a vizualizace, 3D grafy, atd. Jedná se konkrétně o grafy lineární, krabicové, bodové, plošné, tepelné mapy, histogramy, bublinové grafy a mnoho dalších. [41], [42]

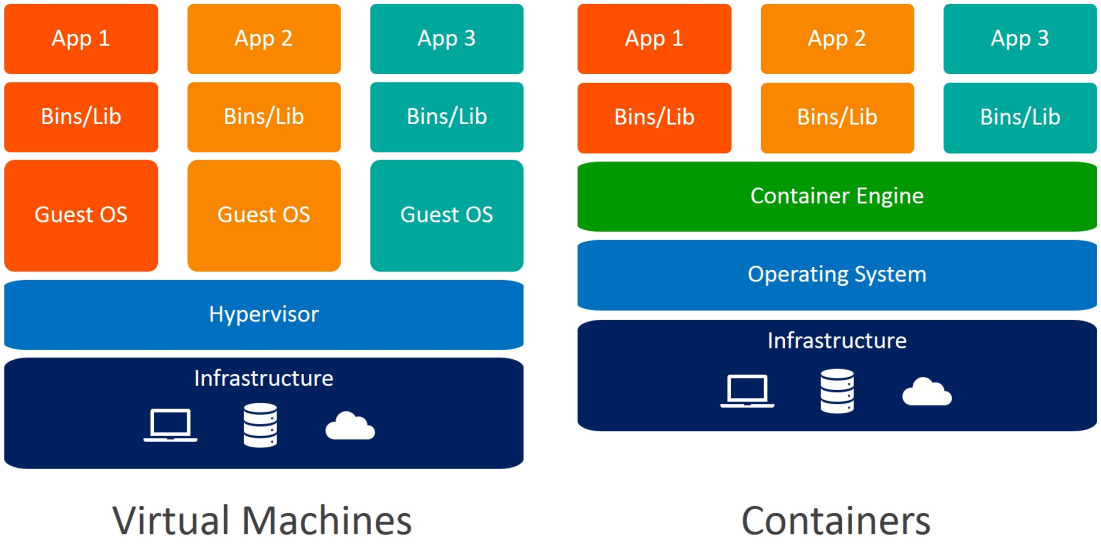
Knihovna je postavena na původní javascriptové Plotly knihovně a díky tomu je hojně využívána hlavně při tvorbě interaktivních webových vizualizací a dashboardů. Tyto vizualizace mohou být zobrazeny v Jupyter Noteboocích, nebo uloženy jako samostatný html soubor nebo součástí webové aplikace postavené na webovém frameworku pro Python. [41]

Oblíbenost této knihovny spočívá v mnoha možnostech a funkcích, které poskytuje. Všechny grafy vytvořené v Plotly jsou interaktivní a dovolují uživateli zobrazit detailní informace o daném objektu po najetí myší nad konkrétní bod grafu. Dále umožňuje libovolně přibližovat či posouvat viditelnou část zobrazeného grafu. Výsledné grafy zobrazené online může uživatel snadno stáhnout do svého zařízení jako vektorovou grafiku nebo lze grafy snadno serializovat do formátu JSON pro následné zpracování v jiném software. Možná nejdůležitější výhodou je snadné vytvoření a manipulace s Plotly grafy na webu. [43]

3 DOCKER

Docker je technologie umožňující vývojářům vytvářet software a spouštět služby v izolovaných kontejnerech. Byl vyvinut v roce 2013 společností dotCloud a byl napsán v jazyce Go. Služby či aplikace v Docker kontejnerech dokáží běžet nezávisle na prostředí kde jsou spouštěny, jelikož tyto kontejnery obsahují všechny potřebné knihovny a veškeré nastavení nutné ke správnému běhu dané aplikace či služby. Proto je Docker tolik oblíbený, jelikož například aplikace poběží s Dockerem úplně stejně jak v cloudu, tak i na lokálním počítači vývojáře, a to i v několika instancích souběžně. [44], [45]

Hlavním principem funkce Dockeru je spouštění izolovaných kontejnerů, neboli instancí obrazů (image), s využitím služeb jádra hostujícího operačního systému. Funkce tohoto systému je založena na kontejnerové virtualizaci, která šetří zdroje systému využíváním jádra hostujícího operačního systému na rozdíl od nutnosti virtualizovat pro každou kontejnerizovanou službu vlastní operační systém. [44]



Obrázek 4: Kontejnerizace vs virtualizace. [46]

3.1 Kontejnerová virtualizace

Tato kapitola byla zpracována podle zdroje [47].

Kontejnerovou virtualizaci, která je využívána například Dockerem lze nazvat také jako virtualizaci na úrovni jádra operačního systému. Jak již bylo zmíněno výše, tak všechny

služby spouštěné v kontejnerech sdílí společně jádro hostujícího operačního systému spolu s jeho vlastními službami. Služby jádra jsou rozšířeny o schopnost odlišit jednotlivé skupiny souvisejících procesů a jejich náležící souborové systémy. Tyto skupiny souvisejících procesů se nazývají kontexty. Jedinou podmínkou pro funkčnost tohoto typu virtualizace je stejná hardwarová architektura virtualizovaného systému s hostujícím operačním systémem.

Tento způsob virtualizace je považován za nejefektivnější, jelikož nedochází k žádnému zbytečnému vytěžování dostupných systémových zdrojů jako u klasické virtualizace. Tam je pro běh aplikace zapotřebí vytvářet plnohodnotný virtualizovaný operační systém. Jediné operace čerpající systémové zdroje pro řízení chodu kontejnerové virtualizace jsou operace zajišťující izolaci procesů, jejich souborových systémů a síťový provoz. Proto je výkon takto virtualizovaných služeb podobný, jako kdyby běžely nativně. Nevýhodou kontejnerové virtualizace je bezpečnost, jelikož pokud selže jádro, tak spolu s ním selžou i všechny na něm běžící služby.

3.2 Komponenty Dockeru

Docker funguje na architektuře klient-server. Klient komunikuje s daemonem, který se stará o vytváření kontejnerů a následně jejich běh a distribuci. Řídí funkci objektů Dockeru jako jsou obrazy, kontejnery, sítě a volumes. Dalším klientem v Dockeru je Docker Compose, který se stará o běh aplikací složených z několika spolupracujících kontejnerů.

- **Image** - Image, neboli obraz, je předpis s instrukcemi pro vytvoření Docker kontejneru. Velice často jsou obrazy založené na jiném již existujícím obrazu, který je dostupný ve veřejném Docker registru pod názvem Docker Hub. Uživatel může vytvořit i vlastní obraz složený z již existujícího obrazu z registru a konfigurace dalších potřebných komponent pro danou aplikaci. Vlastní obraz je sestaven pomocí konfigurace uvedené v souboru Dockerfile. Instrukce v Dockerfile definují jednotlivé vrstvy, které budou vytvořeny v obrazu. [44]
- **Container** - Kontejner je konkrétní běžící instance daného obrazu. Kontejnerů lze vytvořit z jednoho obrazu libovolné množství dle dostupných zdrojů nezávisle na sobě. Lze je připojit k více sítím, dále je možné jim připojit úložiště pro zaznamenávání změn stavu uvnitř kontejneru. Docker poskytuje možnost vytvořit nový kontejner na základě aktuálního stavu jiného kontejneru. Kontejnery jsou od ostat-

ních kontejnerů a také od hostujícího zařízení poměrně dobře izolovány a uživatel může míru jejich izolace měnit dle potřeby. Lze je ovládat příkazy v konzoli nebo pomocí Docker API. [44], [48]

Tabulka 1: Ukázka příkazů pro manipulaci s Docker kontejnery. [49]

Příkaz	Popis
docker container create	vytvoří nový kontejner dle specifikovaného obrazu
docker container start	spuštění jednoho nebo více zastavených kontejnerů
docker container restart	restartuje jeden nebo více kontejnerů
docker container stop	zastavení jednoho nebo více běžících kontejnerů
docker container kill	vynucené ukončení jednoho nebo více kontejnerů
docker container inspect	zobrazení informací o vybraných kontejnerech
docker container exec	spuštění příkazu uvnitř běžícího kontejneru

- **Network** - Velmi významnou komponentou Dockeru jsou široké možnosti síťového nastavení. Docker kontejnery lze totiž propojovat mezi sebou nebo také s aplikacemi běžícími mimo Docker. Subsystém sítě Dockeru funfuje na principu připojitelnosti pomocí ovladačů. Základní síťové funkce jsou ve výchozím nastavení poskytovány několika hlavními ovladači jako jsou bridge, host, overlay atd. Výchozím ovladačem je bridge, který dovoluje kontejnerům komunikovat prostřednictvím hostitelského zařízení. Nastavení sítě v Dockeru lze upravit pomocí příkazů *docker network příkaz*. [50]
- **Volume** - Docker volumes jsou nástrojem pro uchovávání dat vytvořených uvnitř kontejnerů. Jedná se o sdílený adresář mezi kontejnerem a hostujícím operačním systémem. Tento sdílený adresář umožňuje snadno vkládat potřebné soubory z lokálního zařízení do kontejneru. Pokud je volume odstaněno, tak dojde ke smazání všech dat kontejneru. [51]

3.3 Dockerfile

Tato kapitola byla zpracována podle zdrojů[52] a[53].

Dockerfile je textový soubor obsahující veškeré příkazy, které uživatel může použít k vytvoření obrazu. Pomocí přednastavených příkazů v tomto souboru lze tedy automaticky vytvářet obrazy příkazem *docker build*, který musí být spuštěn ve stejném adresáři, kde se

nachází daný Dockerfile. Docker daemon poté provádí sestavení obrazu pomocí spouštění příkazů v pořadí, v jakém jsou uvedeny v souboru Dockerfile.

Struktura souboru Dockerfile je tvořena jednotlivými instrukcemu, kde každá instrukce je uvedena na novém řádku. Instrukce jsou značeny klíčovými slovy a za nimi jsou uvedeny potřebné parametry oddělené mezerou. Instrukce RUN, COPY a ADD vytváření nové vrstvy obrazu. Ostatní instrukce vytvářejí pouze dočasné obrazy, tudíž nezvětšují velikost buildu. Dobrou praktikou je instalovat do obrazu pouze nutné balíčky, aby byla velikost buildu co nejmenší.

Tabulka 2: Klíčová slova souboru Dockerfile. [54]

Klíčové slovo	Popis
FROM	definuje obraz, podle kterého bude nový obraz sestaven
LABEL	přidává označení novému obrazu
RUN	umožňuje spouštět linuxové příkazy, například instalovat balíčky
CMD	instrukce ke spouštění příkazů po startu kontejneru
ENV	definuje proměnné prostředí v běžícím kontejneru
COPY	zkopíruje vybraný obsah do adresáře v kontejneru
WORKDIR	nastavuje pracovní adresář pro instrukce RUN, COPY, ADD a CMD
EXPOSE	instrukce definující port, na kterém bude kontejner naslouchat

3.4 Docker Compose

Docker compose je nástroj pro spouštění aplikací, které jsou složené z několika kontejnerů. Konfigurace nástroje compose je uvedena v souboru typu YAML pod názvem *docker-compose.yaml*. Konfigurační soubor obsahuje nastavení jednotlivých služeb, které by měly v kontejnerech běžet. Vícekontejnerovou aplikaci lze poté spustit jediným příkazem *docker-compose up*. Pokud je k příkazu přidán přepínač *-d*, tak je kontejner spuštěn na pozadí a nebudou vypisovány logy aplikace do konzole. Zastavení běžících kontejnerů postavených na compose lze příkazem *docker-compose stop*. Pro zastavení kontejnerů a jejich následné odstranění je možné zadat do konzole příkaz *docker-compose down*. Pokud by uživatel chtěl s odstraněním kontejneru odstranit i jeho data uvnitř volume adresáře, tak je možné přidat parametr –volumes za příkaz odstranění kontejneru, tedy *docker-compose down --volumes.* [55], [56]

4 NÁVRH SYSTÉMU

Tato kapitola pojednává o požadavcích, které jsou kladeny na praktickou část bakalářské práce, aby splňovala zadání. Je zde popsán návrh řešení IoT systému spolu s aplikačním a hardwarovým diagramem, který znázorňuje grafickou podobu řešení.

4.1 Požadované vlastnosti

Cílem této bakalářské práce bylo vytvořit škálovatelný dashboard pro přehledné zobrazení výsledných měření senzorů. Požadavky na výslednou aplikaci umožňující ukládání a zobrazování podrobnějších informací z jednotlivých senzorů byly následující:

- zobrazení lokace, název typu senzoru a jeho veličiny,
- zobrazení minimální, maximální a průměrné naměřené hodnoty z každého jednotlivého senzoru,
- grafické zobrazení naměřených hodnot,
- odstranění a přidání senzoru,
- uživateli umožní vytvářet jednotlivé instance senzorů včetně podrobné parametrizace senzoru.

4.2 Návrh řešení

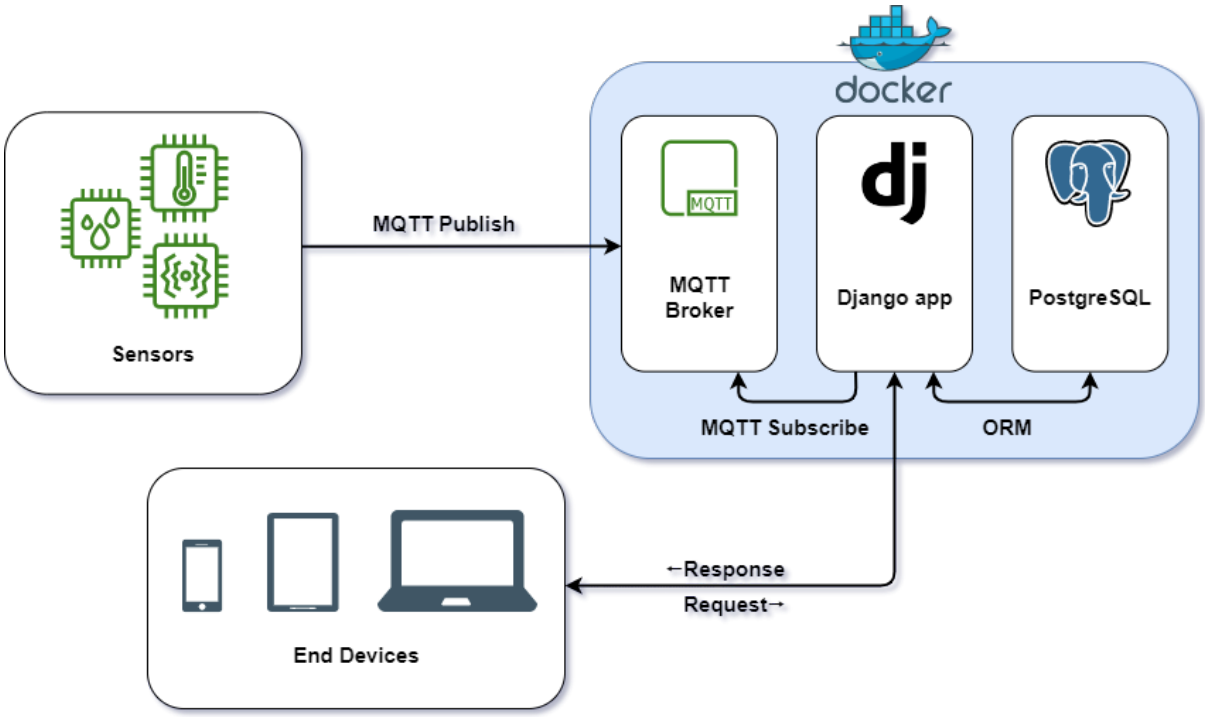
Výsledný IoT systém, ve kterém je hlavní komponentou aplikace poskytující přehled o datech ve formě škálovatelného dashboardu je složen z několika částí. Nejprve však bylo zapotřebí vybrat komunikační protokol, který zajistí odesílání a přijímání dat mezi senzory a aplikací. Pro tento účel byl vybrán protokol MQTT díky jeho nenáročnosti na provoz a dobré škálovatelnosti.

Na vrstvě snímání tohoto IoT systému se nachází senzory měřící různé veličiny, například teplotu, vlhkost, rychlost či svítivost. Tyto senzory jsou připojeny k modulu ESP32, který se stará o čtení dat ze senzorů v pravidelných intervalech a odesílá načtená data pomocí jednotlivých témat protokolem MQTT na MQTT Broker.

Pro zjednodušení vývoje i nasazení výsledného řešení byla vytvořena vícekontejnerová aplikace v Dockeru obsahující všechny potřebné služby a knihovny ke správné funkci výsledné aplikace. První kontejner obsahuje MQTT Broker, který se chová jako server řídící

provoz doručování zpráv v IoT systému. Ve druhém kontejneru je nainstalována databáze PostgreSQL zajišťující perzistenci všech dat, která byla načtena ze senzorů a následně přijata aplikací. Poslední kontejner obsahuje hlavní komponentu tohoto systému, kterou je webová aplikace vytvořená pomocí Python webového frameworku Django. K implementaci škálovatelného dashboardu byla zvolena právě webová aplikace, jelikož ji lze snadno zobrazit v jakémkoliv zařízení, které disponuje webovým prohlížečem. Implementace v Dockeru umožňuje snadné nasazení aplikace bez ohledu na to, jaký systém je na daném zařízení nainstalován. V tomto konkrétním řešení bylo pro hostování Docker kontejneru vybráno zařízení Raspberry Pi, které je velice populární v IoT projektech díky kombinaci malé energetické náročnosti, dobrého výkonu a malé velikosti zařízení.

4.2.1 Schéma aplikace

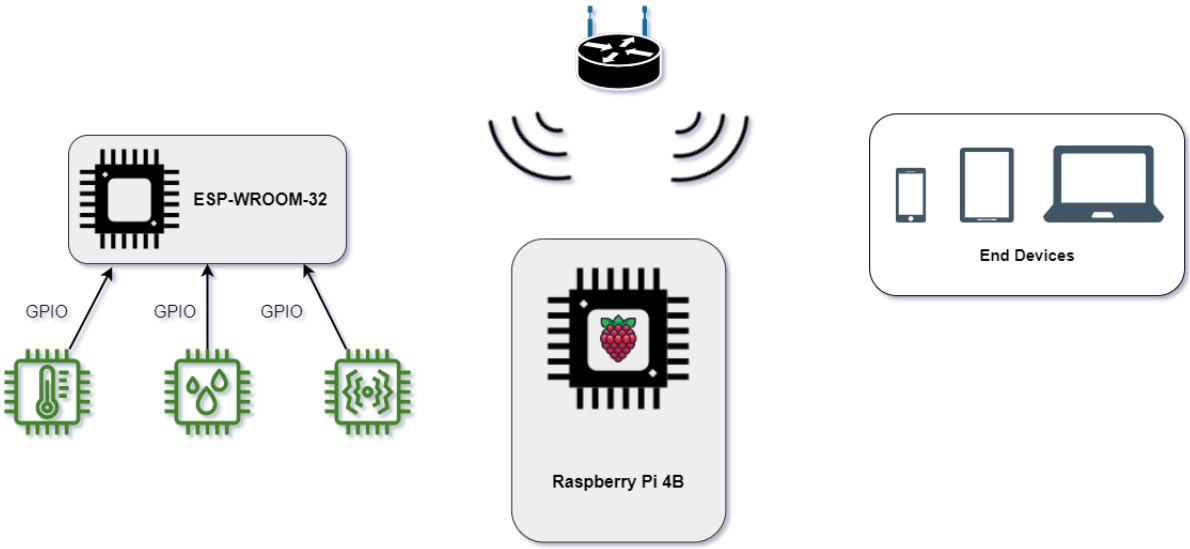


Obrázek 5: Aplikační schéma. Zdroj vlastní.

Na obrázku 5 lze vidět aplikační schéma znázorňující zejména rozvržení softwarových komponent systému a nastínění průběhu komunikace mezi jednotlivými komponentami. Klient komunikuje s Django webovou aplikací na základě request-response modelu. Django aplikace je zároveň v pozici klienta v rámci komunikace MQTT protokolem, který funguje

na základě komunikačního modelu publish-subscribe. Tedy Django jako klient odebírá zprávy na specifikovaném tématu od MQTT Brokeru. Ten přijímá zprávy od senzorů a řídí jejich přesměrování Django aplikaci. Posledním článkem aplikačního schématu je PostgreSQL databáze pro zajištění perzistence načtených dat senzory. Komunikace mezi Djangem a databází je řízena pomocí ORM, neboli objektově-relačního mapování.

4.2.2 Schéma hardware



Obrázek 6: Diagram hardwaru. Zdroj vlastní.

Obrázek 6 znázorňuje použitá fyzická zařízení, která spolu interagují v rámci navrženého IoT systému. Senzory jsou připojeny k GPIO pinům na zařízení ESP-WROOM-32. To pomocí Wi-Fi signálu komunikuje prostřednictvím protokolu MQTT v lokální síti s aplikací, která je dostupná na zařízení Raspberry Pi. Koncová uživatelská zařízení jako jsou mobilní telefony, tablety nebo počítače mohou v lokální síti komunikovat s webovým serverem na Raspberry Pi a interagovat s obsahem webové aplikace.

5 IMPLEMENTACE SYSTÉMU

Kapitola implementace systému pojednává o způsobu implementace požadovaných vlastností tak, aby byly splněny cíle této práce. Jsou zde představeny všechny hardwarové komponenty, které jsou potřebné k dosažení výsledného řešení. Dále jsou popsány použité softwarové technologie spolu s postupem jejich implementace pro vytvoření škálovatelného dashboardu.

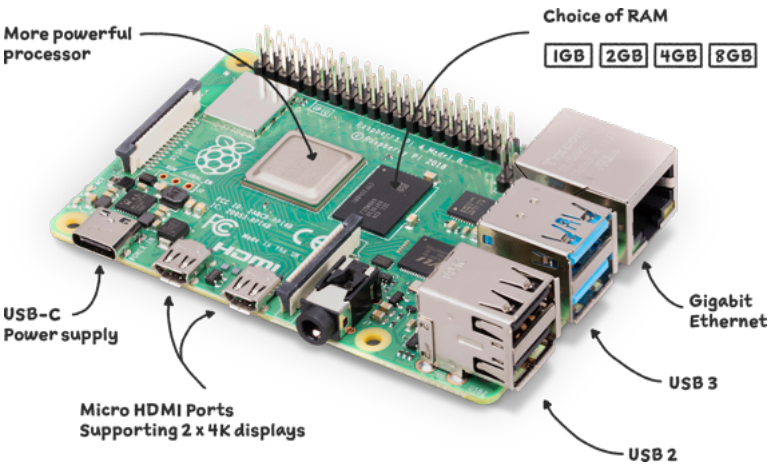
5.1 Hardwarové komponenty

V této kapitole budou vyjmenovány a popsány hardwarové komponenty, které byly použity k vytvoření výsledného řešení. Mimo aktivní prvky jako jsou Raspberry Pi, ESP32 a různé senzory byly při realizaci řešení použity i komponenty jako nepájivé pole, vodiče do nepájivého pole a pullup rezistory s hodnotou odporu 4,7 kΩ.

5.1.1 Raspberry Pi

Raspberry Pi je levný a malý jednodeskový počítač s velice dobrým výkonem vzhledem k jeho velikosti. Je vyvíjen společností Raspberry Pi Foundation, která začala tyto jednodeskové počítače vyrábět především k podpoře výuky programování a počítačové vědy u dospělých a dětí. Je hojně využíván k výuce jazyka Python a k různým projektům včetně implementace IoT systémů. Lze ho také využívat jako klasický stolní počítač. [57]

Raspberry Pi je dostupný v mnoha verzích, které se od sebe liší převážně ve výkonnosti procesoru, velikosti operační paměti, množství GPIO pinů a množství různých portů dostupných na desce počítače. Mikroprocesor obsažený na desce je postaven na architektuře ARM. Na tento jednodeskový počítač lze nainstalovat různé distribuce Linuxu, RISC OS vytvořený pro ARM procesory nebo verzi Windows přímo pro Raspberry Pi pod názvem Windows IoT Core. Nejčastějším OS instalovaným na Raspberry Pi je však Raspbian, který je založený na Debianu a byl vytvořen přímo pro tento jednodeskový počítač. [58]



Obrázek 7: Raspberry Pi 4 Model B. [59]

Zařízení Raspberry Pi bylo vybráno jako prvek pro hostování Docker kontejnerů s hlavními softwarovými komponentami této bakalářské práce. Konkrétně byla použita verze Raspberry Pi 4 Model B 8GB RAM. Na zařízení byl pro potřeby této práce nainstalován operační systém Raspbian, verze Raspbian GNU/Linux 11 (bullseye), pomocí oficiálního nástroje Raspberry Pi Imager. Výběr tohoto zařízení byl učiněn na základě poměru dobrého výkonu zařízení s jeho energetickou nenáročností, malou velikostí a dobrou cenou. Z těchto důvodů je Raspberry Pi jedním z nejvyužívanějších zařízení pro IoT projekty.

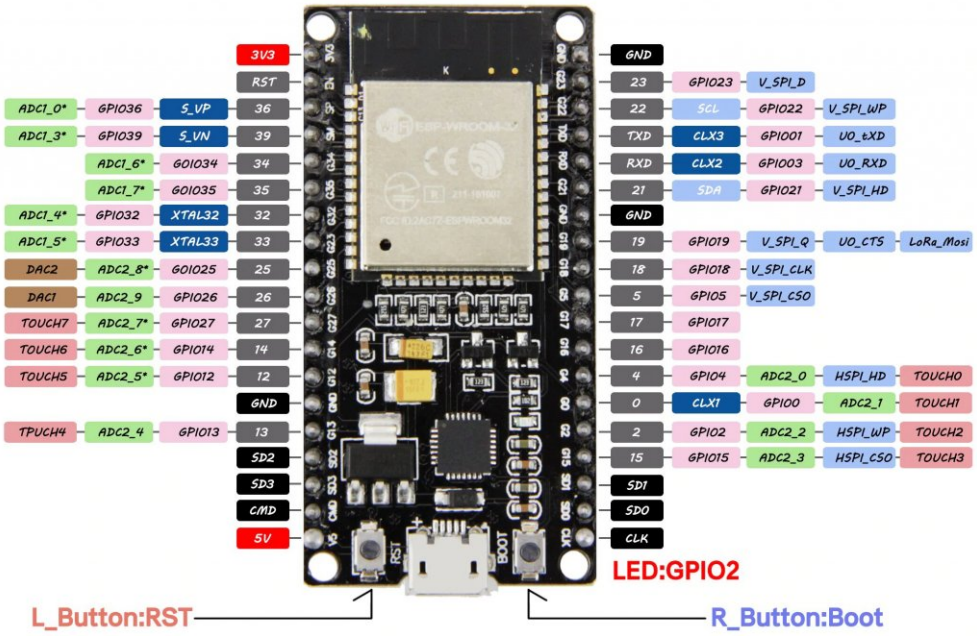
5.1.2 ESP32

Tato kapitola byla zpracována podle zdroje [60]. ESP32 je výkonný, cenově dostupný a energeticky nenáročný mikrokontroler, který byl vyvinut společností Espressif Systems. Pro potřeby realizace praktické části této práce byl vybrán model ESP-WROOM-32, jelikož má integrované moduly pro Wi-Fi, Bluetooth i Bluetooth LE. Proto je také ideálním mikrokontrolerem pro práci se senzory s nízkým výkonem v IoT systémech a průmyslových prostředích.

Tento model disponuje dvoujádrovým ESP32-D0WDQ6 čipem s frekvencí 80 - 240 MHz, který má integrovány senzory pro měření teploty a Hallova jevu. Jak je možné vidět na obrázku 8, vývojová deska s čipem ESP-WROOM-32 obsahuje celkem 38 pinů k připojení a ovládání různých periferií. Ty byly využity k připojení senzorů pro čtení naměřených hodnot. [61]

Vývojovou desku lze programovat několika různými způsoby jako například pomocí Arduino IDE, Espressif IDF, Micropythonu apod. Pro nahrání skriptu při řešení této

práce bylo využito Arduino IDE a připojení desky přes sériový port. Podrobnější popis programování ESP32 pro potřeby praktické části této práce bude popsán v kapitole Programování ESP32.



Obrázek 8: ESP-WROOM-32 pinout. [62]

5.1.3 Senzory

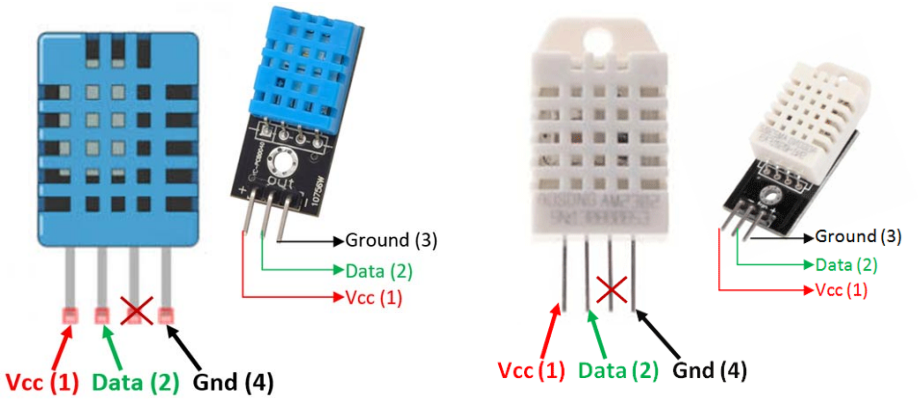
V této kapitole jsou vyjmenovány a popsány senzory, které byly použity při testování funkce komunikace ezi prvky IoT systému pomocí MQTT protokolu a zobrazování dat ve škálovatelném dashboardu.

- **DHT 11** - Senzor DHT 11 je nízkonákladovou verzí senzoru pro měření teploty a vlhkosti vzduchu. DHT senzory jsou složeny z termistoru a kapacitního čidla vlhkosti. Obsahuje jednoduchý digitálně analogový převodník, který konvertuje načtená analogová data na digitální signál obsahující záznamy o teplotě a vlhkosti.

  Obsahuje 4 piny, jejichž popis je možné vyčíst z obrázku 9 vlevo. Vstupní napětí se může pohybovat od 3V do 5V. Je schopen měřit v časových intervalech jednou za sekundu. Vlhkost dokáže měřit v rozmezích hodnot 20-90% s odchylkou 5%. Měřitelné rozmezí teplot se pohybuje mezi 0-50°C s odchylkou +-2°C. [63]

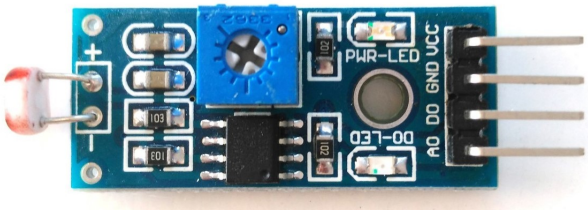
- **DHT 22** - Senzory DHT 22 jsou velice podobné senzorům DHT 11. Mají stejně uspořádané piny jak lze vidět na obrázku 9 vpravo. Pokud je senzor zabudován v modulu, tak obsahuje zabudovaný pullup rezistor a filtrační kondenzátor, jinak je nutné tyto součástky použít externě. Také měří teplotu a vlhkost vzduchu, ale jsou o něco dražší a liší se v ostatních charakteristikách. [66]

  Vstupní napětí se pohybuje mezi 3V a 5V. Jejich měření jsou mnohem přesnější a dokáží měřit větší rozsah hodnot než sezory typu DHT 11. Konkrétně měření vlhkosti dokáží provést v rozmezích hodnot 0-100% s odchylkou maximálně mezi 2-5%. U teploty lze naměřit hodnoty v rozmezí -40-80°C s odchylkou do +- 0.5°C. Senzor je schopen provádět měření v časových intervalech jednou za dvě sekundy. [64]



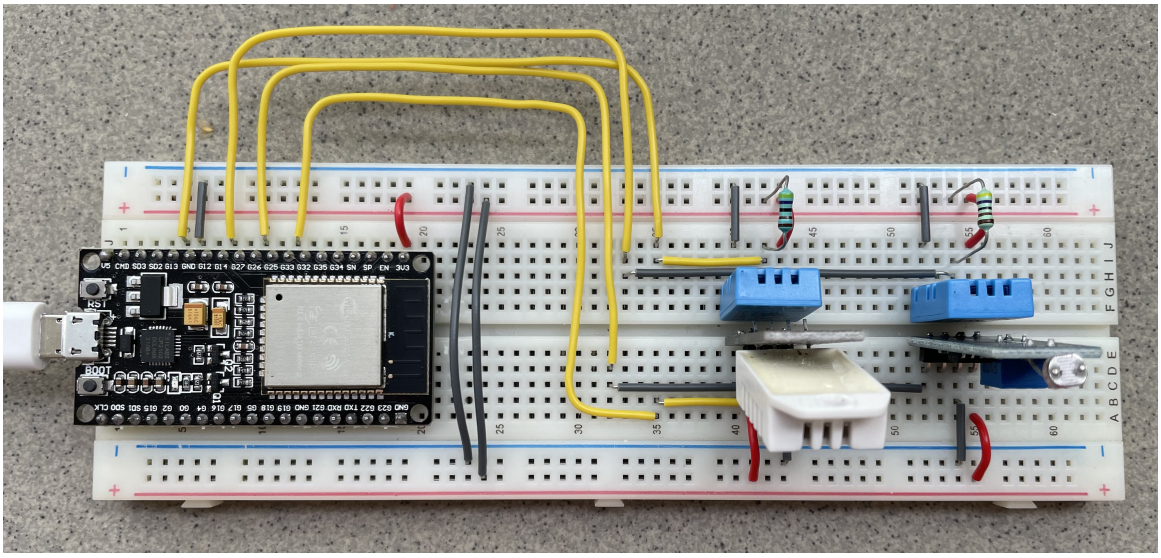
Obrázek 9: DHT 11 a DHT 22 [65], [66]

- **LDR MH-Sensor-Series** - Light Dependent Resistor, česky odporový světelný senzor nebo také fotorezistor je využíván pro měření intenzity světla. Tento senzor dokáže fungovat v analogovém i digitálním režimu. Obsahuje 4 piny, jak je vidět na obrázku 10 čteno zleva: analogový výstup, digitální výstup, GND, VCC. Čím jasnější je světlo, tím je nižší elektrický odpor a zároveň jsou nižší hodnoty na analogovém výstupu. Citlivost digitálního výstupu lze nastavit potenciometrem dostupným na desce modulu. Vstupní napětí je mezi 3,3V a 5V a výstupní hodnoty se pohybují mezi 0V až 5V. Modul desky obsahuje LED diody indikující vstupní napětí a detekci světla. [67]



Obrázek 10: LDR MH-Sensor-Series modul. [68]

5.1.4 Výsledné zapojení



Obrázek 11: Výsledné zapojení. Zdroj vlastní.

Jak lze vidět na obrázku 11, mikrokontrolér ESP-WROOM-32 byl zapojen do nepájivého pole. Celý systém je napájen pomocí mikro USB. K pinům G13 a G14 na mikrokontroléru jsou připojeny senzory DHT11 přes pull-up rezistory hodnoty 4,7 kΩ. Senzor DHT 22 je zasazen do modulu, který již obsahuje pull-up rezistor, tedy žádný rezistor nemusel být přidáván externě. Data z modulu senzoru DHT 22 jsou čtena na pinu G26. Posledním testovaným senzorem byl fotorezistor LDR MH-Sensor-Series modul, který je připojený k pinu G33 na ESP32.

5.2 Výběr programovacího jazyka

Jelikož je dominantním prvkem v navrhovaném IoT systému zařízení Raspberry Pi, tak byl vybrán programovací jazyk Python, který je primárně používaným jazykem na této

platformě. Pro vývoj webové aplikace byl zvolen webový framework jazyka Python pod názvem Django. A pro implementaci grafů a MQTT komunikace byly použity Python knihovny Plotly a Paho MQTT Client.

Kód ovládající mikrokontrolér ESP-WROOM-32 na čtení dat ze senzorů a odesílání MQTT zpráv byl naprogramován v jazyce C. Další variantou výběru programovacího jazyka byl MicroPython, jehož použití bylo původně zamýšleno, protože ostatní software byl implementován také v Pythonu. Od jeho použití bylo následně upuštěno po nastudování způsobu využívání MicroPythonu s ESP32 na základě předchozích zkušeností s Arduino IDE a nativní podpoře jazyka C jak v Arduinu, tak na platformě ESP. [69]

5.3 Programování ESP32

Mikrokontroller ESP-WROOM-32 byl programován v Arduino IDE pomocí připojení přes sériový port k Raspberry Pi. Při nastavování Arduino IDE a vytváření programu v jazyce C bylo postupováno podle návodu na webové stránce `https://randomnerdtutorials.com/esp32-mqtt-publish-dht11-dht22-arduino/`. Kód dostupný na této stránce byl při implementaci rozšířen o podporu více druhů senzorů a příslušná MQTT témata.

Nejprve bylo nutné v Arduino IDE nainstalovat podporu typu desky DOIT ESP32 DEVKIT V1. Poté byly nainstalovány knihovny pro senzory DHT a externí knihovny pro asynchronní TCP komunikaci (`https://github.com/me-no-dev/AsyncTCP`) a asynchronního MQTT klienta (`https://github.com/marvinroger/async-mqtt-client`).

Import použitých knihoven lze vidět na následující ukázce kódu:

```
#include "DHT.h"
#include <WiFi.h>
#include <AsyncMqttClient.h>
extern "C" {
    #include "freertos/FreeRTOS.h"
    #include "freertos/timers.h"
}
```

5.3.1 Průběh programu

V setup části programu, která je provedena pouze jednou na začátku po spuštění programu je započato čtení dat z DHT senzorů a zajištěno připojení k Wi-Fi a MQTT brokerovi. V metodě loop je zavolána metoda na přečtení dat ze senzorů a následně jsou načtená data odeslána na příslušné téma pomocí MQTT protokolu v časových intervalech jednou za minutu. Pokud by se data z některého ze senzorů nepodařilo přečíst, tak je aktuální průběh cyklu ukončen ještě před pokusy o odeslání dat MQTT protokolem. V takovém případě se přejde na následující iteraci cyklu a čeká se na uplynutí minuty od posledního času přečtení dat ze senzorů. Kód tohoto programu v jazyce C je přiloze v příloze Příloha C.

5.3.2 Nastavení MQTT

V deklarační části kódu jsou definovány konstanty obsahující potřebná data ke správnému nastavení komunikace pomocí MQTT protokolu. Pro správnou funkci IoT systému je nutné nastavit korektní IP adresu zařízení, které hostuje Docker kontejner s MQTT brokerem. Přehled definice IP adresy MQTT brokeru, portu a používaných témat je možné vidět na následující ukázce kódu.

```
// Raspberry Pi MQTT Broker
#define MQTT_BROKER IPAddress(192, 168, 0, 101)
#define MQTT_PORT 1886

// MQTT Topics
#define MQTT_PUB_DHT11_TEMP "esp32/dht11/temperature/1"
#define MQTT_PUB_DHT11_HUM  "esp32/dht11/humidity/1"
#define MQTT_PUB_DHT11_TEMP_2 "esp32/dht11/temperature/2"
#define MQTT_PUB_DHT11_HUM_2 "esp32/dht11/humidity/2"
#define MQTT_PUB_DHT22_TEMP "esp32/dht22/temperature"
#define MQTT_PUB_DHT22_HUM  "esp32/dht22/humidity"
#define MQTT_PUB_LDR_LIGHT "esp32/LDR/light"
```

V metodě setup je předáno nastavení IP adresy brokeru a portu instanci MQTT klienta. Dále jsou zde také nastaveny callbacky zajišťující převážně výpis informací do sériové

konzole o akcích jako jsou publish zprávy a připojení nebo odpojení klienta. Nastavuje se tu také hodnota časovače pro znovupřipojení k MQTT na 2 sekundy.

Ukázka kódu zobrazující implementaci metody publish pro odeslání MQTT zprávy:

```
// Publish an MQTT message on topic esp32/dht22/temperature
uint16_t packetIdPub6 = mqttClient.publish(
    MQTT_PUB_DHT22_TEMP, 1, false, String(dht22_temp).c_str());
Serial.printf("PacketID: %i, Topic: %s, Message: %.2f\n",
    packetIdPub6, MQTT_PUB_DHT22_TEMP, dht22_temp);
```

5.4 Docker

Jak bylo zmíněno již v kapitole pojednávající o návrhu aplikace, implementace v Docker kontejnerech byla vybrána pro zjednodušení vývoje i nasazení aplikace.

Pro spuštění Docker daemonu při startu systému je potřeba zadat příkaz:

```
sudo systemctl enable docker
```

Konfigurace Dockeru k vytvoření webové aplikace v Djangu, PostgeSLQ databáze a MQTT brokera je rozdělena do dvou konfiguračních souborů. Prvním je *Dockerfile*, ve kterém je obsažena konfigurace pro vytvoření nového obrazu, na kterém budou postaveny použité kontejnery. Druhým konfiguračním souborem je *docker-compose.yaml*, který obsahuje nastavení kontejnerů služeb potřebných pro funkci této vícekontejnerové aplikace.

5.4.1 Dockerfile

Soubor Dockerfile nastavuje, aby byl nový obraz vytvořen podle obrazu python verze 3 z veřejného repozitáře. Dále nastavuje pracovní adresář a kopíruje soubory z lokálního zařízení do pracovního adresáře uvnitř kontejneru. Poté pomocí příkazu RUN zajišťuje spuštění instalace knihoven Pythonu, které jsou uvedeny jako závislosti v souboru requirements.txt pro potřeby této webové aplikace. Nakonec spouští příkaz uvnitř kontejneru pro spuštění webové aplikace.

Ukázka konfigurace souboru Dockerfile (Zdroj vlastní):

```
FROM python:3
MAINTAINER SilviaCmilanska
COPY ./app /app
WORKDIR /app
RUN pip install --upgrade -r requirements.txt
CMD ["python","manage.py","runserver","0.0.0.0:8990"]
```

5.4.2 docker-compose.yaml

V tomto konfiguračním souboru jsou uvedena nastavení kontejnerů jednotlivých služeb vícekontejnerové aplikace. Jedná se konkrétně o kontejnery PostgreSQL databáze, Eclipse Mosquitto MQTT brokera a Django webové aplikace.

Soubor obsahuje 3 hlavní sekce: version, services a networks. V sekci version je nastavena verze compose, který bude sestavovat výsledný kontejner. Sekce services obsahuje konfiguraci jednotlivých služeb. Jejich konfigurace je popsána v následujícím seznamu:

- **PostgreSQL** - Služba databáze byla nazvána jako *db* a byla vytvořena podle obrazu posgres:13. Konfigurace obsahuje základní nastavení databáze, mapování portů a nastavení lokálního adresáře, kam budou ukládána data kontejneru. Příkaz networks nastavuje síť dockeru a statickou IP adresu uvnitř kontejneru. Dále je nastaveno spuštění kontejneru při restartu docker daemona a zajištěno spuštění až po službě mqtt brokera.

```
db:
  image: postgres:13
  environment:
    POSTGRES_USER: app
    POSTGRES_DB: app
    POSTGRES_PASSWORD: app
  ports:
    - 5432:5432
  volumes:
    - .data:/var/lib/postgresql/data
```

```
  networks:
    default:
      ipv4_address: 172.21.0.3
  restart: unless-stopped
  depends_on:
    - mqtt
```

- **Eclipse Mosquitto Broker** - Tato služba byla v konfiguraci nazvána jako *mqtt* a byla vytvořena podle obrazu eclipse-mosquitto z repozitáře. Také obsahuje nastavení volumes, restartu služby, statické IP adresy uvnitř kontejneru a mapování portů. Broker je dostupný na portu 1886 na hostujícím zařízení. Nemá nastaveny závislosti pro spuštění, jelikož je prvním spouštěným kontejnerem.

```
mqtt:
  image: eclipse-mosquitto
  volumes:
    - .mqtt_data:/mosquitto/:rw
  ports:
    - 1886:1886
    - 9006:9001
  networks:
    default:
      ipv4_address: 172.21.0.2
  restart: unless-stopped
```

- **Django** - Služba pro Django byla nazvána jako *app* a byla sestavena podle obrazu definovaného souborem Dockerfile. Konfugurace nastavuje stejně jako u předchozích služeb mapování portů, lokální úložiště, síť se statickou IP adresou a restart kontejneru. Spuštění služby je podmíněno předchozím spuštěním služeb databáze a MQTT.

```
app:
  build:
    context: .
    dockerfile: Dockerfile
```

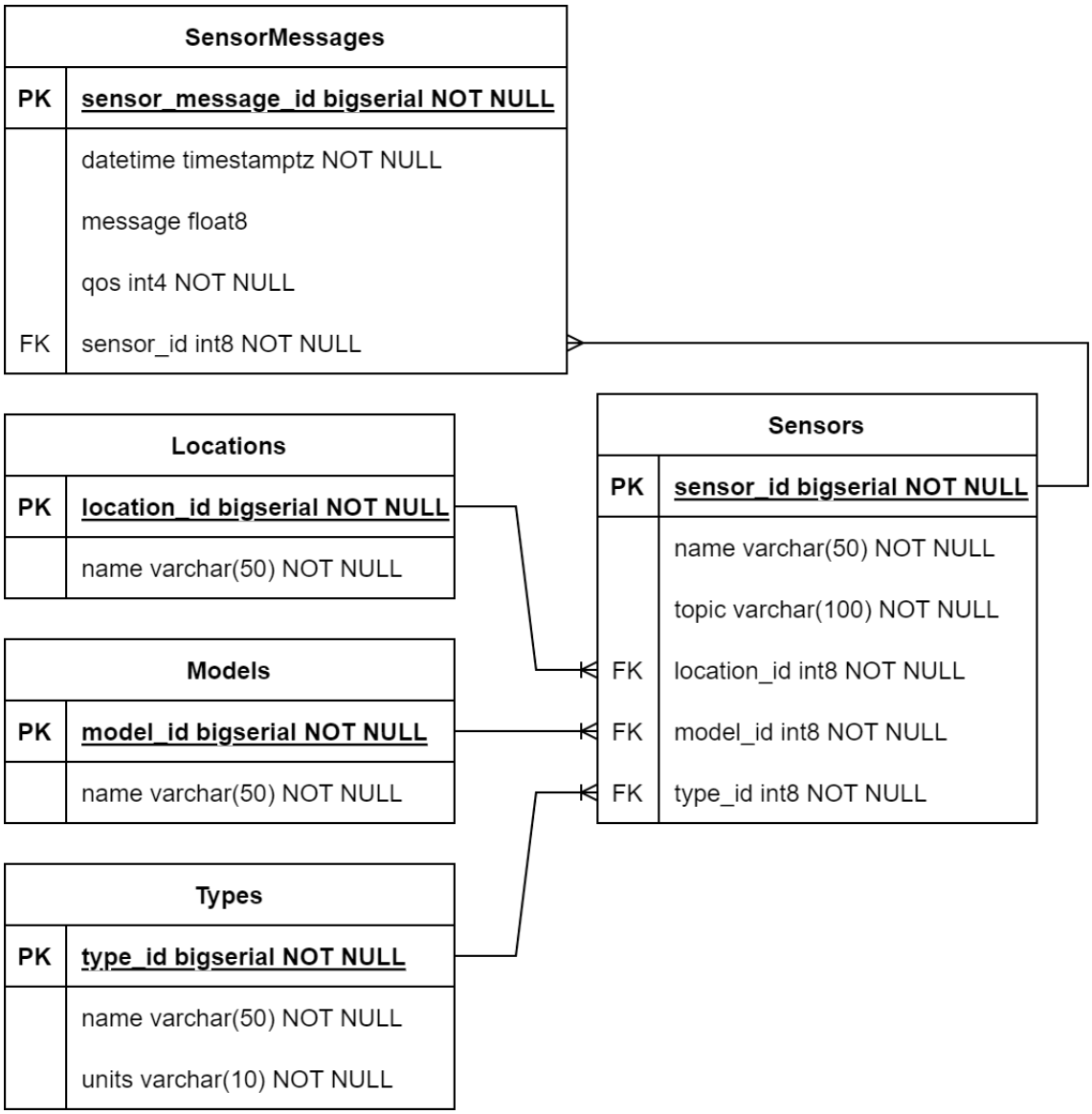
```
    ports:
      - "80:8990"
    volumes:
      - ./app:/app
    networks:
      default:
        ipv4_address: 172.21.0.4
    restart: unless-stopped
    depends_on:
      - db
      - mqtt
```

Poslední sekcí konfigurace je sekce networks, ve které je nastaveno vytvoření sítě s názvem *default* typu bridge ve specifikované podsíti a s nastavením IP adresy výchozí brány.

```
networks:
  default:
    driver: bridge
    ipam:
      config:
        - subnet: 172.21.0.0/16
          gateway: 172.21.0.1
```

5.5 Návrh databáze

Databáze PostrgreSLQ běžící v Docker kontejneru obsahuje 5 tabulek, které uchovávají data o senzorech pro potřeby webové aplikace. Databáze obsahuje dvě hlavní tabulky. První je nazvána *Sensors* a uchovává základní data o senzorech. Jsou na ni napojeny další 4 tabulky: *Locations*, *Models* a *Types* jsou číselníky a tabulka *SensorMessages* uchovává přijaté MQTT zprávy obsahující načtená data ze senzoru a informaci k jakému senzoru zpráva patří. Výsledný databázový model je možné vidět na obrázku 12.



Obrázek 12: Databázový model. Zdroj vlastní.

5.6 Konfigurace MQTT Brokeru

Po vytvoření Docker kontejneru s obrazem eclipse-mosquitto MQTT brokeru je nutné do nastaveného adresáře *.mqtt_data* pomocí docker volumes přesunout konfiguraci pro správný běh brokeru.

Je nutné ve složce *.mqtt_data* v lokálním úložišti vytvořit pro potřeby kontejneru složku pod názvem *conf* a v ní vytvořit konfigurační soubor *mosquitto.conf*. V tomto konfiguračním souboru je nastaveno ukládání dat a logování, port na kterém broker naslouchá a povolení připojení klientů i mimo localhost bez uvedení přihlašovacích údajů. Příklad výsledné podoby konfiguračního souboru je uveden níže.

```
# Obsah konfiguračního souboru mosquitto.conf
    persistence true
    persistence_location /mosquitto/data/
    log_dest file /mosquitto/log/mosquitto.log

    listener 1886
    allow_anonymous true
```

5.7 Webová aplikace

Výsledkem praktické části této bakalářské práce je škálovatelný dashboard implementovaný jako webová aplikace ve frameworku Django. Umožňuje jednoduše přidávat libovolný počet senzorů spolu se základními informacemi o nich a případně je odstraňovat. Hlavním účelem této aplikace je poskytnout uživateli přehledné zobrazení informací o uložených senzorech a hlavně o jejich naměřených hodnotách ve formě přehledného grafu historických hodnot spolu se základními statistickými údaji daného senzoru.

5.7.1 Struktura projektu

V této kapitole budou popsány soubory vytvářející hlavní strukturu projektu vytvořeného pomocí frameworku Django. Tento konkrétní projekt obsahuje jednu aplikaci pod názvem *application* a ta je sestavena pomocí několika šablon. Celá struktura projektu, která tu bude následně popsána je znázorněna v následujícím úseku kódu.

```
.
|-- application
|   |-- admin.py
|   |-- apps.py
|   |-- forms.py
|   |-- __init__.py
|   |-- migrations
|   |-- models.py
|   |-- mqtt.py
```

```
|   |-- __pycache__
|   |-- static
|   |-- templates
|   |-- templatetags
|   |-- tests.py
|   |-- urls.py
|   |-- views.py
|-- manage.py
|-- project
|   |-- asgi.py
|   |-- __init__.py
|   |-- __pycache__
|   |-- settings.py
|   |-- urls.py
|   |-- wsgi.py
|-- requirements.txt
```

Základní struktura Django projektu je vytvořena pomocí vygenerování Python souborů při vytvoření projektu. Jedná se zejména o soubory pro konfiguraci projektu. Kořenový adresář aplikace obsahuje textový soubor *requirements.txt* specifikující seznam potřebných knihoven, které jsou nainstalovány během tvorby kontejneru s Django aplikací. Dalším důležitým souborem je Python soubor *manage.py*, díky kterému je možné spravovat aplikaci z příkazové řádky. V dalších souborech jsou nastavovány url adresy a obecné nastavení aplikace jako je připojení k databázi, cesty ke statickým souborům, používání šablonovacího systému atp.

Následně vytvořená aplikace pod názvem *application* obsahuje podle návrhového vzoru MVT soubory definující šablony ve složce *templates*, pohledy s aplikační logikou v souboru *views.py* a modely definující vzhled databáze pomocí ORM v souboru *models.py*. Dalšími důležitými Python soubroy ve složce aplikace jsou například *admin.py*, ve kterém jsou registrovány databázové tabulky do Django admin rozhraní pro jednoduchou správu obsahu databáze, nebo soubor *forms.py* ve kterém je kód definující vzhled vygenerovaného Django formuláře pro uživatelské přidávání a úpravu dat v databázi. Tato aplikace

obsahuje také přidaný Python soubor s názvem *mqtt.py*, který řeší nastavení a komunikaci MQTT klienta pro přijímání zpráv ze senzorů a následně jsou zde tyto zprávy ukládány do databáze.

5.7.2 Připojení databáze

V souboru projektu *settings.py* je uvedeno nastavení pro připojení databáze PostgreSQL pomocí nainstalovaného ovladače psycopg2. Aplikace je připojována k databázi běžící v Docker kontejneru, jejíž nastavení je specifikováno v souboru *docker-compose.yaml*. Konfigurace v souboru *settings.py* vypadá následovně:

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql_psycopg2',
        'NAME': 'app',
        'USER': 'app',
        'PASSWORD': 'app',
        'HOST': 'db',
        'PORT': 5432,
    }
}
```

Veškerá práce s databází je následně prováděna pomocí ORM. Databázové tabulky jsou definovány v Djangu jako modely v souboru *models.py*. Veškeré změny podoby databáze v tomto souboru jsou přeneseny pomocí migrace nástrojem *manage.py* z konzole Django aplikace v Docker kontejneru.

5.7.3 MQTT klient

Pro řešení přijímání MQTT zpráv v aplikaci byla implementována knihovna Paho MQTT Client. Jelikož aplikace potřebuje pouze přijímat zprávy, které byly odeslány ze senzorů, tak implementuje pouze připojení k brokeru a odebírání nových zpráv.

Nejprve je vytvořena instance MQTT klienta, které jsou následně zaregistrovány callbacky pro akce při připojení k brokeru a při přijetí nové zprávy. Následně je klient připojen k brokeru na IP adrese, která je staticky nastavena Docker kontejneru provozujícímu

MQTT broker. Je zde specifikován také port, na kterém broker naslouchá a interval, v jakém jsou zasílány kontrolní ping zprávy pro monitorování zda je spojení stále otevřeno.

Při přijetí serverové odpovědi CONNACK potvrzující úspěšné připojení klienta k brokeru je zavolán callback *on_connect*, kde je klientovi nastaveno odebírání zpráv pomocí metody subscribe na tématu „esp32/#“ s QoS 0, aby stejné zprávy nebyly doručovány opakovaně. Znak „#“ ve stringu tématu se chová jako wildcard a může ho nahradit jakýkoliv string. To znamená, že klient odebírá všechny zprávy z témat, která začínají na „esp32/“. Pokud je ze serveru přijata zpráva publish, tak je zavolán callback *on_message*, ve kterém je řešeno zpracování přijaté zprávy. Přijatá zpráva je uložena do databázové tabulky Messages ke správnému senzoru podle uvedeného tématu.

Příklad kódu zajišťujícího vytvoření nového záznamu v databázové tabulce Messages je uveden níže:

```
Messages.objects.create(
    datetime=timezone.now(),
    message=float(msg.payload.decode("utf-8")),
    qos=0,
    sensor=Sensors.objects.get(topic=msg.topic))
```

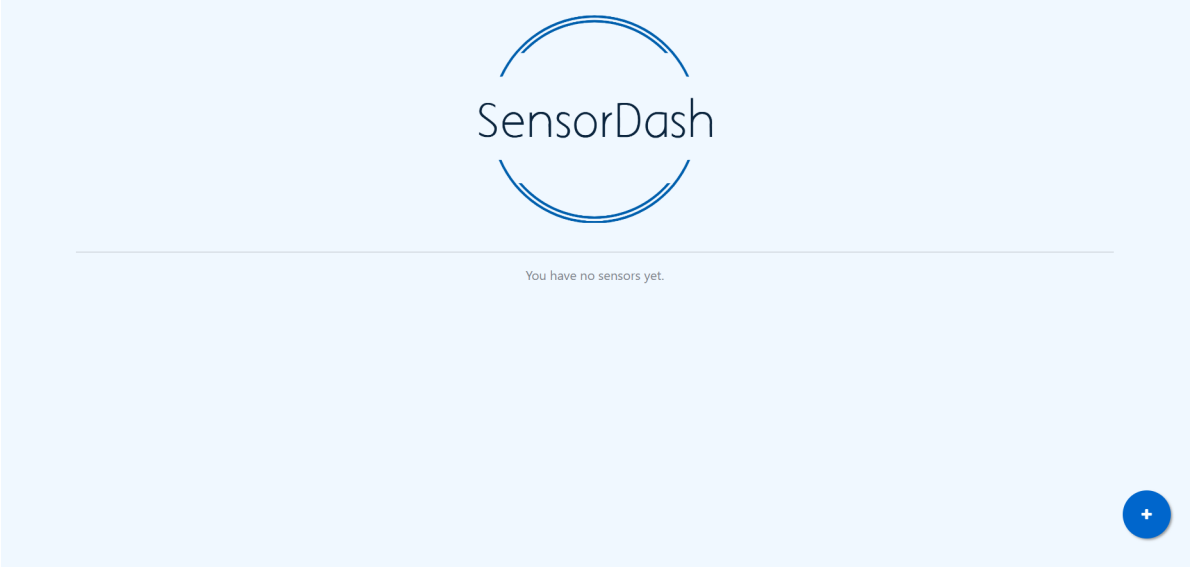
Pro správný běh aplikace je nutné, aby se MQTT klient připojil k MQTT brokeru ještě před spuštěním všech ostatních modulů aplikace. Toto je zajištěno přidáním spuštění smyčky MQTT klienta v souboru aplikace *___init___.py*. Je tak učiněno pomocí metody *loop_start()*, která vytvoří nové vláko a spustí v něm MQTT klienta. Běh MQTT klienta je tedy pro aplikaci neblokující. Tato metoda také řeší automaticky potřebné znovupřipojení k brokeru, pokud došlo k odpojení klienta.

```
# Spuštění MQTT klienta v~souboru aplikace __init__.py
from . import mqtt
mqtt.client.loop_start()
```

5.7.4 Uživatelské rozhraní

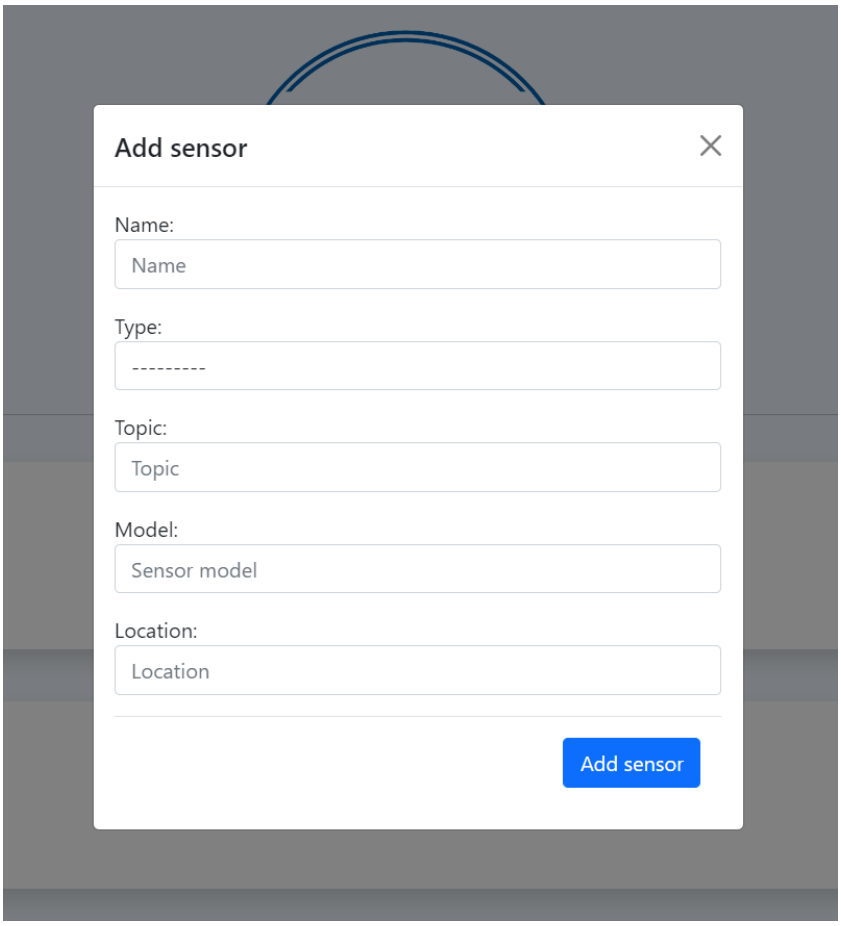
Webová aplikace škálovatelného dashboardu disponuje jednou hlavní obrazovkou, která poskytuje veškeré komponenty potřebné ke zobrazení informací o senzorech. V horní části

obrazovky je zobrazeno logo a v sekci pod ním je list jednotlivých karet, které patří daným senzorům. V pravé dolní části okna je zobrazeno tlačítko otevírající formulář v modálním okně pro přidání nového senzoru do dashboardu. Pokud v aplikaci nejsou ještě registrovány žádné senzory, tak je v dashboardu místo karet senzorů zobrazena zpráva „You have no sensors yet.“.



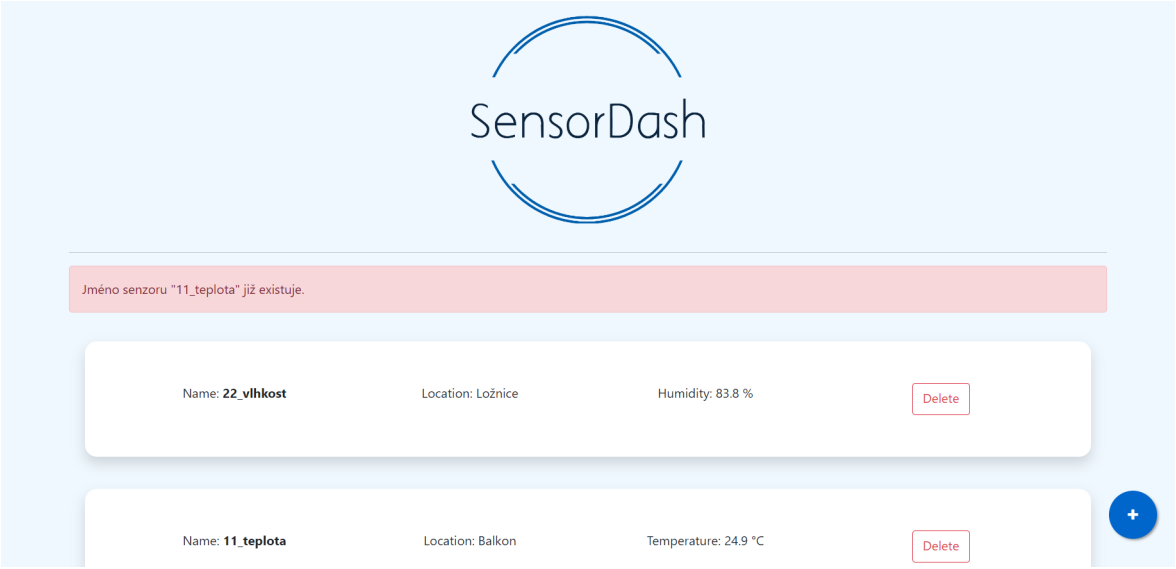
Obrázek 13: Dashboard bez senzorů. Zdroj vlastní.

Do dashboardu je možné přidávat libovolný počet senzorů pomocí tlačítka plus v pravém dolním rohu, které zobrazí formulář v modálním okně. Formulář je generován pomocí nástroje Django forms, který se řídí podle modelů databáze a nastavení formulářů v souboru *forms.py*. V tomto souboru je možné nastavit, které atributy modelu budou ve formuláři zahrnuty, dále jejich popisky, typ vstupního pole a jeho vzhled pomocí CSS tříd. Jak lze vidět na obrázku 14, všechna pole jsou textová vstupní pole typu *TextInput* až na pole pro výběr typu senzoru, který je nastylován jako výběr z možností uložených v databázi pomocí pole typu *Select*.



Obrázek 14: Modální formulář pro přidání senzoru. Zdroj vlastní.

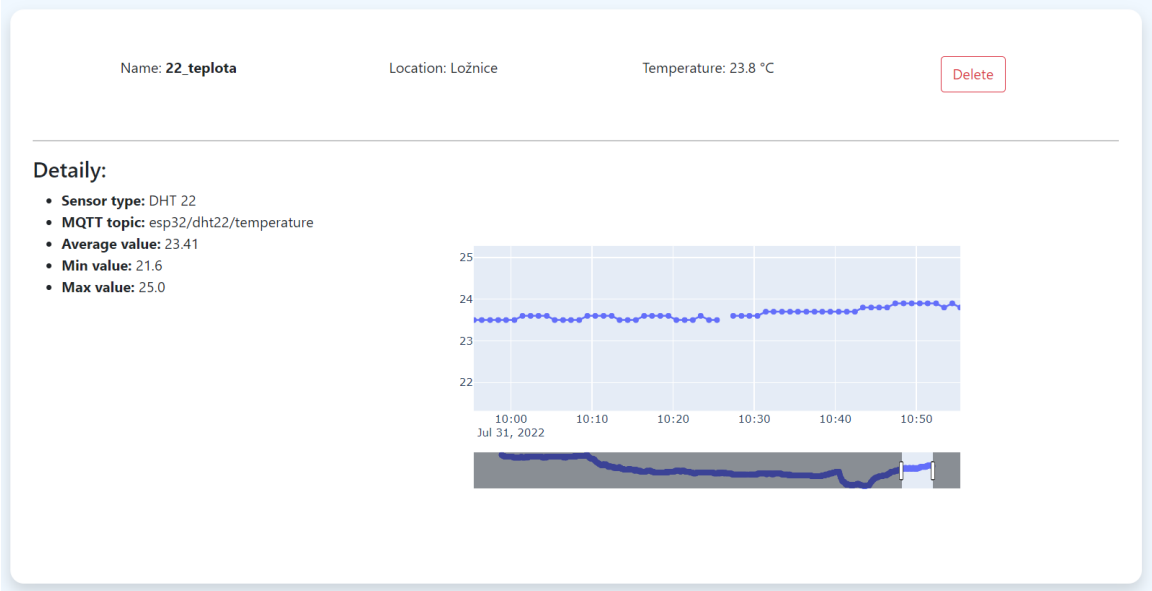
V souboru *forms.py* je také možné zajistit kontrolu zadaných dat a v případě chybného uživatelského vstupu vyvolat výjimku. Django samo o sobě obsahuje ochranu proti SQL injection, Cross Site Scripting, apod. Zobrazuje také upozornění při nevyplnění povinných polí a nedovolí v takovém případě odeslat formulář. V tomto konkrétním řešení byla přidána vlastní kontrola vstupního pole „Name“ pomocí metody *clean* v Django forms. Tato metoda je volána z Views při kontrole, zda je formulář validní a byla v ní implementována kontrola jedinečnosti zadaného jména senzoru. Pokud zadané jméno není jedinečné, tak nový záznam o senzoru není vložen do databáze a uživateli je zobrazena errorová hláška na hlavní stránce dashboardu. Příklad oznámení pokusu o vložení duplicitního jména lze vidět na obrázku 15. Předání a zobrazení této hlášky uživateli je implementováno pomocí nástroje Django messages. Přidání tohoto nástroje do projektu je nastaveno v souboru *settings.py* spolu s jeho nastylování pomocí Bootstrap frameworku.



Obrázek 15: Django messages error. Zdroj vlastní.

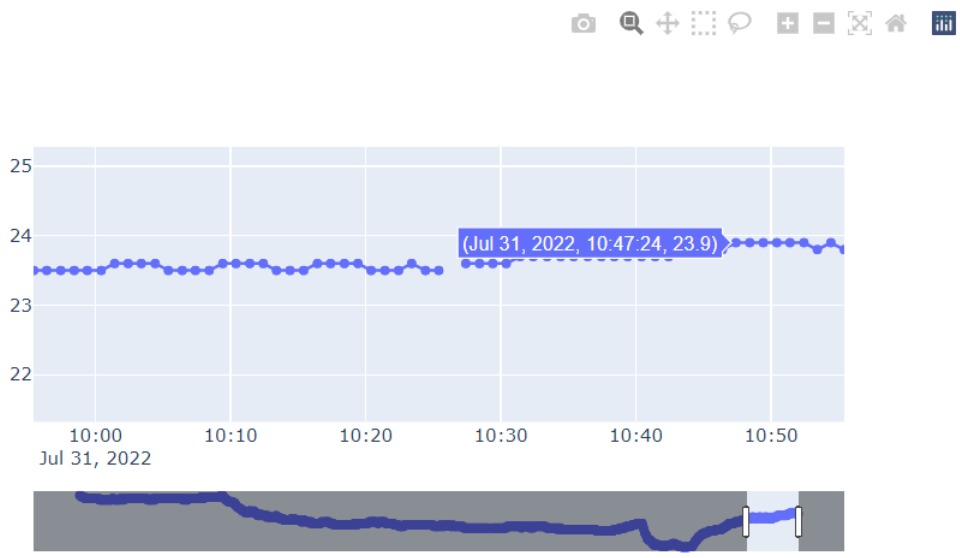
Z obrázku 15 je možné vidět i přidané karty senzorů do dashboardu. Každá karta senzoru obsahuje tento jednoduchý přehled se základními informacemi. Je na nich zobrazeno jméno senzoru, jeho lokace a veličina s poslední naměřenou hodnotou a jednotkou. Dále karta obsahuje tlačítko *Delete* pro případné odstranění senzoru z dashboardu.

Po kliknutí na objekt karty se karta otevře a poskytne uživateli přehled s detailními informacemi o senzoru spolu s interaktivním grafem naměřených historických hodnot.



Obrázek 16: Rozbalená karta senzoru. Zdroj vlastní.

V detailech o senzoru je vypsán typ senzoru, MQTT téma, které aplikace odebírá pro data daného senzoru, průměrná hodnota ze všech měření, a také minimální a maximální naměřená hodnota. V pravé části karty je vykreslen graf se všemi naměřenými hodnotami. Graf byl vytvořen pomocí Python knihovny Plotly. Jedná se konkrétně o Scatter Plot, který je následně převeden do HTML kódu a vyrenderován na webové stránce.



Obrázek 17: Detail grafu. Zdroj vlastní.

Aktuální zobrazení grafu obsahuje naměřená data za posledních 60 minut. Osa x zobrazuje datum a čas, kdy byla hodnota naměřena a osa y zobrazuje naměřenou hodnotu dané veličiny. Pokud je mezi naměřenými hodnotami časový interval větší než dvojnásobek intervalu měření, tedy 2 minuty, tak body poslední a nové hodnoty nebudou v grafu propojeny čarou.

V dolní části grafu je posuvník, díky kterému je možné libovolně měnit šířku aktuálního zobrazení času na ose x. Graf je interaktivní a umožňuje libovolně přizpůsobit velikost aktuálního zobrazení pomocí výběru požadované části myší. Dále lze přejetím myši na vykreslených bodech grafu zobrazit štítek s konkrétní naměřenou hodnotou a časovým údajem. Graf disponuje i ovládacím panelem, který je poskytován knihovnou Plotly a umožňuje uživateli graf libovolně přibližovat, oddalovat, posouvat a také ukládat jako vektorovou grafiku do uživatelova lokálního zařízení.

ZÁVĚR

Cílem této bakalářské práce bylo navrhnout a vytvořit škálovatelný dashboard. Požadovanou vlastností dashboardu měla být pro uživatele možnost přidávat/odebírat senzory a také graficky vizualizovat data ze senzorů. Všechny výše uvedené cíle práce byly splněny vytvořením webové aplikace. Aplikace umožňuje uživateli modifikaci počtu senzorů, následné zobrazení jejich stavů a naměřených hodnot v číselné podobě. Dále je možné zobrazit naměřené historické hodnoty pomocí interaktivních grafů. Uživatel si při přidávání senzorů do dashboardu definuje jedinečný název senzoru, jeho veličinu, model, umístění a MQTT téma, na které daný senzor odesílá data. Pro zápis textu uživatelského rozhraní aplikace byl zvolen anglický jazyk.

K vytvoření funkčního škálovatelného dashboardu bylo zapotřebí nejprve získat zařízení, které bude číst a odesílat data načtená ze senzorů. Pro tuto činnost byl vybrán mikrokontrolér ESP32, ke kterému byly připojeny senzory pomocí nepájivého pole. Na toto zařízení byl přes sériový port nahrán program napsaný v jazyce C, který zajišťuje připojení ESP32 k lokální Wi-Fi síti a k MQTT brokeru. Program čte ve stanoveném časovém intervalu jednou za minutu data ze senzorů a odesílá je pomocí metody publish na MQTT broker.

Jak zde již bylo zmíněno, veškerá komunikace je řízena pomocí MQTT protokolu. MQTT broker zastává pozici serveru a řídí komunikaci, zařízení ESP32 jakožto klient sdílí naměřené hodnoty senzorů a webová aplikace, také jako klient, odebírá zprávy jednotlivých zaregistrovaných senzorů. Použitý protokol pro komunikaci mezi senzory a aplikací byl zvolen tak, aby mohl přenos dat probíhat i mezi nízkoenergetickými IoT zařízeními s malým výkonem a aby nebyla zbytečně zahlcována síť.

Pro zjednodušení přenositelnosti aplikace a jejího nasazení byly vytvořeny tři docker kontejnery pomocí nástroje Docker Compose. V tomto řešení jsou docker kontejnery spuštěny na zařízení Raspberry Pi. V kontejnerech běží MQTT broker řídící komunikaci v IoT systému, dále PostgreSQL databáze zajišťující perzistenci zadaných informací o senzorech a jejich naměřených hodnot pro potřeby aplikace a posledním kontejnerem je Django webová aplikace vytvářející škálovatelný dashboard. Kontejnerům byly v docker síti nastaveny statické IP adresy, aby bylo vždy zajištěno korektní připojení webové aplikace k MQTT brokeru.

Výhodou při použití softwarové platformy Dockeru je možnost spuštění vytvořené aplikace nezávisle na použitém operačním systému, a protože se také jedná o webovou aplikaci, tak je uživateli umožněno jej používat napříč všemi zařízeními s webovým prohlížečem.

Praktický výstup této práce lze využít v domácnostech nebo i malých firmách pro přehlednou vizualizaci aktuálních i historických naměřených hodnot z jednotlivých senzorů v lokální síti. Jedinými požadavky na zprovoznění systému je Wi-Fi síť a schopnost uživatele nahrát na zařízení ESP32 přiložený kód s pozměněnými přihlašovacími údaji do lokální Wi-Fi sítě.

Tuto práci by bylo možné do budoucna rozšířit o správu uživatelů, kde by si každý uživatel mohl registrovat a zobrazovat pouze své senzory. Dále by bylo možné implementovat zabezpečení a šifrovaný přenos zpráv v rámci MQTT protokolu.

POUŽITÁ LITERATURA

[1] GOKHALE, Pradyumna, BHAT, Omkar, BHAT, Sagar. *Introduction to IOT* [online]. India: Tejass Publisheers, 2018, 5(1), 41-44 [cit. 2022-07-31]. ISSN 2393-8021. Dostupné z: https://www.researchgate.net/profile/Omkar-Bhat/publication/330114646_Introduction_to_IOT/links/5c2e31cf299bf12be3ab21eb/Introduction-to-IOT.pdf

[2] POHANKA, Pavel. Internet věcí. *Data Distribution Service* [online]. Česká Republika, Brno: Pavel Pohanka, 06. 10. 2020 [cit. 2022-07-31]. Dostupné z: http://pavelpohanka.cz/internet-of-things/.

[3] KUMARI, Riya. Top 10 Internet of Things (IoT) Examples. *Analytics Steps: A leading source of Technical & Financial content* [online]. India, Noida: Analytics Steps Infomedia LLP, 16. 06. 2021 [cit. 2022-07-31]. Dostupné z: https://www.analyticssteps.com/blogs/top-internet-things-iot-examples.

[4] Insider Intelligence. How IoT and smart city technology works. *Insider Intelligence: Unlock digital opportunities with the world's most trusted forecasts, analysis, and benchmarks* [online]. USA, New York: Insider Intelligence Inc., 15. 04. 2022 [cit. 2022-07-31]. Dostupné z: https://www.insiderintelligence.com/insights/iot-smart-city-technology/.

[5] KHVOYNITSKAYA, Sandra. The IoT history and future. *Itransition: Software Development Company* [online]. USA, Lakewood: Itransition, 25. 11. 2022 [cit. 2022-07-31]. Dostupné z: https://www.itransition.com/blog/iot-history.

[6] Eleven Fifty Academy. The History of the Internet of Things. *Eleven Fifty Academy: Nonprofit Coding and Cybersecurity Bootcamp* [online]. USA, Indianapolis: Eleven Fifty Academy, 2022, 28. 12. 2020 [cit. 2022-07-31]. Dostupné z: https://elevenfifty.org/blog/the-history-of-the-internet-of-things/.

[7] HIDAYET, Aksu. IoT Architecture Layers and Components. *ResearchGate: Find and share research* [online]. Germany, Berlin: ResearchGate GmbH, 06. 02. 2018 [cit. 2022-07-31]. Dostupné z: https://www.researchgate.net/figure/IoT-Architecture-Layers-and-Components_fig1_322975901.

[8] SIKDER, Amit Kumar, PETRACCA, Giuseppe, AKSU, Hidayet. *A Survey on Sensor-based Threats toInternet-of-Things (IoT) Devices and Applications* [online]. USA, Florida: Florida International University, USA, Pennsylvania: Penn State University, 06. 02. 2018 [cit. 2022-07-31]. Dostupné z: `https://www.researchgate.net/publication/322975901_A_Survey_on_Sensor-based_Threats_to_Internet-of-Things_IoT_Devices_and_Applications`.

[9] JENA, Satyabrata. Architecture of Internet of Things (IoT). *GeeksforGeeks: A computer science portal for geeks* [online]. India, Noida: GeeksforGeeks, 2022, 25. 06. 2020 [cit. 2022-07-31]. Dostupné z: `https://www.geeksforgeeks.org/architecture-of-internet-of-things-iot/`.

[10] JAHNKE, Alec. The 4 Stages of IoT Architecture. *Digi International: Industrial IoT (IIoT) Devices and Services for M2M Networking* [online]. USA, Minnetonka: Digi International Inc., 2022, 31. 07. 2020 [cit. 2022-07-31]. Dostupné z: `https://www.digi.com/blog/post/the-4-stages-of-iot-architecture`.

[11] ROSE, Karen, ELDRIDGE, Scott, CHAPIN, Lyman. THE INTERNET OF THINGS: AN OVERVIEW. *Internet Society: Build, Promote, and Defend the Internet* [online]. Switzerland, Geneva: Internet Society. October 2015 [cit. 2022-07-31]. Dostupné z: `https://www.internetsociety.org/wp-content/uploads/2017/08/ISOC-IoT-Overview-20151221-en.pdf`.

[12] BARKAKATI, Nabajyoti, GOLDSTEIN, Mark, WILSHUSEN, Gregory. *Internet of Things: Status and implications of an increasingly connected world*. USA, Washington: U.S. Government Accountability Office, 15. 05. 2017 [cit. 2022-07-31]. Dostupné z: `https://www.gao.gov/assets/gao-17-75.pdf`.

[13] Free Time Learning. What are different communication models in IoT. *Free Time Learning: Online Courses, Tutorials, Interview Questions, Great Learning* [online]. India, Nellore: Free Time Learning, © 2017-2022 [cit. 2022-07-31]. Dostupné z: `https://www.freetimelearning.com/software-interview-questions-and-answers.php?What-are-different-communication-models-in-IoT?&id=1563`.

[14] GOYAL, Piyush. Communication Models in IoT (Internet of Things). *GeeksforGeeks: A computer science portal for geeks* [online]. India, Noida: GeeksforGeeks,

2022, 09. 03. 2021 [cit. 2022-07-31]. Dostupné z: https://www.geeksforgeeks.org/communication-models-in-iot-internet-of-things/.

[15] LINKS, Cees. The Evolution of Wi-Fi networks: from IEEE 802.11 to Wi-Fi 6E. *Wevolver: Knowledge for engineers* [online]. Netherlands, Amsterdam: Wevolver, 24. 05. 2022 [cit. 2022-07-31]. Dostupné z: https://www.wevolver.com/article/the-evolution-of-wi-fi-networks-from-ieee-80211-to-wi-fi-6e.

[16] TechCoder. Wi-Fi Standards Explained. *GeeksforGeeks: A computer science portal for geeks* [online]. India, Noida: GeeksforGeeks, 2022, 17. 03. 2021 [cit. 2022-07-31]. Dostupné z: https://www.geeksforgeeks.org/wi-fi-standards-explained/.

[17] KIRS, Petr. WiFi 6 je tady. Jaké novinky přináší. *CZC.cz: Geek Magazín* [online]. Česká Republika, Ostrava: CZC.cz s.r.o., 19. 09. 2019 [cit. 2022-07-31]. Dostupné z: https://www.czc.cz/geek/wifi-6-je-tady-jake-novinky-prinasi/clanek.

[18] KERACHEVA, Elena. Bluetooth - How it works. *CCM: The latest in tech and digital culture* [online]. France, Bordeaux: CCM Benchmark Group, 22. 01. 2021 [cit. 2022-07-31]. Dostupné z: https://ccm.net/contents/69-bluetooth-how-it-works.

[19] Samsung. What is the maximum range of a Bluetooth connection. *Samsung Levant: Mobiles, TVs, and Home Appliances* [online]. South Korea, Suwon: Samsung, 22. 09. 2020 [cit. 2022-07-31]. Dostupné z: https://www.samsung.com/levant/support/mobile-devices/what-is-the-maximum-range-of-a-bluetooth-connection/.

[20] AVSystem. Everyone is using Bluetooth Low Energy – should you. *AVSystem: Shaping the world of connected devices* [online]. Poland, Kraków: AVSystem, 30. 04. 2021 [cit. 2022-07-31]. Dostupné z: https://www.avsystem.com/blog/bluetooth-low-energy-ble/.

[21] Digi International Inc. What Is Zigbee Protocol Wireless Mesh Networking?: Digi International. *Digi International: Industrial IoT (IIoT) Devices and Services for M2M Networking* [online]. Minnetonka, USA: Digi International Inc., © 2022 [cit. 2022-07-31]. Dostupné z: https://www.digi.com/solutions/by-technology/zigbee-wireless-standard.

[22] AMIN, Imran, SAEED, Atif. *Comprehensive Energy Systems* [online]. Pakistan, Karachi: SZABIST, 28. 02. 2018 [cit. 2022-07-31]. Dostupné z: https://www.sciencedirect.com/topics/engineering/zigbee-protocol.

[23] Z-Wave Alliance. Z-Wave Alliance: About Z-Wave Technology. *The Internet of Things is powered by Z-Wave* [online]. USA, Beaverton: The Z-Wave Alliance, © 2022 [cit. 2022-07-31]. Dostupné z: https://z-wavealliance.org/about_z-wave_technology/.

[24] ORI, Ben. Using Z-Wave Technology in Smart Homes. *IoT For All* [online]. USA, Rockville: IoT For All, 18. 06. 2021 [cit. 2022-07-31]. Dostupné z: https://www.iotforall.com/using-z-wave-technology-in-smart-homes.

[25] Tutorials Point (India) Ltd. HTTP: Overview. *Online Tutorials Library* [online]. Telangana, Hyderabad: Tutorials Point India Private Limited, © 2022 [cit. 2022-07-31]. Dostupné z: https://www.tutorialspoint.com/http/http_overview.htm.

[26] JUNEJA, Manmeet. Difference between MQTT and HTTP protocols. *GeeksforGeeks: A computer science portal for geeks* [online]. India, Noida: GeeksforGeeks, 11. 11. 2021 [cit. 2022-07-31]. Dostupné z: https://www.geeksforgeeks.org/difference-between-mqtt-and-http-protocols/.

[27] GUIDI, Claudio, IATTONI, Riccardo. DZone IoT: Eclipse Mosquitto Connector for Jolie Programming Language. *DZone* [online]. UK, Durham: DZone, , 14. 10. 2020 [cit. 2022-07-31]. Dostupné z: https://dzone.com/articles/eclipse-mosquitto-connector-for-jolie-language.

[28] JavaTpoint. MQTT Protocol: Message Queuing Telemetry Transport Protocol. *Tutorials List: Javatpoint* [online]. India, Noida: JavaTpoint, © 2011-2021 [cit. 2022-07-31]. Dostupné z: https://www.javatpoint.com/mqtt-protocol.

[29] The HiveMQ Team. Introducing the MQTT Protocol - MQTT Essentials: Part 1 *HiveMQ: Enterprise ready MQTT to move your IoT data* [online]. Germany, Bavaria: HiveMQ GmbH, 12. 01. 2015 [cit. 2022-07-31]. Dostupné z: https://www.hivemq.com/blog/mqtt-essentials-part-1-introducing-mqtt/.

[30] STANFORD-CLARK, Andy. IBM Developer. *IBM* [online]. USA, New York: IBM Developer, 02. 01. 2019 [cit. 2022-07-31]. Dostupné z: https://developer.ibm.com/blogs/open-source-ibm-mqtt-the-messaging-protocol-for-iot/.

[31] Raspberry Valley. CoAP Getting Started. *Raspberry Valley* [online]. [cit. 2022-07-31]. Dostupné z: http://raspberry-valley.azurewebsites.net/CoAP-Getting-Started/.

[32] JAFFEY, Toby. MQTT and CoAP, IoT Protocols. *The Eclipse Foundation: The Community for Open Innovation and Collaboration* [online]. Canada, Ottawa: Eclipse Foundation, February 2014 [cit. 2022-07-31]. Dostupné z: https://www.eclipse.org/community/eclipse_newsletter/2014/february/article2.php.

[33] MISHRA, Harshvardhan. What is CoAP Protocol: CoAP Protocol Introduction. *IoTbyHVM - Bits & Bytes of IoT* [online]. India, Uttar Pradesh: IoTbyHVM – Bits & Bytes of IoT, 18. 04. 2019 [cit. 2022-07-31]. Dostupné z: https://iotbyhvm.ooo/what-is-coap-protocol/.

[34] Python 3.10.4 documentation: General Python FAQ. *Welcome to Python.org* [online]. Delaware, United States: Python.org, 2022, 26. 04. 2022 [cit. 2022-07-31]. Dostupné z: https://docs.python.org/3/faq/general.html.

[35] About Python. *Python Institute: PROGRAM YOUR FUTURE* [online]. Szczecin, Polsko: Open Education and Development Group. © 2012-2022 [cit. 2022-07-31]. Dostupné z: https://pythoninstitute.org/about-python.

[36] Django introduction - Learn web development. *MDN Web Docs* [online]. Mountain View, California: Mozilla Foundation, 11. 07. 2022 [cit. 2022-07-31]. Dostupné z: https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/Introduction.

[37] MAKAI, Matt. Django. *Full Stack Python* [online]. © Matt Makai 2012-2022 [cit. 2022-07-31]. Dostupné z: https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/Introduction.

[38] Django Software Foundation. Databases: Django documentation. *Django: The web framework for perfectionists with deadlines* [online]. Django Software Foundation,

© 2005-2022 [cit. 2022-07-31]. Dostupné z: https://docs.djangoproject.com/en/4.0/ref/databases/.

[39] Django MVT Architecture. *AskPython: Python Tutorials for Beginners and Experienced Programmers* [online]. AskPython, © 2022 [cit. 2022-07-31]. Dostupné z: https://www.askpython.com/django/django-mvt-architecture.

[40] Ketan. Django Model View Template (MVT) Overview. *Learn Online Best Java, Python and C Language Tutorials Points* [online]. India, Telangána, 20. 03. 2021 [cit. 2022-07-31]. Dostupné z: https://www.onlinetutorialspoint.com/django/django-model-view-template-mvt-overview.html.

[41] Getting started with plotly in Python. *Plotly Open Source Graphing Libraries* [online]. Canada, Montréal: Plotly, © 2022 [cit. 2022-07-31]. Dostupné z: https://plotly.com/python/getting-started/.

[42] Plotly Python Graphing Library. *Plotly Open Source Graphing Libraries* [online]. Canada, Montréal: Plotly, © 2022 [cit. 2022-07-31]. Dostupné z: https://plotly.com/python/.

[43] Python Plotly Tutorial. *JournalDev: Java, Java EE, Android, Python, Web Development Tutorials* [online]. India, Karnátaka: JournalDev IT Services Private Limited, 22. 03. 2018 [cit. 2022-07-31]. Dostupné z: https://www.journaldev.com/19692/python-plotly-tutorial.

[44] Docker overview. *Docker Documentation* [online]. USA, California: Docker Inc., 29. 07. 2022 [cit. 2022-07-31]. Dostupné z: https://docs.docker.com/get-started/overview/.

[45] OSZTERTÁG, Dávid. Docker: Life Before and After *Avatao: Interactive secure coding training* [online]. Hungary, Budapest: Avatao, 26. 11. 2019 [cit. 2022-07-31]. Dostupné z: https://avatao.com/blog-life-before-docker-and-beyond/.

[46] Docker vs Virtual Machines (VMs) : A Practical Guide to Docker Containers and VMs. *Operate and Manage Kubernetes easily with Weaveworks* [online]. UK, London: Weaveworks, 16. 01. 2020 [cit. 2022-07-31]. Dostupné z: https://www.weave.works/blog/a-practical-guide-to-choosing-between-docker-containers-and-vms.

[47] TOMEČEK, Jaroslav. Virtualizace na úrovni jádra operačního systému. *AbcLinuxu.cz - Linux na stříbrném podnose* [online]. Česká Republika, Praha: Nitemedia s. r. o., 31. 07. 2007 [cit. 2022-07-31]. Dostupné z: https://www.abclinuxu.cz/clanky/system/virtualizace-na-urovni-jadra-operacniho-systemu.

[48] docker container. *Docker Documentation* [online]. USA, California: Docker Inc., 29. 07. 2022 [cit. 2022-07-31]. Dostupné z: https://docs.docker.com/engine/reference/commandline/container/.

[49] docker container start. *Docker Documentation* [online]. USA, California: Docker Inc., 29. 07. 2022 [cit. 2022-07-31]. Dostupné z: tabulkahttps://docs.docker.com/engine/reference/commandline/container_start/#related-commands.

[50] Networking overview. *Docker Documentation* [online]. USA, California: Docker Inc., 29. 07. 2022 [cit. 2022-07-31]. Dostupné z: https://docs.docker.com/network/.

[51] Use volumes. *Docker Documentation* [online]. USA, California: Docker Inc., 29. 07. 2022 [cit. 2022-07-31]. Dostupné z: https://docs.docker.com/storage/volumes/.

[52] Dockerfile reference. *Docker Documentation* [online]. USA, California: Docker Inc., 29. 07. 2022 [cit. 2022-07-31]. Dostupné z: https://docs.docker.com/engine/reference/builder/.

[53] Best practices for writing Dockerfiles. *Docker Documentation* [online]. USA, California: Docker Inc., 29. 07. 2022 [cit. 2022-07-31]. Dostupné z: https://docs.docker.com/develop/develop-images/dockerfile_best-practices/.

[54] JETHVA, Hitesh. How Dockerfile Works?. *Linux Hint* [online]. USA, California: Linux Hint LLC, 2020 [cit. 2022-07-31]. Dostupné z: https://linuxhint.com/dockerfile_beginner_guide/.

[55] Overview of Docker Compose. *Docker Documentation* [online]. USA, California: Docker Inc., 29. 07. 2022 [cit. 2022-07-31]. Dostupné z: https://docs.docker.com/compose/.

[56] Get started with Docker Compose. *Docker Documentation* [online]. USA, California: Docker Inc., 29. 07. 2022 [cit. 2022-07-31]. Dostupné z: https://docs.docker.com/compose/gettingstarted/.

[57] What is a Raspberry Pi? *Teach, learn, and make with the Raspberry Pi Foundation* [online]. UK, Cambridge: Raspberry Pi Foundation, 06. 09. 2014 [cit. 2022-07-31]. Dostupné z: https://www.raspberrypi.org/help/what-%20is-a-raspberry-pi/.

[58] OKOI, Divine. 20 Operating Systems You Can Run on Raspberry Pi in 2021. *FOSSMint: Everything About Linux and FOSS* [online]. India, Mumbai: FOSSMint, 21. 09. 2021 [cit. 2022-07-31]. Dostupné z: https://www.fossmint.com/operating-systems-for-raspberry-pi/.

[59] Buy a Raspberry Pi 4 Model B. *Raspberry Pi* [online]. UK, Cambridge: Raspberry Pi, [cit. 2022-07-31]. Dostupné z: https://www.raspberrypi.com/products/raspberry-pi-4-model-b/.

[60] ESP32-WROOM-32 Datasheet. *Wi-Fi & Bluetooth MCUs and AIoT Solutions I Espressif Systems* [online]. China, Shanghai: Espressif Systems, © 2022 [cit. 2022-07-31]. Dostupné z: https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32_datasheet_en.pdf.

[61] CAMERON, Neil. *Electronics Projects with the ESP8266 and ESP32: Building web pages, applications, and WiFi enabled devices*. Edinburgh: Apress, 2021. ISBN 978-1-4842-6335-8.

[62] IoT ESP-WROOM-32 2.4GHz Dual-Mode WiFi+Bluetooth rev.1, CP2102. *LaskaKit.cz: by Makers for Makers* [online]. Česká Republika, Rychnov nad Kněžnou: LaskaKit, © 2022 [cit. 2022-07-31]. Dostupné z: https://cdn.myshoptet.com/usr/www.laskakit.cz/user/shop/big/1202-5_esp32-pinout.jpg?6137b46c.

[63] *DHT11 Humidity & Temperature Sensor* [online]. [cit. 2022-07-31]. Dostupné z: https://www.mouser.com/datasheet/2/758/DHT11-Technical-Data-Sheet-Translated-Version-1143054.pdf.

[64] lady ada. Overview: DHT11, DHT22 and AM2302 Sensors. *Adafruit Learning System* [online]. USA, New York: Adafruit Industries, 29. 07. 2019 [cit. 2022-07-31]. Dostupné z: `https://learn.adafruit.com/dht`.

[65] DHT11 Sensor Pinout, Features, Equivalents & Datasheet. *Components101 - Electronic Components Pinouts, Details & Datasheets* [online]. Components101, © 2021 [cit. 2022-07-31]. Dostupné z: `https://components101.com/sensors/dht11-temperature-sensor`.

[66] DHT22 Sensor Pinout, Specs, Equivalents, Circuit & Datasheet. *Components101 - Electronic Components Pinouts, Details & Datasheets* [online]. Components101, © 2021 [cit. 2022-07-31]. Dostupné z: `https://components101.com/sensors/dht22-pinout-specs-datasheet`.

[67] MH Photoresistor Light Sensor Module - Einstronic Enterprise. *Einstronic Enterprise: Arduino, 3D Printer, Internet Of Things, Robotic Kits, Components and Parts Seller In Malaysia* [online]. Malajsie, Sabah: Einstronic Enterprise, 05. 11. 2020 [cit. 2022-07-31]. Dostupné z: `https://einstronic.com/product/photoresistor-light-sensor-module/`.

[68] LDR Photoresistor Light Sensor for Arduino. *Home - Made in the Gong* [online]. Australia, Albion Park Rail: Made in the Gong, 16. 09. 2020 [cit. 2022-07-31]. Dostupné z: `https://madeinthegong.com.au/product/ldr-photoresistor-light-sensor-for-arduino/`.

[69] KONDOOR, Harish. *Kick-Start to MicroPython using ESP32 / ESP8266*. Kamataka: Independently Published, 2021. ISBN 979-8574626757.

[70] SANTOS, Rui. ESP32 MQTT - Publish DHT11/DHT22 Temperature and Humidity Readings (Arduino IDE) *Random Nerd Tutorials: Learn ESP32, ESP8266, Arduino, and Raspberry Pi* [online]. Portugal, Porto: RandomNerdTutorials.com, © 2013-2022 [cit. 2022-07-31]. Dostupné z: `https://randomnerdtutorials.com/esp32-mqtt-publish-dht11-dht22-arduino/`.

SEZNAM PŘÍLOH

PŘÍLOHA A

Tato příloha obsahuje seznam použitých hardwarových komponent pro realizaci praktické části.

- Raspberry Pi 4 Model B 8GB RAM
- ESP-WROOM-32
- 2x senzor DHT 11
- 1x senzor DHT22
- 1x LDR Module MH-Sensor-Series
- Nepájivé pole
- Vodiče do nepájivého pole
- Micro USB kabel

PŘÍLOHA B

Tato příloha obsahuje kód v jazyce C pro programování mikrokontroléru ESP32. Kód byl vytvořen podle návodu a příkladu ze zdroje [70]. Soubor je přiložen v archivu s názvem CmilanskaS_SkalovatelnyDashboard_SN_1cast_2022.zip.

PŘÍLOHA C

Tato příloha obsahuje zdrojový kód Django projektu webové aplikace spolu s konfiguračními soubory Dockeru a daty Docker kontejnerů. Projekt je přiložen v archivu s názvem CmilanskaS_SkalovatelnyDashboard_SN_2cast_2022.zip a je ho možné spustit například ve vývojovém prostředí PyCharm.